Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Кибернетики и информационной безопасности»

Отчет по курсовой работе

по предмету «Защита информации в глобальных сетях»

на тему:

«Подмена cookie файлов»

Выполнили: студенты группы

М092201(76)

Дворянинов П.В.

Куликова А.А.

Шауи Р.

Проверил:

Маторин Ф.А.

Москва 2023

Цель работы

В данной курсовой работе я планирую написать программное обеспечение с функцией личного кабинета пользователя, а для повышения безопасности процесса аутентификации реализовать и протестировать контроль доступа вызова API методов.

Введение

Для реализации программного обеспечения с функцией личного кабинета и аутентификацией следует использовать архитектуру клиент-сервер. Данная архитектура распределяет задания или сетевые нагрузки между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами.

Для построения сервера используется библиотека NodeJS Express. Express — это серверная платформа, которая используется NodeJS для создания веб-приложений. Для хранения данных о пользователе используется свободная объектно-реляционная система управления базами данных Postgres, которая связана с сервером при помощи API библиотеки «pg». В качестве клиенте используем утилиту Postman.

Для написания личного кабинета пользователя и его тестирования потребуется реализовать данные API методы:

* /api/registration/ – регистрация пользователя;
* /api/login/ – аутентификация пользователя с проверкой по токену;
* /api/user/ – вывод списка пользователя, только для авторизованных пользователей;
* /api/encrypt/ – тестирование механизма кодирования данных;
* /api/decrypt/ – тестирование механизма декодирования данных.

Далее приступим к реализации программного обеспечения.

Выполнение

Инициализируем запуск программного обеспечения, что показано на листинге 1, и запустим сервер базы данных Postgres.

Листинг 1 – Инициализация программного обеспечения

const express = require('express')

const authRouter = require('./routes/auth.routes.js')

const PORT = process.env.PORT || 8080

const app = express();

app.use(express.json())

app.use('/api', authRouter)

app.listen(PORT, ()=>console.log('init'))

Написали все необходимые маршруты API для работы и тестирования программного обеспечения, что представлено на листинге 2.

Листинг 1 – API методы

const express = require('express')

const router = express.Router()

const authController = require('../controller/auth.controller.js')

const {check} = require("express-validator")

const authMiddleware = require('../middleware/auth.middleware.js')

router.post('/auth/registration',[

check('username', "Имя пользователя обязательно").notEmpty(),

check('password', "Пароль должен быть от 4 до 10 символов").isLength({min:4, max:10})

], authController.registration)

router.post('/auth/login', authController.login)

router.get('/auth/user', authMiddleware("admin"), authController.getUserList)

router.post('/auth/encrypt', authController.encodeToken)

router.post('/auth/decrypt', authController.decryptToken)

module.exports = router

Реализовали процесс проверки токена пользователя, который отправляется вместе с заголовком запроса на сервер программного обеспечения. Токен декодируется и проверяется при помощи методов библиотеки «jsonwebtoken» или «crypto», в зависимости от выбранного типа разработки, что показано на листинге 3.

const jwt = require('jsonwebtoken')

const {secretDev, secretProd, mode} = require('../config')

const crypto = require('crypto-js')

module.exports = function (roleValue) {

return function (req, res, next) {

if (req.method === "OPTIONS") {

next()

}

try {

if (!req.headers.authorization) {

return res.status(403).json({message: "Authorization error"})

}

const token = req.headers.authorization.replace('Bearer ','')

if (!token) {

return res.status(403).json({message: "Пользователь не авторизован"})

}

if (mode==='dev') {

// Decrypt crypto

const bytes = crypto.AES.decrypt(token, secretDev);

const {role} = JSON.parse(bytes.toString(crypto.enc.Utf8));

if (role !== roleValue) {

return res.status(403).json({message: "У вас нет доступа"})

}

} else {

// Decrypt JWT

const {role} = jwt.verify(token, secretProd)

if (role !== roleValue) {

return res.status(403).json({message: "У вас нет доступа"})

}

}

next();

} catch (e) {

console.log(e)

return res.status(403).json({message: "Пользователь не авторизован"})

}

}

};

Далее, на листинге 4, представлены контроллеры, которые позволяют программе взаимодействовать с базой данных при помощи методов:

* /api/registration/ – регистрация пользователя;
* /api/login/ – аутентификация пользователя с проверкой по токену;
* /api/user/ – вывод списка пользователя, только для авторизованных пользователей;
* /api/encrypt/ – тестирование механизма кодирования данных;
* /api/decrypt/ – тестирование механизма декодирования данных.

