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 № 5 з дисципліни
«Безпека програмного забезпечення»

“**Засвоєння базових навичок роботи з валідацією токенів**”

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

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

Київ 2024

ЛАБОРАТОРНА РОБОТА № 5

Тема роботи: Засвоєння базових навичок роботи з валідацією токенів.

Мета роботи: Засвоїти базових навичок роботи з валідацією токенів.

Основне завдання: Розширити Лабораторну работу 4 перевіркой сігнатури JWT токена. Приклади SDK https://auth0.com/docs/quickstart/backend. У випадку асиметричного ключа, public є можливість отримати за посиланням https://kpi.eu.auth0.com/pem, або за формулою https://[API_DOMAIN]/pem Надати код рішення.

Виконання основного завдання:

Для виконання роботи буде використаний сервіс ідентифікації користувачів, створений для виконання другої та третьої робіт на платформі Auth0.

Auth0 – це платформа управління ідентифікацією та доступом (IAM), яка дозволяє розробникам легко додавати безпечну автентифікацію в додатки. Auth0 надає різноманітні методи входу, включаючи соціальні мережі (Google, Facebook, Twitter), електронну пошту, пароль, а також багатофакторну аутентифікацію. Платформа пропонує гнучкі налаштування для забезпечення різних рівнів безпеки і управління доступом, дозволяючи масштабувати рішення для будь-яких типів додатків і бізнесів. З Auth0 компанії можуть зосередитися на своїх основних продуктах, залишаючи питання кібербезпеки та управління користувачами на надійну платформу.

Процес створення нового ресурсу показаний на рисунках 1.1 – 1.3. Під час цих етапів буде створений сервіс аутентифікації користувачів з відповідними встановленими правилами та конфігурацією входу виключно за допомогою паролю.

