Лекція 1

1) Що таке React
2) Налаштування та інструменти
3) JSX
4) Компоненти
5) Компоненти класси vs Функціональні компоненти

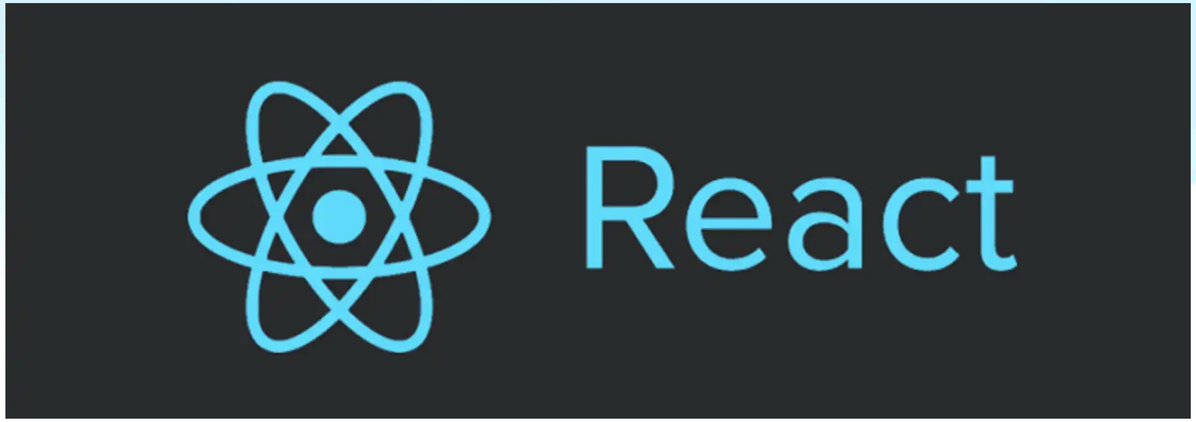
React

React – це бібліотека для створення елементів інтерфейсу користувача.

У React немає вбудованої маршрутизації, HTTP-модуля тощо. Проте є багата екосистема, яка дозволить вирішити будь-яке завдання.

При створенні застосунку з використанням React розробник не взаємодіє безпосередньо з DOM-деревом. Його завдання – описати інтерфейс за допомогою компонентів (шаблон) та керувати зміною даних (модель). React, при зміні даних моделі, сам оновить інтерфейс за шаблоном.

React – мультиплатформний, розмітку можна рендерити на сервері (Next.js), писати нативні (React Native) або десктопні (Electron) застосунки.



Налаштування та інструменти

Для створення React-застосунку необхідні Node.js, Webpack, Babel, React і DevTools. Можна написати свою Webpack-збірку або взяти будь-яку хорошу з GitHub.

Найпоширеніші збірки для створення React-застосунку наразі це Vite **та** Create React App

Ми з вами будемо розгядати саме Create React App

Налаштування та інструменти

Create React App

Для навчання та маленьких/середніх проектів рекомендується використовувати утиліту від авторів React.

- Абстрагує всю конфігурацію, дозволяючи зосередитись на написанні коду
- Включає необхідні інструменти: Webpack, Babel, ESLint тощо.
- Розширюється додатковими пакетами з екосистеми React
- Має функцію вилучення, яка видаляє абстракцію і відкриває конфігурацію

Для того що створити свій перший додаток відкрийте термінал в папці де буде знаходитись ваш проект і пропишіть команду:

“npx create-react-app ім’я_проекту” або “npx create-react-app .” Де крапка буде означати створити проект у поточний папці.

Налаштування та інструменти

Create React App

npx — інструмент, призначений для того, щоб допомогти стандартизувати використання npm-пакетів. Постачається з npm версії 5.2.0 та вище. npm спрощує встановлення та керування залежностями, розміщеними в реєстрі, а npx спрощує використання CLI-утиліт та інших файлів, що виконуються, без необхідності їх встановлення в систему або проект.

Налаштування та інструменти

Create React App

Так буде виглядати структура папок після ініціалізації проекту за допомогою “create react app”

```
my-app/
  README.md
  node_modules/
  package.json
  public/
    index.html
    favicon.ico
  src/
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
```

Для коректного білду проекту ці файли повинні **мати з точні назви:**

- public/index.html це шаблон сторінки
- src/index.js є точкою входу JavaScript

Ви можете видалити або перейменувати інші файли.

Ви можете створювати підкаталоги всередині src. Webpack обробляє лише файли всередині src. Вам потрібно **помістити файли JS і CSS у src,** інакше webpack їх не побачить.

