Usefull triki

Хендлъра от събмит бутона на логин формата.

--------------------------------------------------------------------------------------------------------

handleSubmit = ev => {

ev.preventDefault();

requester.post(‘user’, ‘login’, ‘basic’, this.state)

.then( res => {

sessionStorage.setItem(‘authtoken’, res.\_ckmd.authtoken);

});

}

--------------------------------------------------------------------------------------------------------

Тук ‘requester’ е наш клас, който сме импортнали, а ‘post’ е негов метод, който ще прави зявката към сървъра.

‘sessionStorage’ е клас на браузера, който използваме директно и на него записваме аутентикиращия токен, който получаваме от сървъра при успешен логин. Чрез този токен указваме на сървъра, че сме ние. Полето се зачиства, когато браузерът бива затворен.

Имаме и клас ‘localStorage’(пак на браузера), което се зачиства само ръчно. - т.е. по-трайно е.

Event driven architecture

------------------------------observer--------------------------------------------

let subscription = {

‘loginUser’ : [ ]

};

export default {

events: {

loginUser: ‘loginUser’

},

subscribe: (eventName, fn) =>

subscription[eventName].push(fn),

trigger: (eventName, data) =>

subscription[eventName].forEach(fn => fn(data))

}

----------------------------------------------------------------------------------

-------------------------------------header-----------------------------------

Export default class Header extends Component {

Constructor(props) {

Super(props);

this.state = { username: null };

Observer.subscribe(observer.events.loginUser, this.userLoggedIn);

}

userLoggedIn = username => this.setState({ username });

...

---------------------------------------------------------------------------------

Използва се така наречения ‘observer’. Това е клас(в случая не е разписан, като клас), който приема функции от някои компоненти(които искат да се абонират за промени), като ги записва в свой регистър – „subscription[eventName].push(fn)”. При настъпване на даден евент, от списъка на ‘observer’-а, се преравя списъка с компоненти, регистрирани за този евент, и се вика подадената от тях функция – „subscription[eventName].forEach(fn => fn(data))”. По този начин се ъпдейтват лейаутите на компоненти нямащи нищо общо един с друг.

Пример:

От логин компонента се клика бутон логин. Приемат се успешно данните от сървара и в хедър компонента се появява надпис „здравей, бай пешо”.

Чрез предварително закачения лисъснър в обсървъра за логин, компонента хедър разбира, че потребителя се е логнал. Понеже хедър и логин компонентите не са в пряка връзка, за да си сътрудничат, а хедър трябва да знае, кога точно да покаже поздрава.

фгфгф

Alt+Shift+F - hot key for code formating

38 min. Session storage autotication

Ivan – 1:18:00 - fetch request

return (

    <div className="App">

      <header className="App-header">

        <img src={logo} className="App-logo" alt="logo" />

        <p>

          Edit <code>src/App.js</code> and save to reload.

        </p>

        <a

          className="App-link"

          href="https://reactjs.org"

          target="\_blank"

          rel="noopener noreferrer"

        >

          Learn React

        </a>

      </header>

      <h1>troll test app</h1>

    </div>

  );

-----------------------------------------------------------------

**Redux**

Actions – обикновен обект описващ случилото се. Всяка промяна е описана, като действие. Накрая за да свържем състоянието и действието пишем функция наречена reducer.

/\*

\* action types

\*/

export const ADD\_TODO = 'ADD\_TODO'

export const TOGGLE\_TODO = 'TOGGLE\_TODO'

export const SET\_VISIBILITY\_FILTER = 'SET\_VISIBILITY\_FILTER'

/\*

\* other constants

\*/

export const VisibilityFilters = {

SHOW\_ALL: 'SHOW\_ALL',

SHOW\_COMPLETED: 'SHOW\_COMPLETED',

SHOW\_ACTIVE: 'SHOW\_ACTIVE'

}

/\*

\* action creators

\*/

export function addTodo(text) {

return { type: ADD\_TODO, text }

}

export function toggleTodo(index) {

return { type: TOGGLE\_TODO, index }

}

export function setVisibilityFilter(filter) {

retur

-------------------------------------------------------------------------------------

Reducers – функция, която приема състоянието и действието, като аргументи връща следващото състояние на приложението. редукторът трябва да е чист. С подадените му аргументи да изчисли следващото състояние и да го върне. Без странични ефекти, без API повиквания, без мутации. Подаваните елементи от state, както и самия той, не трябва да се променят. Прави се тяхно копие, то се променя и накрая се връща промененото вече копие, като next state.

