Web Frameworks

Laboratory 03

mgr Sara Jurczyk

React – useful comends

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

Create-react-app

The way we created components during our last class should be used only to get the idea of creating components, and using JSX slows down the application:

```
<!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>
```

To create purely React application we can use a tool called create-react-app:

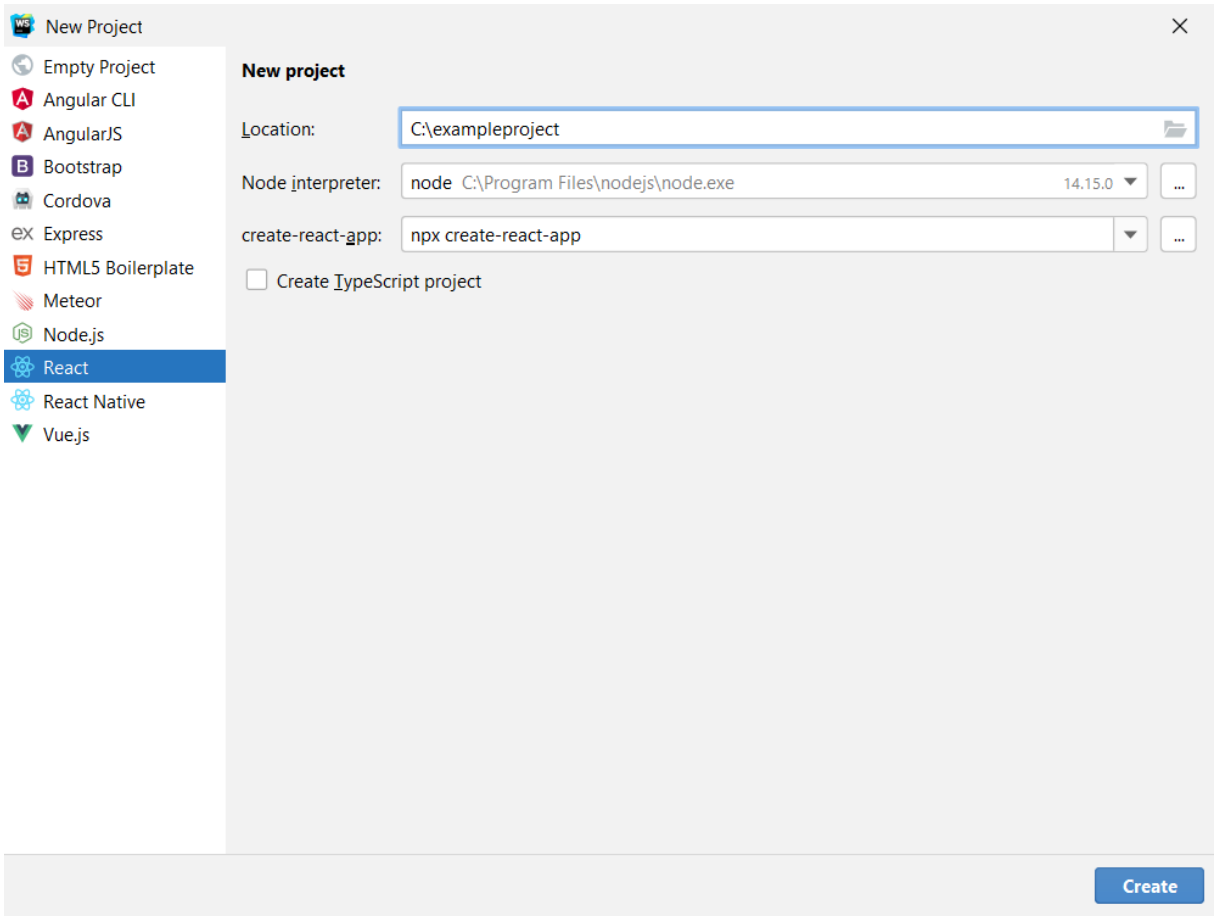
https://github.com/facebook/create-react-app

