https://doi.org/10.31891/2219-9365-2024-79-6
УДК: 681.3.06

ПАНЧИШИН Богдан
Національний Університет "Львівська Політехніка"
https://orcid.org/0009-0009-1366-1602
e-mail: panchyshyn33@gmail.com

АНАЛІЗ ПРИНЦИПІВ РОБОТИ КОМПОНЕНТІВ ФРЕЙМВОРКА REACT JS

У статті проаналізовані і описані принципи роботи та структура react компонентів. Також наведені приклади їх ефективного застосування на практиці верстки веб-застосунку. Представлені варіанти створення компонентів React, які є основою для його модульної та багаторазово використовуваної архітектури.

Ключові слова: react компоненти, властивості state та props, react js, virtual dom.

PANCHYSHYN Bohdan
Lviv Polytechnic National University

ANALYSIS OF THE WORKING PRINCIPLES OF REACT JS FRAMEWORK COMPONENTS

This topic is relevant because React is the most popular JavaScript programming language framework. Knowing the principles of building and the structure of react components has the ability to create extremely complex web applications that will be fast, functional and efficient at the same time.

The main focus of React is in managing the state of components and updating the used interface with minimal resource consumption. Below you can see the basic principles in React:

1. Components
2. Virtual DOM
3. Unidirectional data flow
4. JSX

Component reuse makes it easy to create and display similar UI elements that behave the same across applications. This facilitates code reuse and supports modularity and maintainability of the codebase. The component approach in React allows you to create small and reusable blocks of code elsewhere in your web application.

Knowledge of the principles of work and the structure of react components provides opportunities to create functional, simple, fast and optimized web applications that are easily modernized during development.

One of the key advantages of using react components is the ability to break complex and large user interfaces into small blocks that can be used anywhere in the project and more than once. Each component contains its own logic, user interface, and state, which in turn makes codebases easier to maintain, especially in large projects, and operates independently, making debugging, testing, and code analysis easier.

In addition, the examples given and analyzed above will help to better understand the very work of components, their structure, options for creation and application in projects, and how to correctly and successfully manipulate data using states and properties.

Keywords: react components, state properties and props, react js, virtual dom.

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Було складно та ризиковано оновлювати інтерфейс користувача шляхом безпосереднього маніпулювання об'єктною моделлю документа (DOM). Рідке використання DOM під час прямих маніпуляцій часто призводило до неефективного рендерингу та труднощів у підтримці відповідного інтерфейсу користувача. Низька продуктивність була типовою проблемою традиційних веб-додатків, особливо коли вони мали справу з великими та динамічними інтерфейсами користувача.

DOM може бути надто сильно модифікований, що призведе до повільного відтворення та затримки роботи користувача. Функціональність інтерфейсу користувача не вдалося оновити на основі змін у стані програми. Відсутність архітектури на основі компонентів перешкоджала здатності розбити інтерфейс користувача на модульні та багаторазово використовувані компоненти у звичайних підходах. Співпраця заважала, а програми було важко масштабувати.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Однією з ключових переваг використання компонентів react – це можливість розбивати складні та великі інтерфейси користувача на маленькі блоки, які можна використовувати в будь-якому місці проекту і не один раз. Кожен компонент містить в собі власну логіку, інтерфейс користувача і стан, що в свій час дозволяє легше підтримувати кодові бази, особливо у великих проектах та працює незалежно, що полегшує роботу в налагодженні, тестуванні та аналізі коду.

Структуру компонента react та його комунікацію між батьківським та дочірнім компонентами, а також застосування та призначення state і props досліджено у цій роботі. Якщо state надають можливість

керувати даними в середині компоненти, то props – забезпечують передачу цих даних між компонентами. Одно-направлений потік в компоненті дозволяє краще керувати проектом. Віртуальний DOM – ререндерить тільки ту частину веб-інтерфейсу, яка саме була змінена при взаємодії з користувачем, що забезпечує кращу продуктивність.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою роботи є: Дослідити принципи роботи та структуру компонентів React JS.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Ця тема є актуально, оскільки react є найбільш популярним фреймворком мови програмування JavaScript. Знання принципів побудови та структури react компонентів дає можливість створювати надскладні веб-застосунки, які в водночас будуть швидкими, функціональними та ефективними.

