**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

по дисциплине «**Современные средства *Front-end* разработки»**

на тему: «**Разработка приложения с использованием *React*-библиотеки**»

Выполнил: магистрант гр. МАГ 40-21

Вергейчик А.П.

Принял: преподаватель

Белявский Ю.В.

Гомель, 2019

**Цель работы:** изучить основные приёмы работы с *React-*библиотекой для создания динамических одностраничных приложений.

**Задание:** создать одностраничное приложение с использованием *React*-библиотеки и компонентного подхода.

**Ход работы:**

Для создания *React*-приложения использовался запрос *npx create-react-app my-app*. Данный запрос создает стандартную архитектуру *React*-приложения и подключит все необходимые зависимости и библиотеки (*babel*, *webpack*, *eslint* и другие). На рисунке 1 представлена структура созданного приложения.

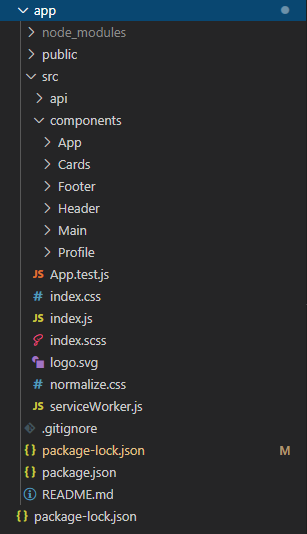


Рисунок 1 – Структура созданного приложения

В папке *api* содержатся данные для вывода на странице. В папке *components* содержатся все созданные компоненты приложения. В *index.js* собираются все компоненты.

На рисунке 1 представлена страница авторизации.

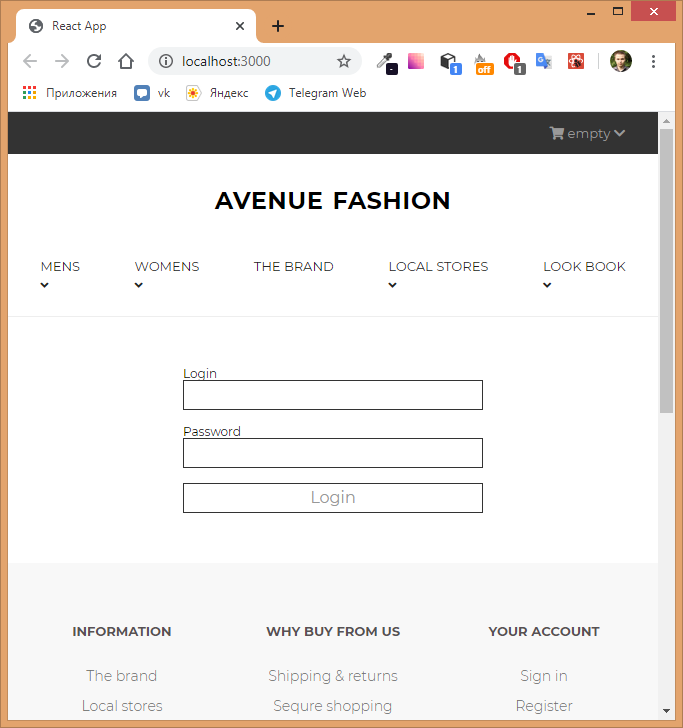


Рисунок 1 – Страница авторизации

Все формы проходят проверку на валидацию. В случае не соответствия входных данных и ожидаемых, выводится соответствующее сообщение (рисунок 2).

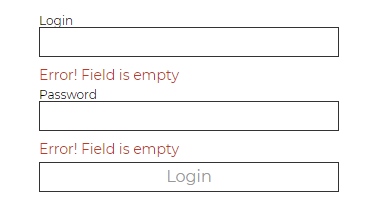


Рисунок 2 – Валидация данных

После авторизация на странице отображаются карточки с товарами (рисунок 3).

