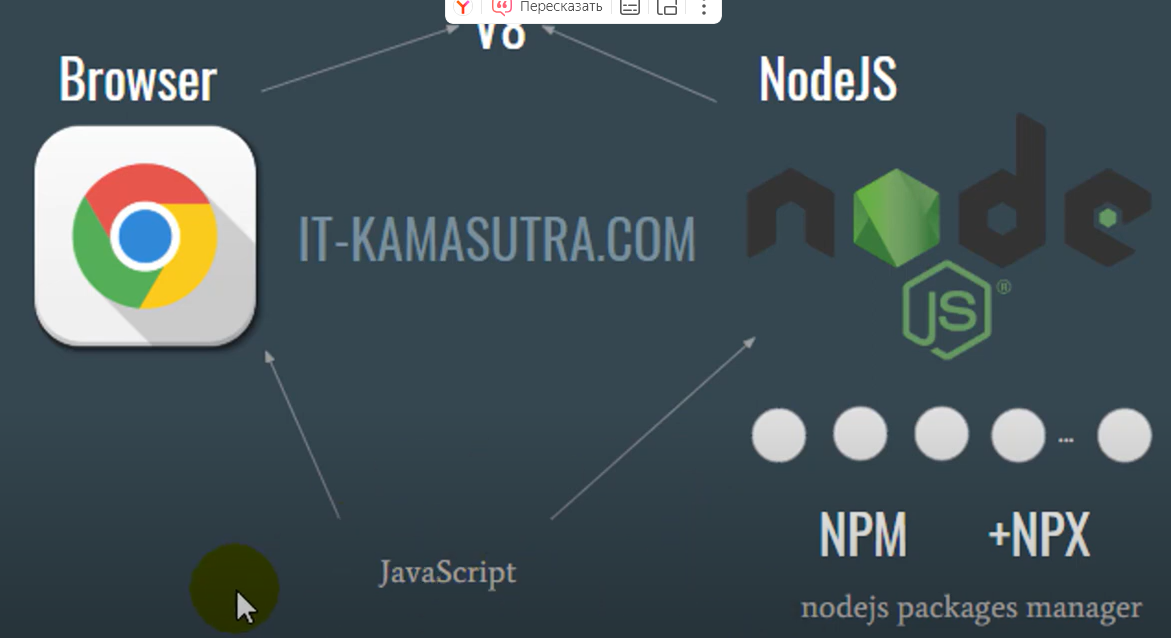
<https://www.youtube.com/watch?v=TPYgQvY9VVQ&list=PLcvhF2Wqh7DNVy1OCUpG3i5lyxyBWhGZ8&index=5>



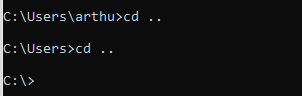
Браузер и node.js являются одинаковыми программами для обработки js кода. Браузер более красивый (визуальная часть), а node.js это консоль (серверная часть, базы данных).

https://github.com/facebook/create-react-app

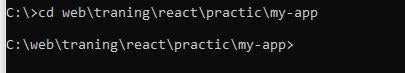
Первым делом необходимо установить Node.js.

1. Необходимо установить node.js
2. Заходим в консоль Node.js через поиск, заходим папку в которой хотим разместить проект.
3. Вводим команды

Для начала необходимо создать папку и перейти в нее в консоли node.js. Спуститься на уровень ниже:



Заходим в папку:



После перехода в созданную папку, вводим команды

npx create-react-app my-app

cd my-app

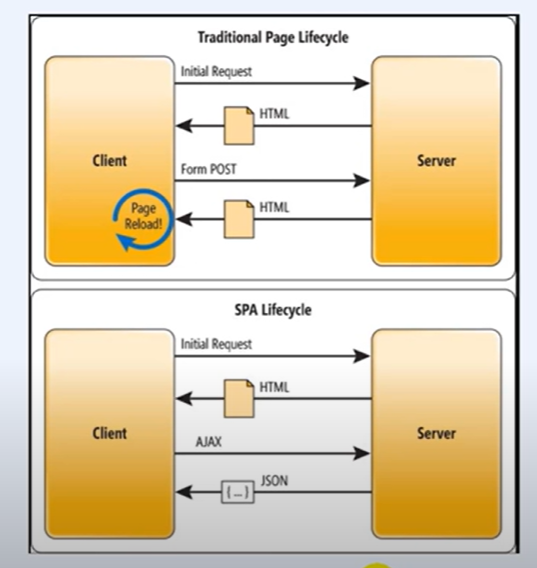
npm start

Если выходят ошибки, значит необходимо реакт установить глобально

При помощи команды:

npm install -g react-scripts

Single page application (SPA)



SPA – это небольшая HTML страница с большим кол-м JS при помощи которого эта страница динамически наполняется контентом.

JSX – специальный язык, который позволяет писать html код внутри JS

# Компонента

Компонента – это функция, возвращающая разметку JSX. Это тег. Название компонента обязательно должно быть с большой буквы.



import './App.css';

Это значит, что App.css импортируется в наш файл на нашем уровне.

const App = () => {

  return (

    <div className="App">

      <div>hello</div>

      <ul>

        <li>html</li>

        <li>css</li>

      </ul>

    </div>

  );

}

В return должен быть общий div или любой другой элемент, но он должен быть единственным, а все остальное внутри данного элемента.

App – это компонента. Данная функция (компонента) вызывается в файле index.js при помощи тега <App />. В переменную root попадает элемент из index.html, затем в этот элемент помещается содержимое <App /> и сервер отправляет пользователю index.html внутри которого много js кода.

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

**<App />**

  </React.StrictMode>

);

