iJCEM
International Journal of Core Engineering & Management

International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

COMPARATIVE ANALYSIS OF STATE MANAGEMENT APPROACHES FOR SERVER-SIDE RENDERED ANGULAR APPLICATIONS

Sri Rama Chandra Charan Teja Tadi
Software Developer
Raven Software Solutions Inc.
charanteja.tadi@gmail.com
West Des Moines, Iowa, USA.

Abstract

The state management in server-side rendered (SSR) Angular applications presents challenges and opportunities that affect application performance, maintainability, and end-user experience. This paper conducts a comparative analysis of various state management approaches relevant to SSR in Angular, such as the use of services, NgRx, and Akita. By critical analysis, the paper pinpoints the benefits and trade-offs of applying each method based on scalability, integration simplicity, and responsiveness. The research also discusses the best methods of using these state management solutions, focusing on the need for adaptive methods withstanding changing application demands. Based on performance indicators, the research offers worthwhile insights for choosing the most suitable state management strategy to obtain maximum performance in SSR contexts.

Index Terms: State Management, Angular, Server-Side Rendering (SSR), Predictive Analytics, Performance Optimization, User Experience.

I. INTRODUCTION

The advent of web applications has revolutionized software development significantly, with server-side rendering (SSR) offering better performance and usability. Angular applications have, in turn, adopted SSR to mitigate issues related to client-side rendering, including initial load time and search engine optimization. In SSR designs, server-side pre-rendering of pages involves sending fully rendered HTML to the client, a strategy weighing the advantages of dynamic content against the need for speed and efficiency. Several state management strategies well adapted to server-side rendered Angular applications have been investigated, comparing performance, usability, and scalability in the context of SSR.

Being one of the widely used JavaScript frameworks, Angular is used to build interactive single-page applications (SPAs) but introduces complexity in managing state when it comes to server-rendered environments. State management is essential in such an environment to give a smooth user experience and obtain maximum resource utilization. State management strategies for server-side environments have been discussed to concentrate on best practices and measures that result in improved application performance and user satisfaction.

With the continued development of web technology, this work adds to the increasing number of publications on SSR and Angular applications. Comparing various state management strategies - ranging from conventional to contemporary reactive methods - their comparative advantages and corresponding drawbacks are highlighted, offering an insight beneficial to developers and researchers alike.

International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

A. Background and Motivation

The web development landscape has been greatly influenced by recent frameworks like Angular, which facilitate the development of complex SPAs that favor user interactivity and participation. Nonetheless, with more applications, performance issues become more visible, especially with regard to application state management on the server side. It was found that performance is largely lost due to ill-advised architectural decisions and poor state management strategies in Angular apps [3].

SSR solutions have been recognized as a viable solution to such problems, given that pre-rendering of HTML may result in faster loading times [4]. Even though the advantages are there, this shift from client-side rendering to SSR is not without difficulties, especially with regard to handling the state between server and client interactions. This has resulted in a thorough review of current state management strategies geared towards SSR settings, with the aim of determining efficient methods that can eliminate state-bound performance bottlenecks.

In addition, the need to utilize innovative programming methods in order to optimize web application development has been highlighted by studies [2].

Understanding the complexities of state management is crucial since it has a direct influence on application performance and user satisfaction. Seeking optimization in state management not only responds to technical requirements but also complies with user experience requirements in an increasingly competitive online environment, thus the necessity for this study.

B. Objectives of the Study

The primary objective of this study is to perform a comparative analysis of various state management approaches for server-side rendered Angular applications. The strengths and weaknesses of these methodologies are evaluated in terms of performance, scalability, and user experience. Specifically, the objectives are as follows:

- Analyze Existing State Management Techniques: A thorough examination of conventional and contemporary state management techniques is conducted. The study assesses how these techniques cater to the unique demands of SSR within Angular applications, focusing on attributes such as complexity, maintainability, and integration capabilities [3].
- Evaluate Performance Metrics: A systematic comparison of state management approaches under real-world conditions is carried out using key performance metrics. Metrics such as loading time, response time, and throughput are considered to provide a comprehensive performance profile for each method under analysis [4].
- User Experience Assessment: The impact of state management on user experience is assessed through empirical techniques such as surveys and usability testing. The influence of different approaches on user satisfaction and application interaction is examined.
- Provide Recommendations: Based on the comparative analysis, actionable recommendations for developers considering various state management strategies in SSR Angular implementations are formulated. These best practices are informed by both quantitative performance data and qualitative user feedback, ensuring a holistic approach to optimization [4].
- Contribute to the Academic Dialogue: Insights from this research contribute to the academic and professional discourse surrounding SSR and state management in web applications, providing valuable information for future research and development in the ever-evolving landscape of web technologies [2].

