УДК 004.4’23

МЕТОДИ ЕФЕКТИВНОГО УПРАВЛІННЯ СТАНОМ У ANGULAR ДОДАТКАХ

Коробейник В.С.
e-mail: volodymyr.korobeinyk@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This work is devoted to analyzing state management approaches in Angular applications, including NgRx, Akita, and ngx-base-state. The features, advantages, and disadvantages of each approach are examined in terms of performance, scalability, and ease of implementation. NgRx provides centralized state management but requires a significant amount of boilerplate code. Akita balances structure and simplicity, while ngx-base-state is suitable for smaller projects. The study identifies optimal strategies depending on application scale and provides recommendations for choosing a state management solution for different types of Angular applications.

Сучасні веб-додатки, такі як CRM-системи та онлайн-платформи, вимагають ефективного управління станом для підтримки, масштабування та продуктивності. Angular пропонує різні підходи — від використання RxJs у сервісах до комплексних рішень на основі NgRx, Akita та ngx-base-state. Однак розробники часто стикаються з дублюванням логіки, зниженням продуктивності через неефективну роботу з станом, складністю тестування та масштабування. Метою роботи є аналіз методів управління станом, та надання рекомендацій щодо вибору інструментів залежно від масштабу проєкту та визначення оптимальних архітектурних патернів для Angular-додатків. Для цього розглянемо кожен із популярних стейтменеджерів, їх особливості, переваги та недоліки.

У підході з ngx-base-state створюються локальні стейти для різних частин застосунку, сама бібліотека по суті є обгорткою навколо базового підходу з RxJs [1-2] та сервісами. Загальна схема така, що компонент взаємодіє с сервісом, який у свою чергу працює з API сервісом для запитів на сервер та маніпулює даними в стейті. Стейт розшарює потік даних через сервіс на компонент (див. рис. 1).

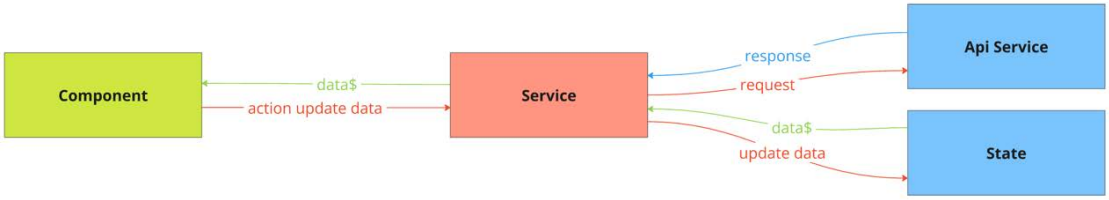

Рисунок 1 – Підхід з Ngx-base-state

Цей підхід є простим і зручним для локального управління станом, особливо в невеликих і середніх додатках. Автор даної бібліотеки хотів

створити просте рішення для гнучкого керування станом, протилежне NgRx з великою кількістю шаблонного коду. Серед його переваг — легкість впровадження, реактивність завдяки RxJs і можливість ізоляції стану в сервісах, що сприяє зручній організації коду. Водночас для великих додатків із складною логікою цей підхід може виявитися менш ефективним, оскільки може виникнути проблема із підтримкою великої кількості сервісів та дублювання станів, що ускладнює масштабування.

У підході з Akita стан зберігається в Entity Store [3], що забезпечує централізоване управління та реактивність завдяки RxJs. Akita використовує модельно-орієнтований підхід, де стан представлений у вигляді окремих моделей, а доступ до нього здійснюється через Query. Оновлення стану відбувається на рівні сервісів, які взаємодіять з store та API сервісами, що мінімізує шаблонний код (див. рис. 2).

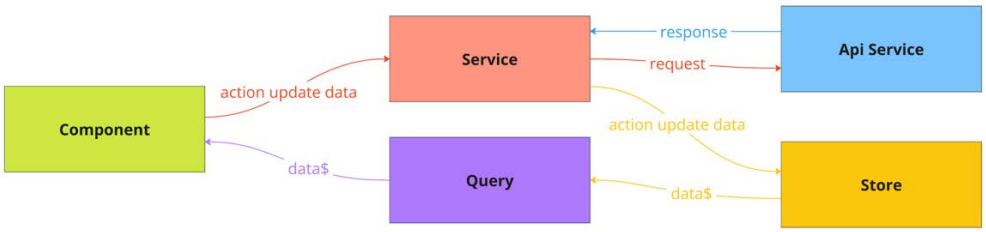

Рисунок 2 – Підхід з Akita

Завдяки вбудованому кешуванню та простому API Akita дозволяє легко керувати станом у середніх і великих додатках. Перевагами є менша складність у порівнянні з NgRx, легше впровадження та підтримка реактивного оновлення. Проте Akita не має підтримки сигналів, та має менш активну спільноту та посередню документацію, що може ускладнити довгострокову підтримку в проєктах.

