Для того чтобы запустить реакт у нас в проекте, переходим <https://github.com/facebook/create-react-app>

В консоли node.js выбираем папку с проектом и пишим команду

Чтобы выбрать папку пишем сначала d: - перешли на диск D

А далее cd PROJECT\ React.js\ 01-first-project

Теперь мы вбиваем npx create-react-app my-app

my-app ‘то название проекта, Которое может быть любым

После установки командой запускаем реакт, предварительно перейдя в папку my-app

И устанавливается пакет node\_modules, который в дальнейшем я буду устнавливать по аналогии с gulp, тоесть просто копируя в новый проект файл package.json

class в реакте ClassName

**Что такое компонета**

Компонента – это функция, возвращающая разметку (jsx(скриптовый html))

**const** *App* = () =>  
{  
 **return**(  
 <**div className="App"**>  
 <**p className="ds"**>Привет Наташка</**p**>  
 </**div**>  
 )  
};

App – это компонента, то есть функция, которая возвращает html размету. Название компоненты должно быть с большой буквы!!! Далее попробую прописать дальнейшую судьбу этого компонента.

Вызывается он в файле index.js, который находится на одном уровне с файлом App.js, в котором находится компонент App

………

**import** *App* **from './App'**; импортируй компонент App из файла App

………

ReactDOM.render(<**App** />, ***document***.getElementById(**'root'**));

Переводится это буквально так – ReactDOM.отрисуй тэг <App />, а это не что инное, как компонент(функция) App,которую мы вставляем как тэг, и вставляем его в крнтейнер с айдишником **root** , котрый находится в файле index.html . А в этом файле всего лишь пустой контейнер, в который в последствии с помощью js будет формироваться разметка страницы.

Реакт всецело состоит из компонентов. Для примера мы один компонент можем составить из разных компонентов, кторые мы будем вызывать:

**const** *App* = () =>  
{  
 **return**(  
 <**div className="container App"**>  
 <**Fun\_header**/>  
 <**Fun\_footer**/>  
 </**div**>  
 )  
};

**const** *Fun\_header*=()=>  
{  
 **return**(  
 <**header**>  
 <**p**>Это классный хедер</**p**>  
 </**header**>  
 )  
};  
**const** *Fun\_footer* = () =>  
{  
 **return**(  
 <**footer**>  
 <**p**>Это классный футер</**p**>  
 </**footer**>  
 )  
};

**export default** *App*;

Но это не является правильным.Каждый компонент должен быть отдельным файлом. Поэтому отдельно создаем файл Header.js =>

Отдельно создаем Footer.js

**const** *Fun\_footer* = () =>  
{  
 **return**(  
 <**footer**>  
 <**p**>Это классный футер</**p**>  
 </**footer**>  
 )  
};  
  
**export default** *Fun\_footer*;

**const** *Fun\_header*=()=>  
{  
 **return**(  
 <**header**>  
 <**p**>Это классный хедер</**p**>  
 </**header**>  
 )  
};  
  
**export default** *Fun\_header*;

И в файле App.js прописываем

**import** *Fun\_header* **from './Header'**;  
**import** *Fun\_footer* **from './Footer'**;  
  
**const** *App* = () =>  
{  
 **return**(  
 <**div className="container App"**>  
 <**Fun\_header**/>  
 <**Fun\_footer**/>  
 </**div**>  
 )  
};

**Стили. CSS модули**

Для каждого компонента можно делать отдельный css файл. Есть возможность в разных компоннтах элементам давать одни и те же классы с разными стилями, не волнуясь, что они перекроют друг друга. Для этого файл со стилями называем таким образом:

Переименовываем Body.css в Body.module.css

Подключение файла со стилями import classes(любое название) from “./ Body.module.css ”

classes – это объект, ключ – значение, где ключ это моё название класса, а значение сгенерированное реактом, пример:

let classes = {

‘’item”:”Body\_\_item\_\_65759”

}

Но теперь название всех классов без исключения задаем так:

<**div className={classes.item}** >…..</div>

Но теперь мы не можем использовать в названии класса дефис, так он записывается как джаваскрипт. Поэтому ибо в названии класса не использовать девис, либо записывать так

<**div className={class[‘item-logo’]}** >…..</div>

Если нужно в элемент записать два и более класса:

<**p className=**{**`**${classes.item} ${classes.active}**`**}>А это классный Боди</**p**>

Пример:

Компонент Body

**import** classes **from './Body.module.css'  
const** *Body*=()=>  
{  
 **return**(  
 <**div className=**{classes.body}>  
 <**ul className="navbar"**>  
 <**li className=**{**`**${classes.item} ${classes.active}**`**}>Пункт 1</**li**>  
 <**li className=**{classes.item}>Пункт 2</**li**>  
 </**ul**>  
  
 </**div**>  
 )  
};  
  
**export default** *Body*;

Будет сгенерирован класс

<li class="Body\_item\_\_2kh\_N">Пункт 2</li>

А в компоненте Footer

<li class="Footer\_item\_\_3wVE4">Пукт 1</li>

В прошлый раз я не понял как это делать поэтому подключал с сервера

подключаю с сервера вот так:

<**script src="https://unpkg.com/react@16/umd/react.development.js"**></**script**>  
<**script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"**></**script**>  
  
*<!-- Don't use this in production: -->*<**script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"**></**script**>

props

Любая компонента это функция, которая может иметь какой-либо параметр. Этот параметр и назыается props, который мы можем в дальней передавать при вызове компонента. Происходит это как с обычной функцией. Пример, у нас есть компонент Post

**const** *Post*=(props)=>  
{  
  
 **return**(  
 <**div className=**{classes.post\_item}>  
 <**div className=**{classes.post\_item\_\_text}>  
 <**p**>{props.**message**}</**p**>  
 </**div**>  
 <**div className”**post\_item\_\_likes**”**>  
 <**p**>{props.**like**} лайков</**p**>  
 </**div**>  
 </**div**>  
 )  
};

Когда мы вызываем компонент(а это ни что инное как функция) мы через атрибуты передаём значения параметров

**const** *MyPosts*=()=>  
{  
 **return**(  
 <**div className=**{classes.wrapper\_posts}>  
 <**Post like=**{**"15"**} **message="Привет"** />  
 </**div**>  
 )  
};

Итог:

Привет

15 лайков

После вызова:

<**p**>{props.**message**}</**p**><p>props={ **message**:”Привет”}

<**p**>{props.**like**}</**p**> =<p>props={ **like**:”15”}

**Переходы между страницами**

Для дого чтобы работали ссылки и мы могла перемещатся с одной страницы на другую в рамках проекта устанавливается модуль роутинга, котрый не установлен по умолчанию.

Npm install react-router-dom -save

-save чтобы запись об установлнном модуле добавилась в package.json

Теперь переходим в самый верхний компонент, в моём случае это App.js , и оборачиваем его в тэг <**BrowserRouter**>. Другие компоненты уже не нужно будет оборачивать в него.. Перед этим мы подключаем его.

**import** {BrowserRouter} **from "react-router-dom"**;

Теперь переходим в компоненты, на которые будем переключаться при нажатии на ссылку. Эти компоненты записываем по новому.

Как было, когда я только начал изучать:

**const** *Profile*=(props)=>  
{  
 **return** (  
 <**div className=**{classes.body}>  
 <**MyPosts**/> */\*компонент с постами\*/* <**Dialogs**/> */\*компонент с диалогами\*/* </**div**>  
  
 )  
};  
**export default** *Profile*;

И как стало:

**const** *Profile* = (props) => {  
 **return** (  
 <**div className=**{classes.body}>  
 <**Route path='/posts' component=**{*MyPosts*}/>  
 <**Route path='/dialogs' component=**{*Dialogs*}/>  
 </**div**>  
 )  
};  
**export default** *Profile*;

Но для того, чтобы в компонент можно было передавать переременные правильнее записывать так :

<**div className="wrapper\_profile"**>  
 <**Route path='/posts' render=**{()=><**ContainerPosts**/>}/>  
 <**Route path='/dialogs' render=**{()=><**ContainerDialogs**/>}/>  
</**div**>

**path='/posts' component=**{*MyPosts*}- Это своего рода якоря на компоненты на которые мы будем переходить при нажатии на ссылку с ссылкой на этот компонент.

И это следующий шаг.

