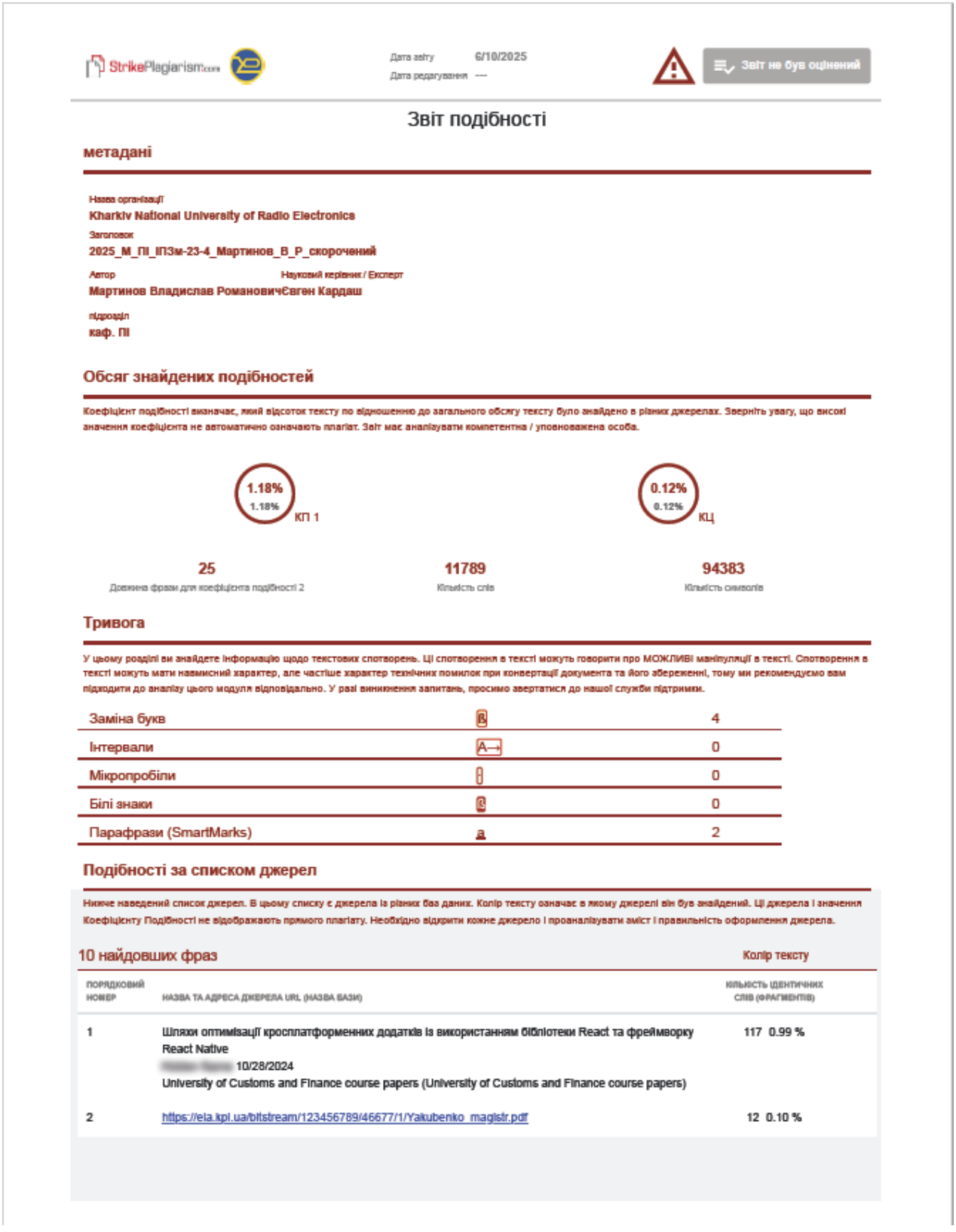
ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism

Дата звіту 6/10/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_M_ПІ_ІПЗм-23-4_Мартинов_В_Р_скорочений
Автор **Мартинов Владислав Романович**
Науковий керівник / Експерт **Євген Кардаш**
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.18% 1.18% КП 1

