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 №6

на тему

**ЗАЩИТА ПО ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ**

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**

Цель данной лабораторной работы – изучить основные методы защиты программного обеспечения от несанкционированного использования и приобрести навыки обфускации кода приложения для предотвращения его несанкционированного копирования.

Предлагается выбрать и реализовать три метода обфускации программного кода из следующих вариантов: один метод, любые два метода из трех одновременно или все три метода одновременно.

**2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате разработки программы были реализованы на выбор 3 метода обфускации программного кода приложения, а именно: лексическая обфускация, обфускация структур данных, обфускация потока управления.

Обфускация структур данных заключалась в разбиении массивов на отдельные переменные и собирания переменных в массиве. Результат представлен на рисунке 1.



Рисунок 1 – Использование обфускации структур данных

Лексическая обфуксация заключается в изменинии имен идентификаторов в программе и удалении отступов. Результат представлен на рисунке 2.

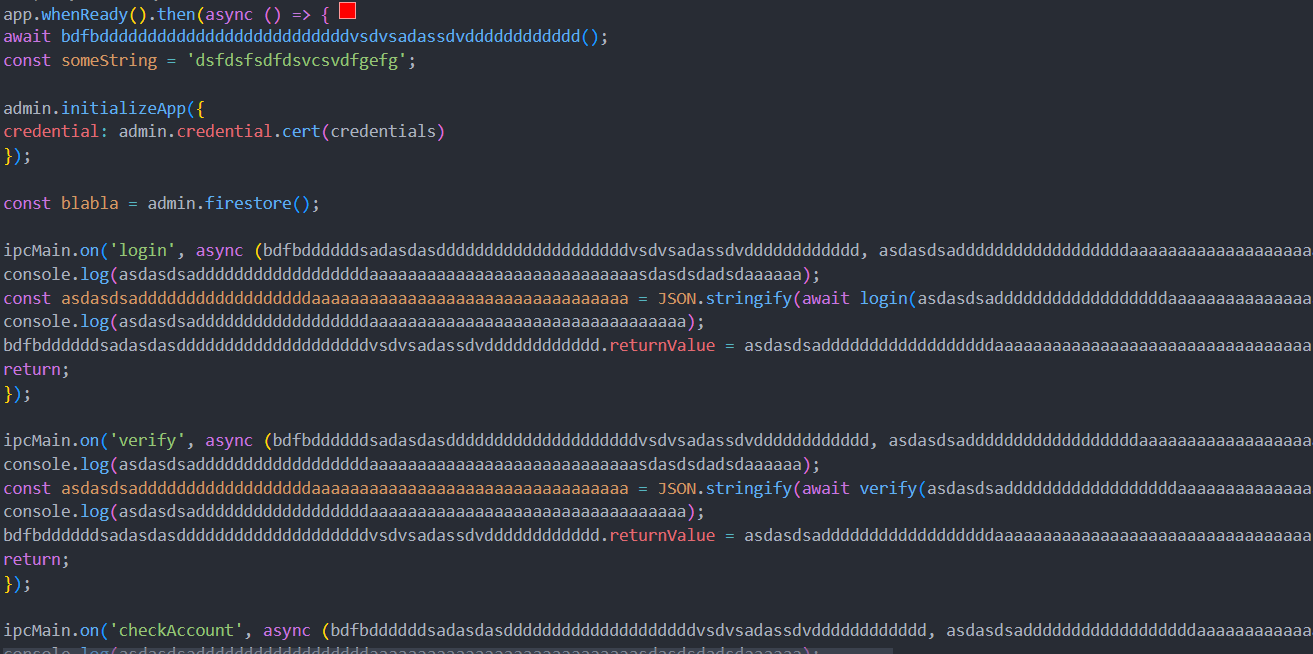


Рисунок 2 – Использование лексической обфуксации

Обфускация потока управления представлена на рисунке и комбинации обфуксаций представлены на рисунках 3-5.