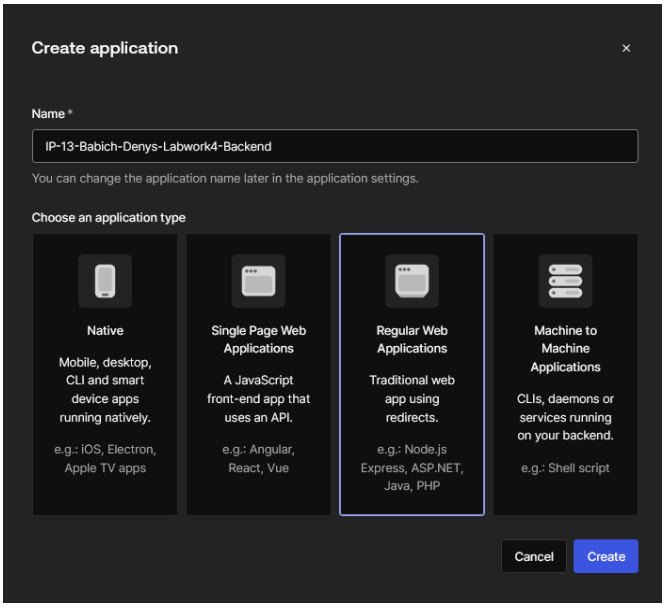

Рисунок 1.1 – Створення сервісу для прийому запитів від клієнта

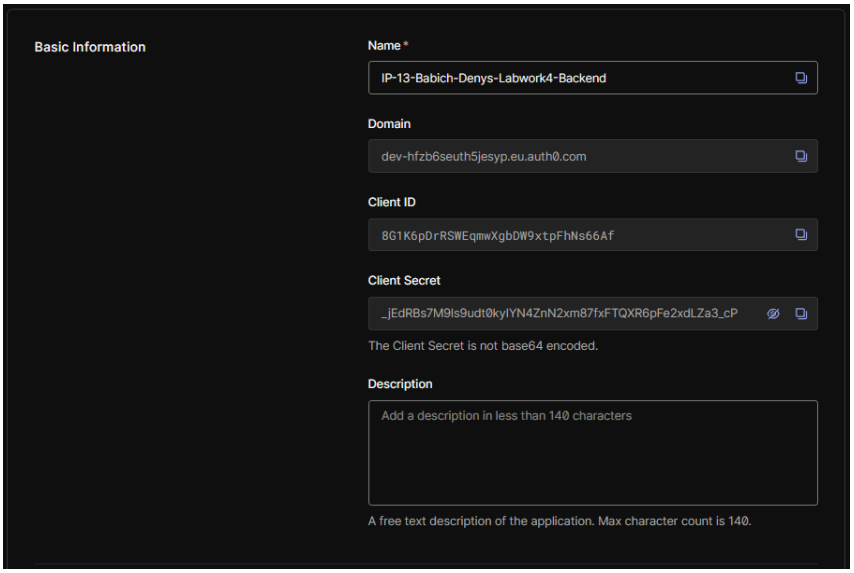

Рисунок 1.2 – Отримані значення полів необхідних для підключення

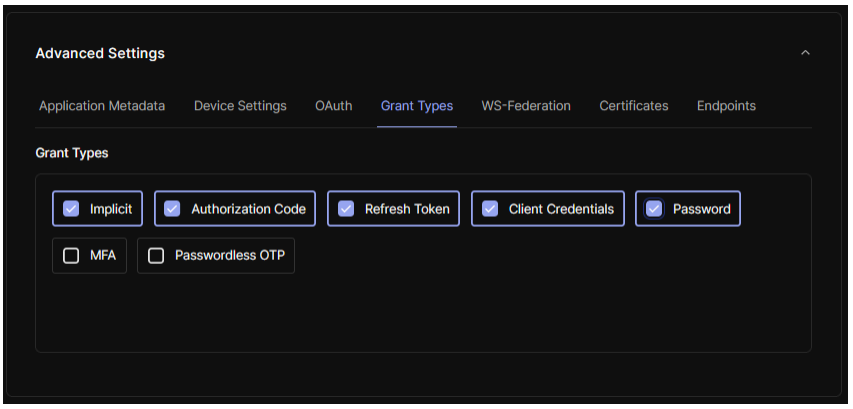

Рисунок 1.3 – Конфігурація з метою встановлення правил підключення за допомогою паролю

Виконавши конфігурацію стороннього сервісу аутентифікації, можна приступати до реалізації самої клієнтської частини, шляхом модифікації вихідного коду з репозиторію, наведеного у завданні.

```
PS D:\Projects\KPI\Software-Security\labs\4\src\token_auth> npm install express axios body-parser uuid dotenv express-session
```

Рисунок 1.4 – Встановлення необхідних модулів

Першим чином було отримано JWT token мого авторизованого користувача. Самі запити на отримання токену користувача та інші операції виконуватимуться за допомогою онлайн-сервісу Postman, який надає зручний графічний інтерфейс для створення та налаштування подібних запитів.

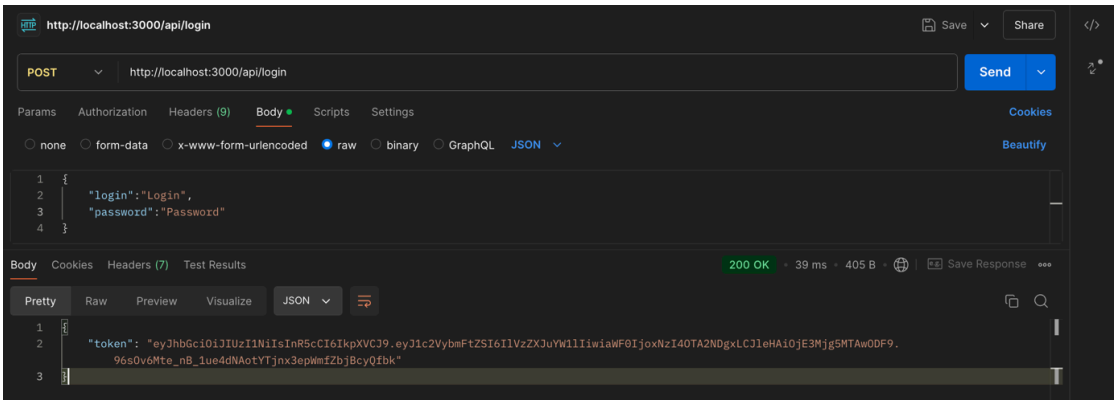

Рисунок 1.5 – Отримання JWT token

Авторизація та аутентифікація наведені на рисунку 1.6.

```
// POST-запит на авторизацію
app.post('/api/login', (req, res) => {
  const { login, password } = req.body;

  // Знаходимо користувача в масиві
  const user = users.find(
    (user) => user.login === login && user.password === password
  );

  if (user) {
    // Зберігаємо дані в сесії
    req.session.username = user.username;
    req.session.login = user.login;

    // Створюємо JWT токен
    const token = jwt.sign(
      { username: user.username },
      'umMFiGfBGCMjywkI-qpg_vRUn8E3sEUaW6I65j58P2lSlymqfmT3gq1k5l7HFDQ_', // Секретний ключ
      { expiresIn: '1h' } // Час дії токена
    );

    // Повертаємо токен
    return res.json({ token });
  } else {
    // Відповідаємо статусом 401, якщо авторизація не вдалася
    return res.status(401).send('Unauthorized');
  }
});
```

