

**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 № 4 з дисципліни  
«Безпека програмного забезпечення»

**“Засвоєння базових навичок OAuth2 авторизаційного протокола”**

**Виконав(ла)**

*ІІ-13 Бабіч Денис*

\_\_\_\_\_  
(шифр, прізвище, ім'я, по батькові)

**Перевірів(ла)**

*Соколовський В. В.*

\_\_\_\_\_  
(шифр, прізвище, ім'я, по батькові)

Київ 2024

## ЛАБОРАТОРНА РОБОТА № 4

**Тема роботи:** Засвоєння базових навичок OAuth2 авторизаційного протокола.

**Мета роботи:** Засвоїти базові навички авторизаційного протокола OAuth2.

**Основне завдання:** Використовуючи наведені налаштування з лабораторної роботи 2 – 3 та приведених запитів модифікувати аплікейшен [https://github.com/Kreolwolf1/auth\\_examples/tree/main/token\\_auth](https://github.com/Kreolwolf1/auth_examples/tree/main/token_auth). Використовуючи перевірку юзера та отримання токена з auth0 (password grant type).

### **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 автентифікацію. 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 автентифікації користувачів з відповідними встановленими правилами та конфігурацією входу виключно за допомогою паролю.

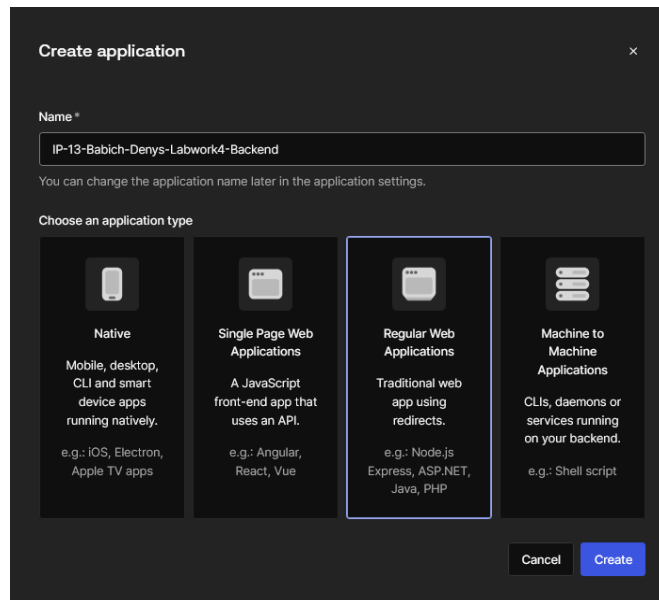


