

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
УНИВЕРСИТЕТ ИТМО

Факультет инфокоммуникационных технологий

Образовательная программа 09.03.03

Направление подготовки (специальность) Мобильные сетевые технологии

О Т Ч Е Т

о курсовой работе

Тема задания: реализация клиентской части веб-приложения для реализации курьерской доставки внутри общежитий с помощью React.JS и Django REST Framework

Обучающийся Егоров Мичил Прокопьевич, К33401

Руководитель: Добряков Д. И., преподаватель

Оценка за курсовую работу ____

Подписи членов комиссии:

____ (Добряков Д. И.)
(подпись)

Дата _____

Санкт-Петербург
2020

ВВЕДЕНИЕ	3
Актуальность	3
Цели и задачи	3
ГЛАВА 1. СРЕДСТВА РАЗРАБОТКИ И ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ	4
1.1 Средства разработки	4
1.2 Функциональные требования	4
ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ	6
2.1. Проектирование серверной части	6
2.2. Реализация общения клиентской и серверной частей	7
2.3. Реализация корзины	10
2.3. Реализация представлений	12
App.js.	12
Shops.js.	13
Cart.js.	15
ЗАКЛЮЧЕНИЕ	22
СПИСОК ЛИТЕРАТУРЫ	23

ВВЕДЕНИЕ

Актуальность

В настоящее время сервисы доставки еды приносят компаниям огромную выручку. Рынок почти что монополизирован, поэтому курьерская доставка с каждым днем растет. С другой стороны студенты хотят заработать не отходя далеко от общежития. Поэтому мы предлагаем сервис заказа курьерской доставки внутри жителей общежития - еду, товар принесут прямо до двери.

Цели и задачи

1. Определение средств разработки
2. Определение функциональных требований
3. Проектирование и реализация серверной части приложения
4. Проектирование и реализация взаимодействия клиентской и серверной части
5. Реализация представлений клиентской части

ГЛАВА 1. СРЕДСТВА РАЗРАБОТКИ И ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

1.1 Средства разработки

В качестве технологии по реализации серверной части был выбран фреймворк языка программирования Python - Django [1]. Вокруг Django быстро сформировалось активное сообщество. Фреймворк стал стремительно развиваться усилиями волонтеров. Значительную роль в успехе Django сыграли несколько известных сайтов, которые использовали этот фреймворк. В их число входят Pinterest, Dropbox, Spotify, сайт The Washington Post. В настоящее время сообщество Django включает более 11 тыс. разработчиков из 166 стран мира.

Клиентская часть приложения реализуется на фреймворке React.JS [2]. React обеспечивает повышенную гибкость благодаря использованию «компонентов» — коротких изолированных участков кода, которые помогают разработчикам создавать сложную логику и UI. React взаимодействует с HTML через virtual DOM — копию реального DOM-дерева элементов страницы. В копии все элементы представлены как объекты JavaScript. Эти элементы, вместе с декларативным стилем программирования React и односторонним связыванием данных, упрощают и ускоряют разработку.

1.2 Функциональные требования

Нужно реализовать страницы, обеспечивающие минимальную жизненную способность приложения:

1. Лэндинг
 - а. Краткая информация о проекте и о команде.
2. Страницы аутентификации и регистрации
 - а. При регистрации пользователь вводит свой телефон, выбирает общежитие, номер комнаты, в которой он живет, персональную информацию и задает пароль доступа к своему личному кабинету.
 - б. При аутентификации требуется только номер телефона и пароль.

3. Страница магазинов
 - a. Список магазинов и возможность фильтрации по ним
4. Страница товаров в магазине
 - a. Позиции, предоставляемые выбранным магазином.
 - b. Рядом с каждой позицией его цена и текущее количество выбранного товара в корзине пользователя
5. Страница корзины
 - a. Список выбранных товаров
 - b. Возможность предложить стоимость доставки
 - c. Добавление заказа в общую стену сделанных заказов
6. Страница личного кабинета
 - a. Информация о пользователе
 - b. История заказов и возможность закрыть действующий заказ
7. Страница списка курьеров
 - a. Список курьеров в текущей общезонитии
8. Страница курьера
 - a. Персональная информация о курьере
 - b. Слоты времени, когда курьер может произвести доставку.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

2.1. Проектирование серверной части

Сначала спроектируем отношение между моделями данных (Рис. 1).