Налаштування та інструменти

Create React App

Доступні скріпти:

npm start

Запускає програму в режимі розробки. Відкрийте http://localhost:3000, щоб переглянути його у браузері.

Сторінка перемальовується автоматично, якщо ви внесете зміни.

npm test

Запускає програму тестування.

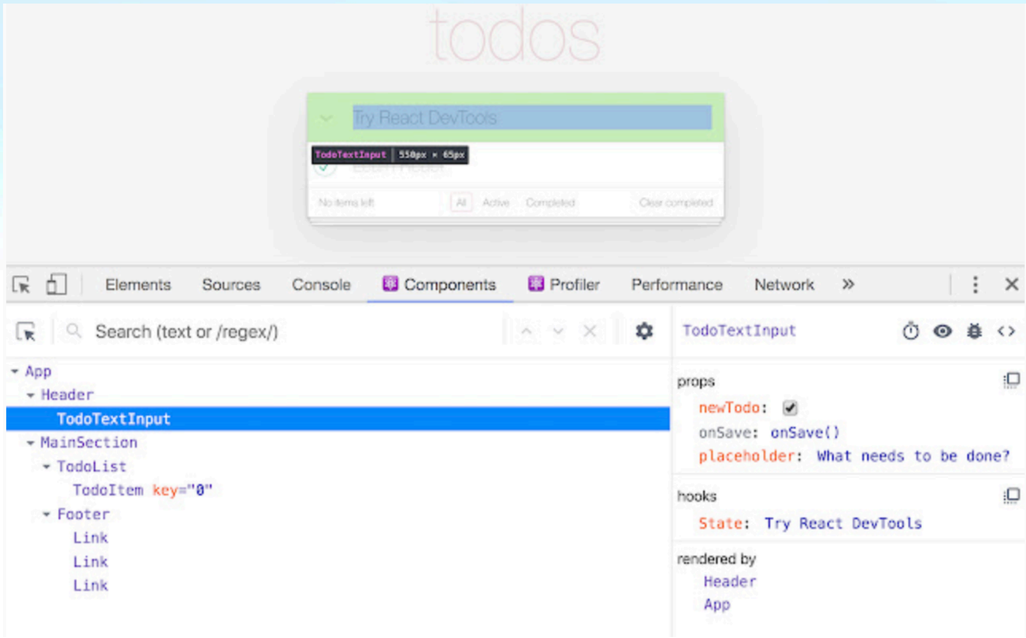
npm build

Білдить програму в папку збірки. Він правильно поєднує React у робочому режимі та оптимізує збірку для найкращої продуктивності.
Мінімізує збірку. Імена файлів містять хеші. Якщо необхідно, імена класів і імена функцій можна ввімкнути для цілей профілювання.

Налаштування та інструменти

React Developer Tools

React Developer Tools — це розширення Chrome DevTools для бібліотеки React JavaScript з відкритим кодом. Це дозволяє вам перевіряти ієрархії компонентів React в інструментах розробника Chrome.

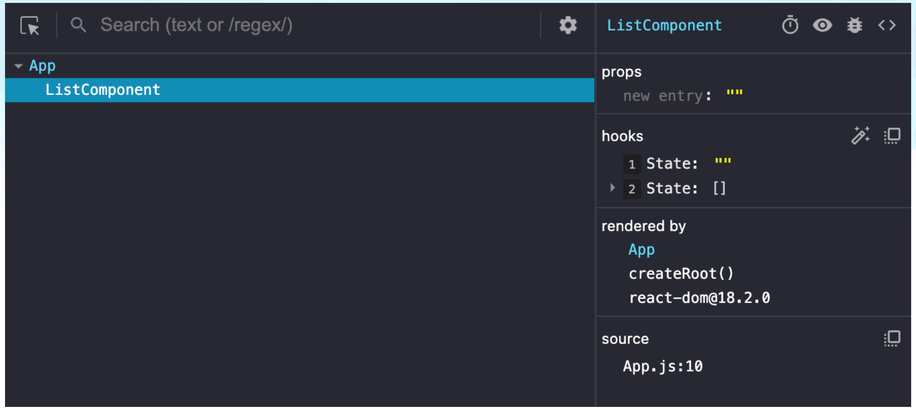


Ви отримаєте дві нові вкладки в Chrome DevTools: «⚛ Components» та «⚛ Profiler».

Налаштування та інструменти

React Developer Tools

Вкладка «Components» показує вам кореневі компоненти React, які були відрендерені на сторінці, а також підкомпоненти, які вони в кінцевому підсумку відобразили.



