

Міністерство освіти і науки України

Національний технічний університет України

„КПІ імені Ігоря Сікорського ”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

## **ЗВІТ**

лабораторна робота №1 з

курсу «Безпека програмного забезпечення»

Тема: «Основні методи авторизації»

Перевірів:

Виконав:

Група ІІІ-11 Головня Олександр

Київ 2024

Завдання:

Лабораторна робота 1

Роздивитись основні методи авторизації

Викачати репозиторій з лекціями [https://github.com/Kreolwolf1/auth\\_examples](https://github.com/Kreolwolf1/auth_examples)

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

Для отримання додаткового балу: модифікувати `token_auth` аплікейшен змінивши токен на JWT.

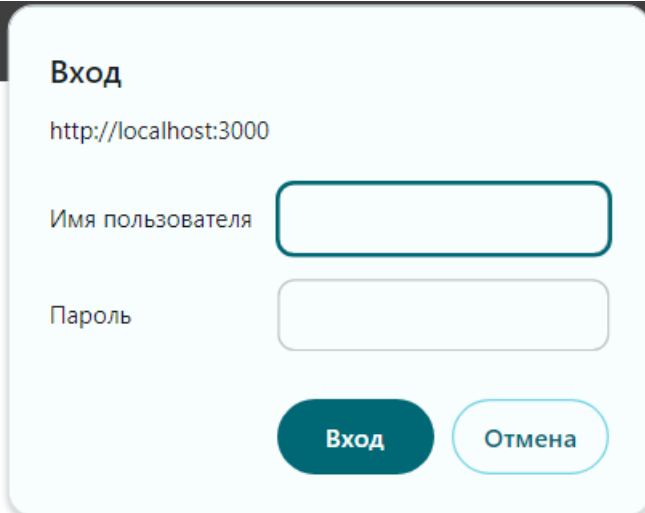
Результати виконання:

**Повний лістинг файлів можна знайти тут:**

<https://github.com/YeaLowww/KPI->

[ALL/tree/main/Fourth%20year/Software%20security](https://github.com/YeaLowww/KPI-ALL/tree/main/Fourth%20year/Software%20security)

1. **basic\_auth.** При запуску `index.js`, за адресою `localhost:3000` було показано вікно для введення логіну та паролю, яке зображено на рисунку 1.



The image shows a web browser window with a light blue background. At the top, the title 'Вход' (Login) is displayed. Below it, the URL 'http://localhost:3000' is shown. There are two input fields: the first is labeled 'Имя пользователя' (Username) and the second is labeled 'Пароль' (Password). Below the input fields are two buttons: a dark blue button labeled 'Вход' (Login) and a light blue button labeled 'Отмена' (Cancel).

Рисунок 1 – Вікно для введення даних авторизації

Цей код створює простий веб-сервер за допомогою Express, який використовує базову аутентифікацію HTTP. Основна логіка роботи коду:

1. Підключається Express
2. Встановлюється порт, на якому буде прослуховуватися сервер: **`const port = 3000.`**

3. Якщо заголовок відсутній, сервер відправляє відповідь статусом 401 та встановлює заголовок WWW-Authenticate, щоб браузер показав діалогове вікно для введення логіну та пароля.
4. Якщо заголовок авторизації присутній, він розкодовується з Base64 і перевіряється логін та пароль. Якщо вони співпадають з **login == 'DateArt' && password == '2408'**, логін зберігається у **req.login**, і виконується далі. Якщо логін був успішно перевірений, сервер відправляє відповідь з привітанням: Hello DateArt, додаючи ім'я користувача з **req.login**.
5. Якщо логін та пароль не співпадають, сервер знову відправляє відповідь статусом 401 та встановлює заголовок WWW-Authenticate.

Увійти можна лише раз

2. **forms\_auth:** В цьому методі, на відміну від попереднього, приступе зберігання сесії у файлі на сервері та використання cookie. Таким чином не доведеться вводити логін та пароль при кожному оновленні сторінки. Основна логіка роботи коду:

1. Підключаються необхідні модулі: uuid, express, cookie-parser, on-finished, body-parser, path, fs.
2. Створюється екземпляр Express: `const app = express();`.
3. Встановлюються `app.use(cookieParser());`.
4. Створюється клас Session, який відповідає за роботу з сесіями. Він має методи `set`, `get`, `init`, `destroy` для зберігання, отримання, ініціалізації та знищення сесій в файлі `sessions.json`.
5. Створюється екземпляр `sessions`.
6. Обробляється кожен запит. Він перевіряє наявність cookies з сесією. Якщо сесія є, вона встановлюється у `req.session`, якщо немає - створюється нова сесія.
7. Є два маршрути: для головної сторінки `/'` і для виходу з системи `/'logout'`.

8. Маршрут `'/api/login'` приймає POST-запит з логіном та паролем користувача, перевіряє їх правильність, і якщо вони вірні, встановлює дані користувача в сесію.

Цей код демонструє простий механізм автентифікації з використанням сесій і куків за допомогою Express.

Також у цьому методі реалізована функція `logout`.