II. LITERATURE REVIEW

Research on server-side rendering (SSR) and state management solutions to Angular development has largely expanded in line with the need for greater performance and best-optimized user experiences within web development. Various research papers have investigated the pivotal role SSR plays in the optimization of load times, general responsiveness, and search engine optimization within Angular development. It has been discovered that SSR is beneficial in the rendering of pre-rendered HTML content, which can be performance-improving for the initial loading process of a web application [5].

Several research studies have concentrated on various facets of state management essential to SSR, highlighting the nuances of asynchronous data loading and client-server ecosystem state synchronization. The applicability of existing work in providing a comparative overview of technology stacks is, however, arguably limited in that state management approaches for SSR are not explicitly covered [7].

Meanwhile, programming technologies destined to be deployed in web applications' state management solutions were put into view but without it necessarily being mentioned if they're suitably relevant in SSR [6].

Research analyzing modern approaches has confirmed the presence of a state management framework-correlated performance measurement in SSR systems. Comparative analysis has defined how different web engineering approaches can be instrumental in having a significant impact on the performance and optimization of state management, yielding knowledge regarding effective pre-rendering techniques [5]. Effective state management is therefore implied to be the key to the success of SSR based on data transition smoothness and, consequently, improving the user experience.

A. Overview of Angular and SSR

Angular has asserted its status as a premier JavaScript framework for constructing modern web applications due to its strong foundational architecture and efficient tooling systems. Built on TypeScript, Angular's design facilitates the development of scalable single-page applications (SPAs) characterized by modularity and responsiveness. Core features, such as dependency injection and two-way data binding, provide developers with an organized model for application growth. However, Angular's reliance on client-side rendering can lead to performance drawbacks, particularly related to initial load time.

To combat these limitations, server-side rendering emerges as a complementary mechanism to Angular's capabilities. By rendering HTML on the server, the framework reduces the time before users can view meaningful content, presenting fully formed pages rather than blank canvases waiting for JavaScript execution [5]. Furthermore, this method enhances search engine indexing since content is available immediately, which is essential in improving visibility in search results. Certain advantages of using appropriate programming technologies for enhancing SSR implementations have been mentioned, although the context may not be strictly about integration with state management [6].

Nevertheless, implementing SSR in Angular applications poses challenges regarding state management fidelity. The need for a coherent synchronization strategy is critical to ensure that both the server's and client's views of the application are in alignment. Consequently, various state management libraries and strategies have been proposed to maintain this consistency, allowing developers to benefit fully from SSR's performance improvements while ensuring that users enjoy a seamless experience.