Рисунок 1.6 – Отриманий код для авторизації

Цей код обробляє POST-запит на /api/login для аутентифікації користувачів. Він перевіряє логін і пароль з тіла запиту, знаходить користувача в масиві, і якщо користувач знайдений, зберігає його дані в сесії та генерує JWT токен, підписаний секретним ключем. Якщо аутентифікація не вдалася, повертає статус 401 (Unauthorized).

Код для перевірки валідності токену наведений на рисунку 1.7.

```
async function refreshToken(req) {
    const { refresh_token } = req.session.tokens;
    try {
        const response = await axios({
            method: 'post',
            url: `https://${process.env.AUTH0_DOMAIN}/oauth/token`,
            headers: { 'content-type': 'application/x-www-form-urlencoded' },
            data: new URLSearchParams({
                grant_type: 'refresh_token',
                client_id: process.env.AUTH0_CLIENT_ID,
                client_secret: process.env.AUTH0_CLIENT_SECRET,
                refresh_token: refresh_token,
            }),
        });

        req.session.tokens.access_token = response.data.access_token;
        req.session.tokens.refresh_token = response.data.refresh_token;
        req.session.tokens.expires_in = Date.now() + response.data.expires_in * 1000;
    } catch (error) {
        console.error('Token refresh failed:', error.response?.data || error.message);
        throw new Error('Token refresh failed');
    }
}
```

Рисунок 1.7 – Код перевірки валідності токену

Цей код перевіряє дійсність JWT токена, отримуючи публічний ключ з Auth0 за допомогою Axios. Функція getPublicKey запитує ключ, а verifyToken використовує його для перевірки токена, вказуючи алгоритм HS256. Якщо токен недійсний, виводиться повідомлення про помилку; якщо дійсний, виводяться декодовані дані токена. Код містить виклик функції verifyToken з переданим токеном (котрий не повністю представлений у фрагменті).

Результат виконання можна побачити на рисунку 1.8.

```
Токен недійсний: secretOrPublicKey must be a symmetric key when using HS256
```

Рисунок 1.8 – Невдала спроба авторизації за семестричним ключем

Було отримано незадовільний результат, тому було вирішено використати

стандартний метод jwt для верифікації токену (рис. 1.9).

```
const axios = require("axios");
const jwt = require("jsonwebtoken");

const secret = "ZRF8Op0tWM36p1_hxXTU-B0K_Gq_-eAVtlrQpY24CasYiDmcXBhNS6IJMNcz1EgB";

// Функція для перевірки токена
function verifyToken(token) {
  jwt.verify(token, secret, { algorithms: ["HS256"] }, (err, decoded) => {
    if (err) {
      console.log("Токен недійсний:", err.message);
    } else {
      console.log("Токен дійсний:", decoded);
    }
  });
}

const token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IlVzZXJuYW1lIiw

verifyToken(token);
```

Рисунок 1.9 – Змінений код для валідації jwt-токену

Цей код перевіряє дійсність JWT токена за допомогою секретного ключа. Він імплементує функцію verifyToken, яка використовує метод jwt.verify для перевірки токена, вказуючи алгоритм HS256. Якщо токен недійсний, виводиться повідомлення про помилку; якщо дійсний, виводяться декодовані дані токена. Код також містить приклад токена, який потрібно перевірити, і виклик функції з цим токеном.

```
Токен дійсний: { username: 'Username', iat: 1728906656, exp: 1728910256 }
```

Рисунок 1.10 – Успішна авторизація

Модифікований код виконує авторизацію через зовнішній сервіс Auth0, замість локальної перевірки логіна і пароля. Таким чином, замість пошуку користувача в масиві users, він надсилає асинхронний запит на Auth0 з параметрами login і password, а також з додатковими полями (client_id, client_secret, audience, scope), що зберігаються у змінних середовища. У разі успішної авторизації сервер отримує access_token від Auth0 і повертає його клієнту для подальшої роботи. Якщо авторизація не вдалася, клієнту повертається помилка з кодом 401. Цей підхід робить авторизацію більш безпечною, використовуючи централізований зовнішній сервіс.

