

Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Ивановский государственный энергетический
университет имени В.И. Ленина»

Кафедра программного обеспечения компьютерных систем

Отчёт по лабораторной работе №5

Конструирование интернет-приложений

Регистрация и аутентификация

Выполнила студент гр. 3-42 Шарабанов Н.А.

Проверил _____ Садыков А.М.

Иваново 2022

Цель лабораторной работы: добавить возможности регистрации и аутентификации пользователей

Задания:

1. Добавить Identity
 - 1.1. Добавить класс пользователя Identity
 - 1.2. Изменить наследование контекста баз данных
 - 1.2. Внедрить Identity
 - 1.3. Внести изменения в БД
2. Добавить в серверную часть регистрацию и аутентификацию
 - 2.1. Добавить классы представления данных для регистрации и аутентификации
 - 2.2. Добавить контроллер Account
 - 2.3. Проверить функцию регистрации
 - 2.4. Проверить функцию входа
 - 2.5. Проверить функцию выхода
 - 2.6. Проверить функцию проверки текущей сессии
3. Добавить в клиентскую часть с маршрутизацией по страницам, регистрацию и аутентификацию пользователей
 - 3.1. Добавить помощников по коду
 - 3.2. Добавить проксирование запросов к серверу
 - 3.3. Добавить маршрутизацию по страницам и отображение информации о пользователе
 - 3.4. Создать компонент входа
 - 3.5. Создать компонент завершения сессии
 - 3.6. Ограничить функции для Гость
 - 3.7. Создать компонент регистрации

1. Добавить Identity

Первым делом, я установил пакет: `install-package Microsoft.AspNetCore.Identity.EntityFrameworkCore -Version 6.0.15`

1.1. Добавить класс пользователя Identity

В папке Models я создал класс user.cs. Рисунок 1.

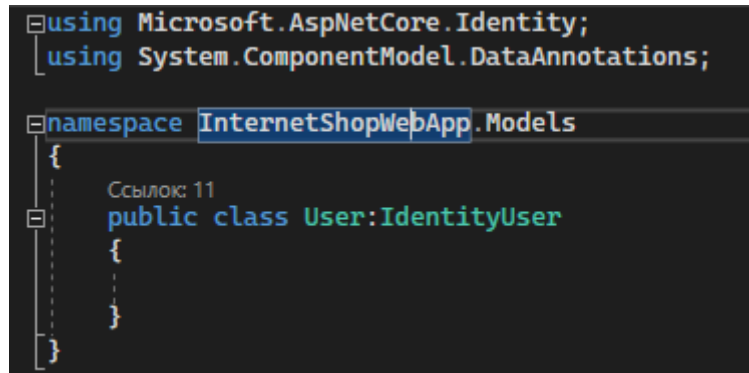


Рисунок 1 – Создание класса

Класс User был унаследован от IdentityUser, который представляет собой пользователя.

1.2. Изменить наследование контекста баз данных

После этого я изменил наследование контекста данных с DbContext на IdentityDbContext. Внёс следующие изменения в ShopContext:

```
using InternetShopWebApp.Models;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace InternetShopWebApp.Context
{
    public class ShopContext : IdentityDbContext<User>
    {
        protected readonly IConfiguration Configuration;

        ...

        protected override void OnModelCreating(ModelBuilder
            modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

1.2. Внедрить Identity

Далее я добавил в Program.cs изменения для работы с Identity:

```
using InternetShopWebApp.Context;
using InternetShopWebApp.Data;
using InternetShopWebApp.Models;
using Microsoft.AspNetCore.Identity;
using System.Text.Json.Serialization;

...

builder.Services.AddControllers();
```

```
// Learn more about configuring Swagger/OpenAPI at
https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
builder.Services.AddIdentity<User, IdentityRole>()
    .AddEntityFrameworkStores<ShopContext>();
builder.Services.AddDbContext<ShopContext>();
builder.Services.AddControllers().AddJsonOptions(x =>
    x.JsonSerializerOptions.ReferenceHandler =
    ReferenceHandler.IgnoreCycles);

...

app.UseHttpsRedirection();

app.UseCors();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

1.3. Внести изменения в БД

После добавления Identity в проект требуется обновить или пересоздать базу данных с новой структурой данных. В консоли диспетчера пакетов я выполнил обе команды:

Add-Migration Identity

Update-Database

После выполнения команд БД обновилась (Рисунок 2)

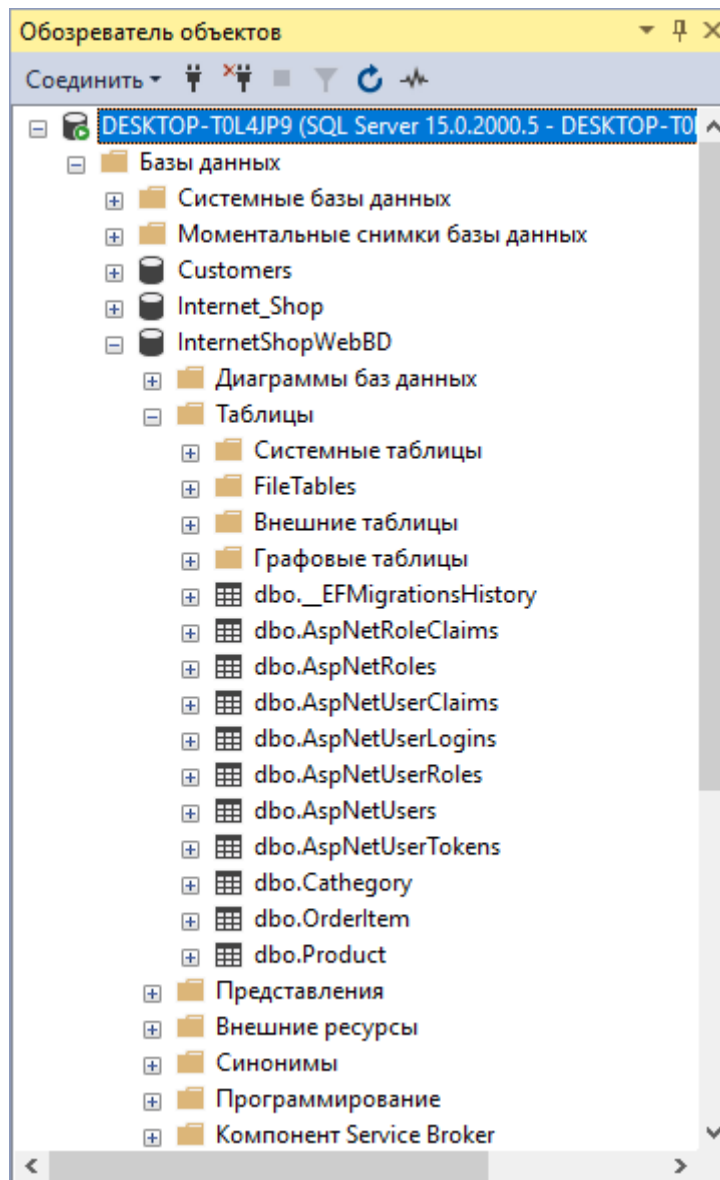


Рисунок 2 – Результаты обновления бд

2. Добавить в серверную часть регистрацию и аутентификацию

2.1. Добавить класс представления данных для регистрации

После изменения бд я добавил новый класс в папке Models, который будет представлять данные для регистрации пользователя RegisterViewModel:

```
using System.ComponentModel.DataAnnotations;

namespace WebAPI.Models
{
    public class RegisterViewModel
    {
        [Required]
        [Display(Name = "Email")]
        public string Email { get; set; }
        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Пароль")]
        public string Password { get; set; }
        [Required]
        [Compare("Password", ErrorMessage = "Пароли не совпадают")]
    }
}
```

```

        [DataType(DataType.Password)]
        [Display(Name = "Подтвердить пароль")]
        public string PasswordConfirm { get; set; }
    }
}