Рисунок 1.1 – Створення сервісу для прийому запитів від клієнта

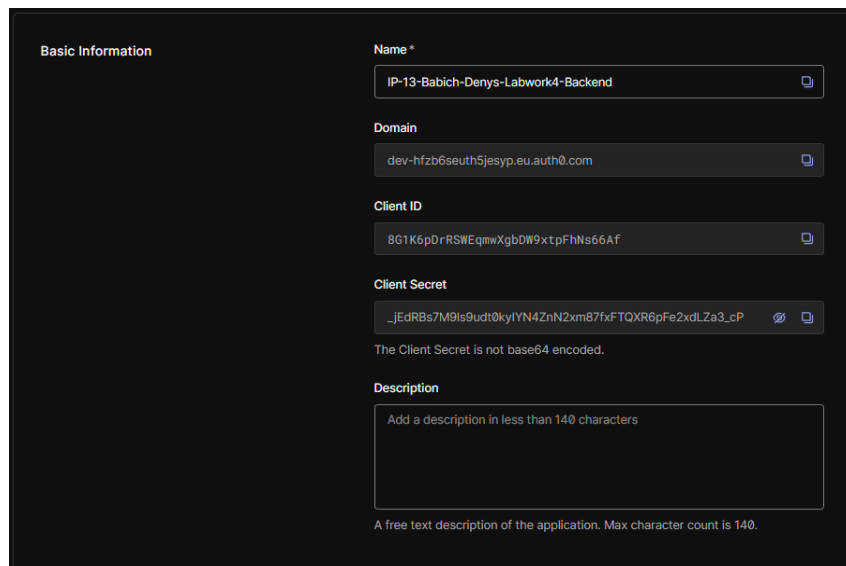


Рисунок 1.2 – Отримані значення полів необхідних для підключення

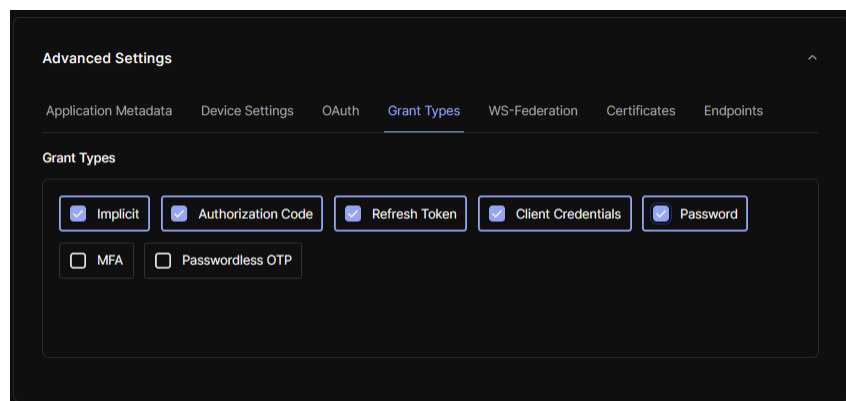


Рисунок 1.3 – Конфігурація з метою встановлення правил підключення за допомогою паролю

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

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

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

  if (user) {
    req.session.username = user.username;
    req.session.login = user.login;

    res.json({ token: req.sessionId });
  }

  res.status(401).send();
});
```

Рисунок 1.5 – Код авторизацію користувачів перед модифікацією

```
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');
  }
});
```

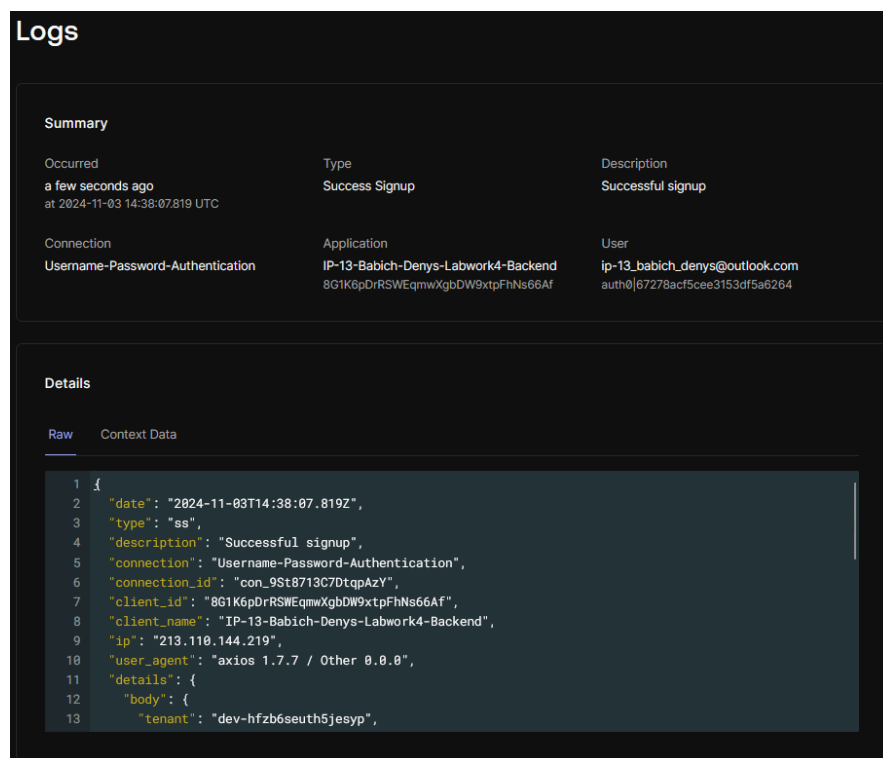
Рисунок 1.6 – Код авторизації користувачів після модифікації

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.

