Zaznaczam, że poniższe notatki były robione przez osobę nie będącą fanem frontendu i nie pracującą w tym obszarze za wiele. Moje doświadczenie z Reactem to raptem pare godzin. A nóż komuś się to przyda.

Zawartość:

- struktura projektu

- komponenty klasowe i funkcyjne

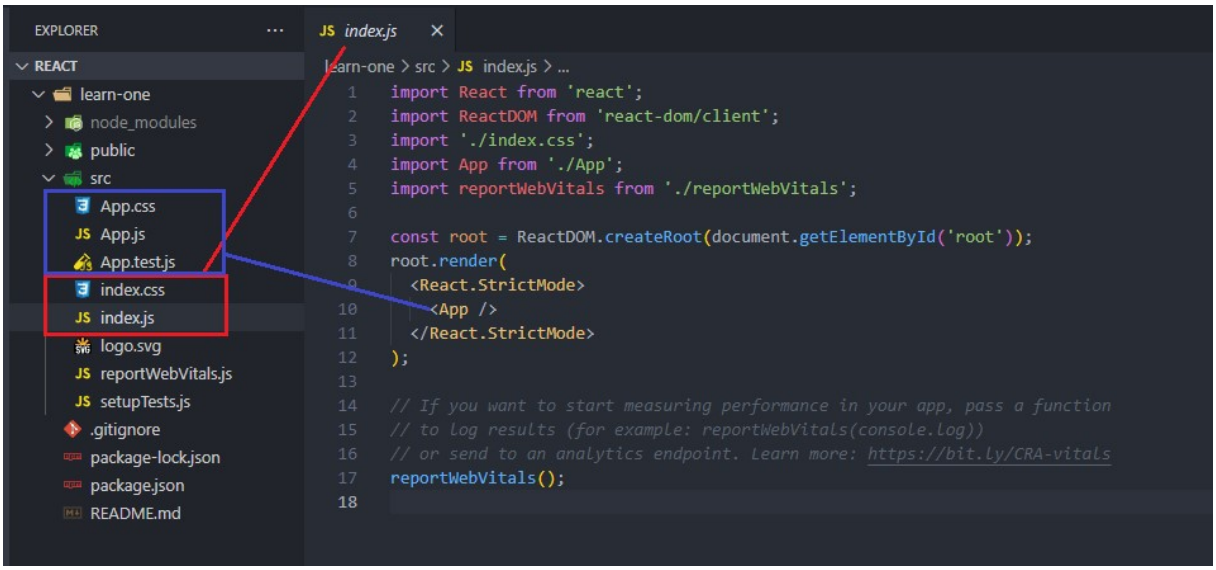
- stan

- props

Otwarcie nowego projektu:

npx create-react-app learn-one

Plik index.js



- index.js jest takim naszym mainem, punktem wejścia do aplikacji
- na obrazku widać jak umieszczany jest w strukturze DOM pierwszy komponent o nazwie App
- komponent App został zdefiniowany w pliku App.js
- komponent App swoje style przechowuje w pliku App.css
- komponent App testy jednostkowe przechowuje w pliku App.test.js (jak opisywałem framework do testów „jest” to tam było, że szuka on plików x.test.js)
- na powyższym obrazku widać także jak sobie wszystko importować
    - jest przykład z App gdzie importowany jest komponent
    - jest przykład ze stylami gdzie importowany jest ./index.css
    - jest przykład jak importować elementy z biblioteki standardowej React
    - każda doinstalowana biblioteka w swoim readme zapewne posiada informacje jak do importować każdy z jej elementów

Komponent – ja to rozumiem jako jakiś element interfejsu. Np. przycisk może być komponentem, który skała się z innych komponentów (napisu, obrazka, tła). Ten przycisk (komponent), może być elementem innych komponentów. Zasada jak z klockami. Z małych klocków budujemy większe i tak dalej.