```

2.2. Добавить контроллер Account

Я добавил новый файл AccountController.cs в папку Controllers со следующим содержанием:

```

using InternetShopWebApp.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using WebAPI.Models;
namespace InternetShopWebApp.Controllers
{
    [Produces("application/json")]
    public class AccountController : Controller
    {
        private readonly UserManager<User> _userManager;
        private readonly SignInManager<User> _signInManager;
        public AccountController(UserManager<User> userManager, SignInManager<User>
signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }
        [HttpPost]
        [Route("api/account/register")]

        [AllowAnonymous]
        public async Task<IActionResult> Register([FromBody] RegisterViewModel
model)
        {
            if (ModelState.IsValid)
            {
                User user = new() { Email = model.Email, UserName = model.Email };
                // Добавление нового пользователя
                var result = await _userManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    // Установка куки
                    await _signInManager.SignInAsync(user, false);
                    return Ok(new { message = "Добавлен новый пользователь: " +
user.UserName });
                }
                else
                {
                    foreach (var error in result.Errors)
                    {
                        ModelState.AddModelError(string.Empty, error.Description);
                    }
                    var errorMsg = new
                    {
                        message = "Пользователь не добавлен",
                        error = ModelState.Values.SelectMany(e => e.Errors.Select(er
=> er.ErrorMessage))
                    };
                    return Created("", errorMsg);
                }
            }
            else

```

```

        {
            var errorMsg = new
            {
                message = "Неверные входные данные",
                error = ModelState.Values.SelectMany(e => e.Errors.Select(er =>
er.ErrorMessage))
            };
            return Created("", errorMsg);
        }
    }
    [HttpPost]
    [Route("api/account/login")]
    //[AllowAnonymous]
    public async Task<IActionResult> Login([FromBody] RegisterViewModel model)
    {
        if (ModelState.IsValid)
        {
            var result =
            await _signInManager.PasswordSignInAsync(model.Email,
model.Password, true, true);
            if (result.Succeeded)
            {
                return Ok(new { message = "Выполнен вход", userName =
model.Email });
            }
            else
            {
                ModelState.AddModelError("", "Неправильный логин и (или)
пароль");
                var errorMsg = new
                {
                    message = "Вход не выполнен",
                    error = ModelState.Values.SelectMany(e => e.Errors.Select(er
=> er.ErrorMessage))
                };
                return Created("", errorMsg);
            }
        }
        else
        {
            var errorMsg = new
            {
                message = "Вход не выполнен",
                error = ModelState.Values.SelectMany(e => e.Errors.Select(er =>
er.ErrorMessage))
            };
            return Created("", errorMsg);
        }
    }
    [HttpPost]
    [Route("api/account/logoff")]
    public async Task<IActionResult> LogOff()
    {
        User usr = await GetCurrentUserAsync();
        if (usr == null)
        {
            return Unauthorized(new { message = "Сначала выполните вход" });
        }
        // Удаление куки
        await _signInManager.SignOutAsync();
        return Ok(new { message = "Выполнен выход", userName = usr.UserName });
    }
    [HttpGet]
    [Route("api/account/isauthenticated")]

```

```

public async Task<IActionResult> IsAuthenticated()
{
    User usr = await GetCurrentUserAsync();
    if (usr == null)
    {
        return Unauthorized(new { message = "Вы Гость. Пожалуйста, выполните
вход" });
    }
    return Ok(new { message = "Сессия активна", userName = usr.UserName });
}
private Task<User> GetCurrentUserAsync() =>
_userManager.GetUserAsync(HttpContext.User);
}
}

```

В конструкторе получают сервисы UserManager и сервис SignInManager, которые аутентифицируют пользователя и устанавливать или удалять его cookie. Метод _userManager.CreateAsync добавляет в базу данных нового пользователя. Результат выполнения метода представляет класс IdentityResult. В случае если переданные параметры пользователя (электронная почта и пароли) не удовлетворяют требованиям, тогда он не будет добавлен. При удачном добавлении пользователя метод signInManager.SignInAsync() устанавливаем аутентификационные cookie для добавленного пользователя. При неудачном добавлении пользователя формируется ответ, содержащий все возникшие ошибки.

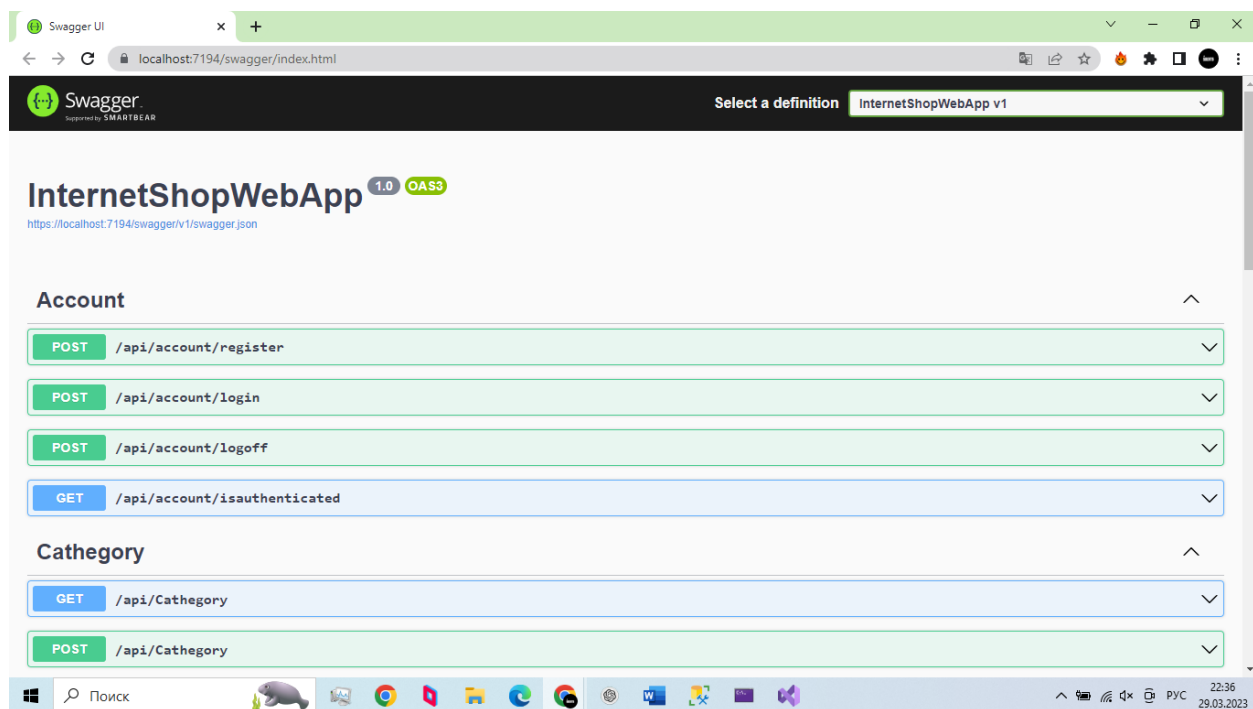


Рисунок 3 – Результаты запуска приложения

2.3. Проверить функцию регистрации

Заголовок Set-Cookie говорит о том, что нужно сохранить cookie у пользователя и указывает название и содержимое. Кроме этого, при установке cookie заполняются

свойства домен, путь, отправка, создано и срок действия. Сохраненные cookie передаются при запросах к домену. При следующих запросах сервер считывает информацию из cookie и проверяет валидность данных.