0.12% 0.12% КЦ

25	11789	94383
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. ЦІ спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	В	4
Інтервали	A⟶	0
Мікропробіли	¶	0
Білі знаки	Ø	0
Парафрази (SmartMarks)	a	2

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Шляхи оптимізації кросплатформенних додатків із використанням бібліотеки React та фреймворку React Native [redacted] 10/28/2024 University of Customs and Finance course papers (University of Customs and Finance course papers)	117 0.99 %
2	https://ela.kpi.ua/bitstream/123456789/46677/1/Yakubenko_magistr.pdf	12 0.10 %

3	Cognitive modeling of factors of influence on the processes of formation and reproduction of fixed assets of agricultural enterprises Olena Yatsukh;	10 0.08 %

з бази даних RefBooks (0.08 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
	джерело: Paperity	
1	Cognitive modeling of factors of influence on the processes of formation and reproduction of fixed assets of agricultural enterprises Olena Yatsukh;	10 (1) 0.08 %

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.99 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Шляхи оптимізації кросплатформенних додатків із використанням бібліотеки React та фреймворку React Native [redacted] 10/28/2024 University of Customs and Finance course papers (University of Customs and Finance course papers)	117 (1) 0.99 %

з Інтернету (0.10 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ela.kpi.ua/bitstream/123456789/46677/1/Yakubenko_magistr.pdf	12 (1) 0.10 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

10
ВСТУП
У сучасному світі, зі зростанням використання мобільних пристроїв, важливість створення кросплатформенних додатків, що забезпечують високу продуктивність та якість користувацького досвіду, значно зросла. Удосконалення створення таких додатків з використанням React та React Native відкриває широкі можливості для розробників завдяки можливості перевикористання коду, високій швидкості розробки та легкості впровадження інновацій і набуває особливої актуальності, оскільки вона стосується не тільки технологічних аспектів розробки, але й економічної ефективності, якості користувацького досвіду та швидкості впровадження інновацій на ринок програмного забезпечення. Відкритий код та активне спілкування у спільноті дозволяють швидко виявляти та усувати проблеми, розробляти нові функції та покращення, а також сприяють обміну знаннями та досвідом між розробниками. Це створює позитивне середовище для інновацій та росту, сприяє швидкому прогресу технологій та підвищує якість кінцевих продуктів 1.
Скінченні автомати (FSM) є одним із найперспективніших інструментів для управління станами у мобільних додатках. Вони забезпечують чітку структурованість, оптимізують складні сценарії роботи додатків, підвищують продуктивність та стабільність роботи, мінімізують ризики помилок. До того ж FSM спрощують розробку програмного продукту, завдяки використанню формального підходу до управління переходами між станами, що в свою чергу, робить програмний код більш зрозумілим і підтримуваним. Застосування FSM є особливо корисним для вирішення практичних задач, пов'язаних із організацією навігації, авторизацією користувачів, обробкою помилок, а також при управлінні взаємодією з мережею та внутрішніми API пристроями. Завдяки цьому скінченні автомати стають незамінним інструментом у кросплатформеній розробці, де

ДОДАТОК Б

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Дослідження методів використання скінченних автоматів для оптимізації та підвищення ефективності кросплатформених мобільних додатків

Мартинов Владислав Романович , ІПЗм-23-4
Науковий керівник: проф. Шубін І.Ю.

червень 2025

1

Дослідження

Актуальність. Скінченні автомати (FSM) є одним із найперспективніших інструментів для управління станами у мобільних додатках. Вони забезпечують чітку структурованість, оптимізують складні сценарії роботи додатків, підвищують продуктивність та стабільність роботи, мінімізують ризики помилок. До того ж FSM спрощують розробку програмного продукту, завдяки використанню формального підходу до управління переходами між станами, що в свою чергу, робить програмний код більш зрозумілим і підтримуваним. Завдяки цьому скінченні автомати стають незамінним інструментом у кросплатформеній розробці, де додатки повинні відповідати високим стандартам ефективності та адаптивності до різних операційних систем.