Также можно вставлять одну компоненту в другую

const App = () => {

  return (

    <div>

      <Header />

      <Technologies />

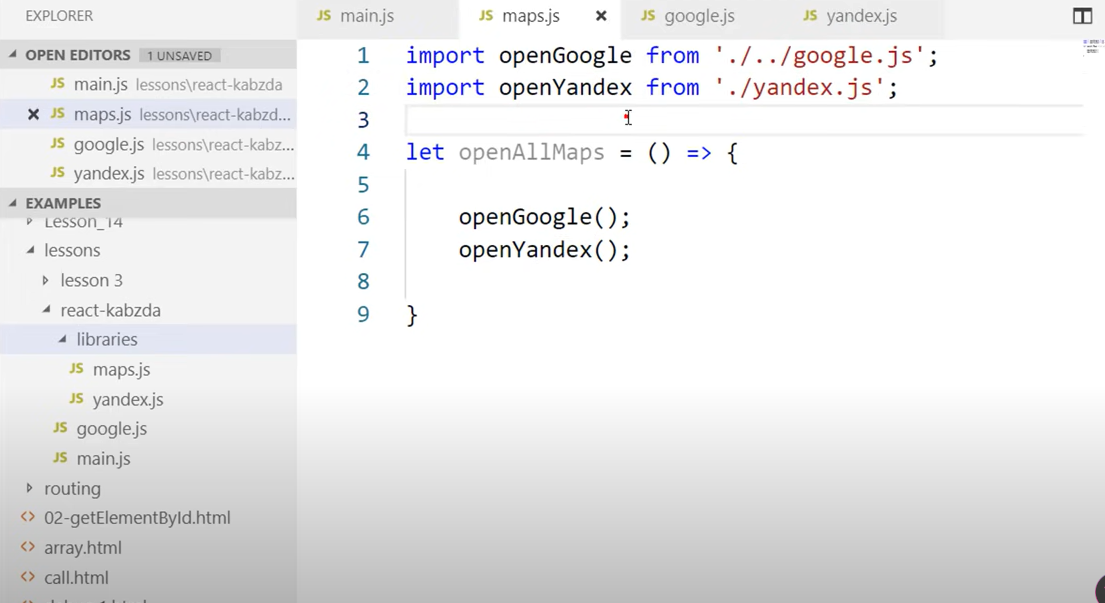
    </div>

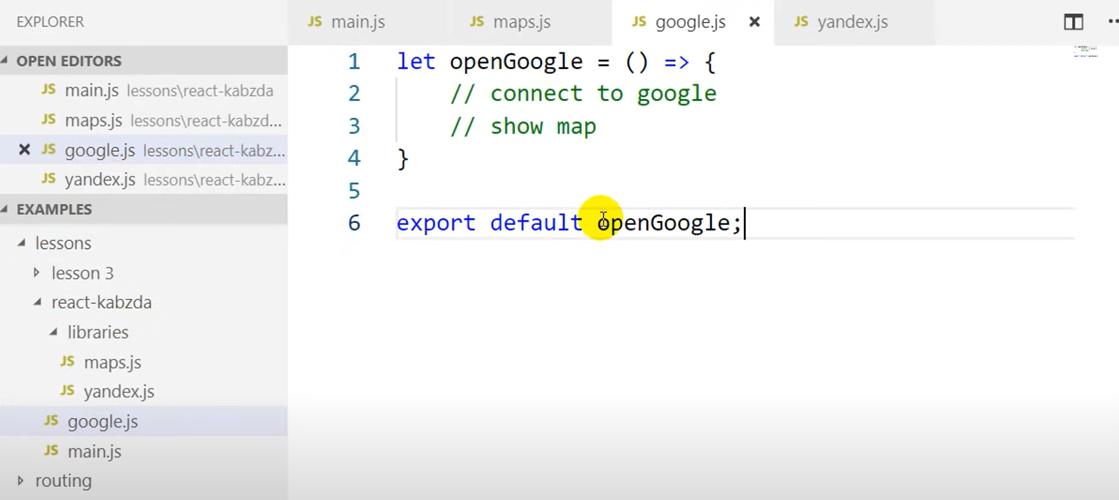
  );

}

# import\export – теория

# Вызываем функцию openGoogle из файла google.js. Для этого данную функцию нужно проимпортировать. Путь прописываем относительно файла в котором вызывается функция.

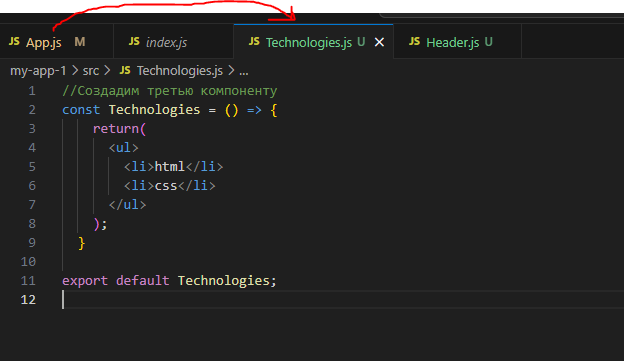




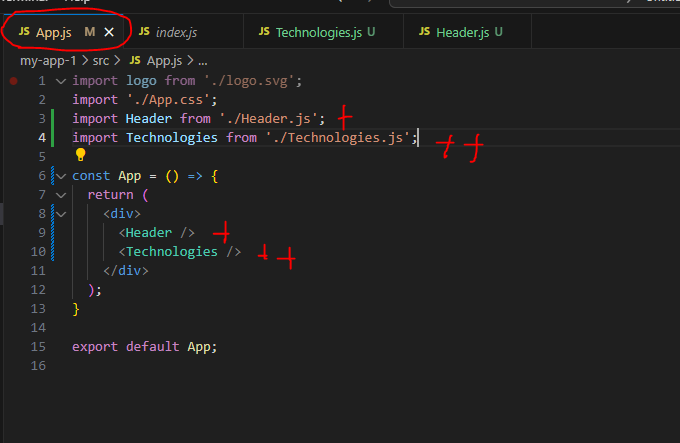
Но для того чтобы что-то импортировать нужно что-то экспортировать.

Для автоматического проставления import нужно установить плагин auto import!!!!!

ПЕРЕНОС КОМПОНЕНТЫ В ОТДЕЛЬНЫЙ JS ФАЙЛ!!!!



Импортируем его в App.js



В react приложении компоненты в отдельных файлах недоступны если не сделать import и export!!!

**Проблемы с default**

Может быть такая ситуация.

import logo from './logo.svg';

import './App.css';

//import Header from './Header.js';

import Technologies from './Header.js';

const App = () => {

  return (

    <div>

      <Technologies />

    </div>

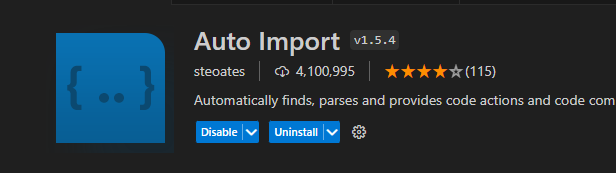
  );

}

export default App;

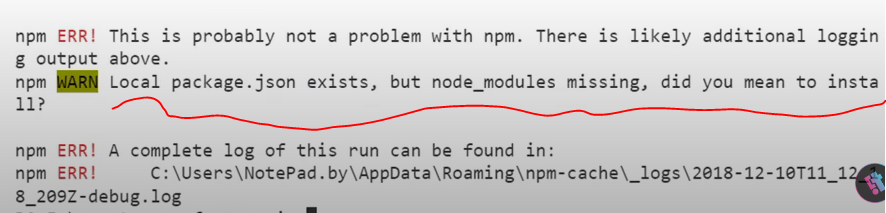
Т.е мы импортируем компоненту Technologies из файла Header.js и вызываем. Но данной компоненты в файле Header.js НЕТ. В результате будет выводится компонента из файла Header.js. ЭТО ПРОИСХОДИТ ИЗ-ЗА export. DEFAULT – это значит, что мы импортируем что-то. На месте Technologies можно написать что угодно

Установим расширение auto\_import

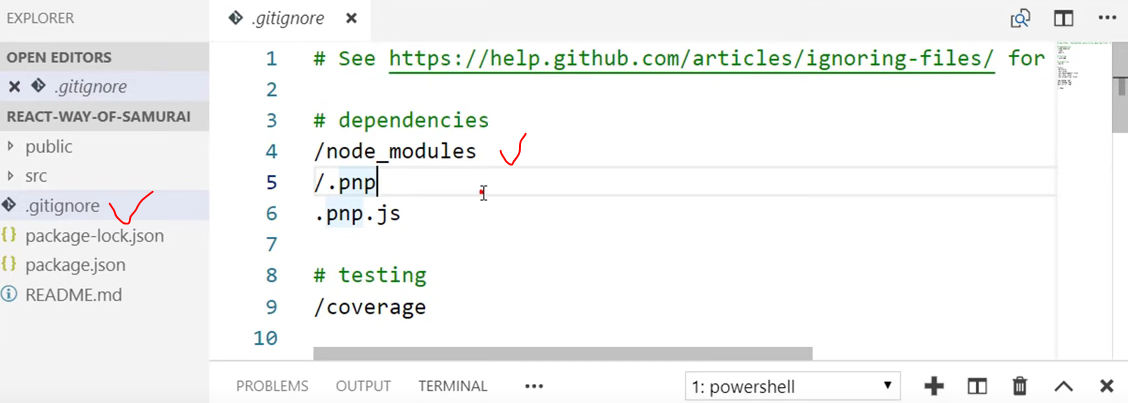


Почитать про module AMD require

Папку node\_modules не заливают на GitHub (никогда). Поэтому, когда клонируем проект и запускаем его при помощи команды git clone может выскочить ошибка.



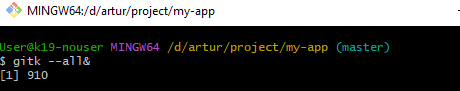
При заливке на GitHub название этой папки прописывается в .gitignore, чтобы она не попала в облачное хранилище.



