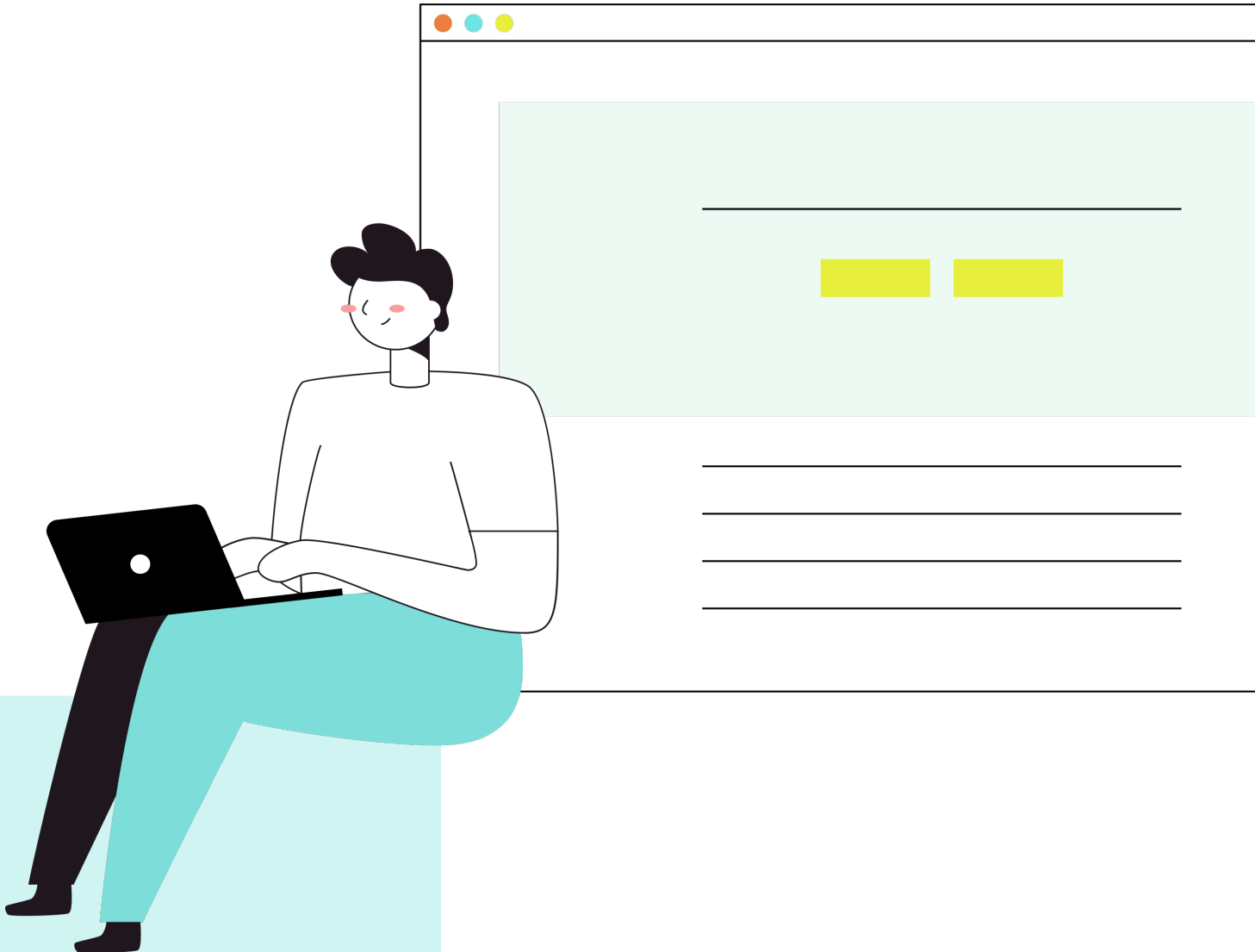
Inovatica

React Hooks

Krystian Bendinger
Filip Wajgner



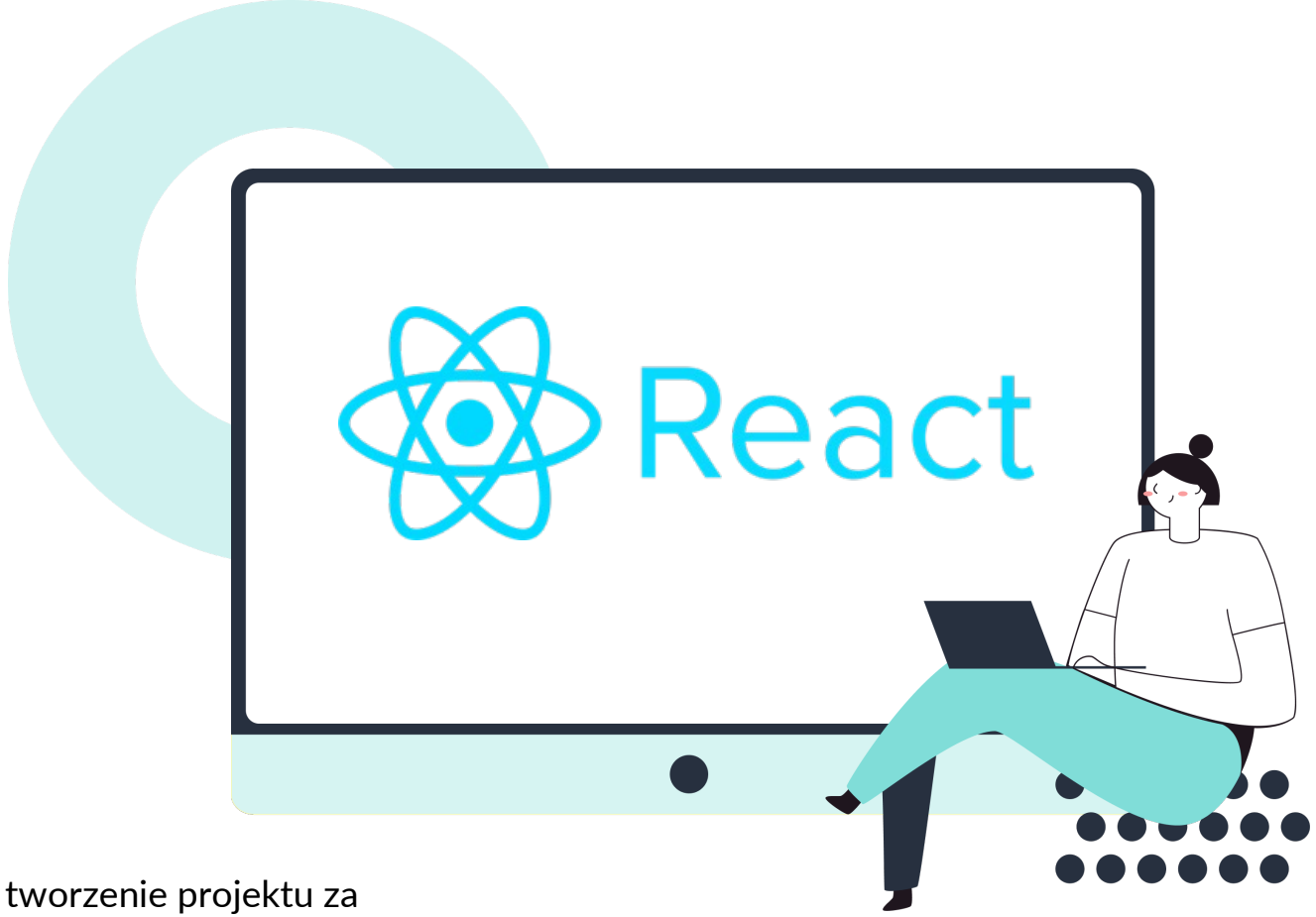
Spis treści

Inovatica

01.

Kilka słów o Reactcie

01 Kilka słów o Reactcie



Stworzony przez
Facebooka w 2013 roku

Biblioteka do tworzenia
interaktywnego UI

Oparty o komponenty,
za pomocą których
budowana jest aplikacja

Proste tworzenie projektu za
pomocą Create React App

JavaScript oraz TypeScript

Aktualnie najbardziej
popularna biblioteka na
frontendzie

Olbrzymia ilość bibliotek,
które można wykorzystać
podczas developmentu

02.

Czym są React Hooks

React Hooks – czym są

1. Funkcje
2. Pozwalają oddzielić logikę od widoku komponentu
3. Pozwalają korzystać z funkcjonalności Reacta bez użycia klas
4. Dostarczane przez Reacta, zewnętrzne biblioteki oraz własne

React Hooks – co przed nimi

Wprowadzone w wersji 16.8 – Luty 2019

Logika musiała znajdować się w komponencie klasowym

