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
«Безпека програмного забезпечення»

“**Основні методи авторизації**”

Виконав(ла) *ІП-13 Бабіч Денис* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Соколовський В. В.* _____
(шифр, прізвище, ім'я, по батькові)

Київ 2024

ЛАБОРАТОРНА РОБОТА № 1

Тема роботи: Основні методи авторизації.

Мета роботи: Ознайомитися з основними методами авторизації.

Основне завдання: викачати репозиторій з лекціями, запустити кожен з 3 аплікейшенів та зробити скріншити запитів до серверу.

1.1 Аналіз basic_auth:

```
"dependencies": {
  "express": "^4.21.0"
}
```

Рисунок 1.1 – Встановлені модулі Node.JS

```
PS D:\Projects\KPI\Software-Security\labs\1\src\1.1\basic_auth> node index.js
Example app listening on port 3000
```

Рисунок 1.2 – Запуск локально серверу на порті 3000

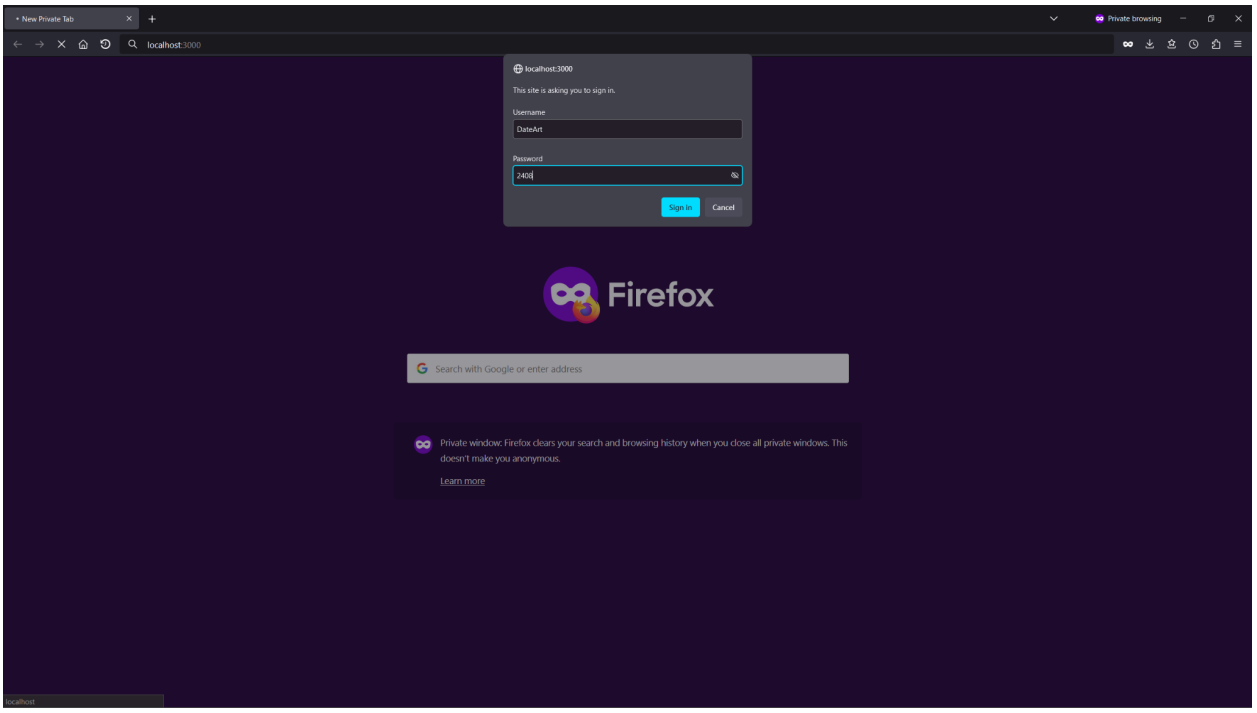

Рисунок 1.3 – Проведення авторизації

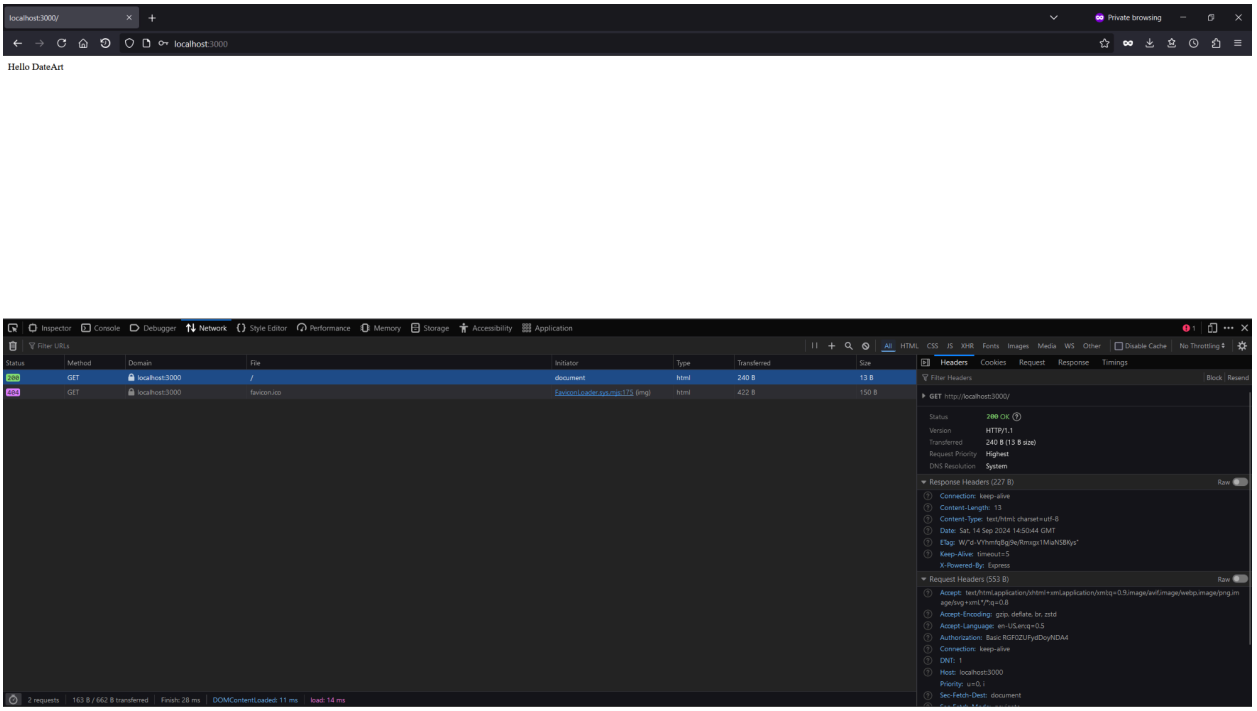

Рисунок 1.4 – Отримана успішна відповідь від сервера 200

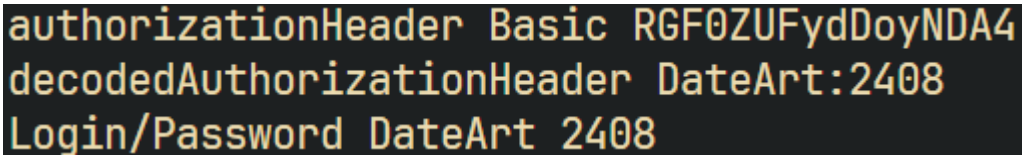

Рисунок 1.5 – Виведений лог серверу, що підтверджує успішну авторизацію

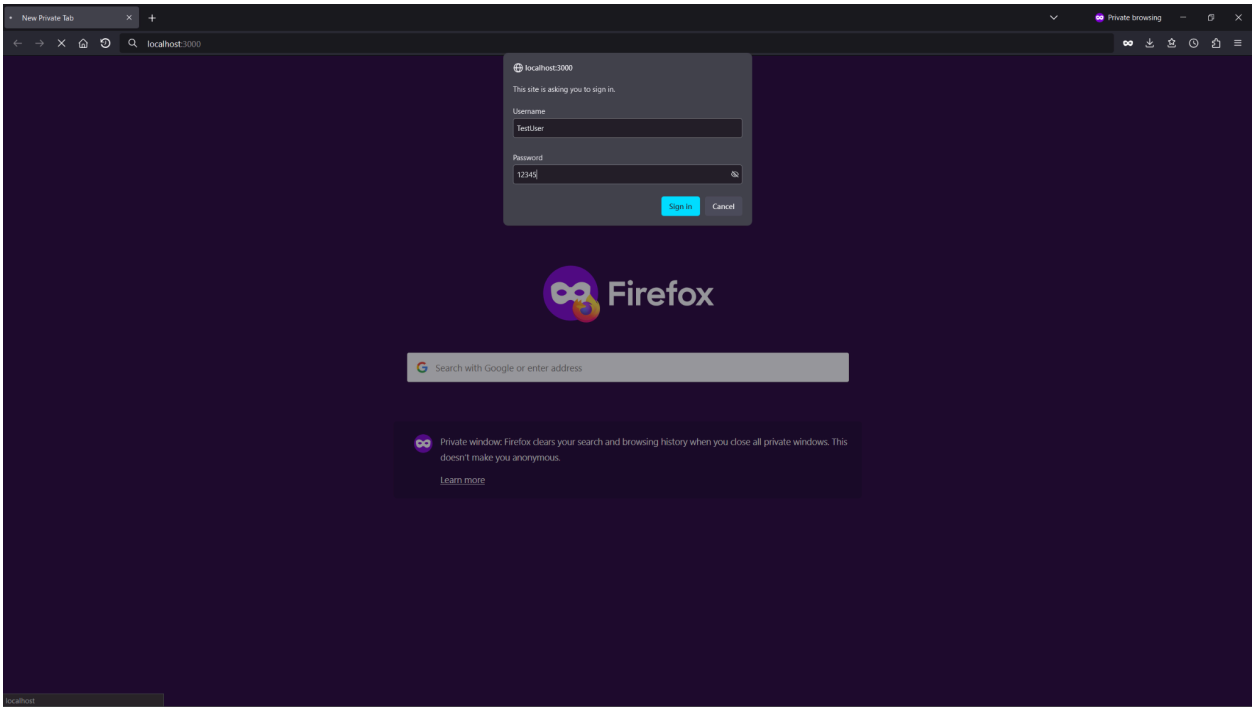

Рисунок 1.6 – Введення некоректних даних

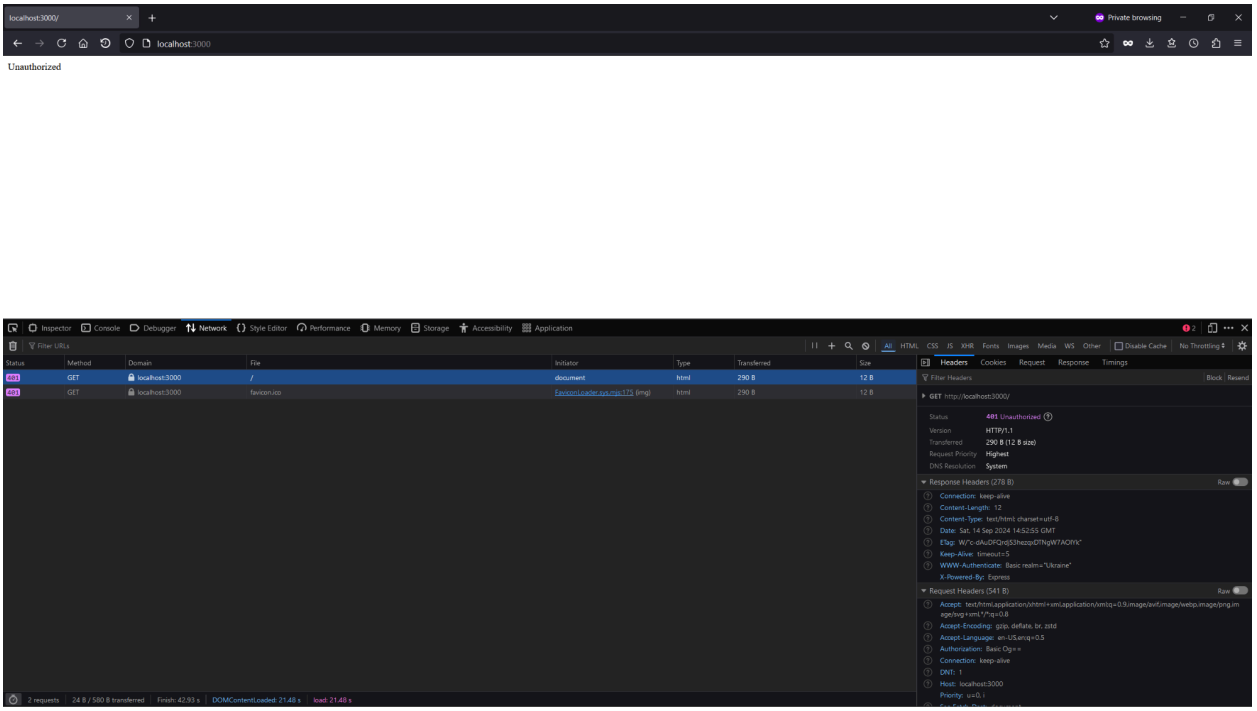

Рисунок 1.7 – Отримання відповіді 401 Unauthorized

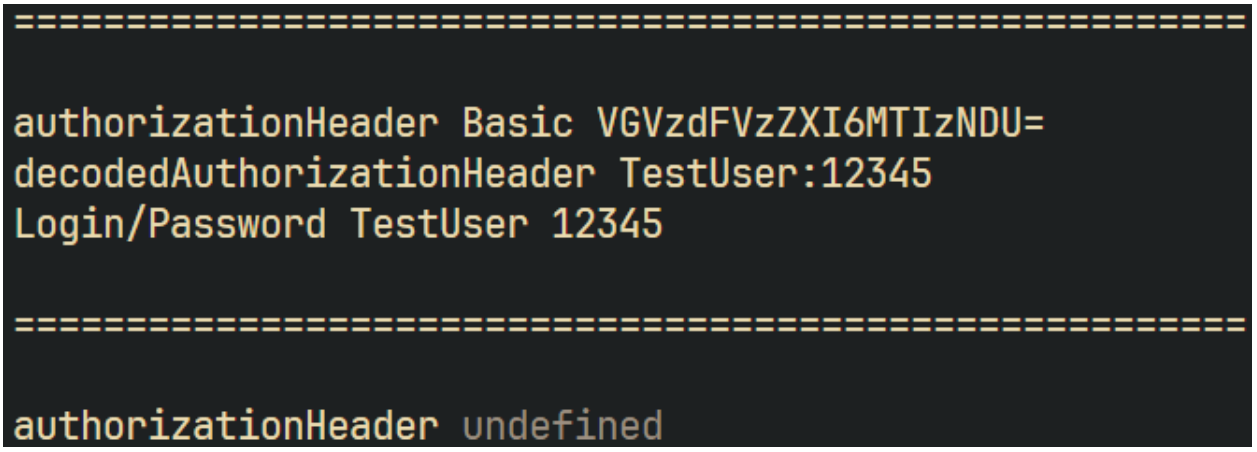

Рисунок 1.8 – Отриманий лог від серверу, який підтверджує неуспішність авторизації

Використаний підхід до авторизації є небезпечним через передачу паролів у відкритому вигляді, оскільки дані авторизації передаються у вигляді рядка, закодованого в Base64, що не є шифруванням. Це легко розшифрувати, і можливо перехопити ці дані через незахищене з'єднання (HTTP).

1.2 forms_auth:

```
PS D:\Projects\KPI\Software-Security\labs\1\src\1.1\forms_auth> npm install express uuid cookie-parser body-parser

added 68 packages in 8s

14 packages are looking for funding
  run `npm fund` for details
```

Рисунок 1.9 – Встановлення необхідних модулів

```
"dependencies": {
  "body-parser": "^1.20.3",
  "cookie-parser": "^1.4.6",
  "express": "^4.21.0",
  "uuid": "^10.0.0"
},
```

Рисунок 1.10 – Перегляд встановлених модулів

```
PS D:\Projects\KPI\Software-Security\labs\1\src\1.1\forms_auth> node index.js
Example app listening on port 3000
```

Рисунок 1.11 – Запуск серверу на localhost:3000

```
login: 'Login',
password: 'Password',
username: 'Username',
```