**const** *NavBar*=()=>  
{  
 **return**(  
 <**div className=**{classes.wrapper\_nav\_bar}>  
 <**ul className=**{classes.nav\_bar\_list}>  
 <**li**><**a href="/posts"**>MyPosts</**a**></**li**>  
 <**li**><**a href="/dialogs"**>Dialog</**a**></**li**>  
 </**ul**>  
 </**div**>  
 )  
};

Теперь, если в какой либо части страницы у нас есть ссылка на загрузку компонента например диалог, достаочно в тэге a прописать

<**a href="/dialogs"**>Dialog</**a**></**li**>

Но так как на нужный контент мы переходим по ссылке у нас проичсходит перезагрузка страницы, а это не должно быть. По этому ссылки

<**li**><**a href='/posts'**>MyPosts</**a**></**li**>  
<**li**><**a href='/dialogs'**>MyPosts</**a**></**li**>

Мы записываем так:

<**li**><**NavLink to="/posts"**>MyPosts</**NavLink**></**li**>  
<**li**><**NavLink to="/dialogs"**>Dialog</**NavLink**></**li**>

Перед этим мы подключаем компонент *NavLink*

**import** {*NavLink*} **from 'react-router-dom'**

При выборе той или инной ссылке обернутой в компонент NavLink ей автоматом ставится класс active, которой можно задать классы и где не нужно будет программно добавлять этот класс

Но активной ссылке можно задать и свой класс, который будет ставится автоматом, для этого:

Прописываем её в activeClassName Это нужно делать у каждой ссылке

<**NavLink activeClassName=**{**"active\_link"**} **to="/posts"**>MyPosts</**NavLink**>

<**NavLink activeClassName=**{**"active\_link"**} **to="/dialogs"**>Dialogs</**NavLink**>

**Вывод данных из массива с помощью цикла**

В реакте есть класнная вещь. Если в компоненте стоит массив, реакт понимает, что это массив и разбирает его. Но нам нужно, чтобы данные не просто попадали в компонент, но попадали туда ввиде тэгов. В джаваскрипте есть метод **map**, который создаёт новый массив, на основе обработки элементов из старого массива.

**var** names = [**'HTML'**, **'CSS'**, **'JavaScript'**];  
  
**var** newnames = names.map(el=>**'<li>${el}</li>'**)  
***console***.log(newnames)

1. Array(3)
   1. 0: "<li>${el}</li>"
   2. 1: "<li>${el}</li>"
   3. 2: "<li>${el}</li>"

Пример:

Мы получаем массив данных

let old\_massive\_users=[

{

id:1,

name:'Олеся'

},

{

id:2,

name:'Оксана'

},

{

id:3,

name:'Маша'

},

{

id:4,

name:'Катя'

}

]

Преобразуем его в новый массив:

**let** new\_massive\_users = old\_massive\_users.map(el=><**DialogsUsers name=**{el.**name**} **id=**{el.**id**}/>)

И подставим в компонент этот массив. Писать циклдля разбора массива не нужно, реакт все сделает сам.

<**div className="dialogs\_container\_users"**>  
 {new\_massive\_users}  
</**div**>

Но эти данные в любом случае не должны быть в компоненте. По меньшей мере они должны быть вынесены во внешний файл, пусть им будет пока index.js. В последующем все данные будут храниться в отдельном файле state.js. А пока дабы поучиться Чтобы массив вынести из компонента на коком бы уровне вложенности он бы не был он дожен пройти через все уровни. А массив данных с помощью props будет передаваться из компонента в компонент. Допустим данные должен получать компонент Dialogs.

В нем массив будет с помощью метода map преобразовываться в массив с тэгами

Весь путь:

Index.js

**let** massive\_massage = [  
 {  
 *// id: 1,* **massage**: **'Привет, как дела?'** },  
 {  
 *// id: 2,* **massage**: **'Когда пойдешь на работу?'** }  
];  
  
ReactDOM.render(<**App massive\_users=**{massive\_users} **massive\_massage=**{massive\_massage}/>, ***document***.getElementById(**'root'**));

App.js

**const** *App* = (props) =>  
{  
 **return**(  
 <**BrowserRouter**>  
 <**div className="wrapper"**>  
 <**Header**/>  
 <**MainContainer massive\_users=**{props.massive\_users} **massive\_massage=**{props.massive\_massage}/>  
 <**Footer**/>  
 </**div**>  
 </**BrowserRouter**>  
  
 )  
};

*MainContainer.js*

**const** *MainContainer* =(props)=>  
{  
 **return** (  
  
 <**div className="body\_container"**>  
 <**div className="container"**>  
 <**div className="body\_container\_\_wrapper"**>  
 <**NavBar**/>  
 <**Profile massive\_users=**{props.massive\_users} **massive\_massage=**{props.massive\_massage}/>  
 </**div**>  
  