```
PORT=3000
AUTH0_DOMAIN=dev-hfzb6seuth5jesyp.eu.auth0.com
AUTH0_CLIENT_ID=8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af
AUTH0_CLIENT_SECRET=_jEdRBs7M9ls9udt0kyIYN4ZnN2xm87fxFTQXR6pFe2xdLZa3_cP47atEIGd_Qey
```

Рисунок 1.7 – Вміст файлу .env, у якому зберігаються змінні середовища, дані підключення до стороннього ресурсу Auth0

Виконавши створення користувачів подібно до того, як це було зроблено у 3-й роботі, було отриманого нового користувача, інформацію про якого можна побачити на рисунку 1.8.



The screenshot displays the Auth0 Logs interface. At the top, the word "Logs" is visible. Below it, there is a "Summary" section with a table of log entries. The first entry is a "Successful signup" event that occurred "a few seconds ago" at "2024-11-03 14:38:07.819 UTC". The table lists details such as "Connection" (Username-Password-Authentication), "Application" (IP-13-Babich-Denys-Labwork4-Backend), and "User" (ip-13\_babich\_denys@outlook.com). Below the summary, there is a "Details" section with tabs for "Raw" and "Context Data". The "Raw" tab is selected, showing a JSON object with the following structure:

```
1 {
2   "date": "2024-11-03T14:38:07.819Z",
3   "type": "ss",
4   "description": "Successful signup",
5   "connection": "Username-Password-Authentication",
6   "connection_id": "con_9St0713C7DtpgAzY",
7   "client_id": "8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af",
8   "client_name": "IP-13-Babich-Denys-Labwork4-Backend",
9   "ip": "213.110.144.219",
10  "user_agent": "axios 1.7.7 / Other 0.0.0",
11  "details": {
12    "body": {
13      "tenant": "dev-hfzb6seuth5jesyp",
```

Рисунок 1.8 – Перегляд логів про створених користувачів

Використавши дані для входу можна перевірити коректність авторизації на ресурсі (рис. 1.9).

Login

ip-13\_babich\_denys@outlook.com

●●●●●●●●

Login

Рисунок 1.9 – Введення даних авторизації

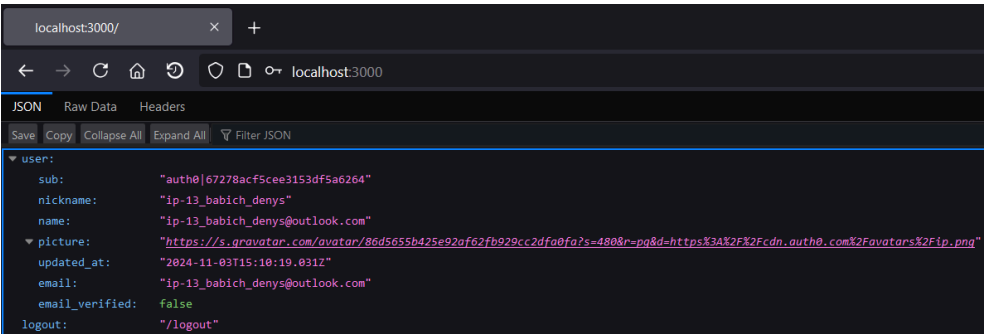


Рисунок 1.10 – Успішна авторизація користувачем

Summary

Occurred	Type	Description
a few seconds ago at 2024-11-03 15:10:19.132 UTC	Success Exchange	Password for Access Token
Connection	Application	User
Username-Password-Authentication	IP-13-Babich-Denys-Labwork4-Backend 8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af	ip-13_babich_denys@outlook.com auth0 67278acf5cee3153df5a6264

Details

Raw

