

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 № 6 з дисципліни
«Безпека програмного забезпечення»

“Засвоєння базових навичок роботи з валідацією токенів”

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 № 6

Тема роботи: Засвоєння базових навичок роботи з валідацією токенів.

Мета роботи: Засвоїти базові навички роботи з OAuth2 протоколом.

Основне завдання: Розширити Лабораторну роботу 4, змінивши логін сторінку на стандартну від SSO провайдера, для цього, треба зробити редірект на API_DOMAIN <https://kpi.eu.auth0.com/authorize> додатково додати параметри Вашого аплікейшена `client_id`, `redirect_uri`, `response_type = code`, `response_mode=query`

https://kpi.eu.auth0.com/authorize?client_id=JlvCO5c2IBHlAe2patn6l6q5H35qxti0&redirect_uri=http%3A%2F%2Flocalhost%3A3000&response_type=code&response_mode=query. Надати код рішення.

Для отримання додаткового балу: додатково розширити завдання обробкою редіректа та отриманням юзер токена за допомогою code grant type.<https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow>

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.

Create application

Name *

IP-13-Babich-Denys-Labwork4-Backend

You can change the application name later in the application settings.

Choose an application type

- Native**
Mobile, desktop, CLI and smart device apps running natively.
e.g.: iOS, Electron, Apple TV apps
- Single Page Web Applications**
A JavaScript front-end app that uses an API.
e.g.: Angular, React, Vue
- Regular Web Applications**
Traditional web app using redirects.
e.g.: Node.js Express, ASP.NET, Java, PHP
- Machine to Machine Applications**
CLIs, daemons or services running on your backend.
e.g.: Shell script

Cancel Create

Рисунок 1.1 – Створення сервісу для прийому запитів від клієнта

Basic Information

Name *

IP-13-Babich-Denys-Labwork4-Backend

Domain

dev-hfzb6seuth5jesyp.eu.auth0.com

Client ID

8G1K6pDrRSWEqmwXgbDW9xtpFhNs66Af

Client Secret

_jEdRBs7M9Is9udt0kylYN4ZnN2xm87fxFTQXR6pFe2xdLZa3_cP

The Client Secret is not base64 encoded.

Description

Add a description in less than 140 characters

A free text description of the application. Max character count is 140.

Рисунок 1.2 – Отримані значення полів необхідних для підключення

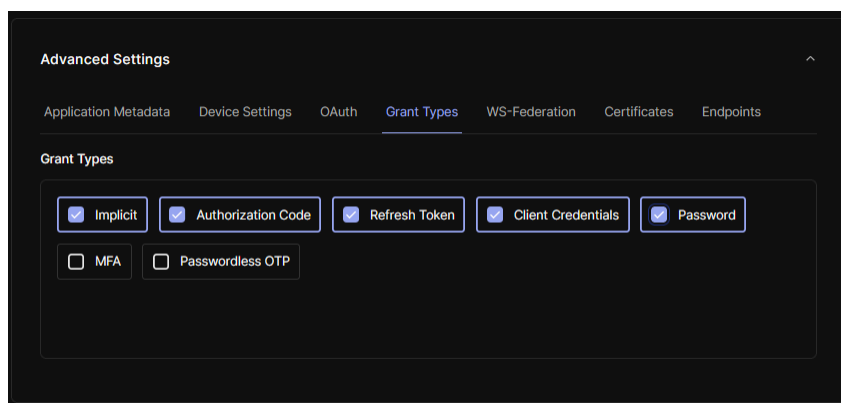


Рисунок 1.3 – Конфігурація з метою встановлення правил підключення за допомогою паролю

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

```
// Маршрут для редіректу на сторінку логіну "редірект": Unknown word.
app.get('/login', (req, res) => {
  const authDomain = 'https://dev-hfzb6seuth5jesyp.eu.auth0.com/authorize';
  const clientId = 'q0SmG7tFBjLJIYxZ5NDhazlumungw0AZ'; "Dhaz": Unknown word.
  const redirectUri = 'http://localhost:3000/callback';
  const responseType = 'code';
  const responseMode = 'query';

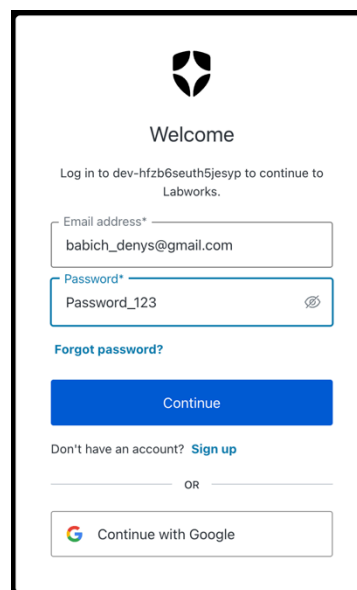
  // Створення URL для редіректу "редірект": Unknown word.
  const loginUrl = `${authDomain}?client_id=${clientId}&redirect_uri=${encodeURIComponent(redirectUri)}&response_type=${responseType}&response_mode=${responseMode}`;

  // Редірект на Auth0 "Редірект": Unknown word.
  res.redirect(loginUrl);
});
```

