Laboratorium programowania: frameworki aplikacji internetowych

Materiały do zajęć laboratoryjnych

React – przydatne komendy

- npm install -g create-react-app
- npx –help
- npx create-react-app nazwaprojektu
- npm run start

Zadania:

1. Dodaj komponent odpowiedzialny za umieszczenie na stronie nagłówka z nazwą przedmiotu i linku do strony internetowej naszego wydziału oraz do strony głównej Reacta.
2. Dodaj komponent, który pozwoli zwrócić informację o godzinie wejścia na stronie. Umieść go jako pierwszy górny element na stronie.
   Wskazówka: wykorzystaj `new Date().toLocaleTimeString()`
3. Poniżej komponentu przedstawiającego języki, które warto znać, aby pisać aplikację z użyciem React, umieść komponent wyświetlający listę języków programowania z którymi spotkałeś się już podczas studiów.
4. Dodaj komponent z listą przedmiotów na jakie uczęszczasz w tym semestrze.
5. Umieść na stronie przycisk „Więcej o przedmiocie” (należy umieścić jedynie sam przycisk, na razie nie wykonuje on żadnej akcji).

Create-react-app

Na poprzednich zajęciach zapoznaliśmy się z ideą komponentu, jednakże przedstawiony tam sposób tworzenia projektu idealnie sprawdzi się jedynie jako wprowadzenie do idei komponentu i powinien być stosowany tylko do początkowej nauki. Informacja o tym fakcie znajduje się także na oficjalnej stronie Reacta z której pobieraliśmy przykładowy plik z komponentem:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello World</title>
    <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>

    <!-- Don't use this in production: -->
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">

      ReactDOM.render(
        <h1>Hello, world!</h1>,
        document.getElementById('root')
      );

    </script>
    <!--
      Note: this page is a great way to try React but it's not suitable for production.
      It slowly compiles JSX with Babel in the browser and uses a large development build of
React.

      Read this section for a production-ready setup with JSX:
      https://reactjs.org/docs/add-react-to-a-website.html#add-jsx-to-a-project

      In a larger project, you can use an integrated toolchain that includes JSX instead:
      https://reactjs.org/docs/create-a-new-react-app.html