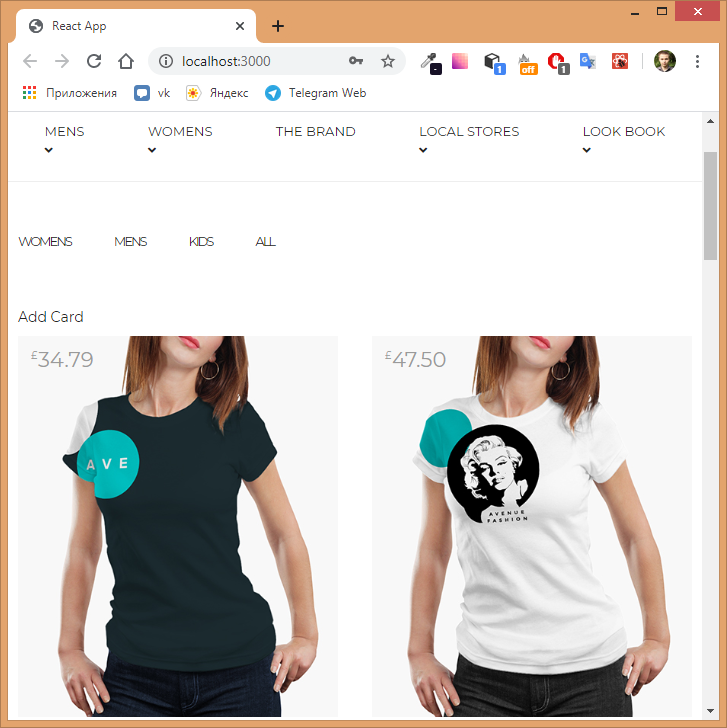


Рисунок 3 – Главная страница

Вся структура страницы представляет собой набор компонентов, которые объединены в один родительский компонент. Данные пробрасывается от родительского компонента к дочерним с помощью свойств (*props*).

При наведении на товар появляется подробная информации об товаре. Также при наведении его можно удалить из списка.

В качестве разметки на страницах используется *JSX*. Хоть *React* может работать без использования *JSX*,но данный синтаксис облегчает разработку и позволяет использовать *JavaScript* вставки в *HTML*.

Еще одной особенностью *React* является *virtual DOM*. Виртуальный *DOM* (*VDOM*) – это концепция программирования, в которой идеальное или «виртуальное» представление пользовательского интерфейса хранится в памяти и синхронизируется с «настоящим» *DOM* при помощи библиотеки, такой как *ReactDOM*. Этот процесс называется согласованием.

Такой подход и делает *API React* декларативным: вы указываете, в каком состоянии должен находиться пользовательский интерфейс, а *React* добивается, чтобы *DOM* соответствовал этому состоянию. Это абстрагирует манипуляции с атрибутами, обработку событий и ручное обновление *DOM*, которые в противном случае пришлось бы использовать при разработке приложения.

**Вывод:** в ходе лабораторной работы было создано *React*-приложение. В Структура приложения подразумевает компонентный подход и разделение на модель, представление и контроллер.

Все вводимые данные проходят валидацию. В случае ошибки выводится соответствующее сообщение.

Представления разрабатывались с использованием *JSX*-синтаксиса. Данный синтаксис позволяет использовать JavaScript и HTML-синтаксис вместе. Но у такго подхода есть один минус. Браузеры не понимают JSX-синтаксис и перед его публикацией необходимо преобразовать с помощью *babel*-компилятора.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Описание компонента App**

import React from 'react';

import { BrowserRouter as Router, Switch, Route } from 'react-router-dom';

import Header from '../Header/';

import Main from '../Main/';

import Footer from '../Footer/';

import LoginForm from './LoginForm';

import Profile from '../Profile/Profile';

export default class App extends React.Component {

constructor(props) {

super(props);

this.state = {

user: {

isAuthorize: false,

login: 'admin',

password: 'admin'

},

fields: {

login: '',

password: ''

},

errors: {}

};

this.appRouter = this.appRouter.bind(this);

this.loginFormInputChange = this.loginFormInputChange.bind(this);

this.initializeUser = this.initializeUser.bind(this);

}

componentWithProps(Component, props) {

return function(matchProps) {

return <Component {...props} {...matchProps} />

}

}

appRouter() {

const { isAuthorize } = this.state.user;

const { fields, errors } = this.state;

if(isAuthorize) {

return(

<Switch>

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

<Route path="/" exact component = { Main } />

<Route component = { Main } />

</Switch>

);

}

else {

return(

<Switch>

<Route path="/" component = {

this.componentWithProps(LoginForm, {

fields,

errors,

loginFormInputChange: this.loginFormInputChange,

initializeUser: this.initializeUser

})

} />

}

</Switch>

);

}

}

loginFormInputChange (field, event) {

let { fields } = this.state;

fields[field] = event.target.value;

this.setState({ fields });

}

isLoginFormValid() {

const { login, password } = this.state.fields;

let errors = {};

let isValid = true;

if(!login) {

isValid = false;

errors.login = 'Error! Field is empty';

}

else {

if(!login.match(/^[a-zA-Z]+$/)){

isValid = false;

errors.login = "Only letters";

}

}

if(!password) {

isValid = false;

errors.password = 'Error! Field is empty';

}

this.setState({ errors });

return isValid;

}

initializeUser(e) {

e.preventDefault();

if(this.isLoginFormValid()) {

const { login, password } = this.state.user;

const { login: \_login, password: \_password } = this.state.fields;

if(login === \_login && password === \_password) {

this.setState({ user: { isAuthorize: true } });

}

}

}

render() {

return(

<div>

<Router>

<Header user = { this.props.userData.user } />

{ this.appRouter() }

</Router>

<Footer />

</div>

);

}

}