Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №4,5

на тему

**РАЗРАБОТКА ЗАЩИЩЁННЫХ ПРИЛОЖЕНИЙ**

Выполнил Толстой Д.В.

Проверил Лещенко Е. А.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc160203598)

[2 Результат выполнения 4](#_Toc160203599)

[Заключение 7](#_Toc160203600)

[Приложение А Листинг кода 8](#_Toc160203601)

# 1 ЦЕЛЬ РАБОТЫ

Целью работы является ознакомление с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз. Получение знаний для разработки приложений, которые используют ролевое управление доступом для разграничения полномочий пользователей. Приобритение навыков защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В ходе выполнения лабораторной работы было создано оконное приложение, позволяющее опубликовывать посты, которые могу увидеть другие пользователи. На рисунке 1 показана страница с постами.

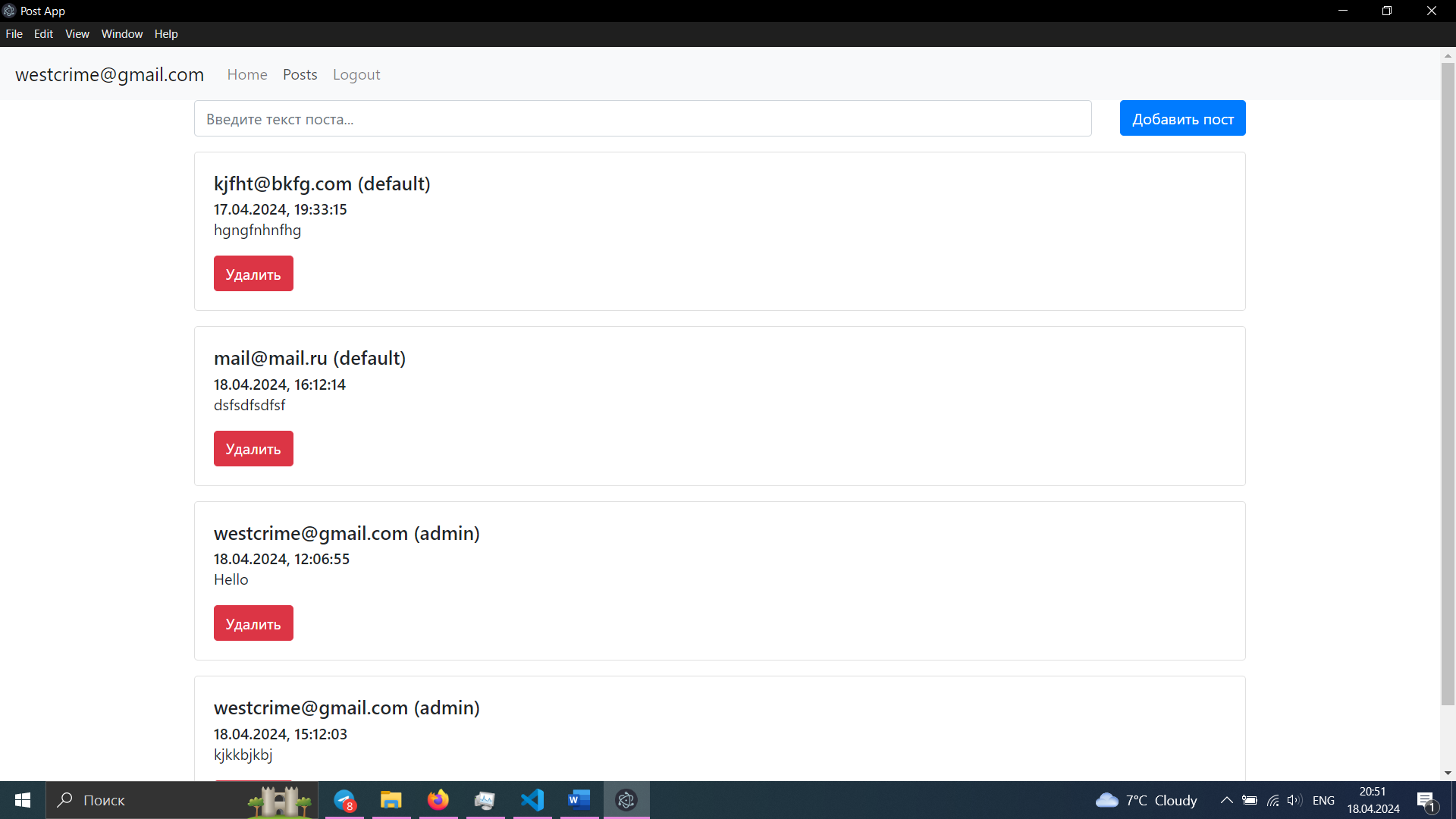


Рисунок 1 – Страница с постами

Реализована авторизация и регистрация пользователей с помощью почты и пароля. Проверка данных для защиты от атак типа *XXS* (рисунок 2), *DOS* атак (ограничение по времени при определенном количестве неверных попыток), защита от *sql*-инъекций (рисунок 3). Также присутствует защита от переполнения буфера путем контроля ввода пользователя (рисунок 4).