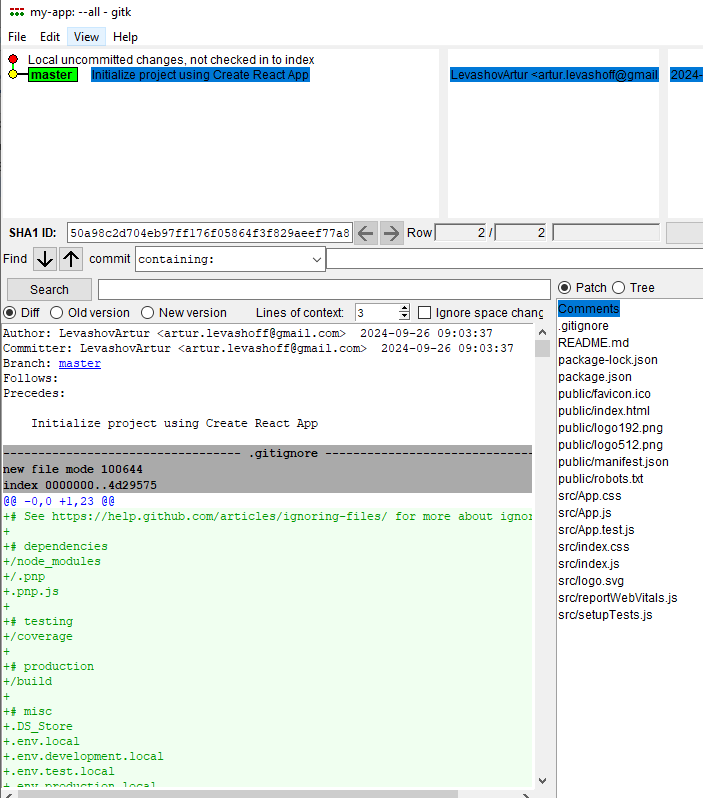
npm install – подгружаем папку node\_modules в проект

Файл package.json – самый основной, в нем прописываются все зависимости

Команда для просмотра веток коммита



Результат



Нужно скопировать ключ SHA1 ID при помощи команды ctrl+insert и вставить SHIFT+INSERT в командную строку git checkout (SHA1 ID)

**Css module**

В стандартном шаблоне react за стили отвечает файл APP.CSS и он подключен в APP.JS. Но что если мы хотим создать отдельные файлы css для каждого шаблона.

CSS модуль — это CSS файл, в котором все имена классов и анимаций имеют локальную область видимости по умолчанию.

Что это значит?

* Первым делом нужно наш css файл переименовать. Header.css -> Header.module.css
* Затем нужно подключить этот css к нашему файлу js

import styles from './Navbar.module.css'

После того как это было сделано, наши стили видоизменились. В файле они прописаны так:

.nav {

*grid-area*: n;

*background-color*: burlywood;

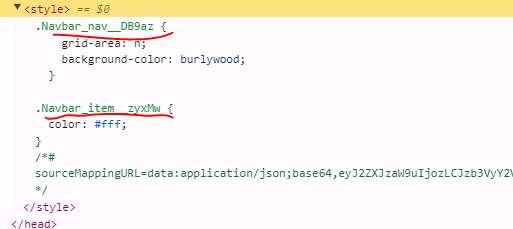
  }

.item {

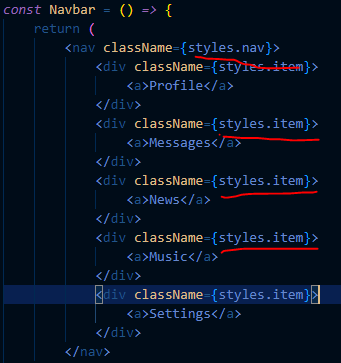
*color*: #fff;

}

Преобразовались в такой вид:



Соответственно подключать нужно в файл jsx таким образом



Оператор *import* в JavaScript файле загружает CSS файл и конвертирует его в объект. Каждое имя класса из CSS файла является свойством объекта (ключ). А значение этого ключа это уникальное имя класса, и уникальность обеспечивает то, что стили не «протекают» в другие компоненты. Вот пример хешированного имени класса: *\_header\_\_1OUvt*.

Т.е мы получаем объект

let styles = {

nav: “Navbar\_nav\_\_DB9az”,

item: “Navbar\_item\_\_zyxMw”

}

Желательно не использовать дефис в названии стилей в модулях поскольку необходимо тогда обращаться

let styles = {

nav-item: “Navbar\_nav\_\_DB9az”,

item-s: “Navbar\_item\_\_zyxMw”

}

{style[“nav-item”]}

* Если элементу принадлежат сразу 2 класса

<div class=”main strong”>

Css (если у элемента есть класс strong)

.main.strong {

font-weight: bold

}

На react это будет выглядеть так

<div className={`${styles.main} ${styles.strong}`}>

Или второй способ

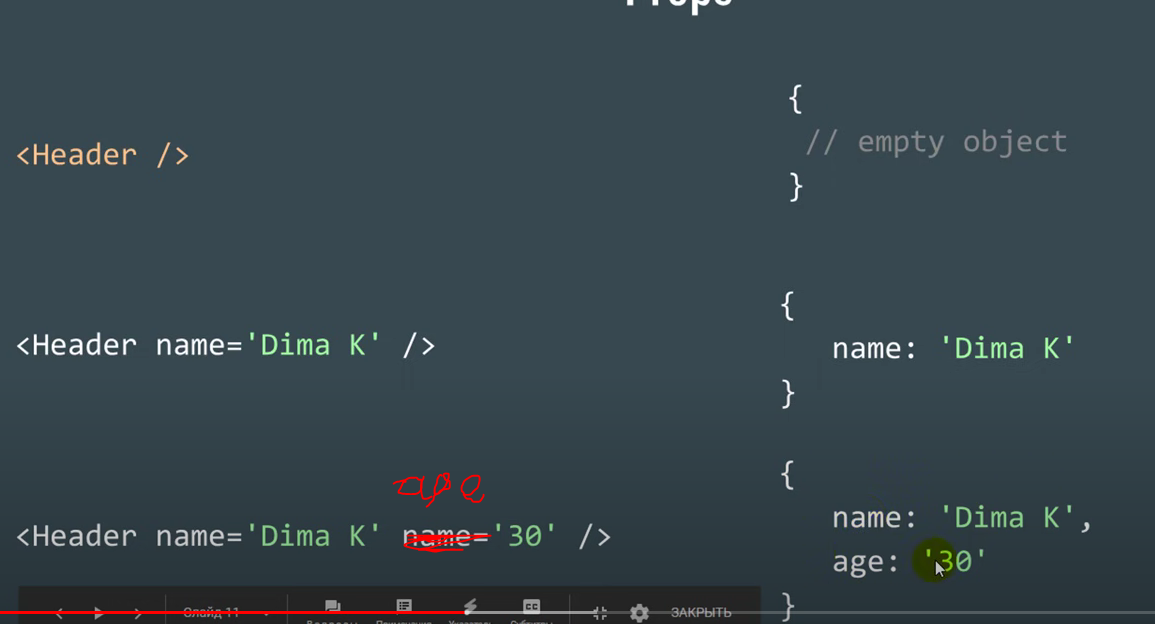
<div className={styles.dialog + ' ' + styles.active}>

# PROPS

По сути props является параметром для компоненты.



Компонента вызывается при помощи тега. Если у тега нет параметров, значит в качестве props будет пустой объект { }, если укажем параметры, то они станут свойствами объекта. Параметры можно указывать любые.



Пример



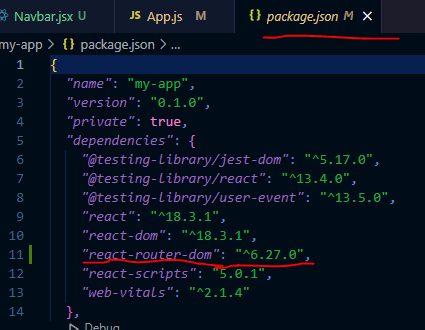
# Route, browser-router, маршрутизация

Routing – это маршрутизация, router – маршрутизатор

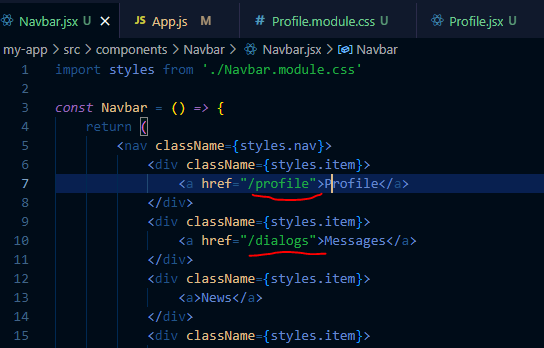
Разработчики CRA – create react app не включили в сборку routing, поэтому этот пакет нужно подгрузить в наш проект в папку node\_modules.

