**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи №1 з дисципліни

«Системи безпеки програм і даних»

„Основні методи авторизації ”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Тарасьонок Дмитро Євгенович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Іваніщев Б. В.*

Київ 2024

ЗМІСТ

[1 Мета лабораторної роботи 3](#_Toc160362441)

[2 Завдання 4](#_Toc160362442)

[2.1 Основне завдання 4](#_Toc160362443)

[2.2 Додаткове завдання 4](#_Toc160362444)

[3 Виконання основного завдання 5](#_Toc160362445)

[3.1 Огляд basic\_auth 5](#_Toc160362446)

[3.2 Огляд forms\_auth 5](#_Toc160362447)

[3.3 Огляд token\_auth 9](#_Toc160362448)

[4 Виконання додаткового завдання 11](#_Toc160362449)

[5 Висновок 18](#_Toc160362450)

# Мета лабораторної роботи

Мета роботи – роздивитися основні методи авторизації

# Завдання

## Основне завдання

Викачати репозиторій з лекціями https://github.com/Kreolwolf1/auth\_examples

Запустити кожен з 3 аплікейшенів та зробити скріншити запитів до серверу.

## Додаткове завдання

Модифікувати token\_auth аплікейшен змінивши токен на JWT.

# Виконання основного завдання

## Огляд basic\_auth

Для початку встановлюємо фреймворк express, а потім запускаємо сервер.

A black screen with white text

Description automatically generated

Сервер слухає на 3000 порті, спробуємо звернутися до нього з логіном DateArt та паролем 2408.



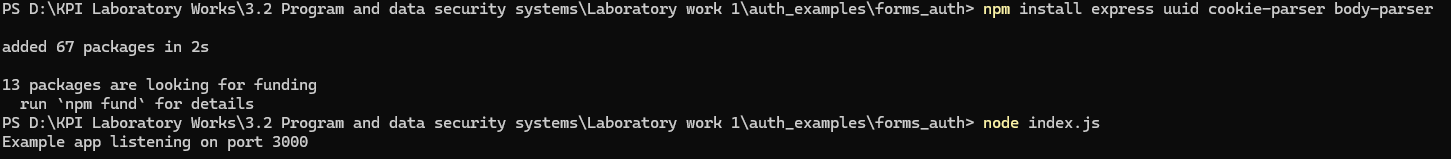
Бачимо успіх. Спробуємо звернутися з іншим паролем.



Змінивши пароль, сервер уже не вітає нас.

## Огляд forms\_auth

Встановлюємо залежності та запускаємо сервер



Заходимо в браузері за адресою <http://localhost:3000>

A screenshot of a login screen

Description automatically generated

Бачимо вікно авторизації. Вводимо логін Login та пароль Password (узяті з коду серверу) та логінимося.

A black screen with white text

Description automatically generated

Усе вдалося. Розглянемо запити детальніше через DevTools у Chrome.

A screenshot of a computer program

Description automatically generated

Бачимо, що це був POST запит за адресою localhost:3000/api/login. Подивимося payload.

A screenshot of a computer program

Description automatically generated

Бачимо що авторизаційні дані передаються просто в сирому JSON вигляді. Оглянемо файл sessions.json.

A computer screen with red and green text

Description automatically generated

Бачимо, що тут з’явилася саме та сесія, яку нам повернув сервер. Спробуємо розлогінитися.

A screenshot of a computer program

Description automatically generated

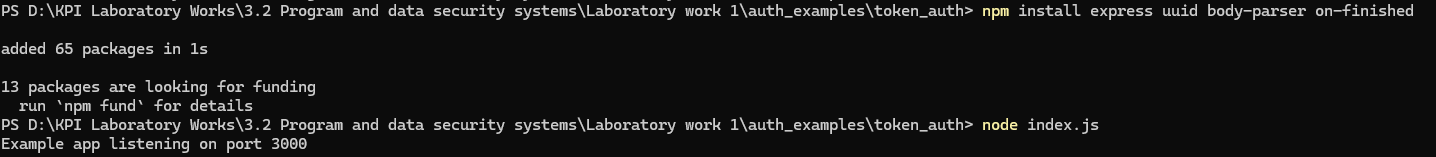
Бачимо, що сервер повернув заголовок Set-Cookie, у якому видаляється наша сесія. Цікаво те, що з файлу session.json не просто не зникла наша сесія, а ще й додалися дві нові. Скоріше за все, це баг самого сервера.

