

WIKTOR JURCZYSZYN

REACT

AGENDA BLOKU

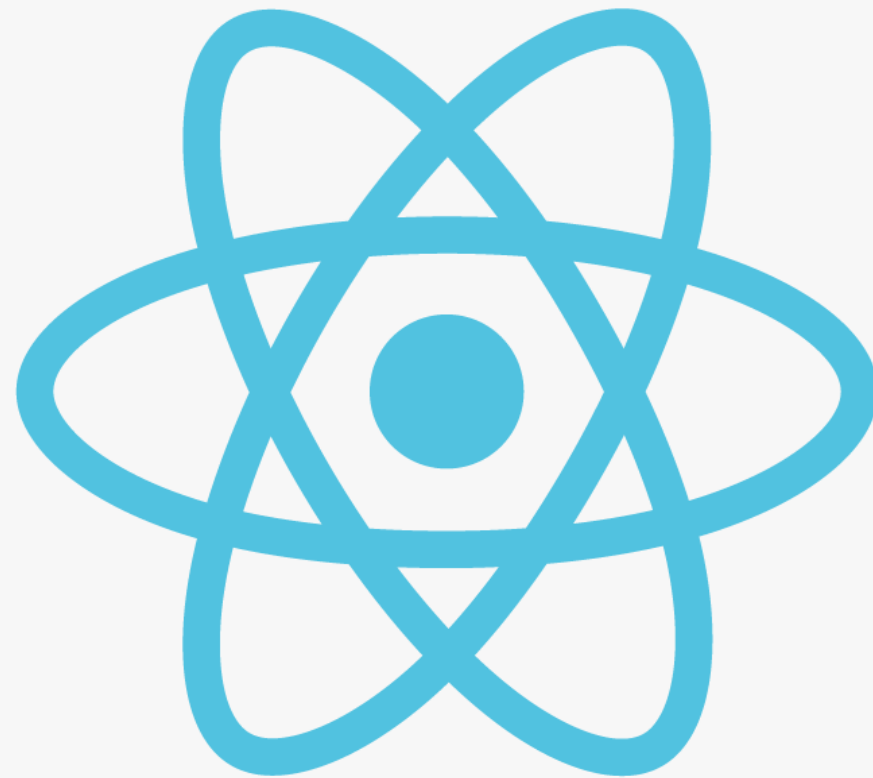
- ▶ React
- ▶ Redux
- ▶ Projekt

NARZĘDZIA

- ▶ IDE lub edytor tekstowy (Visual Code)
- ▶ Node (upewnijmy się, że mamy zainstalowany!)
- ▶ Yarn lub npm
- ▶ Przeglądarka (Chrome)
- ▶ Terminal (command line)

CZEGO SIĘ NAUCZĘ?

- ▶ Wiem co to React i Redux
- ▶ Wiem po co tego używam
- ▶ Potrafię zbudować projekt od zera
- ▶ Potrafię skonfigurować routing
- ▶ Potrafię swobodnie pracować z React i Redux