Актуальність теми зумовлена динамічним розвитком технологій та необхідністю забезпечення якісного доступу до даних на різних пристроях, незважаючи на відмінність їхніх операційних систем, впровадженням сучасних методів управління станами мобільних додатків, які задовольняють потреби користувачів і забезпечують їх конкурентоспроможність на ринку.

Об'єкт та мета дослідження. Метою даного дослідження є вивчення сучасних підходів до управління станами в мобільних додатках та інтеграція кінцевих автоматів для оптимізації управління станами кросплатформених мобільних додатків з метою підвищення їх продуктивності, стабільності та зручності підтримки.

2

Огляд літератури (аналогів)

Аналіз наукових та професійних джерел підтверджує, значну увагу науковців і практиків до використання кінцевих автоматів (FSM), як інструменту для моделювання поведінки систем та управління їхніми станами.

Сучасні дослідження здебільшого спрямовані на оптимізацію архітектур FSM, включаючи зменшення апаратних витрат, підвищення продуктивності пам'яті та вдосконалення алгоритмів переходів між станами. Водночас, питання інтеграції FSM у процеси розробки кросплатформенних мобільних додатків залишаються майже недослідженими. Обмежено розглядається використання FSM для вирішення завдань, пов'язаних із складною логікою навігації, обробкою асинхронних подій або підтримкою інтерактивних інтерфейсів користувача.

Традиційні підходи до управління станами, такі як Redux або MobX, хоча і є ефективними, мають обмеження у контексті складних мобільних додатків. Використання FSM у цьому контексті може стати перспективним напрямом, що дозволить оптимізувати процеси управління станами.

Загалом, результати огляду свідчать про необхідність дослідження інтеграції FSM у кросплатформену мобільну розробку. Подальші дослідження в цій галузі сприятимуть покращенню якості мобільних додатків, підвищенню їхньої стабільності та забезпеченню зручності для кінцевих користувачів.

Постановка задачі

Постановка задачі передбачає дослідження та оцінку ефективності застосування скінченних автоматів для оптимізації та підвищення ефективності кросплатформених мобільних додатків.

Реалізація наукового дослідження складається з наступних етапів:

- аналіз переваг та недоліків застосування FSM на основі огляду наукових джерел;
- розробка методики порівняльного аналізу ефективності використання декларативних та FSM методів для управління станами мобільних додатків;
- визначення критеріїв оцінки ефективності;
- експериментальне дослідження на основі розробки тестового мобільного застосунку з використанням кросплатформеного мобільного фреймворку React-Native для платформ iOS та Android;
- аналіз результатів експериментів та надати рекомендації щодо доцільності використання FSM.

Отримані результати дослідження можуть бути використані при проєктування та реалізації кросплатформених мобільних застосунків.

Методологія

Для проведення експериментального дослідження було розроблено тестові екрани для платформ iOS та Android, з використанням декларативного підходу та FSM.

Інтерфейс розроблених екранів містить базовий функціонал:

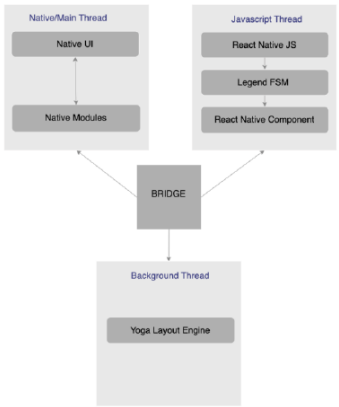
- форму авторизації та валідації;
- узгодження політики конфіденційності та умов надання послуг;
- кнопку створення нового облікового запису;
- емуляція входу з використанням облікового запису Google, Apple, LinkedIn;
- індикатор стану завантаження.

Використані інструменти та технології

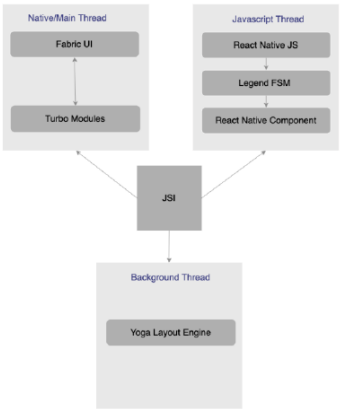
- фреймворм для розробки кросплатформених мобільних додатків React-Native;
- open-source бібліотека реактивного стану Legend-State;
- open-source бібліотека опису стилів React-Native-Unistyles;
- статична мова програмування TypeScript.