      You can also use React without JSX, in which case you can remove Babel:
      https://reactjs.org/docs/react-without-jsx.html
    -->
  </body>
</html>
```

Używanie JSX w skryptach z atrybutem text/babel znacznie spowalnia aplikacje. Korzystanie z JSX wymaga więc zainstalowanego na komputerze środowiska Node.js.

Biblioteka React nie należy do najłatwiejszych. Ma stosunkowo wysoki próg wejścia i m.in. wymaga dobrej znajomości JavaScript. React powstał na potrzeby Facebooka, a nie z myślą o powszechnym wykorzystaniu tego frameworka. W związku z tym pojawiło się narzędzie stworzone przez twórców Reacta, dzięki któremu można szybko utworzyć startową aplikację: create-react-app:

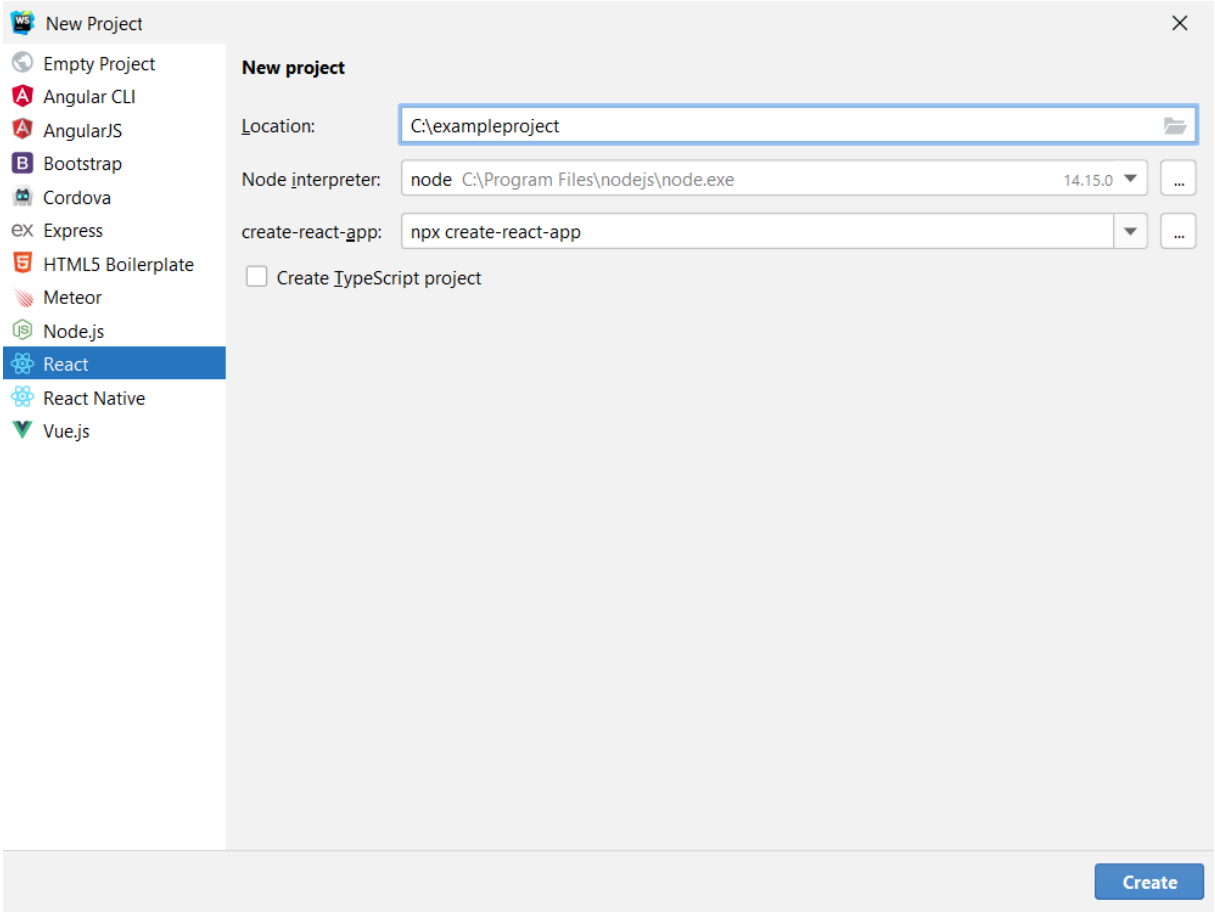
https://github.com/facebook/create-react-app

https://pl.reactjs.org/docs/create-a-new-react-app.html

Projekt przy użyciu create-react-app tworzymy używając komendy:

npx create-react-app nazwaprojektu

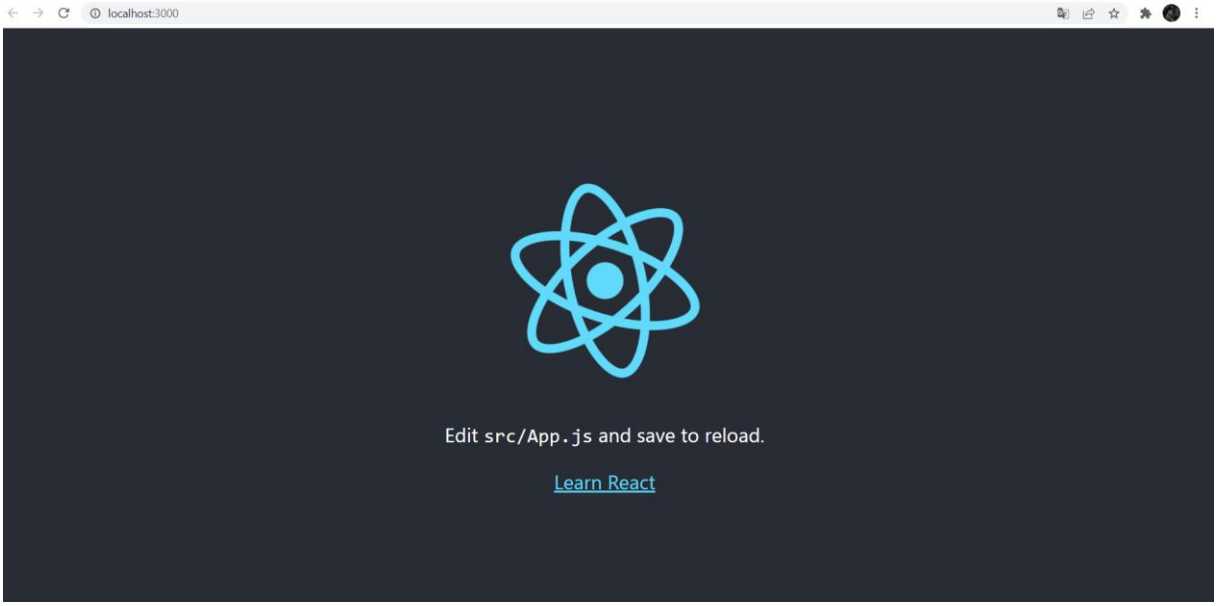
Jeżeli tworzymy projekt w WebStormie, możemy utworzyć nowy projekt automatycznie, bez konieczności wpisywania powyższej komendy:



Aplikację (także tą startową) uruchamiany używając komendy:

npm run start

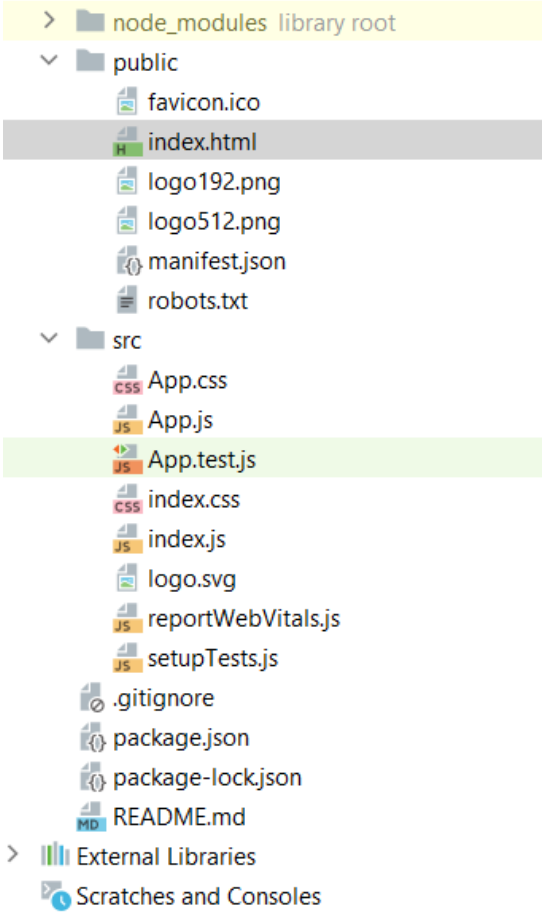
Adres: http://localhost:3000/



Utwórzmy przykładowy projekt: gameapp

STRUKTURA PROJEKTU:

- index.html



- index.js

  - import React oraz ReactDOM