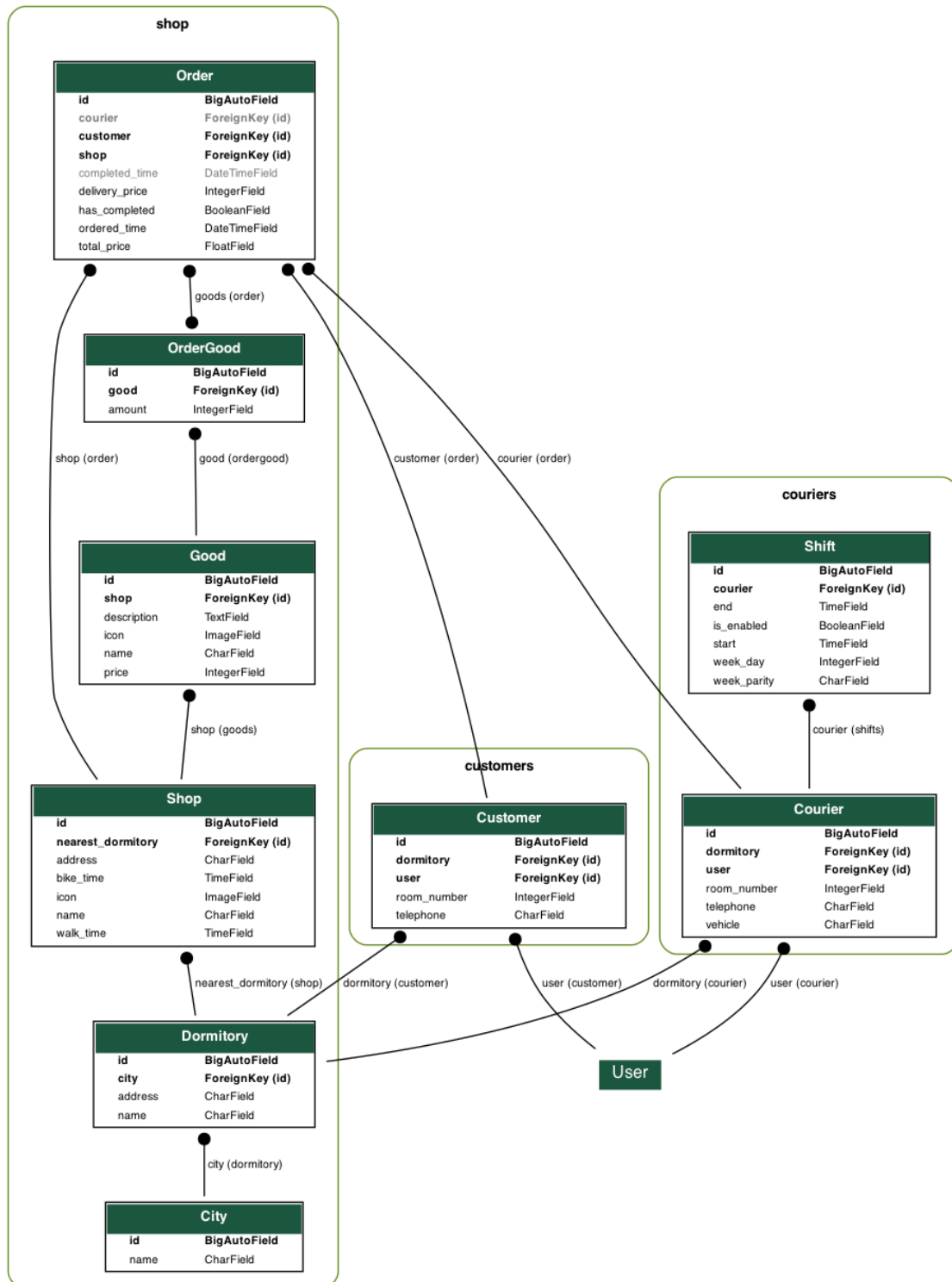


Рисунок 1 - Связь между сущностями

Чтобы обеспечить масштабируемость между городами создадим модели City и привяжем их к общежитиям Dormitory. Каждый пользователь должен быть привязан к своему общежитию и это позволит создать отдельные контексты между разными общежитиями. Пользователь сначала выбирает магазин Shop и оттуда добавлять товары Good к себе в корзину. Полностью собрав он формирует заказ Order состоящие из OrderGood. При создании заказа не сразу привязываем курьера Courier, а он потом добавляется. Курьеры могут работать только в указанные им смены Shift.

Реализуем REST API на основе спроектированных сущностей с помощью дополнения Django Rest Framework [3].

2.2. Реализация общения клиентской и серверной частей

Чтобы упростить запрос к базе данных, напомним базовый класс для формирования HTTP запросов и обращения к энд-поинтам с помощью async/await [4]. Реализуем основные методы: GET и POST, а также производную - получение списка объектов. Нужно также учитывать, что некоторые запросы требуют изменения заголовков HTTP запроса и добавления некоторой фильтрации. Код приведен в листинге 1.

Листинг 1. Базовый класс для обращения в API.

```
const BASE_URL = 'http://127.0.0.1:8000/';

class BaseAPI {
  path = '';

  async getHeader() {
    let headers = await BaseAPI.prototype.getHeader();
    if(sessionStorage.hasOwnProperty('auth_token')) {
      headers['Authorization'] = `Token
${sessionStorage['auth_token']}`
    }
    return headers;
  }

  async _request({suffix = '', filters='', method='GET',
body=null}) {
    let api_url = BASE_URL + this.path;

    if(suffix) {
      api_url += `${suffix}/`;
    }
  }
}
```

```

    }

    if(filters) {
        api_url += `?${filters}`;
    }

    let request_params = {
        headers: await this.getHeader(),
        method: method,
    }

    if(body) {
        request_params['body'] = JSON.stringify(body);
