Рад са догађајима у React-у

У React-у можемо користити опште познате HTML догађаје као што су click, change, mouseover итд . Списак догађаја се може наћи на линку https://www.w3schools.com/tags/ref_eventattributes.asp

Међутим, исто као и у JavaScript-у, у React-у се користи тзв. camelCase синтакса, па се нпр. уместо `onclick` користи `onClick`.

Још једна од разлика јесте та што се React-ови Event handler-и пишу унутар витичастих заграда. Тј. уколико неку на догађј клика желимо да покренемо фунцију *pozdrav()*, уместо *onClick="pozdrav()"* пишемо *onClick={pozdrav}*.

Креирајмо апликацију у gitBash-у у којој ћемо поставити дугме. На клик дугмета ће се покренути функција ће кроз алерт исписаи поруку дана.

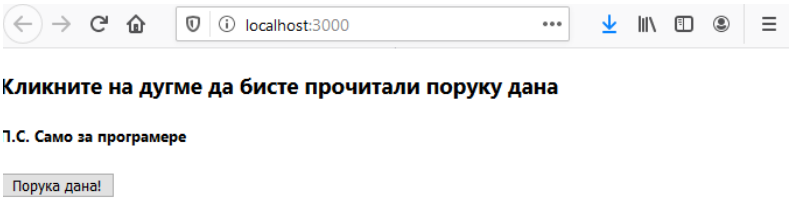
Index.html:

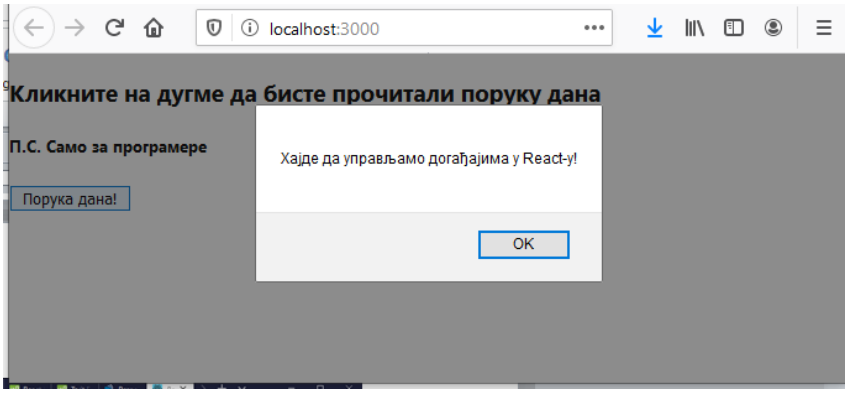
```
<!DOCTYPE html><html lang="en"> <head> <meta charset="utf-8" />
  <title>Догађаји у React-у</title></head> <body>
   <div id="polje"></div>
    </body></html>
```

Index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';



function pozdrav() {
```

```
  alert("Хајде да управљамо догађајима у React-у!");
}
var dugme = (
<div>
        <h3>Кликните на дугме да бисте прочитали поруку дана</h3>
        <h5>П.С. Само за програмере</h5>
        <button onClick={pozdrav}>Порука дана!</button>
</div>
);
ReactDOM.render(dugme, document.getElementById('polje'));
```





Уобичајена пракса јесте да се Event Handler угради у класу унутар методе, као у примеру који следи.

Index.js:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

class PorukaDana extends React.Component {
  poruka() {
    alert("Хајде да управљамо догађајима у React-у!");
  }
  render() {
    return (
          <div>
          <h3>Кликните на дугме да бисте прочитали поруку дана</h3>
          <h5>П.С. Само за програмере</h5>
    <button onClick={this.poruka}>Порука дана</button>
          </div>
    );
  }
}
ReactDOM.render(<PorukaDana />, document.getElementById('polje'));
```

Резултат је наравно исти као и у претходном примеру.