React

REACT

A JavaScript library for building user interfaces

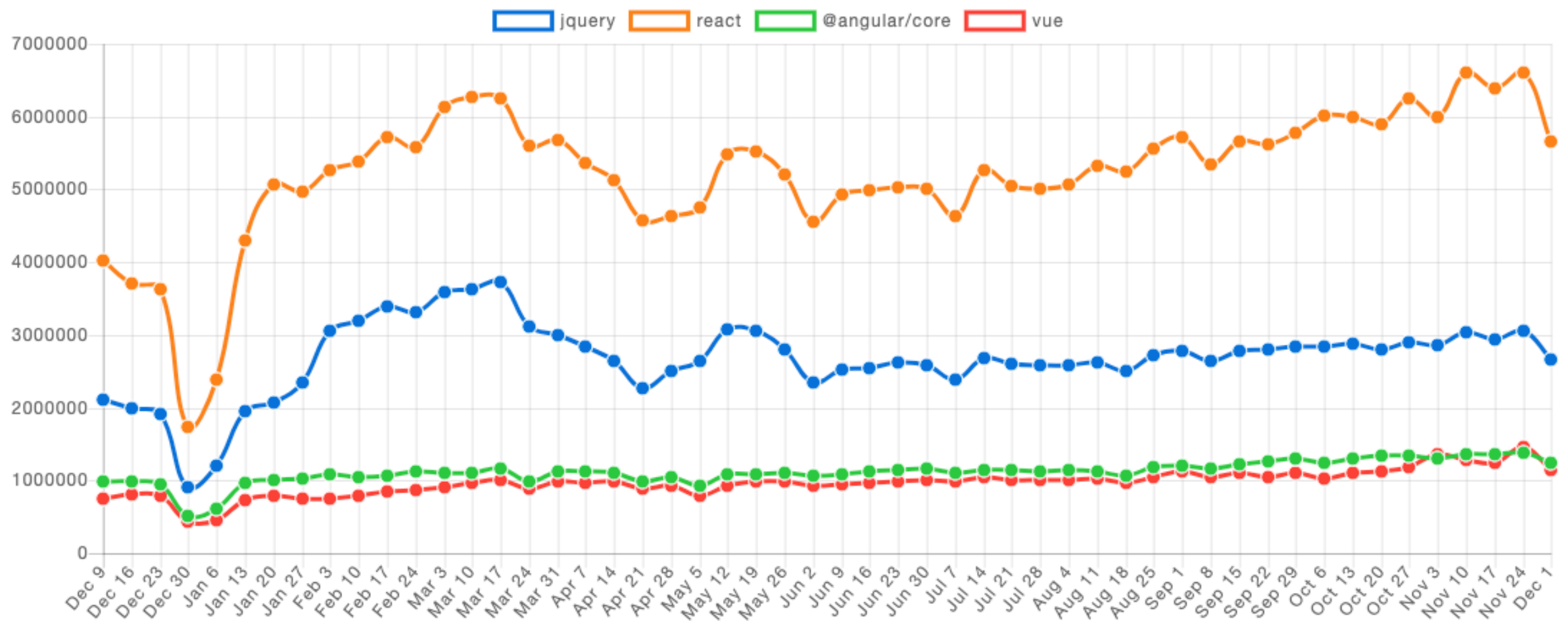
REACT

A JavaScript library for building user interfaces

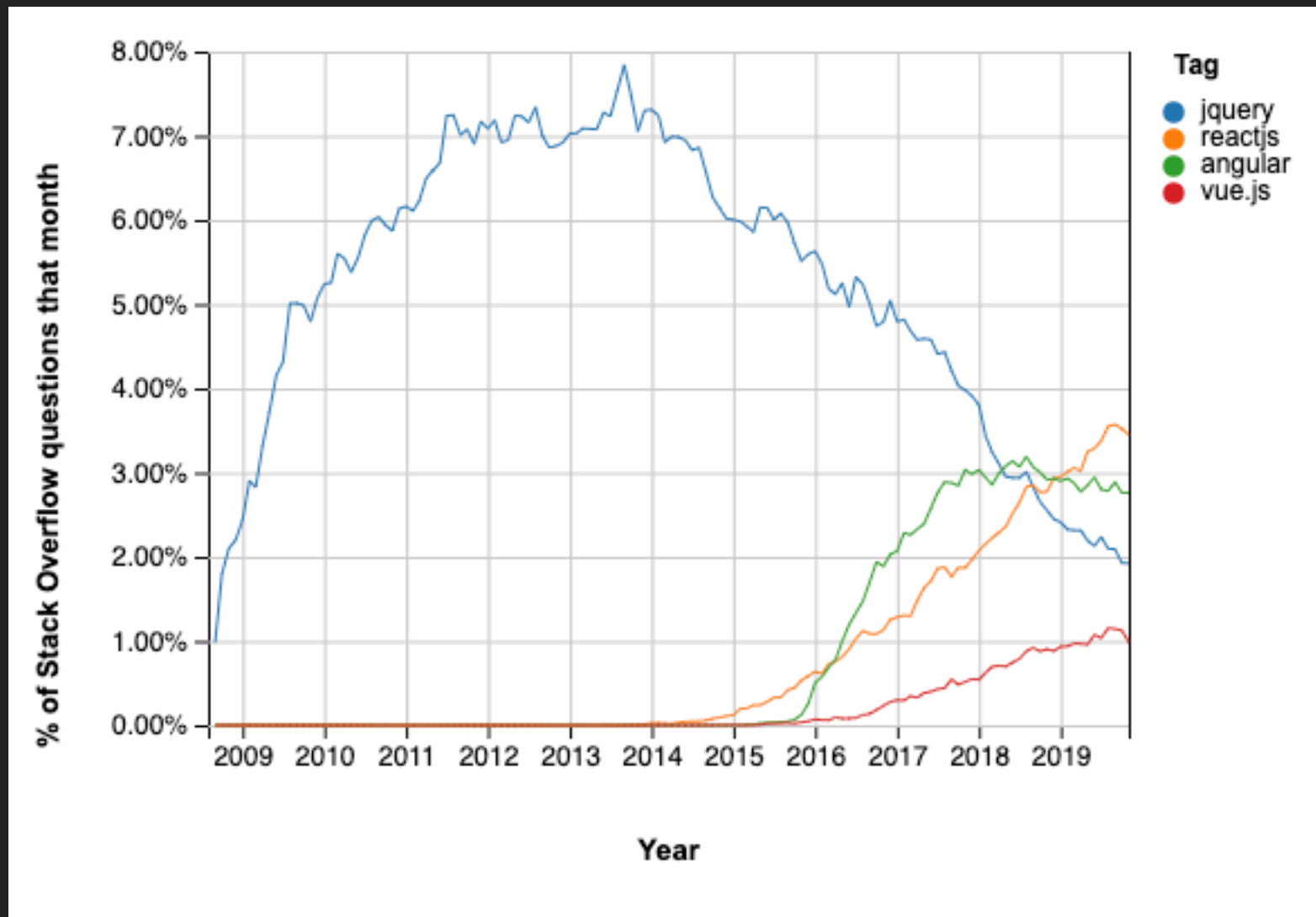
- ▶ Pierwsze wydanie 2013
- ▶ Wytworzone i wspierane przez Facebooka
- ▶ Open source
- ▶ Jedna z 3 wiodących technologii frontendowych
- ▶ React Native

POPULARNOŚĆ

Downloads in past 1 Year ▾

<https://www.npmtrends.com/jquery-vs-react-vs-@angular/core-vs-vue>

POPULARNOŚĆ



<https://insights.stackoverflow.com/trends?tags=jquery%2Creactjs%2Cangular%2Cvue.js>