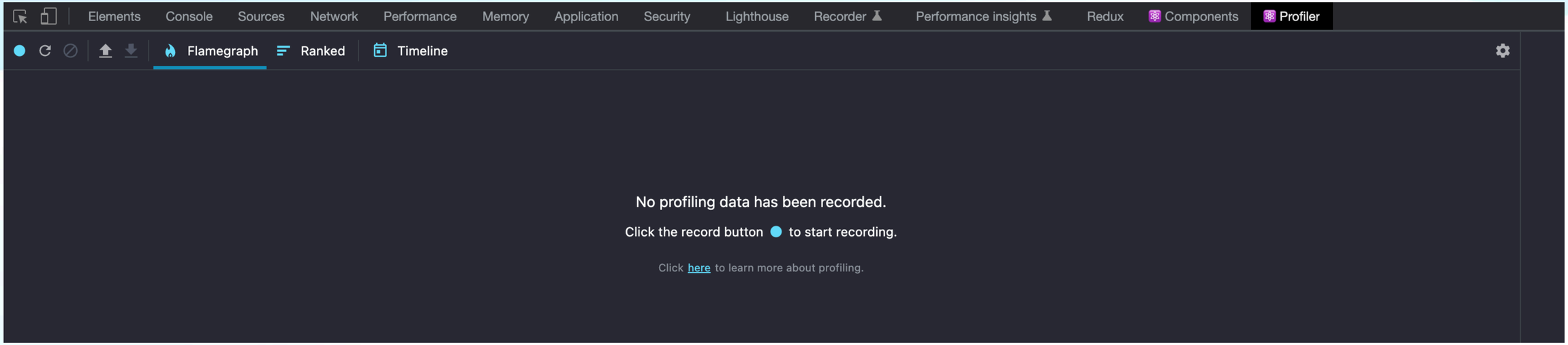
Вибравши один із компонентів у дереві, ви можете перевірити та відредагувати його поточні властивості та стан на панелі праворуч. У навігаційних ланцюжках ви можете перевірити вибраний компонент, компонент, який його створив, компонент, який створив цей, і так далі.

Якщо ви перевірите елемент React на сторінці за допомогою звичайної вкладки Elements, а потім перейдете на вкладку React, цей елемент буде автоматично вибрано в дереві React.

Налаштування та інструменти

React Developer Tools

Вкладка Profiler дозволяє записувати інформацію про продуктивність.



JSX

```jsx
<button onClick={() => onClickHandler(input)}>Add TO DO</button>
```

Це не рядок і не HTML, цей XML-образний синтаксис називається JSX (JavaScript Syntax Extension) - розширення синтаксису JavaScript, за допомогою якого зручно описувати розмітку того, що ми хочемо побачити на екрані.

- Дозволяє використовувати XML-образний синтаксис прямо у JavaScript
- Спрощує код, робить його декларативним та читабельним
- Описує об'єкти - елементи Virtual DOM
- Це не HTML, Babel трансформує JSX у виклики функцій
- У JSX можна використовувати всі можливості JavaScript

JSX створює елементи – найменші будівельні блоки React. Елементи Virtual DOM це звичайні JavaScript об'єкти, тому створювати їх дуже швидко.

JSX дуже схожа на звичайний HTML

```jsx
return (
  <div>
    <p>{count}</p>
    <button onClick={increment}>Increment me</button>
    <button onClick={decrement}>Decrement me</button>
  </div>
);
```

JSX

- Усередині JSX можна використовувати будь-який JavaScript вираз, обертаючи його в фігурні дужки “{ }”.
- Значення атрибутів вказуються в подвійних лапках, якщо це звичайний рядок, та у фігурних дужках, якщо значення обчислюється, або тип відрізняється від рядка.
- Всі атрибути React-елементів іменуються в camelCase нотації.
- JSX-теги можуть бути батьками інших JSX-тегів. Якщо тег порожній або самозакривається, його обов'язково необхідно закрити використовуючи “/>”.

```jsx
<>
  <input
    onChange={onChangeHandler}
    onKeyDown={onEnterHandler}
    value={input}
    placeholder='new task'
  />
  <p>{item.length}</p>
  <ul>
    {item.map((element, index) => (
      <ListItemComponent element={element} index={index} />
    ))}
  </ul>
  <button onClick={() => onClickHandler(input)}>Add TO DO</button>
</>
```

JSX

Правило спільного батька

Невалідний синтаксис, має повертатись одне значення.

```
return (
    <p>{count}</p>
    <button onClick={increment}>Increment me</button>
    <button onClick={decrement}>Decrement me</button>
);
```

Валідний синтаксис.

```
return (
  <div>
    <p>{count}</p>
    <button onClick={increment}>Increment me</button>
    <button onClick={decrement}>Decrement me</button>
  </div>
);
```