Wzorce tworzone przez społeczność – Higher Order Components

React Hooks - motywacja

- Współdzielenie logiki związanego ze stanem aplikacji było trudne
- Złożone komponenty z czasem stawały się ciężkie do zrozumienia
- Mniejszy próg wejścia w przypadku hooków w porównaniu do komponentów klasowych
- Zapobieganie nadmiernemu renderowaniu komponentów przez wydzielenie logiki
- Krótszy kod

React Hooks – co warto wiedzieć przed

Hooki są opcjonalne

Nie ma obowiązku pisać całego kodu opartego o hooki lub przepisywać go - można stopniowo je wprowadzać lub pozostać przy podejściu klasowym

Kompatybilność wsteczna

Hooki nie zawierają żadnych zmian, które zepsują istniejący kod po przejściu na nie

Klasy w Reactcie pozostaną

Na ten moment nie ma planów, aby klasy zniknęły z ekosystemu Reacta co jest potwierdzone dokumentacją

03.

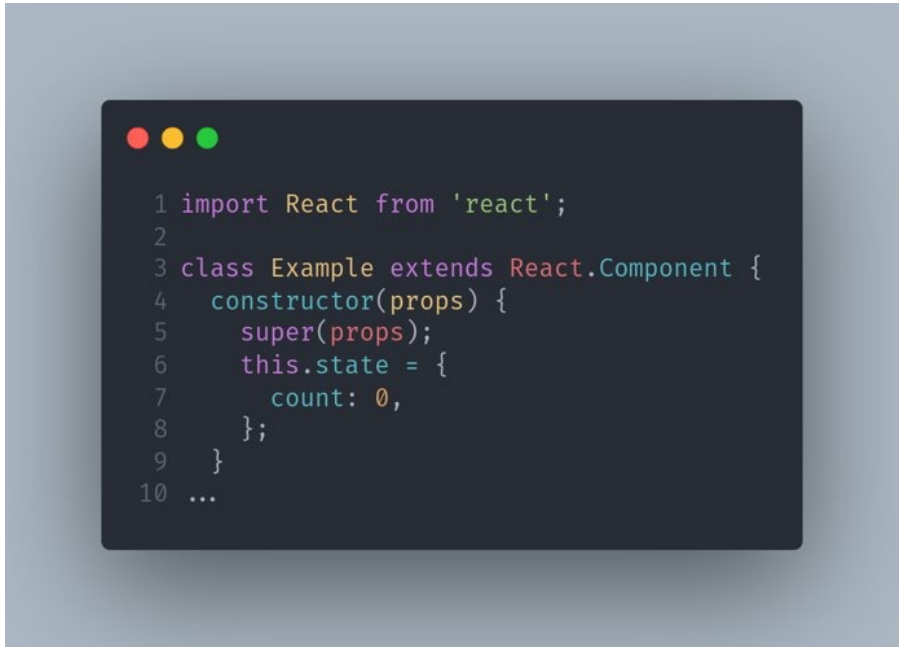
Podstawowe hooki

```
1 const [count, setCount] = useState(0);
```