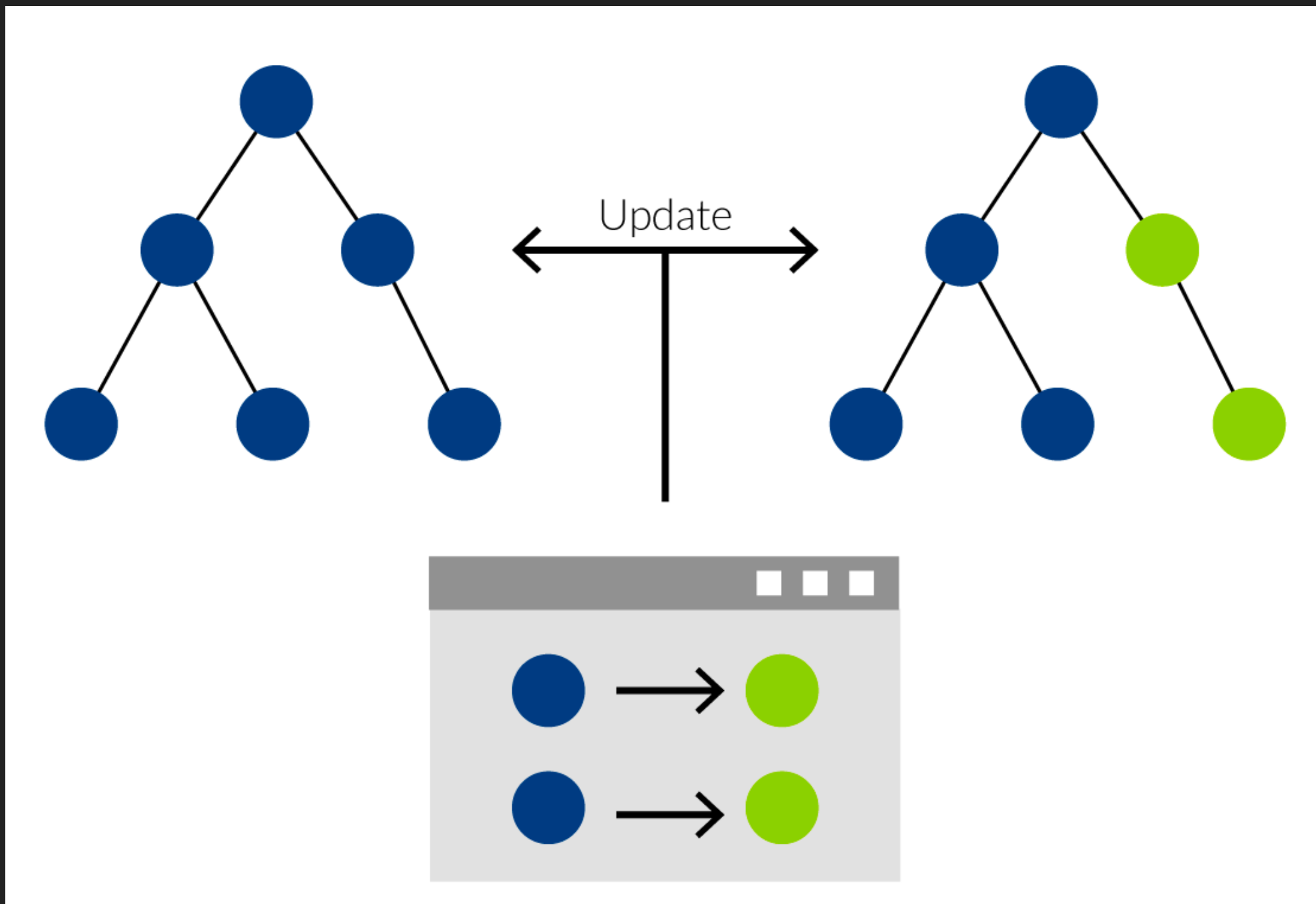
DLACZEGO UŻYWAĆ REACTA

- ▶ Ułatwia pracę
- ▶ Szybki (wydajność) - wirtualny DOM
- ▶ Pozwala pisać reużywalny kod
- ▶ Pozwala na zbudowanie architektury komponentowej
- ▶ Jest popularny (jest to ważne)

WIRTUALNY DOM

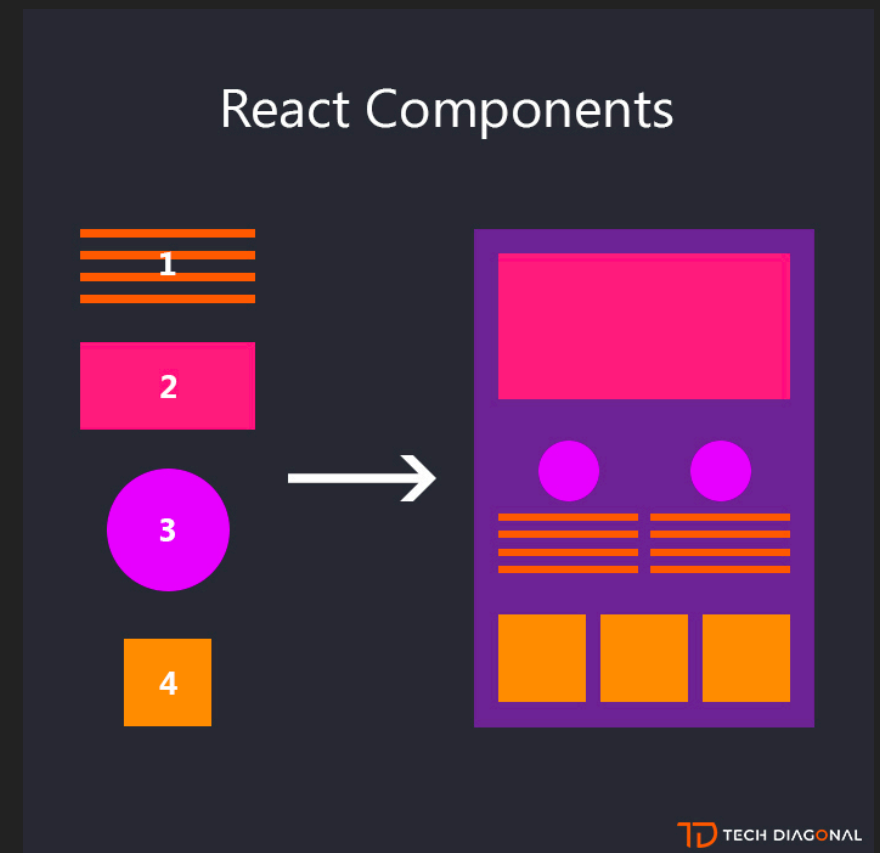
- ▶ Kopia prawdziwego DOM
- ▶ React "pracuje" na wirtualnym DOM
- ▶ React update'uje prawdziwy DOM na podstawie znalezionych różnic między prawdziwym a wirtualnym DOM
- ▶ Zyskujemy znaczną poprawę wydajności
- ▶ Pisząc w React trzeba mieć na uwadze istnienie tego mechanizmu

WIRTUALNY DOM



KOMPONENTY

- ▶ Względy estetyczne
- ▶ Organizacja
- ▶ Główny nurt w świecie frontendu
- ▶ Separation of concerns (rozdział obowiązków)
- ▶ Dumb i smart komponenty



REACT A HTML

- ▶ Mam komponenty, ale co dalej?
- ▶ Przeglądarki a React
 - ▶ Transpilacja (babel)
 - ▶ Wstrzyknięcie do dokumentu HTML

“REACT” A “REACT-DOM”

- ▶ Paczka “react” to operacje związane z samym procesem przetwarzania danych przez React.
- ▶ Paczka “react-dom” to operacje związane z wrzuceniem Reacta do HTML (“render”).
- ▶ <https://stackblitz.com/edit/pure-react-wfjk8vr-yxlwkt>