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

To create a project using create-react-app tool, type in terminal:

npx create-react-app projectname

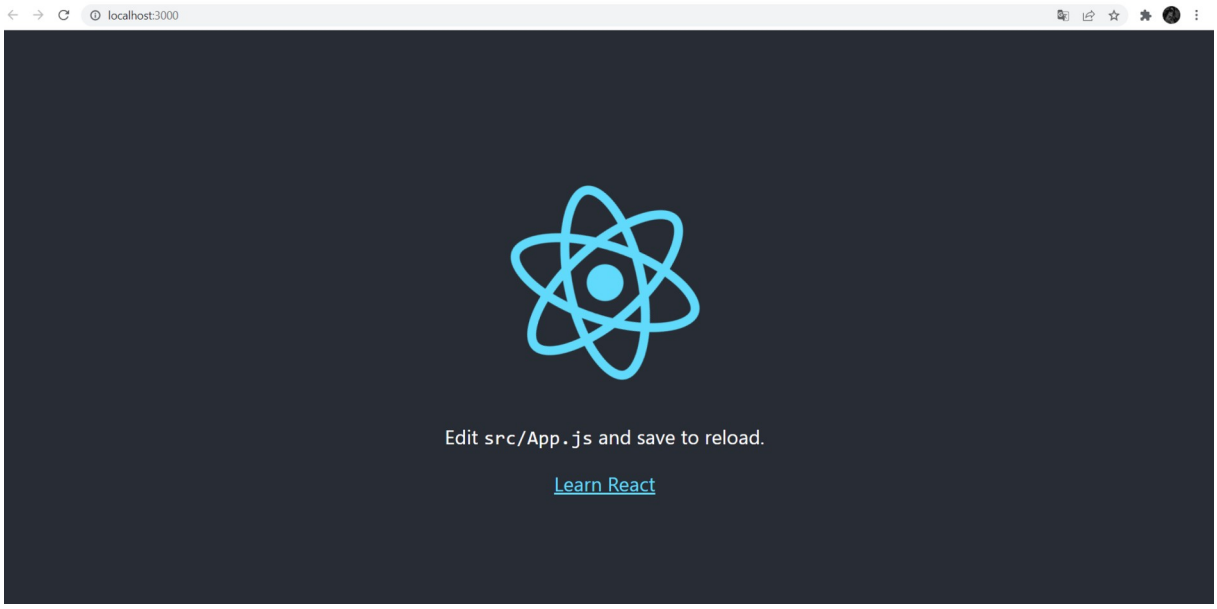
If one uses WebStorm, there is a build-in solution that allows creating a project with create-react-app automatically :



Application can be run by typing in terminal:

npm run start

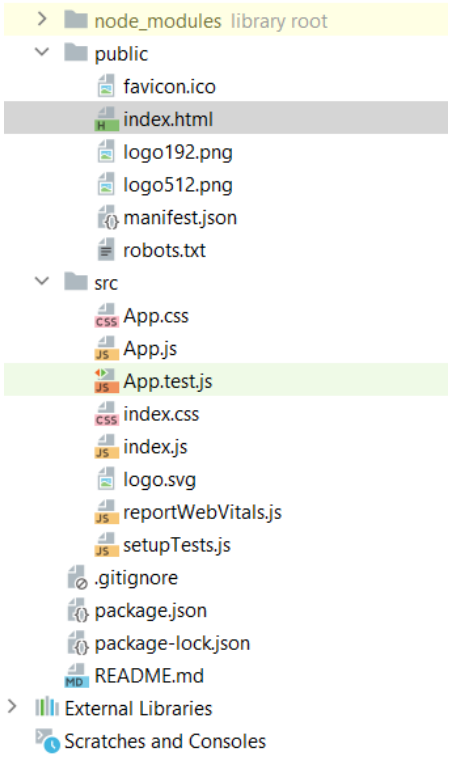
URL of the output: http://localhost:3000/



Create an example project called: gameapp.