5

Архітектура системи для проведення експериментального дослідження



Стара архітектура RN



Нова архітектура RN

6

Опис програмного забезпечення, що було використано у дослідженні

Для проведення експериментального дослідження було обрано відповідні технології, інструменти та середовище розробки:

Мобільні пристрої: IPhone 15 Pro - процесор Apple A17 Pro (64-bit ARM), Samsung Galaxy SE (2022) – процесор ARM64 (Snapdragon).

Технічні параметри фреймворку React-Native

- v 0.76.6;
- supported min iOS version: 15.0;
- supported min Android version: 8.0;
- supported architectures: A - серія чіпів Apple, ARM, ARM64, x86, x86_64.

Середовище розробки:

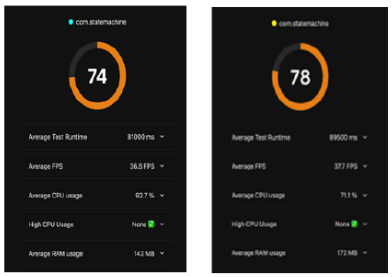
- Cursor Editor – основне середовище редагування коду;
- Xcode – для збірки та тестування iOS-додатка;
- Android Studio – для збірки та тестування Android-додатка.

Зміст проведеного експерименту

Процедура вимірювання продуктивності з використанням інструменту FlashLight за наступною процедурою:

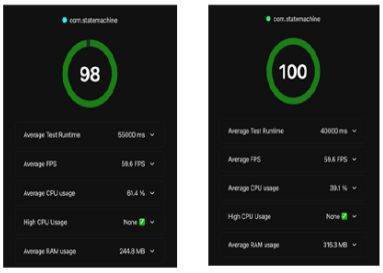
- компіляція проекту для платформ iOS та Android в режимі release mode;
- запуск програми **FlashLight** наступними параметрами: flashlight measure --skipRestart platforms: ios, android;
- активація debug mode для підрахунку кількості загальних перемалювань інтерфейсу;
- взаємодія з інтерфейсом екрану протягом 30 секунд (Declarative / FSM);
- обчислення ключових показників ефективності: Average FPS, Average CPU usage, High CPU Usage, Average RAM Usage;
- структурування та узагальнення результатів проведеного експерименту.

Проведення експерименту



(а) (б)

Рисунок 5.4 – Загальні показники ефективності роботи мобільного додатку (декларативний підхід): (а) стара архітектура; (б) нова архітектура



(а) (б)

Рисунок 5.8 – Загальні показники ефективності роботи мобільного додатку (FSM): (а) стара архітектура; (б) нова архітектура

Порівняння продуктивності мобільного додатку з використанням різних підходів (стара архітектура)

Метрика	Декларативний підхід	Legend State (FSM)	Різниця, %
Average FPS, FPS	36,6	59,6	+ 62,84 %
Average CPU Usage, %	92,70	61,40	- 33,76 %
High CPU Usage	відсутні пікові значення	відсутні пікові значення	-
Average RAM Usage, MB	143	244,8	+ 71,33 %

Порівняння продуктивності мобільного додатку з використанням різних підходів (нова архітектура)

Метрика	Декларативний підхід	Legend State (FSM)	Різниця, %
Average FPS, FPS	37,7	59,6	+ 58,09 %
Average CPU Usage, %	71,10	39,10	- 45,01 %
High CPU Usage	відсутні пікові значення	відсутні пікові значення	-
Average RAM Usage, MB	172	316,3	+ 83,84 %

Аналіз отриманих результатів

Результати із застосуванням старої архітектури:

- FPS зріс із 36,6 до 59,6, що свідчить про збільшення стабільності зображення на + 62,84%;
- середнє навантаження на CPU зменшилося з 92,7% до 61,4%, що підтверджує зниження навантаження на центральний процесор мобільного телефону на - 33,76%;
- використання оперативної пам'яті збільшилося з 143 MB до 244,8 MB, що складає + 71,33%.