React() је погодан за примену тзв. *Arrow* функција (за детаљна објашњења поновити градиво из Напредних метода програмирања). Тако да исти овај код може и да се поједностави.

```
class PorukaDana extends React.Component {

  poruka=()=>alert("Хајде да управљамо догађајима у React-у!");

  render() {

    return (

          <div>

          <h3>Кликните на дугме да бисте прочитали поруку дана</h3>

          <h5>П.С. Само за програмере</h5>

    <button onClick={this.poruka}>Порука дана</button>

          </div>

    );

  }

}
```

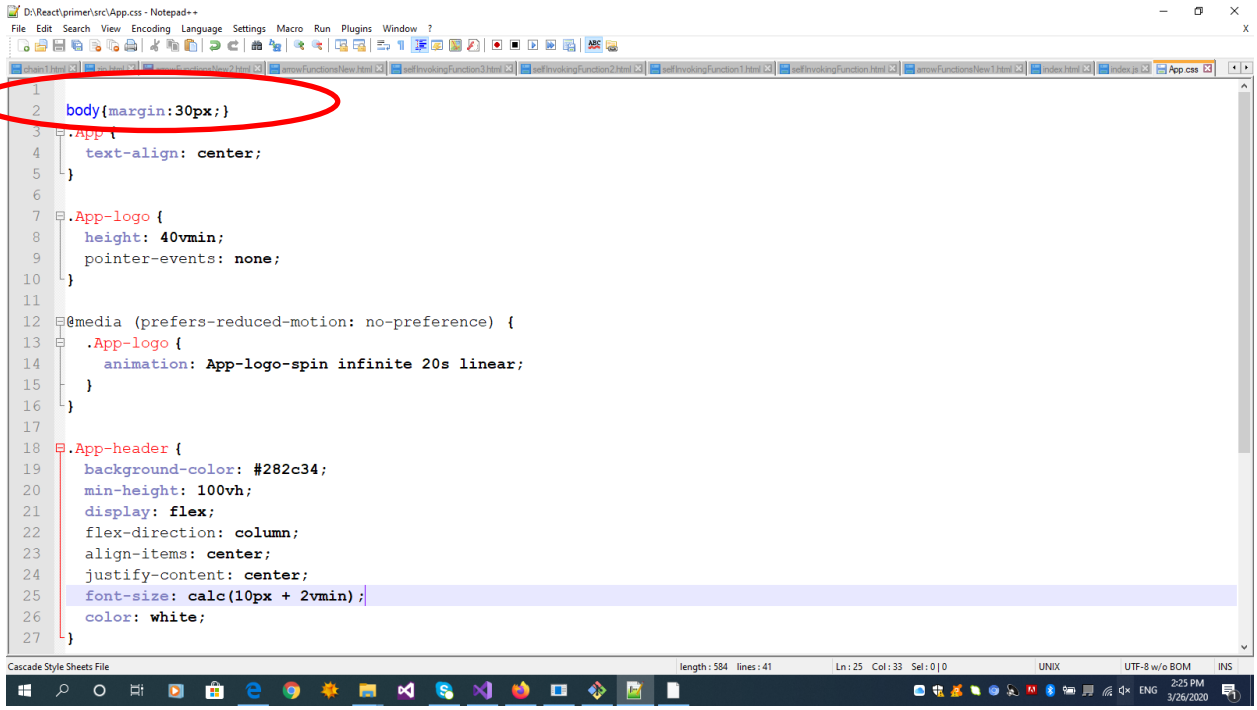
Поновимо још једном чињеницу коју смо већ научили из напредних метода програмирања, да код arrow функција кључна реч `this` увек представља објекат који је дефинисао arrow функцију.