JSX

- ▶ Rozszerzenie składni JavaScript
- ▶ Mieszanka JavaScript i języka template
- ▶ Głównie wykorzystywany w React
- ▶ Ma swoich zwolenników i przeciwników

```
const element = <h1>Witaj, świecie!</h1>;
```

- ▶ [Playground](#)

JAK DEBUGOWAĆ W REACT

- ▶ Wszystkie metody z JS są jak najbardziej dopuszczalne.
- ▶ React devtools (Chrome, Firefox, Edge)
- ▶ Ćwiczenie 1:
 - ▶ Zainstaluj *React Developer Tools*
 - ▶ Znajdź stronę z *Reactem* i zbadaj ją

NODE.JS I NPM - INSTALACJA

- ▶ Sprawdzenie wersji Node.js: w *Command Line* (Windows) lub w *Terminal* (mac/linux)
 - ▶ `node -v` (najlepiej > v8.0)
 - ▶ `npm -v`
- ▶ Jeżeli nie zainstalowane -> <https://nodejs.org/en/download/>
- ▶ Przechodzimy do folderu w którym chcemy utworzyć projekt i wpisujemy komendę: `npx create-react-app my-test`

CREATE REACT APP

- ▶ Stworzenie nowego projektu
- ▶ Webpack
- ▶ Co tam jest? Dostępne komendy
- ▶ "eject" w CRA
- ▶ Instalowanie paczek
 - ▶ przykład z <https://www.npmjs.com/package/react-awesome-button>

REACT – KOMPONENTY KLASOWE

WYPRÓBUJMY REACTA!

- ▶ Na początek wykorzystamy StackBlitz w przeglądarce
- ▶ Idealny do testów
- ▶ Warto założyć konto aby mieć możliwość "forkowania"
- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc?file=public%2Findex.html>

JAK ZBUDOWAĆ KOMPONENT W REACT?

- ▶ `<div>Welcome</div>`
- ▶ `<Title>Welcome</Title>`
- ▶ `<Name />`

ĆWICZENIE

- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc>
- ▶ Zadania:
 - ▶ 1. Zastąp element `<h1>` własnym komponentem *HelloWorldComponent*
 - ▶ 2. Zastąp element `<p>` własnym komponentem *MyTextComponent*
 - ▶ 3. *Zastąp wszystkie `<h2>` jednym komponentem *GreetingsComponent*

ĆWICZENIE - WNIOSKI

- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc>
- ▶ Warto zapamiętać:
 - ▶ Komponent może zwrócić tylko jeden element (albo wiele elementów "opakowanych" w *parenta*) - można wykorzystać pusty znacznik
 - ▶ Nazwy komponentów muszą zaczynać się wielką literą

JS W KOMPONENTACH

- ▶ `<div>{2 + 2}</div>`
- ▶ Kod w JSX zapisujemy w nawiasach: `{...}`

PROPS

- ▶ Parametry przekazywane do komponentu z zewnątrz
- ▶ Raczej unikać przekazywania obiektu
- ▶ Nowy prop powoduje re-render!

```
class AnimalComponent extends React.Component {  
  render() {  
    return <p>This is {this.props.animalName}</p>  
  }  
}  
  
<AnimalComponent animalName="dog" />
```

```
class AnimalComponent extends React.Component {  
  render() {  
    return (  
      <p>  
        This is {this.props.name} and it's {this.props.age}  
      </p>  
    )  
  }  
}  
  
<AnimalComponent name="dog" age={10} />
```

ĆWICZENIE

- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc>
- ▶ Zadania:
 - ▶ 1. Stwórz komponent *GreetNameComponent*:
 - ▶ a) Zastąp każdy z `<h3>` własnym komponentem *GreetNameComponent*
 - ▶ b) Za pomocą *propsów* przekazuj imię do komponentu *GreetNameComponent*

EVENT HANDLERY - ONCLICK (1)

```
1  class Foo extends Component {  
2    handleClick = () => {  
3      console.log('Click happened');  
4    }  
5  
6    render() {  
7      return <button onClick={this.handleClick}>Click Me</button>  
8    }  
9  }
```

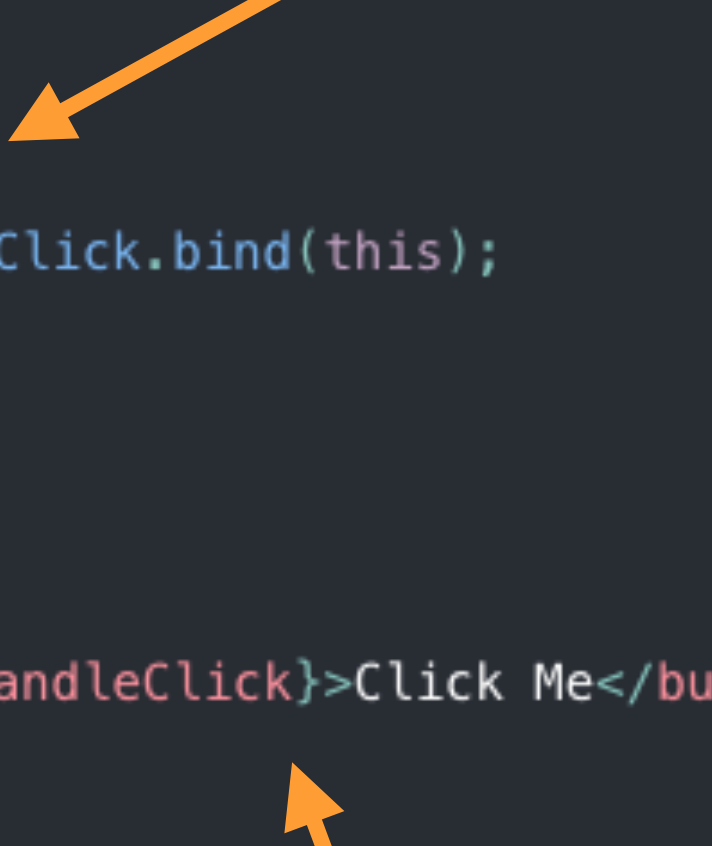
EVENT HANDLERY - ONCLICK (2)

```
class Foo extends Component {  
  handleClick() {  
    console.log('Click happened');  
  }  
  render() {  
    return <button onClick={this.handleClick.bind(this)}>Click Me</button>;  
  }  
}
```


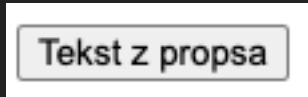


EVENT HANDLERY - ONCLICK (3)

```
class Foo extends Component {  
  constructor(props) {  
    super(props);  
    this.handleClick = this.handleClick.bind(this);  
  }  
  handleClick() {  
    console.log('Click happened');  
  }  
  render() {  
    return <button onClick={this.handleClick}>Click Me</button>;  
  }  
}
```



ĆWICZENIE

- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc>
- ▶ Zadania:
 - ▶ 1. Dodaj nowy komponent *MyButton* renderujący `<button>` z napisem "Click me!"

