

сряда, 5.07.2017

Записки от Workshopa за ReactJS

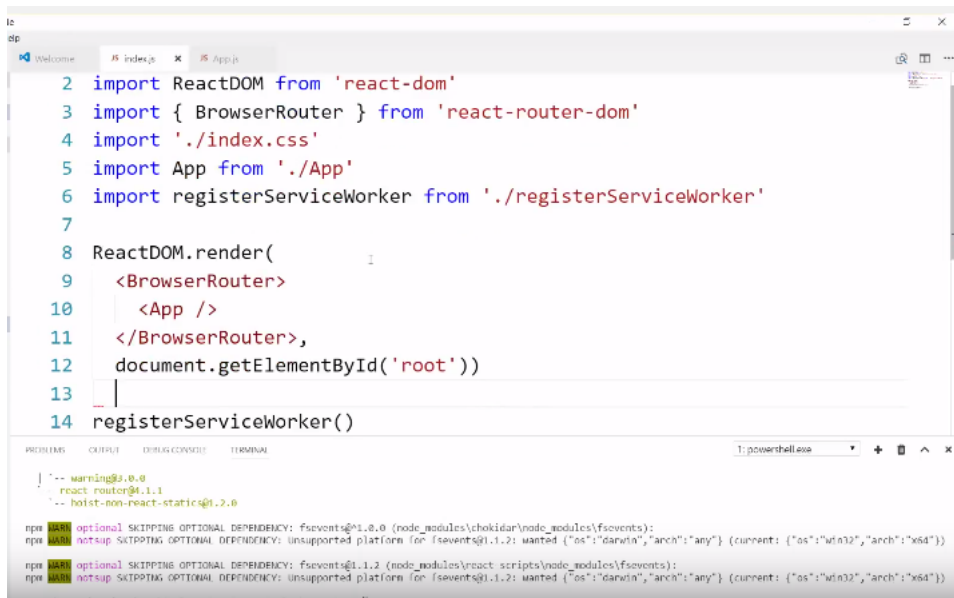
Създаваме си приложението с **create-react-app** име на приложението и след това:

Създаваме си необходимите папки. - actions, stores, components, data ако се ползва flux

След това трябва да си направим **ауентикацията, менюто и раутите(routes).**

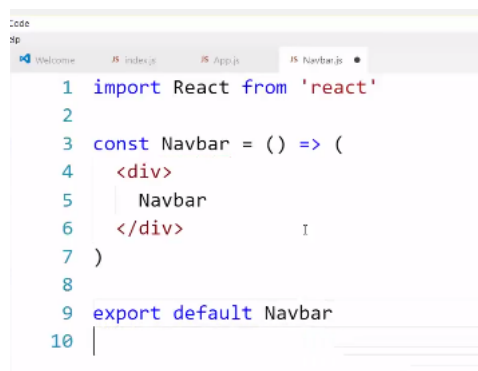
`npm install react-router-dom --save --save-exact`

След това си **импортираме BrowserRouter в index.js** както е показано на следващата картинка:



```
1 import ReactDOM from 'react-dom'
2 import { BrowserRouter } from 'react-router-dom'
3 import './index.css'
4 import App from './App'
5 import registerServiceWorker from './registerServiceWorker'
6
7 ReactDOM.render(
8   <BrowserRouter>
9     <App />
10   </BrowserRouter>,
11   document.getElementById('root'))
12
13 registerServiceWorker()
```

След това ни трябва съответно компонента **Navbar** за навигацията и самите **routes(раутове, адреси)**. Показан е на следващата картинка:



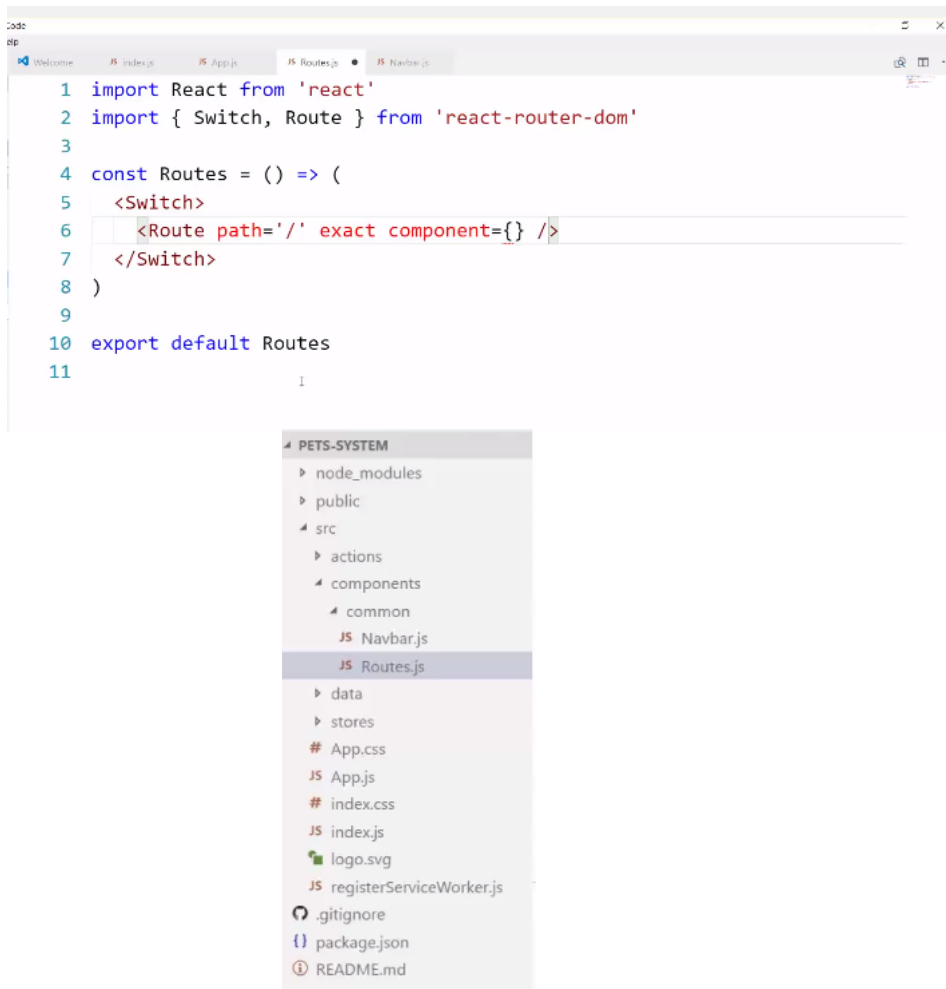
```
1 import React from 'react'
2
3 const Navbar = () => (
4   <div>
5     Navbar
6   </div>
7 )
8
9 export default Navbar
```

Отиваме в **app.js** и го импортираме. И това действие е показано на следващата снимка:

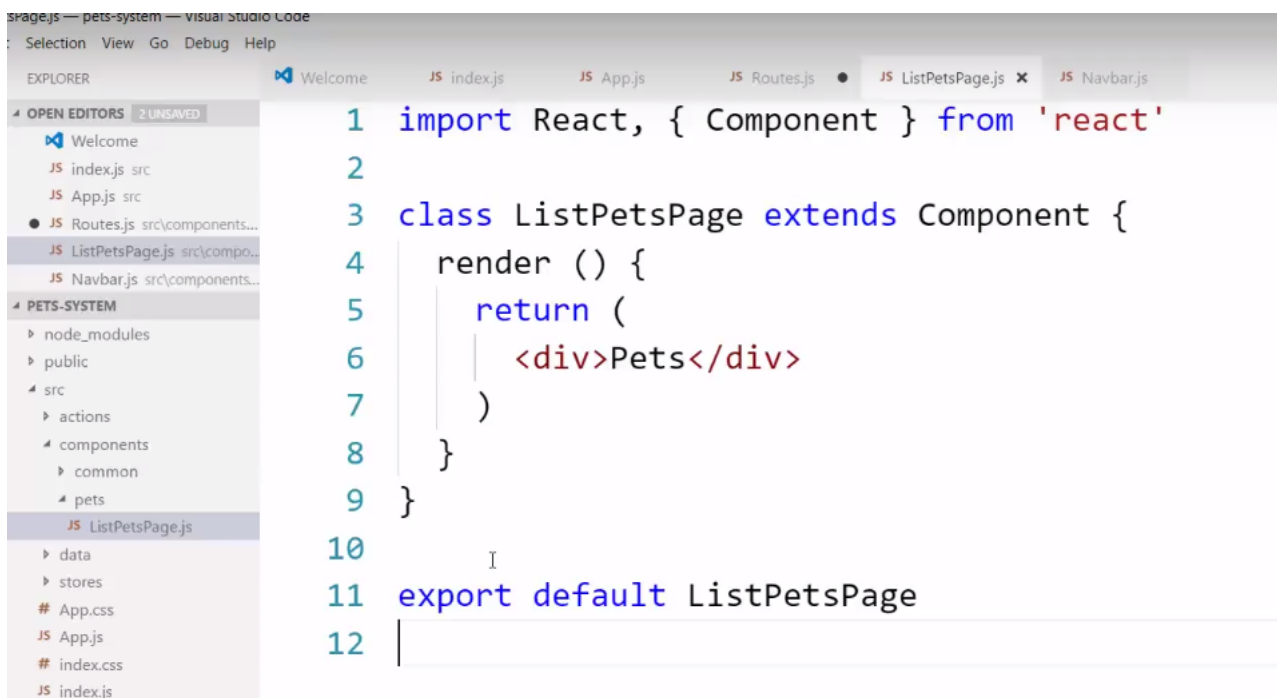


```
1 import React, { Component } from 'react'
2 import Navbar from './components/common/Navbar'
3 import Routes from './components/common/Routes'
4 import './App.css'
5
6 class App extends Component {
7   render () {
8     return (
9       <div className='App'>
10         <Navbar />
11         <Routes />
12       </div>
13     )
14   }
15 }
```

Следващото нещо е да се направят routes(раутовете)



След това си правим **homePage-а**, която в този случай е **ListPetsPage**, където ще се листват животните и, където ще е страницането. По-късно ще се довърши този компонент

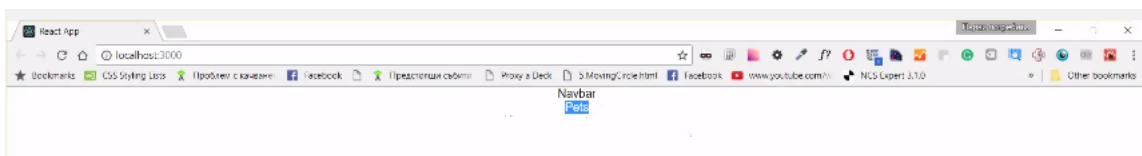


След това трябва да вържм новосъздаденият компонент в раута(route)



```
1 import { Switch, Route } from 'react-router-dom'
2 import ListPetsPage from '../pets/ListPetsPage'
3
4 const Routes = () => (
5   <Switch>
6     <Route path="/" exact component={ListPetsPage} />
7   </Switch>
8 )
9
10 export default Routes
```

Следващата стъпка е да стартираме **сервъра**, за да видим дали не сме изпуснали нещо. Ако всичко е наред дотук.. резултатът би трябвало да е като на следващата снимка:



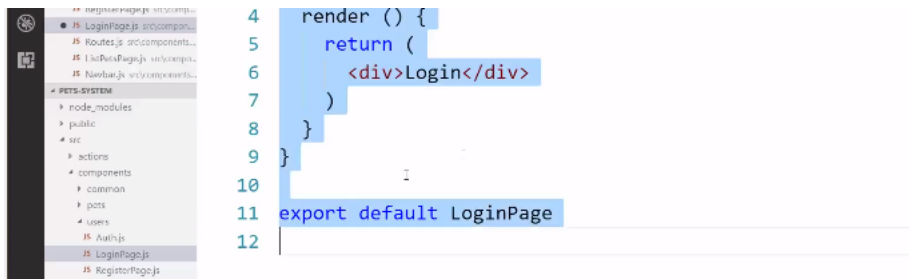
Следващото, което ни трябва са **потребителите**(users):
- Първото, което имаме за тях е т.нар. **Auth**

```
1 class Auth {
2   static authenticateUser (token) {
3     window.localStorage.setItem('token', token)
4   }
5
6   static isAuthenticated () {
7     return window.localStorage.getItem('token') !== null
8   }
9
10  static deauthenticateUser () {
11    window.localStorage.removeItem('token')
12  }
```