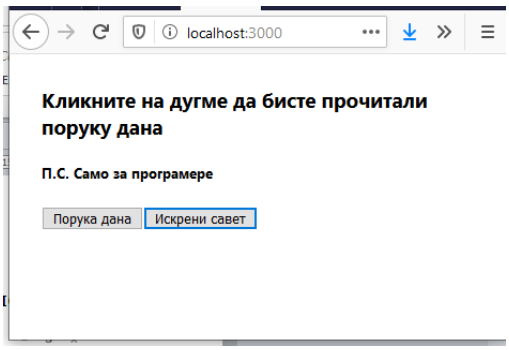
Користећи arrow функције можемо пролсеђивати и параметре:

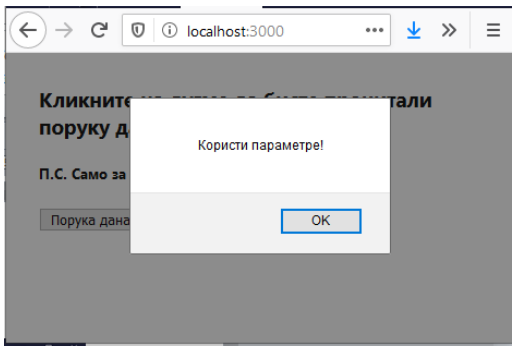
```
class PorukaDana extends React.Component {

  poruka=x=>alert(x);

  render() {

    return (

          <div>

          <h3>Кликните на дугме да бисте прочитали поруку дана</h3>

          <h5>П.С. Само за програмере</h5>

    <button onClick={()=>this.poruka("Хајде да управљамо догађајима у React-у!")}>Порука дана</button>
```

```
<button onClick={()=>this.poruka("Користи параметре!")}>Искрени савет</button>
        </div>
    );
  }
}
```

Додајмо и маргине на тело странице у CSS фајлу апликације, да би све лепше изгледало







Уколико не коритимо arrow функције, већ регуларне, функције (присетимо се напредних метода програмирања, кључна реч this ће се одноити на објекат који је позвао функцију, а нама треба објекат који је креирао функцију (власник функције). Ово можемо исправити коришћењем методе bind(). Илустрације ради, модификоваћемо претхосни пример, тако што ћемо уместо arrow функције користити регуларну функцију.

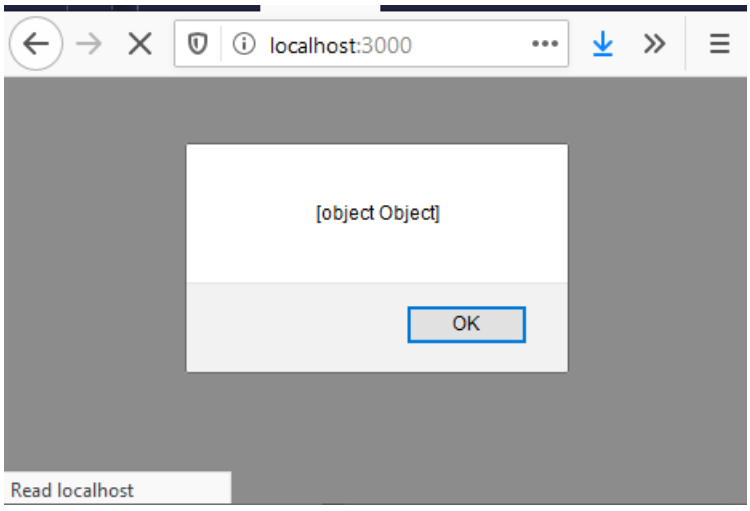
```
class PorukaDana extends React.Component {

    poruka(x){

            alert(x);

            }

 render() {

   return (

        <div>

        <h3>Кликните на дугме да бисте прочитали поруку дана</h3>

        <h5>П.С. Само за програмере</h5>

   <button onClick={this.poruka.bind(this,"Хајде да управљамо догађајима у React-y!")}>Порука дана</button>

        <button onClick={this.poruka.bind(this,"Користи параметре!")}>Искрени савет</button>

        </div>

   );

 }
```

Уколико не користимо методу bind, тј. користимо this.poruka.(this,"Користи параметре!")} уместо this.poruka.bind(this,"Користи параметре!")} , тада ће се функција poruka() покренути приликом учитавања странице уместо кликом на дугме. У примеру испод ћемо ово и проверити.

```
class PorukaDana extends React.Component {

    poruka(x){
```

```
        alert(x);
        }
 render() {
  return (
        <div>
        <h3>Кликните на дугме да бисте прочитали поруку дана</h3>
        <h5>П.С. Само за програмере</h5>
  <button onClick={this.poruka(this,"Хајде да управљамо догађајима у React-у!")}>Порука дана</button>
        <button onClick={this.poruka(this,"Користи параметре!")}>Искрени савет</button>
        </div>
  );
 }
```