```
1 {
2   "date": "2024-11-03T15:10:19.132Z",
3   "type": "sepf",
4   "description": "Password for Access Token",
5   "connection": "Username-Password-Authentication",
6   "connection_id": "con_9St8713C7DtpAzY",
7   "client_id": "8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af",
8   "client_name": "IP-13-Babich-Denys-Labwork4-Backend",
9   "ip": "213.110.144.219",
10  "client_ip": "213.110.144.219",
11  "user_agent": "axios 1.7.7 / Other 0.0.0",
12  "user_id": "auth0|67278acf5cee3153df5a6264",
13  "user_name": "ip-13_babich_denys@outlook.com",
```

Рисунок 1.11 – Підтвердження успішної аутентифікації за допомогою логів сервісу Auth0

**Додаткове завдання:** додатково розширити аплікейшен створенням юзера та перевіркою життя токена (у разі близького завершення – оновити токен використовуючи refresh-token grant type).

### **Виконання додаткового завдання:**

Функція створення користувачів наведена на рисунку 1.12.

```
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');
  }
});
```

Рисунок 1.12 – Реалізація функції реєстрації користувачів

Цей код створює API-ендпоінт для реєстрації нового користувача за допомогою платформи Auth0. Він приймає username, email та password з тіла запиту req.body, а потім відправляє асинхронний POST-запит до Auth0 API за допомогою axios, використовуючи URL-адресу, сформовану на основі змінної AUTH0\_DOMAIN з .env-файлу. Запит містить дані користувача, а також client\_id і назву з'єднання (Username-Password-Authentication), що використовується Auth0 для аутентифікації. Якщо запит до Auth0 успішний, сервер повертає JSON-відповідь з повідомленням про успішну реєстрацію. У разі помилки, наприклад, якщо дані користувача не пройшли валідацію, у відповідь надсилається статус 400 з повідомленням "Registration failed".

Перевірка реєстрація наведена на рисунках 1.13 – 1.17.

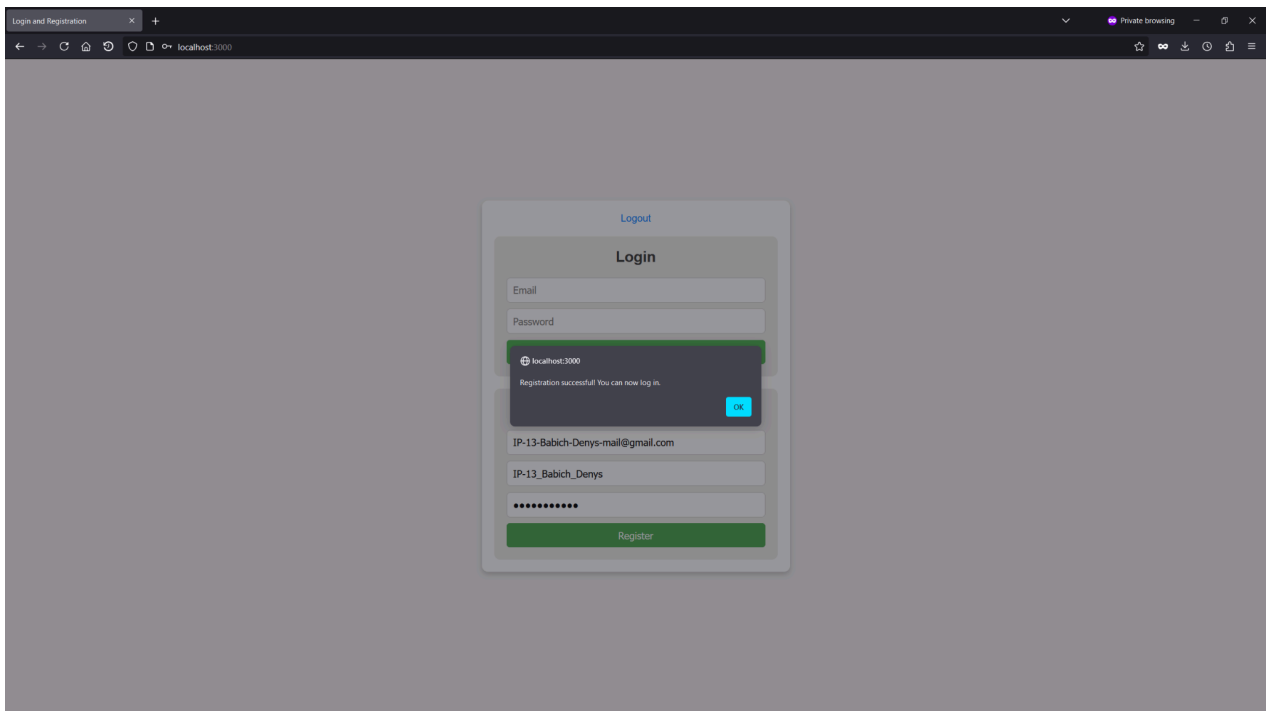


Рисунок 1.13 – Успішна реєстрація користувача за допомогою вказаних даних

### Summary

Occurred	Type	Description
a minute ago at 2024-11-03 16:00:34.537 UTC	Success Signup	Successful signup
Connection	Application	User
Username-Password-Authentication	IP-13-Babich-Denys-Labwork4-Backend 8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af	ip-13-babich-denys-mail@gmail.com auth0 67279e22c242b17e004a5673

### Details

RawContext Data

