# петък, 30.06.2017

**Events & Forms**

Синтаксисът тук е подобен на този в html , само че е **camelCase**, тоест **onclick** в **html** става **onClick** в **react** Всички javascript-ски евънти(събития) , които познаваме съществуват и тук , но се пишат в **camelCase**

Обикновено за да използваме дадена функция, която искаме да закачим към даден **event**(събитие), ние прикачваме метод от компонента. *Това е най-лесният вариянт , когато **JSX**-а (**html** кодът) се намира в същия компонент.*

**КОГАТО ИЗПОЛЗВАМЕ КЛАСОВЕ, ЗАДЪЛЖИТЕЛНО ТРЯБВА ДА БАЙНДНЕМ(bind-нем) this-а. ИНАЧЕ НЯМА ДА РАБОТИ**

Даден метод може да се bind-не или при прикачане на евента , примерно **<h1 onClick={this.headerClick.bind(this)}>Click me!</h1>** или в конструктора както е показано в снимката по-долу.

ndex.js JS App.js

```
import React, { Component } from 'react'
import './App.css'

class App extends Component {
  constructor (props) {
    super(props)

    this.headerClick = this.headerClick.bind(this)
  }

  headerClick (event) {
    window.alert('CLICKED')
    console.log(event)
    console.log(event.target)
  }

  render () {
    return (
      <div className='App'>
        <h1 onClick={this.headerClick.bind(this)}>Click me!</h1>
      </div>
    )
  }
}

export default App
```

**Няма значение кой от двата варианта се прави.. важното е кодът да е консистентен, тоест навсякъде да се прави само по единия начин.**

Ако имаме **<input type='text' />** ,// поведението по подразбиране(по default) на реакт е , че по никакъв начин не можем да хванем нещата за този **input**// . *Тоест нищо не можем да правим с него*

**Тук задължително трябва да направим onChange, за да прихванем стойностите.**

**event.target е самият обект, върху който е кликнато.**

Как можем да вземем информация кой точно **input** се е сменил ?

let field = target.**name** -> *С това вземаме кой точно input ни се е променил*
let value = target.**value** -> *С това вземаме неговата стойност*

---

Друго което се използва често е да подаваме към някакви **child** елементи някаква информация от сорта на **events**(**събития**)

```
class Child extends Component {
  render () {
    return (
      <div>
        <a onClick={this.props.clickMe}>Click me</a>
        <input type='text' name='someInput' onChange={this.props.changeMe} />
      </div>
    )
  }
}
```

Примерно както е показано на горната картинка, имаме някакъв child component , който има input , name и някакви други неща, НО не той решава каква да е логиката.
На следващата снимка е показано как в child елементи чрез props се подават функции, които обработват събития

```
class App extends Component {
  constructor (props) {
    super(props)

    this.headerClick = this.headerClick.bind(this)
    this.inputChange = this.inputChange.bind(this)
  }

  headerClick (event) {
    window.alert('CLICKED')
    console.log(event)
    console.log(event.target)
  }

  inputChange (event) {
    const target = event.target
    let field = target.name
    let value = target.value

    console.log(`${field} - ${value}`)
  }

  render () {
    return (
      <div className='App'>
        <Child headerClick={this.headerClick.bind(this)} inputChange={this.inputChange.bind(this)} />
      </div>
    )
  }
}

export default App
```

---

**Forms (Форми)**

Формите са си нормална форма,като единственото нещо, което трябва да се добави е *какво да прави при **onChange** и при събмит на формата.*