International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

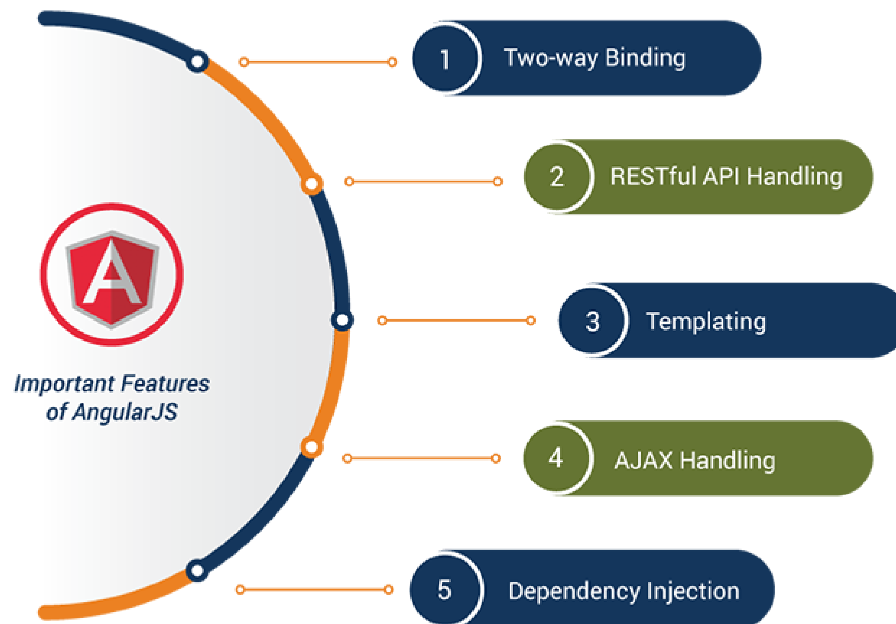

Figure 1: Angular JS Features [11]

B. Importance of State Management

State management is fundamental to the success of Angular applications, particularly in the context of servers that render content dynamically. As modern applications grow in complexity, the demand for robust state management solutions becomes increasingly pressing. Poorly managed application states can lead to issues such as data inconsistency and application failures, undermining the user experience and degrading performance. This is especially evident in SSR scenarios, where timely and consistent data representation is crucial.

In SSR applications, the management of the state must address the need for real-time data synchronization between the server and the client. Properly executed state management can mitigate the risks of presenting outdated or inconsistent data, which is critical for maintaining user trust and satisfaction. Various programming technologies have been identified to streamline state management processes, but their scope appears broader than addressing state management specifically for SSR [6].

Moreover, various design patterns, such as Flux architecture and Redux, provide frameworks for managing application state efficiently. These architectures promote predictability in data flow, making it easier for developers to scale applications and maintain clarity across complex interactions. Effective state management strategies are pivotal for SSR configurations, enabling developers to meet both performance benchmarks and user expectations [5].

In summary, state management emerges as a crucial element that directly impacts the integrity, reliability, and responsiveness of server-side rendered Angular applications. As such, understanding and implementing effective state management strategies is essential for delivering consistent user experiences. By addressing the synchronization challenges inherent in SSR, application fluidity and responsiveness can be significantly enhanced, solidifying the importance of this critical area in modern web development.

III. STATE MANAGEMENT APPROACHES

The state management landscape of server-side rendered Angular applications is a wide range of methodologies, each with specific qualities that suit the requirements of different developers and the needs of different applications. There exists a necessity for adequate state management to uphold user experience and application performance; it coordinates the state of the server and client, thereby reducing discrepancies and latency.

Another common technique for state management is the utilization of centralized stores, in which the state of an application is stored in a single, global location. This design makes it possible for all parts of an application to utilize the same state, enabling simpler data flow and fewer state-mismatch scenarios. Libraries like NgRx and Akita showcase this method with the utilization of state transition management tools and uniform state updates. In addition, these libraries promote the following principles that make applications scalable and maintainable.

Service-based state management is another method that uses services as intermediaries between the components and state information. Data manipulation, retrieval, and persistence logic are encapsulated by the services. Centralized control is avoided, while the application state management is facilitated by this method, along with modularity and flexibility. There are pros and cons of each of the methodologies that are taken into account and result in tailored implementations best suited for specific contexts and purposes.

The challenge is to find the best practices that allow frictionless integration of state management in server-side rendered setups in Angular applications. This means experimentation with multiple techniques without trading off on the corresponding performance cost and SSR best practice support [8].

A. Service-Based Management

Service-based management is used more and more in Angular applications to enable proper state management in server-side rendered (SSR) scenarios. Service-based management is simply about declaring services as intermediaries between application components and backend APIs. By adopting the service-based pattern, API interaction and state management can be managed centrally, resulting in a more structured code base and a more visible separation of concerns.

Perhaps the greatest benefit of service-based management is the way in which it optimizes data access patterns. Services combine data from multiple sources, cache responses to avoid repeated API calls, and handle complicated data streams. In SSR apps, this eliminates the overhead that gets created when multiple components try to fetch data in parallel, resulting in a more efficient data system and better performance. In addition, services are structured to preserve the state of the application and thus enable synchronization among server-rendered material and client-driven operations [9].