Результати із застосуванням нової архітектури:

- FPS зріс із 37,7 до 59,6, що свідчить про збільшення стабільності зображення на + 58,09%;
- показник навантаження на центральний процесор CPU зменшився з 71,1% до 39,1%, що становить - 45,01%;
- зросло використання оперативної пам'яті з 172 MB до 316,3 MB, приріст склав + 83,84%.

Висновки

При теоретичному вивченні скінченних автоматів було досліджено:

- архітектуру FSM, яка є провідним фактором який впливає на ефективність розробки, тестування, використання та підтримання кросплатформених мобільних додатків;
- варіанти поєднання реактивних бібліотек з FSM для підвищення прокуктивності кросплатформених мобільних додатків.

На основі проведеного експериментальне дослідження підтверджено:

- гіпотезу про те, що впровадження State Machine-підходів при створені кросплатформених мобільних додатків підвищують та оптимізують їх ефективність;
- поєднання нового архітектурного стеку React Native з реактивними бібліотеками та автомато-орієнтованим управлінням станом демонструє найкраще співвідношення між ефективністю, адаптивністю та масштабованістю для реальних мобільних сценаріїв.

Ці висновки мають практичне значення для розробників програмного забезпечення в галузі кросплатформених мобільних додатків.

ДОДАТОК В

Апробація у вигляді статті на 13-й Міжнародній науково-технічній конференції Інформаційні системи та технології ICT-2024

Exploring the methods for using state machines to optimize and enhance cross-platform mobile applications

Vladyslav Martynov[1,*,†], Ihor Shubin[1,†], Victoria Skovorodnikova[1,†]

[1] Kharkiv National University of Radio Electronics, Nauky ave., 14, Kharkiv, 61166, Ukraine

Abstract

Nowadays, contemporary mobile apps require robust state management, particularly in cross-platform frameworks such as React Native. Finite State Machines (FSMs) provide a structured methodology for managing state transitions, enhancing stability, performance, and testability in complex applications. This article examines the advantages of FSMs and their practical applications in cross-platform development, with a focus on React Native.

Keywords

FSM, React, React-Native, XState

1. Introduction

As modern mobile applications become increasingly embedded in users' daily routines, delivering responsive and efficient functionality is paramount. These applications serve a vast array of purposes—quick access to information, seamless communication, task management, and beyond—placing high demands on developers to continually enhance functionality, performance, and reliability. Cross-platform frameworks like React-Native empower developers to deploy applications across multiple platforms, streamlining development and reducing resource expenditure. Yet, cross-platform solutions introduce distinct challenges, particularly in state management.

Effective state management remains a central challenge in cross-platform development. Complex applications, especially those with multifaceted user interactions, asynchronous tasks, and extensive use of device resources, rely on smooth state transitions to ensure consistent behavior. Without a robust architecture for state management, applications risk crashes, unpredictable behaviors, and a diminished user experience.

Finite State Machines (FSMs) offer a systematic approach to addressing these challenges. By rigorously defining all possible states and permissible transitions, FSMs enforce predictable and stable application behavior, proving especially useful in managing navigation flows, handling asynchronous operations, and structuring complex logic. Within mobile applications, FSMs excel at optimizing scenarios driven by external and internal events, such as user authentication and server requests, where precise state control is essential.

A major advantage of FSMs is their ability to impose a well-defined structure on application control logic, minimizing errors and simplifying state tracking. This structure enhances testability, allowing developers to independently assess each state and transition and

Information Systems and Technologies (IST-2024), November 26-28, 2024, Kharkiv, Ukraine
* Corresponding author.
† These authors contributed equally.
vladyslav.martynov@nure.ua (V. Martynov); ihor.shubin@nure.ua (I. Shubin); viktoriia.skovorodnikova@nure.ua (V. Skovorodnikova)
0009-0003-5665-2323 (V. Martynov); 0000-0002-1073-023X (I. Shubin); 0000-0003-0436-4197 (V. Skovorodnikova)

proactively identify potential issues. Furthermore, FSMs contribute to performance optimization by streamlining transitions and reducing unnecessary computations, a critical consideration for cross-platform solutions where execution speed directly impacts user satisfaction.