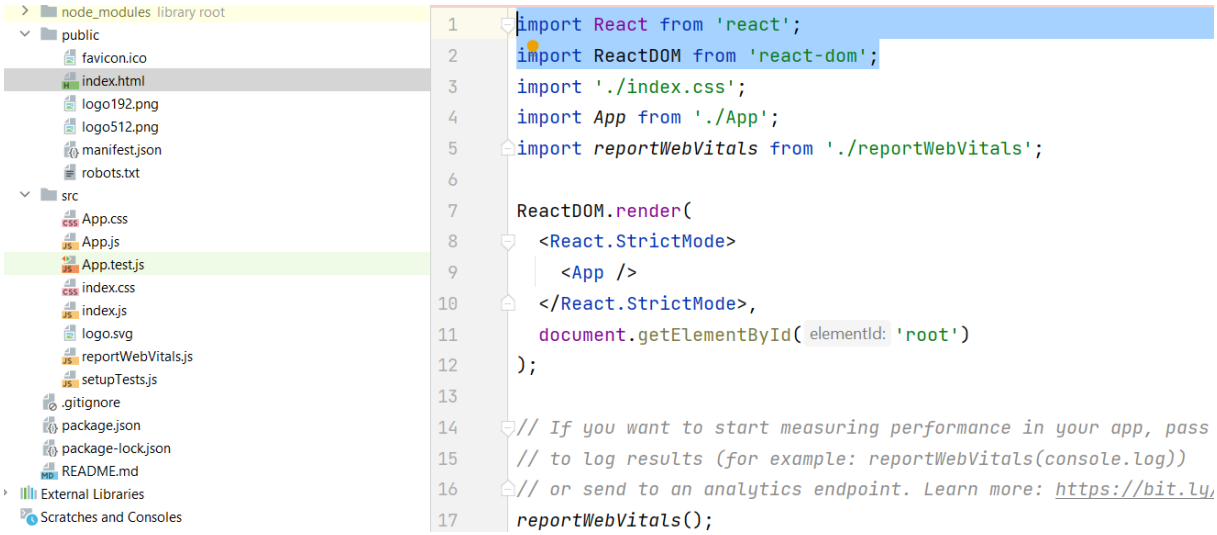
PROJECT STRUCTURE

- index.html

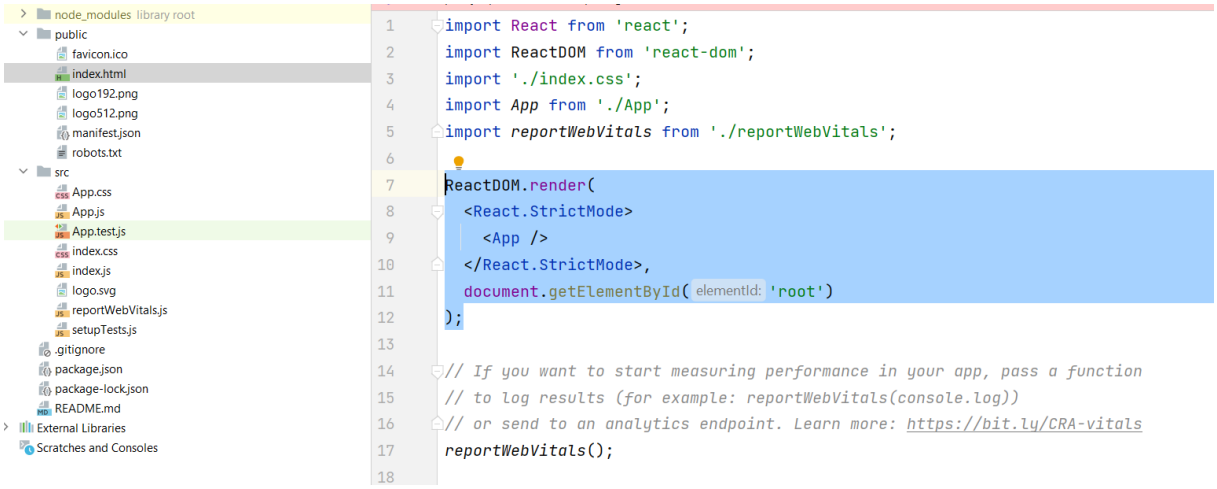


- index.js

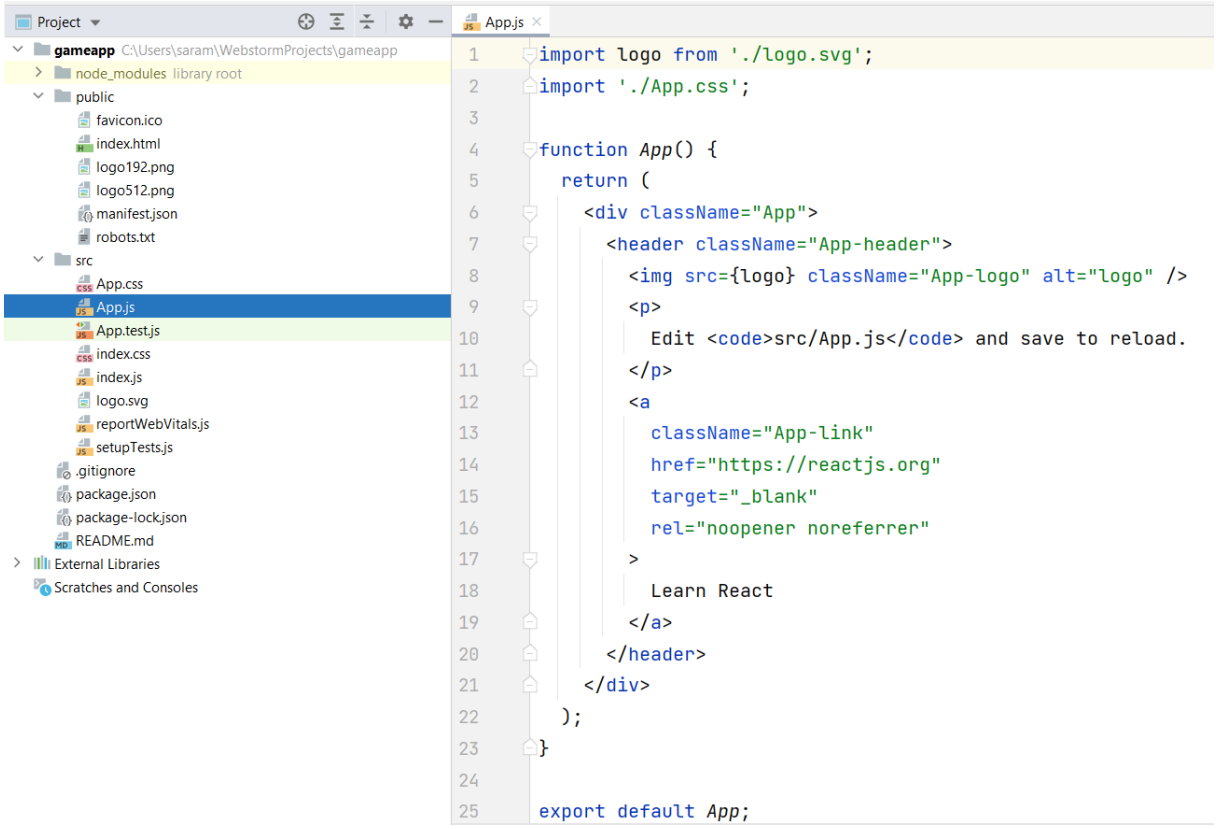
- import React and ReactDOM