Рисунок 1.12 – Дані, які необхідні для авторизації

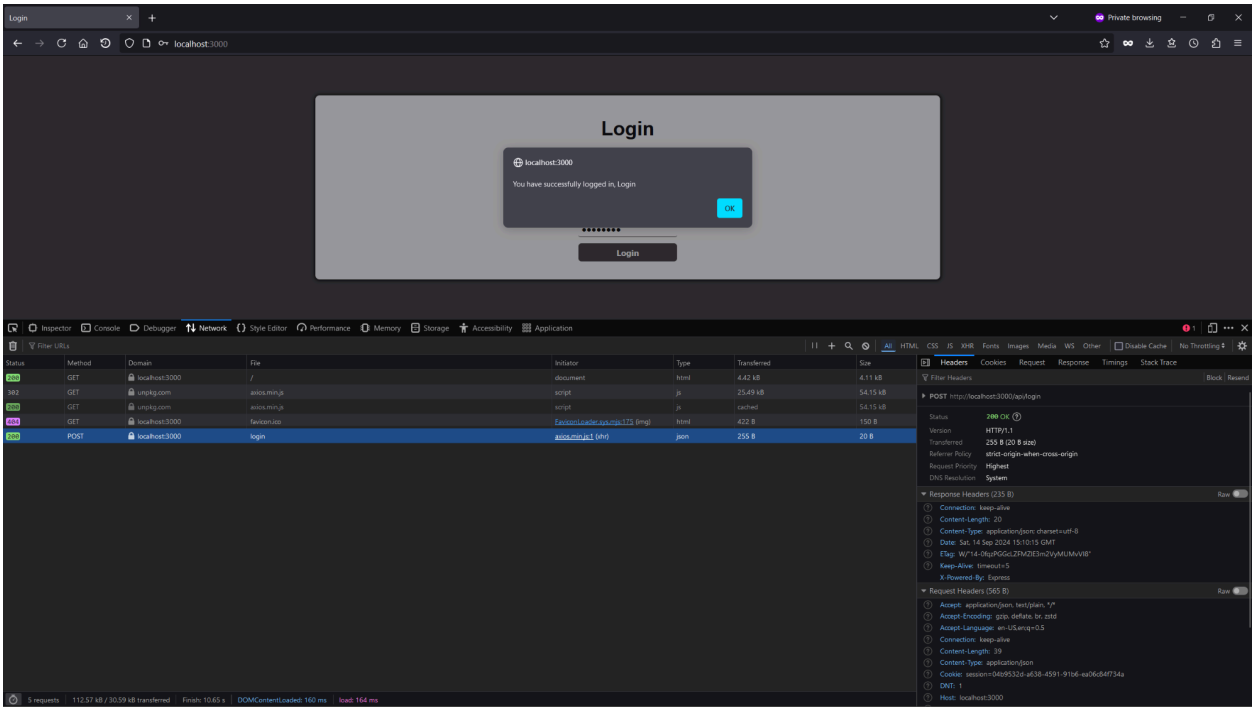

Рисунок 1.13 – Успішна авторизація, отриманий код – 200

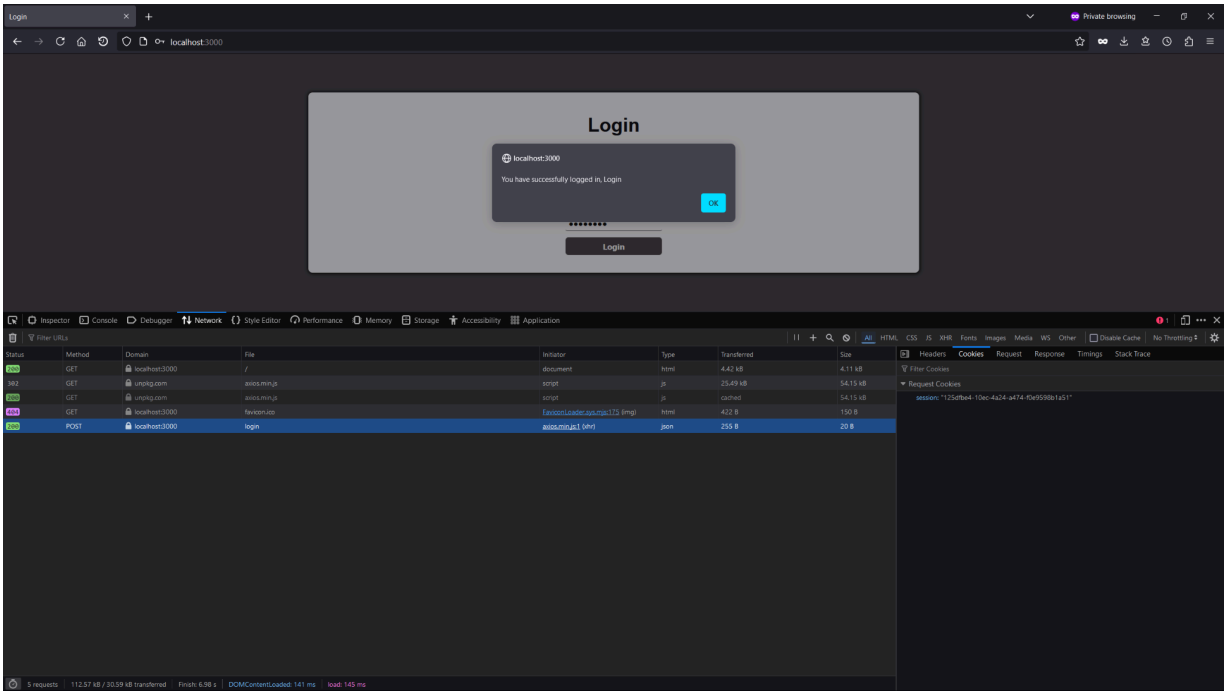

Рисунок 1.14 – Айді сесії зберігається як кукі, що небезпечно через можливі використання кукі-стілерів

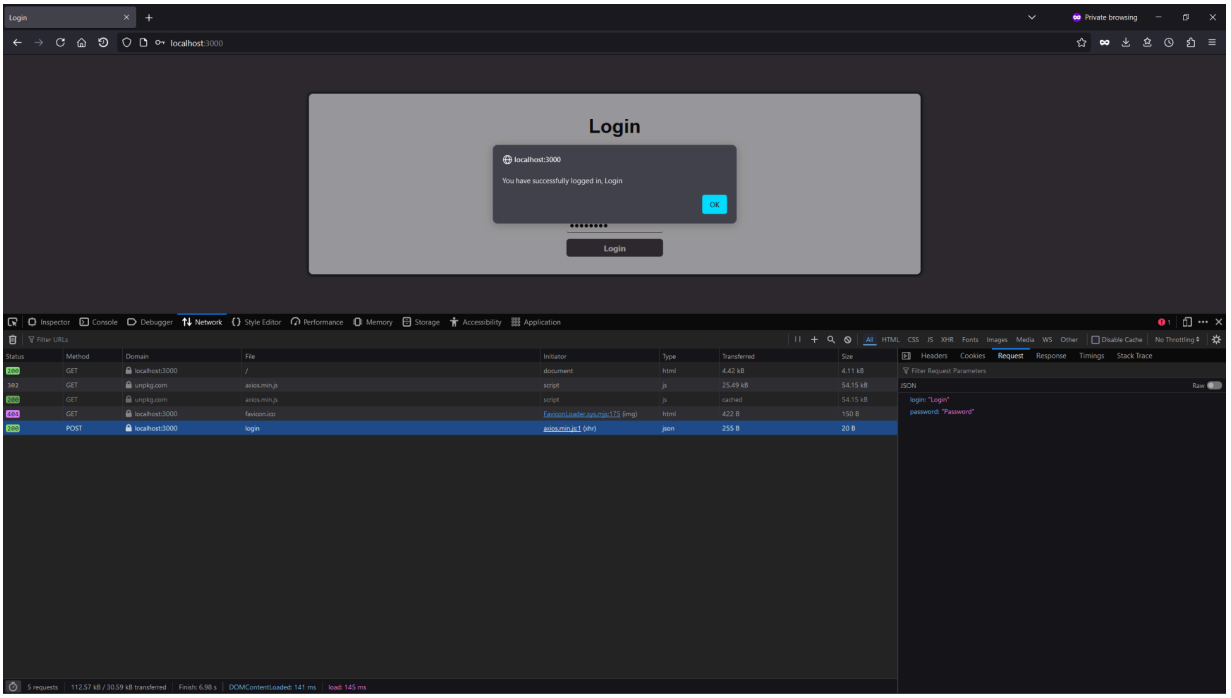

Рисунок 1.15 – Аутентифікація відбувається шляхом пересилання серіалізованго json, який зберігається у sessions.json