    }

    const response = await fetch(api_url, request_params);
    const data = await response.json();

    return data;
}

async list(filters='') {
    const data = await this._request({filters: filters});
    const results = data['results'];
    return results;
}

async get({suffix}) {
    const data = await this._request({suffix: suffix})
    return data;
}

async post({body, suffix=''}) {
    const data = await this._request({suffix: suffix, body:
body, method: 'POST'});
    return data;
}
}

export default BaseAPI;

```

Теперь, чтобы реализовать обращение к объектам магазина нужно отнаследоваться от BaseAPI и переопределить атрибут path. Аналогично для курьеров и товаров.

Листинг 2. Класс для формирования запросов за объектами магазинов

```
import BaseAPI from './APIUtils';

class ShopsAPI extends BaseAPI{
  path = 'api/shop/'
}

export default ShopsAPI;
```

Для формирования заказа нужно собрать тело запроса, где нужно указать с какого магазина делается заказ, указать id заказчика, стоимость доставки и перечислить какие именно товары и в каком количестве. Далее, используя собранное тело, делаем POST запрос. Для получения собственных заказов нужно добавить фильтр по заказчику.

Листинг 3. Формирование заказа

```
import BaseAPI from './APIUtils';

class OrdersAPI extends BaseAPI{
  path = 'api/order/'

  async makeOrder(cart, delivery_price) {
    const body = {
      shop: cart.shop_id,
      customer: cart.customer_id,
      delivery_price: delivery_price,
      goods: cart.goods.map(good => {
        return {
          amount: good.count,
          good: good.id,
        }
      })
    };
    const response = await this.post({body: body});
    return response;
  }

  async getUserOrders(user_id) {
    return this.list(
      `customer=${user_id}`
    )
  }
}

export default OrdersAPI;
```

Для авторизации нужно залогиниться и получать информацию о себе используя JWT-токен.