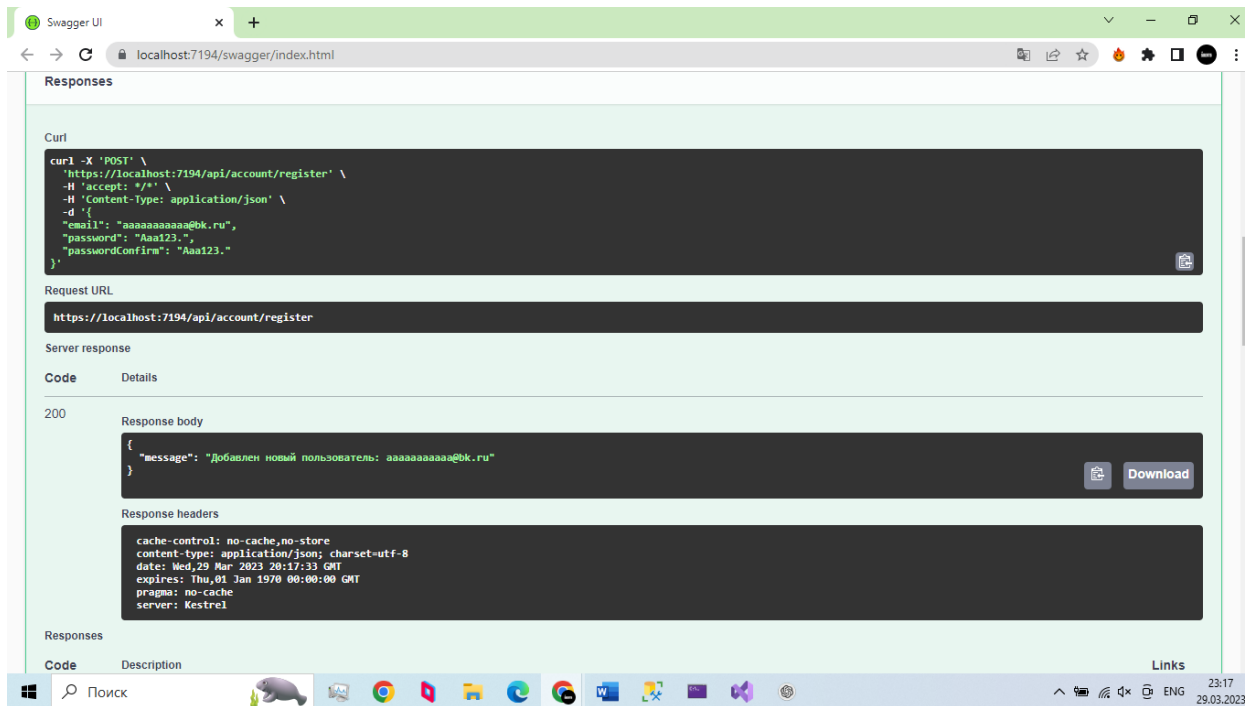


Рисунок 4 – Результаты регистрации

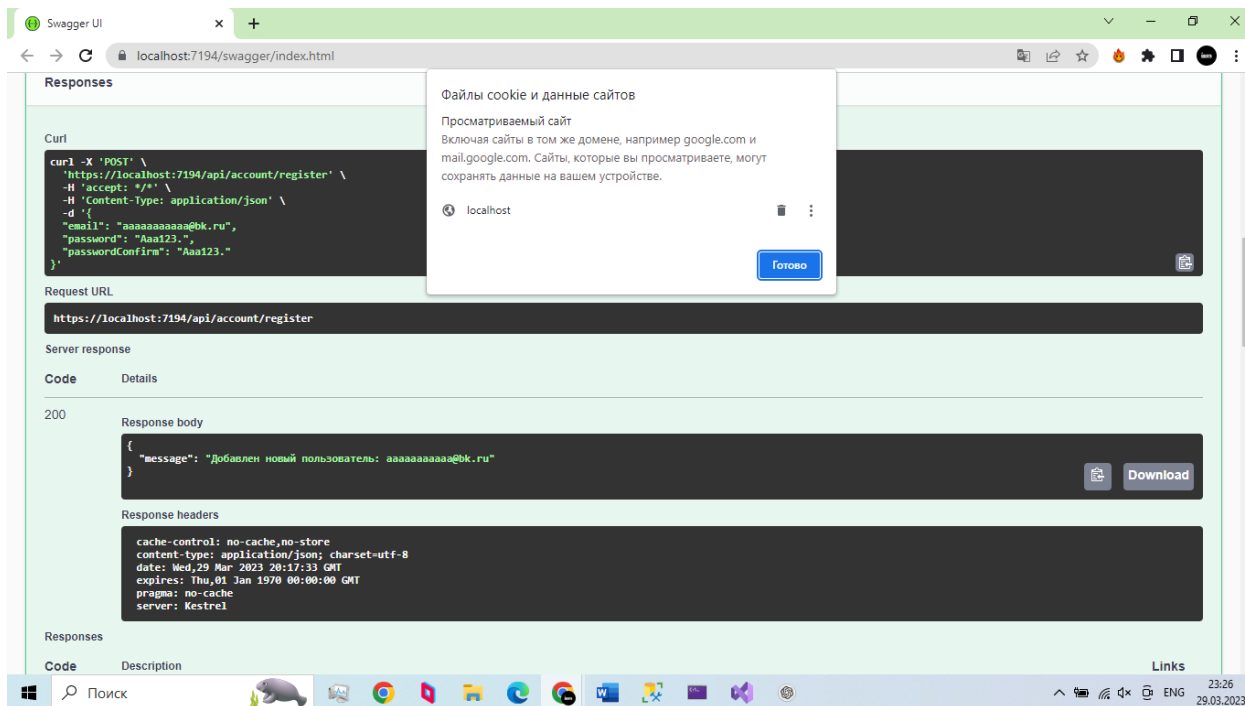


Рисунок 5 – Проверка куки

2.4. Проверить функцию входа

Выполним авторизацию:

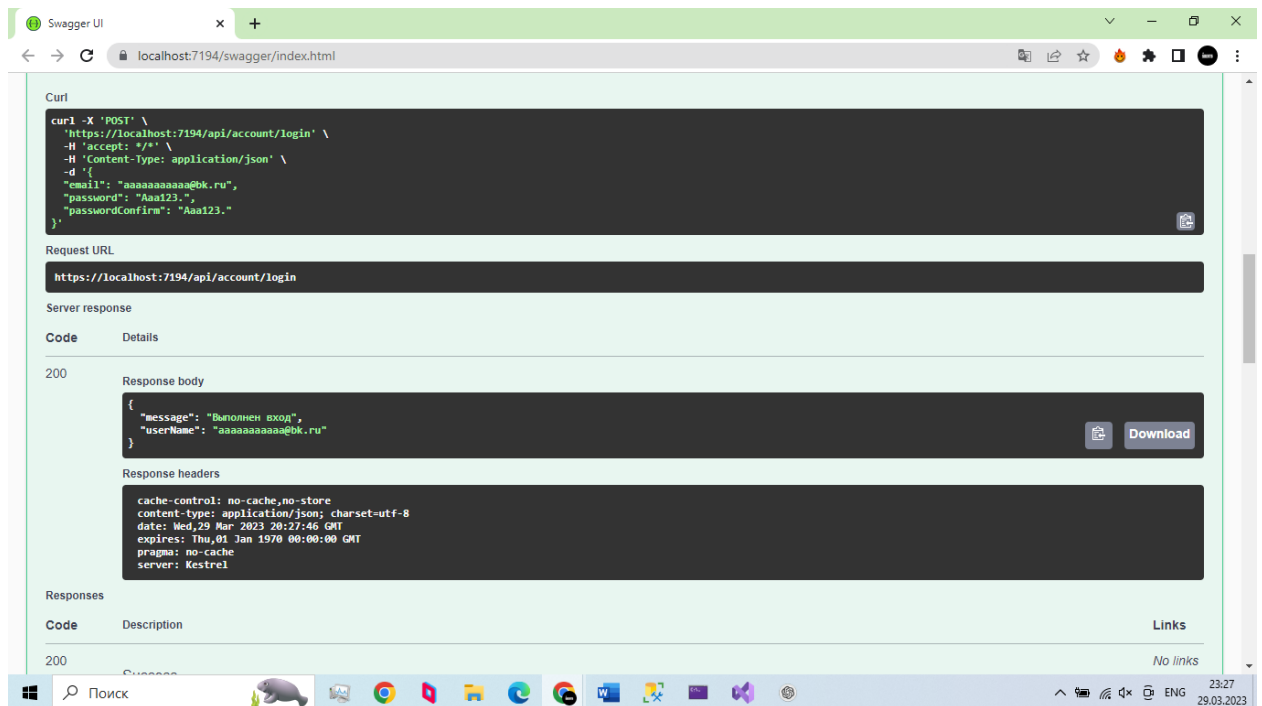


Рисунок 6 – Проверка авторизации

2.5. Проверить функцию выхода

Выполним выход:

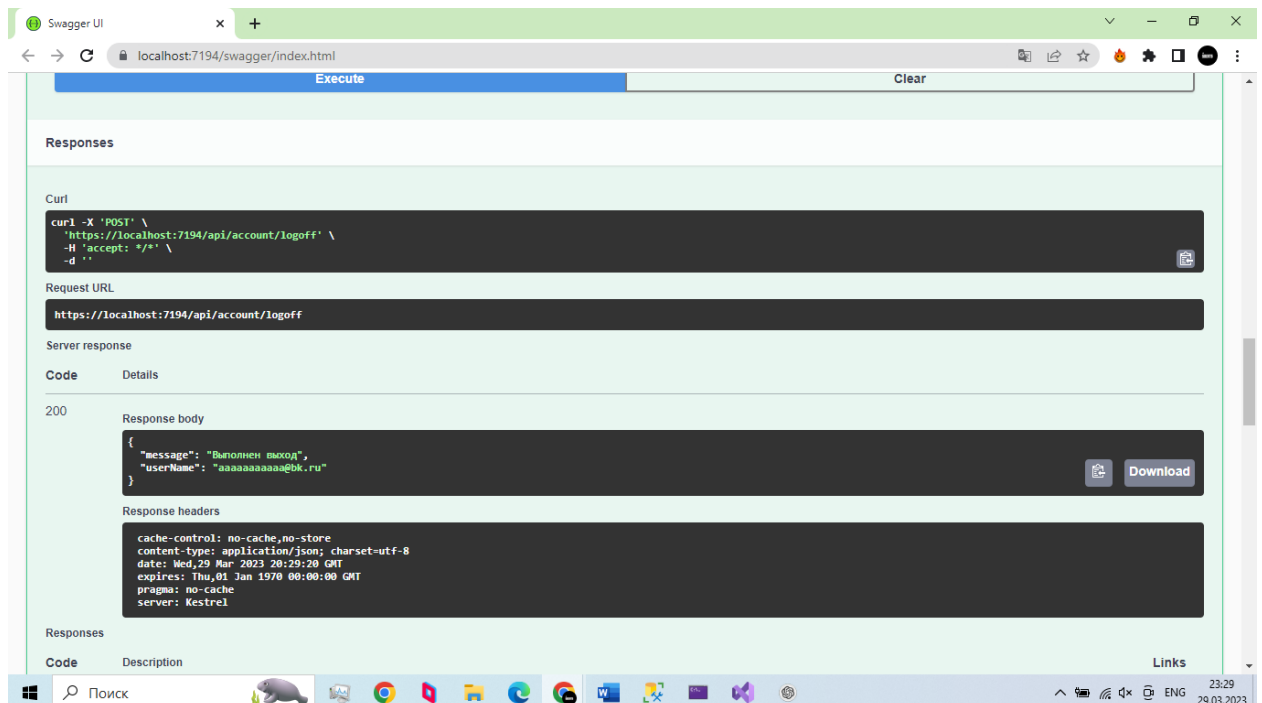


Рисунок 7 – Проверка выхода из аккаунта

2.6. Проверить функцию проверки текущей сессии

Проверим, авторизованы ли мы:

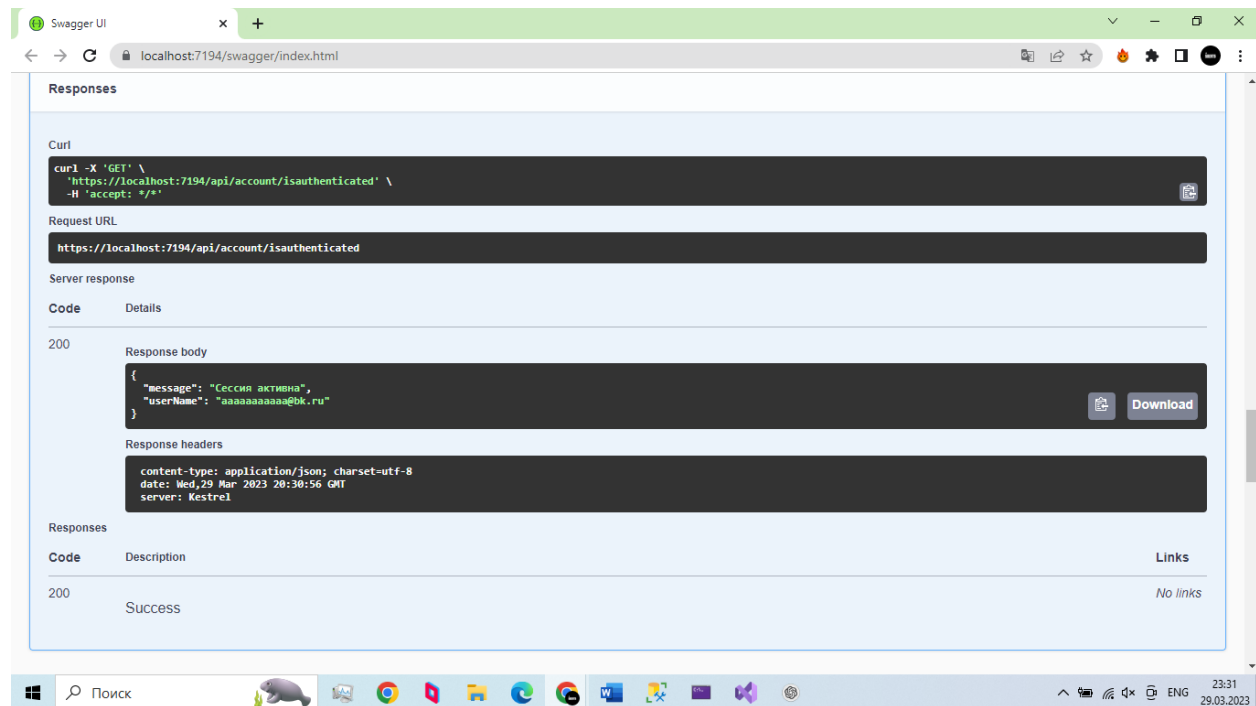


Рисунок 8 – Проверка авторизации

3. Добавить в клиентскую часть с маршрутизацией по страницам, регистрацию и аутентификацию пользователей