Plik App.js

learn-one > src > JS App.js > ...

```
import './App.css';

function App() {
  return (
    <div className="App">
      <h1 id="Header1">Hello World!</h1>
    </div>
  );
}

export default App;
```

- do zdefiniowania komponentu App została użyta funkcja (więcej na ten temat niżej)
- ponownie widzimy tutaj mechanizm importowania styli CSS
- **na co warto zwrócić uwagę to, że do nadawania elementom klas służy atrybut className a nie class**
- można także używać standardowego id **(jeżeli instancjonujemy dany komponent więc niż raz w aplikacji to można spodziewać się błędów bo każde id może występować tylko raz na stronie – dlatego klasy są lepsze)**
- aby dało się zaimportować element z tego pliku w innym należy go wyeksortować

Plik App.css

learn-one > src > App.css > ...

```
.App {
  text-align: center;
}

#Header1 {
  color: red;
}
```

- tutaj raczej nic nadzwyczajnego nie zobaczymy, zwykły CSS

Komponenty:

- komponenty czyli te nasze klocki można definiować za pomocą funkcji lub klas
- dawniej robiło się to za pomocą klas jeżeli chciało się mieć komponent, który może się zmieniać i funkcji jeżeli komponent nie wymagał zmiany (chodzi o zmianę swojego stanu)
- teraz wszyscy się przenieśli na funkcję (dzięki hook'om, możliwe jest zmienianie stanu)
- z tego co czytałem komponenty klasowe generują więcej kodu po „kompilacji" i co za tym idzie są cięższe

Komponent funkcyjny

learn-one > src > JS ComponentFunc.js > default

```
import React from "react";

function ComponentFunc(props){
    //some logic here executed once
    return(
        <div>
            <h4 style={{color:"red"}}>Komponent funkcyjny!</h4>
            <h5>{props.text}</h5>
            <hr></hr>
        </div>
    );
}

export default ComponentFunc;
```

- zdefiniowano powyżej komponent o nazwie ComponentFunc
- zwraca on prostą strukturę html
- komponent ten jest funkcją, więc całe jej ciało wykona się linia po linii, jeżeli oprócz return umieścimy tam jakąś logikę
- w kodzie zawarto przykład kodu css umieszczonego bezpośrednio w znaczniku
- **warto zwrócić uwagę na parametr props, który pozwala przekazywać parametry do komponentów z zewnątrz (przykład niżej – w momencie wywołania)**

Komponent klasowy

learn-one > src > JS ComponentClass.js > default

```
import React from "react";

class ComponentClass extends React.Component{
    //some definitions here

    constructor(){
        super(); //class base class constructor
        //some logic here
    }

    render(){
        return(
            <div>
                <h4 style={{color:"red"}}>Komponent klasowy!</h4>
                <h5>{this.props.text}</h5>
                <hr></hr>
            </div>
        );
    }
}

export default ComponentClass;
```

- każdy kto programował obiektowy kojarzy konstrukcję klasy, która zawierać może konstruktor, zmienne oraz metody
- komponent klasowy dziedziczy z React.Component
- kod w takim komponencie nie jest wykonywany linia po linii (wykonywany jest konstruktor a następnie metoda render())
- **props w tym przypadku jest polem dziedziczonym z React.Component o czym świadczy słowo kluczowe this**
- metoda render() służy do definiowania struktury html komponentu

Wywołanie:

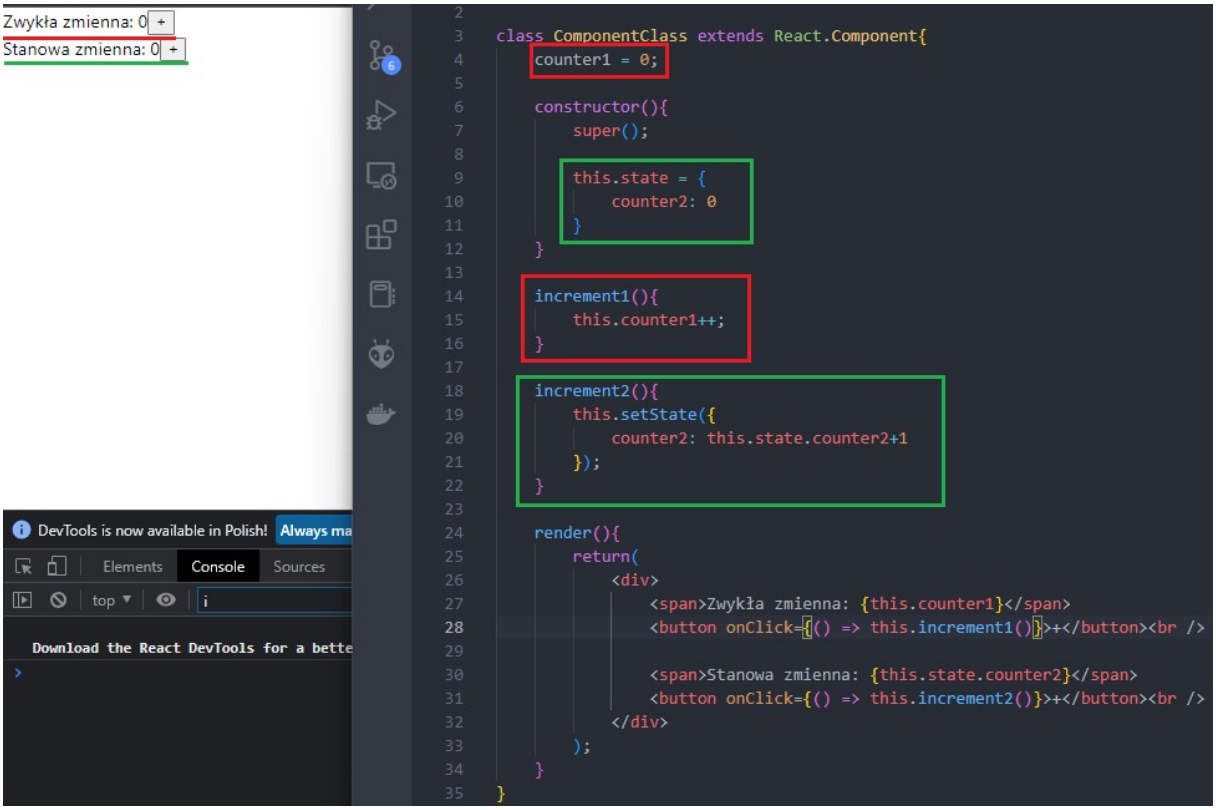
```
learn-one > src > JS index.js > ...
1   import React from 'react';
2   import ReactDOM from 'react-dom/client';
3   import './index.css';
4   import reportWebVitals from './reportWebVitals';
5
6   import ComponentFunc from './ComponentFunc';
7   import ComponentClass from './ComponentClass';
8
9   const root = ReactDOM.createRoot(document.getElementById('root'));
10  root.render(
11    <React.StrictMode>
12      <ComponentFunc text="Ten popularniejszy!" />
13      <ComponentClass text="Ten mniej popularny!" />
14    </React.StrictMode>
15  );
16
```

- oba komponenty zostały zaimportowane
- za pomocą atrybutu text (sami sobie go wymyślamy – byle by nie użyć czegoś co jest używane przez frawework jak np. id) przekazujemy parametr do komponentu (dokładniej do pola obiektu **props**, o którym wspomniałem wyżej)
- atrybutów oczywiście może być więcej

Stan:

- stan to zestaw zmiennych, które opisują stan komponentu
- zmiana dowolnej zmiennej stanu wymusza odświeżenie komponentu i wszystkich komponentów potomych
- zwykłe zmienne (pola w klasie) nie mają takiej możliwości – nawet jeżeli zmienimy ich wartość komponent się nie odświeży