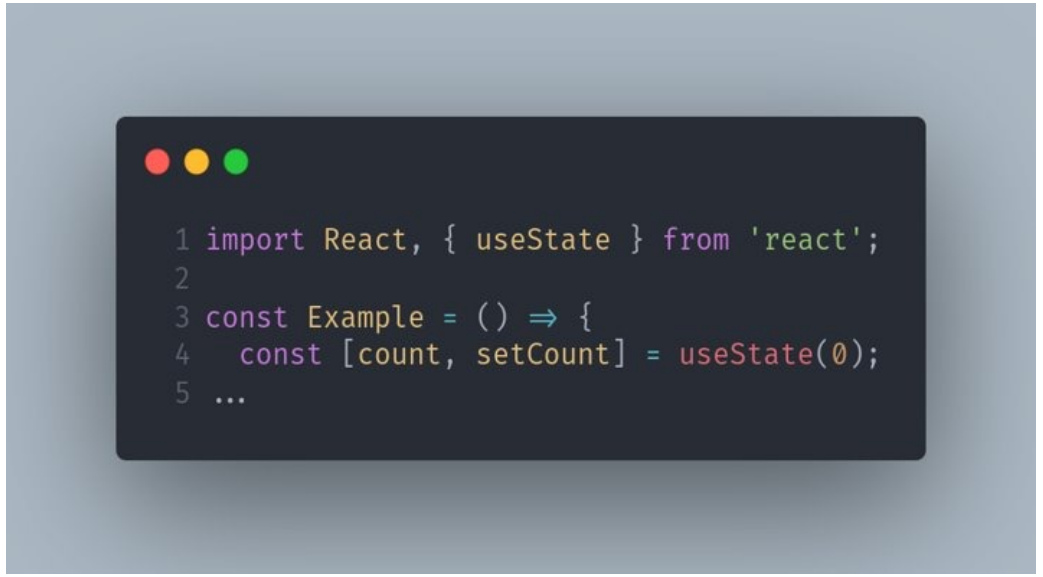
useState

Pozwala na wykorzystanie zmiennej stanu w komponencie funkcyjnym

Porównanie komponentu klasowego oraz funkcyjnego



```
1 import React from 'react';
2
3 class Example extends React.Component {
4   constructor(props) {
5     super(props);
6     this.state = {
7       count: 0,
8     };
9   }
10 ...
```



```
1 import React, { useState } from 'react';
2
3 const Example = () => {
4   const [count, setCount] = useState(0);
5 ...
```

Porównanie komponentu klasowego oraz funkcyjnego

```
1 <p>Clicked: {this.state.count}</p>
```

```
1 <p>Clicked: {count}</p>
```

Porównanie komponentu klasowego oraz funkcyjnego

```
1 <button onClick={() ⇒ this.setState({ count: this.state.count + 1 })}>Click me</button>
```

```
1 <button onClick={() ⇒ setCount(count + 1)}>Click me</button>
```

Porównanie komponentu klasowego oraz funkcyjnego

```jsx
import React from 'react';

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

  render() {
    return (
      <React.Fragment>
        <p>Clicked: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>Click me</button>
      </React.Fragment>
    );
  }
}

export default Example;
```

```jsx
import React, { useState } from 'react';

const Example = () => {
  const [count, setCount] = useState(0);

  return (
    <React.Fragment>
      <p>Clicked: {count}</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </React.Fragment>
  );
};

export default Example;
```

Zwracana wartość

Standardowe podejście to użycie mechanizmu dekompozycji tablicy

```
1 const [count, setCount] = useState(0);
```

```
1 let countStateVariable = useState(0);
2 let count = countStateVariable[0];
3 let setCount = countStateVariable[1];
```

Wartość początkowa

Możliwość podania dowolnego
typu bez żadnych ograniczeń

```
const [count, setCount] = useState(0);
const [book, setBook] = useState('exampleBook');
const [loading, setLoading] = useState(false);
const [selectedValue, setSelectedValue] = useState(null);
const [bookList, setBookList] = useState(['exampleBook', 'exampleLongerBook']);
const [todoList, setTodoList] = useState([{ title: 'exampleTodo', done: false }]);
```

```
useEffect(() => {
  document.title = `Clicked ${count} times`;
}, [count]);
```

useEffect

Pozwala na uruchomienie efektów ubocznych w komponentach funkcyjnych, reakcję na zmianę wartości oraz czyszczenie po efekcie

