ROADMAP



10. Default опасность

**Export default <name>** может создать путаницу, т.к. при импорте <name> может называться произвольно. Очень важно правильно указать путь, откуда мы импортируем!

При импорте

**import <name> from ‘./folder/file.ts’** важно правильно указать имя и путь!

Импорт компонент можно делать автоматически:

**ALT + Enter** - автоимпорт компонент в WebStorm. При автоимпорте **проверяй path!!!**

11. Соц сеть, шаг 1, верстка, grid

**Делаем верстку на любой технологии, можно на FLEX, GRID, FLOAT.**

**В дано случае будем делать верстку на GRID**

12. Разбивка на компоненты.

Теперь разбиваем нашу разметку на компоненты.

Для этого создадим отдельную папку, в ней отдельные файлы, название пишем **с большой буквы!**

**Не забываем делать импорт-экспорт.**

13. Git

**Ссылка на репозиторий Димыча (там есть TYPESCRIPT):**

https://github.com/it-kamasutra/react-way-of-samurai

**Модули node** не закидываются в репозиторий, автоматом прописывается в GITIGNORE! Но в файл package.json сохраняются все зависимости! Из него node Подтягивает инфо какие модули установить.

Пишем NPM install для установки зависимостей, т.к. при создании проекта их нет

14. CSS Modules

Лучше создавать компоненты, и к ним писать отдельные CSS стили.

В файле компоненты нуно указывать импортируемый стиль CSS, например:

import './Navbar.css'**;**

Но, даже без импорта при совпадении стилей, они будут влиять на классы с таким же названием. Поэтому, если нужно, чтобы CSS стили были прописаны только для конкретной компоненты, в имени файла нужно указывать module, например:

**Navbar.module.css**

При этом все уникальные className, которые генерирует React, содержатся в объекте **classes** в виде:

(чтобы увидеть этот объект, его можно вывести в console.log(classes) )

{

’именование классов, которое я придумал’ : ’именование классов, которые сгенерировал React’,

’именование классов, которое я придумал’ : ’именование классов, которые сгенерировал React’,

…

}

Поэтому, в файле компонента пишем следующее:

- Для импорта объекта стилей:

import classes from './Profile.module.css'**;**

- Для присваивания классов в className вносим следующие значения (classes.posts, например):

export function Profile() {  
 return (  
 <div className="content">  
 <div>  
 <img  
 src="https://i.siteapi.org/no2IMQtVA-Jko-HOxCRaQpEC3FE=/fit-in/900x1000/center/top/filters:format(png)/5045b586a05f134.s.siteapi.org/img/d35c0d1ac7c8dd12e1e4146f14df612b9c31a4a3.jpg"  
 alt="nature"/>  
 </div>  
 <div>Ava + description</div>  
 <div>My posts  
 <div>New post</div>  
 <div className={classes.posts}>  
 <div className={classes.item}>Post1</div>  
 <div className={classes.item}>Post2</div>  
 </div>  
 </div>  
 </div>  
 )  
}

Чтобы дать тегу несколько классов, используем ШАБЛОННЫЕ СТРОКИ:

**className = {`${classes.nav} ${classes.active}`}**

Для аналога стилей CSS:

**.nav . active {**

**…**

**}**

Т.к. при определение классов мы обращаемся к свойствам объекта classes (можно при импорте назвать его по-другому), то мы не можем использовать в именах классов дефис. Можем, если только будем использовать другой синтаксис типа classNmae={classes.[“header-rm”]} вместо {classes.header}. **Поэтому, в именах классов ДЕФИС НЕ ИСПОЛЬЗУЕМ, можно camelCase, можно нижнее подчеркивание\_\_**

15. Структура папок, новые компоненты, улучшаем CSS

Сгруппируем в отдельные папки файл компонента и файл стилей.

Дробим наши полученные компоненты.

В папке исходного компонента создаем папку с именем компонента следующе уровня, вместе с ним и стили css.

Так мы можем использовать одно из главных преимуществ компонент, мы можем их потом дублировать тегами!

16. PROPS

Компонент - это функция, возвращающая разметку JSX

она же

Компонент - это тег

в React функция вызывается с параметром props

export function Header(props) {  
 return (  
 <header className={classes.header}>  
 <img  
 src="https://www.phpro.be/uploads/media/sulu-100x100/00/440-react%404x.png?v=2-0"  
 alt="logo"/>  
 </header>  
 )  
}

React всегда вкидывает объект в функцию (компоненту)

Когда мы указываем компоненту без аттрибутов props, в компоненту приходит пустой оъект.

Компоненту ИЗВНЕ мы настраиваем с помощью атрибутов.