```
node_modules library root
public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
.gitignore
package.json
package-lock.json
README.md
External Libraries
Scratches and Consoles
```

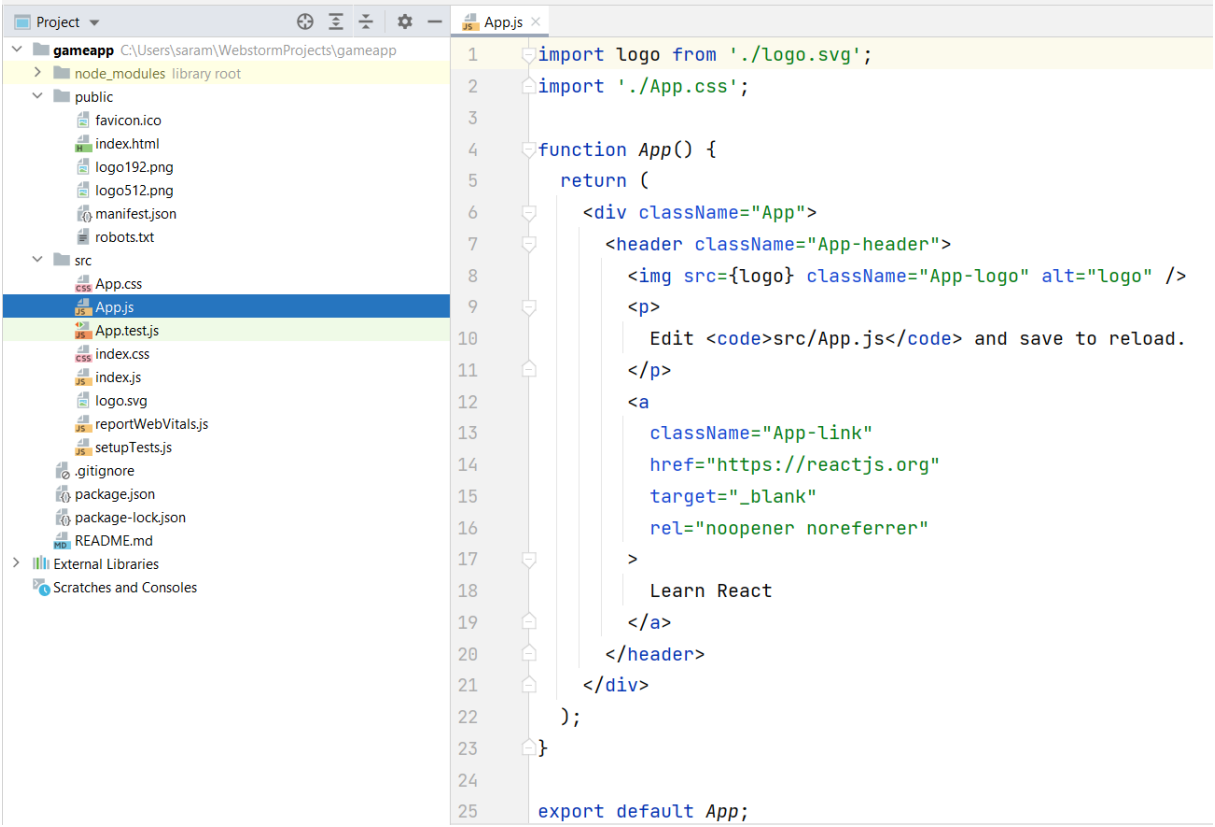
```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  import reportWebVitals from './reportWebVitals';
6
7  ReactDOM.render(
8    <React.StrictMode>
9      <App />
10   </React.StrictMode>,
11   document.getElementById( elementId: 'root')
12 );
13
14 // If you want to start measuring performance in your app, pass
15 // to log results (for example: reportWebVitals(console.log))
16 // or send to an analytics endpoint. Learn more: https://bit.ly/
17 reportWebVitals();
```

  - insert kodu Reacta w HTML

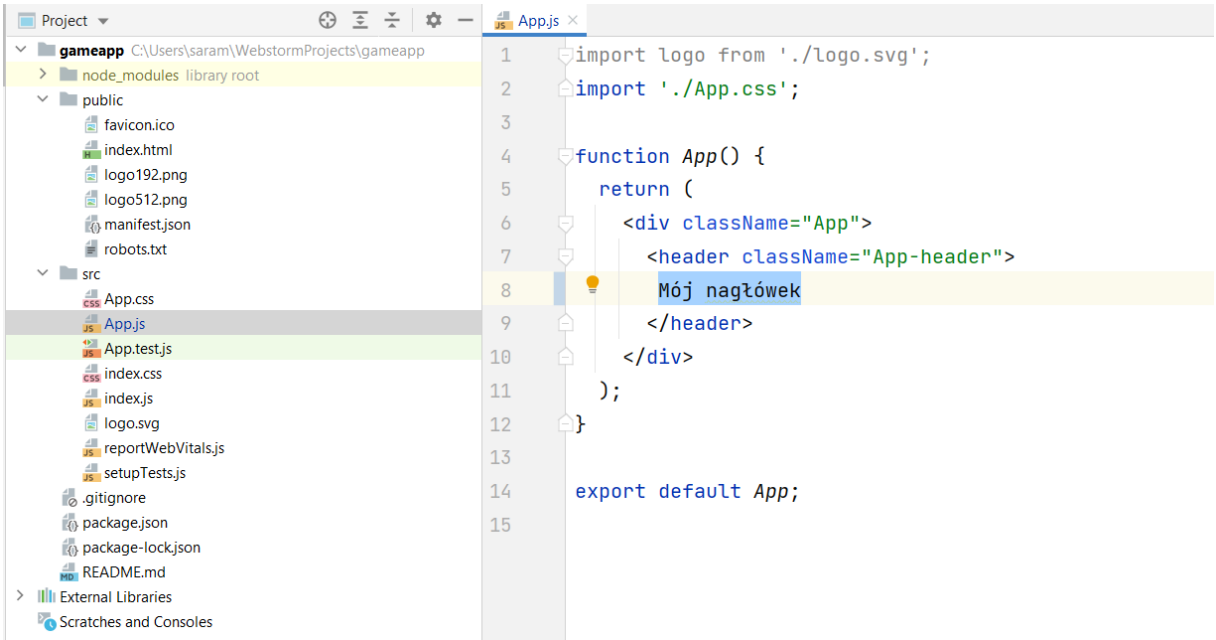
```
node_modules library root
public
    favicon.ico
    index.html
    logo192.png
    logo512.png
    manifest.json
    robots.txt
src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    reportWebVitals.js
    setupTests.js
.gitignore
package.json
package-lock.json
README.md
External Libraries
Scratches and Consoles
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById( elementId: 'root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