Рисунок 3 – Использование обфускации потока управления



Рисунок 4 – Использование первого и второго метода одновременно

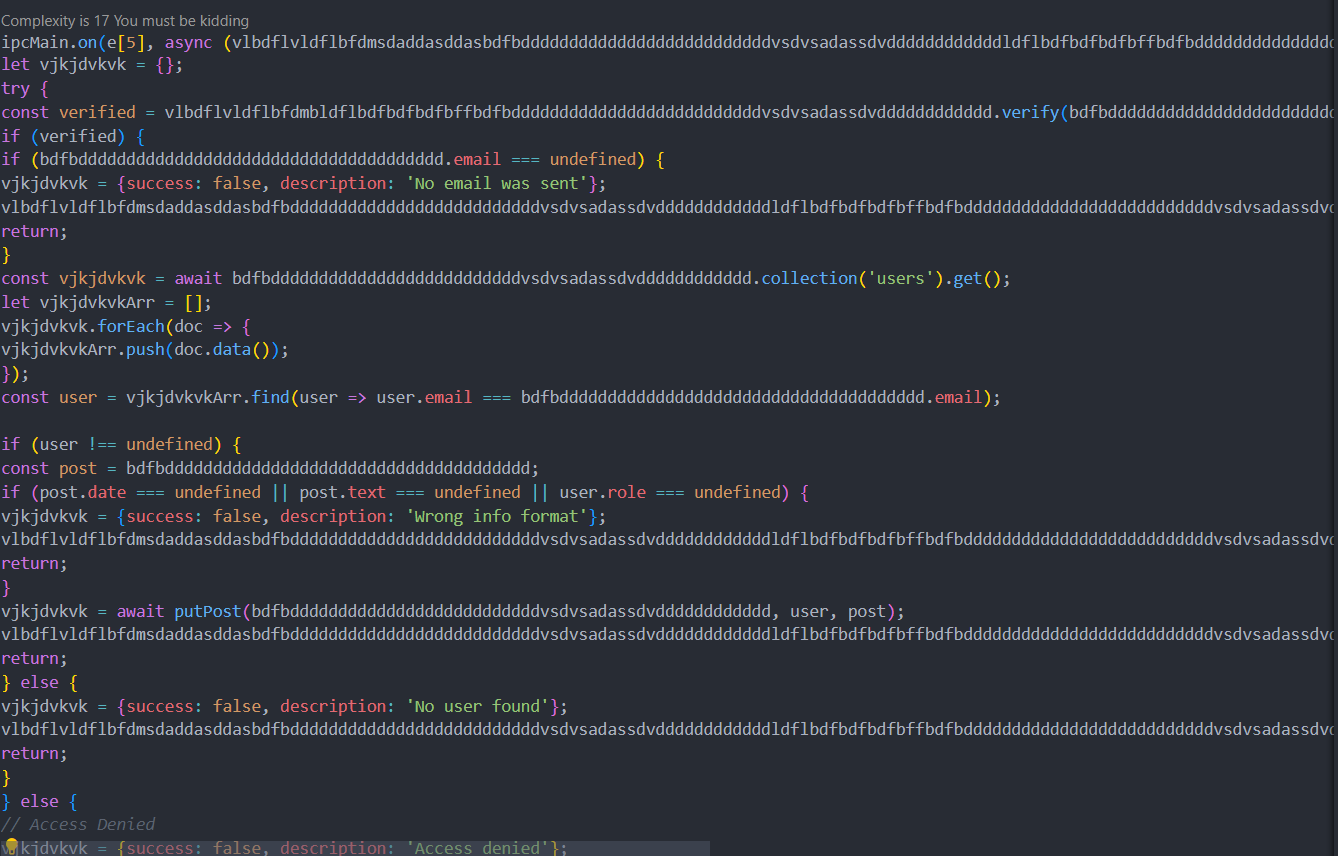


Рисунок 5 – Использование всех трех методов одновременно

**ЗАКЛЮЧЕНИЕ**

В рамках этой лабораторной работы были выбраны и реализованы три метода обфускации программного кода, применяемые для защиты разработанного приложения от несанкционированного использования. Методы могут применяться как по отдельности, так и в комбинациях: один метод, любые два метода из трех одновременно или все три метода одновременно.