 </**div**>  
 </**div**>  
  
 )  
};  
**export default** *MainContainer*

**const** *ContainerDialogs* =(props)=>  
{  
 **let** massive\_userse = props.massive\_users.map(el=><**DialogsUsers name=**{el.**name**} **id=**{el.**id**}/>)  
 **let** massive\_massage = props.massive\_massage.map(el=><**MassegeFromUsers message=**{el.**massage**}/>)  
 **return** (  
 <**div className='dialogs\_container'**>  
 <**div className="dialogs\_container\_\_wrapper"**>  
 <**div className="dialogs\_container\_users"**>  
 {massive\_userse}  
 </**div**>  
  
 <**div className="dialogs\_container\_message"**>  
 {massive\_massage}  
 </**div**>  
  
 </**div**>  
 </**div**>  
 )  
};  
**export default** *ContainerDialogs*;

*Body.js*

**const** *Body* = (props) => {  
 **return** (  
 <**div className="wrapper\_profile"**>  
 <**Route path='/posts' render=**{()=><**ContainerPosts** />}/>  
 <**Route path='/dialogs' render=**{()=><**ContainerDialogs massive\_users=**{props.massive\_users} **massive\_massage=**{props.massive\_massage}/>}/>  
 </**div**>  
 )  
};  
**export default** *Body*;

*ContainerDialogs.js*

**const** *ContainerDialogs* =(props)=>  
{  
 **let** massive\_userse = props.massive\_users.map(el=><**DialogsUsers name=**{el.**name**} **id=**{el.**id**}/>)  
 **let** massive\_massage = props.massive\_massage.map(el=><**MassegeFromUsers message=**{el.**massage**}/>)  
 **return** (  
 <**div className='dialogs\_container'**>  
 <**div className="dialogs\_container\_\_wrapper"**>  
 <**div className="dialogs\_container\_users"**>  
 {massive\_userse}  
 </**div**>  
  
 <**div className="dialogs\_container\_message"**>  
 {massive\_massage}  
 </**div**>  
  
 </**div**>  
 </**div**>  
 )  
};  
**export default** *ContainerDialogs*;

**Упаковываем данные в state**

А теперь выведем внешние данные в отдельный файл state.js, который будет храниться в отдельной папке STATE.

Теперь путь данных от новоо файла в компонент *ContainerDialogs* выглядит так.

Файл state.js

**let** state={  
 **massages**:{  
 **massive\_users**:[  
 {  
 **id**:1,  
 **name**:**'Олеся'** },  
 {  
 **id**:2,  
 **name**:**'Оксана'** }  
 ]  
 }  
};  
**export default** state

В index.js подключаем его командой import

**import** state **from './redux/state'**ReactDOM.render(<**App appState=**{state}/>, ***document***.getElementById(**'root'**));

И передаем параметром в функцию App

**const** *App* = (props) =>  
{  
 **return**(  
 <**BrowserRouter**>  
 <**div className="wrapper"**>  
 <**Header**/>  
 <**MainContainer appState=**{props.appState}/>  
 <**Footer**/>  
 </**div**>  
 </**BrowserRouter**>  
  
 )  
};

**const** *MainContainer* =(props)=>  
{  
 **return** (  
  
 <**div className="body\_container"**>  
 <**div className="container"**>  
 <**div className="body\_container\_\_wrapper"**>  
 <**NavBar**/>  
 <**Profile appState=**{props.appState}/>  
 </**div**>  
  
 </**div**>  
 </**div**>  
  
 )  
};

**const** *Body* = (props) => {  
 **return** (  
 <**div className="wrapper\_profile"**>  
 <**Route path='/posts' render=**{()=><**ContainerPosts** />}/>  
 <**Route path='/dialogs' render=**{()=><**ContainerDialogs appState=**{props.appState}/>}/>  
 </**div**>  
 )  
};

**const** *ContainerDialogs* =(props)=>  
{  
 **let** massive\_userse = props.appState.**massages**.**massive\_users**.map(el=><**DialogsUsers name=**{el.**name**} **id=**{el.**id**}/>)  
 **let** massive\_massage = props.appState.**massages**.**massive\_massage**.map(el=><**MassegeFromUsers message=**{el.**massage**}/>)  
 **return** (  
 <**div className='dialogs\_container'**>  
 <**div className="dialogs\_container\_\_wrapper"**>  
 <**div className="dialogs\_container\_users"**>  
 {massive\_userse}  
 </**div**>  
  