Листинг 4. Реализация аутентификации.

```
import BaseAPI from './APIUtils';

export default class AuthAPI extends BaseAPI {
  path = 'auth/';

  async login({login, password}) {
    const body = {
      username: login,
      password: password,
    }
    const data = await this.post({body: body, suffix:
'token/login'});
    return data;
  }

  async me(auth_token) {
    const data = await this.get({suffix: 'users/me'});
    return data;
  }
}
```

2.3. Реализация корзины

Корзину Cart будем хранить в сериализованном виде sessionStorage и каждый раз собирать при добавлении новых товаров или уменьшении количества. Также для простоты добавим модель CartGood, в котором будем хранить информацию о товарах в корзине.

Листинг 5. Реализация корзины.

```
class Cart {
  constructor() {
    this.goods = [];
    this.totalPrice = 0;
    this.toJson = this.toJson.bind(this);
  }

  toJson() {
    return {
      goods: this.goods.map(good => {
        return good.toJson()
      }),
      totalPrice: this.totalPrice,
    }
  }
}
```

```

    }

    static fromJson(json) {
        let cart = new Cart();
        cart.goods = json.goods.map(jsonGood => {
            return CartGood.fromJson(jsonGood)
        });
        let totalPrice = 0;
        for(let i = 0; i < cart.goods.length; i++) {
            totalPrice += cart.goods[i].price *
cart.goods[i].count;
        }
        cart.totalPrice = totalPrice;
        return cart;
    }

    appendGood(good) {
        this.totalPrice += good.price;
        for(var i = 0; i < this.goods.length; i++) {
            console.log(this.goods[i].id, good.id);
            if(this.goods[i].id === good.id) {
                this.goods[i].count += 1;
                return;
            }
        }
        good['count'] = 1;
        this.goods.push(CartGood.fromJson(good));
    }

    decreaseGood(good) {
        for(var i = 0; i < this.goods.length; i++) {
            if(this.goods[i].id === good.id) {
                this.goods[i].count -= 1;
                if(this.goods[i].count === 0) {
                    this.goods.splice(i, 1);
                }
                this.totalPrice -= good.price;
                break;
            }
        }
    }
}

class CartGood {
    constructor(id, count, price, name, icon) {
        this.id = id;
        this.count = count;
        this.price = price;
        this.name = name;
        this.icon = icon;
    }
}

```

```

    }

    static fromJson(jsonGood) {
        return new CartGood(
            jsonGood.id,
            jsonGood.count,
            jsonGood.price,
            jsonGood.name,
            jsonGood.icon,
        )
    }

    toJson() {
        return {
            id: this.id,
            count: this.count,
            price: this.price,
            name: this.name,
            icon: this.icon,
        }
    }
}

export default Cart;

```

2.3. Реализация представлений

App.js.

Подключим библиотеку `react-router-dom` [5] для реализации маршрутизации. Маршруты представлены в листинге 6.

Листинг 6. Маршруты приложения

```

<Routes>
  <Route index path="/" element={<Landing />} />
  <Route path="/goods/:shopId" element={<Goods
addGood={this.addGood} decreaseGood={this.decreaseGood}
cart={this.state.cart}/>}/>
  <Route path="/shops" element={<Shops />}/>
  <Route path="/auth" element={<Auth />}/>
  <Route path="/account" element={<Account />}/>
  <Route path="/couriers/:courierID" element={<Courier />}/>
  <Route path="/couriers/" element={<Couriers />}/>
  <Route path="/cart/" element={<Cart addGood={this.addGood}
decreaseGood={this.decreaseGood} cart={this.state.cart}/>}/>
  <Route path="/registration/" element={<Registration />}/>
  <Route path="*" element={<BadPage />} />
</Routes>

```

Будем хранить корзину в App.js. Если при инициализации App.js нет корзины в sessionStorage, инициализируем пустую корзину и добавим его в сериализованном виде. Достаем корзину и сохраним его в стейте. Далее для представления добавим методы addGood и decreaseGood, который будет принимать товар и изменять корзину в стейте и изменяя его в sessionStorage. Методы представлены в листинге 7.