Рисунок 1.16 – Приклад вмісту файлу sessions.json, який зберігає створені сесії

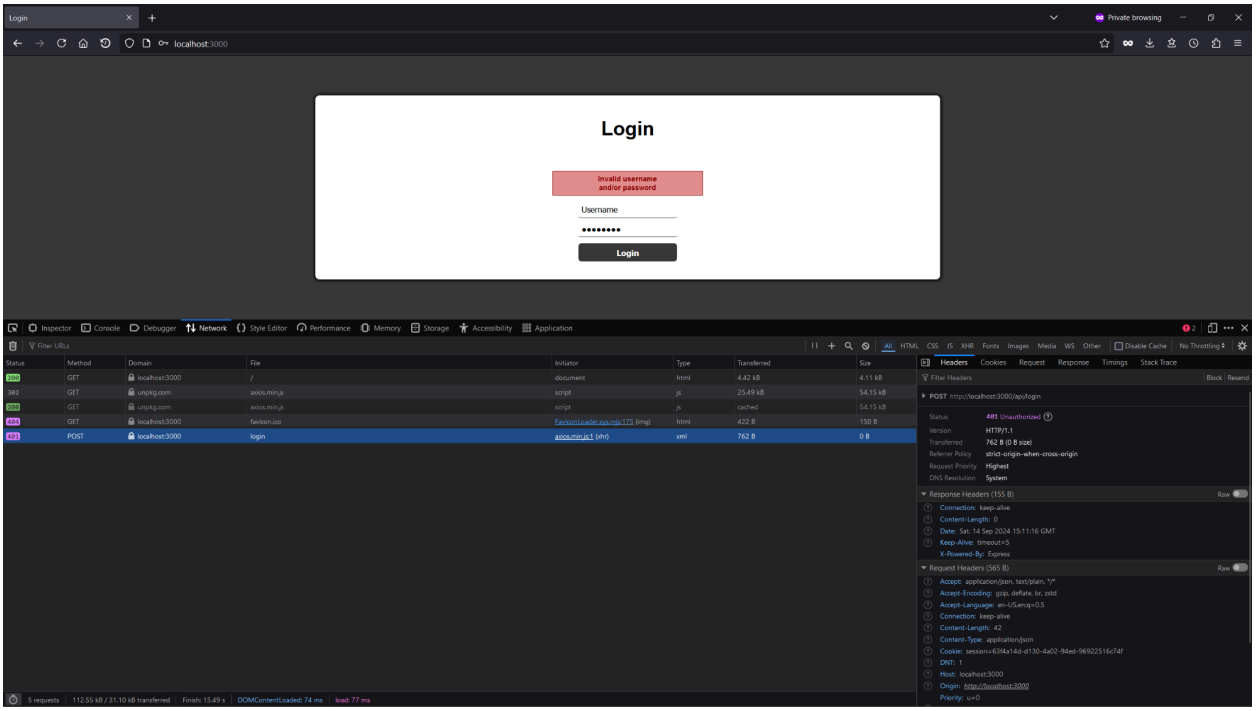

Рисунок 1.17 – Спроба неуспішної авторизації, повернутий код – 401

Варто згадати про загальну дивну поведінку застосунку, оскільки під час спроб авторизації та успішного їх проведення створюється відразу декілька сесій у sessions.json. Закономірним є також те, що значення сесій не видаляються після деавторизації.

Недоліками використаного підходу є збереження інформації про сесії у sessions.json, який може бути видалений, пошкоджений і через це користувачі втратять доступу до сервісу. Також відсутнє шифрування сесійних даних. Окремо варто згадати про вразливість до session hijacking, оскільки якщо веб-ресурс працює через без HTTPS, то кукі можуть бути перехоплені за допомогою атак типу "людина посередині" (MITM). Окрім того, код не має механізмів захисту від session fixation (атак, коли зловмисник встановлює свій власний ідентифікатор сесії для користувача).

1.3 token_auth:

```
PS D:\Projects\KPI\Software-Security\labs\1\src\1.1\token_auth> npm install express uuid cookie-parser body-parser

added 68 packages in 3s

14 packages are looking for funding
  run `npm fund` for details
```

Рисунок 1.18 – Встановлення необхідних модулів

```
PS D:\Projects\KPI\Software-Security\labs\1\src\1.1\token_auth> node index.js
Example app listening on port 3000
```

