# Lesson 14 (css)

1. active существует по умолчанию, его не нужно создавать. Классы, импортированные из Dialogs.module.css, применяются в {} и с меткой, такие классы локальны и имеют уникальные индетефикаторы. Использовать два класса можно двумя способами:

let c1 = 'item'  
let c2 = 'active'  
//' item active'  
let classes = {c1+' '+c2};  
let classesnew = {`${c1} ${c2}`};

<div className={s.dialog + ' ' + s.active}>Dimych</div>

# Lesson 25 (map)

let dialogsData = [  
 {id: 1, name: 'Dimych'},  
 {id: 2, name: 'Andrey'},  
 {id: 3, name: 'Anya'},  
 {id: 4, name: 'Sveta'},  
 {id: 5, name: 'Viktor'},  
 {id: 6, name: 'Valera'},  
 ]

<DialogItem name={dialogsData[0].name} id={dialogsData[0].id}/>,  
<DialogItem name={dialogsData[1].name} id={dialogsData[1].id}/>,  
<DialogItem name={dialogsData[2].name} id={dialogsData[2].id}/>,

1. Можно положить в массив, и вызвать массив:

let dialogsElements = [  
 <DialogItem name={dialogsData[0].name} id={dialogsData[0].id}/>,  
 <DialogItem name={dialogsData[1].name} id={dialogsData[1].id}/>,  
 <DialogItem name={dialogsData[2].name} id={dialogsData[2].id}/>,]

{dialogsElements}

Метод массива map – проходит по каждому элементу массива, и создает новый массив, согласно условию ( в данном случае – если имя мужское – возвращает 1, если нет – 0 )

let oldArray = ['dima','olga','andrey']  
  
let newArray = oldArray.map (name => isMaleName(name) ? 1:0)  
  
получаем : [1,0,1]

1. Вместо dialogsElements иcпользуем map, в итоге создается столько dialogsData, сколько у нас сейчас в диалог дата (при добавлении нового обьекта , он сразу отрисуется)

let dialogsElements = dialogsData  
 .map (dialog => <DialogItem name={dialog.name} id={dialog.id}/> )

# Lesson 26

BLL – bussines logic layer - данные, библеотека(redux) хранятся в BLL

UI – user interface – то,где мы работаем (react)

UI отвечается за отрисовку приходящих данных ( взаимодействие с пользователем), поэтому отправляем массивы данных(такие, как dialogs) в index.tsx

let dialogs = [  
 {id: 1, name: 'Dimych'},  
 {id: 2, name: 'Andrey'},  
 {id: 3, name: 'Anya'},  
 {id: 4, name: 'Sveta'},  
 {id: 5, name: 'Viktor'},  
 {id: 6, name: 'Valera'},  
 ]

# Lesson 27-28

Было

<Route path= '/profile' component={Profile}/>

Стало, что позволяет прокинуть пропсы

<Route path='/profile' render={() => <Profile/>}/>

Вынос данных вверх,было :

function Profile() {  
 let posts = [  
 {id: 1, messages: 'Hi,how are you?', likesCount: 11, img: 'https://upload.wikimedia.org/wikipedia/commons/thumb/0/07/Spb\_metro\_logo.svg/1280px-Spb\_metro\_logo.svg.png'},  
 {id: 2, messages: 'its my first posts', likesCount: 41, img: 'https://upload.wikimedia.org/wikipedia/commons/thumb/e/e5/NASA\_logo.svg/200px-NASA\_logo.svg.png'},  
 ]  
  
 return (  
 <div>  
 <ProfileInfo/>  
 <MyPosts posts={posts} />  
 </div>

Стало:

function Profile(props:PostsType) {  
 return (  
 <div>  
 <ProfileInfo/>  
 <MyPosts posts={props.posts} />  
 </div>

В компоненту приходят данные, и она передает их дальше (используем props.)

Типизируем таким образом: три массива, у каждого свой массив с типизацией.

type AppType = {  
 posts:Array<PostType>  
 messages:Array<MessageType>  
 dialogs:Array<DialogType>  
}

В итоге вынесли все данные в index.tsx

***Объекты*** – это ассоциативные массивы с рядом дополнительных возможностей.

Они хранят свойства (пары ключ-значение), где:

* Ключи свойств должны быть строками или символами (обычно строками).
* Значения могут быть любого типа.