Создадим атрибут компоненту <Header/>

<Header name="Dima K"/>

Тогда React автоматически создаст props объект и отрисовывается будет этот компонент уже с объектом. Т.е. компонент как функция будет иметь доступ к объекту как будто аргумент у функции.

{  
 name: "Dima K"  
}

Как использовать объект props в компоненте (функции) мы прописываем в фигурных скобках {}

export function Header(props) {  
 return (  
 <header className={classes.header}>  
 <img  
 src="https://www.phpro.be/uploads/media/sulu-100x100/00/440-react%404x.png?v=2-0"  
 alt="logo"/>  
 </header>  
 <span>{props.name}</span>  
 )  
}

18. Страница Dialogs

<App/> -это функция, которая возвращает JSX и принимает параметры props.

Один из принципов функционального программирования - функция должна возвращать какое-то значение, основываясь только на входящих данных (предсказуемое поведение)

19.Route, browser-router, маршрутизация

При нажатии на **ссылки вида <a href=”/profile”>** браузер все равно загружает страницу **index.html,** но при этом **менятся ulr.** Если мы хотим, чтобы что-то отрисовывалось в зависимости от url, мы должны использовать **ROUTING.**

Встроенного пакета в Create-React-App нет, поэтому устанавливаем пакет роутинга **через терминал** через команду **npm i react-router-dom -save.**

При использовании yarn и Typescript дописываем:

**yarn add react-router-dom @types/react-router-dom**

**ВАЖНО!** Папка node-modules в нашем проекте не попадает в удаленный репозиторий, нужно обязательно прописать установку этого роутера в конфиге. Поэтому к команде установки модуля добавляем **-save.** Это сохранит зависимости в конфиг- файле package.json и потом при клоне автоматически установит пакет.

Чтобы использовать Routing нужно сделать обрамление всего кода компонента в тег(функцию) **<BrowserRouter>** (Также нужно выполнить **import {BrowserRouter, Route} from 'react-router-dom'**

При использовании **<BrowserRouter>** не обязательно отдельно подключать отдельную библиотеку для подключения истории.

**Итого порядок создания ссылок в приложении React.**

* Добавить модуль router(см выше)
* В ссылке тега <NavLink to=”/Music”> пишем имя компонента Music
* В компоненте отрисовки этих страниц, весь код оборачиваем тегом <BrowserRouter>…</BrowserRouter>
* Отображение компонент пишем в теге вида <Route path='/music' component={Music}/>

Получается примерно следующее:

function App() {  
 return (  
 <BrowserRouter>  
 <div className="app-wrapper">  
 <Header/>  
 <Navbar/>  
 {/\*<Profile/>\*/}  
 <div className="app-wrapper-content">  
 <Route path='/dialogs' component={Dialogs}/>  
 <Route path='/profile' component={Profile}/>  
 <Route path='/news' component={News}/>  
 <Route path='/music' component={Music}/>  
 <Route path='/Settings' component={Settings}/>  
 </div>  
 </div>  
 </BrowserRouter>  
 )**;**}

20. NavLink

При нажатии на **ссылки вида <a href=”/profile”>** браузер перезагружает всю страницу целиком, а это противоречит концепции SPA, поэтому вместо ссылок в тегах <a> используем ссылки в тегах(они же компоненты) **<NavLink>.** NavLink нужно импортировать!

Вместо атрибута href=’’ нужно использовать атрибут **to=’’.** Все атрибуты компонент превращаются в PROPS. Итого получаем ссылки такого вида:

import React from 'react'**;**import classes from './Navbar.module.css'**;**import {NavLink} from 'react-router-dom'**;**export function Navbar() {  
 return (  
 <nav className={classes.nav}>  
 <div className={classes.item}>  
 <NavLink to="/profile">Profile</NavLink></div>  
 <div className={classes.item}>  
 <NavLink to="/dialogs">Messages</NavLink></div>  
 <div className={classes.item}>  
 <NavLink to="/News">News</NavLink>  
 </div>  
 <div className={classes.item}>  
 <NavLink to="/Music">Music</NavLink></div>  
 <div className={classes.item}>  
 <NavLink to="/Settings">Settings</NavLink></div>  
 </nav>  
 )  
}

При этом, если посмотреть в devtools, компоненты NavLink будут отображаться как теги <a> с сcылками href.

При нажатии на ссылку, определенную как NavLink, он добавляет ссылке **класс active** (это моно увидеть в DevTools)

Но классы CSS мы должны брать из объекта CLASSESб поэтому просто прописать стили классу active работать не будет. Чтобы заработал класс active, нужно добавить к Navlink такой атрибут:

activeClassName={classes.activeLink}

где .activeLink - это имя класса для состояния active.