Рисунок 1.19 – Запуск серверу локально на localhost:3000

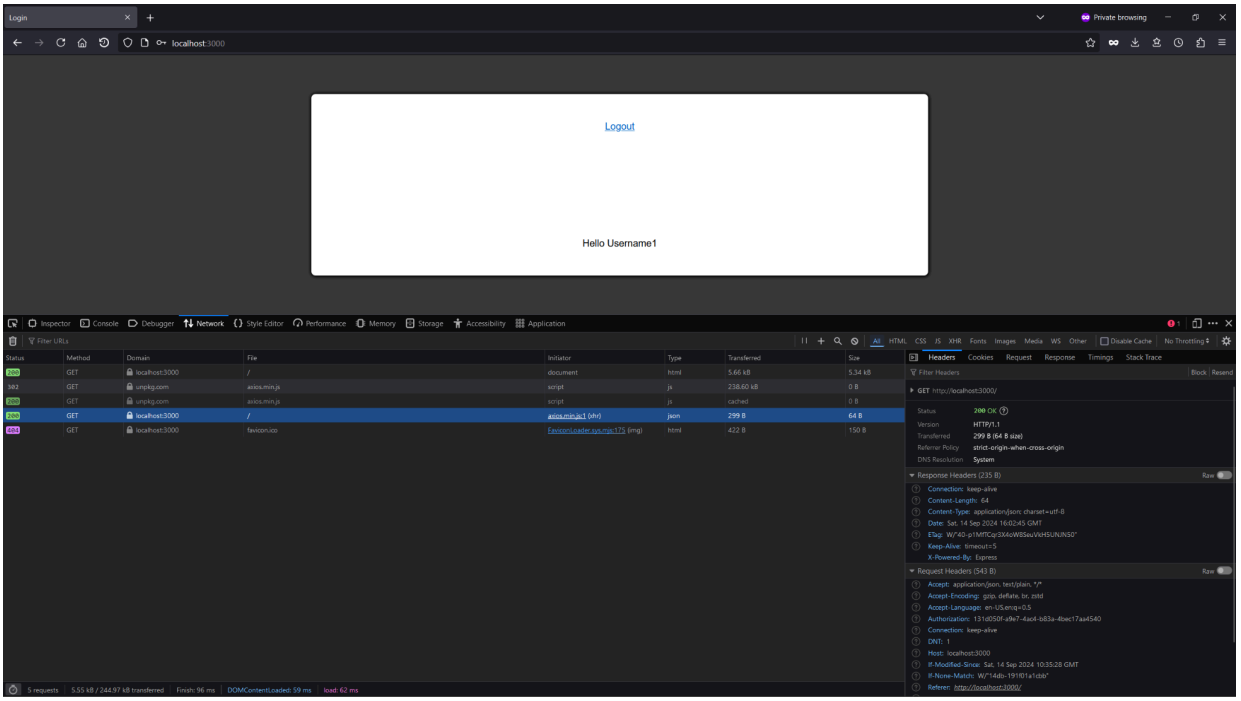

Рисунок 1.20 – Успішна авторизація, повернення коду 200

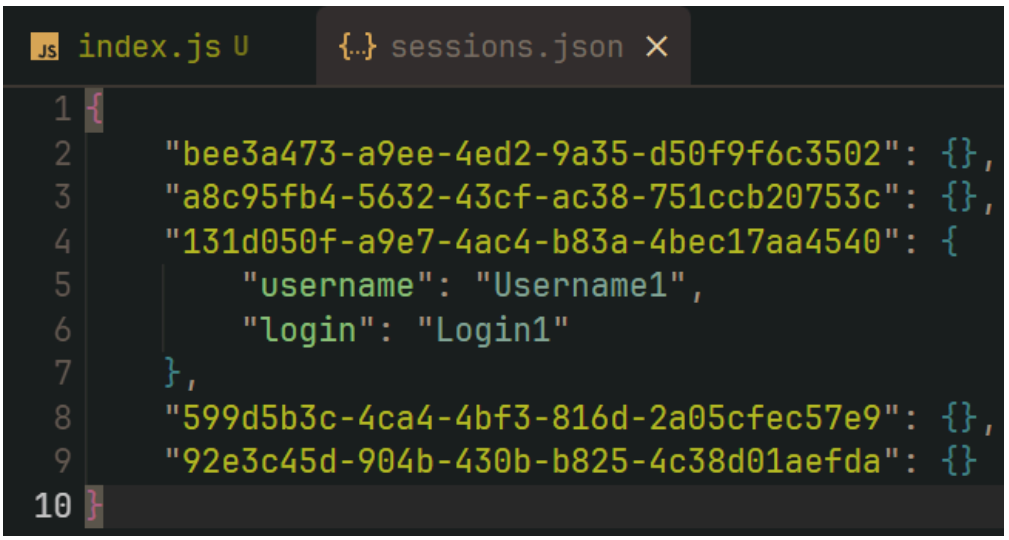

Рисунок 1.21 – Демонстрація наявної проблеми зі створенням фантомних сесій

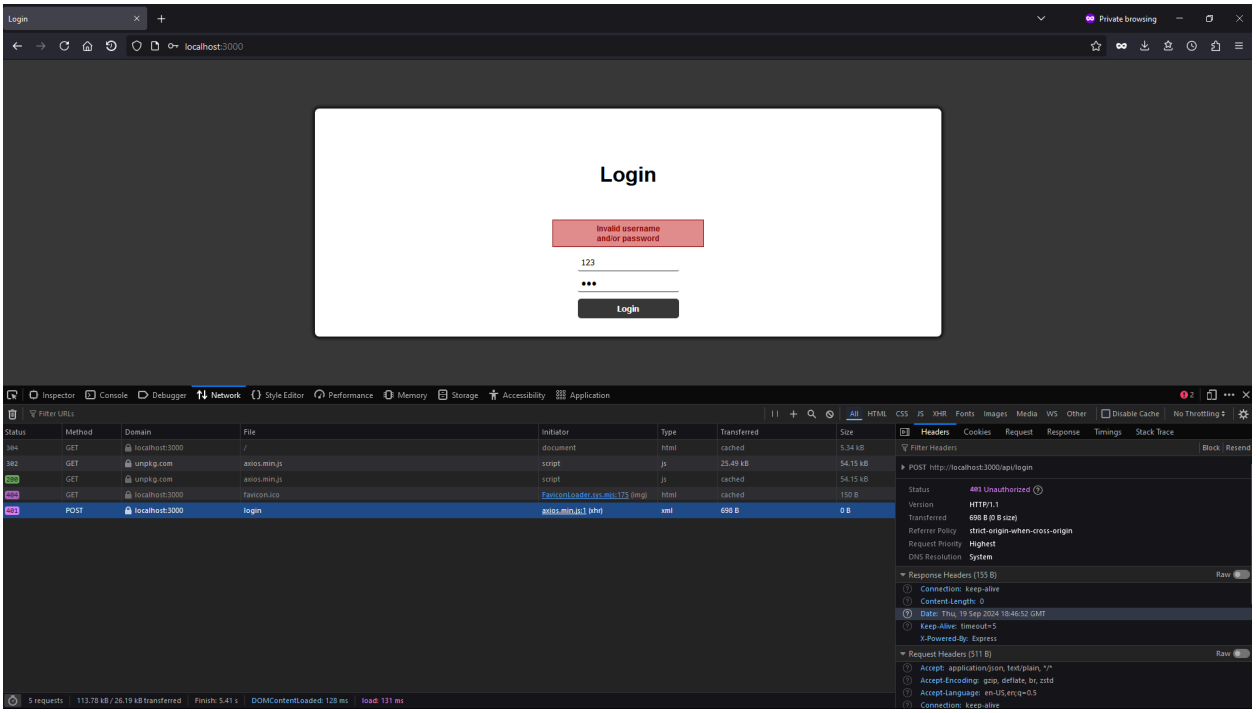

Рисунок 1.22 – Приклад невдалої авторизації, повренення коду 401

Загалом, описані технічні проблеми актуальні й для цього прикладу: незрозумілі створені фантомні сесії, збереження актуальних сесій на сервері у локальному файлі sessions.json, відсутність тайм-аутів для вже авторизованих сесій, відсутність захисту від CSRF (Cross-Site Request Forgery), оскільки будь-який інший сайт може використовувати актуальну сесію користувача у власних цілях, відсутність шифрування токена айді сесії, відсутність захисту від Session Fixation, коли зловмисник може присвоїти сесію користувачу, а потім використати після успішної авторизації користувача.