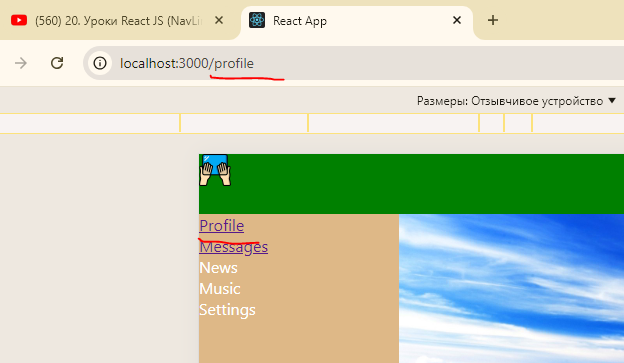


Сам пакет устанавливает в папку node\_modules, но при загрузке проекта в репозиторий папка node\_modules игнорируется, а следовательно и наш пакет тоже. При загрузке с репозитория route работать не будет и придется вручную подкачивать этот файл. Для того, чтобы решить эту проблему необходимо прописать -save. Эта команда занесет изменения в файл package.json. Это позволит при помощи одной команды npm i подгрузить все пакеты необходимые для работы программы.



Прописываем маршруты по которым мы сможем обратиться к нашим страницам





Теперь необходимо научить реакт понимать какую из перечисленных страниц нужно выбирать при клике по ссылке.



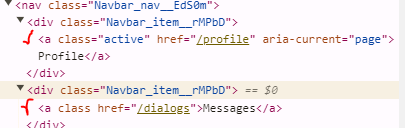
Обрамляем всю конструкцию обязательными тегами <BrowserRouter>, <Routes>. Они проимпортируются автоматически или прописать как написаны в import. Далее необходимо прописать path=”” и сам element=””. Данная конструкция отлавливает адрес в адресной строке, который мы прописали в теге <a>

НО ПРИ ПЕРЕХОДЕ НА ДРУГУЮ СТРАНИЦУ САЙТ ПЕРЕЗАГРУЖАЕТСЯ. ДЛЯ ТОГО ЧТОБЫ ЭТОГО ИЗБЕЖАТЬ НУЖНО ВМЕСТО <a> ПРОПИСАТЬ <NAVLINK>

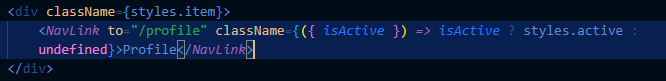
Для использования NavLink необходим import.



Но в режиме разработчика <navlink> отображается как <a>



Если хотим добавить активное состояние ссылке



Css

.item a.active {

*color*: gold;

}

# Route exact

Мы прописали новые ссылки

 <div className={styles.dialog + ' ' + styles.active}>

                    <*NavLink* to="/dialogs/1">Andrey</*NavLink*>

                </div>

                <div className={styles.dialog}>

                    <*NavLink* to="/dialogs/2">Artur</*NavLink*>

                </div>

                <div className={styles.dialog}>

                    <*NavLink* to="/dialogs/3">Katya</*NavLink*>

                </div>

                <div className={styles.dialog}>

                    <*NavLink* to="/dialogs/4">Victor</*NavLink*>

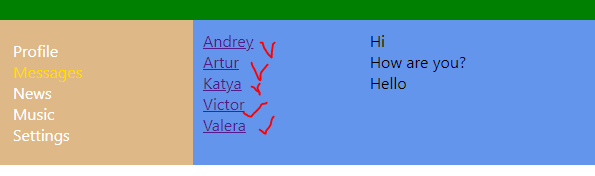
                </div>

                <div cclassName={styles.dialog}>

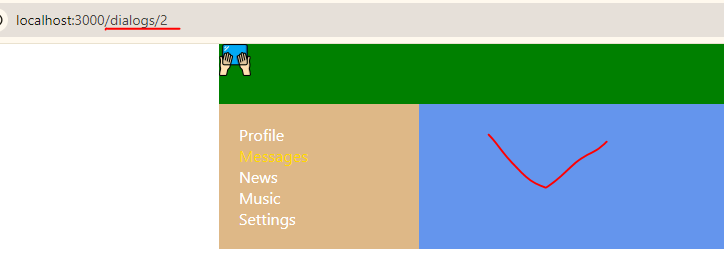
                    <*NavLink* to="/dialogs/5">Valera</*NavLink*>

                </div>

Но при клике по ним весь контент пропадает



Результат



Это стандартное поведение, поскольку на отрисовку мы прописали такую ссылку

        <*Routes*>

          <*Route* path="/dialogs" element={<*Dialogs*/>}/>

          <*Route* path="/profile" element={<*Profile*/>}/>

        </*Routes*>

Наша ссылка выглядит так: /dialogs/2 а реакт пытается найти ссылку /dialogs. Раньше для достижения такого эффекта требовался параметр exact

<*Route exact* path="/dialogs" element={<*Dialogs*/>}/>

Теперь этот функционал по умолчанию. Если мы хотим, чтобы ссылки не пропадали нужно написать так:

<*Route* path="/dialogs/\*" element={<*Dialogs*/>}/>

Это значит, что route отловит начало ссылки, а \* означает что продолжение ссылки может быть любым.

**Разбиение на компоненты dialogs**

<div className={styles.dialog}>

                    <*NavLink* to="/dialogs/2">Artur</*NavLink*>

                </div>

                <div className={styles.dialog}>

                    <*NavLink* to="/dialogs/3">Katya</*NavLink*>

                </div>

                <div className={styles.dialog}>

                    <*NavLink* to="/dialogs/4">Victor</*NavLink*>

                </div>

                <div cclassName={styles.dialog}>

                    <*NavLink* to="/dialogs/5">Valera</*NavLink*>

                </div>

Вот такой повторяющийся код можно разбить на компоненты.

import styles from './Dialogs.module.css';

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

*const* DialogItem = (*props*) *=>* {

*let* path = "/dialogs/" + *props*.id;

    return (

        <div className={styles.dialog + ' ' + styles.active}>

{\*/вместо path можно написать так to={"/dialogs/" + *props*.id}>\*/}

            <*NavLink* to={path}>{*props*.name}</*NavLink*>

        </div>

    )

}

*const* Dialogs = (*props*) *=>* {

    return (

        <div className={styles.dialogs}>

            <div className={styles["dialog-items"]}>

                <*DialogItem* name="Andrey" id="1" />

                <*DialogItem* name="Artur" id="2" />

                <*DialogItem* name="Katya" id="3" />

                <*DialogItem* name="Victor" id="4" />

                <*DialogItem* name="Valera" id="5" />

            </div>

            <div className={styles.messages}>

                <div className={styles.message}>Hi</div>

                <div className={styles.message}>How are you?</div>

                <div className={styles.message}>Hello</div>

            </div>

        </div>

    )

}

export default Dialogs;

# UI - BLL, выносим данные в сторону

Все данные в props должны подгружаются с БД. На данный момент мы прописали их фиксировано при вызове компоненты. Попытаемся вытянуть их в объект пытаясь создать имитацию получения данных из таблицы в БД.

BLL – Business Logic Layer или просто данные

UI – user interface

*const* Dialogs = (*props*) *=>* {

*let* dialogsData = [

        {id: 1, name: 'Andrey'},

        {id: 2, name: 'Artur'},

        {id: 3, name: 'Katya'},

        {id: 4, name: 'Victor'},

        {id: 5, name: 'Valera'},

    ];

*let* messagesData = [

        {id: 1, message: 'Hi'},

        {id: 2, message: 'How are you?'},

        {id: 3, message: 'Hello'},

    ];