Итого наша компонента выглядит так:

import React from 'react'**;**import classes from './Navbar.module.css'**;**import {NavLink} from 'react-router-dom'**;**export function Navbar() {  
 return (  
 <nav className={classes.nav}>  
 <div className={classes.item}>  
 <NavLink to="/profile"  
 activeClassName={classes.activeLink}>Profile</NavLink>  
 </div>  
 <div className={classes.item}>  
 <NavLink to="/dialogs"  
 activeClassName={classes.activeLink}>Messages</NavLink>  
 </div>  
 <div className={classes.item}>  
 <NavLink to="/News"  
 activeClassName={classes.activeLink}>News</NavLink>  
 </div>  
 <div className={classes.item}>  
 <NavLink to="/Music"  
 activeClassName={classes.activeLink}>Music</NavLink>  
 </div>  
 <div className={classes.item}>  
 <NavLink to="/Settings"  
 activeClassName={classes.activeLink}>Settings</NavLink>  
 </div>  
 </nav>  
 )  
}

а модуль CSS следующим образом:

.nav {  
 grid-area: nav**;** background-color: darkkhaki**;**}  
  
.item a {  
 color: whitesmoke**;**}  
  
.item a.activeLink {  
 color: gold**;**}

22.Route exact

Если в библиотечную компоненту **<Route>** добавить **exact**, то он будет отслеживать путь по точному совпадению.

<Route exact path='/dialogs' component={Dialogs}/>

При смене ulr НЕ обязательно нужно делать новый ROUT, т.е. мы можем и нам нужно при клике подгружать не новую компоненту, а просто содержимое (текст)

24. UI - BLL, выносим данные в сторону

BLL - business logic layer (REDUX)

UI - user interface (REACT)

25. Отображаем компоненты с посощью метода массива MAP

Имея исходный массив данных, мы на основании него можем отрисовывать массив компонентов.

**const postsData = [  
 {id: 1, message: "Hi! How are you?"},  
 {id: 2, message: "It is my first post"},  
 ]**  
  
 const postsElements = postsData.map((item)=> {  
 return (<Post message={item.message}/>)  
 })  
  
 return (  
 <div className={classes.postsBlock}>  
 <h3>My posts</h3>  
 <div>  
 <div>  
 <textarea>Input your thoughts here...</textarea>  
 </div>  
 <div>  
 <button>Add post</button>  
 <button>Remove</button>  
 </div>  
 </div>  
 <div className={classes.posts}>  
 **{postsElements}**  
 </div>  
 </div>  
 )  
}

При изменении исходного массива данных будет автоматически отображаться нужное количество компонент с данными из массива

26. Структура проекта, еще раз про ui-bll

- index.ts не является компонентой проекта.

- из intex.ts вызываем App.

- ВСЕ содержимое App оборачивается обязательно BrowserRouter (или HashRouter )

function App() {  
 return (  
 <BrowserRouter>  
 <div className="app-wrapper">  
 <Header/>  
 <Navbar/>  
 {/\*<Profile/>\*/}  
 <div className="app-wrapper-content">  
 <Route path='/dialogs' component={Dialogs}/>  
 <Route path='/profile' component={Profile}/>  
 <Route path='/news' component={News}/>  
 <Route path='/music' component={Music}/>  
 <Route path='/Settings' component={Settings}/>  
 </div>  
 </div>  
 </BrowserRouter>  
 );  
}

- из App далее идет ветвление всех остальных компонент

- Все компоненты должны находиться в отдельных файлах.

**BLL (Business Logic Layer) redux**

BLL это некая абстракция, которая отвечает за хранение данных. Из нее PROPS приходят в компоненту. На сегодня BLL это REDUX.

Будем переносить все данные в BLL (index.tsx)

**UI (user interface) react**

Данные в реакт приходят из BLL

27. Components vs Render, прокидываем PROPS через Render

Если мы перенесем все данные в index.tsx, то компиляция будет невозможна, т.к. компоненты не смогут передать данные из index.tsx из-за ROUTE. Решить эту проблему можно двумя путями:

1. Используем атрибут **render.** Router ожидает получить или component или render, одно из двух

function App() {  
 return (  
 <BrowserRouter>  
 <div className="app-wrapper">  
 <Header/>  
 <Navbar/>  
 {/\*<Profile/>\*/}  
 <div className="app-wrapper-content">

<Route path='/dialogs' **component**={Dialogs}/>  
//или  
  
 <Route path='/dialogs' **render**={Dialogs}/>  
  
 </div>  
 </div>  
 </BrowserRouter>  
 );  
}

Render принимает не функцию(компоненту), а стрелочную функцию, которая вызовет нужную компоненту

28. Выносим данные в index.tsx

Если мы перенесем все данные в index, нужно создать типизацию, а также прокинуть все данные через все props.