3.1. Добавить помощников по коду

Например, приложение для vs code

Prettier - Code formatter

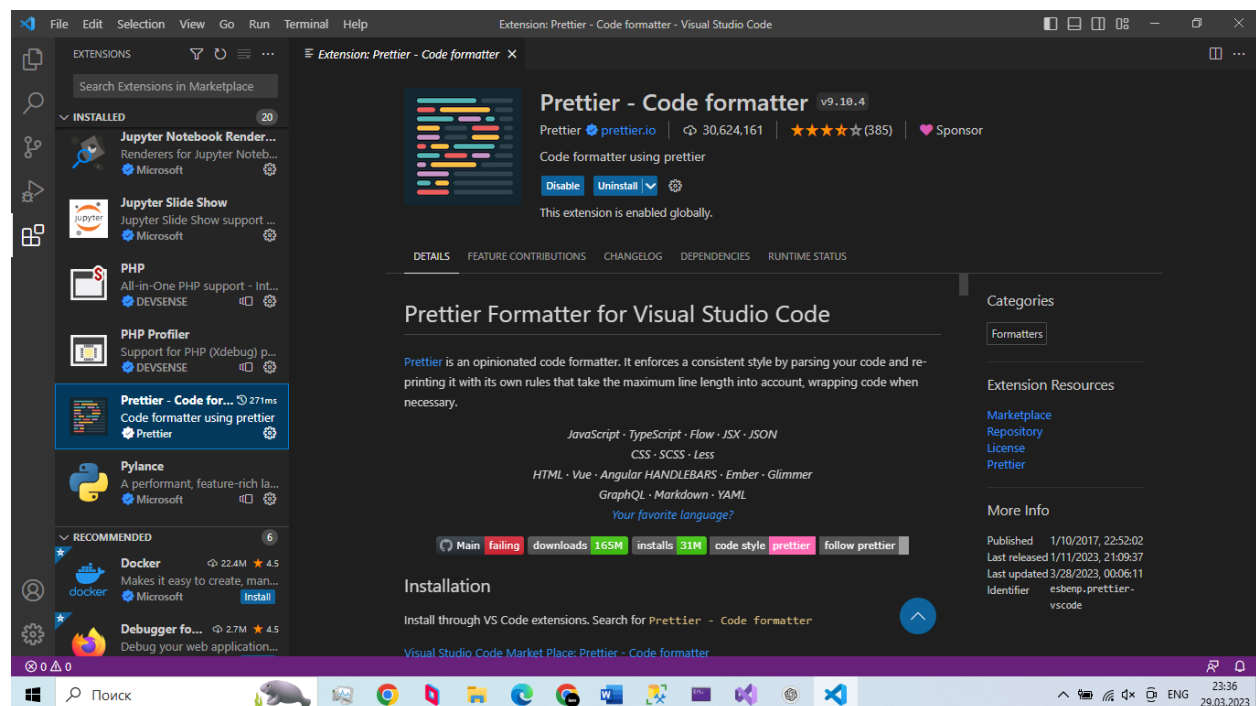


Рисунок 9 – Установка Prettier

ESLint

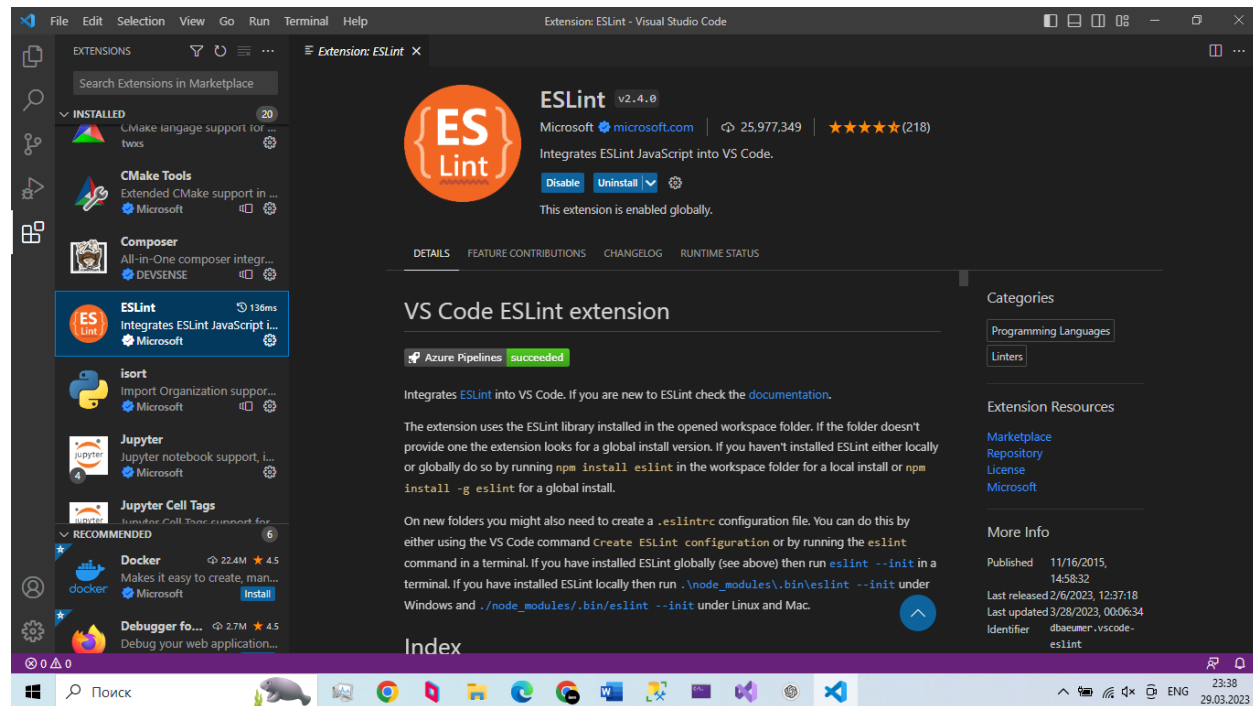


Рисунок 9 – Установка ESLint

Потом необходимо установить в коде с помощью этих команд:

```
npm i -D eslint
```

```
npm init @eslint/config
```

Для исключения ошибки по типам props можно добавить в .eslintrc.js

```
"rules": {  
  "react/prop-types": "off"  
}
```

3.2. Добавить проксирование запросов к серверу

В package.json была добавлена следующая строчка:

```
{  
  "name": "f-internet-shop-web-app",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "@testing-library/jest-dom": "^5.16.5",  
    "@testing-library/react": "^13.4.0",  
    "@testing-library/user-event": "^13.5.0",  
    "react": "^18.2.0",  
    "react-dom": "^18.2.0",
```

```
"react-scripts": "5.0.1",
"web-vitals": "^2.1.4"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
},
"devDependencies": {
  "eslint": "^8.36.0",
  "eslint-config-google": "^0.14.0",
  "eslint-plugin-react": "^7.32.2",
  "react-router-dom": "^6.9.0"
```

```

},
"proxy": "https://localhost:7194"
}

```

Проект был перезапущен, во всех API я обращался к серверу по относительному пути: return await fetch('api/account/login', requestOptions,)

3.3. Добавить маршрутизация по страницам и отображение информации о пользователе

Для начала необходимо установить пакет react-router-dom в devDependencies с помощью команды:

```
npm i -D react-router-dom
```

После установки в index.js нужно внести изменения:

```

import React, {useState} from 'react';
import ReactDOM from 'react-dom/client';
import {BrowserRouter, Route, Routes} from 'react-router-dom';

import OrderItem from './Components/OrderItem/OrderItem';
import OrderItemCreate from './Components/OrderItemCreate/OrderItemCreate';
import Layout from './Components/Layout/Layout';
import LogIn from './Components/Authorization/LogIn';

const App = () => {
  const [OrderItems, setOrderItems] = useState([]);
  const addOrderItem = (OrderItem) => setOrderItems([...OrderItems, OrderItem]);
  const removeOrderItem = (removeId) =>
    // eslint-disable-next-line camelcase
    setOrderItems(OrderItems.filter(({order_Item_Code}) =>
      // eslint-disable-next-line camelcase
      order_Item_Code !== removeId));
  const [user, setUser] = useState({isAuthenticated: false, userName: ''});

  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout user={user} />} />
        <Route index element={<h3>Главная страница</h3>} />
        <Route
          path="/OrderItems"
          element={
            <>