```
1 {
2   "date": "2024-11-03T16:00:34.537Z",
3   "type": "ss",
4   "description": "Successful signup",
5   "connection": "Username-Password-Authentication",
6   "connection_id": "con_9St8713C7DtqpAzY",
7   "client_id": "8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af",
8   "client_name": "IP-13-Babich-Denys-Labwork4-Backend",
9   "ip": "213.110.144.219",
10  "user_agent": "axios 1.7.7 / Other 0.0.0",
11  "details": {
12    "body": {
13      "tenant": "dev-hfzb6seuth5jesyp",
```

Рисунок 1.14 – Повідомлення про успішну реєстрацію у логах авторизаційного сервісу Auth0



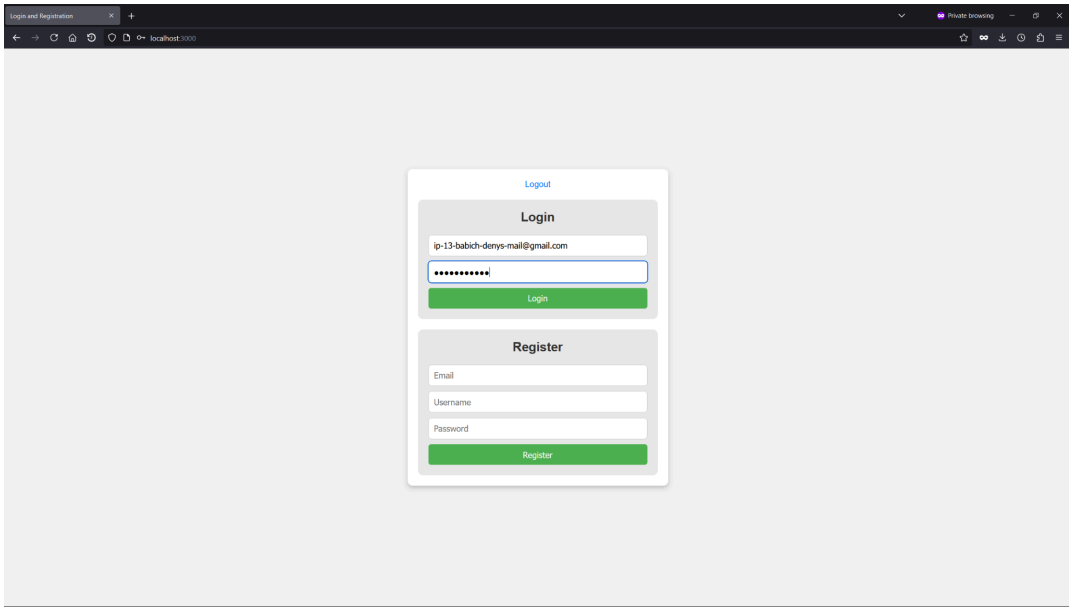


Рисунок 1.15 – Введення зареєстрованих даних для авторизації

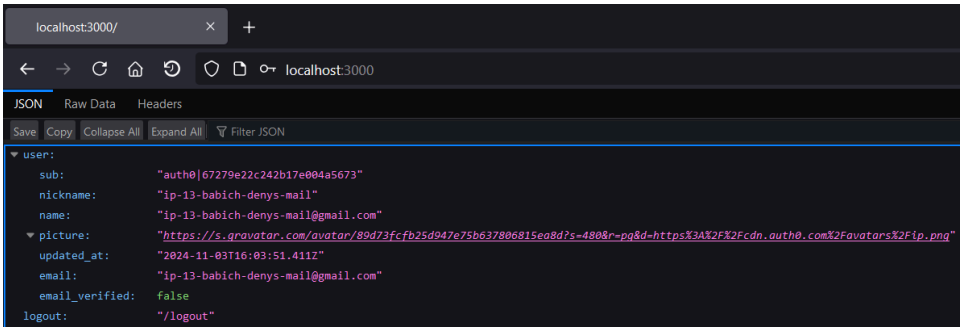


Рисунок 1.16 – Повідомлення про успішну авторизацію

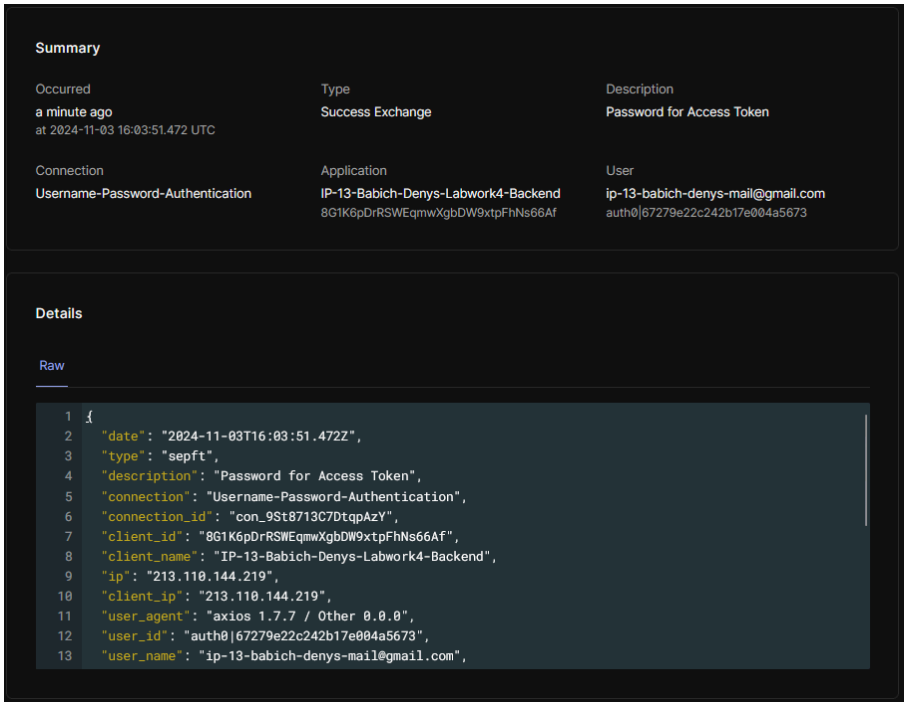


Рисунок 1.17 – Запис про успішну авторизацію у логах на сервісі Auth0

Другим підзавданням додаткового завдання є реалізацію механізму оновлення токена доступу буде реалізований завдяки асинхронній функції, яка за 60 секунд до скидання активної сесії клієнта, автоматично пролонгує існуючих токен авторизації без необхідності повторної авторизації та введення даних користувача.