Листинг 7. Конструктор представления App.js и методы взаимодействия с корзиной

```
constructor() {
  super();

  if(!sessionStorage.hasOwnProperty('cart')) {
    sessionStorage.setItem('cart', JSON.stringify(new
    CartModel().toJson()));
  }

  let cart = JSON.parse(sessionStorage.getItem('cart'));
  this.state = {
    cart: cart,
  }

  this.addGood.bind(this);
  this.decreaseGood.bind(this);
}

addGood = (good) => {
  this.state.cart.appendGood(good);
  sessionStorage.setItem('cart', JSON.stringify(this.state.cart.toJson()));
}

decreaseGood = (good) => {
  this.state.cart.decreaseGood(good);
  sessionStorage.setItem('cart', JSON.stringify(this.state.cart.toJson()));
}

render() {
  ...
}
```

Shops.js.

Подключим реализованный ранее класс ShopsAPI.js и объявим объект shop_api. После монтирования магазина сделаем GET запрос за списком магазинов. Также в стейте будем хранить переменную filter_name - какие строки искать в именах магазинов.

Листинг 8. Представление магазинов.

```

import React, { Component } from 'react'
import ShopsAPI from '../api/shopsAPI'
import CompactShop from '../components/CompactShop'
import HeaderTitle from '../components/HeaderTitle'

export default class Shops extends Component {
  constructor(props) {
    super(props);

    this.state = {shops: [], filter_name: ''}
    this.api = new ShopsAPI();
  }

  componentDidMount() {
    this.setShops = this.setShops.bind(this);
    this.filterShops = this.filterShops.bind(this);

    this.setShops();
  }

  async setShops(page = 1) {
    const shops = await
this.api.list(`name=${this.state.filter_name}`);

    this.setState({
      shops: shops
    })
  }

  filterShops(e) {
    e.preventDefault();
    this.setShops();
  }

  render() {
    return (
      <div>
        <HeaderTitle title="Рестораны"/>

        <form action="" onSubmit={this.filterShops}>
          <div class="row">
            <div class="col-md-9">
              <input class="form-control"
type="text" onChange={(e) => this.setState({filter_name:
e.target.value})} placeholder="Название магазина"/>
            </div>
            <div class="col-md-3">
              <input class="btn btn-delmitary"
type="submit" value="Поиск" style={{height: "38px"}} />
            </div>
          </div>
        </form>
      </div>
    )
  }
}

```

```

        </div>
      </div>
    </form>

    <div className="mt-5 row row-cols-1 row-cols-md-3
g-4">
      {this.state.shops.length !== 0 ?
this.state.shops.map((shop) =>
        <CompactShop key={shop.id} shop={shop}/>
      ): <p>Идет загрузка...</p>}
    </div>
  </div>
)
}
}

```

Во время запроса будем показывать сообщение “Идет загрузка...”, чтобы пользователь не подумал, что сайт завис.