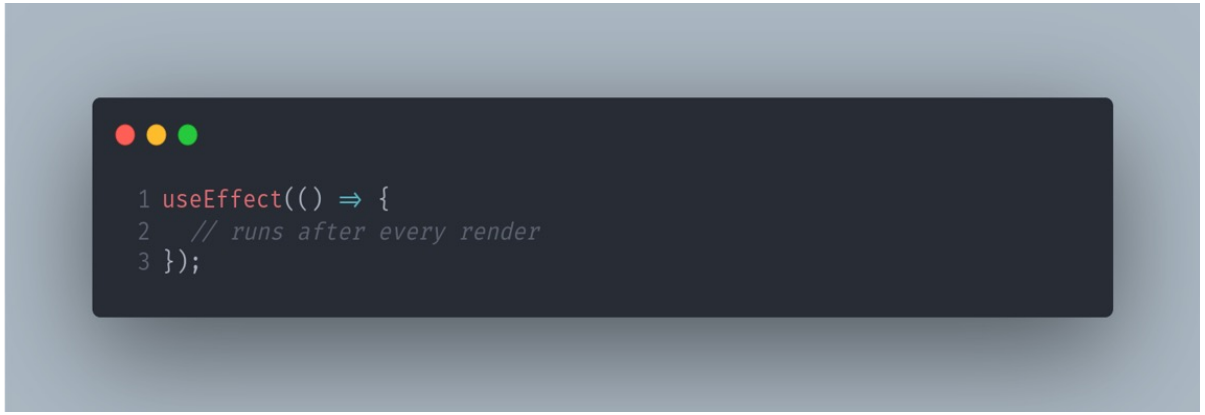
Porównanie komponentu klasowego oraz funkcyjnego

```jsx
import React from 'react';

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

  componentDidMount() {
    document.title = `Clicked ${this.state.count} times`;
  }
  componentDidUpdate() {
    document.title = `Clicked ${this.state.count} times`;
  }

  render() {
    return (
      <React.Fragment>
        <p>Clicked: {this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
      </React.Fragment>
    );
  }
}

export default Example;
```

```jsx
import React, { useEffect, useState } from 'react';

const Example = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Clicked ${count} times`;
  }, [count]);

  return (
    <React.Fragment>
      <p>Clicked: {count}</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </React.Fragment>
  );
};

export default Example;
```

Uruchamianie useEffect

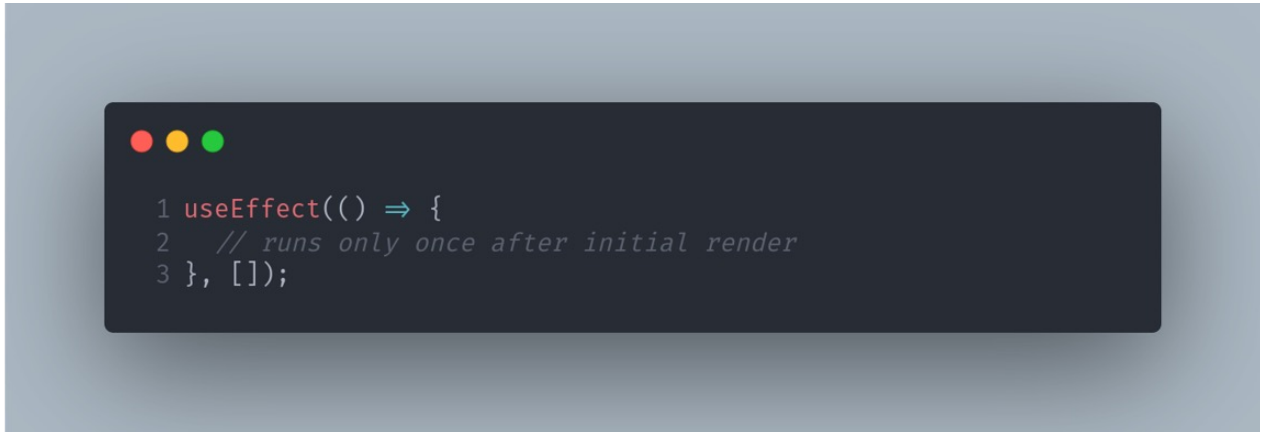
Brak tablicy zależności – efekt będzie uruchamiany po każdym renderze



```
useEffect(() => {
  // runs after every render
});
```

Uruchamianie useEffect

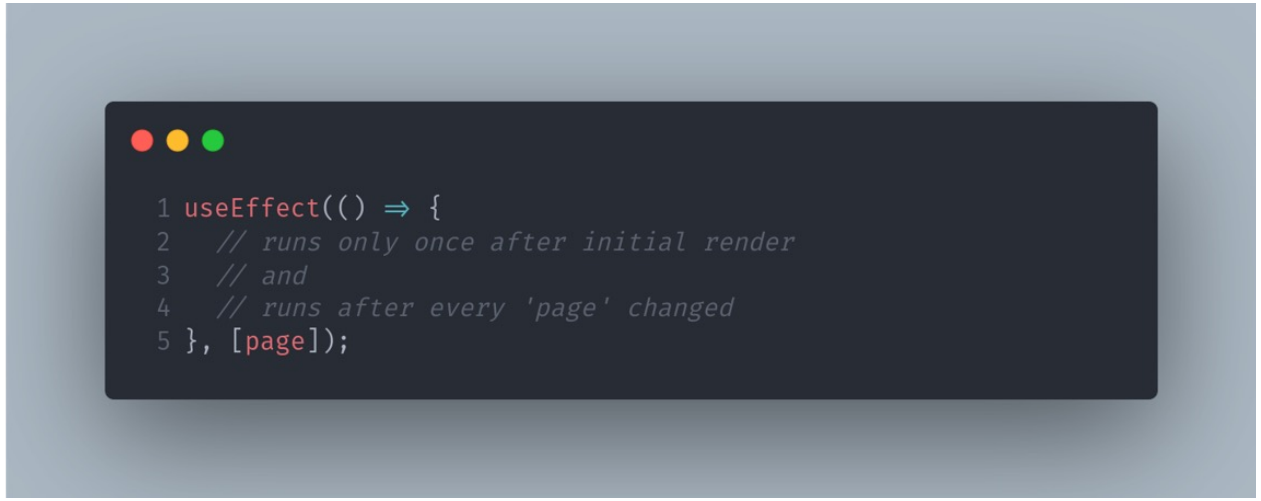
Pusta tablica zależności – efekt uruchamiany tylko raz (inicjalnie)



```
useEffect(() => {
  // runs only once after initial render
}, []);
```