```
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();
});
```

Рисунок 1.18 – Отриманий код оновлення токена авторизації за 60 секунд до скидання активної поточної сесії

Цей код реалізує автоматичне оновлення токена доступу в додатку. Функція `refreshToken` надсилає запит до Auth0 для отримання нового токена доступу, використовуючи наявний `refresh_token`. Якщо токен успішно оновлено, новий токен і його термін дії зберігаються в сесії користувача.

Middleware-функція перевіряє, чи існує токен у сесії і чи спливає він протягом хвилини. Якщо токен скоро спливає, викликається `refreshToken`, щоб оновити його до продовження запиту. Якщо оновлення токена не вдається, користувач отримує статус помилки 401.

З метою перевірки коректності роботи автопродлонгації токenu активної сесії, було змінено конфігурацію сервісу авторизації, де встановленні відповідні значення скидання поточної сесії до 60 секунд у неактивному режимі та 120 секунд у активному стані. Відповідні налаштування конфігурації можна побачити на рисунках 1.19 – 1.20. Перегляд системних логів зображені на рисунках 1.21 – 1.22, де можна побачити початковий запис з авторизацією користувача та подальші звернення до API з метою оновлення токenu сесії.

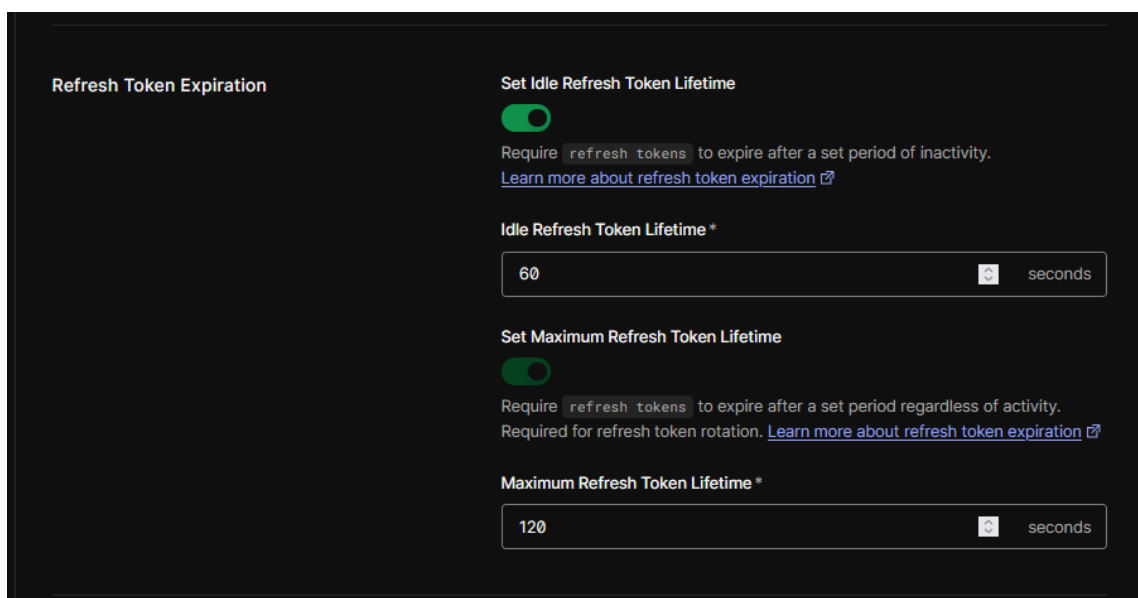


Рисунок 1.19 – Конфігурація сервісу авторизації Auth0 з метою встановлення обмеження на термін дії токenu авторизації активної сесії

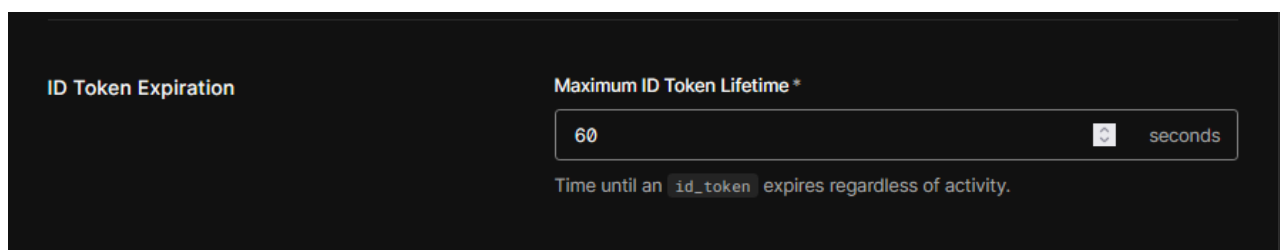


Рисунок 1.20 – Конфігурація сервісу авторизації Auth0 з метою встановлення обмеження на термін дії токenu авторизації активної сесії

Summary

Occurred	Type	Description
a minute ago at 2024-11-03 16:21:30.919 UTC	Success Exchange	Password for Access Token
Connection	Application	User
Username-Password-Authentication	IP-13-Babich-Denys-Labwork4-Backend 8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af	ip-13-babich-denys-mail@gmail.com auth0 67279e22c242b17e004a5673

Details

Raw