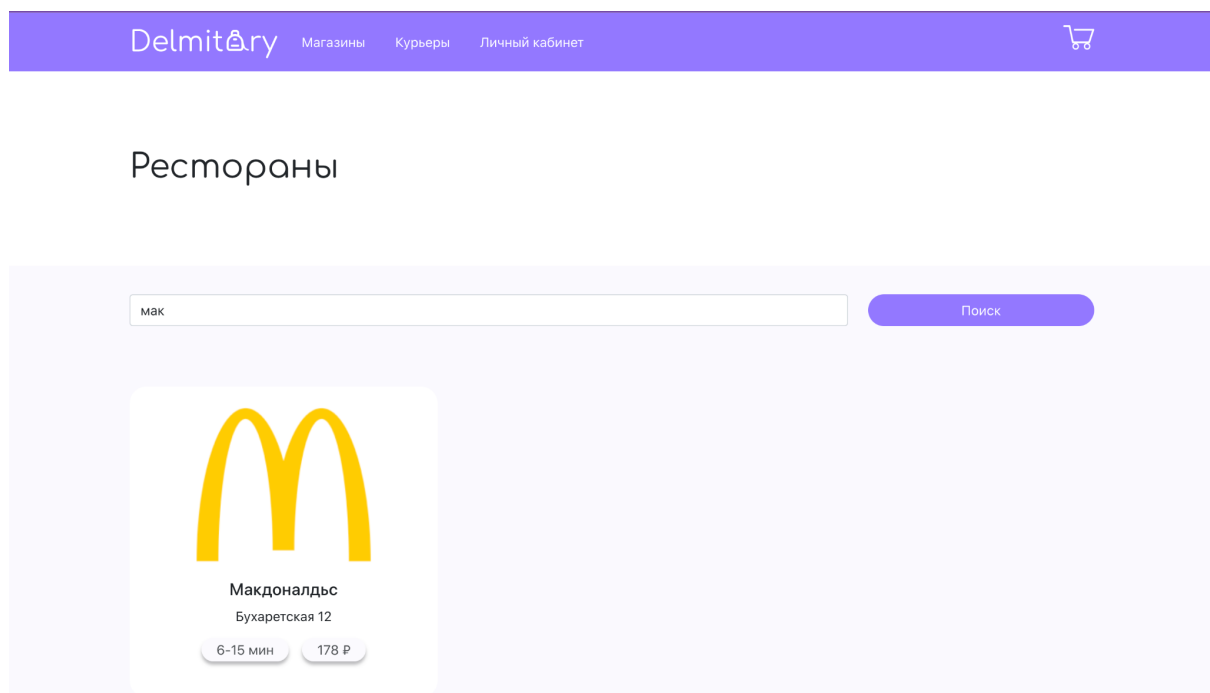


Рисунок 2 - Страница списка магазинов

Cart.js.

Будем принимать в качестве параметров методы изменения корзины из App.js и установим базовую цену заказа в пятьдесят рублей. При оформлении заказа покажем

alert, что все прошло успешно, очистим корзину и перенаправим на страницу личного кабинета.

Листинг 9. Представление корзины

```
import React, { Component } from 'react'
import { Link } from 'react-router-dom'
import HeaderTitle from '../components/HeaderTitle'
import CartItem from '../components/CartItem'
import OrderAPI from '../api/ordersAPI'
import './Cart.css'

export default class Cart extends Component {
  constructor() {
    super();

    this.api = new OrderAPI();
    this.changePrice = this.changePrice.bind(this);
    this.state = {
      delivery_price: 50,
    }
    this.makeOrder = this.makeOrder.bind(this)
    this.addGood.bind(this);
    this.decreaseGood.bind(this);
  }

  addGood = (good) => {
    this.props.addGood(good);
  }

  decreaseGood = (good) => {
    this.props.decreaseGood(good);
  }

  async makeOrder() {
    const response = await
    this.api.makeOrder(this.props.cart, this.state.delivery_price);
    if(!response.hasOwnProperty('error')) {
      alert("Спасибо что выбрали нас! Ищем курьера...")
      sessionStorage.removeItem('cart');
      document.location.replace('/account');
    }else {
      alert('Что-то пошло не так, попробуйте позднее.')
    }
  }

  changePrice(value) {
```



```

        if(!value) value = 0;
        this.setState({
            delivery_price: parseInt(value),
        })
    }

    render() {
        return (
            <div>
                <div>
                    <HeaderTitle title={"Моя корзина"} />
                </div>

                <div class="row">
                    <div className="col-md-8">
                        <h2>Заказы</h2>

                        {this.props.cart.goods.length != 0 ?
                            this.props.cart.goods.map((good) =>
                                <CartItem addGood={this.addGood} decreaseGood={this.decreaseGood}
                                good={good}/>)
                            : <Link to="/shops">Выберите товар из
                                списка</Link>}

                    </div>
                    <div className="col-md-4">
                        <h2>Итого</h2>
                        <div class="order-info card mt-4">
                            <div class="card-body"
                                style={{padding: "0px 36px 18px 36px"}}>
                                <div class="d-flex
                                    justify-content-between">
                                    <p> Всего:</p>
                                    <p>
                                        {this.props.cart.totalPrice} ₸</p>
                                </div>
                                <label> Доставка: </label>
                                <input class="form-control"
                                    type="number" value={this.state.delivery_price ?
                                        this.state.delivery_price : ''} onChange={(e) =>
                                            this.changePrice(e.target.value)} />
                                <div class="mt-3 d-flex
                                    justify-content-between">
                                    <p> Итого:</p>
                                    <p>
                                        {this.props.cart.totalPrice + this.state.delivery_price} ₸</p>
                                </div>
                                <div class="text-center">
                                    <button class="btn
                                        btn-delmitary" onClick={this.makeOrder}>Оформить заказ</button>
                                </div>