```

```

        <OrderItemCreate user={user} addOrderItem={addOrderItem} />
      <OrderItem
        user={user}
        OrderItems={OrderItems}
        setOrderItems={setOrderItems}
        removeOrderItem={removeOrderItem}
      />
    </>
  }
/>
<Route
  path="/login"
  element={<LogIn user={user} setUser={setUser} />}
/>
  <Route path="*" element={<h3>404</h3>} />
</Route>
</Routes>
</BrowserRouter>
);
};

```

```

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  // <React.StrictMode>
  <App />,
  // </React.StrictMode>
);

```

После изменений я создал компонент layout:

```

import React from 'react';
import {Outlet, Link} from 'react-router-dom';
const Layout = ({user}) => {
  return (
    <>
      <div>
        {user.isAuthenticated ? (
          <h4>Пользователь: {user.userName}</h4>
        ) : (
          <h4>Пользователь: Гость</h4>
        )}
      </div>
      <nav>
        <Link to="/">Главная</Link> <span> </span>
        <Link to="/OrderItems">Строки заказа</Link> <span> </span>
        <Link to="/login">Вход</Link> <span> </span>
      </nav>
      <Outlet />
    </>
  );
};

```

```
export default Layout;
```

3.4. Создать компонент входа

Далее я создал папку LogIn и в ней создал новый компонент авторизации LogIn.js:

```
import React, {useState} from 'react';
import {useNavigate} from 'react-router-dom';
const LogIn = ({user, setUser}) => {
  const [errorMessages, setErrorMessages] = useState([]);
  const navigate = useNavigate();
  const logIn = async (event) => {
    event.preventDefault();
    const {email, password} = document.forms[0];
    // console.log(email.value, password.value)
    const requestOptions = {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({
        email: email.value,
        password: password.value,
        passwordConfirm: password.value,
      }),
    };
    return await fetch(
      'api/account/login',
      requestOptions,
    )
      .then((response) => {
        // console.log(response.status)

        response.status === 200 &&
        setUser({isAuthenticated: true, userName: ''});
        return response.json();
      })
      .then(
        (data) => {
          console.log('Data:', data);
          if (
            typeof data !== 'undefined' &&
            typeof data.userName !== 'undefined'
          ) {
            setUser({isAuthenticated: true, userName: data.userName});
            navigate('/');
          }
          typeof data !== 'undefined' &&
          typeof data.error !== 'undefined' &&
          setErrorMessages(data.error);
        },
        (error) => {
          console.log(error);
        },
      );
  };
};
```



```

    };
    const renderErrorMessage = () =>
      errorMessages.map((error, index) => <div key={index}>{error}</div>);
    return (
      <>
        {user.isAuthenticated ? (
          <h3>Пользователь {user.userName} успешно вошел в систему</h3>
        ) : (
          <>
            <h3>Вход</h3>
            <form onSubmit={logIn}>
              <label>Пользователь </label>
              <input type="text" name="email" placeholder="Логин" />
              <br />
              <label>Пароль </label>
              <input type="text" name="password" placeholder="Пароль" />
              <br />
              <button type="submit">Войти</button>
            </form>
            {renderErrorMessage()}
          </>
        )}
      </>
    );
  };
};
export default LogIn;

```

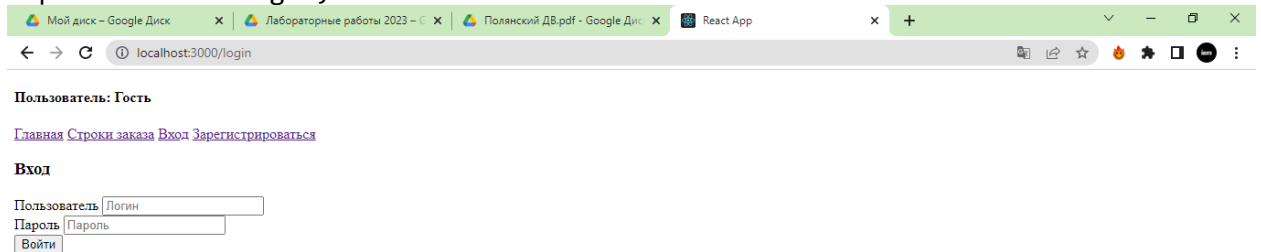


Рисунок 10 – Результат запуска приложения

navigate("/") после успешного входа происходит навигация на главную страницу. В случае ошибок выводится их список.

3.5. Создать компонент завершения сессии

Был создан компонент LogOut в папке Authorization:

```
import React from 'react';
import {useNavigate} from 'react-router-dom';

const LogOff = ({setUser}) => {
  const navigate = useNavigate();
  const logOff = async (event) => {
    event.preventDefault();

    const requestOptions = {
      method: 'POST',
    };

    return await fetch("api/account/logoff", requestOptions)
      .then((response) => {
        response.status === 200 &&
        setUser({isAuthenticated: false, userName: ''});
        response.status === 401 && navigate('/login');
      });
  };

  return (
    <>
      <p></p>
      <form onSubmit={logOff}>
        <button type="submit">Выход</button>
      </form>
    </>
  );
};

export default LogOff;
```