2. Review of the literature

The concept of state is very familiar. Consider a toaster: initially, the toaster is in the off state; when you push the lever down, a transition is made to the on state and the heating elements are turned on; finally, when the timer expires, a transition is made back to the off state and the heating elements are turned off.

A finite state machine (FSM) consists of a set of states s_j and a set of transitions between pairs of states s_i, s_j . A transition is labeled condition/action: a condition that causes the transition to be taken and an action that is performed when the transition is taken. FSMs can be displayed as a state diagram Fig.1.

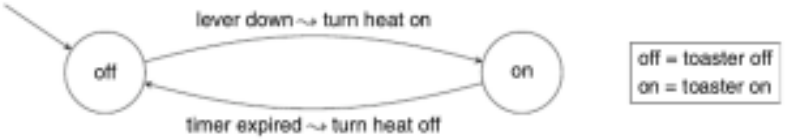


Figure 1: State diagram

A state is denoted by a circle labeled with the name of the state. States are given short names to save space and their full names are given in a box next to the state diagram. The incoming arrow denotes the initial state. A transition is shown as an arrow from the source state to the target state. The arrow is labeled with the condition and the action of the transition. The action is not continuing; for example, the action turn left means set the motors so that the robot turns to the left, but the transition to the next state is taken without waiting for the robot to reach a specific position [1].

React Native is an open source framework for building Android and iOS applications using React and the app platform's native capabilities. With React Native, you use JavaScript to access your platform's APIs as well as to describe the appearance and behavior of your UI using React components: bundles of reusable, nestable code [2].

Unlike hybrid frameworks that run within a web view, React Native compiles to native components, providing applications with a look and feel similar to those built with platform-specific languages like Swift for iOS or Kotlin for Android.

However, managing complex user flows in React Native can be challenging due to platform-specific differences, which impact component behavior, navigation patterns, and resource management. These differences often require additional logic to handle varied platform responses, such as asynchronous tasks, permission handling, and native animations, which can lead to increased complexity and potential inconsistencies in user experience. Addressing these challenges effectively requires a structured approach to state management, particularly when handling intricate navigation and interactive flows across platforms.

Finite State Machines (FSMs) offer an effective solution for such state management challenges by enforcing a strict and predictable sequence of states and transitions. In scenarios involving complex workflows—like multi-step forms, authentication flows, or dynamic content rendering—FSMs provide developers with a clear framework for defining each state and its permissible transitions. This structure mitigates the unpredictability that can arise from asynchronous tasks or varying platform behavior, helping to ensure consistent responses to user actions across iOS and Android.

By using FSMs within React Native applications, developers gain the ability to encapsulate complex user interactions in a way that is both testable and maintainable. Each state transition is explicitly defined, which reduces the likelihood of unintended behaviors and facilitates debugging. In cross-platform environments where maintaining uniformity is essential, FSMs provide a reliable approach for handling intricate application states, enhancing both the predictability and resilience of mobile applications.

3. Experiments

In this experiment, the basic authentication screen will be implemented with advanced, specialized library called Xstate and then will be compared with default React Native implementation.

XState is a state management and orchestration solution for Javascript and Typescript apps. It uses event-driven programming, state machines, statecharts, and the actor model to handle complex logic in predictable, robust, and visual ways. XState provides a powerful and flexible way to manage application and workflow state by allowing developers to model logic as actors and state machines. It integrates well with React, Vue, Svelte, and other frameworks and can be used in frontend, backend, or wherever Javascript runs [3].

XState enhances the clarity and robustness of state management through its advanced syntax, which allows developers to encapsulate and manage state logic directly within the machine configuration. XState's internal context functions as a dedicated data store, managing state-specific information such as validation flags, input values, and error messages. This encapsulation minimizes reliance on external state management solutions, consolidating all relevant logic and data within a single, cohesive unit.