```


```

        </div>
      </div>
    </div>
  </div>
</div>
)
}
}

```

Моя корзина

Заказы




Биг мак

298 Р

-

2

+



Липтон Айс Ти - Зеленый Чай

195 Р

-

3

+

Итого

Всего:

560 Р

Доставка:

Итого:

610 Р

Оформить заказ

Рисунок 3 - Страница оформления заказа

Auth.js.

Для авторизации нужно заполнить форму. Будем сохранять значения введенных данных имени пользователя и пароля. При нажатии на кнопку, попробуем сделать запрос авторизации. Если мы получили `auth_token`, то добавим его в `sessionStorage` и перенаправим на страницу магазинов, если нет, уведомим об этом пользователя.

Листинг 10. Представление страницы аутентификации

```

import React, { Component } from 'react'
import AuthAPI from '../api/authAPI'
import { Navigate } from 'react-router'
import './Auth.css';
import { Link } from 'react-router-dom'

export default class Auth extends Component {
  constructor() {

```

```

    super();

    this.state = {
      login: '',
      password: '',
      isSuccess: false,
    }

    this.api = new AuthAPI();
  }

  componentDidMount() {
    this.login = this.login.bind(this);
    this.setLogin = this.setLogin.bind(this);
    this.setPassword = this.setPassword.bind(this);
  }

  async login(e) {
    e.preventDefault();

    const data = await this.api.login({
      login: this.state.login,
      password: this.state.password,
    });

    if(data.hasOwnProperty('auth_token')) {
      sessionStorage.setItem('auth_token',
data['auth_token']);
      this.setState({
        isSuccess: true
      });
    } else {
      alert('Неправильный логин или пароль');
    }
  }

  setLogin(login) {
    this.setState({
      'login': login,
    })
  }

  setPassword(password) {
    this.setState({
      'password': password,
    })
  }

  render() {
    if(this.state.isSuccess) {

```

```

        return <Navigate replace to='/shops/'/>
    }

    return (
        <div className="authform">
            <h3 class="comfortaaTitle"> Войти </h3>

            <form onSubmit={this.login}>
                <div className="mb-3">
                    <label htmlFor="exampleInputEmail1"
className="form-label">Логин</label>
                    <input onChange={ (e) =>
this.setLogin(e.target.value)} type="text"
className="form-control" id="exampleInputEmail1"
aria-describedby="emailHelp"/>
                </div>
                <div className="mb-3">
                    <label htmlFor="exampleInputPassword1"
className="form-label">Пароль</label>
                    <input onChange={ (e) =>
this.setPassword(e.target.value)} type="password"
className="form-control" id="exampleInputPassword1"/>
                </div>
                <div className="mb-3 form-check">
                    <input type="checkbox"
className="form-check-input" id="exampleCheck1"/>
                    <label className="form-check-label"
htmlFor="exampleCheck1">Запомнить меня</label>
                </div>
                <div className="our-button" style={{padding:
"0 61px"}}>
                    <button type="submit" className="btn
btn-delmitary">Submit</button>
                </div>
                <Link
to={` /registration/`} ><p>Регистрация</p></Link>
            </form>
        </div>
    )
}
}

