МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
филиал «РКТ» МАИ в г. Химки Московской области

Специальность 09.02.03 «Программирование в компьютерных системах»

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

ПМ.02«Разработка и администрирование баз данных»

Студент

Группы МП-32____Казнаков К.В._____ / (_____)

Руководитель

практики от организации _____ / (_____)

Руководитель

практики от филиала Шумаев А.Ю. / (_____)

2020г.

ПРОГРАММА ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ

По специальности 09.02.03 «Программирование в компьютерных системах»

Дата	Наименование выполняемых работ	Количество дней практики
11.06.20	Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику.	4
12.06.20	Описание структуры сети, в которой может функционировать разработанная база данных.	4
13.06.20	Описание сетевого оборудования необходимого для работы с базой данных.	4
15.06.20	Описание программного обеспечения необходимого для работы с базой данных по сети.	4
16.06.20-18.06.20	Разработка клиентских интерфейсов для клиент-серверных приложений.	16
20.06.20	Создание инфокоммуникационной системы. Выполнение сетевых настроек для взаимодействия с СУБД.	4
22.06.20-24.06.20	Создание концептуальной модели данных. Создание логической модели данных. Создание физической модели данных. Описание механизмов обеспечения целостности базы данных.	16
25.06.20-01.07.20	Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных. Структурирование запросов базы данных. Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных.	50
02.07.20-03.07.20	Определение способов управления правами пользователей. Описание распределения прав пользователей и управления ими в базе данных.	12
03.07.20-04.07.20	Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных.	12
06.07.20	Составление отчёта по практике	6
07.07.20	Подготовка к защите отчёта по практике.	6
08.07.20	Итоговая аттестация по производственной практике - зачёт	6

Руководитель практики от филиала «РКТ» МАИ преподаватель

Шумаев А.Ю.

Дата _____ _____

(подпись)

АТТЕСТАЦИОННЫЙ ЛИСТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

студент Казнаков Кирилл Владимирович

ФИО

обучающийся на 3-ем курсе по специальности СПО 09.02.03 «Программирование в компьютерных системах»

успешно прошел производственную практику по профессиональному модулю **ПМ.02 «Разработка и администрирование баз данных»**

в объеме 144 часа с 11.06.2020г. по 08.07.2020г.

в организации филиала «РКТ» МАИ

Виды и качество выполнения работ

Виды работ, выполненных обучающимся во время практики	Объем работ	Качество выполнения работ в соответствии с технологией и (или) требованиями организации, в которой проходила практика
Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику.	4	
Описание структуры сети, в которой может функционировать разработанная база данных.	4	
Описание сетевого оборудования необходимого для работы с базой данных.	4	
Описание программного обеспечения необходимого для работы с базой данных по сети.	4	
Разработка клиентских интерфейсов для клиент-серверных приложений.	16	
Создание инфокоммуникационной системы. Выполнение сетевых настроек для взаимодействия с СУБД.	4	
Создание концептуальной модели данных. Создание логической модели данных. Создание физической модели данных. Описание механизмов обеспечения целостности базы данных.	16	
Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных. Структурирование запросов базы данных. Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных.	50	

Определение способов управления правами пользователей. Описание распределения прав пользователей и управления ими в базе данных.	12	
Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных.	12	
Составление отчёта по практике	6	
Подготовка к защите отчёта по практике.	6	
Итоговая аттестация по производственной практике - зачёт	6	

Руководитель практики от филиала «РКТ» МАИ Шумаев А.Ю.

Дата_____ _____ _____

подпись *Расшифровка подписи*

ДНЕВНИК ПРОХОЖДЕНИЯ ПРАКТИКИ

Дата	Наименование выполняемых работ	Подпись руководителя
11.06.20	Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику.	
12.06.20	Описание структуры сети, в которой может функционировать разработанная база данных.	
13.06.20	Описание сетевого оборудования необходимого для работы с базой данных.	
15.06.20	Описание программного обеспечения необходимого для работы с базой данных по сети.	
16.06.20-18.06.20	Разработка клиентских интерфейсов для клиент-серверных приложений.	
20.06.20	Создание инфокоммуникационной системы. Выполнение сетевых настроек для взаимодействия с СУБД.	
22.06.20-24.06.20	Создание концептуальной модели данных. Создание логической модели данных. Создание физической модели данных. Описание механизмов обеспечения целостности базы данных.	
25.06.20-01.07.20	Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных. Структурирование запросов базы данных. Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных.	
02.07.20-03.07.20	Определение способов управления правами пользователей. Описание распределения прав пользователей и управления ими в базе данных.	
03.07.20-04.07.20	Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных.	
06.07.20	Составление отчёта по практике	
07.07.20	Подготовка к защите отчёта по практике.	
08.07.20	Итоговая аттестация по производственной практике - зачёт	

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью производственной практики является процесс подготовки студентов работе на различных предприятиях, в условиях реального производства. Во время прохождения производственной практики происходит закрепление и конкретизация результатов теоретического учебно-практического обучения, приобретение студентами умения и навыков практической работы по присваиваемой квалификации и избранной специальности или профессии.

Трансформация практики, максимально приближенной к будущей профессиональной деятельности, в учебный процесс — явление закономерное, обусловленное требованиями Государственных образовательных стандартов РФ.

Производственная практика проходила в филиале «Ракетно-космическая техника» государственного бюджетного образовательного учреждения высшего образования «Московский авиационный институт» в г. Химки Московской области по специальности 09.02.03 – Программирование в компьютерных системах.

1 Анализ предметной области

1.1 Анализ систем контроля версий

Для совместной работы в команде, используются системы управления контроля версиями для внесения изменений в разрабатываемый продукт. Системы контроля версиями облегчает разработку продукта тем, что все изменения, внесённые в проект, можно отследить и в случае возникновения проблем, вернуться к прошлой версии продукта. Существует множество систем контроля версий, они отличаются удобством и поколениями.

Функции системы контроля версий

- хранение несколько версий одного и того же документа (история версий);
- хранение истории разработки;
- при необходимости возвращение к более ранним версиям документа;
- определение, кто и когда сделал изменение;
- совмещение изменений, сделанных разными разработчиками;
- реализация альтернативных/экспериментальных вариантов проекта.