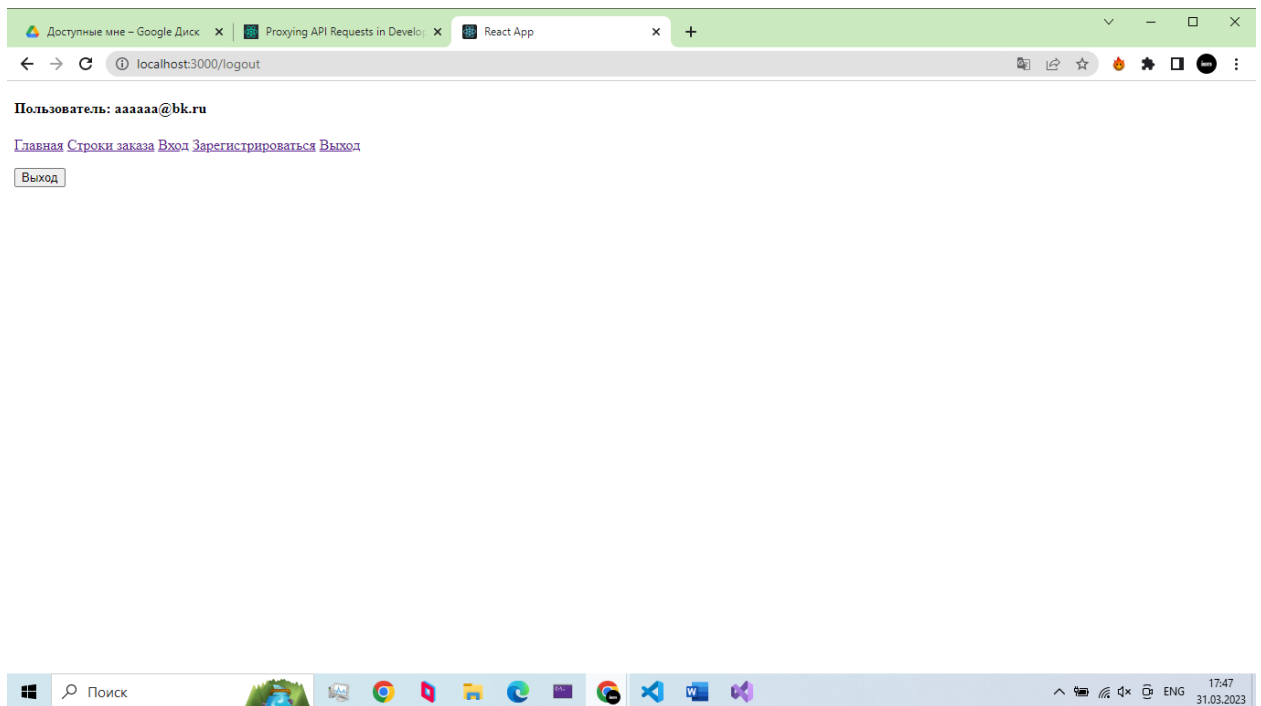


Рисунок 11 – Приложение до выхода из аккаунта

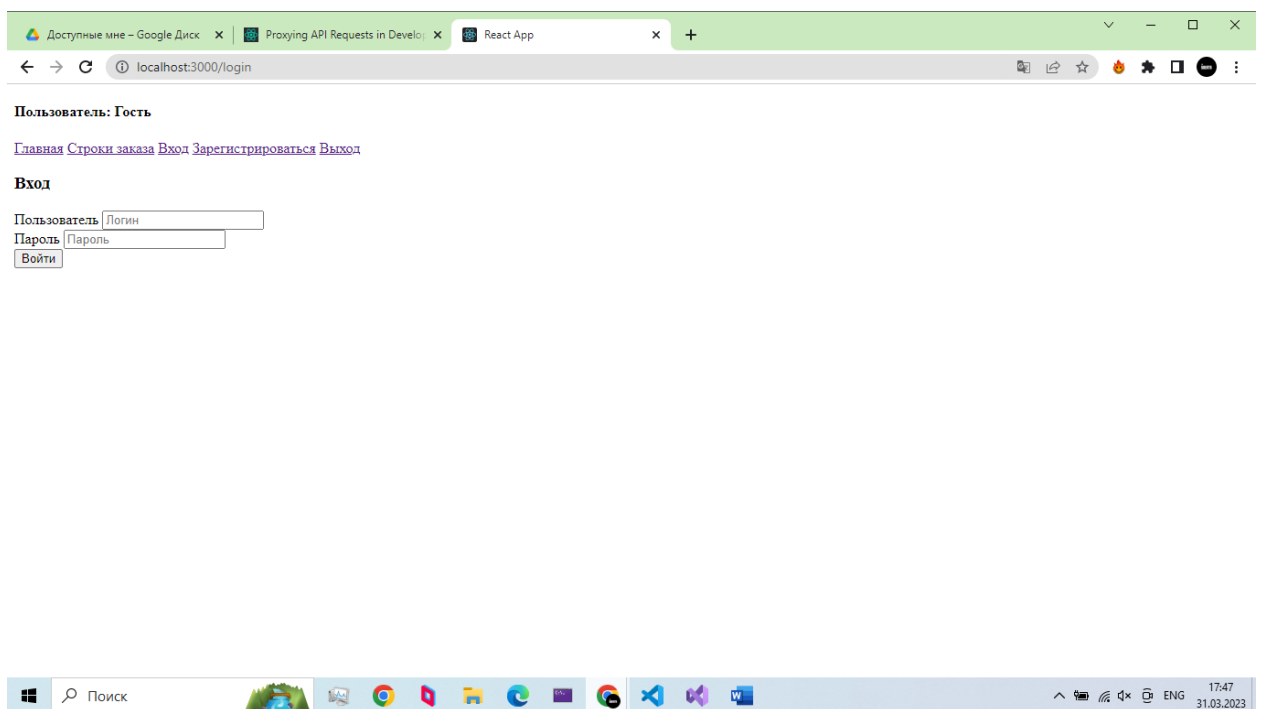


Рисунок 12 – Результат выхода из аккаунта

3.6. Ограничить функции для Гость

Далее необходимо внести изменения в компоненты `OrderItem` и `OrderItemCreate`

```
const OrderItem = ({user, OrderItems, setOrderItems, removeOrderItem}) => {
  useEffect(() => {
    const getOrderItems = async () => {
      const requestOptions = {
        method: 'GET',
```



Рисунок 13 – Результат ограничения возможностей гостя

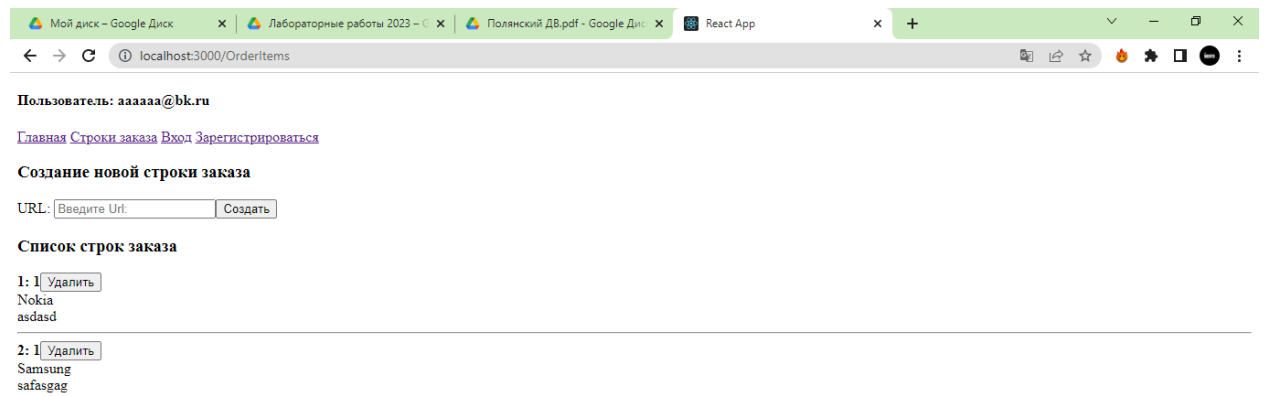


Рисунок 14 – Результат возможностей авторизованного пользователя

3.7. Создать компонент регистрации

Для того, чтобы реализовать регистрацию нужно сначала создать компонент Register:

```
import React, {useState} from 'react';
import {useNavigate} from 'react-router-dom';

const Register = ({user, setUser}) => {
  const [errorMessages, setErrorMessages] = useState([]);
  const [registrationSuccess, setRegistrationSuccess] = useState(false);

  const navigate = useNavigate();
  const register = async (event) => {
    event.preventDefault();
    const {email, password, reppassword} = document.forms[0];
    // console.log(email.value, password.value)
    const requestOptions = {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({
        email: email.value,
        password: password.value,
        passwordConfirm: reppassword.value,
      }),
    };
    return await fetch(
      'api/account/register',
      requestOptions,
    )
      .then((response) => {
        // console.log(response.status)

        response.status === 200 &&
        setUser({isAuthenticated: true, userName: email.value});
        return response.json();
      })
      .then(
        (data) => {
          console.log('Data:', data);
          if (
            typeof data !== 'undefined' &&
            typeof data.userName !== 'undefined'
          ) {
            setUser({isAuthenticated: true, userName: data.userName});
            setRegistrationSuccess(true); // <-- добавьте эту строку
            navigate('/');
          }
          typeof data !== 'undefined' &&
          typeof data.error !== 'undefined' &&
          setErrorMessages(data.error);
        }
      )
  };
};
```

```

    },
    (error) => {
      console.log(error);
    },
  );
};
const renderErrorMessage = () =>
  errorMessages.map((error, index) => <div key={index}>{error}</div>);
return (
  <>
    {user.isAuthenticated ? (
      <h3>Пользователь {user.userName} уже вошел в систему</h3>
    ) : (
      <>
        <h3>Регистрация</h3>
        <form onSubmit={register}>
          <label>Пользователь </label>
          <input type="text" name="email" placeholder="Логин" />
          <br />
          <label>Пароль </label>
          <input type="text" name="password" placeholder="Пароль" />
          <br />
          <label>Повторите Пароль </label>
          <input type="text" name="reppassword"
            placeholder="Пароль" />
          <br />
          <button type="submit">Зарегистрироваться</button>
        </form>
        {registrationSuccess && (
          // eslint-disable-next-line max-len
          <p>Регистрация прошла успешно. Вы будете перенаправлены на главную
страницу.</p>
        )}
        {renderErrorMessage()}
      </>
    )}
  </>
);
};
export default Register;