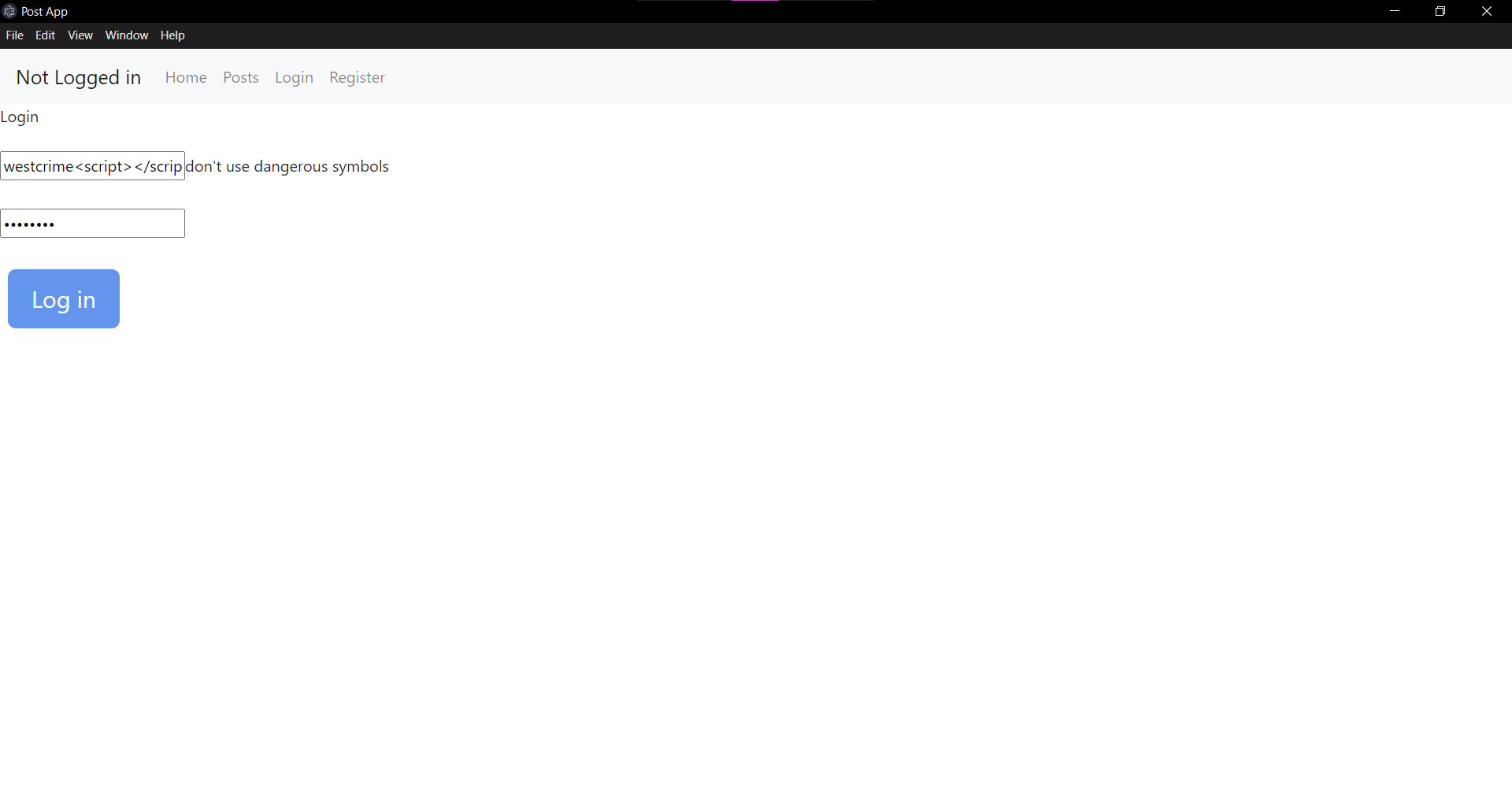


Рисунок 2 – Защита от *Xxs* атак

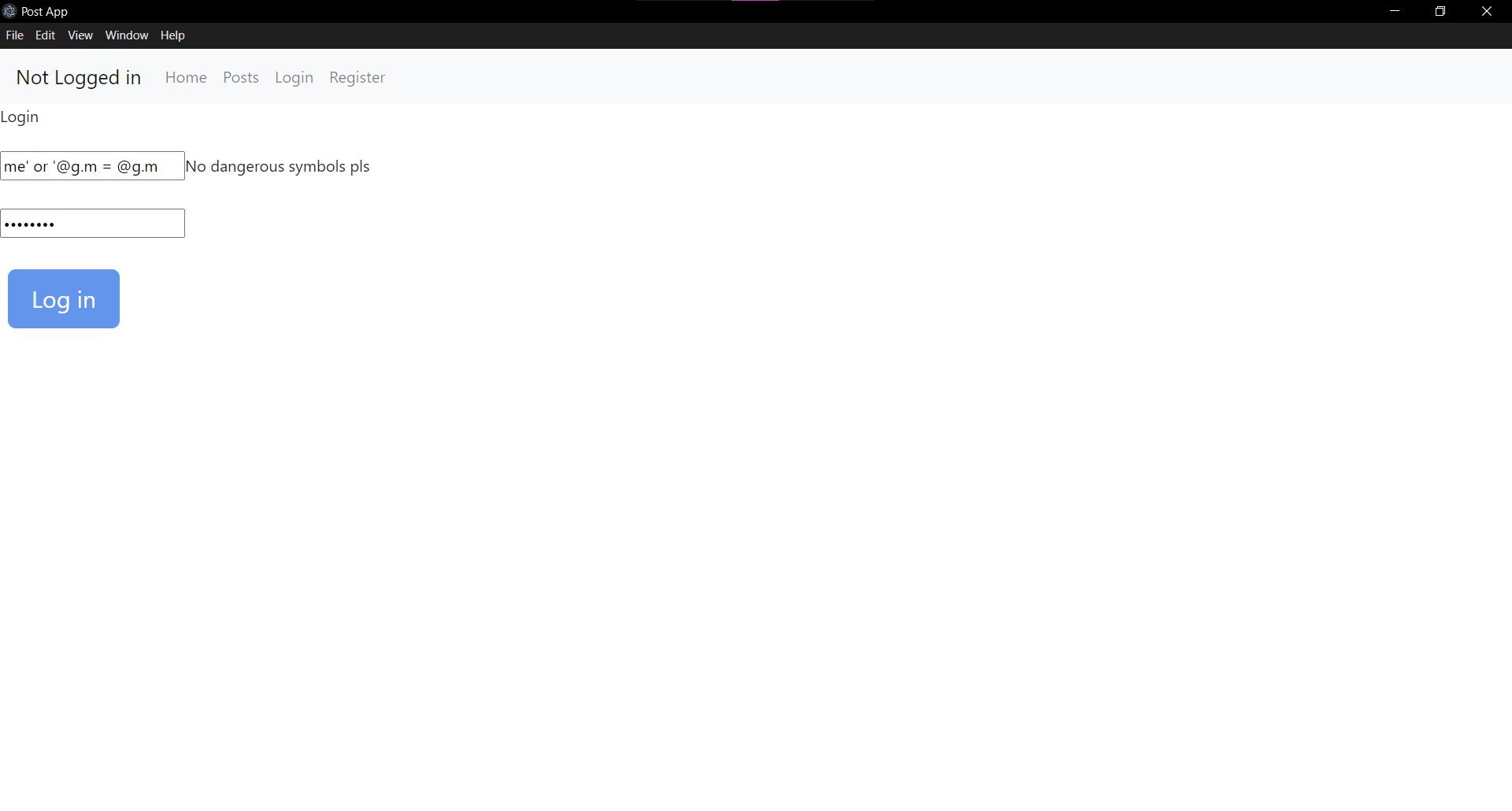


Рисунок 3 – Защита от *sql*-инъекций



Рисунок 4 – Защита от переполнения буфера

Также использован принцип разделения привилегий, то есть только пользователи с ролью админа могут удалять посты, и только авторизированные пользователи могут их писать (рисунок 5).