Система контроля версиями Git – является самой распространённой системой контроля версиями. Отличается от других систем контроля версиями функциональностью, гибкостью и скоростью её использования. [1]

Центрального репозитория в Git не существует, все копии создаются равными – это значит, что разработчики могут обмениваться изменениями друг с другом непосредственно перед объединением своих изменений в официальную ветвь.

Система контроля версий Mercurial – так же является распространённой системой контроля версиями, она использует многие из тех же технологий, что и Git, но делает это иначе. Это вторая по популярности система управления версиями, но используется она гораздо реже. [2]

1.2 Анализ JavaScript библиотек

React – это JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React используется для разработки одностраничных и мобильных приложений. Его цель – предоставить высокую скорость работы, простоту и

масштабируемость. Часто React используется с другими библиотеками, такими как Redux и MobX. [3] На рисунке 1 представлен пример использования React в HTML с JSX и JavaScript.

```
<div id="myReactApp"></div>

<script type="text/babel">
  class Greeter extends React.Component {
    render() {
      return <h1>{this.props.greeting}</h1>
    }
  }

  ReactDOM.render(<Greeter greeting="Hello World!" />, document.getElementById('myReactApp'));
</script>
```

Рисунок 1 – Пример использования React

Прежде чем приступать к разработке веб-приложения с использованием React необходимо установить Node.js для того, чтобы видеть изменения, которые вносятся в исходный код проекта.

После установки Node.js можно приступить к созданию основы React-приложения, для этого можно использовать пакет create-react-app от компании Facebook. В пакете create-react-app уже находится уже большинство инструментов, которые могут быть полезны в момент разработки.

Для того чтобы глобально установить create-react-app, необходимо ввести в командную строку команду: «npm i -g create-react-app».

Затем, для создания шаблона приложения, выполните такую команду: «create-react-app react-intro».

На этом предварительная подготовка закончена. Для запуска приложения выполните следующие команды: «cd react-intro» и «npm start».

После выполнения этих команд, откроется браузер и будет выполнен переход на страницу с таким адресом: http://localhost:3000/. Далее переходим в папку с созданным проектом.

Файл index.html – находится в папке проекта, который находится по адресу public/index.html. На рисунке 2 изображён код с файла index.html.

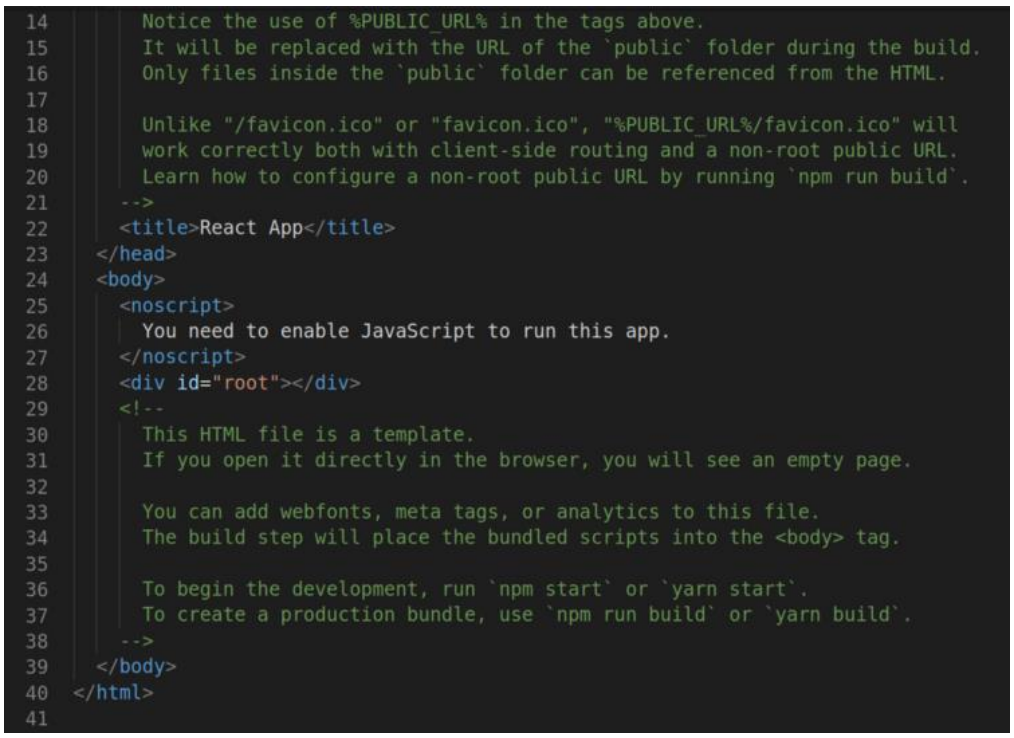

Рисунок 2 – Код с файла index.html.

В этом файле точку интереса представляет строка . Именно тут будет находиться React-приложение. Весь этот элемент будет заменён на код приложения, а всё остальное останется неизменным. [4]

Файл index.js – находится в папке src. Именно этот файл выполняет развёртывание React-приложения. И, между прочим, исходный код приложения будет размещаться в директории src. На рисунке 3 изображен код файла index.js.

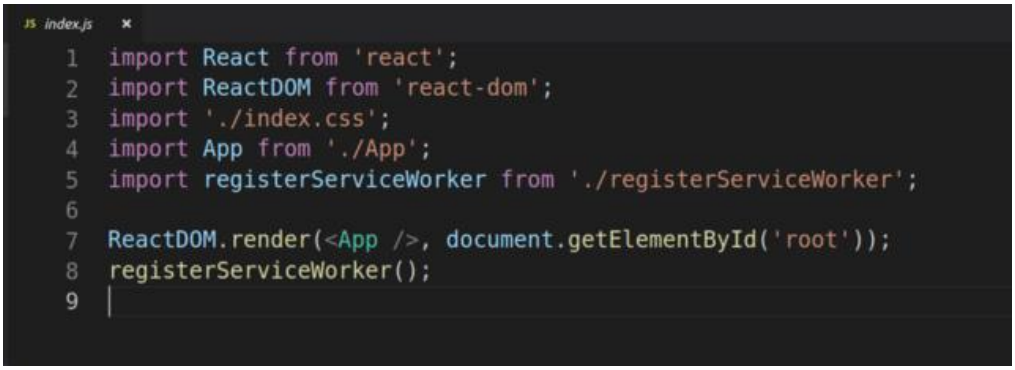

Рисунок 3 – Кол файла index.js

В этом файле точкой интереса служит строка кода «ReactDOM.render(<App />, document.getElementById('root'));», ответственная за вывод самого React приложения на страницу.