A screenshot of a computer program

Description automatically generated

## Огляд token\_auth

Встановлюємо всі залежності та запускаємо сервер



Для даного методу авторизації будемо працювати через curl, оскільки браузер не хоче коректно відображати відповіді від сервера.

Робимо запит на авторизацію, знову передаючи логін та пароль у вигляді JSON.



Подивимося, що відбулося з файлом sessions.json.

A screenshot of a computer program

Description automatically generated

Знову створилася велика кількість сесій, у тому числі й наша. Спробуємо тепер розлогінитися. Вказуємо токен уже не як кукі, а в окремому хедері Authorization.



Цікаво те, що сесія знову не видалилася з файлу sessions.json, хоча її було передано вірно. Сервер локально видаляє сесію зі свого асоціативного масиву, але ці зміни не записуються у файл, через що й відбувається така ситуація. Однак, виправлення даної помилки не входить у задвдання даної лабораторної роботи.

# Виконання додаткового завдання

Мною було змінено файл index.html:

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Login</title>  
 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  
</head>  
<body>  
<main id="main-holder">  
 <h1 id="login-header">Login</h1>  
  
 <div id="login-error-msg-holder" style="display: none;">  
 <p id="login-error-msg">Invalid username and/or password</p>  
 </div>  
  
 <form id="login-form">  
 <input type="text" name="login" id="username-field" class="login-form-field" placeholder="Username">  
 <input type="password" name="password" id="password-field" class="login-form-field" placeholder="Password">  
 <input type="submit" value="Login" id="login-form-submit">  
 </form>  
  
 <a href="#" id="logout" style="display: none;">Logout</a>  
</main>  
  
<script>  
 document.getElementById("login-form").addEventListener("submit", function (e) {  
 e.preventDefault();  
 const login = document.getElementById("username-field").value;  
 const password = document.getElementById("password-field").value;  
  
 axios.post('/api/login', {login, password})  
 .then(response => {  
 sessionStorage.setItem('authToken', response.data.token);  
 alert('You are now logged in!');  
 document.getElementById("logout").style.display = "block";  
 document.getElementById("login-form").style.display = "none";  
 })  
 .catch(error => {  
 document.getElementById("login-error-msg-holder").style.display = "block";  
 });  
 });  
  
 document.getElementById("logout").addEventListener("click", function (e) {  
 e.preventDefault();  
 sessionStorage.removeItem('authToken');  
 alert('You are now logged out!');  
 document.getElementById("login-form").style.display = "block";  
 this.style.display = "none";  
 });  
  
 if (sessionStorage.getItem('authToken')) {  
 document.getElementById("login-form").style.display = "none";  
 document.getElementById("logout").style.display = "block";  
 }  
  
 document.addEventListener("DOMContentLoaded", function () {  
 const token = sessionStorage.getItem('authToken');  
 if (token) {  
 axios.get('/', {  
 headers: {  
 'Authorization': `Bearer ${token}`  
 }  
 }).then(response => {  
 const {username} = response.data;  
  
 if (username) {  
 const mainHolder = document.getElementById("main-holder");  
 const loginHeader = document.getElementById("login-header");  
  
 mainHolder.append(`Hello ${username}`);  
  
 loginForm.remove();  
 loginErrorMsg.remove();  
 loginHeader.remove();  
 logoutLink.style.opacity = 1;  
 }  
 }).catch(error => {  
 });  
 }  
 });  
</script>  
</body>  
  
<style>  
 html {  
 height: 100%;  
 }  
  
 body {  
 height: 100%;  
 margin: 0;  
 font-family: Arial, Helvetica, sans-serif;  
 display: grid;  
 justify-items: center;  
 align-items: center;  
 background-color: #3a3a3a;  
 }  
  
 #main-holder {  
 width: 50%;  
 height: 70%;  
 display: grid;  
 justify-items: center;  
 align-items: center;  
 background-color: white;  
 border-radius: 7px;  
 box-shadow: 0px 0px 5px 2px black;  
 }  
  
 #login-error-msg-holder {  
 width: 100%;  
 height: 100%;  
 display: grid;  
 justify-items: center;  
 align-items: center;  
 }  
  