Additionally, service-based management facilitates modular development. Services may be written, tested, and serviced individually. Encapsulation facilitates reuse within various modules and applications, reducing development effort and increasing productivity. However one needs to take good care in getting the correct logic into services in order to ensure proper asynchronous data handling with an eye to the variation in the rendering behavior of client-side vs. server-side applications.

Lastly, while organization and adaptability are promoted under service-based management, there has to be careful designing to avoid it amounting to mismanaging states and, in turn, leading to synchronization problems under SSR environments. Organizations need to, therefore, establish

International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

trade-offs between service-based architecture and central store mechanisms for their Angular applications [8].

B. NgRx Overview

NgRx is a full-fledged state management library for Angular applications following the Redux pattern. It offers a comprehensive framework for application state management using reactive programming concepts. NgRx is designed to implement a deterministic state management system that simplifies sharing and managing the state among components, thereby enhancing maintainability and testability in applications.
Effectively, NgRx enables a unidirectional data flow to make state changes under control. NgRx has distinct components: actions, reducers, selectors, and effects, and each of these has a coherent function. Actions represent changes to the state, reducers define how the state changes as a consequence of actions, and selectors are employed for the reading of state data in the best possible manner. Effects provide room for side effects such as API calls or communication with other libraries outside, thus decoupling such behavior from the intrinsic state management features [9].
Figure 2 shows the NgRx state management life cycle and the interaction of its main elements, such as actions, reducers, selectors, and effects.

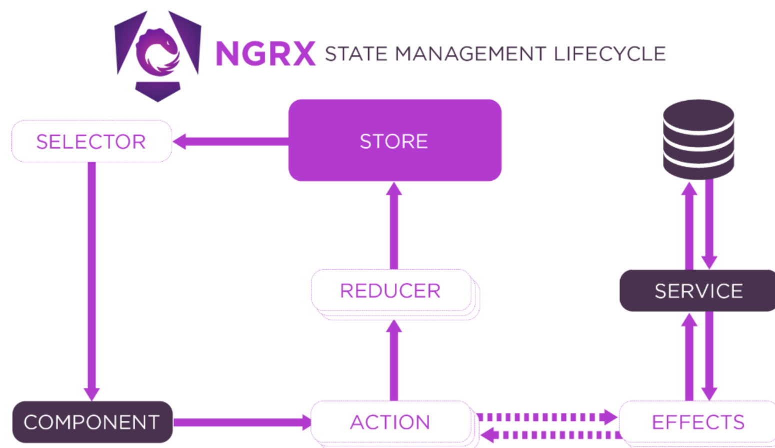

Figure 2: NGRX State Management Life Cycle [12]

With server-side rendered Angular applications, NgRx provides an organized way to coordinate the content produced by the server and client behavior. This comes to the forefront in that SSR entails rendering the initial application state on the server prior to hydrating it on the client. NgRx offers utilities that are easy to integrate into SSR pipelines so that the state can be persisted on both the server and client consistently. Moreover, the observable nature of NgRx suits Angular's reactive programming model quite well, facilitating a more responsive user interface.
However, the implementation of NgRx does demand a clear grasp of its paradigms. The intricacy involved in applying Redux patterns can initially be a learning challenge, especially for developers who are not familiar with reactive programming or state management principles. Nevertheless, for teams seeking to develop scalable and maintainable applications, utilizing NgRx can yield considerable long-term benefits, especially in large-scale systems where predictable state management is crucial [9].

C. Akita Framework

iJCEM International Journal of Core Engineering & Management
International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

Akita is another powerful state management library built specifically for Angular apps with simplicity and usability in mind but without compromising on functionalities. Akita has a less complex API than NgRx, hence appealing to developers who are looking for a simple yet potent solution. Akita is optimally suited for applications where the additional complexity of Redux-like state management is not needed.
One of the positives of Akita is that it supports store functionality where local and remote states can be handled in a straightforward manner. Defining entities within the store itself encourages cleanliness when it comes to structuring and managing the application state. Services and models with data access and state logic are handled by Akita, and this results in more compact modularization along with less boilerplate code compared to what other state management methods can provide.
In server-side rendered applications, Akita's methodology is natively available to match the demands of fluid data management since it makes the way data is queried and updated easier. This kind of performance is essential in SSR applications, where the state needs to be maintained consistently across server-rendered views and the client-side updates that follow. Akita's native observables enable seamless changes in state updates in SSR scenarios.
Besides, Akita's lightweight design provides a flatter learning curve, which makes it ideal for smaller projects or teams where rapid development cycles are necessary. With its ongoing development, Akita's feature set is growing to incorporate more features that enhance developers' ability to create responsive, scalable applications with an optimized state management approach.