Эта строка сообщает React о том, что нужно взять компонент App и поместить его в div-элемент root, который был определён в только что рассмотренном файле index.html.

Конструкция <App /> - очень похожа на HTML-код, но это — образец JSX-кода, который представляет собой особый синтаксис JavaScript, используемый React. Обратите внимание на то, что эта конструкция начинается с заглавной буквы A, на то, что это именно <App />, а не <app />. Это так из-за используемого в React соглашения по именованию сущностей. Такой подход позволяет системе различать обычные HTML-теги и компоненты React. Если имена компонентов не будут начинаться с заглавной буквы, React не сможет вывести их на страницу.

Если в некоем .js-файле планируется использовать JSX, там необходимо импортировать React, воспользовавшись следующей командой: «import React from 'react';»

Файл App.js. На рисунке 4 изображён код файла App.js.

```
1  import React, { Component } from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4
5  class App extends Component {
6    render() {
7      return (
8        <div className="App">
9          <header className="App-header">
10           <img src={logo} className="App-logo" alt="logo" />
11           <h1 className="App-title">Welcome to React</h1>
12         </header>
13         <p className="App-intro">
14           To get started, edit <code>src/App.js</code> and save to reload.
15         </p>
16       </div>
17     );
18   }
19 }
20
21 export default App;
22
```

Рисунок 4 – Код файла App.js

Для того, чтобы создать компонент React, необходимо сначала создать класс, являющийся наследником React.Component. Именно эту задачу решает строка class App extends Component. Все компоненты React должны содержать реализацию метода render, в котором, исходя из названия этого элемента в нём выполняется рендеринг компонента, формирование описания его визуального представления. Этот метод должен вернуть HTML-разметку для вывода её на страницу.

Атрибут className — это эквивалент атрибута class в HTML. Он используется для назначения элементам CSS-классов в целях их стилизации. Ключевое слово class в JavaScript является зарезервированным, его нельзя использовать в качестве имени атрибута.

Свойства (props) — это одна из центральных концепций React. В сущности, свойства — это и есть параметры, которые передаются компонентам. На рисунке 5 изображен пример использования свойства.

```
const Greetings = (props) => <div>Hey you! {props.firstName} {props.lastNam
e}!</div>;

const App = () => (
  <div>
    <Greetings firstName="John" lastName="Smith" />
  </div>
);
```

Рисунок 5 – Пример использования свойства

Тут создан компонент Greetings и воспользовались им для того, чтобы поприветствовать человека, которого зовут John Smith, из компонента App. Весь этот код приведёт к формированию следующей HTML-разметки. На рисунке 6 изображен код HTML разметки.

```
<div>
   <div>Hey you! John Smith!</div>
</div>
```

Рисунок 6 – Пример использования свойства

Фигурные скобки в выражениях вроде {props.name} используются для выделения JavaScript-кода. Компоненту Greetings передаются, в виде параметров, свойства firstName и lastName. Работая с ними, происходит обращение к свойствам объекта props.

Стоит обратить внимание на то, что компоненту передаётся единственный объект props, а не два значения, представляющие свойства firstName и lastName.

Так же код можно упростить, воспользовавшись возможностями ES6 (ECMAScript 2015) по деструктурированию объектов. На рисунке 7 изображен упрощённый код.

```
const Greetings = ({ firstName, lastName }) => <div>Hey you! {firstName} {lastName}!</div>;
```

Рисунок 7 – Упрощённый код деструктурирования объектов

Данный код нарушает принцип единственной ответственности (Single Responsibility Principle, SRP). SRP – это один из важнейших принципов программирования, которого следует придерживаться. Он говорит о том, что модуль должен решать только одну задачу и должен делать это качественно. Если разрабатывать проект, не следуя только одному этому принципу, код такого проекта может превратиться в кошмарную конструкцию, которую невозможно поддерживать.

Нарушить принцип единственной ответственности чаще всего можно нарушить, когда несвязанные друг с другом механизмы размещают в одних и тех же файлах.

jQuery – это JavaScript-библиотека с набором функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM (от англ. Document Object Model — объектная модель документа), обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX, (от англ. Asynchronous Javascript and XML — асинхронный JavaScript и XML). [5]

Работу с jQuery можно разделить на 2 типа:

1. Получение jQuery-объекта с помощью функции $(). Например, передав в неё CSS-селектор, можно получить jQuery-объект всех элементов HTML, попадающих под критерий и далее работать с ними с помощью различных методов jQuery-объекта. В случае, если метод не должен возвращать какого-либо значения, он возвращает ссылку на jQuery объект, что позволяет вести цепочку вызовов методов согласно концепции текучего интерфейса.
2. Вызов глобальных методов у объекта $, например, удобных итераторов по массиву.