    return (

        <div className={styles.dialogs}>

            <div className={styles["dialog-items"]}>

                <*DialogItem* name={dialogsData[0].name} id={dialogsData[0].id} />

                <*DialogItem* name={dialogsData[1].name} id={dialogsData[1].id} />

                {/\*<DialogItem name="Andrey" id="1" />

                <DialogItem name="Artur" id="2" />

                <DialogItem name="Katya" id="3" />

                <DialogItem name="Victor" id="4" />

                <DialogItem name="Valera" id="5" />\*/}

            </div>

            <div className={styles.messages}>

                <*Message* message={messagesData[0].message}/>

                <*Message* message={messagesData[1].message}/>

                <*Message* message={messagesData[2].message}/>

                {/\*<Message message="Hi"/>

                <Message message="How are you?"/>

                <Message message="Hello"/>\*/}

                {/\*<div className={styles.message}>Hi</div>

                <div className={styles.message}>How are you?</div>

                <div className={styles.message}>Hello</div>\*/}

            </div>

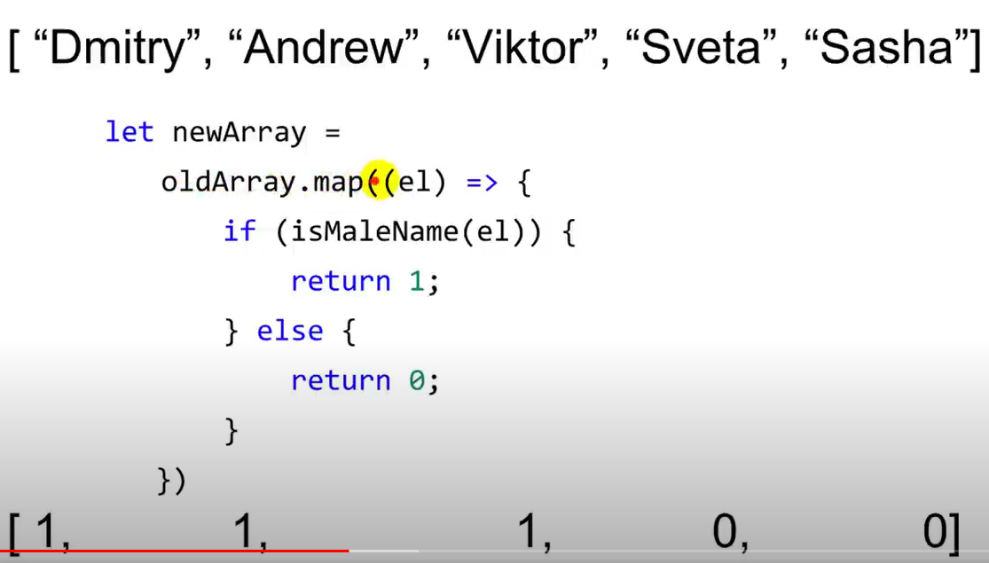
        </div>

    )

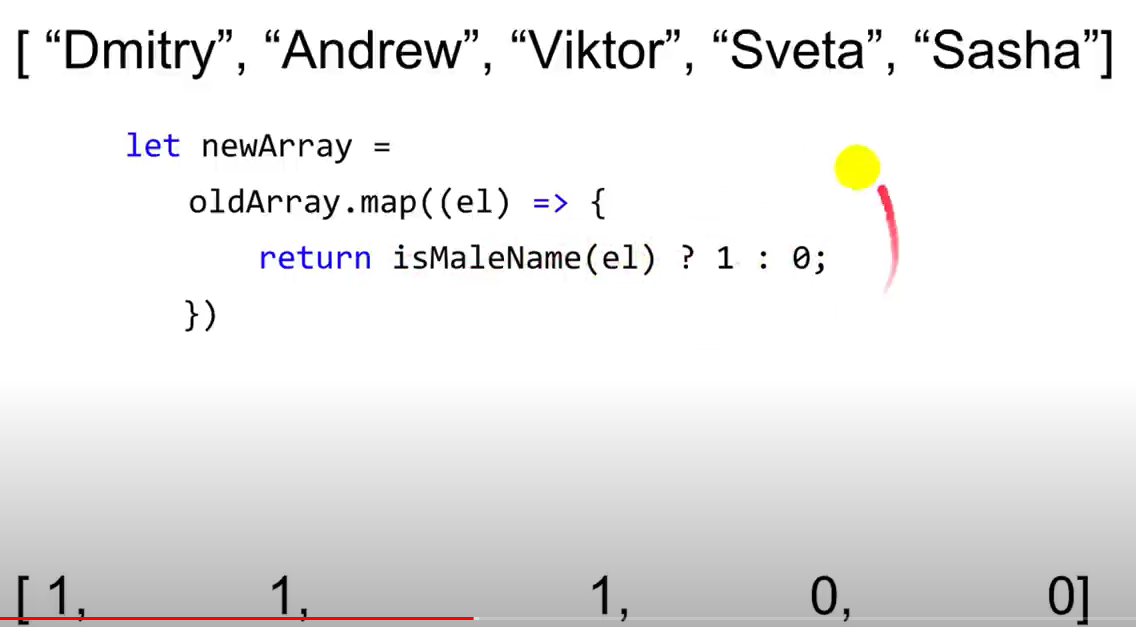
}

# Метод массива – map

Данный метод необходим для преобразования элементов массива



Преобразование имен в пол(жен = 0 или муж = 1). Тоже самое



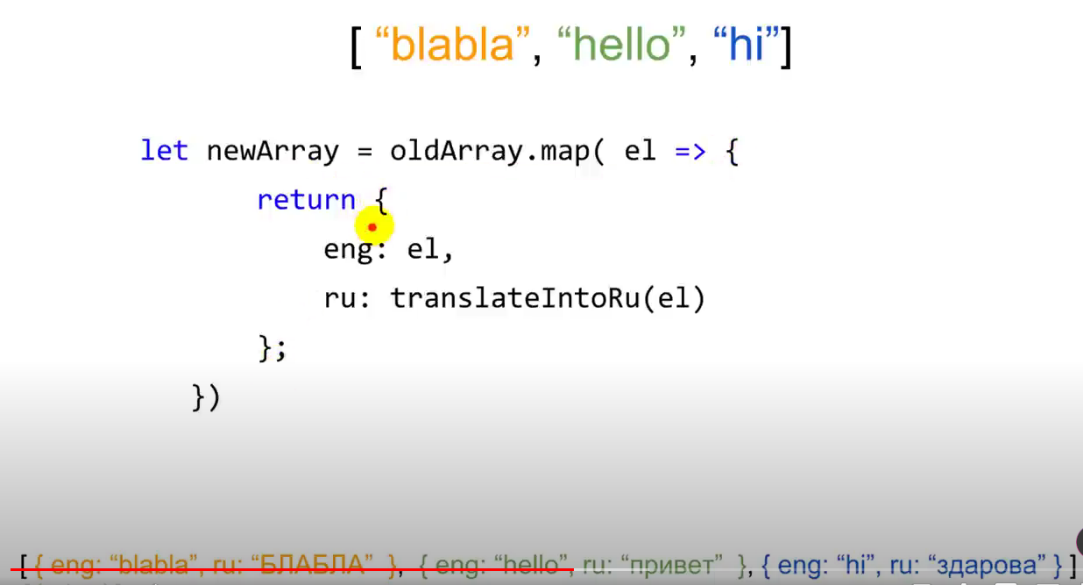
Следующее преобразование



Аналогичный синтаксис



У нас имеется массив строк, а должны получить массив объектов



Самый частый вариант: имеется массив объектов, а нужно получить массив jsx элементов



import styles from './Dialogs.module.css';

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

*const* DialogItem = (*props*) *=>* {

*let* path = "/dialogs/" + *props*.id;

    return (

        <div className={styles.dialog + ' ' + styles.active}>

            <*NavLink* to={path}>{*props*.name}</*NavLink*>

        </div>

    )

}

*const* Message = (*props*) *=>* {

    return (

        <div className={styles.message}>{*props*.message}</div>

    )

}

*const* Dialogs = (*props*) *=>* {

    {/\*Данные\*/}

*let* dialogsData = [

        {id: 1, name: 'Andrey'},

        {id: 2, name: 'Artur'},

        {id: 3, name: 'Katya'},

        {id: 4, name: 'Victor'},

        {id: 5, name: 'Valera'},

    ];

*let* messagesData = [

        {id: 1, message: 'Hi'},

        {id: 2, message: 'How are you?'},

        {id: 3, message: 'Hello'},

    ];

    {/\*Преобразуем массив объектов в массив jsx элементов dialog \*/}

*let* dialogsElements = dialogsData.map( (*dialog*) *=>* {

        return (

            <*DialogItem* name={*dialog*.name} id={*dialog*.id} />

        )

    });

    {/\*Преобразуем массив объектов в массив jsx элементов dialog \*/}

*let* messagesElements = messagesData.map( (*message*) *=>* {

        return (

            <*Message* message={*message*.message}/>

        )