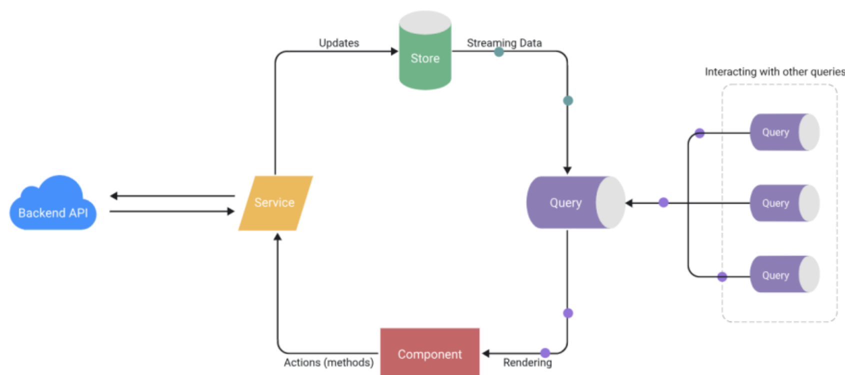

Figure 3: State Management using Akita [13]

IV. COMPARATIVE ANALYSIS

Comparative analysis of state management strategies in Angular applications examines different frameworks, such as NgRx, Akita, and service-based state management strategies, and their advantages and disadvantages in managing application states. By comparative analysis, their impact on real-world applications regarding performance, scalability, and development experience becomes apparent.
NgRx, whose origin traces back to a Redux-inspired framework, is of vital importance to those applications where maximum control of state handling is needed. Its structure allows it to handle states in a deterministic manner, something that is quite essential in scenarios where intricate state modifications are probable [2]. NgRx's core leverages observables through its dependency on RxJS, providing support for the handling of asynchronous streams of data in a competent way. This

iJCEM
International Journal of Core Engineering & Management
International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

aspect benefits SSR apps as it facilitates seamless server-client transfer with good performance and reduced latency gaps that are prone to adversely affect the users' experience.

Nonetheless, NgRx's structural rigidity and sometimes excessive setup may discourage its application in less complex applications where the overhead may be too much compared to the benefits. Excessive boilerplate code utilized may limit rapid iterations, particularly within agile development scenarios.

Akita, on the contrary, proves useful in scenarios where deployment speed and flexibility are paramount. Unlike NgRx, Akita contains less boilerplate, and states can be effortlessly defined and easily implemented quickly in applications. This efficiency is particularly warranted when development cycles need to be quick, allowing teams to respond to changing requirements or user feedback quickly [1]. Akita's entity management supports managed state management and works seamlessly into existing Angular apps. However, with increasing applications, discipline in ensuring state consistency across components increases, and reliability is a concern with more complex interactions.

Service-based management provides an architecturally loose solution that enables domain logic to encapsulate in modular services. It promotes reusable services and can easily reduce development complexity. However, the issue is how to facilitate efficient state synchronization among services because inconsistencies will occur when different components rely on information handled by encapsulated layers of services. Hence, strong mechanisms need to be put in place to ensure consistency between server-side states and client updates, especially in high-interactivity-demanding applications.

Therefore, although NgRx is a strong tool for complex state management situations because of its systematic approach, Akita benefits from the best speed and convenience appropriate for smaller projects or projects experiencing rapid development stages. Management through services can be provided with modularity of flexibility to cater to dynamic applications but comes with a higher degree of complexity in terms of ensuring state consistency.