На рисунке 8 изображен пример использования функции jQuery с AJAX.

```
$.ajax({
  type: "POST",
  url: "some.php",
  data: {name: 'John', location: 'Boston'},
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

Рисунок 8 – Пример использования функции jQuery

JSON (от англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Несмотря на происхождение от JavaScript, формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON. [6]

На рисунке 9 изображен пример JSON-представления данных об объекте, описывающем человека.

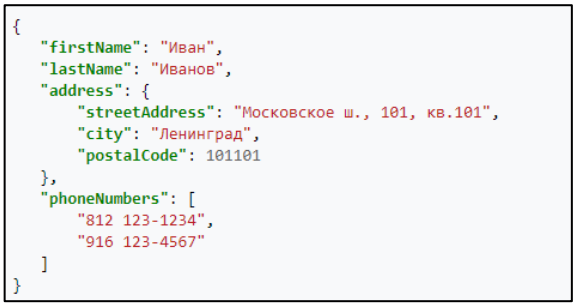

Рисунок 9 - Пример JSON-представления данных об объекте

JSON-текст представляет собой одну из двух структур:

– Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка, значением — любая форма.

– Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

– Структуры данных, используемые JSON, поддерживаются любым современным языком программирования, что и позволяет применять JSON для обмена данными между различными языками программирования и программными системами.

– В качестве значений в JSON могут быть использованы:

– запись — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.

– массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми. Массив может быть пустым, т.е. не содержать ни одного значения.

– число (целое или вещественное).

Node.js - это кроссплатформенная среда выполнения для JavaScript, среда с открытым кодом, с возможностью модификации, которая работает на серверах.

Платформа Node.js построена на базе JavaScript движка V8 от Google, который используется в браузере Google Chrome. Данная платформа, в основном, используется для создания веб-серверов, однако сфера её применения этим не ограничивается.

В основе Node.js, помимо других решений, лежит опенсорсный JavaScript-движок V8 от Google, применяемый в браузере Google Chrome и в других браузерах. Это означает, что Node.js пользуется наработками тысяч инженеров, которые сделали среду выполнения JavaScript Chrome невероятно быстрой и продолжают работать в направлении совершенствования V8.

В традиционных языках программирования (C, Java, Python, PHP) все инструкции, по умолчанию, являются блокирующими, если только разработчик явным образом не позаботится об асинхронном выполнении кода. В результате если, например, в такой среде, произвести сетевой запрос для загрузки некоего JSON-кода, выполнение потока, из которого сделан запрос, будет приостановлено до тех пор, пока не завершится получение и обработка ответа.

JavaScript значительно упрощает написание асинхронного и неблокирующего кода с использованием единственного потока, функций обратного вызова (коллбэков) и подхода к разработке, основанной на событиях. Каждый раз, когда нам нужно выполнить тяжёлую операцию, мы передаём соответствующему механизму коллбэк, который будет вызван сразу после завершения этой операции. В результате, для того чтобы программа продолжила работу, ждать результатов выполнения подобных операций не нужно.

Подобный механизм возник в браузерах. Мы не можем позволить себе ждать, скажем, окончания выполнения AJAX-запроса, не имея при этом возможности реагировать на действия пользователя, например, на щелчки по кнопкам. Для того чтобы пользователям было удобно работать с веб-страницами, всё, и загрузка данных из сети, и обработка нажатия на кнопки, должно происходить одновременно, в режиме реального времени.

1.3 Дополнительные инструменты

Microsoft Visual Studio Code — это редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы,

сочетания клавиш и файлы конфигурации. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией.

Плагины установленные в Visual Studio Code для более удобной работы с языками программирования:

– ES7 – это языковой стандарт ECMAScript 2016, в каждом выпуске стандартов вносятся изменения и добавление новых функций и операторов для более удобного и профессионального написания кода. [7]

– JSON FIX – это плагин который форматирует код и делает его более читаемым разработчику.

Jest — это фреймворк для автоматизированного тестирования JavaScript, разработанный для обеспечения уверенности в правильной работе любого JavaScript кода. Он позволяет писать тесты с приемлемым, знакомым и функциональным API, и быстро достигать желаемых результатов.

2 Разработка программного продукта

2.1 Структурная схема и макет страниц веб-приложения

С любой страницы клиентского веб-приложения возможно перейти на любую другую страницу. Что облегчает навигацию по веб-приложению. На рисунке 10 изображена схема клиентского веб-приложения. [8]

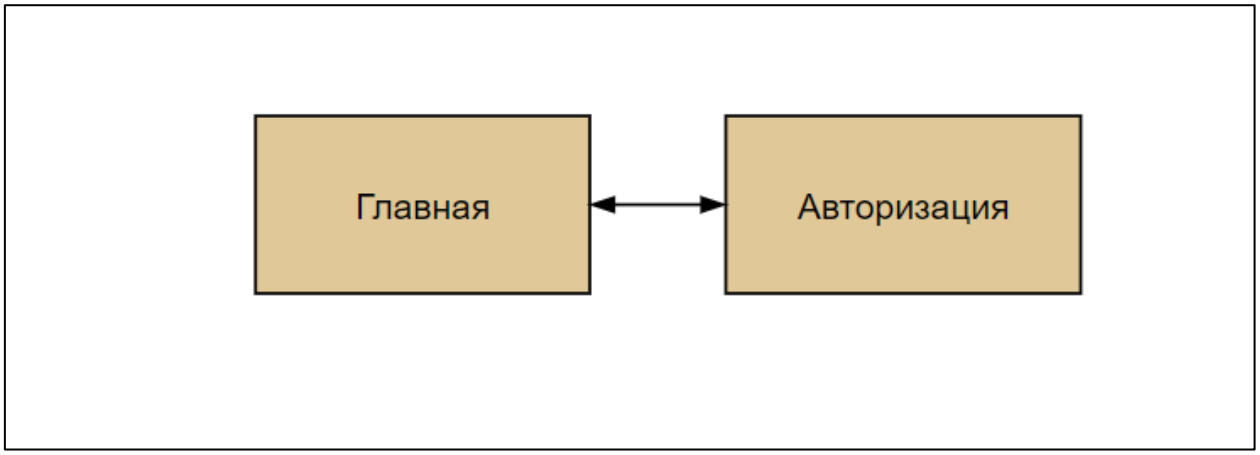

Рисунок 10 - Схема клиентского веб-приложения

На рисунке 11 изображён макет главной страницы веб-приложения. В блоке HEADER содержатся элемент перемещения по странице веб-приложения. В блоке CONTENT содержатся доступные фильмы и информация к ним.

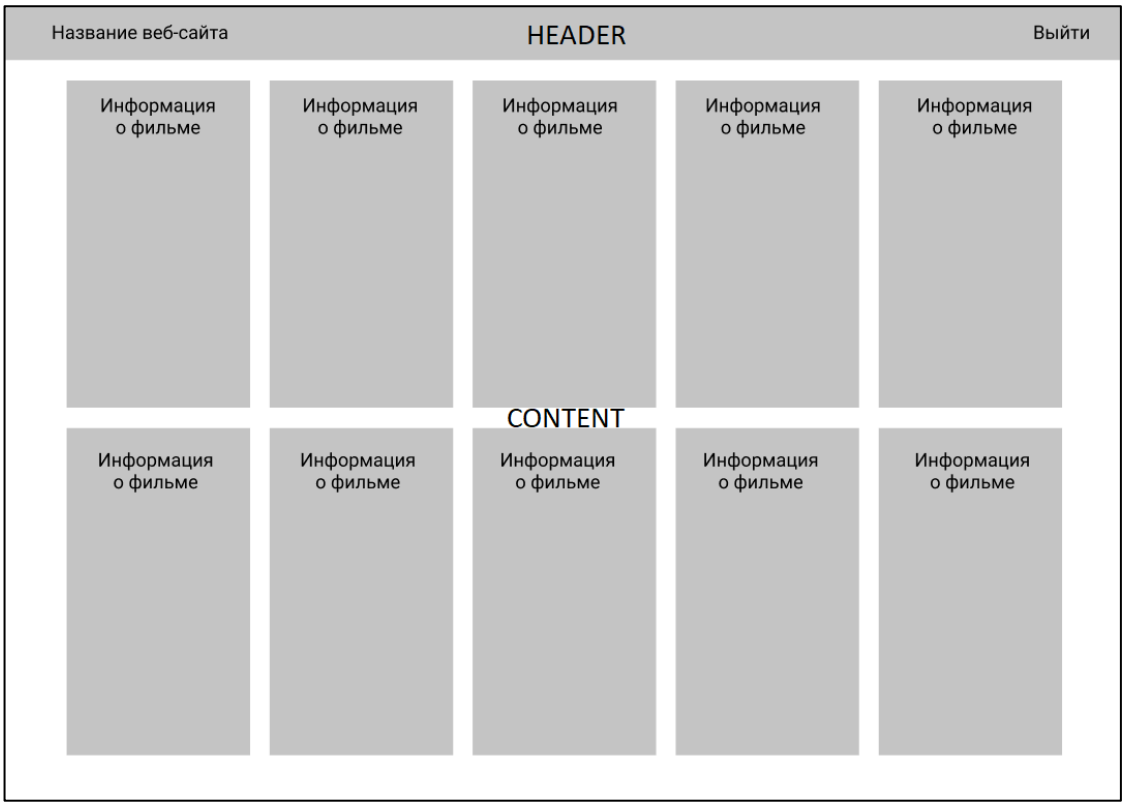

Рисунок 11 – Макет главной страницы веб-приложения

На рисунке 12 изображён макет страницы авторизации в веб-приложении.

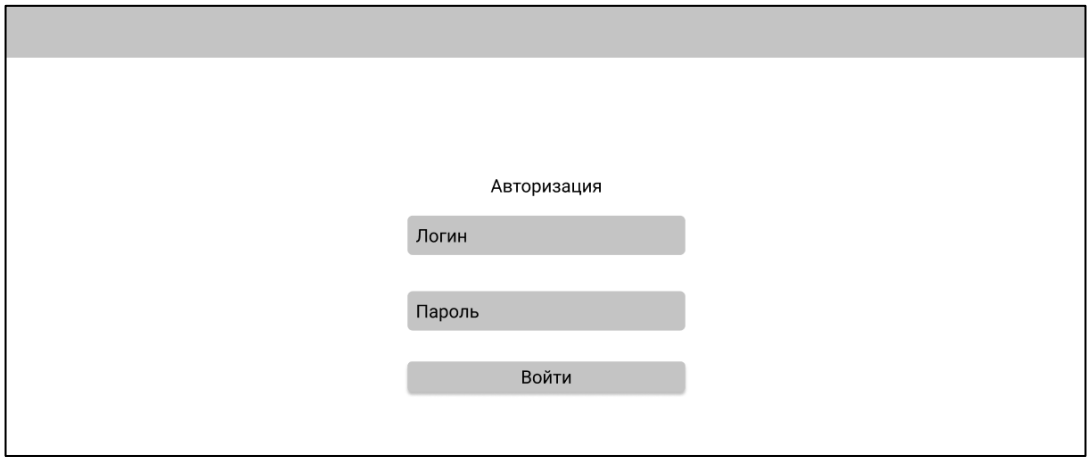

Рисунок 12 – Макет страницы авторизации веб-приложения

На рисунке 13 изображён макет главной страницы для пользователя с привилегиями администратора веб-приложения. В блоке HEADER содержится кнопка для выхода из аккаунта. В блоке CONTENT содержатся информация о фильмах и возможность их редактирования, удаления, добавления.

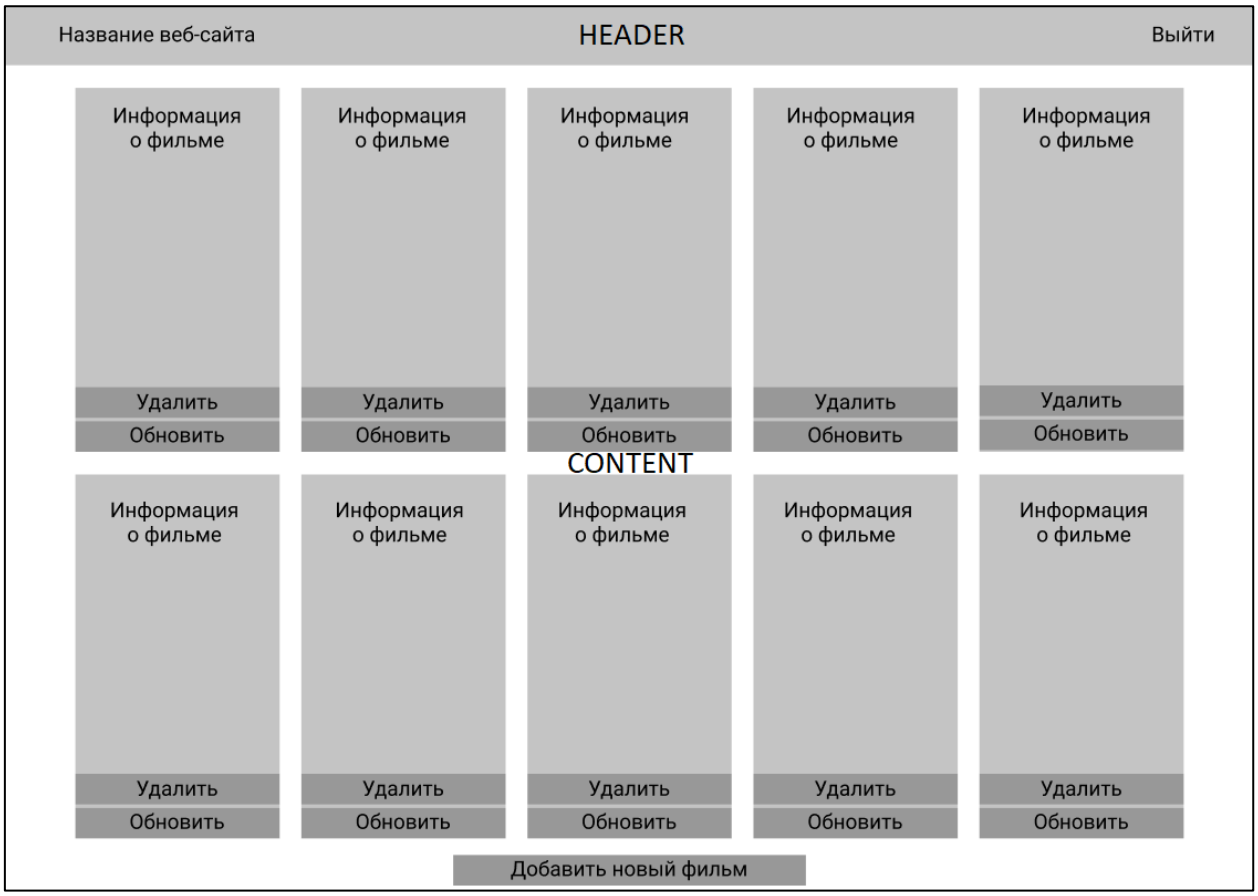

Рисунок 13 – Макет главной страницы пользователя с привилегиями администратора

2.2 Разработка серверной части программного продукта

Разработка любого программного продукта начинается с серверной части, так называемого «Бекэед». Бекэнд — это серверная часть приложения, которая не видна пользователям. Сюда относится: авторизация, хранение и обработка данных, email рассылки и тому подобное. На рисунке 14 изображены данные в формате JSON, который выполняет функцию передачи информации о фильмах.