# Lesson 29

1. Создали файл state.tsx, перенесли данные в стайт.
2. Типизировали

# Lesson 32

1. Создали в стейт функцию, которая добавляет сообщения запушивая его в массив, прокинули ее через пропс

export let addPost = (postMessage:string) => {  
 let newPost:PostsType = {  
 id:new ***Date***().getTime(),//уникальный ид  
 messages:postMessage,  
 likesCount: 0,  
 img:’’  
 ***state***.profilePage.posts.push(newPost)

# Lesson 33

1. Создали функцию ререндер, обернули ей все содержимое индекс и вынесли в отдельный файл, в индексе вызываем функцию со стейтом в параметре

# Lesson 34

Flux круговорот – при вводе значение в инпут, это значение сначала попадает в стейт и только потом отприсовывается.

1. При изменении инпута, в updateNewPostText записывается текущее состояние инпута, далее при нажатии кнопки функция аддПост отрисовывает сообщение, записанное в newPostText

let addPost = () => { props.addPost()}  
const onChangeHandler = (event: ChangeEvent<HTMLTextAreaElement>) => {  
 props.updateNewPostText(event.currentTarget.value)  
}

<textarea  
 onChange={onChangeHandler}  
 value={props.newPostText}  
 ref={newPostElement}> </textarea>  
</div>  
<div>  
 <button onClick={addPost}> Add post</button>

export let addPost = () => {  
 let newPost:PostsType = {  
 id:new ***Date***().getTime(),  
 messages:***state***.profilePage.newPostText,  
 likesCount: 0,  
 img: 'https://   
 }  
 ***state***.profilePage.posts.push(newPost)  
 ***state***.profilePage.newPostText = ''  
 renderTree(***state***)  
}  
export let updateNewPostText = (newText:string) => {  
 ***state***.profilePage.newPostText = newText  
 renderTree(***state***)  
}

# Lesson 34

1. renderTreeee = observer вместо содержимого рендерТри теперь будет обсерв

let renderTreeee = () => {  
 ***console***.log('123')  
}  
export const subscribe = (observer:any) => {  
 renderTreeee = observer  
}

1. рендерТри не изменится, перменная работает только внутри функции

export const subscribe = (observer:any) => {  
 let renderTreeee = observer  
}

# Lesson 39

1. Аналогичные записи(мы возвращаем обьект, и чтобы фигурные скобки не воспринялись как тело функции, нужно обернуть их в круглые)

export const addPostActionCreator = () => ({type: ADD\_POST})

export const addPostActionCreator = () => {  
 return {type: ADD\_POST}  
}

# Lesson 41

1.Было

if (action.type === ADD\_POST) {  
 let newPost:PostsType = {  
 id:new ***Date***().getTime(),  
 messages:state.newPostText,  
 likesCount: 0,  
 }  
 state.posts.push(newPost)  
 state.newPostText = ''  
  
 } else if(action.type === UPDATE\_NEW\_POST){  
 state.newPostText = action.newText  
}

2.Стало(аналог иф елс)

switch (action.type) {  
 case ADD\_POST:  
 let newPost: PostsType = {  
 id: new ***Date***().getTime(),  
 messages: state.newPostText,  
 likesCount: 0,  
 }  
 state.posts.push(newPost)  
 state.newPostText = '';  
 return state  
  
 case UPDATE\_NEW\_POST:  
 state.newPostText = action.newText;  
 return state  
 default:  
 return state  
}

1. Reducer.ы – это функции, которые получают свою часть стейта и экшн, и возвращает измененый стейт (вынесли функции из диспатча, для деструкторизации)
2. Автоматическая типизация(в функции нужно дописать as const):

type AddPostActionType = ReturnType<typeof addPostCreator>

export const addPostCreator = () => {  
 return {  
 type: ADD\_POST  
 } as const  
}

# Lesson 42

1. Redux – библиотека, с встроенными фугкциями(диспатч,сабскрайб итд), используется для хранения стейта и методов
2. Создаем стор(используя старый стор)
3. let ***store*** = ***createStore***(reducer)
4. Каждому разделу стейта присваеваем свои методы
5. let reducer = combineReducers({  
    sidebar: sidebarReducer,  
    dialogsPage: dialogsReducer,  
    profilePage: profileReducer  
   })

# Lesson 43

1.