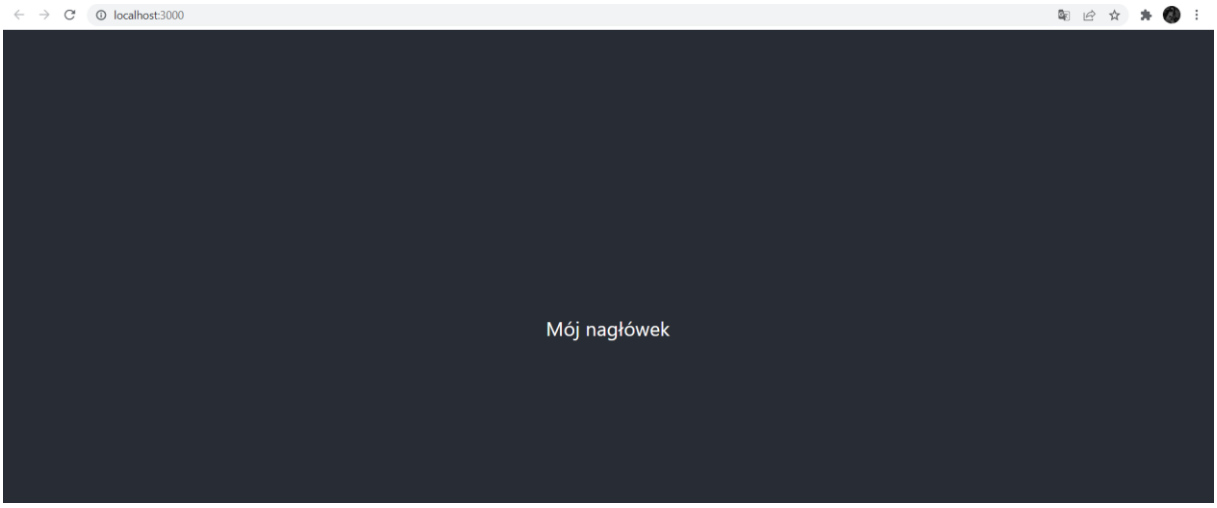
- App. Js – główny komponent aplikacji



Przykład: zmieńmy header na nasz własny napis, na przykład „Mój nagłówek”