...state оператора „три точки” прави плитко копие на обекти и масиви! ([object spread operator proposal](https://redux.js.org/recipes/using-object-spread-operator))

Ако използваме switch е много важно да използваме default: return state(накрая), за да върнем предишния стейт ако никой от кейсовете не пасне.

Редукторна композиция - [reducer composition](https://redux.js.org/basics/reducers#splitting-reducers)

---------------------------------------------------------------------------------------------

Store – обединява actions и reducers и има следните отговорности:

* Задържа стейта на приложението
* Позволява достъп до стейта, чрез getState()
* Позволява ъпдейтването на стейта, чрез dispatch(action)
* Регистрира лисънър, чрез subscrib(listener)
* Handles unregistering of listeners via the function returned by subscribe(listener).

Има само един Store в приложение на Redux. Когато искате да разделите логиката си за обработка на данни, ще използвате редукторна композиция вместо много Store.

const store = createStore(todoApp)

втория параметър е по избор и уточнява състоянието на клиента да отговаря на състоянието на Redux приложение работещо на сървъра.

Чрез dispatch се викат действията, за да се актуализира state.

Store.dispatch(addTodo(‘lern about actions’))

--------------------------------------------------------------------------------------------

========================================

React

# Using React.createRef()

This is currently the recommended way to use references as of React 16.3 (See [the docs](https://reactjs.org/docs/refs-and-the-dom.html#creating-refs) for more info). If you're using an earlier version then see below regarding callback references.

You'll need to create a new reference in the constructor of your parent component and then assign it to a child via the ref attribute.

class FormEditor extends React.Component {

constructor(props) {

super(props);

this.FieldEditor1 = React.createRef();

}

render() {

return <FieldEditor ref={this.FieldEditor1} />;

}

}

In order to access this kind of ref, you'll need to use:

const currentFieldEditor1 = this.FieldEditor1.current;

This will return an instance of the mounted component so you can then use currentFieldEditor1.state to access the state.

Just a quick note to say that if you use these references on a DOM node instead of a component (e.g. <div ref={this.divRef} />) then this.divRef.current will return the underlying DOM element instead of a component instance.

Въпроси:

1 – за frontend-a, посочените 4 са варианти, които фирмата ги устройва?

2 – използва ли се action атрибута на формата или винаги е чрез отделна функция?

3 – бутон или Инпут за събмит?

4 – трябва ли да се стремя да разбия формата на отделни класове?

5 – редно ли е да се достъпва state на децата? Използват ли се refs?

6 – ОК ли е, да се предават колбеци от родител на дете, за да може родителя да получи данни от действията на детето?

7 – валидациите на бекенда със специална библиотека ли стават или сам си пишеш regular expression?

8 – при код ревю да копираш промис за заявка, както ги правиш ти.

9 – трябва ли да правя отделен брънч за логин и регистер?

10 – частното репо платено и без визуализация на брънчове.

11 – каква мерна еденица използв, px, %, rem, m?

# [Difference between 'export' and 'export default' in JavaScript? [duplicate]](https://stackoverflow.com/questions/42478661/difference-between-export-and-export-default-in-javascript)

// Three different export styles

export foo;

export default foo;

export = foo;

// The three matching import styles

import {foo} from 'blah';

import foo from 'blah';

import \* as foo from 'blah';

Roughly compiles to:

exports.foo = foo;

exports['default'] = foo;

module.exports = foo;

var foo = require('blah').foo;

var foo = require('blah')['default'];

var foo = require('blah');

!!! TO DO !!!

activeClassName="selected"

- in navigation class. Когато се натисне logout и се редиректне към homeменюто сотава немаркирано.