A. Evaluation Criteria

Assessing the state management solutions for server-side Angular applications requires a structured and compact approach that considers performance, usability, and flexibility. The following are the most important assessment criteria for each solution:

1. Performance Criteria: Assessing responsiveness against different loads and measuring factors such as initial load time and delay in user interactions is essential. Effective performance optimizes user satisfaction and engagement.
2. Scalability: Proper systems of state management must not only accommodate today's application size but also be resilient enough to enable future growth. Support for and capacity to handle growing complexity and feature sets without diminishment is critical.
3. Usability by developers: Assessing how simple the framework is to develop and deploy is crucial. Metrics of measurement include ease of access to documentation, volume of community support available, and quality of developer experience.
4. Maintainability: The ability of the framework to support simple updates and tuning is important. A low-maintenance framework decreases the possibility of technical debt accumulation, which can paralyze future development.
5. Consistency Management Across Client-Server Interaction: For SSR cases, it is essential to get a consistent application state when transitioning between server-rendered and client-

rendered pages. Seamless synchronization mechanisms enhance user experience by removing inconsistencies.

6. Community and Ecosystem Support: Extensive community presence results in plenty of shared resources, rapid problem-solving, and ongoing framework advancement. An active ecosystem keeps the framework current with changing web standards.

B. Performance Metrics

The measurement of performance metrics is central to determining how efficient different state management approaches in server-side rendered Angular applications are. Some of the most important performance metrics are:

1. Initial Load Time: The measurement of the duration it takes to display useful content to the user. Improving initial loads leads to improved user retention.
2. Time to Interaction (TTI): The Time it takes to make interaction feasible by a user after initial content loading is done is important. Reduced TTI improves perceived speed and usability.
3. Resource Utilization: Understanding memory use by various state management frameworks dictates performance. An optimized framework reduces resource usage for a more performance-centric experience.
4. User Interaction Response Time: Measuring user action to application response latency gives insight into responsiveness. Brief response times enhance user interaction.
5. System Stability and Error Tracking: Error frequency and types caused by state management implementations give insight into robustness. Lower error frequencies are associated with more stable state management solutions.
6. Data Synchronization Latency: Data synchronization latency between the client and servers must be reduced in SSR systems to provide decent performance. Low latency with highly efficient frameworks provides a better user experience.

C. Key Findings

The study of state management trends for server-side rendered Angular applications has provided basic insights into development practices and framework usage. The key findings are:

- NgRx's Strengths: Its rigid architecture leans towards complex state transitions. State change predictability enhances code reliability and makes debugging easier but may make small projects cumbersome [2].
- Akita's Fast Development Advantages: Akita's speed and simplicity of deployment are enticing for teams that have to get development cycles to speed without being encumbered by setup hassles. State management may be managed effectively with low friction, but consistency should be recalled fervently in larger applications [1].
- Service-Based Management Flexibility: This pattern allows the rapid development of applications through the modular usage of services. However, handling the persistent state when numerous services share state logic is difficult, requiring disciplined design to prevent issues.
- Other Performance Behaviors: Performance tests reveal that despite greater setup overhead, NgRx mostly beats solutions such as Akita in a more complex context based on the mature state management functionality. Quick startup performance, though, remains Akita's domain for the most basic scenarios [1].

International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

- Maintaining a Uniform State of Application Across Contexts: It is essential to sustain a uniform application state when transitioning from server-side to client-side interaction. Selecting frameworks with built-in synchronization mechanisms enhances the overall user experience.