 - ▶ 2. Dodaj *event handler* tak, aby po kliknięciu *MyButton* pojawiał się alert "Brawo!"
 - ▶ 3. Zmień komponent *MyButton* tak, aby przekazywany był przez *props* do komponentu:
 - ▶ a) tekst wyświetlany na przycisku 
 - ▶ b) napis wyświetlany przez alert

ĆWICZENIE Z KOMPONENTAMI – MAP

- ▶ Zbuduj komponent wyświetlający markę auta
- ▶ Wyświetl auta z tablicy
- ▶ ["Audi", "Alfa Romeo", "BMW", "Opel", "Skoda", "Volvo"]

ĆWICZENIE Z KOMPONENTAMI – FILTER

- ▶ Zbuduj komponent wyświetlający markę auta
- ▶ Wyświetl auta z tablicy
- ▶ ["Audi", "Alfa Romeo", "BMW", "Opel", "Skoda", "Volvo"]
- ▶ Zmień tablice string na tablicę obiektów i dodaj pole "maker" określające producenta.
- ▶ Odfiltruj auta z grupy VW

KOMPONENT KLASOWE

KOMPONENT STATELESS

```
class HelloMessage extends React.Component {  
  render() {  
    return (  
      <div>  
        Hello {this.props.name}  
      </div>  
    );  
  }  
}
```

KOMPONENT STATEFUL

```
class Timer extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { seconds: 0 };  
  }  
  
  tick() {  
    this.setState(state => ({  
      seconds: state.seconds + 1  
    }));  
  }  
  
  componentDidMount() {  
    this.interval = setInterval(() => this.tick(), 1000);  
  }  
  
  componentWillUnmount() {  
    clearInterval(this.interval);  
  }  
  
  render() {  
    return (  
      <div>  
        Seconds: {this.state.seconds}  
      </div>  
    );  
  }  
}
```

STATE

- ▶ Daje możliwość "pamiętania" zmiennych
- ▶ Zmiany w stanie możemy wykonywać tylko przy pomocy funkcji z biblioteki React
- ▶ Musi być to obiekt
- ▶ Zmiana stanu powoduje re-render!

```
25 class ClickCounter extends React.Component {  
26     constructor(props) {  
27         super(props);  
28         this.state = { numberOfClicks: 0 };  
29     }  
30     render() {  
▶ 31         return <p>Number of clicks: {this.state.numberOfClicks}</p>;  
32     }  
33 }  
34
```

STATE

```
21  class MyNumberComponent extends React.Component {
22      constructor(props) {
23          super(props);
24          this.state = { myNumber: 997 };
25      }
26
27      myClick() {
28          this.setState({ myNumber: 100 });
29      }
30
31      render() {
32          console.log("render!");
33          return (
34              <div>
35                  <button onClick={this.myClick.bind(this)}>Minus 100!</button>
36                  <p>My current number: {this.state.myNumber}</p>
37              </div>
38          );
39      }
40  }
```

STATE

```
23  class MyNumberComponent extends React.Component {
24      constructor(props) {
25          super(props);
26          this.state = { myNumber: 997 };
27      }
28
29      myClick() {
30          this.setState({ myNumber: this.state.myNumber - 100 });
31      }
32
33      render() {
34          console.log("render!");
35          return (
36              <div>
37                  <button onClick={this.myClick.bind(this)}>Minus 100!</button>
38                  <p>My current number: {this.state.myNumber}</p>
39              </div>
40          );
41      }
42  }
```

STATE

- ▶ Tylko komponenty klasowe
- ▶ Obiekt
- ▶ `this.setState({ licznik: 997 });`
- ▶ `this.setState({ licznik: this.state.licznik + 1 })`
- ▶ `this.setState((state) => ({ licznik: state.licznik + 1 })`

STATE

▶ Przykład:

▶ <https://stackblitz.com/edit/react-fcvico>

ĆWICZENIE ZE STATE 1

- ▶ Dodajemy komponent
- ▶ Dodajemy stan z licznikiem
- ▶ Dodajemy przycisk i tekst wyświetlający nasz licznik
- ▶ Na kliknięcie zwiększamy licznik o 1
- ▶ Dodajemy przycisk resetujący licznik
- ▶ *Dodajemy dodatkowy input (oraz stan) na podanie "step" dla licznika

ĆWICZENIE ZE STATE 2

- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc>
- ▶ Zadania:
 - ▶ 1. Dodaj stan (*state*) o nazwie *myMoney* do komponentu *App* oraz daj mu początkową wartość numeryczną 100. Wyświetl wartość stanu *myMoney* w komponencie *App*.
 - ▶ 2. Dodaj dwa przyciski - jeden który będzie zmniejszał wartość stanu *myMoney* o 10, a drugi zwiększał o 10. *Nie pozwól zejść poniżej 0!*
 - ▶ 3. ***Wykonaj zadanie 2. ale tworząc nowy komponent *MoneyButton* do którego przekażemy dwa propsy: a) napis który ma być na nim wyświetlony; b) funkcję która wykona się po kliku***

RENDER

- ▶ Kluczowa metoda dla komponentu klasowego
- ▶ Wymagana!
- ▶ Musi zwracać jakiś element JSX
- ▶ w niej NIE WOLNO używać setState!

KOMUNIKACJA “W GÓRĘ”

- ▶ Jak przekazać dane z “child” do “parent”?