У підході з NgRx стан зберігається в глобальному Store, який є централізованим місцем для управління станом додатка. Для передачі та оновлення даних використовуються actions, reducers, selectors [4], а для асинхронних операцій такі як запити до API використовуються effects (рис. 3).

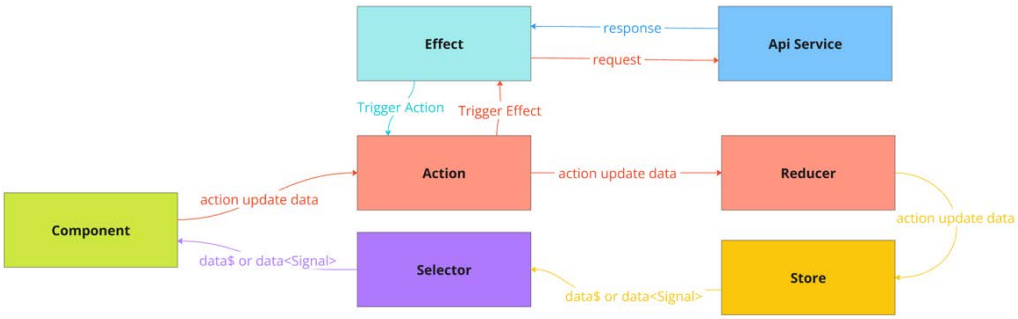

Рисунок 3 – Підхід з NgRx

Цей підхід базується на шаблоні Redux і забезпечує чітке відокремлення бізнес-логіки від компонентів. Завдяки централізованому управлінню станом та строгій архітектурі NgRx полегшує масштабування додатка, роблячи його більш структурованим і передбачуваним. Також у NgRx є підтримка сигналів, що говорить про актуальність інструменту, в цілому він має велике ком'юніті та часто використовується для розробки. Однак NgRx може бути складним для освоєння через складність архітектури. Також він потребує написання значної кількості шаблонного коду та необхідність використання додаткових бібліотек. У невеликих проектах використання NgRx може спричинити зниження продуктивності через зайві абстракції та накладні витрати.

Порівняння підходів до управління станом в Angular додатках показало, що Ngx-base-state поєднує простоту та реактивність, але його може не вистачати для складних систем. У великих проєктах його використання може призвести до створення великої кількості сервісів та стейтів без чіткої організації, що ускладнює підтримку та масштабування додатка. Akita збалансована між структурованістю та простотою, зменшує обсяг шаблонного коду порівняно з NgRx і добре підходить для середніх і великих проєктів, хоча її популярність і підтримка обмежені. NgRx забезпечує централізоване управління станом, підходить для великих додатків, але потребує більше ресурсів на освоєння й створює багато шаблонного коду, що може ускладнювати розробку.

Для малих проєктів доцільно використовувати локальне управління станом із RxJs, Signal або Ngx-base-state, оскільки ці підходи дозволяють швидко організувати управління даними без зайвої складності. Середні проєкти виграють від поєднання локального і глобального станів, де Akita може бути зручною альтернативою NgRx завдяки простішій інтеграції та меншій кількості шаблонного коду. У великих системах оптимальним вибором є NgRx із модульним підходом, lazy loading і кешуванням [5], що допомагає ефективно працювати з даними та зменшувати навантаження на додаток. При виборі підходу потрібно враховувати масштаб та особливості проєкту, а також вимог до продуктивності та підтримки додатку.

Список використаних джерел:

1. Reactive Programming with RxJS. URL: https://rxjs.dev/guide/overview (дата звернення: 25.02.2025).

2. Ngx-base-state Documentation. URL: https://github.com/ngx-base-state (дата звернення: 25.02.2025).

3. Akita Documentation. URL: https://opensource.salesforce.com/akita/ (дата звернення: 25.02.2025).

4. NgRx Documentation. URL: https://ngrx.io/guide/store (дата звернення: 25.02.2025).

5. Angular Best Practices. URL: https://v17.angular.io/guide/lazy-loading-ngmodules (дата звернення: 25.02.2025).