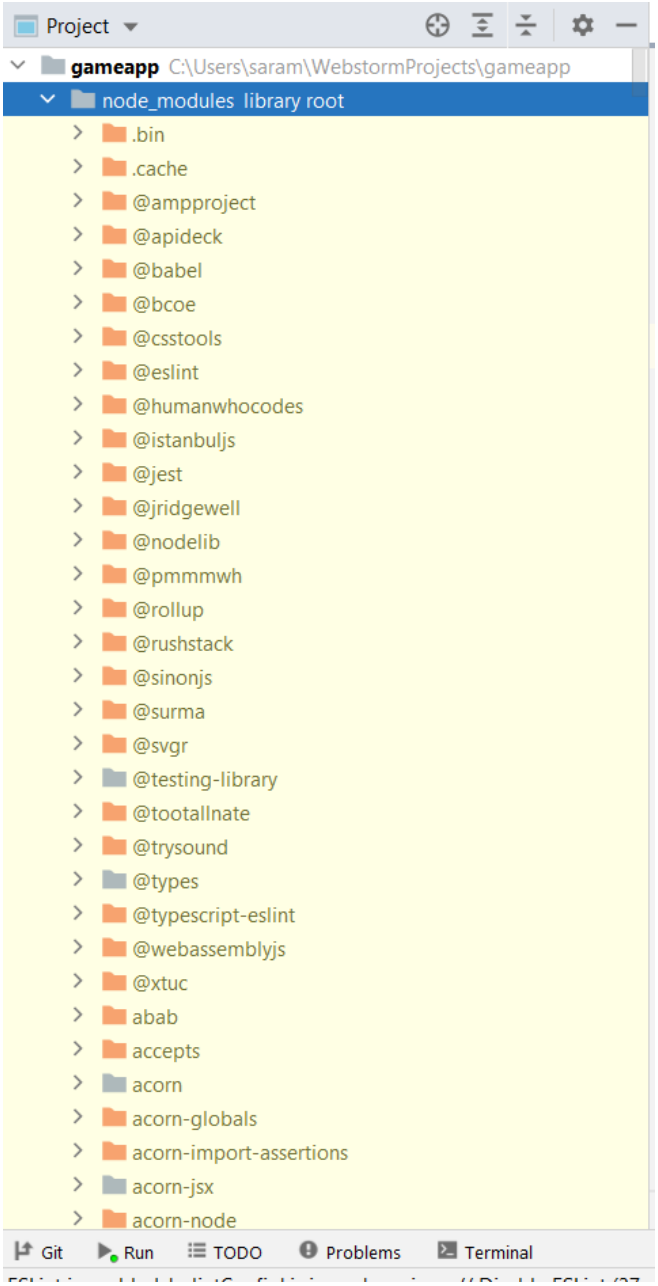
Zmiana zostanie odświeżona w czasie rzeczywistym:



Zauważmy, że faktycznie nie używamy tutaj czystego html, a JSX. Słowo class jest zastrzeżone do deklarowania klasy, dlatego do określania klasy, na przykład w celu zdefiniowania stylu, używamy className:

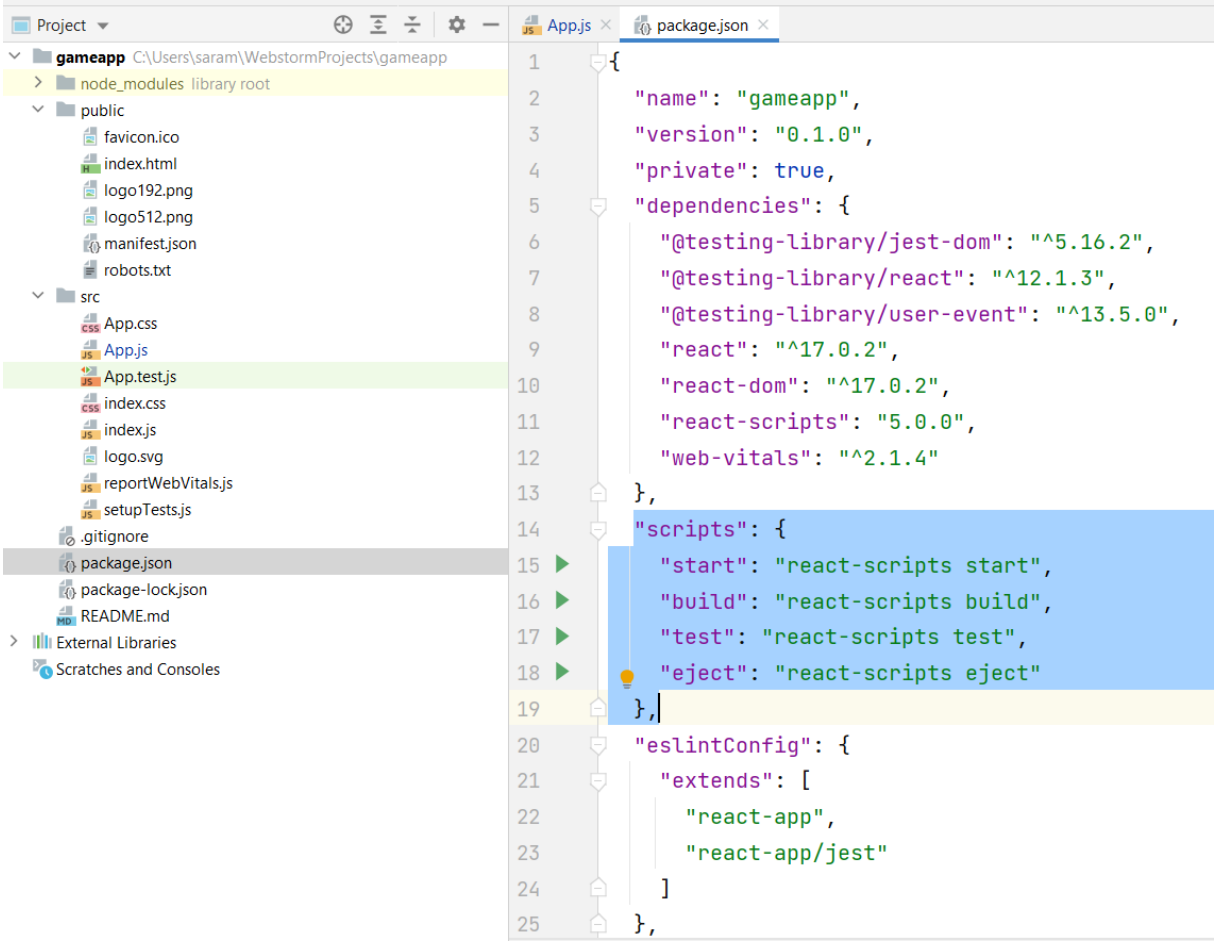
```
App.js
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
       Mój nagłówek
      </header>
    </div>
  );
}

export default App;
```

- Node_modules - pakiety i biblioteki



Nie musimy dokładnie znać ich struktury.

- package.json



Przykład

Usuńmy katalog src. Wprowadzimy swój własny kod od podstaw. Spróbujemy umieścić dwa komponenty na wzór tych, które napisaliśmy na poprzednich zajęciach.

Utwórzmy nowy pusty katalog **src**.

Dodajemy do tego katalogu własny plik **index.js.**

W pierwszej kolejności importujemy reacta:

```
import React from 'react';
import ReactDOM from 'react-dom';
```

Dla testów wyrenderujmy napis „Wszystko działa” odwołując się do id elementu root z index.html:

```
ReactDOM.render(<h1>Wszystko działa</h1>,document.getElementById('root'))
```

```
index.js   App.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3
4
5  ReactDOM.render(<h1>Wszystko działa</h1>,document.getElementById( elementId: 'root'))
6
```

Rozszerzmy teraz aplikację o nasz pierwszy komponent, będący jednocześnie głównym komponentem. Dodajmy plik App.js i za jego pomocą wyświetlmy na ekranie informację, że aplikacja działa:

```
import React, {Component} from 'react'

class App extends Component{
    render(){
        return <div>
            <h1>React App</h1>
            <h2>Aplikacja działa</h2>
        </div>
    }
}

export default App;
```

Export default App jest konieczny do współdzielenia pliku.

Jeśli dodamy {Component}, to nie będzie trzeba robić extends React.Component, tylko można od razu extends Component.

Wyrenderujmy teraz nasz nowo utworzony komponent App. Przejdźmy do index.js i dopiszmy:

```
index.html   index.js   App.js
portfolio > src > index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import App from './App'
4
5  ReactDOM.render(<App></App>,document.getElementById('root'))
```

I zamieńmy poprzedni nagłówek na <App></App>.