4. Results

To evaluate the performance of both the React Native default implementation and XState implementations, we used three key metrics: the React Native UI thread, the JS thread, and FPS drop. The UI and JS thread metrics measure the responsiveness of the application by tracking frame rates on each thread, with higher values indicating smoother performance. FPS drop, on the other hand, reflects any decrease in frames per second during interactions, with lower values indicating fewer disruptions in user experience. These metrics provide a comprehensive view of each implementation's ability to handle complex interactions smoothly.

Based on Table 1 metrics, XState demonstrates superior performance across multiple dimensions. In the XState implementation, both the UI and JS threads maintain a steady average of 60 frames per second (FPS). This consistent performance indicates that XState efficiently handles state transitions and user interactions without overloading either thread. The minimal FPS drop (2-3) further suggests that XState is capable of processing complex interaction patterns smoothly, keeping the application responsive and ensuring a seamless user experience even under heavy usage conditions. This level of stability is crucial in maintaining an interactive flow, especially on an authentication screen where real-time validation and feedback are essential.

In contrast, the React Native default implementation shows performance limitations, with the UI thread averaging 55 FPS and the JS thread at 50 FPS. This reduction in frame rates suggests that it introduces more load on the system, potentially due to less optimized transition handling or higher overhead in managing state changes. The FPS drop is also significantly higher, ranging from 10 to 12 FPS during intensive interactions. This larger drop indicates that default implementation experiences performance bottlenecks under complex user actions, resulting in occasional stutters and a less fluid experience. Such delays can negatively impact user perception, especially in scenarios where users expect immediate feedback, such as entering login credentials.

In summary, XState's structured approach and optimized handling of state transitions provide a clear advantage in terms of performance, maintaining high frame rates on both UI and JS threads and minimizing FPS drops.

Table 1
Performance metrics

Metrics	XState	React Native default implementation
UI Thread	60	55
JS Thread	60	50
FPS Drop	2-3	10-12

5. Conclusion

This study demonstrates the effectiveness of using finite state machines to manage complex interactions within mobile applications, particularly in cross-platform frameworks like React Native. By implementing an authentication screen in two variants — React Native default implementation and the XState library—we explored the performance, maintainability, and user experience benefits of each approach.

The XState implementation proved to be superior in terms of performance, with consistently high frame rates and minimal FPS drops, ensuring a smooth and responsive experience. Its advanced abstractions, such as context handling, hierarchical state management, and visual state charts, make it especially well-suited for applications with intricate interaction flows. The ability to visualize state charts not only aids in understanding and debugging complex workflows but also facilitates collaboration among team members and provides a form of living documentation as the application evolves.

React Native default implementation, demonstrated some limitations in handling resource-intensive transitions and maintaining consistent frame rates under heavy user interactions. Although it offers basic state management, the overhead associated with managing states manually makes it less suitable for highly interactive or complex forms, especially when compared to the modular and declarative approach offered by XState.

In conclusion, XState is a powerful tool for managing complex forms and dynamic interactions across various applications. Its structured approach to state management makes it versatile enough for a wide range of use cases, including multi-step forms, onboarding flows, authentication processes, and any scenario requiring predictable state transitions. The library's modular design allows developers to maintain a clean codebase, improve performance, and enhance user experience. As mobile applications continue to grow in complexity, adopting a well-defined state management solution like XState can play a critical role in building robust, scalable, and maintainable applications.

Future work could explore extending this approach to more intricate applications involving real-time updates, multi-user collaboration, or machine learning-driven interactions. Furthermore, the potential for combining XState with other tools in the React Native ecosystem, such as gesture handlers and animations, opens up possibilities for crafting highly interactive and intuitive user experiences.

References

[1] Finite State Machine URL: http://surlli/xbonql (date of access 20.11.2024)
[2] React-Native URL: http://surLli/bwzllc (date of access 18.11.2024)
[3] XState URL: https://stately.ai/docs/xstate (date of access 15.11.2024)

ДОДАТОК Г

Апробація у вигляді тез на XXVIII Міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті»

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.

ДОДАТОК Д

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-4 (група)

Мартинов Владислав Романович
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	

зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

11.06.2025