# ПРИЛОЖЕНИЕ А Листинг кода

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 vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd from 'jsonwebtoken';

import { v4 as uuidv4 } from 'uuid';

import \* as crypto from 'crypto';

import { fileURLToPath } from 'url';

import { dirname } from 'path';

import \* as dotenv from 'dotenv'

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

dotenv.config()

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

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

const e = ['login', 'verify', 'checkAccount', 'register', 'getPosts', 'postPost', 'deletePost', 'window-all-closed'];

const createWindow = async () => {

const win = new BrowserWindow({

width: 1600,

height: 900,

webPreferences: {

contextIsolation: true,

nodeIntegration: false,

sanbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddox: 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 vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddSecretKey = 'dsfdsfsdfdsvcsvdfgefg';

admin.initializeApp({

credential: admin.credential.cert(credentials)

});

const bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd = admin.firestore();

ipcMain.on(e[0], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

console.log(bdfbdddddddddddddddddddddddddddddddddddddd);

const { email, password } = bdfbdddddddddddddddddddddddddddddddddddddd;

const vjkjdvkvk = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('users').get();

let vjkjdvkvkArr = [];

vjkjdvkvk.forEach(doc => {

vjkjdvkvkArr.push(doc.data());

});

const user = vjkjdvkvkArr.find(user => user.email === email);

// If found, compare the hashed passwords and generate the vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd token for the user

const bdfbddddddddddddddddddddddddddvsdvsdvdddddddddddd = crypto.createHash('md5').update(password).digest('hex');

if (user !== undefined) {

if (bdfbddddddddddddddddddddddddddvsdvsdvdddddddddddd !== user.password) {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: false, description: 'Wrong password' });

return;

} else {

let loginData = {

email,

signInTime: Date.now(),

}

console.log(loginData);

console.log("Creating token...");

try {

const token = vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.sign(loginData, vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddSecretKey);

console.log("Token created:", token);

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: true, token });

return;

} catch (error) {

console.error("Error creating token:", error);

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: false, description: 'Error creating token' + error});

return;

}

}

} else {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: false, description: 'No user found'});

return;

}

});

ipcMain.on(e[1], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

console.log(bdfbdddddddddddddddddddddddddddddddddddddd);

try {

const verified = vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.verify(bdfbdddddddddddddddddddddddddddddddddddddd.token, vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddSecretKey);

if (verified) {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: true });

} else {

// Access Denied

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: false });

}

} catch (error) {

// Access Denied

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ status: 'Invalid auth', success: false });

}

console.log(vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue);

return;

});

ipcMain.on(e[2], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

console.log(bdfbdddddddddddddddddddddddddddddddddddddd);

const vjkjdvkvk = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('users').get();

let vjkjdvkvkArr = [];

vjkjdvkvk.forEach(doc => {

vjkjdvkvkArr.push(doc.data());

});

const user = vjkjdvkvkArr.find(user => user.email === bdfbdddddddddddddddddddddddddddddddddddddd.email);

if (user !== undefined) {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: true, user: user });

} else {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: false, description: 'No user found' });

}

console.log(vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue);

return;

});

ipcMain.on(e[3], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

const { email, password } = bdfbdddddddddddddddddddddddddddddddddddddd;

const vjkjdvkvk = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('users').get();

let vjkjdvkvkArr = [];

vjkjdvkvk.forEach(doc => {

vjkjdvkvkArr.push(doc.data());

});

const user = vjkjdvkvkArr.find(user => user.email === email);