Uruchamianie useEffect

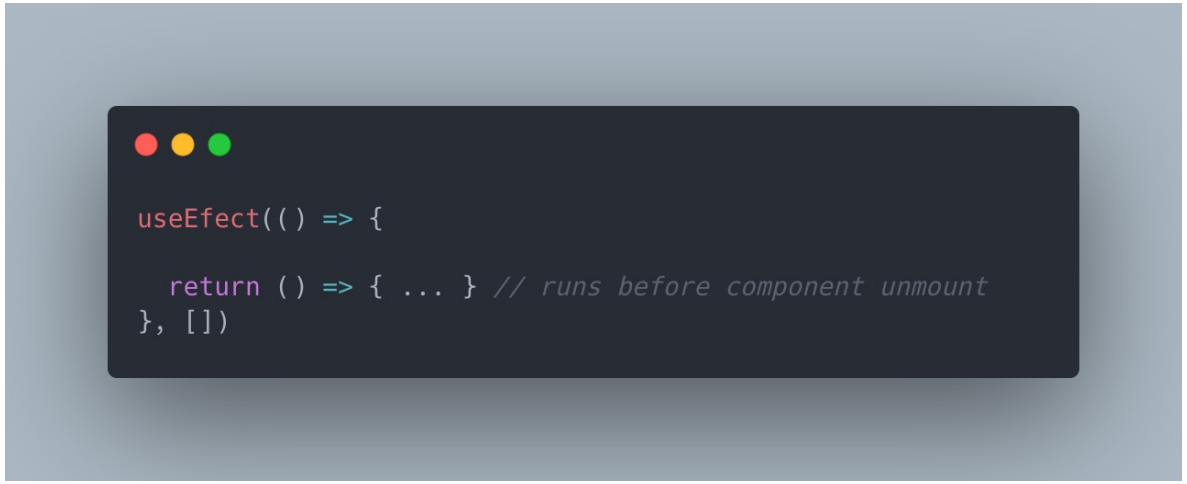
Tablica zależności z elementami – efekt uruchamiany kiedy elementy w tablicy się zmienią



```
useEffect(() ⇒ {
  // runs only once after initial render
  // and
  // runs after every 'page' changed
}, [page]);
```

Czyszczenie w useEffect

Czyszczenie po efekcie – wykonuje się w momencie odmontowania komponentu



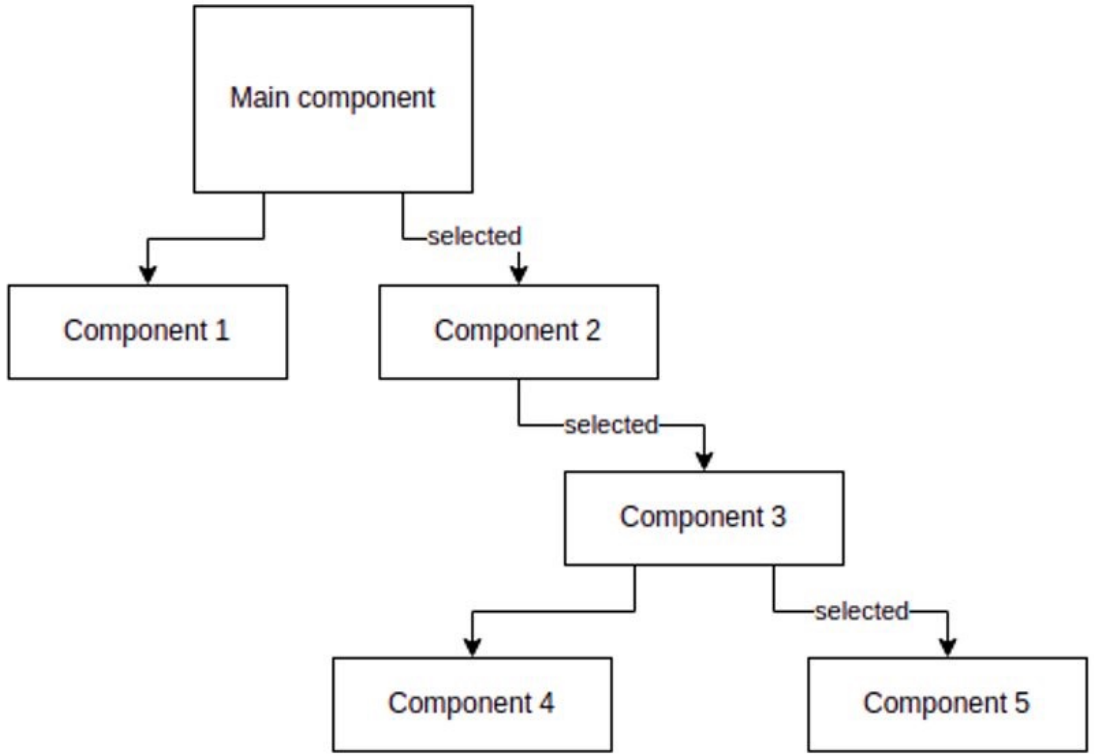
```
useEfect(() => {

  return () => { ... } // runs before component unmount
}, [])
```

```
const { selected } = useContext(ExampleContext);
```