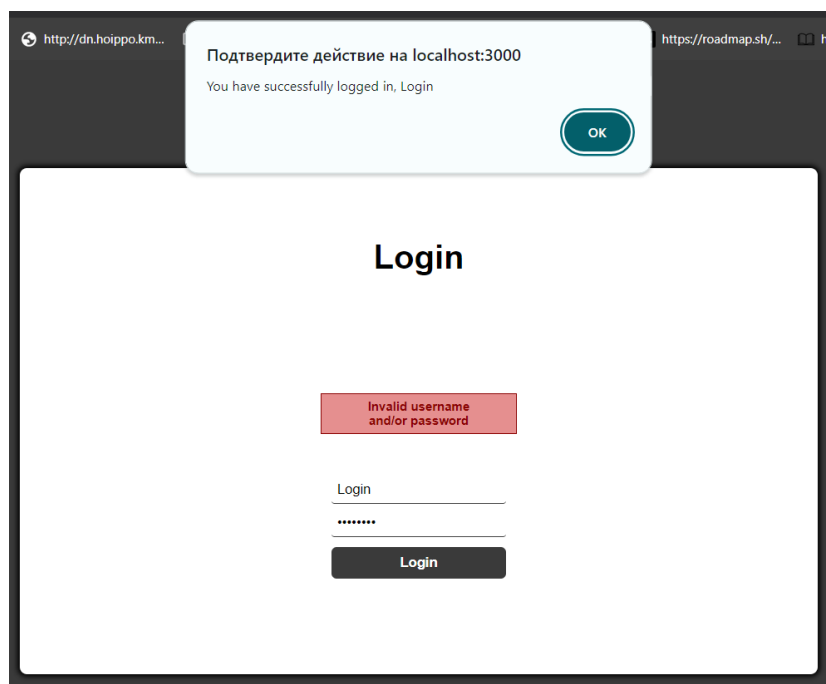


Рисунок 2 – Вигляд сторінки при успішній авторизації.

3. **token\_auth**: Схожа з попередньою сторінка. При аналізі коду було помічено, що у якості токена передається **id** сесії з файлу з попередньої роботи. Щоб уникнути роботи з сесіями та використовувати JsonWebToken трохи змінимо код.

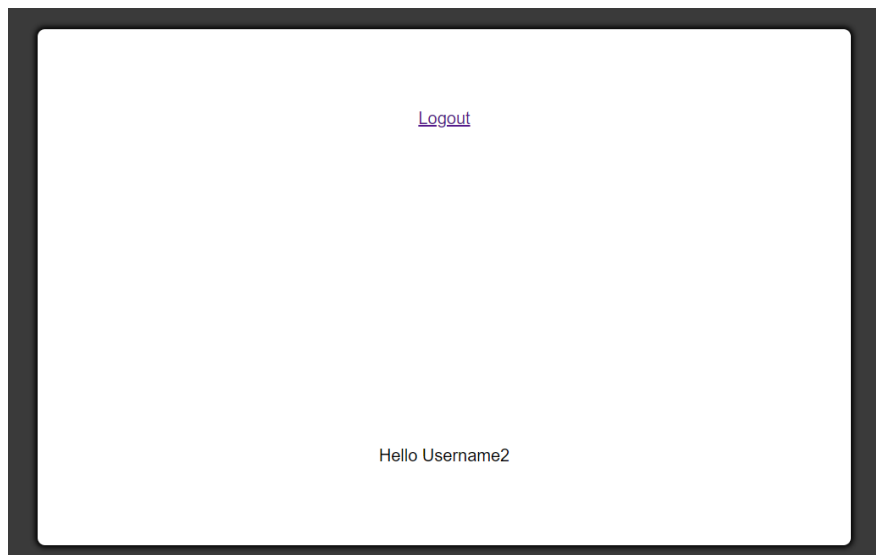


Рисунок 3 – Вигляд сторінки при успішній авторизації.

Основні відмінності цього коду від попередніх:

1. Використання бібліотеки **jsonwebtoken** для створення та перевірки JWT токенів.
2. Метод **verifyToken** перевіряє наявність JWT токена у заголовку **Authorization**, перевіряє його валідність та декодує вміст токена. Якщо токен валідний, він зберігає дані користувача у **req.user** і передає у наступний обробник.
3. Замість використання класу **Session** для зберігання та управління сесіями, використовується створення JWT токена під час автентифікації користувача.

Отже, основна різниця полягає в тому, що перший підхід використовує сесії та куки для зберігання стану сесій, тоді як другий підхід використовує JWT токени для зберігання інформації про автентифікацію на клієнтському боці.

Висновки: У ході виконання лабораторної роботи було опрацьовано основні методи авторизації на веб-сервісах. Було порівняно базову авторизацію, авторизацію, що будується на сесіях та на токенах. Також було модифіковано `token_auth` аплікейшен змінивши токен на JWT.(додаткове завдання)

## Лістинг коду.

```
const uuid = require('uuid');
const express = require('express');
const onFinish = require('on-finished');
const bodyParser = require('body-parser');
const path = require('path');
const port = 3000;
const fs = require('fs');
const jwt = require('jsonwebtoken');

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

const JWT_SECRET = 'secret';

const users = [
  {
    login: 'Login2',
    password: 'Password2',
    username: 'Username2',
  },
  {
    login: 'Login1',
    password: 'Password1',
    username: 'Username1',
  }
]

app.get('/', verifyToken, (req, res) => {
  res.json({
    username: req.user.username,
    logout: 'http://localhost:3000/logout'
  });
});

app.get('/logout', (req, res) => {
  res.redirect('/');
});

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, login: user.login },
      JWT_SECRET);

    res.json({ token });
  } else {
    res.status(401).send('Unauthorized');
  }
});

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

function verifyToken(req, res, next) {
  const token = req.header('Authorization');
```

```
    if (!token) {
      return res.sendFile(path.join(__dirname + '/index.html'));
    }

    jwt.verify(token, JWT_SECRET, (err, decoded) => {
      if (err) {
        return res.sendFile(path.join(__dirname + '/index.html'));
      }

      req.user = decoded;
      next();
    });
  }
}
```