const db = require('../db')

const bcrypt = require('bcryptjs')

const jwt = require('jsonwebtoken');

const { validationResult } = require('express-validator')

const {secretDev, secretProd, mode} = require("../config")

const crypto = require('crypto-js')

const generateAccessToken = (id, role) => {

if (mode === 'dev') {

const payload = {

id,

role,

secretDev,

}

// Encrypt crypto

const encryptedData = crypto.AES.encrypt(JSON.stringify(payload), secretDev).toString();

return encryptedData

} else {

const payload = {

id,

role

}

// Encrypt JWT

return jwt.sign(payload, secretProd, {expiresIn: "24h"} )

}

}

class authController {

async registration(req, res) {

try {

const errors = validationResult(req)

if (!errors.isEmpty()) {

return res.status(400).json({message: 'Registration error', errors})

}

const {username, password} = req.body

const user = await db.query('SELECT \* from userList WHERE username = $1', [username])

if (user.rows.length) {

return res.status(400).json({message: 'User existed error'});

}

const hashPassword = bcrypt.hashSync(password, 5)

const role = await db.query('SELECT \* from roleList WHERE value = $1', ['user'])

if (role.rows.length === 0) {

return res.status(400).json({message: 'Role not existed error'});

}

const newUser = await db.query('INSERT INTO userList (username, password, role\_id) VALUES ($1, $2, $3) RETURNING \*', [username, hashPassword, role.rows[0].id])

res.status(200).json({message: 'User added', user: newUser.rows[0]});

} catch (e) {

console.log(e);

res.status(400).json({message: 'Registration error'});

}

}

async login(req, res) {

try {

const {username, password} = req.body

const user = await db.query('SELECT \* from userList WHERE username = $1', [username])

if (!user.rows.length) {

return res.status(400).json({message: 'User not existed error'});

}

const validPassword = bcrypt.compareSync(password, user.rows[0].password);

if (!validPassword) {

return res.status(400).json({message: 'Not valid password error'});

}

const role = await db.query('SELECT \* from roleList WHERE id = $1', [user.rows[0].role\_id])

if (role.rows.length === 0) {

return res.status(400).json({message: 'Role not existed error'});

}

const token = generateAccessToken(user.rows[0].id, role.rows[0].value)

return res.json({token})

} catch (e) {

console.log(e)

res.status(400).json({message: 'Login error'})

}

}

async getUserList(\_, res) {

try {

const userList = await db.query('SELECT \* from userList')

res.status(200).json(userList.rows)

} catch (e) {

console.log(e)

}

}

async encodeToken(req, res) {

const payload = req.body

// Encrypt crypto

const encryptedData = crypto.AES.encrypt(JSON.stringify(payload), payload.secret).toString();

return res.json(encryptedData)

}

async decryptToken(req, res) {

const {token, secret} = req.body

const bytes = crypto.AES.decrypt(token, secret);

const payload = JSON.parse(bytes.toString(crypto.enc.Utf8));

return res.json(payload)

}

}

module.exports = new authController()

Запустим программу с ранее записанными пользователями и их ролями. Для тестирования программного обеспечения обозначим две роли: админ и пользователь, если для записи указана роль админ, то пользователь имеет доступ к методу /api/user/ – получение списка всех пользователей системы, иначе выводим сообщение об ошибке.