useContext

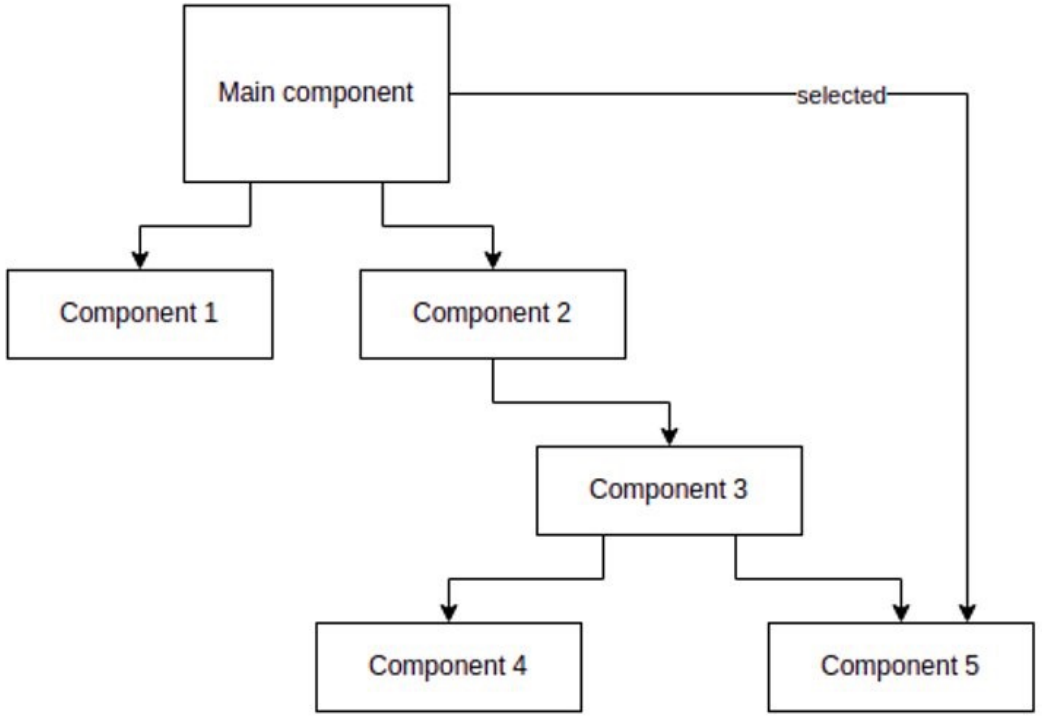
Pozwala na wykorzystanie kontekstu w komponencie funkcyjnym

Przekazywanie argumentów w dół



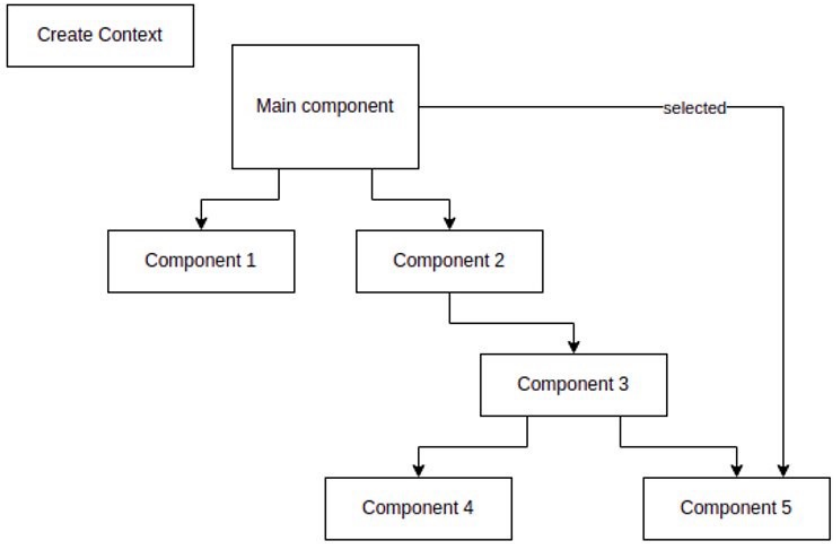
Context API

Kontekst zapewnia sposób przekazywania argumentów w drzewie bez konieczności ich podawania na każdym poziomie



```
1 const ExampleContext = React.createContext();
2 ExampleContext.displayName = 'ExampleContext';
```

Działanie Context API




```
const MainComponent = () => {
  const [selected, setSelected] = useState('example');

  return (
    <ExampleContext.Provider value={selected}>
      <Component2 />
    </ExampleContext.Provider>
  );
};
```

```
1 const Component5 = () => {
2   const { selected } = useContext(ExampleContext);
3
4   return <React.Fragment>{selected}</React.Fragment>;
5 };
```

04.

Zaawansowane hooki

useMemo

Służy do zmemoizowania wartości, używany, aby utrzymać stabilną referencję na obiekt oraz unikać powtarzania ciężkich obliczeń

```
const memoizedValue = useMemo(
  () => ({
    title: newTitle,
    description: newDescription,
  }),
  [newTitle, newDescription],
);
```

useCallback

Służy do zmemoizowania całego callbacku, używany, aby utrzymać stabilną referencję na funkcję

```
1  const memoizedCallback = useCallback(() => {
2    add(a, b);
3  }, [a, b]);
```

```
1 const inputRef = useRef(null);
```

useRef

Poza ekosystemem Reacta (obszarem śledzenia zmian), przetrzymuje referencje na węzeł w DOM lub na mutowalną wartość, która jest zapamiętywana pomiędzy renderami, ale jej zmiana nie powoduje przerendereowania komponentu

Przykład użycia hooka useRef

```jsx
import React, { useRef, useEffect } from 'react';

const Example = () => {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return (
    <React.Fragment>
      <input ref={inputRef} />
    </React.Fragment>
  );
};

export default Example;
```

05.