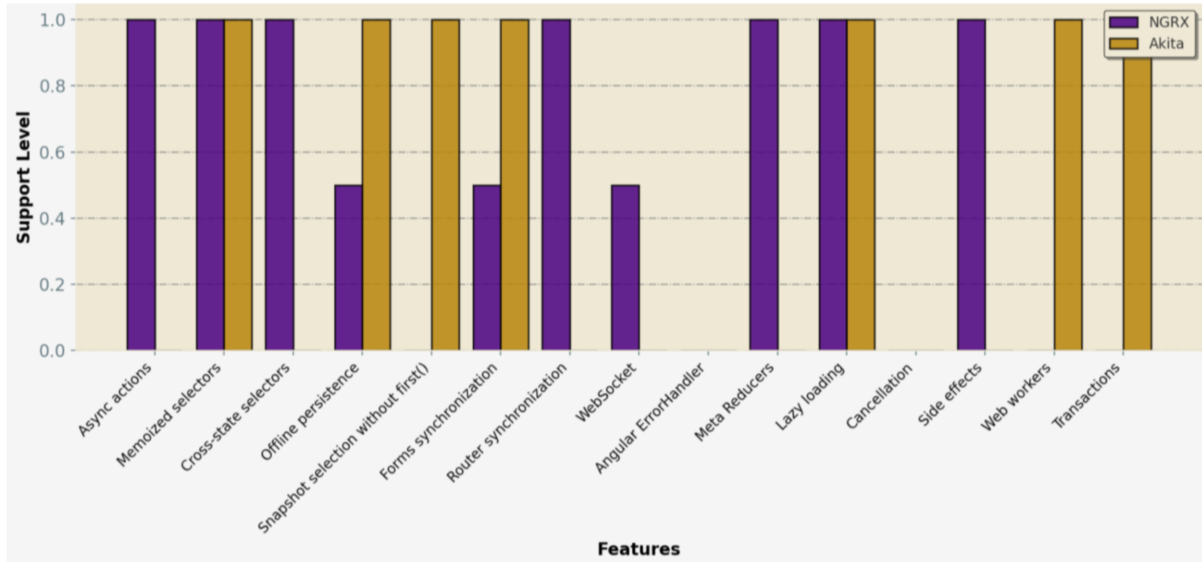

Figure 4: Comparison of NgRx vs. Akita Features

V. BEST PRACTICE RECOMMENDATIONS

For efficient state management of server-side rendered Angular applications, best practices that improve performance, maintainability, and user experience should be adopted. According to the literature available, including observations on AngularJS development, the following guidelines offer a systematic approach:

- Assess Framework Appropriateness: The choice of a state management framework should be made according to the particular needs of the application. It is important to understand the strengths of frameworks like NgRx, Akita, or service-based management. NgRx is best suited for applications with intricate state interactions and high predictability requirements because of its organized nature. Akita is best suited for projects that need quick development without too much boilerplate, whereas service-based management provides flexibility for dynamic applications.
- Apply Consistent State Synchronization: Consistent application state between the server and client is essential in server-side rendered applications. Frameworks must support efficient synchronization mechanisms. For instance, NgRx has tools for side effect management, which can help simplify updates. Through these capabilities, user interfaces can be made to match the latest application state on both the server and client side.
- Maximize Performance with Lazy Loading and State Management Strategies: Performance can be maximized, particularly under the paradigm of SSR, by implementing lazy loading techniques in order to postpone loading content until required. Accompanied by state management strategies such as state chunking, memory consumption can be reduced to

iJCEM
International Journal of Core Engineering & Management

International Journal of Core Engineering & Management
Volume-6, Issue-09, 2020 ISSN No: 2348-9510

expedite initial loading. These types of techniques yield a better user experience, particularly in bigger apps where performance rates are crucial.

- Highlight Error Handling and Recovery: Strong error handling is essential for state management systems. Consistent strategies for tracking errors and handling them should be in place. Embedding error handling directly in state management processes layers ensures that state transition or API call failures do not critically impact user interactions. This improves application resilience and user trust.
- Perform Thorough Testing: Ongoing testing of state management implementations is part of ensuring application integrity. Unit tests for state-related logic, integration tests for component interactions, and complete end-to-end tests need to be created to mimic actual user scenarios. The investment in systematic testing not only identifies problems early but also enforces stability as the application grows.
- Have Clear Documentation and Code Quality: Keeping code well-commented and clean assures long-term maintainability. The documentation should consist of state management approaches described, flow diagrams as needed, and easy-to-understand instructions for subsequent developers. Implementing tools such as ESLint for code quality and readability enhancements can limit technical debt, making the codebase easier to navigate [10].
- Leverage Community Expertise: Drawing on the community at large can offer insights. Contributions to forums, discussions, and reading success stories or setbacks of others in controlling state within Angular applications must be encouraged. Such interactions can expose best practices and creative strategies that may not necessarily be self-evident from the documentation.
- Stay Up-to-Date on Framework Changes: The world of technology is always changing, particularly with frameworks such as Angular. Keeping up with updates, new features, and best practices by reading the official Angular blog, participating in workshops, and talking to thought leaders in the industry is essential. Ongoing education allows the most recent developments to be utilized, resulting in improved application performance and user experiences.

