УДК 004.7

# EXPLORING METHODS FOR USING FINITE STATE MACHINES TO OPTIMIZE AND ENHANCE CROSS-PLATFORM MOBILE APPLICATIONS

Мартинов В.Р.
e-mail: vladyslav.martynov@nure.ua
Kharkiv National University of Radio Electronics, Department of Software Engineering
Kharkiv, Ukraine

In the modern world, mobile applications play a crucial role in communication, productivity, and entertainment. As these applications grow in complexity, efficient state management becomes a significant challenge, especially in cross-platform frameworks like React Native. Traditional approaches such as Redux often lead to performance bottlenecks and maintenance difficulties due to complex state handling. Finite State Machines (FSMs) provide a structured methodology for state transitions, enhancing stability, predictability, and maintainability. This work explores the application of FSMs in React Native, demonstrating their advantages over conventional state management techniques and evaluating their impact on performance and usability.

Mobile applications are a fundamental part of contemporary digital interactions, facilitating communication, productivity, and entertainment. The rise of cross-platform development has provided significant advantages, enabling developers to write code once and deploy it across multiple platforms. However, this approach introduces inherent challenges, particularly in state management. Issues such as state inconsistencies, redundant computations, and inefficient updates often degrade the user experience, increasing the complexity of debugging and performance optimization.

Finite State Machines (FSMs) present a potential solution to these issues by offering a structured approach to handling application states. Unlike conventional state management techniques, FSMs define explicit transitions between states, reducing ambiguity and ensuring predictable application behavior. By employing FSMs, developers can create more robust workflows, especially in complex scenarios such as authentication, navigation, and real-time user interactions. This study examines the implementation of FSMs in a React Native application and evaluates their impact on performance and maintainability.

State management in mobile applications is one of the most critical yet challenging aspects of development. Traditional approaches such as Redux and the Context API have been widely adopted, but as applications scale, significant limitations emerge. Redux, for instance, relies on a global state container, leading to excessive re-renders and unnecessary computations, making it difficult to maintain a clean architecture. Additionally, developers often

encounter asynchronous state updates, resulting in race conditions and unpredictable behavior, making debugging tedious and time-consuming. Moreover, existing solutions require tracking multiple Boolean flags, bloating the codebase and reducing readability. Consequently, applications suffer from performance degradation due to excessive event processing and uncontrolled updates, reducing responsiveness.

To address these challenges, modern state management solutions are increasingly using modular and event-driven architectures. One such approach is Nitro-Modules, which enhances state encapsulation by offering independent, reactive state containers that seamlessly integrate with finite state machines. Unlike monolithic state managers, Nitro-Modules allows fine-grained control over state updates, reducing unnecessary redraws and increasing application modularity. Combining this approach with FSM further improves state transitions and ensures that updates are predictable and efficient across different parts of the application.

Finite State Machines represent a paradigm shift in handling state transitions in mobile applications. Unlike traditional approaches, FSMs employ a well-defined structure where states are explicitly defined, and transitions occur based on a finite set of predefined rules. This structured approach eliminates the need for disparate Boolean flags and unpredictable event handling, ensuring a more predictable and reliable application flow. By encapsulating state changes into well-defined transitions, FSMs simplify debugging and improve testability. Additionally, the modularity of FSMs enhances maintainability, making it easier to extend applications without introducing unforeseen side effects.

While FSMs have been widely utilized in embedded systems and game development, their potential remains largely untapped in mobile applications, particularly in cross-platform frameworks such as React Native. This research takes the first step toward integrating FSMs into React Native applications and demonstrates how structured state management can resolve longstanding inefficiencies, resulting in more performant and maintainable applications.

To evaluate the effectiveness of FSMs in state management, this study implements FSMs in a React Native application using XState, an advanced library designed to manage complex state transitions. The primary objective is to measure improvements in UI responsiveness, JavaScript execution efficiency, and overall application stability. A practical test case is developed in the form of a mobile authentication flow – a scenario that involves managing multiple asynchronous requests, handling validation states, and dynamically responding to user interactions. In standard React Native implementations using Redux or the Context API, these transitions are managed manually, often leading to excessive re-rendering, increased memory consumption, and race conditions in the authentication logic.

By integrating FSMs via XState, the application benefits from explicit state definitions and structured transitions, ensuring a deterministic flow at each

authentication step. This implementation eliminates unnecessary computations, as state changes occur only when predefined conditions are met. Additionally, FSMs provide a clear visual representation of application states, making debugging more intuitive and facilitating collaboration among development teams.

Empirical measurements from experiments reveal significant performance improvements. The FSM-based implementation maintains consistently high frame rates and ensures smooth UI responsiveness. Unlike traditional approaches, where uncontrolled state updates often lead to frame drops and performance degradation, FSMs optimize transitions, reduce resource overhead, and improve execution speed. Furthermore, applications utilizing FSMs exhibit fewer inconsistencies, as each transition is explicitly validated before execution.

The adoption of FSMs in React Native marks a transformative shift in mobile development methodologies. While traditional state management techniques have long struggled with inefficiencies, FSMs offer a robust alternative that not only enhances performance but also simplifies maintainability. The findings of this study highlight the untapped potential of FSMs in mobile application development, signaling the beginning of a new era in structured state management solutions.

The results emphasize the benefits of FSMs in cross-platform mobile development, demonstrating their ability to improve performance, maintainability, and scalability. By applying FSMs in React Native applications, developers can mitigate common issues related to inconsistent states and complex user interactions.

Future research could expand on these findings by exploring FSM applications in real-time collaborative systems, AI-driven user interfaces, and highly interactive mobile experiences. Investigating the integration of FSMs with other state management paradigms may further refine best practices for achieving optimal performance in cross-platform applications. As mobile development continues to evolve, adopting structured methodologies like FSMs will be essential for building robust, user-friendly applications that meet the demands of modern digital ecosystems.

Список використаних джерел:

1. What is Nitro? | Nitro Modules. Welcome to Nitro! | Nitro Modules. URL: https://nitro.margelo.com/docs/what-is-nitro .

2. About the New Architecture · React Native. React Native · Learn once, write anywhere. URL: https://reactnative.dev/architecture/landing-page .

3. Comparison with other frameworks | Nitro Modules. Welcome to Nitro! | Nitro Modules. URL: https://nitro.margelo.com/docs/comparison .

4. XState | Stately. Stately | Build complex logic intelligently. URL: https://stately.ai/docs/xstate .

5. Jagdale D. Finite State Machine in Game Development. International Journal of Advanced Research in Science, Communication and Technology. 2021. P. 384–390. URL: https://doi.org/10.48175/ijarsct-2062.