Висновок:

Ця робота дала глибше розуміння того, як працює аутентифікація та авторизація в сучасних веб-додатках за допомогою JWT токенів. У процесі роботи було впроваджено механізм перевірки токенів, використовуючи як секретні ключі, так і публічні ключі з зовнішніх сервісів, таких як Auth0. Це дозволило зрозуміти, як такі технології забезпечують безпеку даних і запобігають несанкціонованому доступу.

Генерація токенів після успішної аутентифікації користувачів продемонструвала важливість їх подальшої верифікації при кожному запиті. Такий підхід забезпечує безпеку системи без необхідності зберігання сесій на сервері, що, у свою чергу, покращує масштабованість додатка. Також було виявлено, як легко токени можуть бути підроблені без належного підпису та перевірки, тому вивчено важливість використання алгоритмів, таких як HS256.

Здобутий досвід дозволив переконатися, що ресурси системи можна ефективно захистити, гарантуючи доступ до захищених даних і функціоналу виключно для автентифікованих користувачів.

ДОДАТОК А

ПРОГРАМНИЙ КОД ЛАБОРАТОРНОЇ РОБОТИ

index.js

```
require('dotenv').config();
const express = require('express');
const path = require('path');
const axios = require('axios');
const session = require('express-session');

const app = express();
app.use(express.json());
const port = process.env.PORT || 3000;
app.use(express.urlencoded({ extended: true }));

app.use(
    session({
        secret: 'IP-13-Babich-Denys',
        resave: false,
        saveUninitialized: true,
        cookie: { secure: false },
    })
);

async function refreshToken(req) {
    const { refresh_token } = req.session.tokens;
    try {
        const response = await axios({
            method: 'post',
            url: `https://${process.env.AUTH0_DOMAIN}/oauth/token`,
```

```
            headers: { 'content-type': 'application/x-www-form-urlencoded' },
            data: new URLSearchParams({
                grant_type: 'refresh_token',
                client_id: process.env.AUTH0_CLIENT_ID,
                client_secret: process.env.AUTH0_CLIENT_SECRET,
                refresh_token: refresh_token,
            }),
        });

        req.session.tokens.access_token = response.data.access_token;
        req.session.tokens.refresh_token = response.data.refresh_token;
        req.session.tokens.expires_in = Date.now() + response.data.expires_in * 1000;
    } catch (error) {
        console.error('Token refresh failed:', error.response?.data || error.message);
        throw new Error('Token refresh failed');
    }
}

app.use(async (req, res, next) => {
    if (req.session.tokens && Date.now() > req.session.tokens.expires_in - 60000) {
        try {
            await refreshToken(req);
        } catch (error) {
            return res.status(401).send('Failed to refresh token');
        }
    }
    next();
});
```

```
app.get('/', async (req, res) => {
    if (req.session.tokens) {
        try {
            const { access_token } = req.session.tokens;
            const response = await axios.get(
                `https://${process.env.AUTH0_DOMAIN}/userinfo`,
                {
                    headers: {
                        Authorization: `Bearer ${access_token}`,
                    },
                }
            );
            return res.json({
                user: response.data,
                logout: '/logout',
            });
        } catch (error) {
            console.error('Error:', error.response?.data || error.message);
            req.session.destroy();
        }
    }
    res.sendFile(path.join(__dirname, 'index.html'));
});

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

```
app.post('/api/login', async (req, res) => {
  try {
    const { login, password } = req.body;
    const response = await axios({
      method: 'post',
      url: `https://${process.env.AUTH0_DOMAIN}/oauth/token`,
      headers: { 'content-type': 'application/x-www-form-urlencoded' },
      data: new URLSearchParams({
        grant_type: 'password',
        username: login,
        password: password,
        client_id: process.env.AUTH0_CLIENT_ID,
        client_secret: process.env.AUTH0_CLIENT_SECRET,
        audience: `https://${process.env.AUTH0_DOMAIN}/api/v2/`,
        scope: 'offline_access openid profile email',
      }),
    });

    req.session.tokens = {
      access_token: response.data.access_token,
      refresh_token: response.data.refresh_token,
      expires_in: Date.now() + response.data.expires_in * 1000,
    };

    res.json({ success: true, token: response.data.access_token });
  } catch (error) {
    console.error('Login failed:', error.response?.data || error.message);
    res.status(401).send('Login failed');
```