```
1 {
2   "date": "2024-11-03T16:21:30.919Z",
3   "type": "sepf",
4   "description": "Password for Access Token",
5   "connection": "Username-Password-Authentication",
6   "connection_id": "con_9St8713C7DtpqAzY",
7   "client_id": "8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af",
8   "client_name": "IP-13-Babich-Denys-Labwork4-Backend",
9   "ip": "213.110.144.219",
10  "client_ip": "213.110.144.219",
11  "user_agent": "axios 1.7.7 / Other 0.0.0",
12  "user_id": "auth0|67279e22c242b17e004a5673",
13  "user_name": "ip-13-babich-denys-mail@gmail.com",
```

Рисунок 1.21 – Початковий запит на отримання токену

✓	API Operation	Update a client	3 minutes ago	N/A	N/A
✓	API Operation	Update a client	4 minutes ago	N/A	N/A
✓	API Operation	Update a client	5 minutes ago	N/A	N/A
✓	Success Exchange	Password for Acce...	7 minutes ago	Username-P...	IP-13-Babich...

Рисунок 1.22 – Перегляд логів

Таким чином можна побачити, що відповідні запити дійсно приходили на віддалений сервер й авторизаційний токен активної сесії успішно продовжувався.

## **Висновок:**

У ході виконання лабораторної роботи були засвоєні базові навички роботи з авторизаційним протоколом OAuth2, зокрема, його реалізація через сторонній сервіс Auth0 із використанням password grant type для отримання access\_token.

Виконані налаштування на платформі Auth0 дозволили створити захищену авторизацію, яка забезпечує перевірку користувачів, отримання токенів доступу, а також управління сесіями. Крім того, реалізовано додатковий функціонал, зокрема автоматичне оновлення токена доступу через refresh-token grant type для продовження активної сесії без необхідності повторного введення логіна і пароля користувача. Це дозволяє оптимізувати процес аутентифікації та підвищити безпеку, оскільки запити на оновлення токенів обробляються централізовано на сторонньому сервері.

Завдяки використанню Auth0 з'явилася можливість спростити авторизаційні механізми для клієнтського додатка, підвищуючи надійність і зменшуючи вразливість до потенційних загроз безпеці. Реалізована автоматична пролонгація токена доступу також демонструє переваги централізованого керування сесіями, що особливо корисно для додатків з високими вимогами до кібербезпеки.

**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