    })

    return (

        <div className={styles.dialogs}>

            <div className={styles["dialog-items"]}>

                {dialogsElements}

                {/\*

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

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

                <DialogItem name="Andrey" id="1" />

                <DialogItem name="Artur" id="2" />

                <DialogItem name="Katya" id="3" />

                <DialogItem name="Victor" id="4" />

                <DialogItem name="Valera" id="5" />\*/}

            </div>

            <div className={styles.messages}>

                {messagesElements}

                {/\*

                <Message message={messagesData[0].message}/>

                <Message message={messagesData[1].message}/>

                <Message message={messagesData[2].message}/>

                <Message message="Hi"/>

                <Message message="How are you?"/>

                <Message message="Hello"/>

                <div className={styles.message}>Hi</div>

                <div className={styles.message}>How are you?</div>

                <div className={styles.message}>Hello</div>\*/}

            </div>

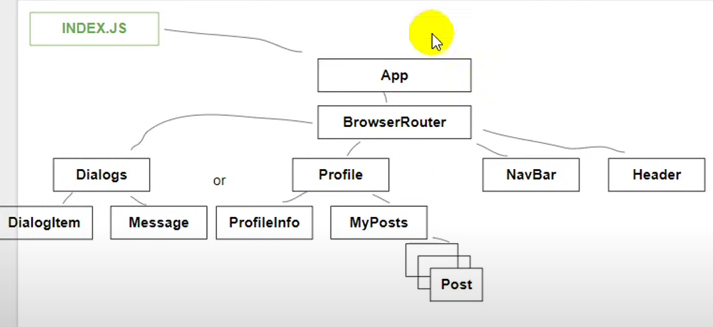
        </div>

    )

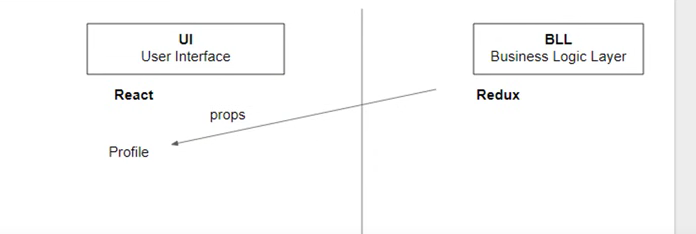
}

export default Dialogs;

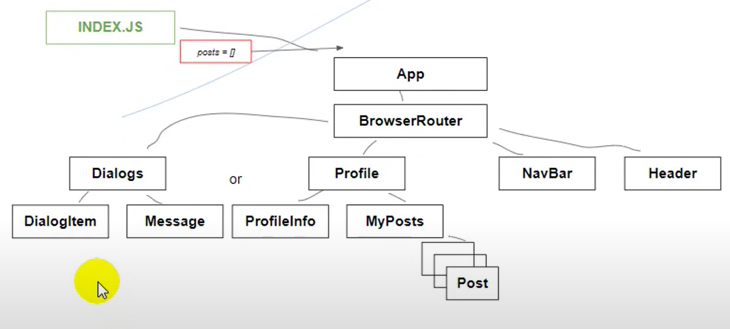
**Структура проекта**



Архитектура приложения



Так получается, что наши данные находятся внутри MyPosts. Это нарушение архитектуры приложения, поэтому необходимо вынести в index.js



**Вынос данных в index.js**

Наша задача освободить компоненты от данных!!! Данные должны приходить в компоненту.

1. Перенесем данные из MyPosts на уровень выше в Profile.

При вызове компоненты MyPosts написали атрибут postData и передали ему массив объектов. В дальнейшем этот массив попадет в нашу компоненту через props

*const* Profile = () *=>* {

*let* postData = [

        {id: 1, message: 'Hi, how are you', count: 15},

        {id: 2, message: 'It s my first post', count: 20}

    ];

    return (

        <div>

            <*ProfileInfo*/>

            <*MyPosts* postData={postData}/>

      </div>

    )

}

MyPosts.Ранее обращались к массиву данных напрямую. Теперь получаем в props массив объектов и обращаемся к нашему массиву.

*props*.postData.map(………

*const* MyPosts = (*props*) *=>* {

    {/\*Преобразуем массив объектов в массив jsx элементов\*/}

*let* postsElements = *props*.postData.map( (*post*) *=>* {

        return(

            <*Post* message={*post*.message} count={*post*.count} />

        )

    });

    return (

            <div className={styles["posts-block"]}>

                <h3>My posts</h3>

                <div>

                    <div>

                        <textarea></textarea>

                    </div>

                    <div>

                        <button>Add post</button>

                    </div>

                    <div>

                        <button>Remove</button>

                    </div>

                </div>

                <div className={styles.posts}>

                    {postsElements}

                    {/\*<Post message={postData[0].message} count={postData[0].count} />

                    <Post message={postData[1].message} count={postData[1].count} />\*/}

                </div>

            </div>

    )

}

1. Перенесем данные из Profile на уровень выше в App.js

Обращаемся к атрибуту postData через props

*const* Profile = (*props*) *=>* {

    return (

        <div>

            <*ProfileInfo*/>

            <*MyPosts* postData={*props*.postData}/>

      </div>

    )

}

В самом атрибуте postData прописываем наш массив данных

<*Profile* postData={postData}/>

*const* App = () *=>* {

*let* postData = [

    {id: 1, message: 'Hi, how are you', count: 15},

    {id: 2, message: 'It s my first post', count: 20}

  ];

  return (

    <*BrowserRouter*>

      <div className='app-wrapper'>

        <*Header*/>

        <*Navbar*/>

        <div className="app-wrapper-content">

        <*Routes*>

          <*Route* path="/dialogs" element={<*Dialogs*/>}/>

          <*Route* path="/profile" element={<*Profile* postData={postData}/>}/>

        </*Routes*>

        </div>

      </div>

    </*BrowserRouter*>

)}

1. Финальная часть переносим данные из App.js в index.js

*const* App = (*props*) *=>* {

  return (

    <*BrowserRouter*>

      <div className='app-wrapper'>

        <*Header*/>

        <*Navbar*/>

        <div className="app-wrapper-content">

        <*Routes*>

          <*Route* path="/dialogs" element={<*Dialogs*/>}/>

          <*Route* path="/profile" element={<*Profile* postData={*props*.postData}/>}/>

        </*Routes*>

        </div>

      </div>

    </*BrowserRouter*>

)}

Index.js

*let* postData = [

  {id: 1, message: 'Hi, how are you', count: 15},

  {id: 2, message: 'It s my first post', count: 20}

];

*const* root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <*React.StrictMode*>

    <*App* postData={postData} />

  </*React.StrictMode*>

);

ПРОЧИТАТЬ ПРО DEBUGGER!!!

Итоговый вариант переноса всех данных

{/\*ПОСТЫ \*/}

*let* postData = [

  {id: 1, message: 'Hi, how are you', count: 15},

  {id: 2, message: 'It s my first post', count: 20}

];

{/\*ДИАЛОГИ\*/}

*let* dialogsData = [

  {id: 1, name: 'Andrey'},

  {id: 2, name: 'Artur'},

  {id: 3, name: 'Katya'},

  {id: 4, name: 'Victor'},

  {id: 5, name: 'Valera'},

];

{/\*СООБЩЕНИЯ\*/}

*let* messagesData = [

  {id: 1, message: 'Hi'},

  {id: 2, message: 'How are you?'},

  {id: 3, message: 'Hello'},

];

*const* root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

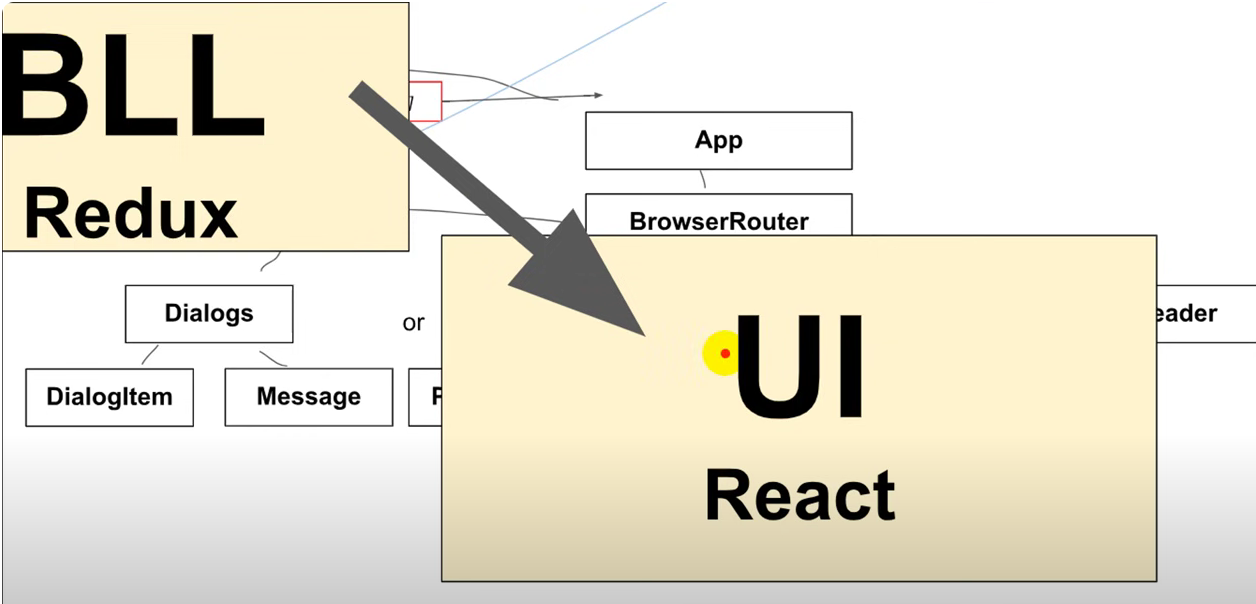
  <*React.StrictMode*>

    <*App* postData={postData} dialogsData={dialogsData} messagesData={messagesData} />