Zapis ReactDOM.render**(<App />,**document.getElementById('root')) zadziała tak samo.

Dodajmy teraz komponent Greeting, który utworzyliśmy na ostatnich zajeciach. Dodajemy nowy plik greeting.js:

```
import React, {Component} from 'react'

class Greeting extends Component {
    render(){
        var headline = 'Nasz komponent z ostatnich zajęć'
        return (<div>
            <h1>Hello world.</h1>
            <h2>{headline}</h2>
            <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut
rutrum erat risus, in semper lorem interdum a. In eget pretium urna, in
commodo orci. </p>

        </div>)
    }
}
export default Greeting;
```

Teraz wystarczy umieścić komponent w App.js:

```
index.js   Greeting.js   App.js
1   import React, {Component} from 'react'
2   import Greeting from "./Greeting";
3
4   class App extends Component{
5       render(){
6           return <div>
7               <h1>React App</h1>
8               <h2>Aplikacja działa</h2>
9               <Greeting/>
10          </div>
11      }
12  }
13
14  export default App;
15
```

Dodajmy teraz komponent, który będzie odpowiadał za umieszczenie na stronie listy języków programowania, które należy znać (analogicznie do komponentu z ostatnich zajęć). Dodajmy plik languages.js:

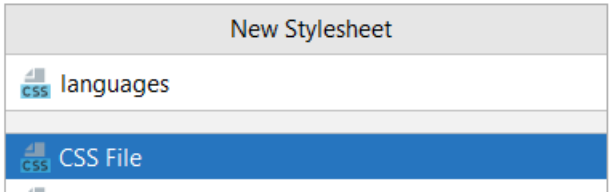
```
import React, {Component} from 'react'

class Languages extends Component {

    render() {
        var arrLanguages = ['HTML', 'JavaScript', 'CSS']
        const languagesList = arrLanguages.map(lan => <li
key={lan}>{lan}</li>)
        return <div>
            <h3>Języki które należy znać:</h3>
            <ul>{languagesList}</ul>
        </div>
    }
}

export default Languages;
```

```
index.js   greeting.js   App.js   languages.js
1  import React, {Component} from 'react'
2
3  class Languages extends Component {
4
5      render() {
6          var arrLanguages = ['HTML', 'JavaScript', 'CSS']
7          const languagesList = arrLanguages.map(lan => <li key={lan}>{lan}</li>)
8          return <div>
9              <h3>Języki które należy znać:</h3>
10             <ul>{languagesList}</ul>
11         </div>
12     }
13 }
14
15 export default Languages;
16
```

Aby umieścić go na stronie umieszczamy go w App:

```
index.js   greeting.js   App.js   languages.js
1  import React, {Component} from 'react'
2  import Greeting from "./greeting";
3  import Languages from "./languages";
4
5  class App extends Component{
6      render(){
7          return <div>
8              <h1>React App</h1>
9              <h2>Aplikacja działa</h2>
10             <Greeting/>
11             <Languages/>
12         </div>
13     }
14 }
15
16 export default App;
17
```

Jak dodać style do projektu? Zmieńmy na przykład nagłówek h3 w komponencie Languages. Dodajmy nowy katalog css. Następnie dodajmy plik css:



I odpowiedni styl:

```
languages.js   languages.css
1  h3 {
2      color: lavender;
3  }
4
```

Aby zastosować zmiany, należy jeszcze zaimportować ten plik do pliku js z komponentem, w którym chcemy go użyć:

languages.js languages.css

```
1  import React, {Component} from 'react'
2  import './css/languages.css'
3
4  class Languages extends Component {
5
6      render() {
7          var arrLanguages = ['HTML', 'JavaScript', 'CSS']
8          const languagesList = arrLanguages.map(lan => <l
9          return <div>
10             <h3>Języki które należy znać:</h3>
11             <ul>{languagesList}</ul>
12         </div>
```

Nasz projekt jak na razie jest bardzo statyczny. Za tydzień będziemy mówić o stanach i pozwolimy na zmiany treści na stronie, na przykład w wyniku kliknięcia w przycisk. Poćwiczmy jednak najpierw tworzenie prostych komponentów na przedstawionych w pliku zadaniach.