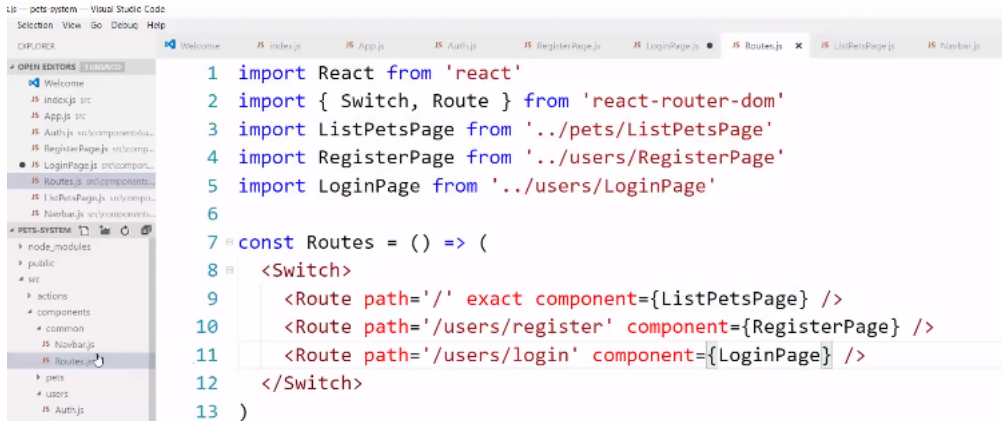
След това ни трябва нови две странички - **RegisterPage** и **LoginPage**

```
1 import React, { Component } from 'react'
2
3 class RegisterPage extends Component {
4   render () {
5     return (
6       <div>Register</div>
7     )
8   }
9 }
10
11 export default RegisterPage
```

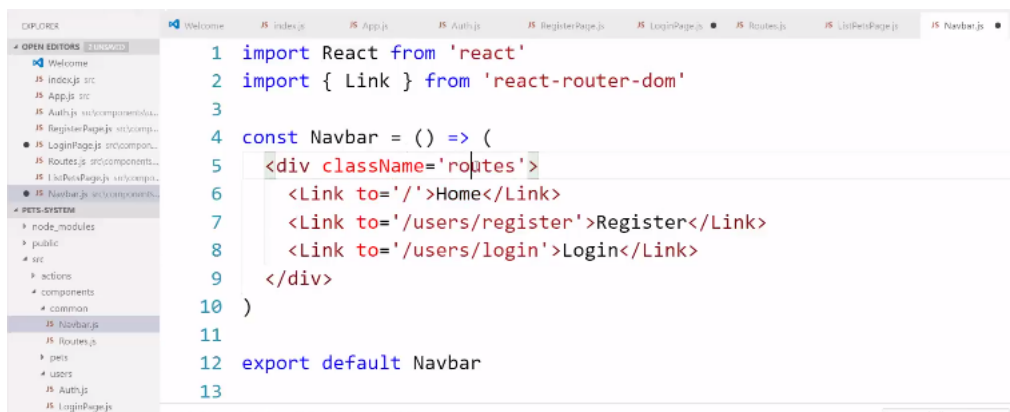
```
1 import React, { Component } from 'react'
2
3 class LoginPage extends Component {
```



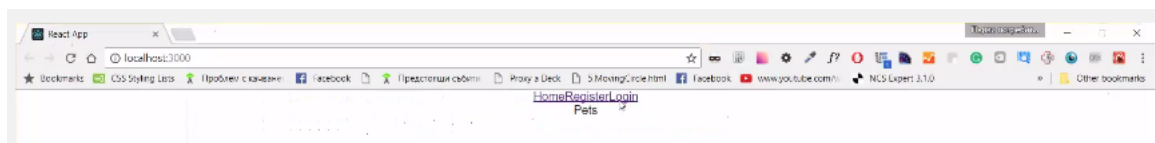
След това трябва отново да си ги добавим в *раутовете(routes)*