  </*React.StrictMode*>

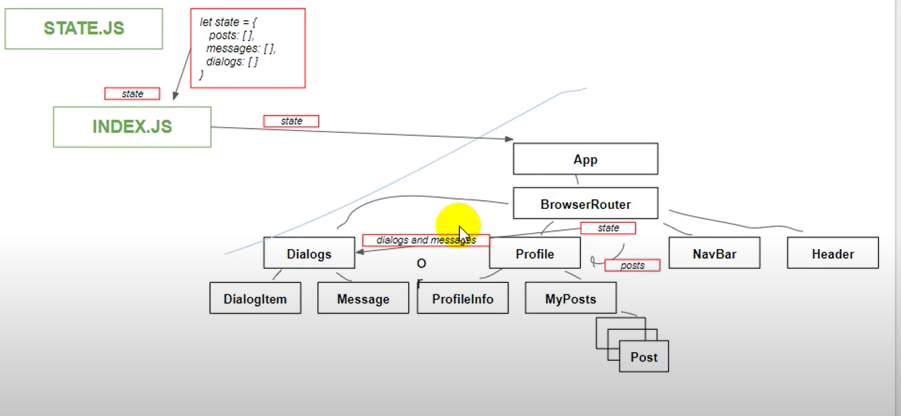
);

Обновленная структура



**Упаковка данных в state**

Вынесем наши входные данные в отдельный файл state.js. Также объединим наши данные в один объект, т.е. создадим объект, свойства которого будут массивами. Структура нашего state.js : для каждой страницы отдельные данные. Таким образом нужно создать отдельные подобъекты, которые будут служить неким разделителем для отдельных страниц.



state.js

*let* state = {

    profilePage: {

      postData: [

        {id: 1, message: 'Hi, how are you', count: 15},

        {id: 2, message: 'It s my first post', count: 20}

      ]

    },

    dialogsPage: {

      dialogsData: [

        {id: 1, name: 'Andrey'},

        {id: 2, name: 'Artur'},

        {id: 3, name: 'Katya'},

        {id: 4, name: 'Victor'},

        {id: 5, name: 'Valera'},

      ],

      messagesData: [

        {id: 1, message: 'Hi'},

        {id: 2, message: 'How are you?'},

        {id: 3, message: 'Hello'},

      ]

    }

}

export default state;

В index.js импортируем данные из файла state.js и передаем в качестве параметра компоненте App

import state from './redux/state.js';

*const* root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <*React.StrictMode*>

    <*App* appState={state} />

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

  </*React.StrictMode*>

);

В app.js через props получаем наш объект данных, который содержится в атрибуте appState. В компоненте dialogs указываем атрибут state в который передаем свойство dialogsPage, которое в свою очередь содержит набор массивов данных необходимых для работы страницы dialog

        <*Routes*>

          {/\*<Route path="/dialogs" element={<Dialogs dialogsData={props.appState.messagesPage.dialogsData} messagesData={props.appState.messagesPage.messagesData}/>}/>

          <Route path="/profile" element={<Profile postData={props.appState.profilePage.postData}/>}/>\*/}

          <*Route* path="/dialogs" element={<*Dialogs* state={*props*.appState.dialogsPage} />}/>

          <*Route* path="/profile" element={<*Profile* state={*props*.appState.profilePage}/>}/>

        </*Routes*>

В Dialogs.jsx получаем данные в props:

dialogsPage: {

      dialogsData: [

        {id: 1, name: 'Andrey'},

        {id: 2, name: 'Artur'},

        {id: 3, name: 'Katya'},

        {id: 4, name: 'Victor'},

        {id: 5, name: 'Valera'},

      ],

      messagesData: [

        {id: 1, message: 'Hi'},

        {id: 2, message: 'How are you?'},

        {id: 3, message: 'Hello'},

      ]

    }

Прописываем *props*.state.dialogsData И *props*.state.messagesData

*const* Dialogs = (*props*) *=>* {

    {/\*Преобразуем массив объектов в массив jsx элементов dialog \*/}

*let* dialogsElements = *props*.state.dialogsData.map( (*dialog*) *=>* {

        return (

            <*DialogItem* name={*dialog*.name} id={*dialog*.id} />

        )

    });

    {/\*Преобразуем массив объектов в массив jsx элементов dialog \*/}

*let* messagesElements = *props*.state.messagesData.map( (*message*) *=>* {

        return (

            <*Message* message={*message*.message}/>

        )

    })

    return (

        <div className={styles.dialogs}>

            <div className={styles["dialog-items"]}>

                {dialogsElements}

            </div>

            <div className={styles.messages}>

                {messagesElements}

                {/\*

            </div>

        </div>

    )

}

export default Dialogs;

# onClick, ref, VirtualDOM

Есть возможность указать обработчик события для элемента внутри через анонимную функцию callback

<button onClick={ () *=>* {alert('js.com')} }>Add post</button>

Также можно написать функцию отдельно, а затем вызвать

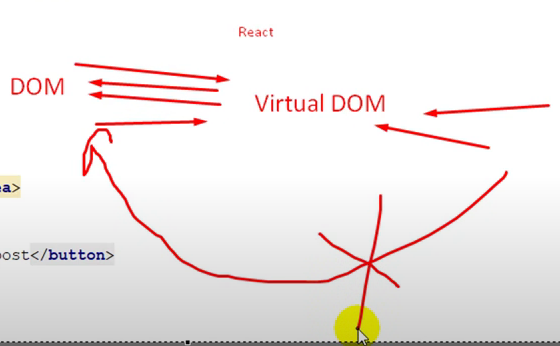
*let* addPost = () *=>* {

        alert('js.com');

    }

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

В react мы не можем обращать в DOM напрямую, т.к. есть VirtualDOM



В нативном JS мы могли через document получить какой-то элемент, которому прописан класс или id.

В js:

<textarea id="text"></textarea>

*let* addPost = () *=>* {

*let* text = document.getElementById('text');

        alert(text);

    }

В react так нельзя делать. Такие ссылки прописываются через ref.

1. Необходимо на странице прописать:

import React from 'react';

2. Необходимо создать ref

*let* newPostElement = React.createRef();

3. Затем прописать ref у элемента

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

4. В самой функции находим этот элемент

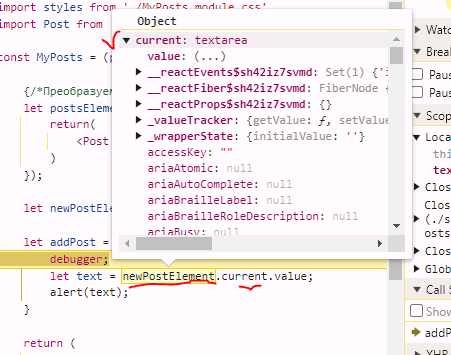
*let* addPost = () *=>* {

        debugger;

*let* text = newPostElement.current.value;

        alert(text);

    }



**Прокидывание callback через props**

Необходимо прописать функцию, которая будет получать текст сообщения из textarea и передавать в базу. Данная функция должна находится на стороне BLL и перекидываться через props в UI.

В state.js.

