УДК 004.41

МОДУЛЬНА АРХІТЕКТУРА У REACT NATIVE-ДОДАТКАХ

Височин І.М., Кравець Н.С.
e-mail: illia.vysochyn@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

In modern mobile development, creating scalable, flexible, and maintainable applications is crucial. Large React Native applications often suffer from monolithic structures, making maintenance, testing, and further development challenging. Implementing a modular architecture helps address these issues by dividing the application into independent modules, each containing its own logic, UI components, and API requests. The adoption of modular architecture in React Native significantly improves development efficiency and prepares applications for future scalability and performance optimization.

У сучасній мобільній розробці важливо створювати масштабовані, гнучкі та підтримувані проєкти. React Native дозволяє швидко розробляти мобільні додатки, але великі проєкти часто страждають від монолітної структури, що ускладнює їхню підтримку, імплементацію нових модулів [1], тестування та подальший розвиток.

Метою дослідження є аналіз ефективності модульної архітектури у розробці мобільних застосунків на React Native, розробка та тестування застосунку з використанням модульного підходу для оцінки його впливу на продуктивність, підтримуваність та масштабованість.

Для досягнення поставленої мети використовувалися такі методи: аналіз літератури та сучасних практик у сфері мобільної розробки, експериментальне моделювання, що передбачає створення тестового застосунку з модульною архітектурою (Feature-Based) та його порівняння з монолітним аналогом, а також емпіричне тестування, яке включає вимірювання продуктивності (час рендерингу, споживання пам’яті), оцінку зручності тестування та внесення змін у код.

Модульна архітектура в React Native – це підхід до організації коду, який передбачає поділ застосунку на ізольовані модулі або функціональні блоки. Кожен модуль містить власну логіку, UI-компоненти, API-запити та управління станом, що дозволяє оптимізувати підтримку та розширення застосунку [2]. Основними перевагами модульної архітектури є гнучкість, оскільки вона дозволяє легко додавати новий функціонал без значного впливу на решту застосунку; зручність тестування, адже модулі можна перевіряти незалежно один від одного; покращена масштабованість, яка дає змогу впроваджувати зміни без ризику ламати основну систему; а також розподіл відповідальності, що дає можливість працювати над різними частинами застосунку одночасно кільком командам розробників.

Одним із основних підходів до модульного структурування програмного забезпечення є Feature-Based Architecture (архітектура, заснована на функціональних модулях). У цьому підході кожен функціональний блок застосунку, наприклад, "авторизація", "профіль користувача" або "чати", є окремим самостійним модулем. Кожен із цих модулів містить власні компоненти інтерфейсу, бізнес-логіку, маршрути, сервіси та API-запити, що дозволяє організувати код таким чином, щоб зміни в одному функціональному блоці не впливали на інші. Це забезпечує хорошу масштабованість, полегшує тестування та підтримку, а також сприяє командній розробці, коли різні групи можуть паралельно працювати над окремими функціональними модулями.

Альтернативний підхід – Layer-Based Architecture (шарова архітектура), яка передбачає поділ застосунку на рівні абстракції. Наприклад, можна виокремити рівень UI-компонентів, рівень бізнес-логіки, рівень сервісів, рівень роботи з API або базою даних, а також рівень навігації. Такий підхід дозволяє чітко розмежувати відповідальність між різними частинами застосунку, що спрощує управління складними структурами. Наприклад, якщо в проєкті використовується загальний сервіс авторизації, його можна легко підключити до будь-якого модуля без дублювання логіки.

Розмежований підхід забезпечує ізольованість кожного модуля, що значно покращує підтримку застосунку та робить його масштабованим. Важливим аспектом при модульному підході є зниження зв’язності між модулями за допомогою Dependency Injection (DI). Це дозволяє змінювати внутрішню логіку кожного модуля без впливу на загальну архітектуру проєкту. У React Native для цього можна використовувати React Context API, custom hooks або спеціальні бібліотеки, такі як inversify.js.

Ще одним ключовим аспектом модульної архітектури є ефективний менеджмент стану, оскільки кожен модуль повинен мати змогу керувати своїми даними незалежно від інших. Серед популярних підходів до управління станом у React Native можна виокремити Redux Toolkit, який є оптимізованою версією Redux і дозволяє ізолювати логіку управління станом у межах кожного модуля (Slice-based architecture); Zustand, що є легковаговим менеджером стану з мінімальними накладними витратами, який дозволяє створювати незалежні стореджі для кожного модуля; а також Recoil, який базується на атомарному підході і дає змогу легко розподіляти дані між модулями, зберігаючи їхню ізольованість. Вибір конкретного підходу залежить від складності застосунку та вимог до продуктивності [4].

Оптимізація продуктивності є важливим фактором у мобільній розробці, тому застосування Lazy Loading дозволяє завантажувати модулі лише за потреби, зменшуючи час початкового рендерингу застосунку. У React Native реалізація Lazy Loading можлива через React.lazy() та

Suspense, що дає змогу динамічно підключати компоненти лише в момент їхнього використання [5]; Code Splitting, який розділяє код на менші частини, що зменшує розмір початкового бандлу; а також спеціальні бібліотеки, такі як react-native-dynamic-bundle, які допомагають здійснювати динамічне завантаження ресурсів залежно від їхнього використання.

Очікується, що модульний підхід значно покращить масштабованість застосунку, дозволяючи швидше додавати новий функціонал без ризику порушення роботи інших компонентів. Використання Redux Toolkit або Zustand у контексті модульної архітектури зменшить складність управління станом та покращить ізоляцію бізнес-логіки. Lazy Loading дозволить скоротити час завантаження застосунку на 15-30% порівняно з монолітною архітектурою, а використання Dependency Injection спростить тестування та рефакторинг, уможливлюючи незалежні зміни в модулях. Очікується також, що підхід Feature-Based Architecture покращить підтримуваність коду, скорочуючи час внесення змін на 20-40%.

Таким чином, впровадження модульної архітектури у React Native-додатках дозволяє створювати більш гнучкі, продуктивні та масштабовані рішення. Поділ застосунку на незалежні модулі сприяє покращенню його підтримуваності, спрощує тестування та підвищує ефективність роботи команд розробників. Використання сучасних підходів до менеджменту стану, Lazy Loading, Dependency Injection та добре продуманої модульної структури дозволяє зменшити складність розробки та оптимізувати продуктивність мобільних застосунків. У майбутньому розвиток React Server Components, можливість підтримки WebAssembly та поява нових інструментів для оптимізації продуктивності можуть зробити модульний підхід ще більш ефективним для мобільних застосунків.

Список використаних джерел:

1. Konovalov, B. , & Rutkas, A. (2024). Review Of Approaches To Integrating Cartographic User Interfaces Into React Native Mobile Applications. Collection of Scientific Papers «SCIENTIA», (April 5, 2024; Valencia, Spain), 51-53.

2. Optimizing React Native Project Structure: The Modular Architecture Approach // HackerNoon. URL: https://hackernoon.com/optimizing-react-native-project-structure-the-modular-architecture-approach (дата звернення: 01.03.2025).

3. React Native Render Pipeline // React Native Documentation. URL: https://reactnative.dev/architecture/render-pipeline (дата звернення: 01.03.2025).

4. Alexander Benedikt Kuttig. Professional React Native. Birmingham: Packt Publishing, 2022. – 268 p. – P. 88-102.

5. Sakhniuk M., Boduch A. React and React Native. Birmingham : Packt Publishing, 2019. – 508 p. – P. 121-131.