Hooki są wszędzie

Przykłady bibliotek posiadających hooki

i18next

React Hook Form

React Query

React Leaflet

MATERIAL-UI

Ant Design

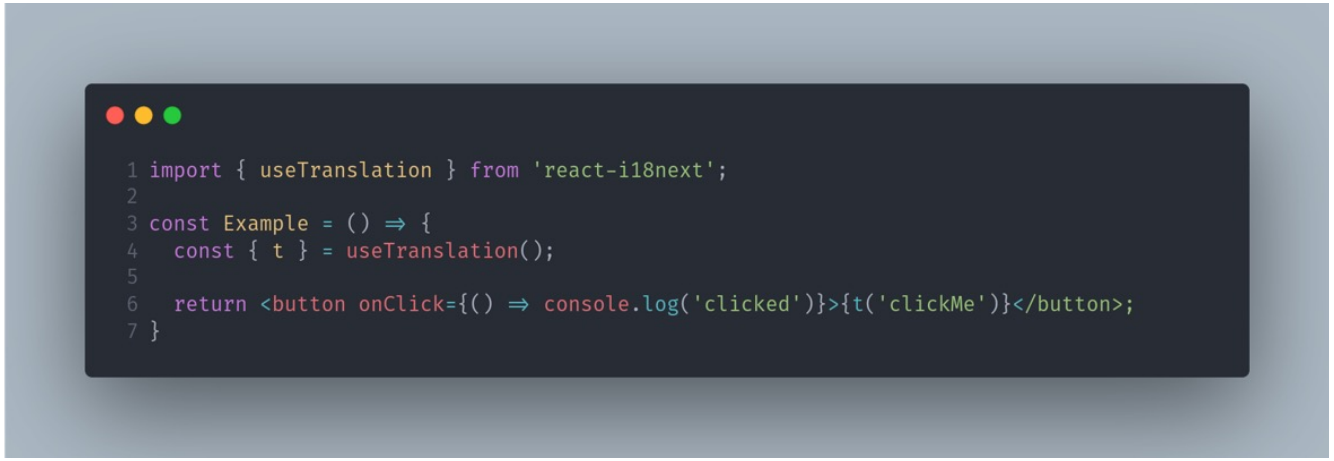
Formik

Redux

Ant Design

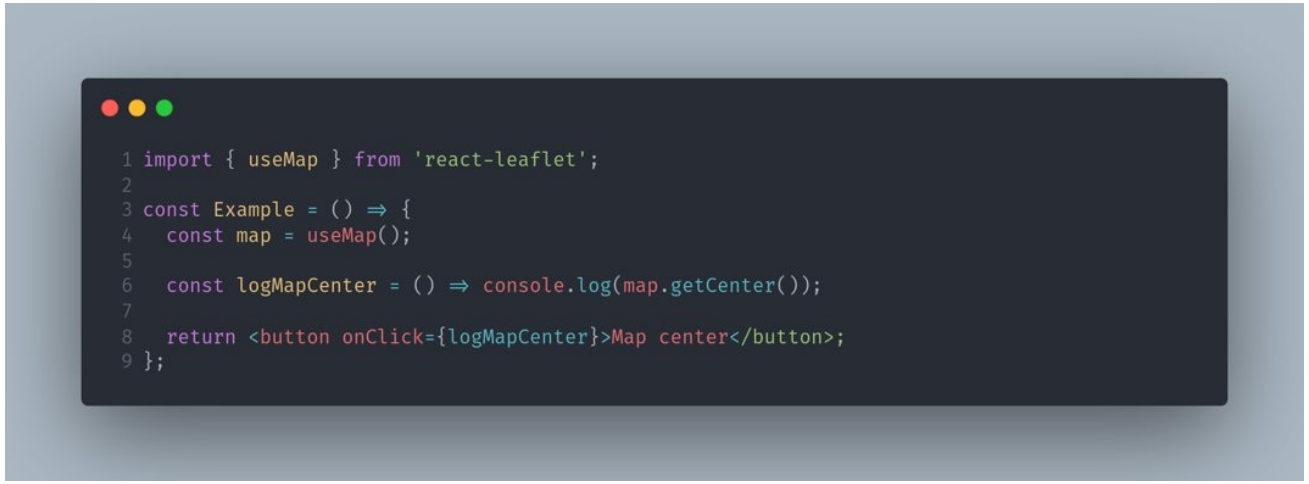
```jsx
import { Form } from 'antd';

const Example = () => {
  const [form] = Form.useForm();

  const handleChangeExampleField = value => form.setFieldsValue({ exampleField: value });

  return (
    <Form name='exampleForm' form={form}>
      ...
    </Form>
  );
};
```

i18next



```
import { useTranslation } from 'react-i18next';

const Example = () => {
  const { t } = useTranslation();

  return <button onClick={() => console.log('clicked')}>{t('clickMe')}</button>;
}
```

React Lefleat



```import { useMap } from 'react-leaflet';

const Example = () => {
  const map = useMap();

  const logMapCenter = () => console.log(map.getCenter());

  return <button onClick={logMapCenter}>Map center</button>;
};```
```

06.

Własne hooki

Inovatica

Hook sprawdzający token i zwracający zalogowanego użytkownika

```
const useAuthData = () => {
  const user = useSelector(state => getUser(state));
  const loading = useSelector(state => getLoadingStatus(state));
  const dispatch = useDispatch();

  useEffect(() => {
    dispatch(authActions.checkToken());
  }, [dispatch]);

  return { user, loading };
};
```

Hook odpowiedzialny za ustawienia paginacji w tabeli

```tsx
export const useTablePaginationConfig = ({ page, total, setPage }) => {
  const { t } = useTranslation();

  const paginationConfig = useMemo(
    () => ({
      current: page,
      pageSize: defaultPageSize,
      total: total,
      showTotal: (total, range) => t('tablePaginationRange', { rangeStart: range[0],
            rangeEnd: range[0], total: total }),
      hideOnSinglePage: true,
      onChange: setPage,
    }),
    [page, total, setPage, t],
  );

  return paginationConfig;
};
```

Dedykowany hook pod konkretny context

```
const useExample = () => {
  const ctx = useContext(ExampleContext);

  if (!ctx) {
    throw new Error('Component beyond ExampleContext');
  }

  return ctx;
};
```

07.