Додаткове завдання: модифікувати token_auth аплікейшен змінивши токен на JWT.

JWT (JSON Web Token) – це компактний, самодостатній токен, який використовується для передачі інформації між двома сторонами. Він складається з трьох частин: заголовка (header), корисного навантаження (payload) і підпису (signature). JWT часто використовується для автентифікації, коли сервер видає токен після успішного входу користувача, а клієнт зберігає його і надсилає з кожним запитом, щоб підтвердити свою ідентичність.

```
"dependencies": {
  "body-parser": "^1.20.3",
  "cookie-parser": "^1.4.6",
  "express": "^4.21.0",
  "jsonwebtoken": "^9.0.2",
  "uuid": "^10.0.0"
}
```

Рисунок 1.23 – Список використаних модулів

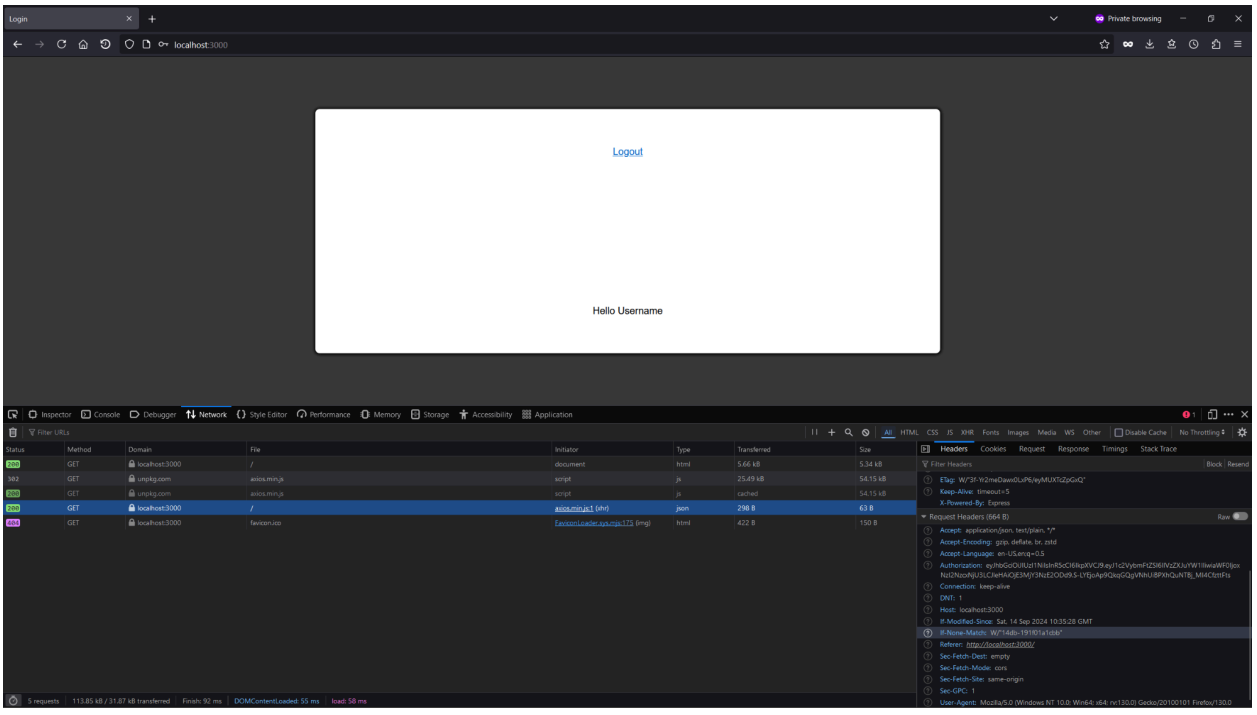

Рисунок 1.24 – Успішна авторизація, де значення JWT-токена зберігається у атрибуті Authorization

