УДК 004.4’236

МЕТОДИ ЕФЕКТИВНОГО УПРАВЛІННЯ СТАНОМ У VUE ДОДАТКАХ

Корнев О.М.
e-mail: oleksandr.kornev@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study examines state management approaches in scalable Vue applications, focusing on Vuex, Pinia, and Composition API. Vuex offers a centralized state management model suitable for large projects but has a complex syntax. Pinia provides a simpler alternative with better TypeScript support, making it ideal for modern applications. Composition API enables lightweight local state management but is less scalable. The study highlights the benefits and limitations of each approach, emphasizing the importance of selecting the right strategy based on project needs to ensure flexibility, maintainability, and scalability.

Зростання складності сучасних веб-додатків потребує ретельного підходу до управління їх станом. Особливо це актуально для Vue-додатків, які завдяки своїй гнучкості та популярності застосовуються у проектах різного масштабу. Однак, масштабованість додатків залежить не лише від їхньої функціональності, але й від правильного підходу до управління станом. Масштабовані Vue-додатки часто стикаються з проблемами, що ускладнюють їх розвиток і підтримку: змішування бізнес-логіки з логікою компонента, неконтрольоване зростання складності стану та труднощі в тестуванні. Наприклад, перевантажені компоненти ускладнюють тестування та повторне використання, а погано структурований глобальний стан створює труднощі з доступом до даних і збільшує ризик конфліктів. Для вирішення цих проблем необхідно чітко розділяти логіку управління станом, бізнес-логіку та UI. Використання інструментів, таких як Vuex чи Pinia, допомагає структурувати стан і спрощує роботу з додатками.

У сучасних Vue-додатках існує кілька підходів до управління станом. Найпоширеніші з них: Vuex, Pinia та локальний стан із використанням Composition API. Кожен підхід має свої особливості, які необхідно враховувати залежно від потреб проєкту. Важливо не лише обрати відповідний інструмент, а й правильно організувати його використання, уникаючи надмірної складності та забезпечуючи зручність роботи з даними. Наприклад, у великих проєктах доцільно структурувати глобальний стан за модулями, а для локального керування даними використовувати реактивність Composition API. Також варто враховувати майбутнє масштабування додатку, можливість розширення

функціональності та підтримку сторонніх бібліотек, що можуть впливати на вибір підходу до управління станом.

Vuex забезпечує централізоване управління станом із підтримкою одностороннього потоку даних, розподіляючи стан на модулі для впорядкованості [1-2]. Це знижує дублювання даних між компонентами, полегшує моніторинг і налагодження за допомогою інструментів, як-от Vue DevTools. Vuex складається з кількох основних частин: State містить глобальні дані, до яких звертаються компоненти, Mutations змінюють стан, однак вони повинні бути синхронними, Actions використовуються для асинхронних операцій, таких як запити до Backend API, після чого вони комітять мутації, а Vue Components викликають дії через dispatch і отримують оновлений стан для відображення (див. рис. 1).

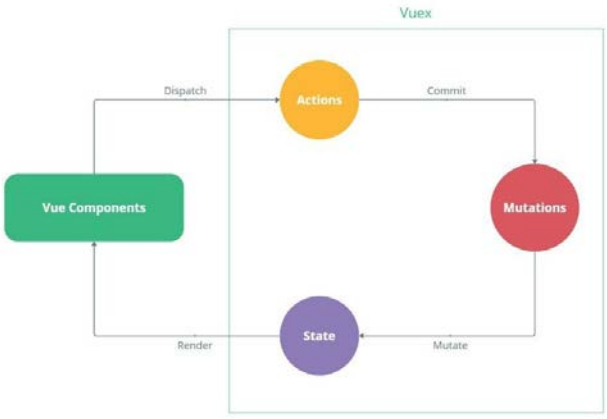

Рисунок 1 – Управління станом за допомогою Vuex

Однак Vuex вимагає багато шаблонного коду, що ускладнює його використання для новачків і робить менш гнучким у порівнянні з сучасними рішеннями, такими як Pinia.

Pinia, створений на базі Composition API, пропонує спрощений синтаксис, зменшений обсяг шаблонного коду та кращу інтеграцію з TypeScript [3]. Це робить його більш гнучким і зручним у використанні для сучасних проєктів. Pinia підтримує реактивність завдяки використанню ref та computed, що дозволяє напряму змінювати стан без потреби у мутаціях, як у Vuex. Його API мінімізує шаблонний код, дозволяючи визначати стан, дії та гетерів у простій та зрозумілій формі. Крім того, Pinia дозволяє створювати кілька сторів, що полегшує модульність та організацію логіки в масштабованих додатках. Завдяки офіційній підтримці Vue DevTools розробники отримують зручні інструменти для налагодження. Однак через новизну інструмента його екосистема менш розвинена, ніж у Vuex, що може ускладнити використання специфічних плагінів або інтеграцій.

Composition API забезпечує простий та ефективний підхід для управління даними в невеликих додатках [1]. Він не потребує сторонніх бібліотек і підходить для ізольованих компонентів. Відповідно до

діаграми, управління станом у Composition API базується на трьох основних частинах: View відповідає за відображення інтерфейсу користувача, State зберігає дані та оновлюється при зміні інформації, а Actions виконують зміну стану та реагують на події (див. рис. 2).

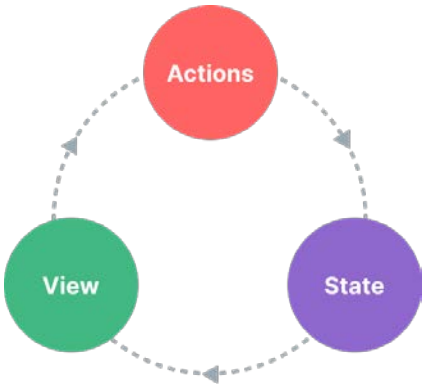

Рисунок 2 – Управління станом за допомогою Composition API

Цей підхід дозволяє зменшити складність додатку, зосереджуючись лише на необхідних функціях.

Запропоновані рекомендації висвітлюють різні підходи до організації управління станом у масштабованих Vue-додатках, враховуючи їхню специфіку та сферу застосування. Використання Vuex забезпечує перевірену часом централізовану модель управління станом, яка особливо добре підходить для великих проєктів із складною архітектурою та потребою в інтеграції з широкою екосистемою плагінів. Pinia, як сучасна альтернатива, пропонує більш простий синтаксис, кращу підтримку TypeScript і гнучкість у реалізації, що робить її ідеальним вибором для проєктів, орієнтованих на нові технології.

Натомість використання локального стану з Composition API є оптимальним рішенням для ізольованих компонентів або невеликих модулів, де централізація не є критичною. Такий підхід дозволяє уникнути зайвого ускладнення додатку, водночас залишаючись простим і інтуїтивним.

Поєднання цих підходів дозволяє збалансувати продуктивність, масштабованість і підтримуваність проєкту, даючи змогу адаптуватися до змінних потреб як на етапі розробки, так і у процесі підтримки додатку. Ретельний вибір інструменту залежить від масштабу проєкту, вимог до його функціональності та рівня експертизи команди.

Список використаних джерел:

1. Vue.js Official Guide. URL: https://vuejs.org. (дата звернення: 25.02.2024).

2. Vuex Documentation. URL: https://vuex.vuejs.org. (дата звернення: 25.02.2024).

3. Pinia Documentation. URL: https://pinia.vuejs.org. (дата звернення: 25.02.2024).