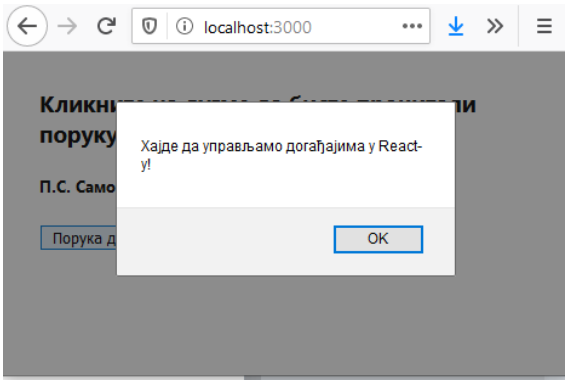
Приликом учитавања добијамо:

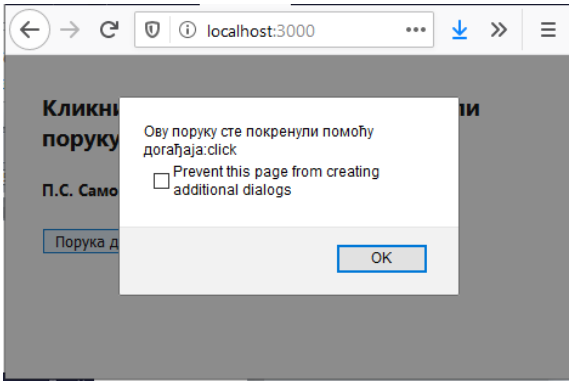


Док се кликом на дугмад не дешава ништа.

Event handler-и имају приступ догађајима у React-у који су покренули функцију. У нашем случају, у питању је догађај клика на дугме. Посматрајмо следећи пример:

```
class PorukaDana extends React.Component {

     poruka(x,y){

          alert(x);

          alert("Ову поруку сте покренули помоћу догађаја:"+y.type);//ovde b predstavlja dogadjaj, tj. click

          }

  render() {

    return (

          <div>

          <h3>Кликните на дугме да бисте прочитали поруку дана</h3>

          <h5>П.С. Само за програмере</h5>

    <button onClick={this.poruka.bind(this,"Хајде да управљамо догађајима у React-у!")}>Порука дана</button>

          <button onClick={this.poruka.bind(this,"Користи параметре!")}>Искрени савет</button>

          </div>

    );

  }
```





Исто можемо урадити користећи arrow функцију:

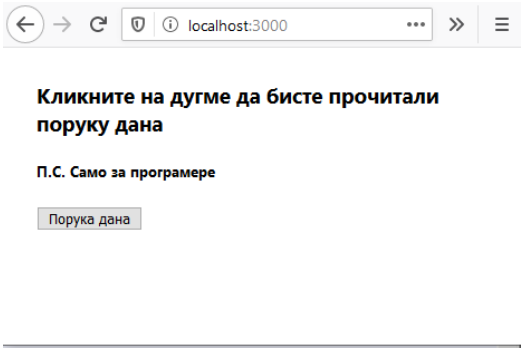
```
class PorukaDana extends React.Component {

    poruka(x,y){

        alert(x);

        alert("Ову поруку сте покренули помоћу догађаја:"+y.type);//ovde b predstavlja dogadjaj, tj. click

        }

  render() {

    return (

        <div>

        <h3>Кликните на дугме да бисте прочитали поруку дана</h3>

        <h5>П.С. Само за програмере</h5>

    <button onClick={(ev)=>("Хајде да управљамо догађајима у React-у!",ev)}>Порука дана</button>

        <button onClick={(ev)=>("Користи параметре!",ev)}>Искрени савет</button>

        </div>

    );

  }

}



class PorukaDana extends React.Component {

    poruka(x,y){

        alert(x);

        alert("Ову поруку сте покренули помоћу догађаја:"+y.type);//ovde b predstavlja dogadjaj, tj. click

        }
```