- inserting React component into HTML div of root id



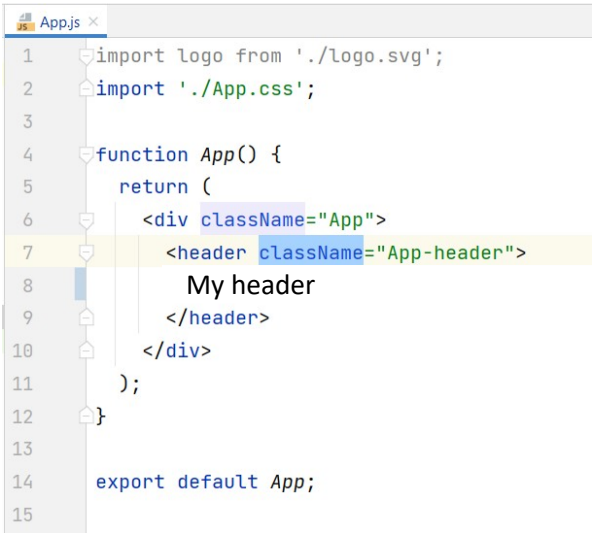
- App. Js – main component of the application



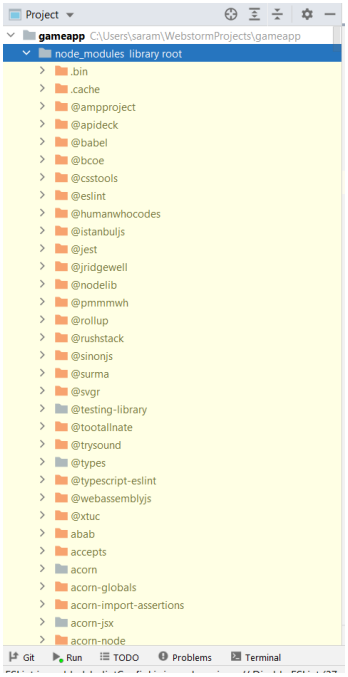
EXAMPLE: change the content of header and replace it with a paragraph „My header”.

The change should be seen on the screen dynamically, without refreshing the project or site.