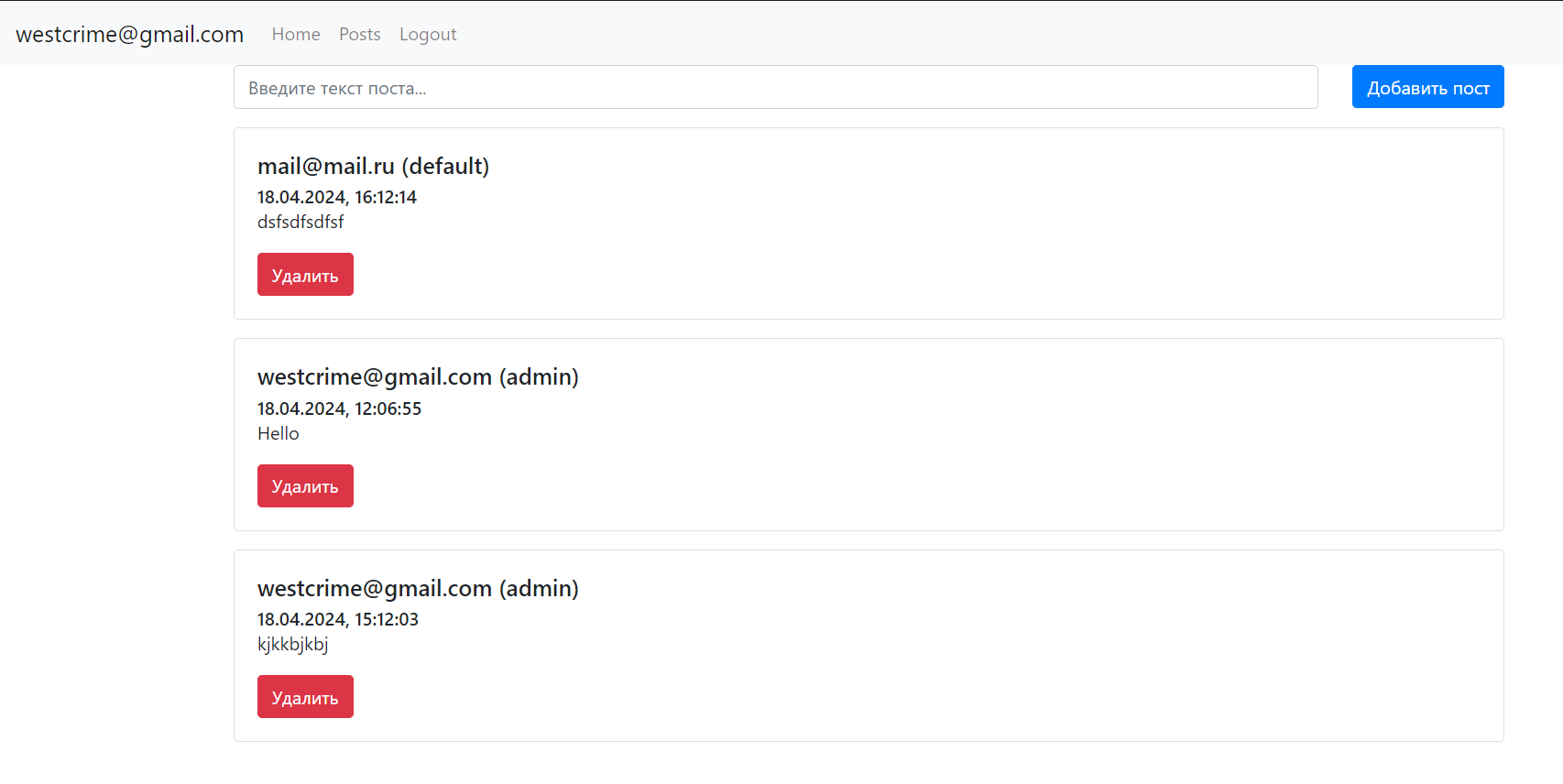


Рисунок 5 – Возможности пользователя с ролью администратора

Был разработан защищенный инсталлятор для установки приложения путем запуска одного исходного файла (рисунок 6). После этого, приложение может быть запущено как и любое другое приложение.

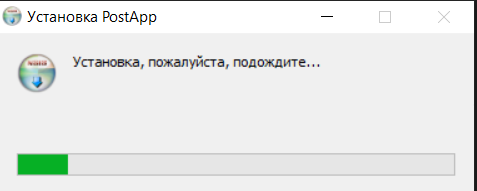


Рисунок 6 – Инсталлятор приложения

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы выполнены ознакомление с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз. Получение знаний для разработки приложений, которые используют ролевое управление доступом для разграничения полномочий пользователей. Приобритение навыков защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

# ПРИЛОЖЕНИЕ А Листинг кода

main.js

import { app, BrowserWindow, protocol } from 'electron'

import { ipcMain } from 'electron/main'

import path from 'node:path'

import url from 'node:url'

import credentials from "./key.json" with { type: "json" };

import admin from "firebase-admin"

import login from "./auth/login.js";

import register from "./auth/register.js";

import verify from './auth/verify.js';

import { fileURLToPath } from 'url';

import { dirname } from 'path';

import \* as dotenv from 'dotenv'

import checkAccount from './auth/checkAccount.js';

import getPosts from './posts/getPosts.js';

import putPost from './posts/putPost.js';

import deletePost from './posts/deletePost.js';

dotenv.config()