След това е *Navbar*-а трябва да започнем да добавяме някакви линкове. И понеже изглеждат доста грозно, затова е добавен *css* класът *routes*



Ако няма грешки и всичко е наред би трябвало да се вижда по следния начин:



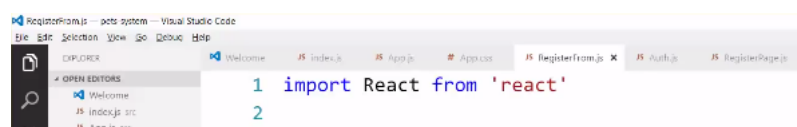
Както се вижда от картинката, линковете са доста грозни и затова им сложихме *css* class *routes* или *мени* и сега отиваме във файл *App.css*, за да добавим показаното на картинката по-долу стилизиращо правило.


```

21 .menu a {
22   display: inline-block;
23   padding-left: 5px;
24   padding-right: 5px;
25 }

```

Следващата стъпка е да си създадем формата за регистрация на потребители. Отиваме в папка *users* и създаваме нов файл **RegisterForm.js**





```

1 import React from 'react'
2
3 const RegisterForm = (props) => (
4   <form>
5     <div>{props.error}</div>
6     <label htmlFor='email'>E-mail</label>
7     <input
8       type='email'
9       name='email'
10      placeholder='E-mail'
11      value={props.user.email}
12      onChange={props.onChange} />
13   <br />

```

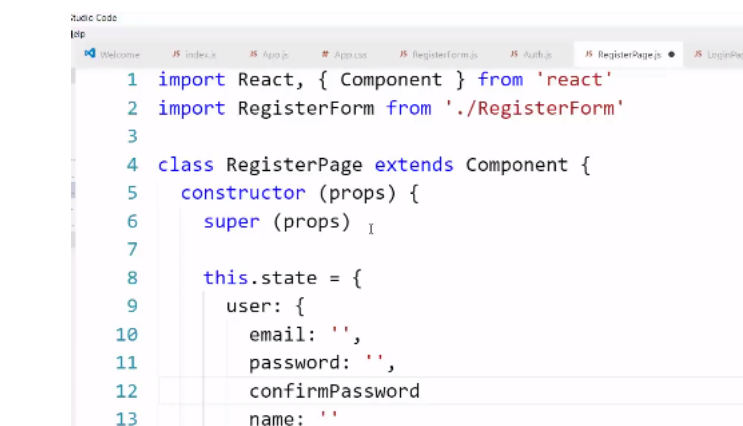
След като е готова формата, се връщаме в **RegisterPage** компонента, за да я добавим и след това да и подаваме необходимите **props**. На картинките не е целия код, но е ориентиран. Като цяло гледаме от **RegisterForm** компонента, какво очаква да получи от родителя и след това, когато добавяме формата компонент в бащиния компонент, който е в случая **RegisterPage**, и му подаваме от него необходимата информация. В по-долната снимка такава е **user** и **onChange**, но не е цялата. Трябва след **onChange** да се добави още - **error = {this.error}** и **onSave** метод. **Да не се забравя да се създадат методи, които да обработват onChange и onSave**. На следващата картинка е показан метод за обработка на **onChange** метод.

```

18 handleUserChange (event) {
19   const target = event.target
20   const field = target.name
21   const value = target.value
22
23   const user = this.state.user
24   user[field] = value
25   this.setState({ user })
26 }

```

VS Code tooltip for `setState` shows: `setState(state: any, callback?: () => any): void`. Below the code, there are checkboxes for `user` and `handleUserChange`.



```

1 import React, { Component } from 'react'
2 import RegisterForm from './RegisterForm'
3
4 class RegisterPage extends Component {
5   constructor (props) {
6     super (props)
7
8     this.state = {
9       user: {
10         email: '',
11         password: '',
12         confirmPassword: '',
13         name: ''

```

```

19   return (
20     <div>
21       <h1>Register User</h1>
22       <RegisterForm
23         user={this.state.user}
24         onChange={this.handleUserChange.bind(this)} />
25     </div>
26   )

```

```

27 }
28 }
29
30 export default RegisterPage
31

```

При условие, че всичко е наред и няма грешки би трябвало да се вижда тази форма в браузъра. На този етап бутонът submit все още не работи.