Основний фокус React полягає в управлінні станом компонентів та оновленні користувацького інтерфейсу з мінімальними витратами ресурсів. Основні принципи у React:

1. Компоненти – Користувацький інтерфейс розбивається на невеликі незалежні компоненти, кожен компонент представляє собою окрему частину інтерфейсу та містить в собі логіку і представлення про нього.

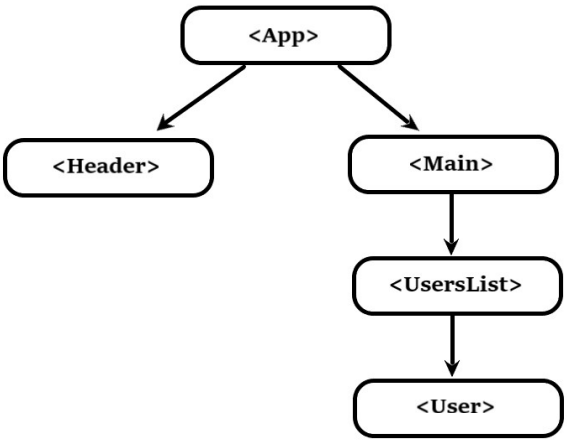

Рис. 1. Структура веб-додатку побудованого на React компонентах

2. Virtual DOM – використовується для ефективного управління оновленням інтерфейсу, замість прямого змінення реального DOM, react створює Virtual DOM та порівнює його з поточним станом для визначення мінімальних змін які потрібно внести.

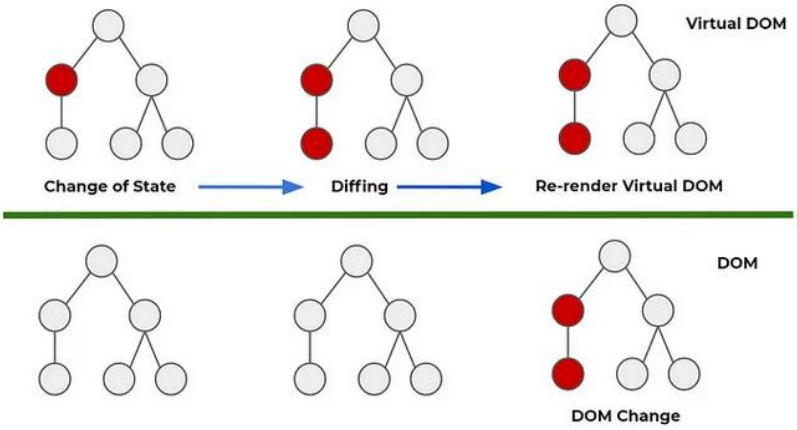

Рис. 2. Принцип роботи react над оновленням інтерфейсу з допомогою Virtual DOM

3. Одно-направлений потік даних – дані в react, як правило передаються вниз по ієрархії компонентів це означає що, зміни в батьківському компонентів також можуть впливати і на дочірні, але не навпаки, це забезпечує передбачуваність та управління при розробці додатків.

ISSN 2219-9365

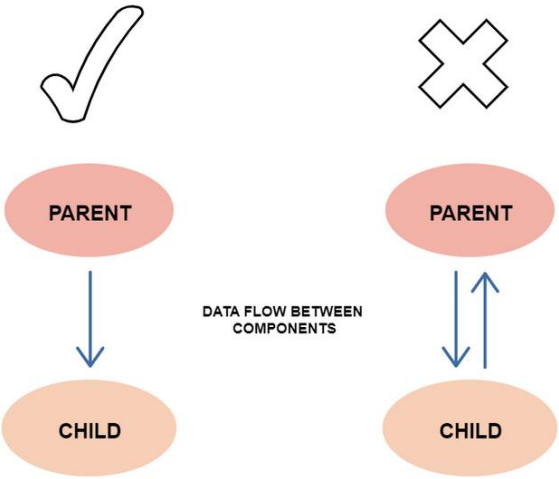

Рис.3. Односпрямований потік даних в react

4. JSX - це синтаксичне розширення JavaScript, яке дозволяє описувати структуру користувацького інтерфейсу в середині коду, це робить код більш зрозумілим та зручним для розробників.