JSX

Правило спільного батька

```
return (
  <React.Fragment>
    <p>{count}</p>
    <button onClick={increment}>Increment me</button>
    <button onClick={decrement}>Decrement me</button>
  </React.Fragment>
);
```

Якщо в розмітці зайвий тег-обгортка не потрібний, використовуються фрагменти, схожі на DocumentFragment. Цей вбудований компонент при рендері розчиняється, підставляючи свій вміст.

```
return (
  <>
    <p>{count}</p>
    <button onClick={increment}>Increment me</button>
    <button onClick={decrement}>Decrement me</button>
  </>
);
```

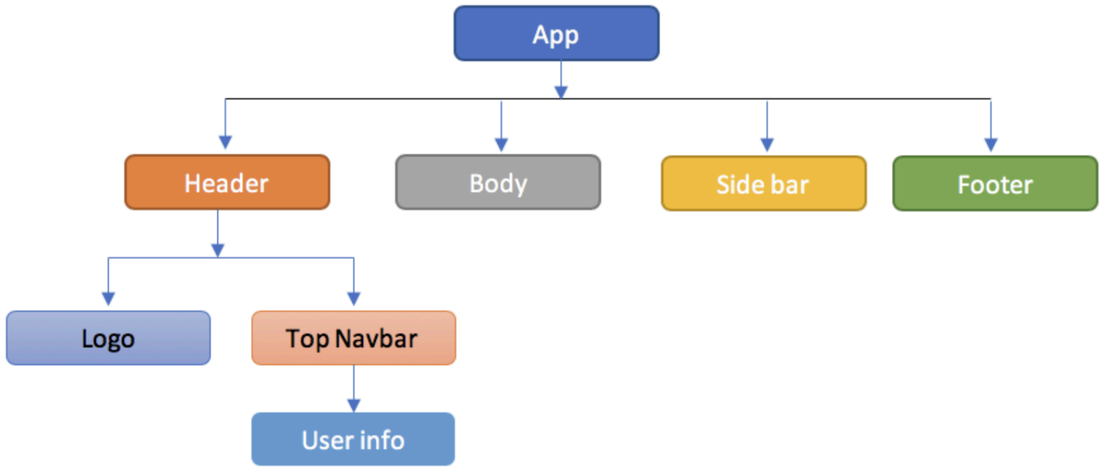
Синтаксис фрагментів можна скоротити та не додавати імпорт Fragment. Babel зробить всі необхідні трансформації, замінивши порожні JSX-теги на React.Fragment.

Компоненти

Компоненти – основні будівельні блоки React-застосунків, за допомогою яких інтерфейс розділяється на незалежні частини.

React-застосунок можна уявити як дерево компонентів. На верхньому рівні стоїть кореневий компонент, у якому вкладена довільна кількість інших компонентів. Кожен компонент повинен повернути JSX-розмітку, тим самим вказуючи, який HTML ми хочемо відрендерити в DOM.

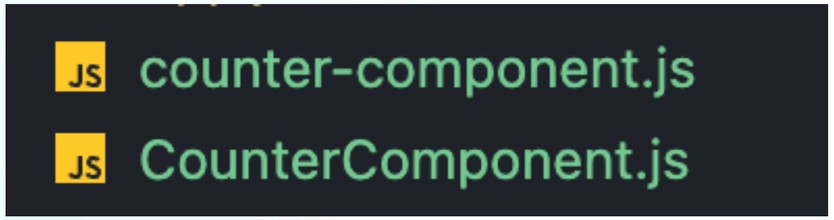
Розробник створює невеликі компоненти, які можна поєднувати, щоб сформувати більші, або використовувати їх як самостійні елементи інтерфейсу. Найголовніше в цій концепції те, що і великі, і маленькі компоненти можна використовувати повторно і в поточному, і в новому проекті.



Компоненти

Ім'я компонента обов'язково повинно починатися з великої літери. Назви компонентів з маленької літери зарезервовані для HTML-елементів. Якщо ви спробуєте назвати компонент card, а не Card, під час рендеру React проігнорує його та відрендерить тег <card></card>.

Гарною практикою вважається називати файли компонентів так само як і сам компонент. Але це не обов’язково.




```jsx
const CounterComponent = () => {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount((prevState) => prevState + 1);
  };

  const decrement = () => {
    setCount((prevState) => prevState - 1);
  };

  return (
    <>
      <p>{count}</p>
      <button onClick={increment}>Increment me</button>
      <button onClick={decrement}>Decrement me</button>
    </>
  );
};

export default CounterComponent;
```

Классові vs Функціональні компоненти

У React є два основні способи створення компонентів: класові компоненти і функціональні компоненти.