3 апочваме да навързваме flux.

- Като начало трябва да инсталираме flux -> `npm install flux --save --save-exact`

- След това трябва да си създадем един dispatcher

The screenshot shows the Visual Studio Code interface with the 'pets-system' project open. The Explorer sidebar on the left shows the file structure, including 'src' and 'actions' folders. The main editor window shows the 'dispatcher.js' file being created with the following code:

```

1 import { Dispatcher } from 'flux'
2
3 export default new Dispatcher()
4

```

-Следващото нещо е да си създадем **actions** в папка **actions**. Създаваме action **UserActions.js**

The screenshot shows the Visual Studio Code interface with the 'pets-system' project open. The Explorer sidebar on the left shows the file structure, including the 'actions' folder. The main editor window shows the 'UserActions.js' file being created with the following code:

```

3 const userActions = {
4   types: {
5     REGISTER_USER: 'REGISTER_USER'
6   },
7   register (user) {
8     dispatcher.dispatch({
9       type: this.types.REGISTER_USER,
10      user
11    })
12  }
13 }
14
15 export default userActions

```

-Следва да направим **UserStore.js** в папка **Stores**.

The screenshot shows the Visual Studio Code interface with the 'pets-system' project open. The Explorer sidebar on the left shows the file structure, including the 'stores' folder. The main editor window shows the 'UserStore.js' file being created with the following code:

```

1 import { EventEmitter } from 'events'
2 import dispatcher from '../dispatcher'
3
4 class UserStore extends EventEmitter {
5   handleAction (action) {
6
7   }
8 }
9
10 let userStore = new UserStore()
11 dispatcher.register(userStore.handleAction.bind(userStore))
12 export default userStore

```



```
1 import { EventEmitter } from 'events'
2 import dispatcher from '../dispatcher'
3 import userActions from '../actions/UserActions'
4
5 class UserStore extends EventEmitter {
6   handleAction (action) {
7     switch (action.type) {
8       case userActions.types.REGISTER_USER: {
9         this.register(action.user)
10        break
11      }
12      default: break
13    }
14  }
15 }
```

-Следва да се направи register функция, която очевидно трябва да регистрира потребител на сървъра и след това да dispatch-ва event, че user-ът се е регистрирал успешно. Това ще стане като във файл в папка data създадем обекти, които ще се извикват по някакъв начин. Създаваме файл userData.js в data directory.(Показан е на следващата картинка).

```
1 class UserData {
2   static register (user) {
3     window.fetch('http://localhost:5000/auth/signup', {
4       method: 'POST',
5       mode: 'cors',
6       body: JSON.stringify(user),
7       headers: {
8         'Accept': 'application/json',
9         'Content-Type': 'application/json'
10      }
11    })
12  }
13 }
```

```
1 class UserData {
2   static register (user) {
3     return window.fetch('http://localhost:5000/auth/signup', {
4       method: 'POST',
5       mode: 'cors',
6       body: JSON.stringify(user),
7       headers: {
8         'Accept': 'application/json',
9         'Content-Type': 'application/json'
10      }
11    })
12     .then(res => res.json())
13   }
14 }
```

-След това трябва да го използваме в UserStore и затова се връщаме там и го импортираме. Сега методът register в UserStore изглежда по следния начин:

```
6 class UserStore extends EventEmitter {
7   register (user) {
8     UserData
9     .register(user)
10    .then(data => this.emit('user_registered', data))
11  }
12 }
```

Тук просто извикваме методът register и му подаваме обекта user и след това, когато се получат данните емитваме(създаваме) event(събитие), че някакъв потребител се е регистрирал и компонентите, които се интересуват просто се прихващат за него.