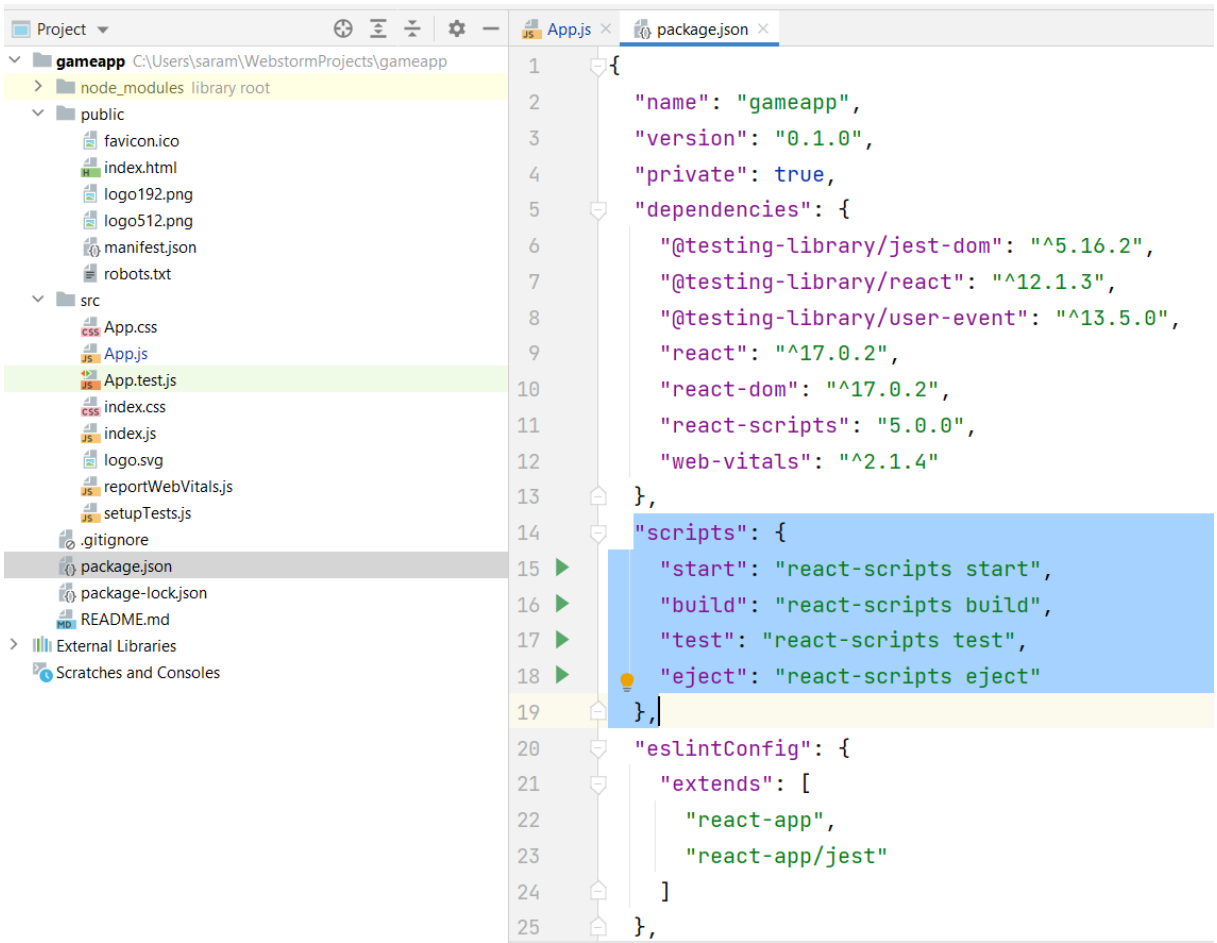
Note that we use JSX, not HTML here. Keyword class is restricted in js for declaring a class – to create a style using a css class, we should use className keyword:



- Node_modules – packages and libraries. We do not need to know the structure of it.



- package.json



EXAMPLE

Let's remove the source src directory and replace it with our own source code. Let's try to place here two component created during our last class and see the differences.

Create a new empty **src** directory. Add to the src directory a new js file: **index.js.**

In the file index.js, let's import React and ReactDOM:

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

And add a content that should be displayed on the screen, e.g.:

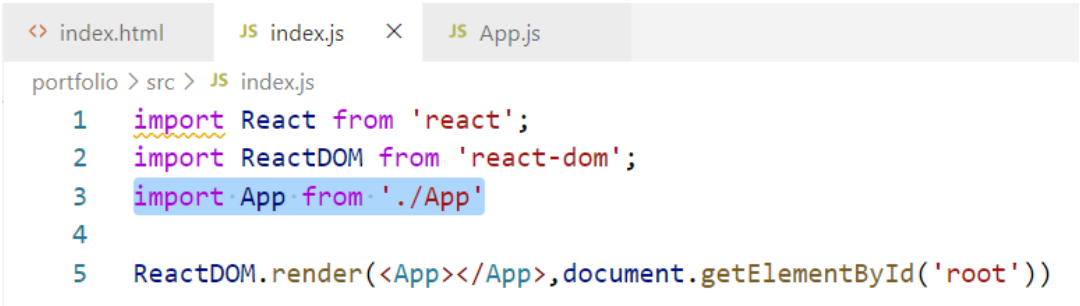
```
ReactDOM.render(<h1>Code is working</h1>,document.getElementById('root'))
```

Let's extend out application by our first component – being the main component of the project. Add to the src directory a new file App.js seing that the component is working:

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

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

export default App;
```

Export default App is needed to share the component.

Now we can render our new component. Let's modify index.js file and render App component rather than an h1 header directly:



We can also write it as: ReactDOM.render**(<App />,**document.getElementById('root')).

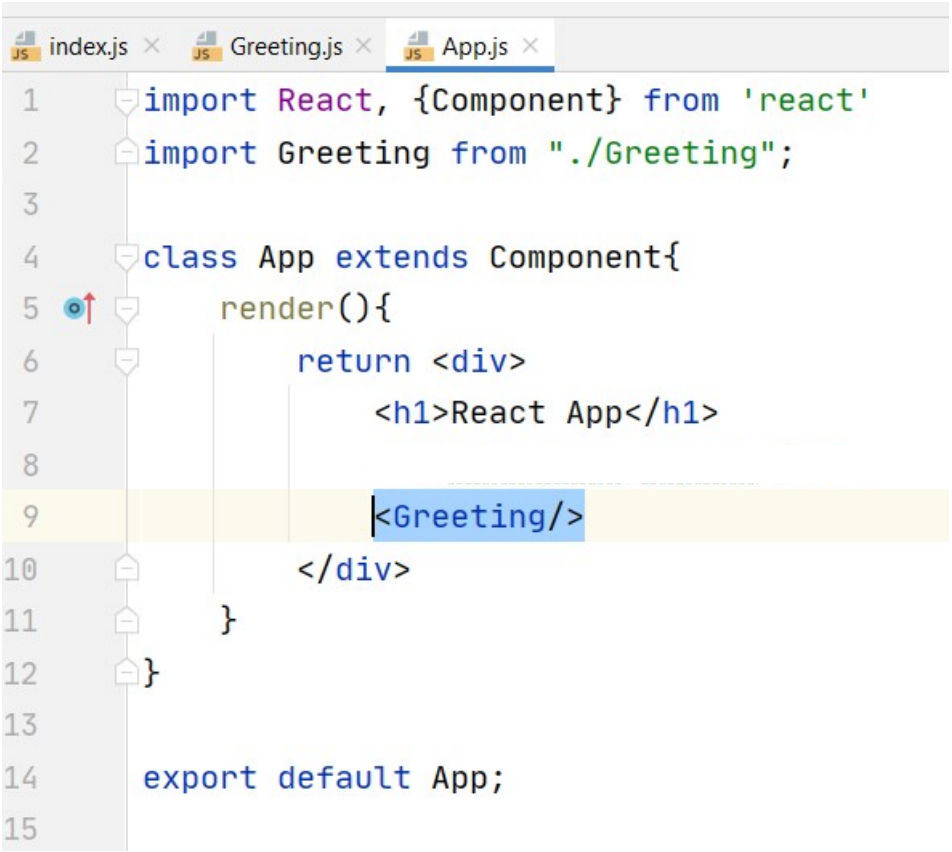
Let's create now a new component Greeting, similar to the one created last week. Create a new file greeting.js with the source code:

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

class Greeting extends Component {
    render(){
        var headline = 'Greeting component from the last class'
        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;
```