Если мы используем метод MAP, то его нужно писать внутри компоненты

29. Упаковываем данные в STATE

Хранить все данные будем в файле STATE.TS.

Из этого файла будем делать импорт данных в INDEX.JSX, чтобы система была контролируемая

31. onClick, ref, Virtual DOM

В Реакте мы не можем напрямую обращаться к DOM, т.е. у нас есть Virtual DOM. Например, м не можем брать данные из document.getElementById и прочее.

Для того, чтобы вытянуть значения из TextArea будем использовать ref.

import React from 'react'

import classes from './MyPosts.module.css'

import { Post } from './Posts/Post'

export *type* MyPostsPropsType = {

  posts: *Array*<PostElementType>

}

export *type* PostElementType = {

  id: *number*

  message: *string*

}

export *function* MyPosts (*props*: MyPostsPropsType) {

*let* postsElements = *props*.posts.map(*item* *=>* {

    return <*Post* message={*item*.message} />

  })

*let* newPostElement = React.createRef() //создаем переменную

  return (

    <div className={classes.postsBlock}>

      <h3>My posts</h3>

      <div>

        <div> //вписываем ref={}

          <textarea ref={newPostElement}>Input your thoughts here...</textarea>

        </div>

        <div>

          <button>Add post</button>

          <button>Remove</button>

        </div>

      </div>

      <div className={classes.posts}>{postsElements}</div>

    </div>

  )

}

31. Прокидываем callback через PROPS

Когда мы нажимаем на textarea, чтобы создать пост, нам нужно добавить сначала эти данные в исходный массив, который хранится в redux. Там же должна храниться функция, которая добавляет данные к исходному массиву

Перейдем в sate.js и создадим функцию добавления постов:

export *let* addPost = (*postMessage*: *string*) *=>* {

*let* newPost = {

    id: 5,

    message: *postMessage*

    // likesCount: 0

  }

  state.postsData.push(newPost)

}

Проверим, работает ли функция, вызвав ее в index.ts (нужно ее импортнуть)

addPost("WoW Samurai");

Далее нам нужно прокинуть функцию в ПРОПС index.tsx

ReactDOM.render(

  <*React.StrictMode*>

    {/\*<App dialogsData={dialogsData} messagesData={messagesData} postsData={postsData}/>\*/}

    <*App* stateData={state} addPost={addPost}/>

  </*React.StrictMode*>,

  document.getElementById('root')

);

и так далее до компоненты MyPosts.tsx. В ней по клику и вызываем функцию.

*let* addPost = () *=>* {

    //debugger

*let* text = newPostElement.current?.value //? значит, что в этом поле моежт быть Null

    if (text) {

      // check if text exists (no undefined). Without this snippet type error is caused

*props*.addPost(text)

    }

  }

  return (

    <div className={classes.postsBlock}>

      <h3>My posts</h3>

      <div>

        <div>

          <textarea ref={newPostElement}></textarea>

        </div>

        <div>

          <button onClick={addPost}>Add post</button>

          <button>Remove</button>

        </div>

      </div>

      <div className={classes.posts}>{postsElements}</div>

    </div>

  )

}

33. Добавление поста на стену (зачатки flux концепции)

FLUX - это архитектура, REDUX - это ее реализация (реализаций много)

Ниже на схеме представлен однонаправленный поток данных. Компоненты(React) могут только отправлять actions, т.е. запрос на изменение данных, могут даже собирать эти данные, могут вызывать функции, пришедшие из-вне. Но менять сове состояние сами не могут, только после команды от STATE, когда он изменился.



НЕЛЬЗЯ ИЗМЕНЯТЬ UI ДО ИЗМЕНЕНИЯ STATE!

Помним - **главный у нас BLL (Business Logic Layer**)**. UI мы можем "убить" в любой момент и отрисовать заново!!!** Даже если в state изменилась какая-то супер маленькая деталь, плевать - берём и отрисовываем ВСЁ заново. Ой, скажете вы, это же не эффективно??) Отчасти вы правы. Но мы на старте не думаем про оптимизацию! Давайте думать про концепцию и архитектуру, а потом будем думать про то, как всё "ускорить". К тому же мы используем React, а он далеко не дурак, чтобы всё перерисовывать, если это не нужно и даже если мы ему типа приказали) FLUX концепция, она немного другая, безусловно. Там есть в цепочке этой немного другие звенья. Но суть ТАКАЯ ЖЕ! Мы будем эту картину формировать поэтапно, постепенно, чтобы у вас было понимание того, что происходит!!! Летим!!!!!!!