// If found, compare the hashed passwords and generate the vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd token for the user

const bdfbddddddddddddddddddddddddddvsdvsdvdddddddddddd = crypto.createHash('md5').update(password).digest('hex');

if (user === undefined) {

console.log({ email, password: bdfbddddddddddddddddddddddddddvsdvsdvdddddddddddd});

const uniqueId = uuidv4();

const vjkjdvkvk = bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('users').doc(uniqueId).set({id: uniqueId, email: email, password: bdfbddddddddddddddddddddddddddvsdvsdvdddddddddddd, role: 'default'});

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: true });

return;

// If no user is found, hash the given password and create a new entry in the login bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd with the email and hashed password

} else {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({ success: false, description: 'This email already exists'});

return;

}

});

ipcMain.on(e[4], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

try {

const vjkjdvkvk = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('posts').get();

let vjkjdvkvkArr = [];

vjkjdvkvk.forEach(doc => {

vjkjdvkvkArr.push(doc.data());

});

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({success: true, posts: vjkjdvkvkArr});

} catch (error) {

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify({success: false, description: `${error}`});

}

return;

});

ipcMain.on(e[5], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

let vjkjdvkvk = {};

try {

const verified = vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.verify(bdfbdddddddddddddddddddddddddddddddddddddd.token, vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddSecretKey);

if (verified) {

if (bdfbdddddddddddddddddddddddddddddddddddddd.email === undefined) {

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

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

const vjkjdvkvk = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('users').get();

let vjkjdvkvkArr = [];

vjkjdvkvk.forEach(doc => {

vjkjdvkvkArr.push(doc.data());

});

const user = vjkjdvkvkArr.find(user => user.email === bdfbdddddddddddddddddddddddddddddddddddddd.email);

if (user !== undefined) {

const post = bdfbdddddddddddddddddddddddddddddddddddddd;

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

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

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

vjkjdvkvk = await putPost(bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, user, post);

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

} else {

vjkjdvkvk = {success: false, description: 'No user found'};

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

} else {

// Access Denied

vjkjdvkvk = {success: false, description: 'Access denied'};

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

} catch (error) {

// Access Denied or Invalid Token

vjkjdvkvk = {status: 'Invalid auth', success: false};

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

});

ipcMain.on(e[6], async (vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd, bdfbdddddddddddddddddddddddddddddddddddddd) => {

let vjkjdvkvk = {};

try {

const verified = vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.verify(bdfbdddddddddddddddddddddddddddddddddddddd.token, vlbdflvldflbfdmbldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddSecretKey);

if (verified) {

if (bdfbdddddddddddddddddddddddddddddddddddddd.email === undefined) {

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

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

const uservjkjdvkvk = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('users').get();

let userArr = [];

uservjkjdvkvk.forEach(doc => {

userArr.push(doc.data());

});

const user = userArr.find(user => user.email === bdfbdddddddddddddddddddddddddddddddddddddd.email);

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

const postId = bdfbdddddddddddddddddddddddddddddddddddddd.postId;

if (postId !== undefined) {

vjkjdvkvk = {};

try {

const vjkjdvkvkbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd2 = await bdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.collection('posts').doc(postId).delete();

vjkjdvkvk = {success: true};

} catch (error) {

vjkjdvkvk = {success: false, description: error};

}

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

} else {

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

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

} else {

vjkjdvkvk = {success: false, description: 'Only admin can delete posts or no user found'};

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

} else {

// Access Denied

vjkjdvkvk = {success: false, description: 'Access denied'};

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

} catch (error) {

// Access Denied or Invalid Token

vjkjdvkvk = {status: 'Invalid auth', success: false};

vlbdflvldflbfdmsdaddasddasbdfbddddddddddddddddddddddddddvsdvsadassdvddddddddddddldflbdfbdfbdfbffbdfbddddddddddddddddddddddddddvsdvsadassdvdddddddddddd.returnValue = JSON.stringify(vjkjdvkvk);

return;

}

});

});

app.on(e[7], () => {

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

})