Патерни для створення маштабованих веб-додатків за допомогою React

Сергій Клевчук, студент[1] (ORCID*:* 0009-0008-6228-2425)
Ірина Пороховніченко, асист. [1] (ORCID: 0000-0001-6341-6394)

[1] *Київський національний університет будівництва і архітектури, Київ, Україна*

АНОТАЦІЯ

У тезі досліджуються найпоширеніші патерни проєктування в бібліотеці React, які допомагають структурувати код, підвищити масштабованість та повторне використання компонентів. Розглянуто підходи Higher-Order Components (HOC), Render Props, Custom Hooks та Context API. Наведено приклади коду та схеми використання. Зроблено висновок, що грамотне застосування цих патернів сприяє створенню підтримуваних, масштабованих та ефективних веб-додатків.

Ключові слова: React, патерни, компоненти, архітектура, масштабованість.

1. ВСТУП

React з'явився у 2013 році як проєкт компанії Facebook і швидко став одним із найпопулярніших інструментів для створення SPA (single-page application). Його успіх пояснюється простотою та високою продуктивністю завдяки віртуальному DOM та можливістю повторного використання компонентів. Проте зі зростанням проєктів виникла потреба у структурованих підходах для уникнення дублювання коду та забезпечення легкості підтримки. У цьому контексті патерни стали ефективним інструментом організації архітектури додатків.

Патерни в React — це узагальнені рішення типових завдань, які дозволяють зменшити складність коду та зробити його більш передбачуваним. Вони забезпечують повторне використання бізнес-логіки, спрощують взаємодію між компонентами та сприяють масштабуванню великих систем.

2. ОСНОВНІ ПАТЕРНИ В REACT

2.1. Higher-Order Components (HOC)

HOC — це функція, яка приймає компонент і повертає новий, доповнений додатковою поведінкою. Завдяки цьому можна винести повторювані завдання в один модуль. Типові приклади — перевірка прав доступу користувача, логування або відправка даних на сервер.

```
function withAuth(Component) {
  return function(props) {
    if (!props.isLoggedIn) return <LoginPage />;
    return <Component {...props} />;
  };
}
```

Основна перевага HOC — повторне використання логіки без дублювання коду. Водночас цей патерн має недолік, відомий як «wrapper hell», коли багаторівнева композиція обгорток ускладнює розуміння структури коду.

2.2. Render Props

Патерн Render Props полягає у передачі функції як дочірнього елемента для динамічного управління вмістом компонента. Це дозволяє будувати універсальні компоненти-контейнери, які надають дані у різних форматах.

```
<DataProvider render={(data) => (
  <h1>{data.title}</h1>
)} />
```

Render Props був особливо популярний до появи React Hooks. Його гнучкість дозволяє уникати дублювання логіки, але він робить код більш вкладеним, що знижує його читабельність.

2.3. Custom Hook

Custom Hooks є найбільш сучасним і рекомендованим способом повторного використання логіки. Вони дозволяють винести бізнес-логіку у функції, які легко тестувати та застосовувати у багатьох компонентах. Наприклад, можна створити `useFetch` для роботи з API або `useForm` для керування станом форм.

```
function useLocalStorage(key, initialValue) {
  const [value, setValue] = React.useState(() =>
{
    return localStorage.getItem(key) ||
initialValue;
  });
  React.useEffect(() => {
    localStorage.setItem(key, value);
  }, [key, value]);
  return [value, setValue];
}
```

Custom Hooks стали стандартом у сучасних проектах завдяки простоті, зрозумілості та легкості інтеграції. Їх перевага — можливість інкапсуляції будь-якої логіки (робота з API, кешування, анімації) у вигляді функцій.

2.4. Context API

Context API забезпечує механізм передачі даних між компонентами без необхідності прокидати пропси через усю ієрархію. Це особливо актуально для глобальних даних: теми інтерфейсу, локалізації чи налаштувань користувача.

```
const ThemeContext =
React.createContext("light");

function Toolbar() {
  const theme = React.useContext(ThemeContext);
  return <div className={theme}>Toolbar</div>;
}
```

Контекст спрощує структуру проєкту та зменшує дублювання коду. Однак його надмірне використання може

викликати проблеми з продуктивністю через часті ререндери компонентів.