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = dirname(\_\_filename);

const createWindow = async () => {

const win = new BrowserWindow({

width: 1600,

height: 900,

webPreferences: {

contextIsolation: true,

nodeIntegration: false,

sandbox: true,

preload: path.join(\_\_dirname, 'preload.js')

}

});

win.webContents.openDevTools();

const startUrl = url.format({

pathname: path.join(\_\_dirname, './app/build/index.html'),

protocol: 'file'

});

win.loadURL(startUrl);

}

app.whenReady().then(async () => {

await createWindow();

const jwtSecretKey = 'dsfdsfsdfdsvcsvdfgefg';

admin.initializeApp({

credential: admin.credential.cert(credentials)

});

const db = admin.firestore();

ipcMain.on('login', async (event, args) => {

console.log(args);

const response = JSON.stringify(await login(args, jwtSecretKey, db));

console.log(response);

event.returnValue = response;

return;

});

ipcMain.on('verify', async (event, args) => {

console.log(args);

const response = JSON.stringify(await verify(args.token, jwtSecretKey));

console.log(response);

event.returnValue = response;

return;

});

ipcMain.on('checkAccount', async (event, args) => {

console.log(args);

const response = JSON.stringify(await checkAccount(db, args.email));

console.log(response);

event.returnValue = response;

return;

});

ipcMain.on('register', async (event, args) => {

const response = await register(args, jwtSecretKey, db);

event.returnValue = JSON.stringify(response);

return;