KOMUNIKACJA “W GÓRĘ”

- ▶ Jak przekazać dane z “child” do “parent”?
- ▶ “funkcja callback”
- ▶ React context
- ▶ Biblioteki state management (np. Redux)

ĆWICZENIE Z KOMUNIKACJĄ W GÓRĘ – PRZYKŁAD WSPÓLNIE

- ▶ Prosty kalkulator (tylko suma)
- ▶ Komponent z wizualizacją wyniku (CalculatorComponent)
- ▶ Komponent zawierający 2 inputy i button (SumComponent)
 - ▶ Button z kalkulacją sumy
 - ▶ Zwrot wyniku to CalculatorComponent (komunikacja "w górę")

ĆWICZENIE Z KOMPONENTAMI

- ▶ Zbuduj komponent wyświetlający markę auta
- ▶ Wyświetl auta z tablicy
- ▶ ["Audi", "Alfa Romeo", "BMW", "Opel", "Skoda", "Volvo"]
- ▶ W komponencie dodaj przycisk który usunie pojazd z wyświetlanej listy

LIFECYCLE METHODS

- ▶ **componentDidMount()** - utworzenie (np. calle do api, dodanie event listeners)
- ▶ **componentDidUpdate(prevProps)** - zmiany
- ▶ **componentWillUnmount()** - destrukcja (np. Usunięcie event listenera).
- ▶ **render()**

<https://stackblitz.com/edit/react-life-cycle-mczebn>

<https://stackblitz.com/edit/react-life-cycle-icsgky>

ĆWICZENIE

- ▶ Czysty projekt: <https://stackblitz.com/edit/react-do5syc>
- ▶ Zadania:
 - ▶ 1. Licznik (setInterval)
 - ▶ 2. Wyświetl aktualny licznik na ekranie
 - ▶ 3. Dodaj flagę która umożliwi zamknięcie (nie renderowanie) lub otwarcie (renderowania) licznika

API W REACT

- ▶ React nie dostarcza nic związanego z zwołaniami do API
- ▶ Można korzystać z przeglądarkowego *fetch*
- ▶ Użyjemy popularnej biblioteki *axios*

```
47   axios
48     .get("https://adres_do_odpytania.pl/query", {
49       |   params: { parametr1: "wartosc1", parametr2: "wartosc2" }
50     | })
51     .then(response => {
52       |   console.log("To otrzymałem z serwera: " + response)
53     | })
54     .catch(error => {
55       |   console.log("Wystąpił taki error: " + error);
56     | });
```

ZADANIE API

- ▶ Korzystając z POKE API wyświetl informacje o pokemonach
- ▶ <https://pokeapi.co/api/v2/pokemon?limit=100>
- ▶ Wyświetlcie 100 pierwszych pokemonów po kliknięciu *buttona*
- ▶ Podaj liczbę pokemonów w input
- ▶ Lifecycle methods - przy starcie aplikacji automatycznie ma się pobrać 10 pokemonów
- ▶ *Obsługa błędów



The RESTful Pokémon API

WARTO PAMIĘTAĆ!

- ▶ Zawsze importujemy React w każdym pliku
- ▶ Komponenty klasowe muszą implementować metodę render!
- ▶ W render NIE UŻYWAMY setState!
- ▶ W razie potrzeby .bind w konstruktorze!

REACT – KOMPONENTY FUNKCYJNE

KOMPONENTY FUNKCYJNE VS KLASOWE

KOMPONENT FUNKCYJNY

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

KOMPONENT KLASOWY

```
class Welcome extends React.Component {  
  render() {  
    return (  
      <h1> Hello, {this.props.name} </h1>  
    );  
  }  
}
```

KOMPONENTY FUNKCYJNE VS KLASOWE

KOMPONENT FUNKCYJNY

```
function AnimalComponent(props) {  
  
  const barkAction = () => {  
    alert('woof, woof!')  
  }  
  
  return (  
    <div>  
      <p>This is {props.name}</p>  
      <button onClick={barkAction}>  
        Bark button!  
      </button>  
    </div>  
  )  
}
```

KOMPONENT KLASOWY

```
class AnimalComponent extends React.Component {  
  constructor(props) {  
    super(props)  
    this.barkAction = this.barkAction.bind(this)  
  }  
  
  barkAction() {  
    alert('woof, woof!')  
  }  
  
  render() {  
    return (  
      <div>  
        <p>This is {this.props.name}</p>  
        <button onClick={this.barkAction}>  
          Bark button!  
        </button>  
      </div>  
    )  
  }  
}
```

KOMPONENTY FUNKCYJNE VS KLASOWE

KOMPONENT FUNKCYJNY

KOMPONENT KLASOWY

?

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

?

```
  tick() {  
    this.setState(state => ({  
      seconds: state.seconds + 1  
    }));  
  }
```

?

```
  componentDidMount() {  
    this.interval = setInterval(() => this.tick(), 1000);  
  }
```

?

```
  componentWillUnmount() {  
    clearInterval(this.interval);  
  }
```

?

```
  render() {  
    return (  
      <div>  
        Seconds: {this.state.seconds}  
      </div>  
    );  
  }  
}
```

KOMPONENTY FUNKCYJNE VS KLASOWE

	Klasowe	Funkcyjne
Stateless	x	x
Statefull	x	x (hooks)

KOMPONENTY FUNKCYJNE VS KLASOWE

KOMPONENT FUNKCYJNY

```
const Timer = () => {  
  
  const [seconds, setSeconds] = useState(0);  
  
  const tick = () => {  
    setSeconds(seconds => seconds + 1);  
  };  
  
  useEffect(() => {  
    const interval = setInterval(tick, 1000);  
  
    return () => clearInterval(interval);  
  }, []);  
  
  return <div>Seconds: {seconds}</div>;  
};
```



KOMPONENT KLASOWY

```
class Timer extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { seconds: 0 };  
  }  
  
  tick() {  
    this.setState((state) => ({  
      seconds: state.seconds + 1,  
    }));  
  }  
  
  componentDidMount() {  
    this.interval = setInterval(  
      () => this.tick(), 1000  
    );  
  }  
  
  componentWillUnmount() {  
    clearInterval(this.interval);  
  }  
  
  render() {  
    return <div>Seconds: {this.state.seconds}</div>;  
  }  
}
```