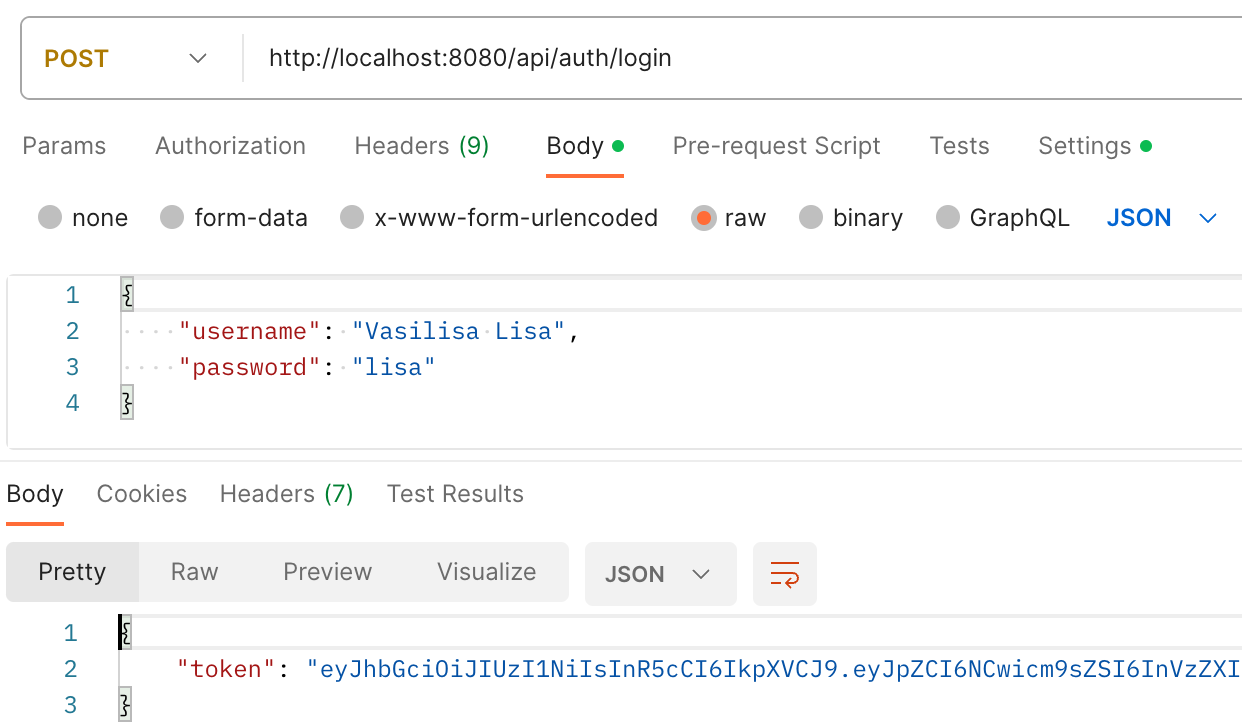


Рис. 1 – Аутентификация пользователя

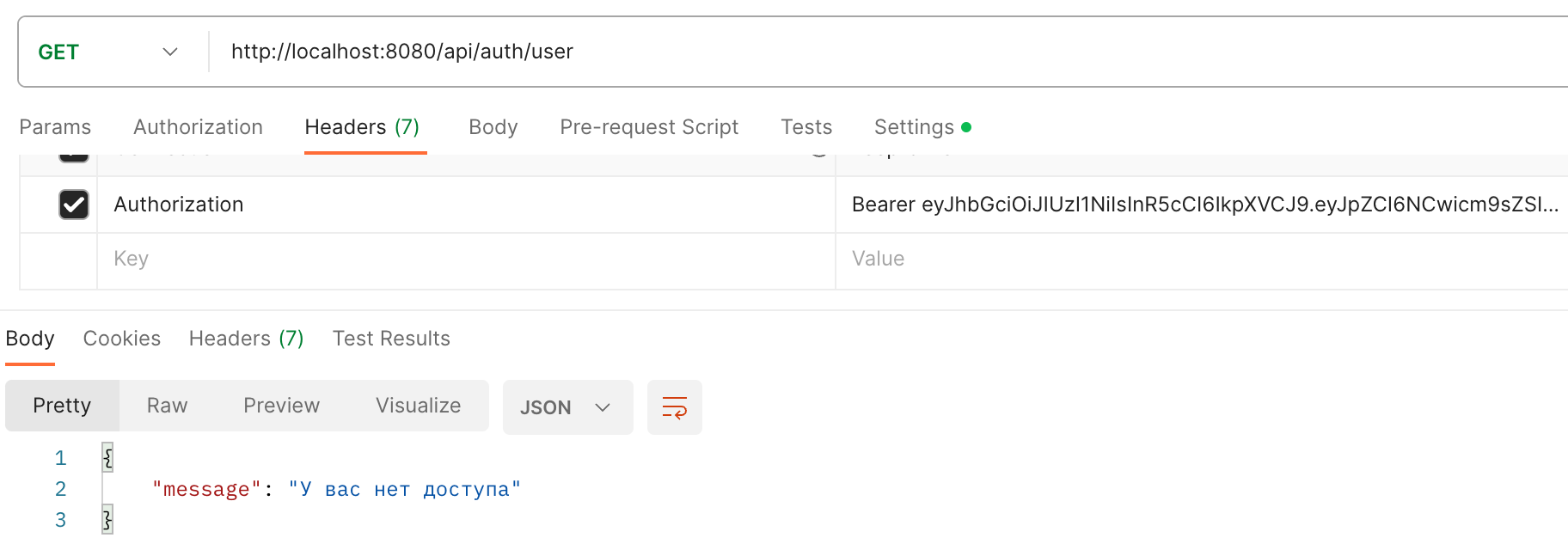


Рис. 2 – Если токен не актуален, выводим сообщение об ошибке

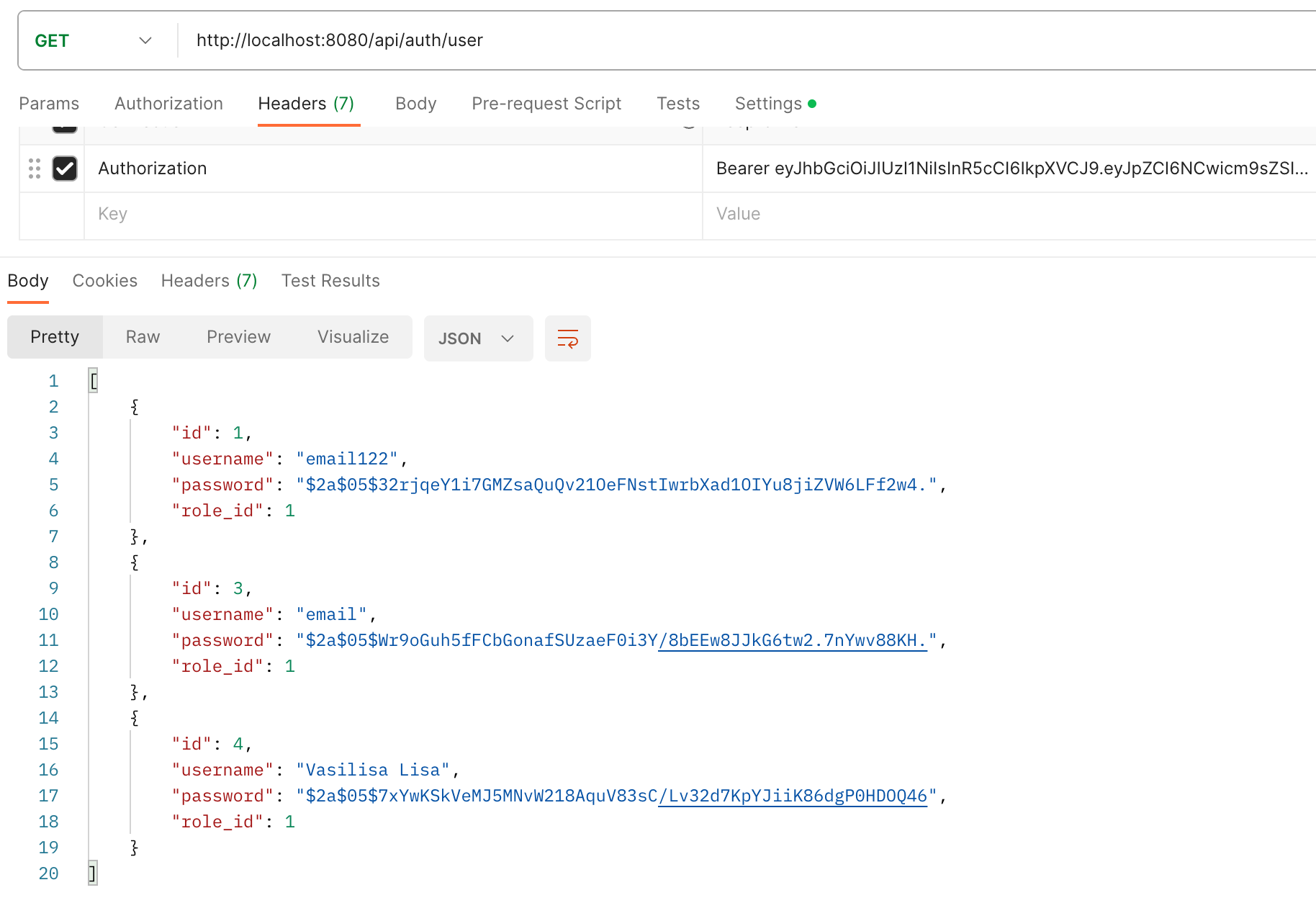


Рис. 3 – Если токен актуален, выводим список пользователей

Токен генерируется за счет кодируемых данных и секретного ключа. Для программного комплекса реализована два типа токена.

Первый тип токена сгенерирован при помощи библиотеки «crypto». Данный токен не обладает повышенной безопасностью, если секретный ключ будет недостаточно надежным и попадет в руки злоумышленнику, то программный комплекс никак не предотвратит и не уведомит настоящего владельца аккаунта о взломе личного кабинета!

Для повышения безопасности аутентификации следует использовать библиотеку «jsonwebtoken». Данная библиотека позволяет обеспечить не только кодирование данных, но и безопасность за счет возможности генерации access и refresh токена и параметра expire, который отвечает за период актуальности сгенерированного токена! Если аккаунт подвергнется взлому, то через определенное время система потребует обновить токен за счет проведения новой сессии аутентификации или отправки refresh токена, который обновит access и refresh токен, за счет этого access токен злоумышленника станет не актуальным и он будет отвергнут системой, но стоит учитывать то, что данная схема будет безопасна только при повторном обновлении токенов настоящим владельцем аккаунта!

Таким образом, схема refresh и access токен ограничивает время, на которое атакующий может получить доступ к сервису. По сравнению с одним токеном, которым злоумышленник может пользоваться неделями и никто об этом не узнает.

Вывод

При выполнении данной курсовой работе написал программное обеспечение с функцией личного кабинета пользователя. Для повышения безопасности процесса аутентификации реализовал и протестировал контроль доступа вызова API методов.