- В менюто bug може да посочиш само 8 игри. Да се добавят останалте в списъка.

- rating table in the home page.

* Урок за redux - http://github.com/viktorpts/react-demos-oct-2017

Express js cursess - <https://softuni.bg/search/results?SearchTerm=express&CourseInstances=true&FastTrackInstances=true&Seminars=true&Questions=true&Users=true&LectureResources=true>

Express js fundamentals - <https://softuni.bg/trainings/2255/expressjs-fundamentals-january-2019#lesson-10708>

<https://softuni.bg/trainings/resources/video/37055/video-14-january-2019-kiril-kirilov-expressjs-fundamentals-january-2019/2255>

<https://softuni.bg/trainings/1798/reactjs-fundamentals-october2017#lesson-7094>

redux - <https://softuni.bg/trainings/resources/video/19274/video-stream-record-07-november-2017-viktor-kostadinov-reactjs-fundamentals-october2017/1798>

redux exersise - <https://softuni.bg/trainings/resources/video/19303/video-screen-08-november-2017-hristian-hristov-reactjs-fundamentals-october2017/1798>

--------------------------------------------------------------------------------------------------------------------------------------

NODE JS

Debugger за node :

* From console(cmd) – node --inspect-brk index - след brk се пише името на файла, от който искаме да започне дебъгването.
* From google chrome browser – chrome://inspect изчакваме малко и като се появи, кликаме Inspect.

И после вече си слагаме break point-a където искаме. Понеже дебъгването ще започне от 1 ред на посочения файл.

NODEMON - следи за рестартиране на сървъра при натискане на ctrl+s

npm install nodemon –g

стартира се от папката където ни е index.js или app.js(както сме избрали да го кръстим). Т.е вместо да пишем постоянно - node index и след това ctrl+c за да прекъснем, просто написваме nodemon index и той започва да следи за промени

!!! - в моя случай трябва package json-a да променя стринга в скрипта за старт да е nodemon, а не node.!!!

Можем да си пишем и сами команди – session and authentication – 2019 – 00:27:27 Уилиам Абат

Стартирането им ще стане с run т.е. ако командата, която сме написали е troll то тогава ще стане npm run troll

РАБОТА С Postman !!!!! - introduktion to expres js – 00:50:00 min.

Заявка от тип ID - /workoutlog/:logId - introduktion to expres js – 01:06:13 min.

Middleware – introduktion to expres js – 01:30:40



Фейсбук използва подобен код за да не позволи достъп до раутовете си, ако потребителя не е логнат. Ако получиш линк към дадена страница, тук с този код то вижда, че не си логнат и те пренасочва към логин страницата. За да не пишеш тази проверка във всеки раут, който използва user/нещо си.

EXPRESS

Сървър от нулата - Architecture and code styling - Уилям Абъд

Правим папка – trollProject и нея викаме cmd

npm init

npm express generator (!!! не е точната команда, да се провери как е)

npm install (инсталира всички пакети, които експрес генератора е добавил в пакич джейсона в депендънси)

npm install –save mongoose

!!! Чист, първоначален APP с express generator - seeion and authentication 00:59:00 !!!

! Sert on environment – in WS code terminal - > $env:NODE\_ENV=”development” !

! repul - in WS code terminal - > node and push enter - все едно си извикал конзолата на браузера(F12) и там можеш да си пишеш разни примерчета за тест

Канектването към базата данни трябва да е само веднъж, а не както е при мен във всеки модел.

seeion and authentication 00:59:50

Session

Leson – seeion and authentication 00:32:34

За да използваме сесии трябва да си добавим един пакет от npm-a

Nmp install –save express-session

Authentication with token

Трябва да си инсталираме jsonwebtoken - npm install –save jsonwebtoken

Създава web token и го разкодира(разчита)

Session and authentication 01:34:30

Bcryptjs -- seeion and authentication 02:03:19

Email-validator -- seeion and authentication 02:19:00