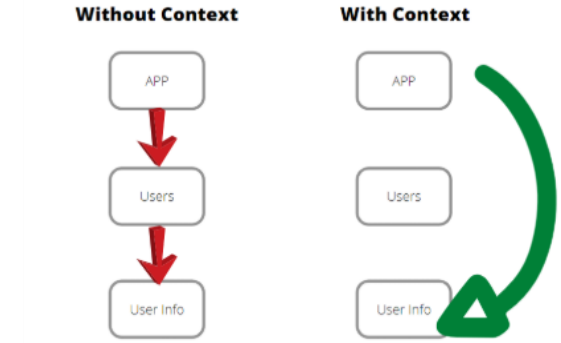

Рисунок 1 – Схема передачі даних через Context API

Таблиця 1 – Порівняльна характеристика патернів у React

Патерн	Переваги	Недоліки
HOC	Повторне використання логіки	Складність при вкладеності
Render Props	Гнучкість у побудові	Зниження читабельності
Custom Hooks	Сучасний і простий підхід	Можливе дублювання логіки
Context API	Зручний для глобальних даних	Може впливати на продуктивність

3. ОБГОВОРЕННЯ

Таблиці та HOC та Render Props мали велике значення на початковому етапі розвитку React, проте сьогодні їх витіснили Custom Hooks і Context API завдяки простоті та зрозумілості. Практика показує, що HOC добре підходять для задач авторизації або відстеження аналітики, але швидко призводять до «wrapper hell» і ускладнюють відлагодження. Render Props забезпечують високу гнучкість та універсальність, однак роблять код менш читабельним, особливо у великих командах. Custom Hooks дозволяють ізолювати бізнес-логіку у функціях, що суттєво полегшує повторне використання та тестування, але при неправильному проектуванні може виникати дублювання. Context API, у свою чергу, спрощує роботу з глобальними даними, проте надмірне його використання спричиняє часті перерендери та втрату продуктивності. Отже, еволюція патернів у React відображає прагнення до більш чистого, передбачуваного та масштабованого коду, але жоден із підходів не є універсальним. У реальних умовах розробники часто комбінують кілька патернів, щоб досягти балансу між простотою та гнучкістю архітектури.

4. ПЕРСПЕКТИВИ РОЗВИТКУ

Якщо React Server Components змінюють парадигму рендерингу, оскільки логіка обробляється на сервері, а клієнт отримує мінімум даних. Це дає змогу зменшити розмір бандлу та підвищити швидкість завантаження. Suspense дозволяє реалізувати асинхронний рендеринг безчисленних перевірок стану завантаження. Signals, які вже працюють у Preact і SolidJS, потенційно можуть стати основою нового механізму управління станом у React. У поєднанні з TypeScript вони забезпечують типобезпечність і передбачуваність архітектури, що особливо важливо для великих корпоративних додатків.

5. ПРАКТИЧНЕ ЗАСТОСУВАННЯ

Розглянемо умовний кейс побудови веб-застосунку у стартапі. На початковому етапі достатньо застосування Custom Hooks для роботи з формами та API, що дозволяє швидко прототипувати функціонал і перевіряти гіпотези. Коли проєкт розширюється, додається Context API для глобальних налаштувань користувача, наприклад теми інтерфейсу чи мови. У середніх проєктах, наприклад SaaS-платформах, доцільно комбінувати Context API з локальними хуками для управління складними станами, що забезпечує баланс між простотою та масштабованістю. У корпоративних системах (банківські додатки, CRM) зазвичай додають Redux або Zustand, щоб централізувати логіку та забезпечити масштабованість. Такі рішення роблять архітектуру більш передбачуваною та зручною для роботи великих команд, де важлива чітка структура коду. Крім того, вони підвищують надійність системи та спрощують її довгострокову підтримку.

6. ВИСНОВКИ

Патерни React є ключовим інструментом для побудови систем різного масштабу. Для невеликих стартапів найзручнішими залишаються Custom Hooks, які дозволяють швидко створювати прототипи та повторно використовувати бізнес-логіку. У середніх проєктах доцільним є поєднання Custom Hooks і Context API для централізованого управління станом та організації глобальних даних, наприклад теми чи локалізації. У великих корпоративних системах, де важливі стабільність і передбачуваність, зазвичай інтегрують Context API із зовнішніми бібліотеками стану (Redux, Zustand), що дає змогу централізувати логіку й забезпечити масштабованість. Вибір патерну завжди має враховувати довгострокову підтримку та складність проєкту. У перспективі розвиток React Server Components, Suspense та Signals може змінити підхід до архітектури, зменшити навантаження на клієнт і зробити розробку більш ефективною.

Список літератури

[1] Abramov D., Clark B. React Documentation. Meta Platforms, 2024.

[2] Freeman E., Robson E. Head First Design Patterns. O’Reilly Media, 2021.

[3] Wieruch R. The Road to React. Independent, 2023

[4] Kent C. Dodds. Advanced React Patterns. Egghead, 2022