```
    }
});

app.post('/api/register', async (req, res) => {
    try {
        const { username, email, password } = req.body;
        const response = await axios({
            method: 'post',
            url: `https://${process.env.AUTH0_DOMAIN}/dbconnections/signup`,
            headers: { 'content-type': 'application/json' },
            data: {
                client_id: process.env.AUTH0_CLIENT_ID,
                email,
                username,
                password,
                connection: 'Username-Password-Authentication',
            },
        });
        res.json({ success: true, message: 'User registered successfully.' });
    } catch (error) {
        console.error('Registration failed:', error.response?.data || error.message);
        res.status(400).send('Registration failed');
    }
});

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

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 and Registration</title>
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
</head>

<body>
    <main id="main-holder">
        <a href="/logout" id="logout">Logout</a>

        <div id="form-container">
            <div id="login-container">
                <h2 id="login-title">Login</h2>
                <div id="login-error-msg-holder">
                        <p id="login-error-msg" class="error-msg">Invalid email <span
id="error-msg-second-line">and/or
                            password</span></p>
                </div>
                <form id="login-form" action="/api/login" method="post">
                                <input type="email" name="login" id="email-field"
class="login-form-field" placeholder="Email"
                        required>
                            <input type="password" name="password" id="password-field"
class="login-form-field"
```

```
                placeholder="Password" required>
            <input type="submit" value="Login" id="login-form-submit">
        </form>
    </div>

    <div id="registration-container">
        <h2 id="registration-title">Register</h2>
        <div id="registration-error-msg-holder">
            <p id="registration-error-msg" class="error-msg">Registration failed. Please try again.</p>
        </div>
        <form id="registration-form" action="/api/register" method="post">
                <input type="email" name="email" id="register-email-field" class="login-form-field"
                placeholder="Email" required>
            <input type="text" name="username" id="register-username-field" class="login-form-field"
                placeholder="Username" required>
                <input type="password" name="register-password" id="register-password-field"
                class="login-form-field" placeholder="Password" required>
            <input type="submit" value="Register" id="registration-form-submit">
        </form>
    </div>
  </div>
</main>

<style>
  * {
```

```
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: Arial, sans-serif;
    display: flex;
    align-items: center;
    justify-content: center;
    min-height: 100vh;
    background-color: #f0f0f0;
}

#main-holder {
    width: 100%;
    max-width: 500px;
    padding: 20px;
    background-color: white;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    text-align: center;
}

#form-container {
    display: flex;
    flex-direction: column;
    gap: 20px;
}
```

```
#login-container,
#registration-container {
    padding: 20px;
    background-color: #e9e9e9;
    border-radius: 10px;
}

h2 {
    margin-bottom: 20px;
    color: #333;
}

.login-form-field {
    width: 100%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 1em;
}

#login-form-submit,
#registration-form-submit {
    width: 100%;
    padding: 10px;
    border: none;
    border-radius: 5px;
    background-color: #4CAF50;
```

```
    color: white;
    font-size: 1em;
    cursor: pointer;
}

#login-form-submit:hover,
#registration-form-submit:hover {
    background-color: #45a049;
}

#logout {
    display: block;
    margin-bottom: 20px;
    color: #007BFF;
    text-decoration: none;
    cursor: pointer;
}

.error-msg {
    color: #D8000C;
    background-color: #FFD2D2;
    padding: 10px;
    border-radius: 5px;
    margin-bottom: 10px;
    display: none;
}

.error-msg.show {
    display: block;
```

```
    }
</style>

<script>
    const session = sessionStorage.getItem('session');
    let token;

    try {
        token = JSON.parse(session).token;
    } catch (e) { }

    if (token) {
        axios.get('/', {
            headers: {
                Authorization: token
            }
        }).then((response) => {
            const { username } = response.data.user;
            if (username) {
                const mainHolder = document.getElementById("main-holder");
                const loginErrorMsg = document.getElementById("login-error-msg");
                loginErrorMsg.remove();
                mainHolder.append(`Hello ${username}`);
                logoutLink.style.opacity = 1;
            }
        });
    }

    const loginForm = document.getElementById("login-form");
```

```
const loginButton = document.getElementById("login-form-submit");
const loginErrorMsg = document.getElementById("login-error-msg");
const logoutLink = document.getElementById("logout");