REACT HOOKS

- ▶ Potrzebujemy ich, aby "zastąpić" klasowe zachowania komponentów.
- ▶ Nowy sposób na zarządzanie stanem komponentu.
- ▶ Inne podejście do lifecycle methods

PODSTAWOWE HOOKI

- ▶ *useState* - pamiętanie stanu
- ▶ *useEffect* - akcje wykonywane w określonych okolicznościach (np. pojawienie się komponentu)
- ▶ *useContext, useMemo, useRef*
- ▶ Można tworzyć własne hooki

ZASADY HOOKS

- ▶ Tworzymy hooki TYLKO w "top level" komponencie (nie wolno w instrukcjach warunkowych, pętlach, itp.)
- ▶ Funkcje stworzone przy pomocy hooks należy używać tylko w komponentach funkcyjnych React (nie wołać w klasycznych funkcjach poza Reactem).

useState

► Pamięta stan

```
22  function ExampleComponent() {
23      const [count, setCount] = useState(0);
24
25      const clickHandler = () => {
26          setCount(count + 1);
27      };
28
29      return (
30          <div>
31              <p>You clicked {count} times</p>
32              <button onClick={clickHandler}>Click me</button>
33          </div>
34      );
35  }
```

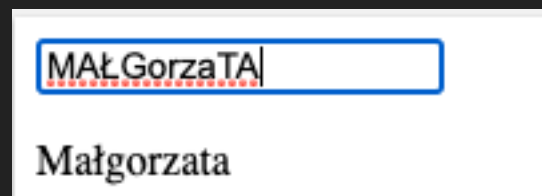
```
function ExampleWithManyStates() {
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
```

Dodajmy Reset button do: <https://stackblitz.com/edit/react-myfunctional1>

ĆWICZENIE Z KOMPONENTAMI FUNKCYJNYMI

Wykorzystajmy Create React App

- ▶ Komponent wyświetlający imię:
 - ▶ Parent komponent z inputem na tekst
 - ▶ Child komponent, który przyjmie imię i je wyświetli
 - ▶ Imię ma być sformatowane (duża tylko pierwsza litera)
 - ▶ HINT - użyj eventu *onChange* (użycie takie samo jak *onClick*) w którym będziesz zapisywać do state'u to co wpisuje użytkownik
- Przykład:



The image shows a white rectangular box containing a text input field and a button. The input field has a blue border and contains the text 'MAŁGorzaTA' in black. Below the input field is a button with the text 'Małgorzata' in black.

useEffect

- ▶ Mutacje, subskrypcje, calle do API, side efekty.

```
useEffect(  
  () => {  
    const result = apiCall.get();  
    console.log(result);  
  },  
  [],  
);
```

```
useEffect(  
  () => {  
    const result = apiCall.get();  
    console.log(result)  
  },  
  [props.type],  
);
```

"Cleanup"

```
useEffect(() => {  
  const subscription = props.source.subscribe();  
  
  return () => {  
    // Clean up the subscription  
    subscription.unsubscribe();  
  };  
});
```

useEffect

- ▶ Hook *useEffect* "kończy się" nawiasem kwadratowym:
(poprawnie mówiąc - "jego drugim parametrem jest *array*")
- ▶ Jeżeli nie podamy tam żadnej wartości to *useEffect* odpali się tylko raz, przy inicjalizacji komponentu
- ▶ Jeżeli podamy tam zmienną, to *useEffect* będzie obserwował wartość tej zmiennej i będzie się odpalał za każdym razem gdy ta wartość się zmieni (można podać kilka zmiennych po przecinku)

```
useEffect(  
  () => {  
    console.log("Zmienna myVar się zmieniła!!")  
  },  
  [myVar]  
);
```


ĆWICZENIE - "LIFECYCLE METHOD" W ŚWIECIE HOOKS

Kontynuując projekt z imieniem:

- ▶ Stwórzcie nowy komponent *MouseComponent*
- ▶ Dodajcie event listener "mousemove"
- ▶ Wyświetlcie pozycję X i Y myszki
- ▶ Pamiętajcie, aby usunąć event listener przy pomocy odpowiedniego *lifecycle method*!
- ▶ Sprawcie, aby po wpisaniu imienia **John Cena** komponent *MouseComponent* zniknął

HOOKS ĆWICZENIE

- ▶ Zastosuj `useEffect` i `setInterval` do generowania błędów: `['alert1', 'alert2', ...]`
- ▶ Po 10s wyświetlamy alert z błędami (`alertList.join(' ,')`)
- ▶ Po 10s od zamknięcia alertu wyświetlamy alert z obecnymi błędami (`setTimeout`)
- ▶ Klikając "ok" potwierdzamy je i usuwamy z kolejki
- ▶ Wszystkie nowe błędy które pojawią się w czasie wyświetlania alertu powinny zostać umieszczone w "kolejce" i zostać wyświetlone w następnym alercie.
- ▶ Nazwa alertu musi być unikalna!

STYLOWANIE W REACT

- ▶ *className* zamiast *class*

```
className="Navbar"
```

- ▶ *style* przyjmuje obiekt postaci

```
style={{ border: "1px solid black", margin: 20, padding: 20 }}
```

- ▶ Użycie Sass lub innego preprocesora wymaga dodania konfiguracji

STYLOWANIE W REACT

- ▶ `import './Style.css'`
 - ▶ Wszystkie style są globalne
- ▶ Inline style
- ▶ CSS Modules
 - ▶ Możliwość separacji stylów
- ▶ Biblioteka z własnymi regułami (np. Styled Components)

STYLOWANIE W REACT

- ▶ `import './Style.css'`
 - ▶ Tworzymy zwykły plik css
 - ▶ Importujemy (działa dzięki webpack)
 - ▶ W komponentach podajemy nazwy klas
- ▶ Do potestowania: <https://stackblitz.com/edit/react-jcyef4>

STYLOWANIE W REACT

▶ Inline styles

- ▶ Rzadziej używane niż klasy
- ▶ Pozwala łatwo zmieniać style przy użyciu JS
- ▶ Raczej nie sprawdza się na większą skalę

```
style={{ border: "1px solid black", margin: 20, padding: 20 }}
```