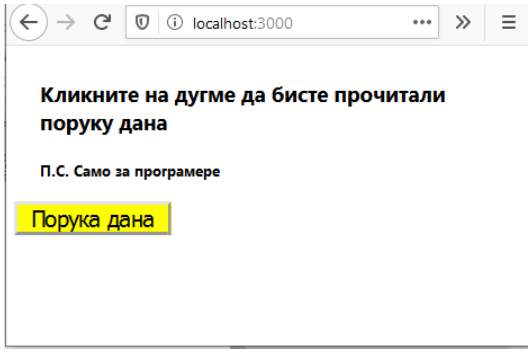
```
  render() {
    return (
          <div>
          <h3>Кликните на дугме да бисте прочитали поруку дана</h3>
          <h5>П.С. Само за програмере</h5>
    <button onClick={(ev)=>this.poruka("Хајде да управљамо догађајима у React-y!",ev)}>Порука дана</button>
          <button onClick={(ev)=>this.poruka("Користи параметре!",ev)}>Искрени савет</button>
          </div>
    );
  }
}
```

У примеру који следи ћемо променити боју позадине дугмета и његову величину прилоком преласка мишем преко њега.

```
class PorukaDana extends React.Component {
      poruka(x){
    var dest=document.getElementById("btn");
     dest.style.background="yellow";
     dest.style.transform="scale(1.5)";
            }

  render() {
```

```
    return (
            <div>
            <h3>Кликните на дугме да бисте прочитали поруку дана</h3>
            <h5>П.С. Само за програмере</h5>
    <button id="btn" onMouseOver={this.poruka.bind(this,this)}>Порука дана</button>
</div>
    );
  }
}

ReactDOM.render(<PorukaDana />, document.getElementById('polje'));
```





Рад са формама

Елементе форми додајемо на исти начин као и било који други XTMЛ садржај.

```
class Forme extends React.Component {
  render() {
    return (
      <form>
```

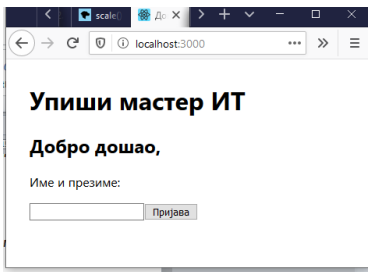
```
        <h1>Упиши мастер ИТ</h1>
        <p>Име и презиме:</p>
        <input
          type="text"
        />
                        <button>Пријава</button>
      </form>
    );
  }
}
ReactDOM.render(<Forme />, document.getElementById('polje'));
```

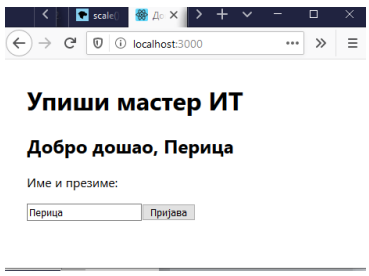
Обрада података из форме

Када обрађујемо податке унете у форму помоћу компоненти, подаци се заправо чувају у компоненти `state`. Променама можемо управљати додвајући ивент хендлер на `onChange` атрибут. Посматрајмо пример који следи:

```
class Forme extends React.Component {
constructor(props) {
   super(props);
   this.state = { student: ' '};
 }
  hendler = (event) => {
   this.setState({student: event.target.value});      /*дефинисали смо arrow функцију - хендлер
```

```
  }


  render() {
    return (
      <form>
        <h1>Упиши мастер ИТ</h1>
                    <h2>Добро дошао, {this.state.student}</h2>   /*позивамо податак који чувамо у стању*/
        <p>Име и презиме:</p>
        <input  type="text" onChange={this.hendler}   />     /*покрећемо хендлер на догађај промене*/
                    <button>Пријава</button>
      </form>
    );
  }
}

ReactDOM.render(<Forme />, document.getElementById('polje'));
```





Наставиће се..... на следећим предавањима