Wtyczka do ESLinta

eslint-plugin-react-hooks

Podczas developmentu wtyczka sprawdza poprawność kodu na podstawie zasad opisanych w Rules of Hooks

Reguły hooków

1. Hooki mogą być wywoływane tylko z komponentów funkcyjnych

```
const [data, setData] = useState(0); // ✗

const Example = () => {
  const [data, setData] = useState(0); // ✓
  ...
}
```

2. Hooki mogą być wywoływane tylko z głównego poziomu

```
const Example = () => {
  const [data, setData] = useState(0); // ✓

  if (exampleCondition) {
    const [otherData, setOtherData] = useState(0); // ✗
  }
  ...
}
```

08.

Podsumowanie

08 Podsumowanie

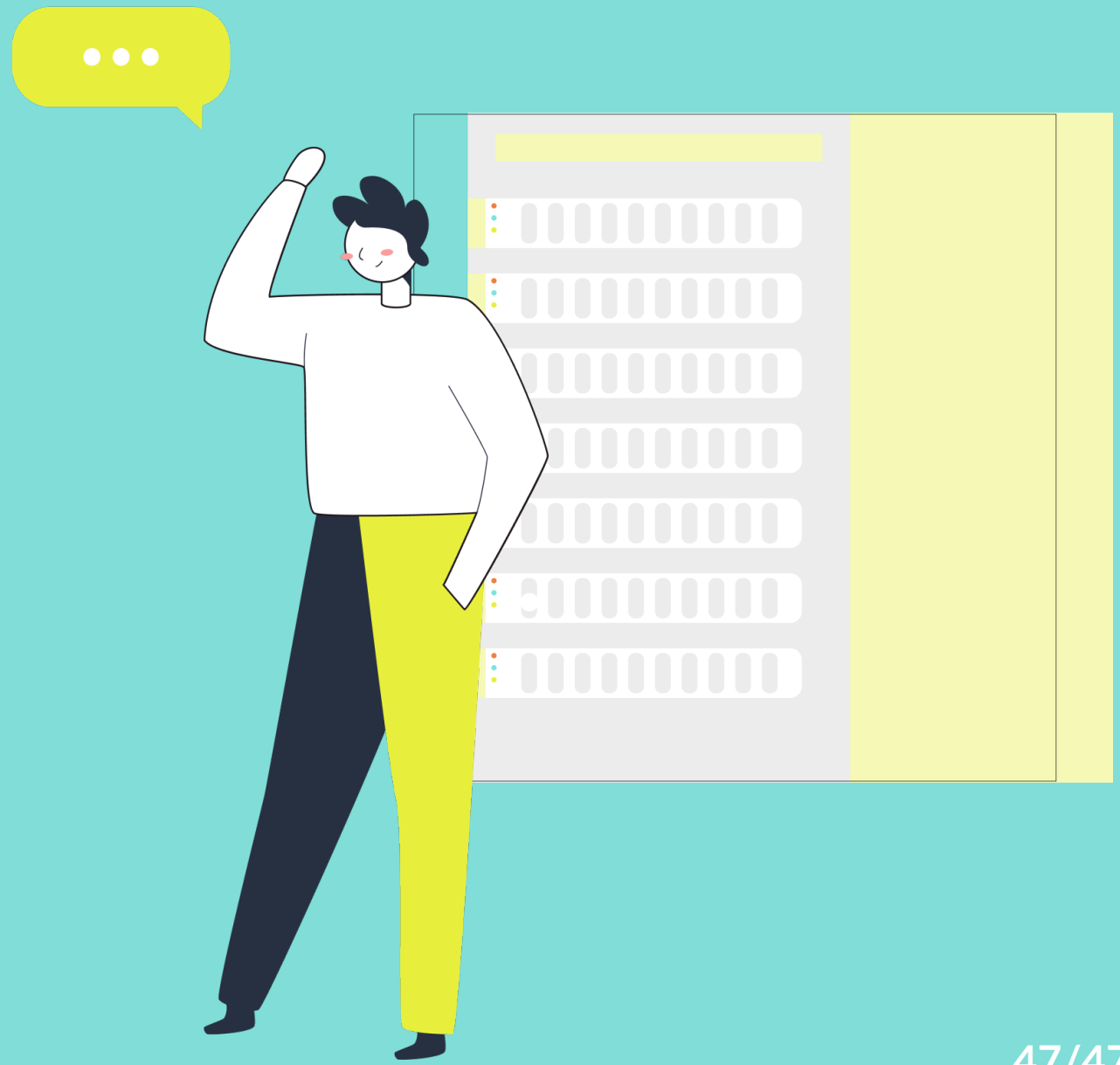
1. Funkcje oddzielające logikę komponentu od widoku
2. Prostsze do nauczenia niż poprzednie wzorce
3. Kod pisany za pomocą hooków może być mniej skomplikowany i krótszy
4. Większość bibliotek zewnętrznych wspiera ich używanie
5. Warto używać wtyczki do ESLinta
6. Wsparcie dla komponentów klasowych nie zniknie

Przydatne linki

- https://reactjs.org/docs/hooks-intro.html
- https://reactjs.org/docs/hooks-rules.html
- https://reactjs.org/docs/hooks-reference.html
- https://www.30secondsofcode.org/react/t/hooks/p/1
- https://ahooks.js.org
- https://nikgraf.github.io/react-hooks

Dziękujemy za uwagę!

Czy są jakieś pytania?



Inovatica