Тоест на всеки **<input />** в нашата форма трябва да закачим **onChange** event

**НАЙ-ДОБРИ ПРАКТИКИ**

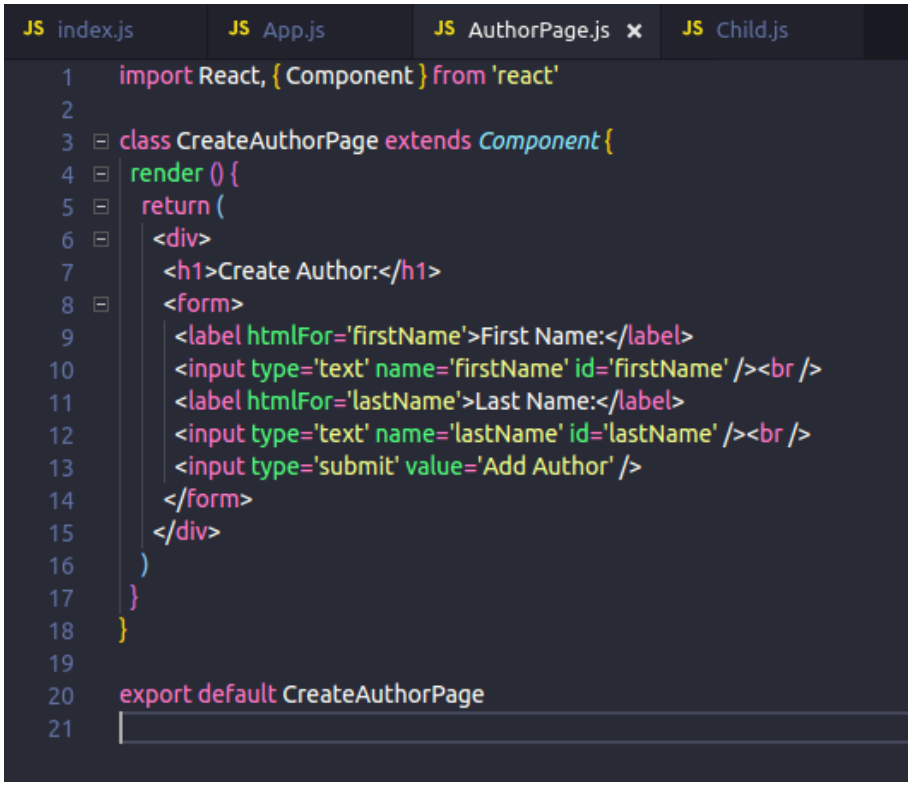
> - ***винаги формата като форма е отделен компонент***

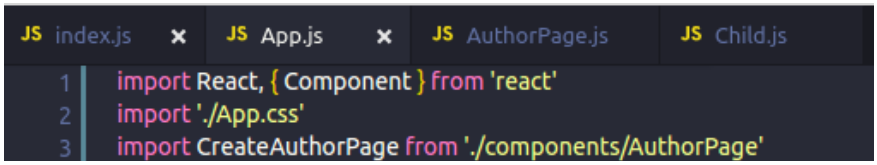
> - ***опитваме се и най-елементарните input-и да ги извадим ако започнат да*** *се повтарят.*

> - ***винаги искаме да имаме client-side validation. Какво означава това ? - да се*** *валидира примерно някакъв input преди да бъде пратен на сървъра. Този вид заявки предотвратява излишни заявки към сървъра.*

> - ***друго нещо, което е добра практика да се прави е да имаме*** *преизползвами(reusable) компоненти(components) дори и за много елементарни неща като <input />*

Две неща са важни за формата.

> 
> 

> 
> 

```
class App extends Component {
  render () {
    return (
      <div className='App'>
        <CreateAuthorPage />
      </div>
    )
  }
}

export default App
```