 <**div className="dialogs\_container\_message"**>  
 {massive\_massage}  
 </**div**>  
  
 </**div**>  
 </**div**>  
 )  
};

События **onClick**

В дальнейшем эта тема будет разбираться более глубоко, но пока чтобы:

Но в реакте к элементам мы обращаемся не кслассическими методами например:

***document***.getElementById(**"root"**)

В реате есть свои методы. Например для того, чтобы обратиться к элементу с id = root мы делаем следующее

1. создаем пустую ссылку и записываем её в переменную

**let** linkElement =React.*createRef*(); реакт, Создай ссылку

2. далее этку переменную мы присваиваем элементу, кототорому мы хотим обращаться

<**button ref=**{LinkButton} **className='buttton\_dialog'**>Отправить</**button**>

Теперь мы можем обращаться к нему, напрмер получим текст кнопки

**let** *OnClick* = ()=>{  
 **let** text = LinkButton.**current**.innerText;  
 *alert*(text)  
}

<**button ref=**{LinkButton} **onClick=**{*OnClick*} **className='buttton\_dialog'**>Отправить</**button**>

Функции так же как и массив данных мы будем создавать в state.js

**export let** *addPost* = (postMassege)=>{  
 **debugger**;  
 **let** perNewPost = {  
 **id**:4,  
 **massage**:postMassege,  
 **likeCount**:0  
 }  
 state.**massages**.**massive\_post**.push(perNewPost)  
  
};

В данном случае экспорт фукции происходит не по дефолту. Поэтому импртируем в файл index.js так в скобках

**import** {*addPost*} **from './redux/state'**

Фукнкия так же как и массив с данными передаётся через props

ReactDOM.render(<**App addPost=**{*addPost*} **appState=**{state}/>, ***document***.getElementById )

В компоненте, Где эта фукцния должна вызываться записываем так:

**let** *OnClickButton* = ()=>{  
 **let** textareaVal = linkTextarea.**current**.**value**;  
 props.addPost(textareaVal)  
}

………

<**button onClick=**{*OnClickButton*} **className="post\_button"** >Опубликовать пост</**button**>

………

Но когда мы будем нажимать на кнопку, новый пост попадет в мсааив с постами, но он не отрисуется на экране. Для этого нам нужно заново перерисовать страницу

**Далее всё будет кручк, что касается передачи данных. Но пока просто чтоб быть последовательным.**

Как уже было сазано, для того, чтобы страница поменялась нужно изменить state

Логика такая, мы нажимаем на кнопку, меняем state, а далее перерисовываем страницу. Для этого мы создаем фунцкцию, которая будет перерисовывать страницу, как только будут какие-либо изменения

**export let** *rerendPageTree* = (state) => {  
 ReactDOM.render(  
 <**BrowserRouter**>  
 <**App addPost=**{*addPost*} **appState=**{state}/>  
 </**BrowserRouter**>, ***document***.getElementById(**'root'**)  
  
 );  
}

Раньше

<**BrowserRouter**>  
 <**App addPost=**{*addPost*} **appState=**{state}/>  
</**BrowserRouter**>, ***document***.getElementById(**'root'**)

Находилась в файле index.js отрабатывала при загрузке страницы, тем самым отрисовывая её со всеми компонентами, которые лежат в App, а в ней в свою очередь Footer, Header,Body и так далее.

Эту функцию мы помещаем в новый файл render.js

Мы ‘кспортируем её в index.js

**import** {*rerendPageTree*} **from './render'**

и вызываем саму функцию, больше там ничего нет

А также вызываем её в state.js

**export let** *addPost* = (postMassege)=>{  
 **debugger**;  
 **let** perNewPost = {  
 **id**:4,  
 **massage**:postMassege,  
 **likeCount**:0  
 }  
 state.**massages**.**massive\_post**.push(perNewPost)  
 *rerendPageTree*(state)  
  
};

в методе, который мы импортируем от тутда в компонент. При этом мы передаём в параметре сам state с данными, например

**et** state={  
 **massages**:{  
 **massive\_post**:[  
 {**id**:1,**massage**:**"Привет"**,**likeCount**:15},  
 {**id**:2,**massage**:**"Пока"**,**likeCount**:14},  
 {**id**:3,**massage**:**"Вот и погоаворили"**,**likeCount**:28}  
 ],

И получается