```

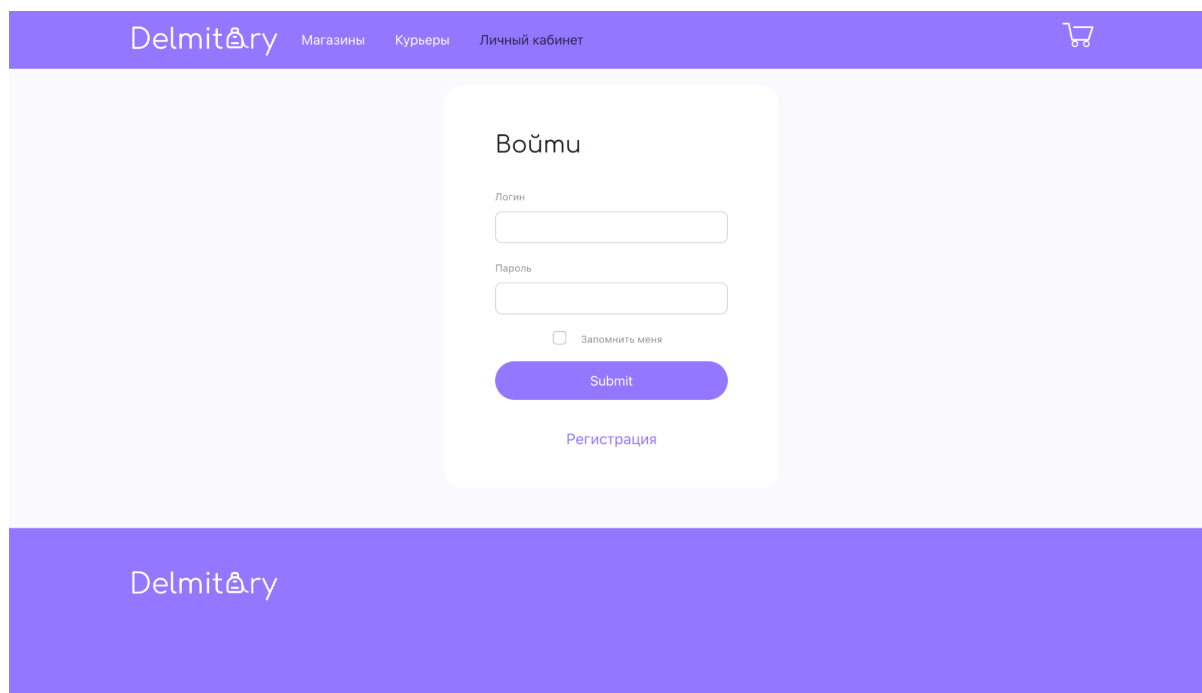


Рисунок 4 - Отрендеренная страница аутентификации

ЗАКЛЮЧЕНИЕ

В процессе данной работы мы познакомились с реализацией клиентской части приложения с помощью фреймворка React.JS. Спроектировали и реализовали формирование и отправку HTTP запросов с помощью axios/fetch и дальнейший рендер полученной информации в виде HTML. Познакомились с пакетным менеджером npm и способами добавления новых зависимостей в проект на примере react-dom-router. Добавили возможность маршрутизации между представлениями и написали некоторые компоненты.

СПИСОК ЛИТЕРАТУРЫ

1. Документация Django [Электронный ресурс] —
<https://docs.djangoproject.com/en/4.0/>. Дата обращения 18.11.2021.
2. Документация React [Электронный ресурс] —
<https://ru.reactjs.org/docs/getting-started.html>. Дата обращения 18.11.2021.
3. Документация Django REST Framework [Электронный ресурс] —
<https://www.django-rest-framework.org>. Дата обращения 18.11.2021.
4. Документация async/await в JavaScript [Электронный ресурс] —
<https://javascript.info/async-await>. Дата обращения 18.11.2021.
5. Документация библиотеки react-router-dom [Электронный ресурс] —
<https://v5.reactrouter.com/web/guides/quick-start>. Дата обращения 18.11.2021.