**Две неща са важни за формата:**

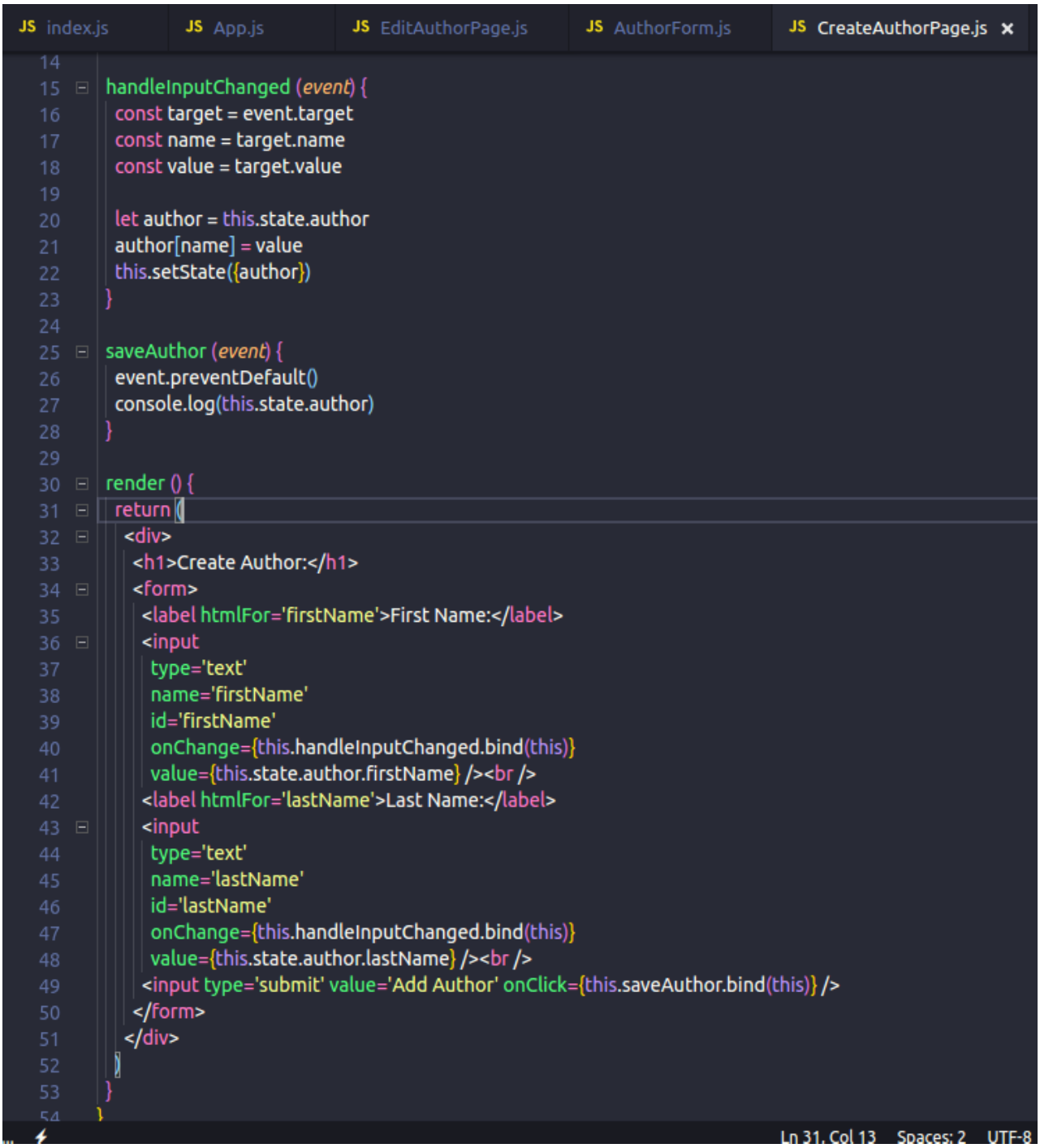
- ***формичката да я опишем, нагласим както е показано на следващата снимка:***

index.js | App.js | EditAuthorPage.js | AuthorForm.js | CreateAuthorPage.js

```
saveAuthor (event) {
  event.preventDefault()
  console.log(this.state.author)
}

render () {
  return (
    <div>
      <h1>Create Author:</h1>
      <form>
        <label htmlFor='firstName'>First Name:</label>
        <input
          type='text'
          name='firstName'
          id='firstName'
          onChange={this.handleInputChanged.bind(this)}
          value={this.state.author.firstName} /><br />
        <label htmlFor='lastName'>Last Name:</label>
        <input
          type='text'
          name='lastName'
          id='lastName'
          onChange={this.handleInputChanged.bind(this)}
          value={this.state.author.lastName} /><br />
        <input type='submit' value='Add Author' onClick={this.saveAuthor.bind(this)} />
      </form>
    </div>
  )
}
}
```

- ***След това да закачим onChange event(в случая се нарича*** *handleInputChange) и save event. Показано е на следващата картинка*


index.js
App.js
EditAuthorPage.js
AuthorForm.js
CreateAuthorPage.js
handleInputChanged (event) {
const target = event.target
const name = target.name
const value = target.value
let author = this.state.author
author[name] = value
this.setState({author})
}
saveAuthor (event) {
event.preventDefault()
console.log(this.state.author)
}
render () {
return (

<h1>Create Author:</h1>
<form>
<label htmlFor='firstName'>First Name:</label>
<input
type='text'
name='firstName'
id='firstName'
onChange={this.handleInputChanged.bind(this)}
value={this.state.author.firstName} /><br />
<label htmlFor='lastName'>Last Name:</label>
<input
type='text'
name='lastName'
id='lastName'
onChange={this.handleInputChanged.bind(this)}
value={this.state.author.lastName} /><br />
<input type='submit' value='Add Author' onClick={this.saveAuthor.bind(this)} />
</form>

)
}
}
Ln 31, Col 13
Spaces: 2
UTF-8