 #login-error-msg {  
 width: 23%;  
 text-align: center;  
 margin: 0;  
 padding: 5px;  
 font-size: 12px;  
 font-weight: bold;  
 color: #8a0000;  
 border: 1px solid #8a0000;  
 background-color: #e58f8f;  
 opacity: 0;  
 }  
  
 #error-msg-second-line {  
 display: block;  
 }  
  
 #login-form {  
 align-self: flex-start;  
 display: grid;  
 justify-items: center;  
 align-items: center;  
 }  
  
 .login-form-field::placeholder {  
 color: #3a3a3a;  
 }  
  
 .login-form-field {  
 border: none;  
 border-bottom: 1px solid #3a3a3a;  
 margin-bottom: 10px;  
 border-radius: 3px;  
 outline: none;  
 padding: 0px 0px 5px 5px;  
 }  
  
 #login-form-submit {  
 width: 100%;  
 padding: 7px;  
 border: none;  
 border-radius: 5px;  
 color: white;  
 font-weight: bold;  
 background-color: #3a3a3a;  
 cursor: pointer;  
 outline: none;  
 }  
</style>  
</html>

Та файл index.js:

const express = require('express');  
const bodyParser = require('body-parser');  
const jwt = require('jsonwebtoken');  
const path = require('path');  
const port = 3000;  
  
const app = express();  
app.use(bodyParser.json());  
app.use(bodyParser.urlencoded({ extended: true }));  
  
const JWT\_SECRET\_KEY = 'IP11\_Tarasonok';  
  
const users = [  
 {  
 login: 'Login',  
 password: 'Password',  
 username: 'Username',  
 },  
 {  
 login: 'Login1',  
 password: 'Password1',  
 username: 'Username1',  
 }  
];  
  
app.post('/api/login', (req, res) => {  
 const { login, password } = req.body;  
 const user = users.find(user => user.login === login && user.password === password);  
  
 if (user) {  
 const token = jwt.sign({ username: user.username, login: user.login }, JWT\_SECRET\_KEY, { expiresIn: '1h' });  
 res.json({ token });  
 } else {  
 res.status(401).send('Unauthorized');  
 }  
});  
  
const authenticateToken = (req, res, next) => {  
 const authHeader = req.headers['authorization'];  
 const token = authHeader && authHeader.split(' ')[1];  
  
 if (token == null) {  
 if (req.path === '/' || req.path === '/api/login') return next();  
 else return res.sendStatus(401);  
 }  
  
 jwt.verify(token, JWT\_SECRET\_KEY, (err, user) => {  
 if (err) return res.sendStatus(403);  
 req.user = user;  
 next();  
 });  
};  
  
app.get('/', authenticateToken, (req, res) => {  
 console.log(req.user);  
 if (req.user) {  
 return res.json({  
 username: req.user.username,  
 logout: 'http://localhost:3000/logout'  
 })  
 }  
 res.sendFile(path.join(\_\_dirname+'/index.html'));  
});  
  
app.listen(port, () => {  
 console.log(`Server listening on port ${port}`);  
});

Продемонструємо роботу зміненого способу авторизації на практиці. При переході на сторінку localhost:3000 відкривається вже знайома нам веб-сторінка з формою авторизації. Логінимося з тими ж даними.

A screenshot of a computer

Description automatically generated

В усьому запит такий же, тільки бачимо, що змінився формат токену. Перезавантажимо головну сторінку та побачимо, що буде.

A screenshot of a computer

Description automatically generated

По-перше бачимо, що коректно відобразився юзернейм. Також можна побачити, як саме передається токен: хедер Authorization, а в ньому значення Bearer <наш токен>.

# Висновок

У ході даної лабораторної роботи ми ознайомилися з базовими способами авторизації. У першому випадку ми напряму передавали авторизаційні дані в запитах, у другому випадку – ми отримували айді сесії, передаючи дані у форматі JSON. Айді сесії при цьому зберігався в кукі. У третьому випадку спосіб отримання токену залишився таким же, але зберігатися вже він почав у localStorage. Далі ми змінили третій спосіб авторизації на JWT. Покращили програму тим, що тепер немає необхідності зберігати сесії локально у файлі, як це було в другому та третьому випадках. Тепер токени зберігаються тільки на стороні користувача й валідуються на стороні сервера за допомогою секретного ключа.