```
[{"id_film":1,"film_name":"Matrix","country":"USA","regisseur":"Wachowski","date_release
":"2001-09-21T04:00:00.000Z"},{"id_film":2,"film_name":"The Matrix
Reloaded","country":"USA","regisseur":"Wachowski","date_release":"2003-09-
20T04:00:00.000Z"},{"id_film":3,"film_name":"The Matrix
Revolutions","country":"USA","regisseur":"Wachowski","date_release":"2003-02-
02T04:00:00.000Z"},{"id_film":7,"film_name":"A Space
Odyssey","country":"USA","regisseur":"Stanley Kubrick","date_release":"1968-06-
09T04:00:00.000Z"},{"id_film":9,"film_name":"Alien","country":"USA","regisseur":"Ridley
Scott","date_release":"1979-09-04T04:00:00.000Z"},
{"id_film":15,"film_name":"test","country":"USA","regisseur":"test","date_release":"2020
-07-21T04:00:00.000Z"}]
```


Рисунок 14 – Данные в формате JSON с информацией о фильмах

2.3 Разработка клиентской части программного продукта

Следующим по важности разработки проекта является клиентская часть. Интерфейс должен быть простым и понятным для пользователя. Эта часть проекта называется «Фронтэнд». Фронтэнд – это часть, работающая в браузере с которой непосредственно взаимодействует пользователь. Это динамические интерфейсы, меню, события по действию пользователя обмен данными с серверной частью, в общем, то, что происходит на клиенте. [7]

Для более удобной разработки веб-приложения, компоненты веб-интерфейса прописываются в отдельных файлах, после чего, на конечной странице компоненты подключаются. На рисунке 15 изображён код страницы веб-приложения, где подключены компоненты.