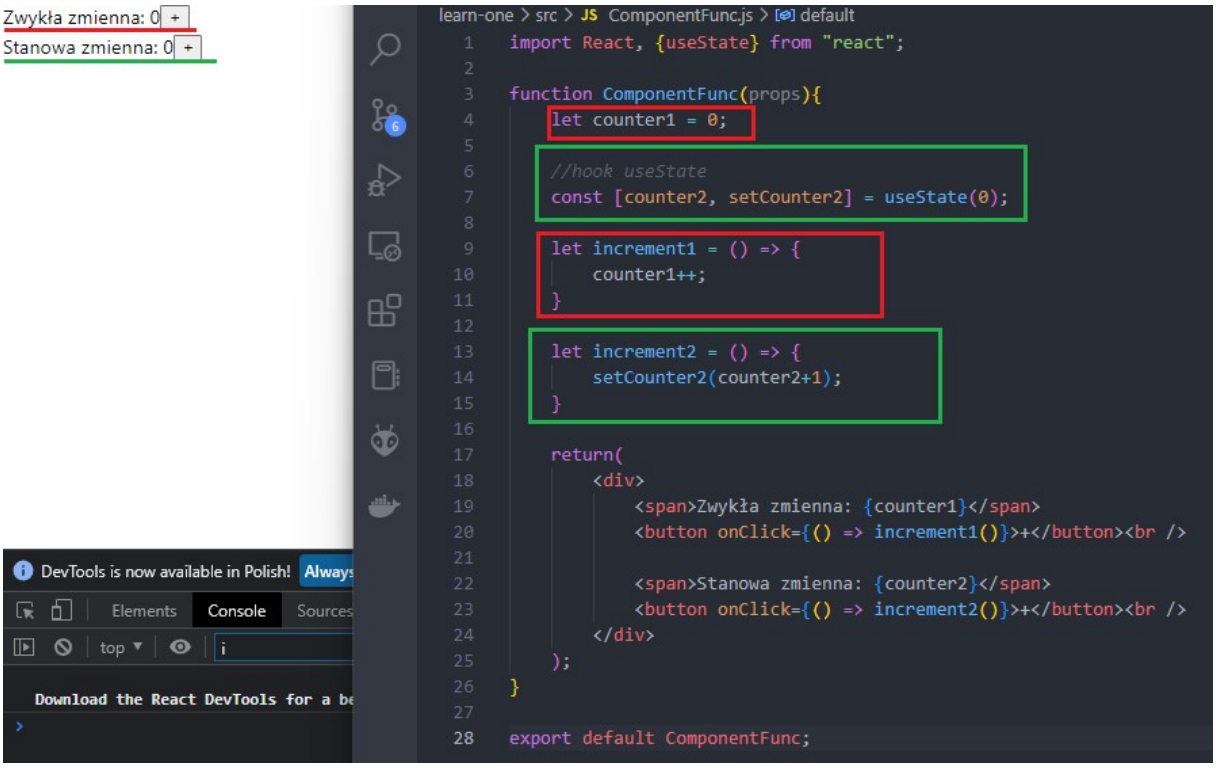
Stan komponentu klasowego:



- kolorem czerwonym zaznaczono
  - zwykłe pole klasy
  - metodę wywoływaną przyciskiem, która zwiększa wartość wcześniej wspomnianego pola klasy
  - wciskając przycisk plus zaznaczony na czerwono wartość wypisana na ekranie **NIE ZMIENIA SIĘ**, ale wartość pola w klasie **ZMIENIA SIĘ**
  - wynika to z tego, że counter1 nie jest zmienną stanu i jego wartość używana tylko **w momencie renderowania komponentu**
- kolorem zielonym zaznaczono
  - definicje zmiennej stanu
  - metodę wywoływaną przyciskiem, która zwiększa wartość wcześniej wspomnianej zmiennej stanu

- wciskając przycisk plus zaznaczony na zielono wartość zmiennej stanu **ZMIENI SIĘ,** komponent zostanie wygenerowany na nowo i na ekranie zobaczymy wartość większą o 1
- **należy pamiętać, że jeżeli zmieniliśmy pole klasy klikając czerwony przycisk to zobaczymy zmiany w momencie kliknięcia zielonego przycisku, ponieważ wymusiliśmy wygenerowanie komponentu jeszcze raz zmianą stanu**
  - plus czerwony – brak zmian (0 i 0)
  - plus czerwony – brak zmian (0 i 0)
  - plus zielony – zmiana (2, 1)

Stan komponentu funkcyjnego:



- kolorem czerwonym zaznaczono
  - zwykłą zmienną counter1
  - callback, który ją zwiększa
  - wciśnięcie czerwonego przycisku nie powoduje żadnych zmian
- kolorem zielonym zaznaczono
  - zmienną stanu counter2
    - jest to tak zwany hook setState(), który dodaje obsługę stanu do komponentu funkcyjnego
    - przyjmuje się konwencje [nazwaZmiennej, setNazwaZmiennej] = useState(wartość domyślna)
  - callback, który ją zwiększa
  - wciśnięcie zielonego plusa powoduje odświeżenie licznika na ekranie (wygenerowanie komponentu na nowo)
- jak widać komponenty funkcyjne są krótsze i przyjemniejsze w pisaniu