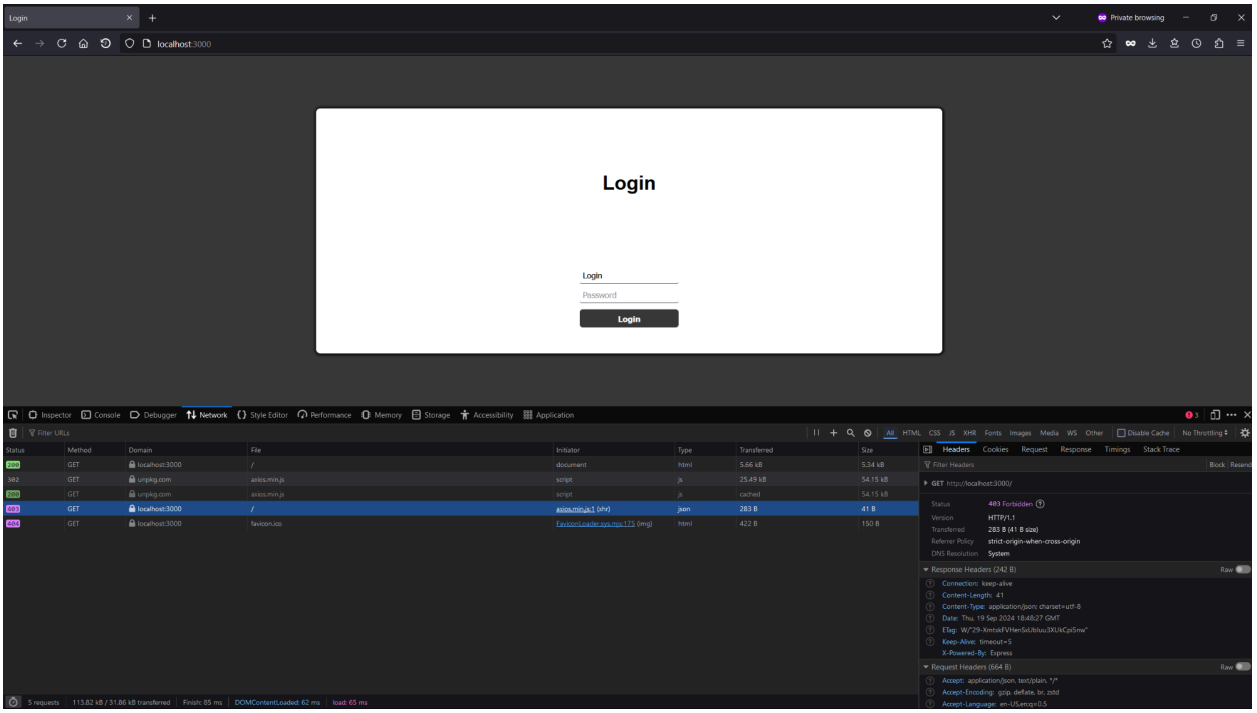

Рисунок 1.25 – Автоматична деавторизація після закінчення актуальності токена

Як уже було сказано, JWT-токен складається з трьох частин, розділених крапками: заголовка (header), корисного навантаження (payload) і підпису (signature). Прикладу формату JWT-токену авторизації:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IlVzZXJuYW1lIiwiaWF0IjoxNzI2NzgzNjI5LCJleHAiOjE3MjY3ODM2NTl9.CUDuv7VXSXp52TJFdcHvcd-ttwHnUsi9X1g6AavmMBU

Де “eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9” – заголовок, який закодований у Base64 має наступне значення: { "alg": "HS256", "typ": "JWT" } (алгоритм підпису – HMAC з SHA-256). Корисне навантаження: “eyJ1c2VybmFtZSI6IlVzZXJuYW1lIiwiaWF0IjoxNzI2NzgzNjI5LCJleHAiOjE3MjY3ODM2NTl9”, яке після декодування має такий вигляд: { "username": "Username", "iat": 1726783629, "exp": 1726783659 } і зберігає значення імені користувача, часу створення токену та часу завершення дії токену (30 секунд у даному випадку). Остання частина – це підпис, має наступне значення: “CUDuv7VXSXp52TJFdcHvcd-ttwHnUsi9X1g6AavmMBU” і слугує криптографічним підипсом, згенерованим на основі значень, записаних у

заголовку, корисному навантаженні та секретного ключа, ця частина слугує для підтвердження цілісності та достовірності токена.

Змінений код за допомогою якого вдалося досягти результату:

```
const path = require('path');
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const JWT_SECRET_KEY = 'NUKE';
const JWT_EXPIRATION = '30s';

const app = express();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

const port = 3000;

const users = [
    {
        login: 'Login',
        password: 'Password',
        username: 'Username',
    },
    {
        login: 'Login1',
        password: 'Password1',
        username: 'Username1',
    }
```

```
]

app.use((req, res, next) => {
    const token = req.headers['authorization'];

    if (token) {
        jwt.verify(token, JWT_SECRET_KEY, (err, decoded) => {
            if (err) {
                return res.status(403).json({ message: 'Token is invalid or expired' });
            }
            req.user = decoded;
            next();
        });
    } else {
        next();
    }
});

app.get('/', (req, res) => {
    if (req.user) {
        return res.json({
            username: req.user.username,
            logout: 'http://localhost:3000/logout'
        });
    }
    res.sendFile(path.join(__dirname + '/index.html'));
});

app.get('/logout', (req, res) => {
```

```
    res.json({ message: 'Logged out' });
});

app.post('/api/login', (req, res) => {
    const { login, password } = req.body;

    const user = users.find((user) => {
        return user.login === login && user.password === password;
    });

    if (user) {
            const token = jwt.sign({ username: user.username }, JWT_SECRET_KEY, {
expiresIn: JWT_EXPIRATION });
        res.json({ token });
    } else {
        res.status(401).send();
    }
});

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
});
```

Висновок: у ході виконання лабораторної роботи було проаналізовано три основні застосунки, які реалізують механізми проведення авторизації: Basic Auth, Forms Auth, та Token Auth.

Кожен із застосунків має свої недоліки та переваги, але всі вони мають певні вразливості, які можуть бути використані зловмисниками. Basic Auth передає паролі у відкритому вигляді, що є серйозною загрозою безпеці. Forms Auth використовує сесії, збережені у файлі, що створює ризики втрати даних або перехоплення сесії, особливо за відсутності HTTPS. Token Auth показав схожі проблеми зі збереженням сесій та відсутністю тайм-аутів.

Було виконано додаткове завдання, де метод Token Auth було модифіковано для використання JWT (JSON Web Token). Використання JWT забезпечило більший рівень безпеки, оскільки токен є самодостатнім і містить підпис, який дозволяє перевірити його цілісність. Окрім цього, у JWT-токена є термін дії, після якого автоматично відбувається деавторизація. Проте для повної безпеки варто використовувати захищене з’єднання (HTTPS) і розглянути додаткові заходи безпеки, як-от захист від CSRF та session fixation.

Загалом, використання JWT є кращим вибором у порівнянні з попередніми методами, але для забезпечення високого рівня безпеки авторизації потрібно розглядати та впроваджувати додаткові механізми захисту.