```
import React from 'react';
import Header from './Components/Header';
import Routing from './Components/Routing';
import {useState} from 'react'

function MainFrame(props) {
    const [accessLevel, setAccessLevel] = useState( initialState: 0);
    return (
        <div>
            <Header accessLevel={accessLevel}  setAccessLevel={setAccessLevel}/>
            <Routing accessLevel={accessLevel} setAccessLevel={setAccessLevel} FrameBody={props.FrameBody}/>
        </div>
    );
}
```

Рисунок 15 – Код страницы веб-приложения с подключёнными компонентами интерфейса

В ходе разработки веб-приложений, компоненты создаются отдельно от основной страницы, для более удобного редактирования компонентов в дальнейшем из-за чего получается модульная система. На рисунке 16 изображен один из компонентов «HEADER».

```
export default function ButtonAppBar({ accessLevel, setAccessLevel}) {
  const classes = useStyles();

  return (
    <div className={classes.root}>
      <AppBar position="static">
        <Toolbar>
          <IconButton edge="start" className={classes.menuButton} color="inherit" aria-label="menu" href="/">
          </IconButton>
          <Typography variant="h6" className={classes.title} href="/">KINOBAR</Typography>
          {!accessLevel ?
            <Button color="inherit" href="/SignIn">Войти</Button>
            : <Button color="inherit" href="/SignIn" onClick = {()=> setAccessLevel(0)}>Выйти</Button>
          }
        </Toolbar>
      </AppBar>
    </div>
  );
}
```

Рисунок 16 – Код компонента «HEADER» веб-приложения

На рисунке 17 изображен фрагмент кода компонента «UpdateForm» отвечающий за форму обновления информации о фильмах. Обновление данных в таблице происходит путём отправления данных и создания JSON запроса, состоящего из данных взятых из форм, затем данные отправляется на серверную часть, где и происходит обновление информации о фильме в таблицах базы данных.