- ▶ Do potestowania: <https://stackblitz.com/edit/react-jcyef4>

STYLOWANIE W REACT

▶ CSS Modules

- ▶ Przy użyciu create-react-app działają "out of the box"
 - ▶ Separacja stylów per moduł
 - ▶ Przykład - CSS modules a zwykły import
-
- ▶ Do potestowania: <https://stackblitz.com/edit/react-8n2aek?file=src%2FApp.js>

STYLOWANIE W REACT – WNIOSKI

- ▶ React zaleca wiązanie stylu z komponentem
- ▶ Unikanie globalnego stylowania jako głównej metody
- ▶ Warto użyć biblioteki (Styled Components)
- ▶ Używać inline styles tylko gdy tego potrzeba

STYLOWANIE W REACT – STYLED COMPONENTS

- ▶ Musimy zaimportować bibliotekę *styled-components*
- ▶ *Styled components* to przedstawiciel modnego ostatnio nurtu nazywanego CSS-in-JS
- ▶ Tworzymy komponenty razem ze stylami, jako całość
- ▶ Łatwe manipulowanie stylami za pomocą JSa
- ▶ Duża społeczność

STYLED COMPONENTS - BASIC

```
1  import React from "react";
2  import styled from "styled-components";
3
4  const MyDiv = styled.div`
5    background: lightblue;
6    border-radius: 3px;
7    border: 5px solid palevioletred;
8    height: 250px;
9    width: 250px;
10 `;
11
12 const App = () => {
13   return (
14     <div>
15       <h1>My beautiful app</h1>
16       <MyDiv />
17     </div>
18   );
19 };
20
21 export default App;
22
```

My beautiful app



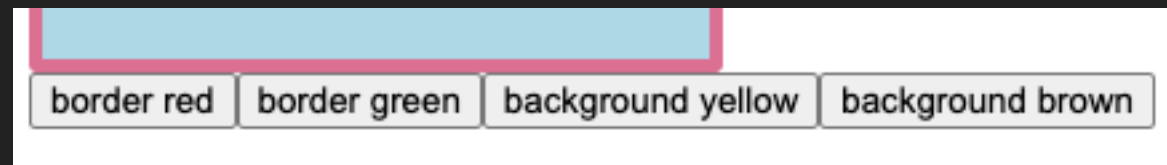
STYLED COMPONENTS - PROPS

```
1  import React from "react";
2  import styled from "styled-components";
3
4  const Title = styled.h1`
5    color: ${props => props.color};
6    font-size: 30px;
7  `;
8
9  const App = () => {
10    return (
11      <div>
12        <Title color="blue">My beautiful app</Title>
13      </div>
14    );
15  };
16
17  export default App;
18
```

My beautiful app

STYLED COMPONENTS – MINI PROJEKT 1

- ▶ Plik startowy: <https://stackblitz.com/edit/react-8acadm>
- ▶ Dodaj cztery buttony z napisami: *"border red"*, *"border green"*, *"background yellow"*, *"background brown"*.
- ▶ Spraw, aby kliknięcie na każdy z tych buttonów zmieniało odpowiedni styl na komponencie *MyDiv*, np. przycisk *"background yellow"* zmienia kolor wypełnienia komponentu *MyDiv* na żółty.
- ▶ TIPS: użyj props dla *styled-component*, a kolory trzymaj w stanie używając *useState* (najpierw to zaimportuj!) i zmieniaj na *onClick*



STYLED COMPONENTS – MINI PROJEKT 2

- ▶ Plik startowy: <https://stackblitz.com/edit/react-jcyef4>
- ▶ Dodać "*dark theme*" aktywowane i dezaktywowane na *button*.
- ▶ Kolory do wykorzystania w "*dark theme*":
 - ▶ Tekst: '#e2e2e2'
 - ▶ Tło: '#121212'

ROUTER

- ▶ "Podstrony z url"
- ▶ Potrzebujemy historii
- ▶ Potrafi emulować "chodzenie" po stronie

ROUTER ZADANIE (7) – WSPÓLNIE

- ▶ Instalacja "react-router-dom"
- ▶ Stwórzcie trzy komponenty (Home, About, Topics)
- ▶ Dodamy dwa route: */about* oraz */topics*
- ▶ Dodajmy przyciski na przełączanie się między routami (docelowo Navbar)
- ▶ Defaultowy route: */home*
- ▶ 404 page

ROUTER ZADANIE (7) – WSPÓLNIE

- ▶ <https://stackblitz.com/edit/react-router-starter-vd7fmo>

REACT CONTEXT DEFINICJA *

```
const SampleContext = React.createContext("default-value")
```

```
<SampleContext.Provider value={this.state.value}>  
  <div>...</div>  
</SampleContext.Provider>
```

- ▶ Dzielenie danych pomiędzy komponentami
- ▶ Dostępny w wersji "klasowej" i "hookowej"

REACT CONTEXT UŻYCIE

```
import React from "react";  
import { SampleContext } from "../SampleContext";
```

```
class Profile extends React.Component {  
  render() {  
    // this.context daje nam wartość  
    return (  
      <div>{this.context}</div>  
    );  
  }  
}
```

```
Profile.contextType = SampleContext;
```

- ▶ Context to może być również obiekt
- ▶ Obiekt może zawierać funkcje

ZADANIE CONTEXT (9)

- ▶ Tłumaczenie dla naszej strony

REF*

```
this.textInput = React.createRef();
```

```
<input type="text" ref={this.textInput} />
```

- ▶ Dzięki niemu możemy dostać referencję do node
- ▶ W pewnych przypadkach przydatny przy pracy z hooks
- ▶ Dostępny w wersji "klasowej" i "hookowej"
- ▶ Stosowane, żeby np. programowo zrobić *focus* na *input*

NA NASTĘPNE ZAJĘCIA

W DOMU

- ▶ Polecam poczytać o Redux

PROJEKT

- ▶ Karta projektu
- ▶ Projekt wysłać mailem na:
wiktor.jurczyszyn@wsb.wroclaw.pl

POLECANE MATERIAŁY DO NAUKI

- ▶ <https://frontendmasters.com/>
- ▶ <https://overreacted.io/why-do-we-write-super-props/>
(super(props))
- ▶ <https://www.newline.co/fullstack-react>
- ▶ <https://javascriptweekly.com/>