Рис.4. Порівняння React з JSX і без JSX

Досліджено приклад коду із повторним використанням react компонентів:

User - це простий компонент, який приймає текстові атрибути і відображає їх в div.

```
import React from "react";
const User = ({ firstName, lastName }) => {
  return (
    <div>
      <span>{firstName}</span>
      <span>{lastName}</span>
    </div>
  );
}
export default User;
```

Компонент **App** імпортує та використовує User кілька разів, передаючи різні текстові значення як атрибути. Відповідно ми також можемо передати не тільки текстові значення, а й інші типи даних.

```
import './App.css';
import User from './comp/User';
function App() {
 return (
   <div className="App">
<User firstName="John" lastName="Wick" />
<User firstName="Harry" lastName="Potter" />
<User firstName="Jimm" lastName="Carry" />
   </div>
 );
}
export default App;
```

ISSN 2219-9365

Повторне використання компонента User полегшує створення та відображення схожих елементів інтерфейсу, які поводяться однаково в різних додатках. Це полегшує повторне використання коду та підтримує модульність і ремонтопридатність кодової бази. Компонентний підхід в React дозволяє створювати невеликі та з можливістю повторно використати блоки коду в іншому місці веб-додатку.

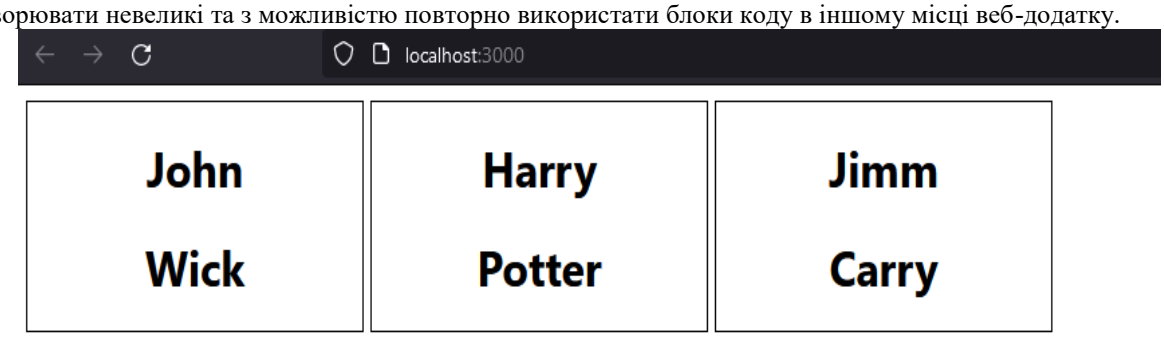

Рис.5. Результат роботи коду

Якщо ми говоримо про структуру компонентів, тут важливо виділити 4 основні критерії:

1. Функціональні компоненти:

Функціональні компоненти – це функції JavaScript, які не мають стану. Вони набагато простіші та зрозуміліші ніж класові компоненти, але з появою React Hooks функціональні компоненти отримали можливість керувати локальним станом і побічними ефектами завдяки хукам useState та useEffect.

```
import React from 'react';
const MyFunctionalComponent = () => {
  return (
    <div>
      {/* JSX representing the component's UI */}
    </div>
  );
};
```

2. Класові компоненти

Класові компоненти розширюють клас React.Component та входять до сімейства JavaScript. Вони побудовані з використанням синтаксису класу ES6 і надають більш розширені функції, такі як методи життєвого циклу та керування станом. Щоб створити компонент класу, вам потрібно успадкувати клас Component з бібліотеки React. Обов’язково потрібний метод render, він повертає вигляд компонента(JSX).

```
import React, { Component } from 'react';
class MyClassComponent extends Component {
  render() {
    return (
      <div>
        {/* JSX representing the component's UI */}
      </div>
    );
  }
}
```

3. Props

У React props (реквізити) надають можливість комунікації від батьківського до дочірнього компонента. Іншими словами – це канал зв’язку, який дозволяє обмінюватись інформацією, при чому ця інформація може бути представлена в будь-якому типі даних. Функціональні компоненти отримують props як параметри, а компоненти класу отримують до них доступ через this.props.

```
// Functional Component
const MyFunctionalComponent = (props) => {
  return <div>{props.message}</div>;
};

// Class Component
```

```
class MyClassComponent extends Component {
  render() {
    return <div>{this.props.message}</div>;
  }
}
```

4. Стани.

Стан — це вбудована функція React, яка надає можливість компонентам керувати внутрішніми даними та зберігати їх. Він представляє поточний стан компонента та може бути оновлений з часом, зазвичай у результаті взаємодії користувача або асинхронних операцій. Тобто до нього можна отримати доступ і змінити його лише в межах цього компонента або його дочірніх компонентів. Функціональні компоненти використовують хук useState, а класові компоненти об'єкт this.state.