});

ipcMain.on('getPosts', async (event, args) => {

const response = await getPosts(db);

event.returnValue = JSON.stringify(response);

return;

});

ipcMain.on('postPost', async (event, args) => {

let response = {};

const isVerified = (await verify(args.token, jwtSecretKey)).success;

if (isVerified) {

if (args.email === undefined) {

response = {success: false, description: 'No email was sent'};

event.returnValue = JSON.stringify(response);

return;

}

const result = await checkAccount(db, args.email);

const user = result.user;

const post = args;

if (post.date === undefined || post.text === undefined || user.role === undefined) {

response = {success: false, description: 'Wrong info format'};

event.returnValue = JSON.stringify(response);

return;

}

if ((await verify(args.token, jwtSecretKey)).success)

{

response = await putPost(db, user, post);

event.returnValue = JSON.stringify(response);

return;

} else {

response = {success: false, description: 'Error while verifying token'};

event.returnValue = JSON.stringify(response);

return;

}

}

});

ipcMain.on('deletePost', async (event, args) => {

let response = {};

const isVerified = (await verify(args.token, jwtSecretKey)).success;

if (isVerified) {

if (args.email === undefined) {

response = {success: false, description: 'No email was sent'};

event.returnValue = JSON.stringify(response);

return;

}

const user = (await checkAccount(db, args.email)).user;

if (user.role !== 'admin') {

response = {success: false, description: 'Only admin can delete posts'};

event.returnValue = JSON.stringify(response);

return;

}

const postId = args.postId;

if (postId !== undefined) {

if ((await verify(args.token, jwtSecretKey)).success)

{

response = (await deletePost(db, postId));

event.returnValue = JSON.stringify(response);

return;

} else {

response = {success: false, description: 'Error while verifying token'};

event.returnValue = JSON.stringify(response);

return;

}

} else {

response = {success: false, description: 'Wrong data format'};

event.returnValue = JSON.stringify(response);

return;

}

}

});

})

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') app.quit();

})

index.jsx

import React, { useState, useEffect } from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import Home from './components/Home.jsx';

import {Route, HashRouter, Routes} from "react-router-dom";

import NoPage from "./components/NoPage.jsx";

import Login from "./components/Login.jsx";

import Logout from "./components/Logout.jsx";

import Posts from "./components/Posts.jsx";

import Register from "./components/Register.jsx";

export default function App() {

const ipcRenderer = window.ipcRenderer;

const [loggedIn, setLoggedIn] = useState(false);

const [email, setEmail] = useState('');

const [role, setRole] = useState('');

useEffect(() => {

// Fetch the user email and token from local storage

const user = JSON.parse(localStorage.getItem('user'))

// If the token/email does not exist, mark the user as logged out

if (!user || !user.token) {

setLoggedIn(false)

return

}

// If the token exists, verify it with the auth server to see if it is valid

ipcRenderer.send('verify', {token: user.token});

ipcRenderer.on('verify', function (evt, message) {

evt = JSON.parse(evt);

setLoggedIn(evt.success);

setEmail(user.email || '');

setRole(user.role || '');

});

}, [])

return (

<HashRouter>

<Routes>

<Route path='/' element={<Home role={role} setRole={setRole} email={email} loggedIn={loggedIn} setLoggedIn={setLoggedIn} />} >

<Route path="/posts" element={<Posts role={role} setRole={setRole} email={email} loggedIn={loggedIn} setLoggedIn={setLoggedIn} setEmail={setEmail}/>} />

<Route path="/login" element={<Login role={role} setRole={setRole} email={email} loggedIn={loggedIn} setLoggedIn={setLoggedIn} setEmail={setEmail} />} />

<Route path="/logout" element={<Logout role={role} setRole={setRole} email={email} loggedIn={loggedIn} setLoggedIn={setLoggedIn} setEmail={setEmail}/>} />

<Route path="/register" element={<Register role={role} setRole={setRole} email={email} loggedIn={loggedIn} setLoggedIn={setLoggedIn} setEmail={setEmail}/>} />

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

</Route>

</Routes>

</HashRouter>

);

}

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

root.render(<App />);