Класові компоненти - це компоненти, які успадковуються від базового класу “`React.Component`“ Вони використовуються, коли потрібно зберігати внутрішній стан компонента або використовувати методи життєвого циклу React. Класові компоненти мають власний об'єкт стану (`state`), який може бути оновлений за допомогою методу ”`setState()`“. Вони також можуть мати методи, такі як ”`componentDidMount()`”, ”`componentDidUpdate()`” та інші, які викликаються на різних етапах життєвого циклу компонента.

```
class MyClassComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    console.log('Component mounted');
  }

  componentDidUpdate(prevProps, prevState) {
    console.log('Component updated');
  }

  handleClick() {
    this.setState(prevState => ({
      count: prevState.count + 1
    }));
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.handleClick()}>Increment</button>
      </div>
    );
  }
}
```

Классові vs Функціональні компоненти

Функціональні компоненти - це простіша форма компонентів, які базуються на функціях. Вони не мають власного стану та методів життєвого циклу, але вони можуть використовувати хуки (hooks), які дозволяють їм взаємодіяти зі станом та функціональністю React. Функціональні компоненти зазвичай коротші та більш прості у використанні.

Hooks - Хуки це новинка в React 16.8. Вони дозволяють вам використовувати стан та інші можливості React без написання класу.

```jsx
function MyFunctionalComponent() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('Component mounted');
    return () => {
      console.log('Component unmounted');
    };
  }, []);

  useEffect(() => {
    console.log('Component updated');
  }, [count]);

  const handleClick = () => {
    setCount((prevCount) => prevCount + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={handleClick}>Increment</button>
    </div>
  );
}

export default MyFunctionalComponent;
```

Классові vs Функціональні компоненти

1. **Синтаксис:** У класових компонентах використовується синтаксис класів JavaScript, де клас компонента розширює базовий клас React.Component. У функціональних компонентах використовується синтаксис функцій JavaScript, де компонент є просто функцією.

2. **Стан компонента**: Класові компоненти можуть мати внутрішній стан, який зберігається у полі `state`. Функціональні компоненти не мають власного внутрішнього стану за замовчуванням. Зате з'явилася можливість використовувати стан та інші функціональні можливості завдяки введенню хуків (hooks) в React.

3. **Життєвий цикл:** Класові компоненти мають життєвий цикл, який складається з методів, таких як `componentDidMount`, `componentDidUpdate`, `componentWillUnmount` та інших. Ці методи дозволяють виконувати певні дії на різних етапах життєвого циклу компонента. У функціональних компонентах можуть використовуватися хуки, такі як `useEffect`, для досягнення подібного функціоналу.

4. **Пропси**: Якщо йдеться про передачу даних у компонент, в обох випадках використовується властивість `props`. У класових компонентах доступ до пропсів здійснюється за допомогою `this.props`, тоді як у функціональних компонентах пропси передаються як аргумент функції.

5. **Розмір коду:** Функціональні компоненти зазвичай мають менше коду, ніж класові компоненти, оскільки вони не вимагають введення додаткових методів та конструкторів.

Классові vs Функціональні компоненти

Переваги класових компонентів:

- Зручний доступ до життєвого циклу компонента, що дозволяє виконувати певні дії на різних етапах (монтаж, оновлення, розмонтаж).
- Легша розширюваність за допомогою наслідування інших класів або компонентів.
- Збереження внутрішнього стану без використання додаткових засобів (хоча з введенням хуків ця перевага стає менш суттєвою).

Переваги функціональних компонентів:

- Простота синтаксису та читабельність коду, оскільки вони вимагають менше шаблонного коду.
- Можливість використовувати хуки (hooks), що спрощує управління станом, ефектами та контекстом.
- Зручність тестування ізоляції компонента, оскільки функціональні компоненти є чистими функціями.

Загалом, у React останніми роками активно просувається використання функціональних компонентів, особливо з введенням хуків, оскільки вони дозволяють писати більш зрозумілий та компактний код. Однак класові компоненти все ще залишаються використовуваними і можуть бути корисними в деяких сценаріях, особливо для старих проектів або випадків, коли необхідно використовувати функціонал, недоступний у функціональних компонентах.

Домашнє завдання

1) Ініціалізувати проект за допомогою CRA
2) Видалити зайві файли
3) Створити власні класові і функціональні компоненти
4) Додати їх до ротового файлу

Корисні посилання

Документація реакт легасі

Документація реакт нова

Create-react-app

Vite

JSX

Відмінності в атрибутах

Рендерінг елементів

React Chrome Developer Tools