```

После того, как мы создали его необходимо подсоединить его в index.js

```

import OrderItem from './Components/OrderItem/OrderItem';
import OrderItemCreate from './Components/OrderItemCreate/OrderItemCreate';
import Layout from './Components/Layout/Layout';
import LogIn from './Components/Authorization/LogIn';
import Register from './Components/Authorization/Register';

```

...

```

<BrowserRouter>
  <Routes>
    <Route path="/" element={<Layout user={user} />} />
    <Route index element={<h3>Главная страница</h3>} />
    <Route
      path="/OrderItems"
      element={
        <>
          <OrderItemCreate user={user} addOrderItem={addOrderItem} />
          <OrderItem
            user={user}
            OrderItems={OrderItems}
            setOrderItems={setOrderItems}
            removeOrderItem={removeOrderItem}
          />
        </>
      }
    />
    <Route
      path="/login"
      element={<LogIn user={user} setUser={setUser} />}
    />
    <Route
      path="/register"
      element={<Register user={user} setUser={setUser} />}
    />
    <Route path="*" element={<h3>404</h3>} />
  </Routes>
</BrowserRouter>

```

После того, как мы добавили маршрутизацию, нужно добавить кнопку для перехода в layout

```

<nav>
  <Link to="/">Главная</Link> <span> </span>
  <Link to="/OrderItems">Строки заказа</Link> <span> </span>
  <Link to="/login">Вход</Link> <span> </span>
  <Link to="/register">Зарегистрироваться</Link> <span> </span>
</nav>

```

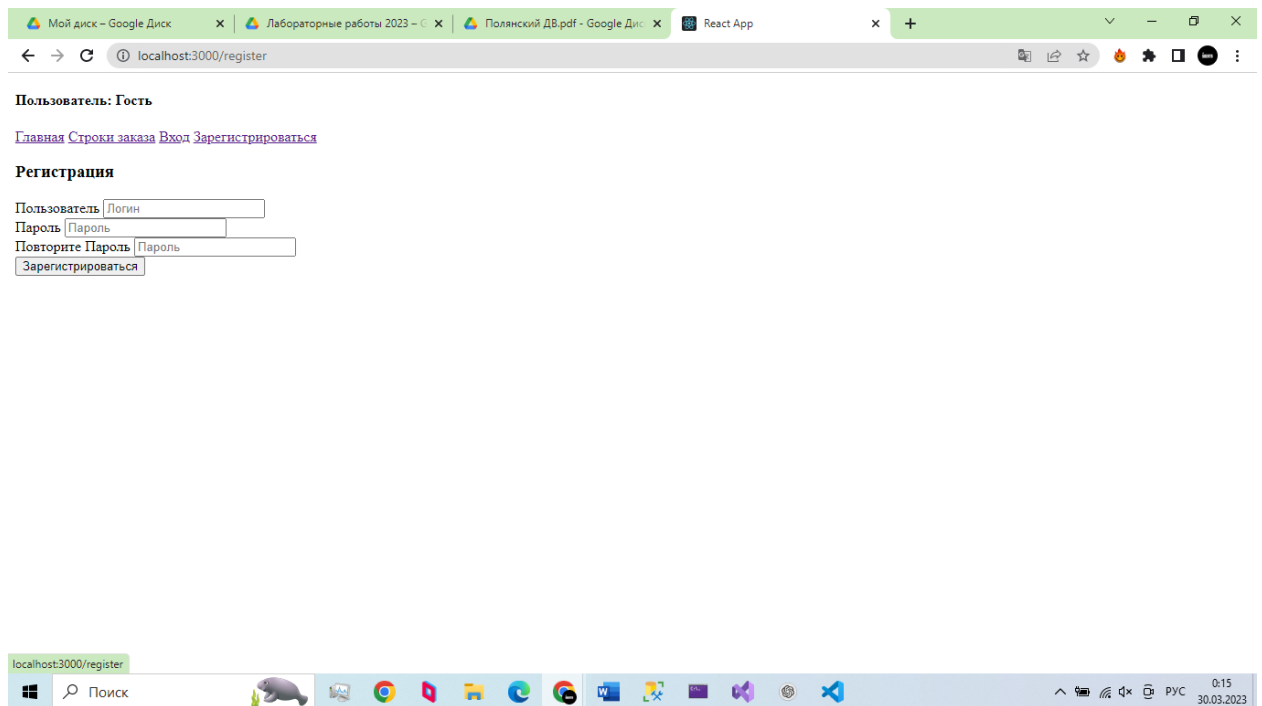


Рисунок 15 – Результат добавления регистрации

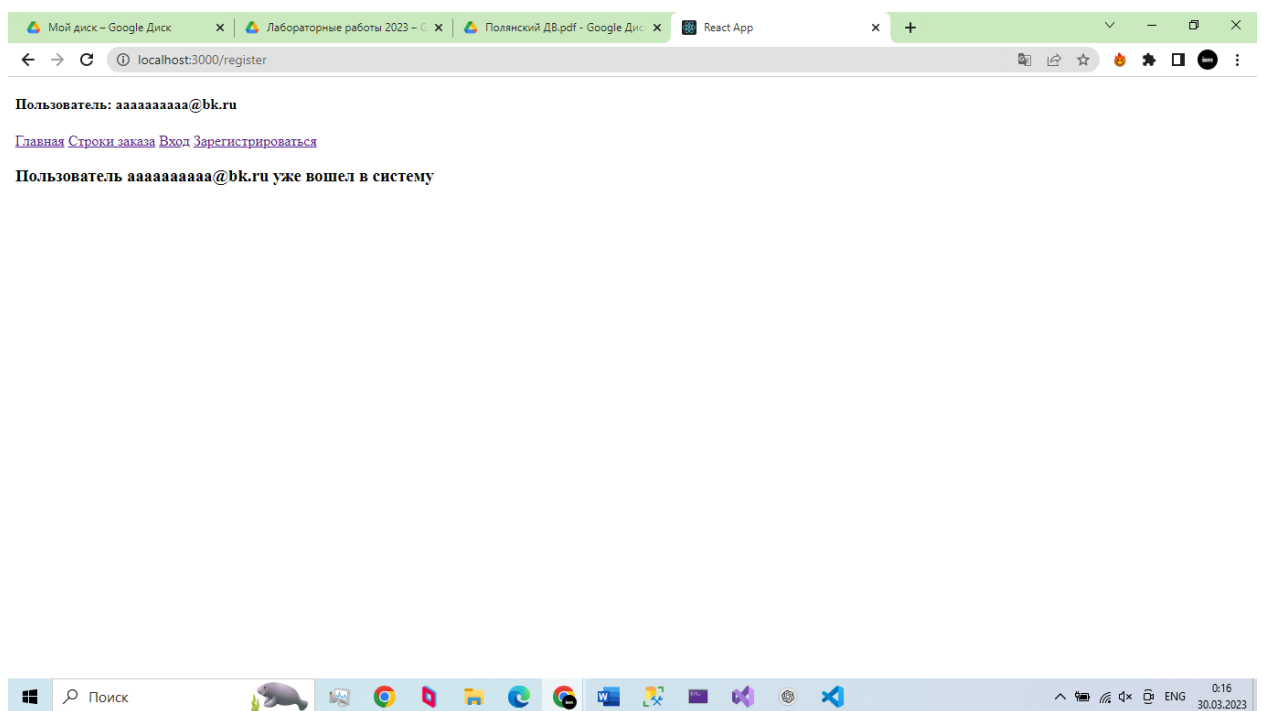


Рисунок 16 – Результат добавления нового пользователя

Вывод

В ходе выполнения лабораторной работы я добавил возможности регистрации и аутентификации пользователей.