Now we should place it in the main App component by adding

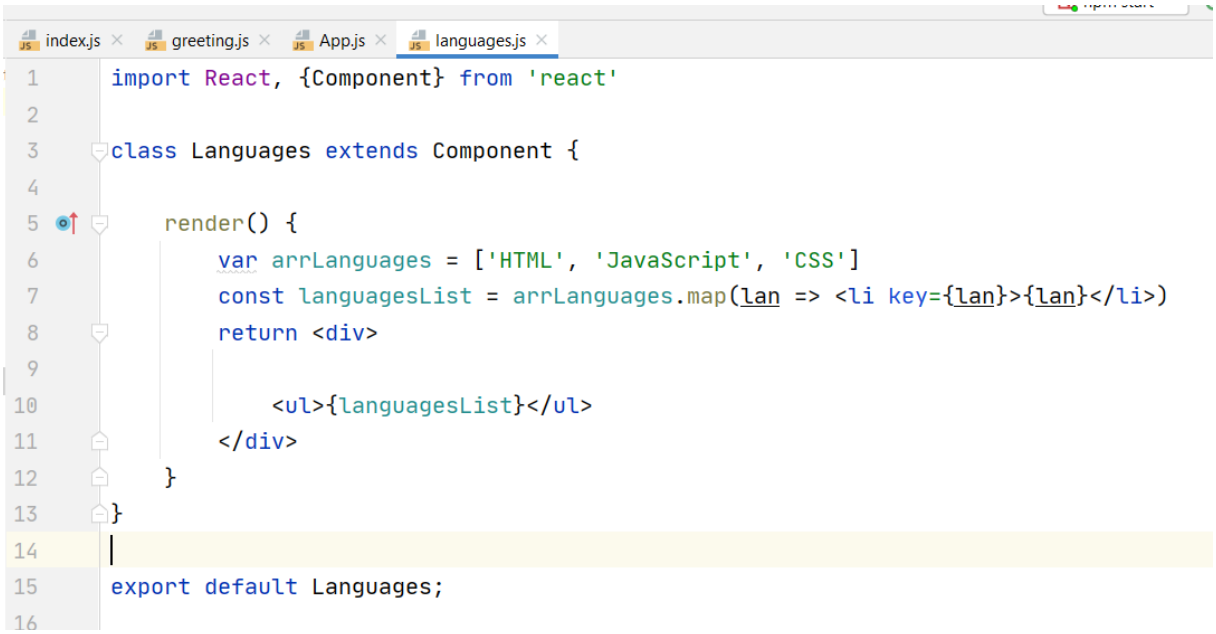
<Greeting />

in return section of the App.js file.



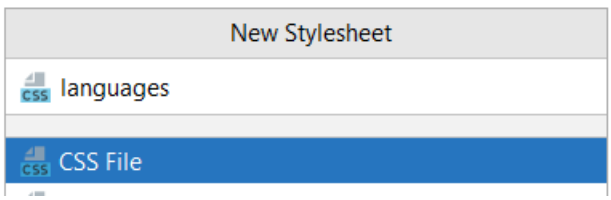
Now let's create another compoent similar to the one from the last class – Language component. Add a new file languages.js with the source code:

```
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>Languages we should know:</h3>
            <ul>{languagesList}</ul>
        </div>
    }
}

export default Languages;
```

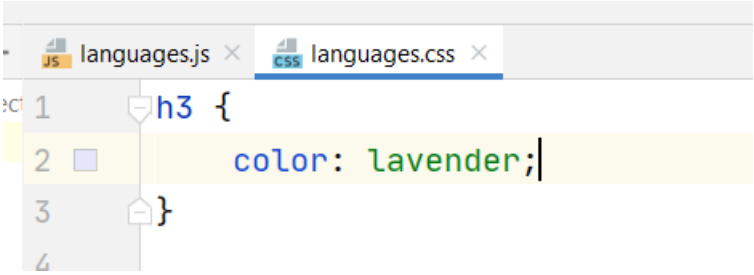


To see it on the website, place it in App component below the Greeting component.

How can we style components? Let's change e.g. the style of h3 header of the component Languages. Add to the src directory a new directory: css. Then, add to it a new css file:



Now we can add some styles, e.g.:



To apply styles, we have to import the css file to the component in which we want to use it:

EXERCISES:

1. Add to the project a component that displays the subject name, link to our university website and link to the official react site.
2. Add component displaying information about the time user visited the site. Place it as the first component – on the top of the page.
   Hint: use `new Date().toLocaleTimeString()`
3. Below the Languages component, display the list of programming languages you have learn so far using a new component.
4. Create a component with a button „More info”. For now, the button does not do any actions.