By incorporating these best practices in the development process, more effective state management can be obtained, and as a result, user-friendly as well as sustainable server-side rendered Angular applications can be created.

VI. CONCLUSION

In conclusion, research on server-side rendered Angular state management libraries presents the key role of quality state management in achieving high-performance, user-focused web applications, with state management libraries such as NgRx and Akita offering developers varied tools and approaches based on project needs, each of which has its merits and demerits. Proper testing of an application should not be underestimated; it is the cornerstone on which lies the most solid software that best serves the needs of users and continues to serve changing needs.

Finally, selecting a state management strategy must be done carefully, taking into account the particular application scenario, team skill, and ultimate project vision. With the use of the learning acquired from this analysis, Angular applications can be developed to scale, perform well, and remain responsive in order to realize the utmost user engagement and satisfaction with proper state management playing a central role in modern web development.

REFERENCES

1. P. Vuorimaa, M. Laine, E. Litvinova, and D. Shestakov, "Leveraging declarative languages in web application development," World Wide Web, vol. 19, no. 4, pp. 519–543, 2016. [Online]. Available: https://doi.org/10.1007/s11280-015-0339-z
2. C. Short, "GrayStarServer: Server-side spectrum synthesis with a browser-based client-side user interface," Publications of the Astronomical Society of the Pacific, vol. 128, no. 968, p. 104503, 2016. [Online]. Available: https://doi.org/10.1088/1538-3873/128/968/104503
3. M. Ramos, M. Valente, and R. Terra, "AngularJS performance: A survey study," IEEE Software, vol. 35, no. 2, pp. 72–79, 2018. [Online]. Available: https://doi.org/10.1109/ms.2017.265100610
4. Z. Xiao, C. Withana, A. Alsadoon, and A. Elchouemi, "A front-end user interface layer framework for reactive web applications," American Journal of Applied Sciences, vol. 14, no. 12, pp. 1081–1092, 2017. [Online]. Available: https://doi.org/10.3844/ajassp.2017.1081.1092
5. K. Wakil and D. Jawawi, "Comparison between web engineering methods to develop multi web applications," Journal of Software, vol. 12, no. 10, pp. 783–793, 2017. [Online]. Available: https://doi.org/10.17706/jsw.12.10.783-793
6. E. Nikulchev, D. Ilin, P. Kolyasnikov, and I. Zakharov, "Programming technologies for the development of web-based platform for digital psychological tools," International Journal of Advanced Computer Science and Applications, vol. 9, no. 8, pp. 1–7, 2018. [Online]. Available: https://doi.org/10.14569/ijacsa.2018.090806
7. J. Heredia and G. Sailema, "Comparative analysis for web applications based on REST services: MEAN stack and Java EE stack," KnE Engineering, vol. 3, no. 9, pp. 82–94, 2018. [Online]. Available: https://doi.org/10.18502/keg.v3i9.3647
8. J. Chou and C. Yang, "Obfuscated volume rendering," The Visual Computer, vol. 32, no. 12, pp. 1593–1604, 2016. [Online]. Available: https://doi.org/10.1007/s00371-015-1143-6
9. S. Chen, U. R. Thaduri, and V. K. R. Ballamudi, "Front-end development in React: An overview," Engineering International, vol. 7, no. 2, pp. 117–126, 2019. [Online]. Available: https://doi.org/10.18034/ei.v7i2.662
10. E. Elrom, AngularJS, 1st ed. Berkeley, CA, USA: Apress, 2016, pp. 101–129. [Online]. Available: https://doi.org/10.1007/978-1-4842-2044-3_5
11. "Automation of Angular Apps: TechnoCast - Summer 2019," QASource, 2019. [Online]. Available: https://blog.qasource.com/automation-of-angular-apps-technocast-summer-2019
12. "@ngrx/store," ngrx - Official Documentation. [Online]. Available: https://ngrx.io/guide/store
13. "A reactive state management tailor-made for JS applications," Akita - Official Documentation. [Online]. Available: https://opensource.salesforce.com/akita/