```
function UpdateForm(props) {
  const classes = useStyles();
  const { id } = props;
  return (
    <Container component="main" maxWidth="xs">
      <CssBaseline />
      <div className={classes.paper}>
        <form className={classes.form} noValidate onSubmit={props.handleSubmit}>
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="film_name" label="Название фильма" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="country" label="Страна" type="text2" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="regisseur" label="Режиссёр" type="text3" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="date_release" type="date" />
          <Button type="submit" fullWidth variant="contained" color="primary" className={classes.submit}>Update dart</Button>
          <Button onClick={() => axios.get( url: `http://localhost:3002/ticket/${id}`).then(response => props.load(response.data[0]))}
                  fullWidth variant="contained" color="primary" className={classes.submit}>Upload data</Button>
        </form>
      </div>
      <Box mt={8}>
      </Box>
    </Container>
  );
}
let Form = reduxForm({ form: "UpdateForm" })(UpdateForm);
```

Рисунок 17 – Фрагмент кода компонента «UpdateForm»

2.4 Тестирование

На стадии тестирования разработанного продукта тестировщики стараются выявить все недостатки реализации того или иного решения. Bug report — это текст или таблица, где описывается ошибка, серьёзность ошибки, версия продукта на котором была найдена ошибка, серьезность ошибки, приоритет исправления ошибки и так далее. Ниже представлен Bug report составленный после тестирования веб-приложения и исправленный в следующих версиях:

Короткое описание: при обновлении информации о фильме в поле для ввода даты выпуска фильма можно ввести буквенные символы.

Проект: KINOBAR

Компонент приложения: веб-интерфейс, форма обновления записей;

Номер версии: 0.8

Серьезность: S3 Значительный

Приоритет: P1 Высокий

Автор: Казнаков Кирилл

Назначен на: Казнаков Кирилл

Окружение:

Браузера + версия: Google Chrome 20.6.2.195 (64-bit)

Описание:

Шаги воспроизведения: зайти на веб-сайт и попробовать ввести буквенные значения в форму, предназначенную для ввода даты выпуска фильма.

Фактический Результат: вводятся буквенные значения.

Ожидаемый результат: ввод только числовых значений.

Градация ошибок и их описание:

Серьезность (Severity) — это атрибут, характеризующий влияние дефекта на работоспособность приложения.

Приоритет (Priority) — это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.

Градация Серьезности дефекта (Severity):

S1 Блокирующая (Blocker) - Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или

ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.

S2 Критическая (Critical) - Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.

S3 Значительная (Major) - Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.

S4 Незначительная (Minor) - Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.

S5 Тривиальная (Trivial) - Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Градация Приоритета дефекта (Priority):

P1 Высокий (High) - Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.

P2 Средний (Medium) - Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.

P3 Низкий (Low) - Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

Порядок исправления ошибок по их приоритетам: High -> Medium -> Low

2.5 Автоматизированное тестирование

Автоматизированное тестирование программного обеспечения — часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс. На рисунке 18 изображен результат автоматического тестирования веб-приложения.

Рисунок 18 – Результат автоматического тестирования

ЗАКЛЮЧЕНИЕ

В ходе прохождения производственной практики в филиале «Ракетно-космическая техника» государственного бюджетного образовательного учреждения высшего образования «Московский авиационный институт» в г. Химки Московской области по специальности 09.02.03 – Программирование в компьютерных системах. Были закреплены теоретические и практические навыки в ходе изучения лекционного и практического материала.

Так же были получены навыки работы в команде, а именно: распределение ролей в проектировании и разработке программных продуктов, планирование и создание концептуальных схем и моделей программных продуктов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Git Документация. – Режим доступа: https://git-scm.com/

2. Mercurial Документация. – Режим доступа: https://www.mercurial-scm.org/

3. Бэнкс Алекс, Порселло Ева. React и Redux: функциональная веб-разработка. — СПб.: «Питер», 2018.

4. React Документация. – Режим доступа: https://reactjs.org/docs/getting-started.html

5. jQuery Документация. – Режим доступа: https://jquery.com/

6. JSON Документация. – Режим доступа: https://www.json.org/json-ru.html

7. ES7 Документация. – Режим доступа: https://www.ecma-international.org/ecma-262/7.0/

8. Марко Беллиньясо. ASP.NET 2.0 Website Programming: Problem - Design - Solution. — М.: «Диалектика», 2016.

```
import React from 'react';
import { makeStyles } from '@material-ui/core/styles';
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import Typography from '@material-ui/core/Typography';
import Button from '@material-ui/core/Button';
import IconButton from '@material-ui/core/IconButton';
const useStyles = makeStyles((theme) => ({
  root: {
    flexGrow: 1,
  },
  menuButton: {
    marginRight: theme.spacing(2),
  },
  title: {
    flexGrow: 1,
  },
}));
/**
 * Верхний тулбар, хедер
 * @param {*} param0
 */
export default function ButtonAppBar({ accessLevel, setAccessLevel}) {
  const classes = useStyles();
  return (
    <div className={classes.root}>
      <AppBar position="static">
        <Toolbar>
          <IconButton edge="start" className={classes.menuButton} color="inherit" aria-
label="menu" href="/">
```

```
        </IconButton>
        <Typography variant="h6" className={classes.title} href="/">KINOBAR</Typography>
        {!accessLevel ?
          <Button color="inherit" href="/SignIn">Войти</Button>
         : <Button color="inherit" href="/SignIn" onClick = {()=>
setAccessLevel(0)}>Выйти</Button>
        }
       </Toolbar>
     </AppBar>
   </div>
  );
}
```

```
import React from 'react';
import Button from '@material-ui/core/Button';
import CssBaseline from '@material-ui/core/CssBaseline';
import TextField from '@material-ui/core/TextField';
import Box from '@material-ui/core/Box';
import { makeStyles } from '@material-ui/core/styles';
import Container from '@material-ui/core/Container';
import { reduxForm, Field } from 'redux-form';
/**
 * Настройка стилей страницы
 */
const useStyles = makeStyles((theme) => ({
  paper: {
    marginTop: theme.spacing(8),
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
  },
  form: {
    width: '100%', // Fix IE 11 issue.
    marginTop: theme.spacing(1),
  },
  submit: {
    margin: theme.spacing(3, 0, 2),
  },
}));
/**
 * Компонет-обёртка тестового поля
 * @param {*} props
 */
const TextFieldWrapper = props => {
  const { input, ...other } = props
```

```
  return <TextField {...input} {...other} />;
};
/**
 * Форма добавления записи
 * @param {*} props
 */
function AddForm(props) {
  const classes = useStyles();
  return (
    <Container component="main" maxWidth="xs">
      <CssBaseline />
      <div className={classes.paper}>
        <form className={classes.form} noValidate onSubmit={props.handleSubmit}>
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required
fullWidth name="film_name" label="Название фильма" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required
fullWidth name="country" label="Страна" type="text2" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required
fullWidth name="regisseur" label="Режиссёр" type="text3" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required
fullWidth name="date_release" type="date" />
          <Button type="submit" fullWidth variant="contained" color="primary"
className={classes.submit}>Добавить запись</Button>
        </form>
      </div>
      <Box mt={8}>
      </Box>
    </Container>
  );
}
let Form = reduxForm({ form: "Form_is_ready" })(AddForm);
export default Form;
```

ПРИЛОЖЕНИЕ В

Routing.js