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

*let* state = {

    profilePage: {

      postData: [

        {id: 1, message: 'Hi, how are you', count: 15},

        {id: 2, message: 'It s my first post', count: 20}

      ]

    },

    dialogsPage: {

      dialogsData: [

        {id: 1, name: 'Andrey'},

        {id: 2, name: 'Artur'},

        {id: 3, name: 'Katya'},

        {id: 4, name: 'Victor'},

        {id: 5, name: 'Valera'},

      ],

      messagesData: [

        {id: 1, message: 'Hi'},

        {id: 2, message: 'How are you?'},

        {id: 3, message: 'Hello'},

      ]

    },

    navbar: {

      friends: [

        {id: 1, name: "Vasya"},

        {id: 2, name: "Sasha"},

        {id: 3, name: "Sveta"},

      ]

    }

}

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

*let* newPost = {

    id: 5,

    message: *postMessage*,

    count: 0

  };

  state.profilePage.postData.push(newPost);

}

Объявили функцию и сразу export. Это не default export поэтому Import данной функции будет немного отличаться. Задача данной функции по клику получить строку которая приходит в параметр *postMessage* затем нужно создать объект, который в дальнейшем будет передаваться в общий объект данных state. Данные будут передаваться в массив state.profilePage.postData.push(newPost); Структура создаваемого объекта newPost должна в точности совпадать с объектами которые находятся в масииве postData.

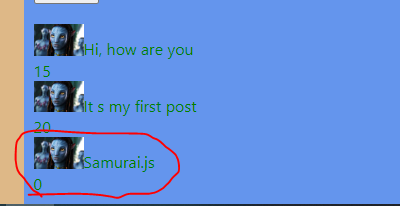
В index.js

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

При вызове до render

addPost('Samurai.js');

Результат



ВСЕ ЧТО НАХОДИТСЯ В STATE.JS МОЖНО ИМПОРТИРОВАТЬ В INDEX.JS, А ДАЛЬШЕ ПРОКИДЫВАЕМ ЧЕРЕЗ PROPS. В ДРУГИЕ ФАЙЛЫ ИМПОРТ ЛУЧШЕ НЕ ДЕЛАТЬ!!!!!!!!!!!!!!

Прокинули в App props

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

Прокинули в Profile

          <*Route* path="/profile" element={<*Profile*

           state={*props*.appState.profilePage}

           addPost={*props*.addPost} />}/>

Прокинули в MyPosts

<*MyPosts*

             postData={*props*.state.postData}

             addPost={*props*.addPost} />

В MyPosts вызвали данную функцию

*let* addPost = () *=>* {

*let* text = newPostElement.current.value;

*props*.addPost(text);

    }

Наша функция получает данные из textarea и записывает в базу. Данные с базы прокидываются на нужную страницу, но проблема в том, что новый элемент не отрисовывается. Это происходит потому, что в файле index.js отрисовка render происходит единожды.

*const* root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <*React.StrictMode*>

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

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

  </*React.StrictMode*>

);

Поэтому необходимо сделать так, чтобы при изменении в stage происходила повторная отрисовка в index.js. И так на каждое изменение в stage.

Для этого нужно написать функцию, которая будет повторно вызывать наш render

В index.js

*const* renderEntireTree = () *=>* {

  root.render(

    <*React.StrictMode*>

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

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

    </*React.StrictMode*>

  );

}

И по хорошему счету нужно вызвать эту функцию в stage.js. Но есть одна проблема. Stage.js мы уже экспортировали и импортировали в index.js. Если мы захотим экспортировать и затем импортировать нашу функцию в stage.js, то возникнет циклическая зависимость, а это нехорошо))

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

*let* newPost = {

    id: 3,

    message: *postMessage*,

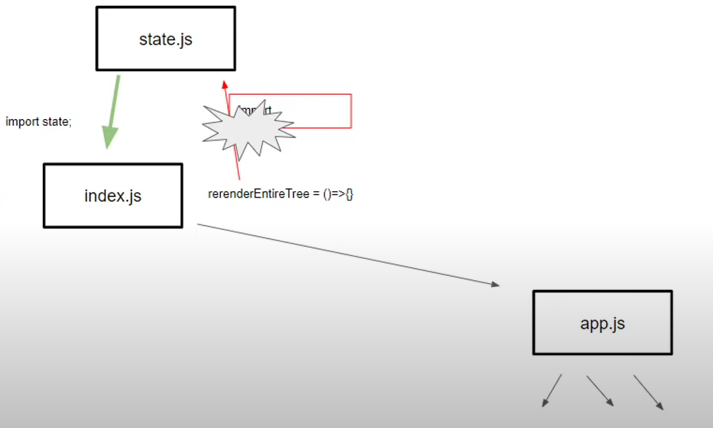
    count: 0

  };

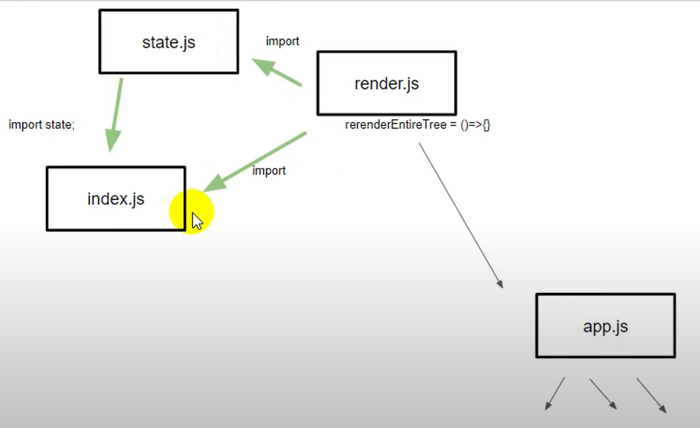
  state.profilePage.postData.push(newPost);

  renderEntireTree();

}



Для решения это проблемы нужно ввести отдельный файл render.js



В таком случает отсутствует циклическая зависимость stage.js не зависит от index.js -> зависит от state.js т.к. импортируется. Но оба они зависят от render.js

ИТОГО

Render.js

/\*Данный файл необходим для рендеринга, эти действие необходимо для того, чтобы небыло цикличности между index.js и state.js и мы могли импортировать функцию renderEntireTree в index.js для дальнейшей отрисовки контента, а также импортировать renderEntireTree в state.js для того чтобы повторно запускать root.render в случае какого либо изменения в state.js  \*/

//----------------Все необходимые импорты из index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App.js';

import reportWebVitals from './reportWebVitals.js';

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

//////////////////Все необходимые импорты из index.js

//--------------Удаляем state чтобы небыло цикличности между render.js и import.js и прокинем в props

//import state from './redux/state.js';

/////////////////Удаляем state чтобы небыло цикличности между render.js и import.js

//-------------Функция для повтрного запуска отрисовки

export *let* renderEntireTree = (*state*) *=>* {

*const* root = ReactDOM.createRoot(document.getElementById('root'));

  root.render(

    <*React.StrictMode*>

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

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

    </*React.StrictMode*>

  );

}

//////////////////Функция для повтрного запуска отрисовки

Index.js

import state from './redux/state.js';

//------------Импорт функции из render.js

import reportWebVitals from './reportWebVitals';

import { renderEntireTree } from './render.js';

///////////////Импорт функции из render.js

renderEntireTree(state);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

state.js

//-------------Импорт функции из render.js

import { renderEntireTree } from "../render.js";

///////////////Импорт функции из render.js

*let* state = {

    profilePage: {

      postData: [

        {id: 1, message: 'Hi, how are you', count: 15},

        {id: 2, message: 'It s my first post', count: 20}

      ]

    },

    dialogsPage: {

      dialogsData: [

        {id: 1, name: 'Andrey'},

        {id: 2, name: 'Artur'},

        {id: 3, name: 'Katya'},

        {id: 4, name: 'Victor'},

        {id: 5, name: 'Valera'},

      ],

      messagesData: [

        {id: 1, message: 'Hi'},

        {id: 2, message: 'How are you?'},

        {id: 3, message: 'Hello'},

      ]

    },

    navbar: {

      friends: [

        {id: 1, name: "Vasya"},

        {id: 2, name: "Sasha"},

        {id: 3, name: "Sveta"},

      ]

    }

}

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

*let* newPost = {

    id: 3,

    message: *postMessage*,

    count: 0

  };

  state.profilePage.postData.push(newPost);

  //Запускается функция, которая будет повторно отрисовывать render с учетом изменившихся данных в state.js в качестве параметра передали state

  renderEntireTree(state);

}

export default state;

Мы практически приблизились к FLUX архитектуре

**FLUX ->react (архитектура)**

Мы не имеем право изменять UI, не изменив BLL, но в нашем примере мы пока, что меняем UI.