logoutLink.addEventListener("click", (e) => {
    e.preventDefault();
    sessionStorage.removeItem('session');
    location.reload();
});

loginButton.addEventListener("click", (e) => {
    e.preventDefault();
    const email = loginForm.login.value;
    const password = loginForm.password.value;

    axios({
        method: 'post',
        url: '/api/login',
        data: {
            login: email,
            password
        }
    }).then((response) => {
        const { username } = response.data;
        sessionStorage.setItem('session', JSON.stringify(response.data));
        location.reload();
    }).catch((response) => {
        loginErrorMsg.classList.add('show');
    });
```

```
});

const registrationForm = document.getElementById("registration-form");
const registrationButton = document.getElementById("registration-form-submit");
const registrationErrorMsg = document.getElementById("registration-error-msg");

registrationButton.addEventListener("click", (e) => {
    e.preventDefault();
    const email = registrationForm.email.value;
    const username = registrationForm.username.value;
    const password = registrationForm['register-password'].value;

    axios({
        method: 'post',
        url: '/api/register',
        data: {
            username: username,
            password: password,
            email: email
        }
    }).then((response) => {
        alert('Registration successful! You can now log in.');
        registrationForm.reset();
    }).catch((response) => {
        registrationErrorMsg.classList.add('show');
    });
});
```

```
    </script>

</body>

</html>
```

.env

```
PORT=3000
AUTH0_DOMAIN=dev-hfzb6seuth5jesyp.eu.auth0.com
AUTH0_CLIENT_ID=8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af
AUTH0_CLIENT_SECRET=_jEdRBs7M9ls9udt0kyIYN4ZnN2xm87fxFTQXR6pF
e2xdLZa3_cP47atEIGd_Qey
```

login.js

```
// Імпортуємо необхідні бібліотеки
const express = require('express');
const jwt = require('jsonwebtoken'); // Для створення токенів JWT
const session = require('express-session'); // Для управління сесіями
const bodyParser = require('body-parser'); // Для парсингу тіла запитів

// Створюємо додаток Express
const app = express();

// Налаштовуємо парсер JSON
app.use(bodyParser.json());

// Налаштовуємо сесії
app.use(
  session({
    secret: 'BABICH_KEY', // Секрет для сесій
    resave: false,
    saveUninitialized: true,
  })
);

// Приклад масиву користувачів
const users = [
  { login: 'user1', password: 'password1', username: 'User One' },
  { login: 'Login', password: 'Password', username: 'User Two' },
];

// POST-запит на авторизацію
```

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

  // Знаходимо користувача в масиві
  const user = users.find(
    (user) => user.login === login && user.password === password
  );
  if (user) {
    // Зберігаємо дані в сесії
    req.session.username = user.username;
    req.session.login = user.login;
    // Створюємо JWT токен
    const token = jwt.sign(
      { username: user.username },

'umMFiGfBGCMjywkI-qpg_vRUn8E3sEUaW6I65j58P2lSlymqfmT3gq1k5l7HFDO
_', // Секретний ключ
      { expiresIn: '1h' } // Час дії токена
    );
    // Повертаємо токен
    return res.json({ token });
  } else {
    // Відповідаємо статусом 401, якщо авторизація не вдалася
    return res.status(401).send('Unauthorized');
  }
});
// Запускаємо сервер
const PORT = 3000;
app.listen(PORT, () => {
```

```
  console.log(`Сервер запущено на порту ${PORT}`);
});
```

verify.js

```
const axios = require("axios");
const jwt = require("jsonwebtoken");

const secret = "ZRF8Op0tWM36p1_hxXTU-B0K_Gq_-eAVtlrQpY24CasYiDmcXBhNS6IJMNcz1EgB";

// Функція для перевірки токена
function verifyToken(token) {
  jwt.verify(token, secret, { algorithms: ["HS256"] }, (err, decoded) => {
    if (err) {
      console.log("Токен недійсний:", err.message);
    } else {
      console.log("Токен дійсний:", decoded);
    }
  });
}

const token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6IlVzZXJuYW1lIiwiaWF0IjoxNzI5MTUwNjAyLCJleHAiOjE3MjkxNTQyMDJ9.u6tsuyLFc_Bs1Z36GzW1dhMUSuquPKRixZkfaIWVE78";

verifyToken(token);
```