```
import React, { Component, useState } from 'react'
import { BrowserRouter as Router, Switch, Route, Link, useHistory } from "react-router-dom";
import SignIn from './SignIn';
import axios from 'axios';
/**
 * Компонент-обёртка для формы входа
 * @param {*} props
 */
function SignInForm(props) {
    const history = useHistory();
    return (
        <SignIn onSubmit={
            values => {
                axios
                    .post(`http://localhost:3002/login/`, values)
                    .then(res => { res.data.success && (history.push('/Application') ||
props.setAccessLevel(res.data.accessLevel)) });
            }
        }
        />
    );
}
/**
 * Компонент, отвечающий за навигацию на странице
 * @param {*} props
 */
export default function Routing(props) {
    const { accessLevel, setAccessLevel } = props;
    const { FrameBody } = props;
```

```
    return (
        <div>
            <Router>
                <Switch>
                    <Route path='/SignIn' render={props => <SignInForm {...props}
setAccessLevel={setAccessLevel} />} />
                    <Route path='/Application' render={props => <FrameBody {...props}
accessLevel={accessLevel} />} />
                    <Route path='/' render={props => <FrameBody {...props} />} />
                </Switch>
            </Router>
        </div>
    )
}
```

```
import React from 'react';
import Avatar from '@material-ui/core/Avatar';
import Button from '@material-ui/core/Button';
import CssBaseline from '@material-ui/core/CssBaseline';
import Link from '@material-ui/core/Link';
import Box from '@material-ui/core/Box';
import LockOutlinedIcon from '@material-ui/icons/LockOutlined';
import Typography from '@material-ui/core/Typography';
import { makeStyles } from '@material-ui/core/styles';
import Container from '@material-ui/core/Container';
import TextField from '@material-ui/core/TextField';
import { reduxForm, Field } from 'redux-form';
const TextFieldWrapper = props => {
  const { input, ...other } = props
  return <TextField {...input} {...other} />;
};
const useStyles = makeStyles((theme) => ({
  paper: {
    marginTop: theme.spacing(8),
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
  },
  avatar: {
    margin: theme.spacing(1),
    backgroundColor: theme.palette.secondary.main,
  },
  form: {
    width: '100%', // Fix IE 11 issue.
```

```
    marginTop: theme.spacing(1),
  },
  submit: {
    margin: theme.spacing(3, 0, 2),
  },
}));
/**
 * Компоенент, содержащий форму входа
 * @param {*} props
 */
function SignInForm(props) {
  const classes = useStyles();
  return (
    <Container component="main" maxWidth="xs">
      <CssBaseline />
      <div className={classes.paper}>
        <Avatar className={classes.avatar}>
          <LockOutlinedIcon />
        </Avatar>
        <Typography component="h1" variant="h5">
          Авторизация
        </Typography>
        <form className={classes.form} noValidate onSubmit={props.handleSubmit}>
          <Field component={TextFieldWrapper}
            variant="outlined"
            margin="normal"
            required
            fullWidth
            id="login"
            label="Логин"
            name="login"
            autoFocus
```

```
          />
          <Field component={TextFieldWrapper}
            variant="outlined"
            margin="normal"
            required
            fullWidth
            name="password"
            label="Пароль"
            type="password"
            id="password"
            autoComplete
          />
          <Button
            type="submit"
            fullWidth
            variant="contained"
            color="primary"
            className={classes.submit}
          >
            Войти
          </Button>
        </form>
      </div>
    </Container>
  );
}
let SignIn = reduxForm({ form: "SignIn" })(SignInForm);
export default SignIn;
```

```
import { applyMiddleware, combineReducers, createStore } from "redux";
import { logger } from "redux-logger";
import { reducer as formReducer } from 'redux-form'
const LOAD = 'redux-form-examples/account/LOAD';
const reducer = (state = {}, action) => {
  switch (action.type) {
    case LOAD:
      return {
        data: { ...action.data, date_release: (new
Date(action.data.date_release)).toISOString().substr(0, 10) }
      };
    default:
      return state;
  }
};

/**
 * Simulates data loaded into this reducer from somewhere
 */
export const load = data => ({ type: LOAD, data });
const store = createStore(
  combineReducers({
    form: formReducer,
    ticket: reducer
  },
  ),
  applyMiddleware(logger))
window.store = store;
export default store
```

```
import React from 'react';
import Button from '@material-ui/core/Button';
import CssBaseline from '@material-ui/core/CssBaseline';
import TextField from '@material-ui/core/TextField';
import Box from '@material-ui/core/Box';
import { makeStyles } from '@material-ui/core/styles';
import Container from '@material-ui/core/Container';
import { reduxForm, Field } from 'redux-form';
import { load } from './store';
import axios from 'axios';
import { connect } from 'react-redux';
const useStyles = makeStyles((theme) => ({
  paper: {
    marginTop: theme.spacing(8),
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
  },
  form: {
    width: '100%', // Fix IE 11 issue.
    marginTop: theme.spacing(1),
  },
  submit: {
    margin: theme.spacing(3, 0, 2),
  },
}));
const TextFieldWrapper = props => {
  const { input, ...other } = props
  return <TextField {...input} {...other} />;
};
/**
 * Форма обновления записи таблицы
```

```
 * @param {*} props
 */
function UpdateForm(props) {
  const classes = useStyles();
  const { id } = props;
  return (
    <Container component="main" maxWidth="xs">
      <CssBaseline />
      <div className={classes.paper}>
        <form className={classes.form} noValidate onSubmit={props.handleSubmit}>
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="film_name" label="Название фильма" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="country" label="Страна" type="text2" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="regisseur" label="Режиссёр" type="text3" />
          <Field component={TextFieldWrapper} variant="outlined" margin="normal" required fullWidth name="date_release" type="date" />
          <Button type="submit" fullWidth variant="contained" color="primary" className={classes.submit}>Update dart</Button>
          <Button onClick={() => axios.get(`http://localhost:3002/ticket/${id}`).then(response => props.load(response.data[0]))}
                  fullWidth variant="contained" color="primary" className={classes.submit}>Upload data</Button>
        </form>
      </div>
      <Box mt={8}>
      </Box>
    </Container>
  );
}
let Form = reduxForm({ form: "UpdateForm" })(UpdateForm);
// You have to connect() to any reducers that you wish to connect to yourself
Form = connect(
  state => ({
```

```
    initialValues: state.ticket.data, // pull initial values from account reducer
  }),
  { load: load }, // bind account loading action creator
)(Form);
export default Form;
```