Розберемо як працювати з властивостями state та props, на прикладі функціональних компонентів.

```
import React, { useState } from 'react';
//Parent Component
const Parent = () => {
  const [count, setCount] = useState(0); // State властивість

  const incrementCount = () => {
    setCount(count + 2)
  };

  return (
    <div>
      <h1>Parent</h1>
      <p>Count: {count}</p>
      <Child count={count} />
      <button onClick={incrementCount} >Increment</button>
    </div>
  );
};

//Child Component
const Child = (props) => {
  const { count } = props; // Props властивість

  return (
    <div>
      <section>
        <h2>Child</h2>
        <p>Received Count from Parent: {count}</p>
      </section>
    </div>
  );
};

export default Parent;
```

Батьківський компонент (Parent):

Початковий стан змінної count визначено завдяки хуку useState та дорівнює 0. У JSX компоненті Parent стан count відображається з допомогою {count}. Створено стрілочну функцію incrementCount, яка використовується, щоб оновити стан count, при натисканні кнопки. При рендері компонента Child стан count передається, як властивість (count={count}).

Дитячий компонент (Child):

Ми отримуємо властивість count від Parent компонента за допомогою props. Потім відображаємо це у Child компоненті за допомогою {count}.

У цьому прикладі показано, як батьківський компонент може керувати та оновлювати внутрішній стан (count) і передавати його як властивість дочірнім компонентам. В свою чергу дочірній компонент отримане значення використовує, щоб відобразити значення лічильника. Розуміючи різницю між state та props, ми з легкістю забезпечуємо чіткий потік даних і полегшуємо процес повторного використання компонентів. Батьківські компоненти зберігають власний стан (state), а дочірні компоненти отримують дані через властивості(props), що полегшує контроль і зв'язок між компонентами.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Знання принципів роботи та структури react компонентів надає можливості створення функціональних, простих, швидких та оптимізованих веб-додатків, які легко піддаються модернізації в ході розробки.

Окрім того наведені та проаналізовані вище приклади, допоможуть краще зрозуміти саму роботу компонентів, їх структуру, варіанти створення та застосування у проектах і як правильно та успішно маніпулювати даними за допомогою станів(state) та властивостей(props).

Література

1. Minnick C. ReactJS Foundations. Building User Interfaces with ReactJS [Електронний ресурс] / Chris Minnick // WROX A Wiley Brand. – 2022. – Режим доступу до ресурсу: https://dl.ebooksworld.ir/books/Beginning.ReactJS.Foundations.Chris.Minnick.Wiley.9781119685548.EBooksWorld.ir.pdf.

2. ReactJS official documentation [Електронний ресурс] // Meta Platforms. – 2013. – Режим доступу до ресурсу: https://react.dev/learn.

3. Dave G. React JS Full Course for Beginners | Complete All-in-One Tutorial | 9 Hours [Електронний ресурс] / Gray Dave. – 2022. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=RVFAyFWO4go.

4. Fedosejev A. React.js Essentials [Електронний ресурс] / Artemij Fedosejev // Packt Publishing Ltd. – 2015. – Режим доступу до ресурсу: https://strukturnifondovi.hr/wp-content/uploads/2017/03/9781783551620-REACTJS_ESSENTIALS.pdf.

References

1. Minnick C. ReactJS Foundations. Building User Interfaces with ReactJS [Electronic resource] / Chris Minnick // WROX A Wiley Brand. – 2022. – Resource access mode: https://dl.ebooksworld.ir/books/Beginning.ReactJS.Foundations.Chris.Minnick.Wiley.9781119685548.EBooksWorld.ir.pdf.

2. ReactJS official documentation [Electronic resource] // Meta Platforms. – 2013. Resource access mode: https://react.dev/learn.

3. Dave G. React JS Full Course for Beginners | Complete All-in-One Tutorial | 9 Hours [Electronic resource] / Gray Dave. – 2022. – Resource access mode: https://www.youtube.com/watch?v=RVFAyFWO4go.

4. Fedosejev A. React.js Essentials [Electronic resource] / Artemij Fedosejev // Packt Publishing Ltd. – 2015. – Resource access mode: https://strukturnifondovi.hr/wp-content/uploads/2017/03/9781783551620-REACTJS_ESSENTIALS.pdf.