Рисунок 1.5 – Код аутентифікації логіну

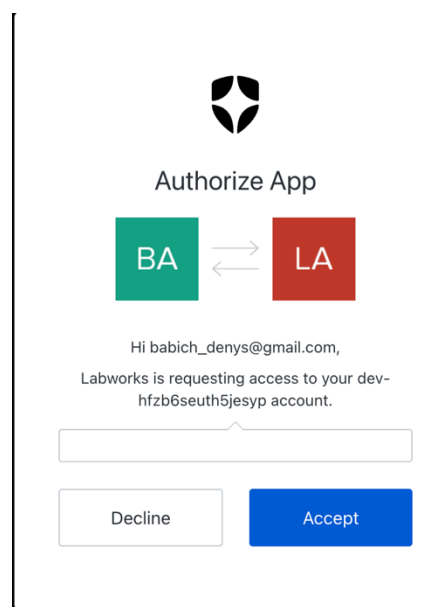
Цей код реалізує маршрут у додатку Node.js із використанням фреймворку Express, який відповідає за перенаправлення користувача на сторінку авторизації через сторонній сервіс аутентифікації Auth0. Коли користувач звертається до маршруту /login, сервер створює URL, який відповідає вимогам Auth0 для початку процесу аутентифікації. Для цього встановлюється домен авторизації Auth0 (authDomain), унікальний ідентифікатор клієнта (clientId), URI для перенаправлення після успішної аутентифікації (redirectUri), тип відповіді (responseType), що вказує на отримання авторизаційного коду, та режим відповіді (responseMode), що визначає передачу даних через рядок запиту URL.

У кінцевому URL всі ці параметри формуються як рядок із дотриманням стандарту URL-енкодування для коректної передачі спеціальних символів. Потім, сервер відповідає користувачеві редіректором на сформований URL, передаючи його в `res.redirect`. Це перенаправляє користувача на сторінку авторизації Auth0, де він зможе увійти в систему або зареєструватися. Після завершення процесу авторизації Auth0 перенаправить користувача назад на вказаний у `redirectUri` маршрут із доданими до запиту параметрами, такими як код авторизації.



The image shows a login form titled "Welcome" with the Auth0 logo at the top. Below the title, it says "Log in to dev-hfzb6seuth5jesyp to continue to Labworks." The form contains two input fields: "Email address*" with the value "babich_denys@gmail.com" and "Password*" with the value "Password_123". There is a "Forgot password?" link below the password field. A blue "Continue" button is at the bottom of the form. Below the button, it says "Don't have an account? [Sign up](#)". At the very bottom, there is a "Continue with Google" button with the Google logo.

Рисунок 1.6 – Успішна авторизація



The image shows an "Authorize App" screen with the Auth0 logo at the top. Below the title, there is a diagram showing a green box labeled "BA" and a red box labeled "LA" connected by two horizontal arrows, one pointing right and one pointing left. Below the diagram, it says "Hi babich_denys@gmail.com, Labworks is requesting access to your dev-hfzb6seuth5jesyp account." There is a large empty input field below the text. At the bottom, there are two buttons: "Decline" and "Accept".

Рисунок 1.7 – Перенаправлення на сторінку провайдер послуг

Після того як користувач успішно авторизувався його переправляє на основну сторінку callback.

```
// Обробка callback після логіну
app.get('/callback', (req, res) => {
  // Тут оброблятиметься відповідь з кодом авторизації
  const code = req.query.code;
  if (code) {
    res.send(`Authorization code: ${code}`);
    // Далі тут можна отримати access token через бекенд
  } else {
    res.send('Authorization code not found');
  }
});
```

Рисунок 1.8 – Обрання маршруту callback

Цей код додає три маршрути до Node.js додатку з використанням Express, кожен із яких виконує конкретну задачу: обробку callback після логіну, доступ до захищеного профілю, і доступ до публічного маршруту.

Перший маршрут, `/callback`, обробляє відповідь після успішної аутентифікації користувача через Auth0. Коли користувач перенаправляється на цей маршрут із параметром `code` у запиті, сервер отримує значення авторизаційного коду з `req.query.code`. Якщо код присутній, сервер відповідає клієнту повідомленням, яке містить цей код. Це зазвичай використовується як перший крок для подальшого обміну коду на токен доступу (`access token`) через бекенд. Якщо ж параметр `code` відсутній, сервер відповідає повідомленням про його відсутність.

[illegible]

Рисунок 1.9 – Результат успішної аутентифікації

Виконання додаткового завдання:

```
app.get('/profile', requireAuth, (req, res) => {
  res.json({ message: 'This is your profile data', accessToken: req.session.access_token });
});

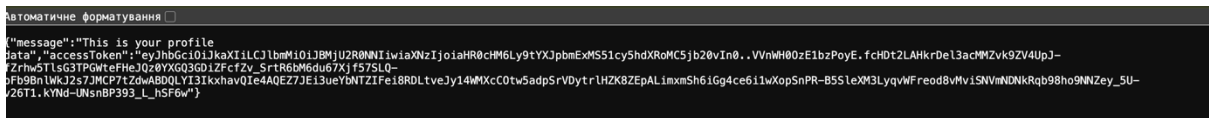
// Публічний маршрут
app.get('/public', (req, res) => {
  res.json({ message: 'This is a public route accessible to anyone.' });
});
```

Рисунок 1.10 – Результат успішної аутентифікації

Цей код визначає два маршрути: /profile, доступний лише для авторизованих користувачів, де сервер відповідає JSON-об'єктом з повідомленням і токеном доступу. /public, відкритий для всіх користувачів і повертає просте повідомлення про публічний статус. Таким чином реалізується захищений доступ до одного ресурсу і відкритий доступ до іншого.

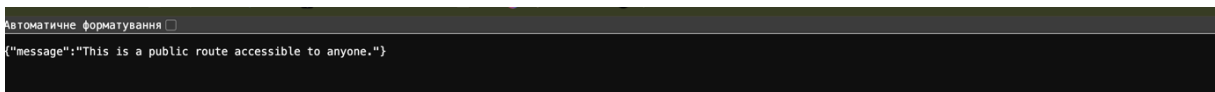
Другий маршрут, /profile, захищений за допомогою middleware requireAuth, який забезпечує, що доступ до маршруту мають лише авторизовані користувачі. Коли користувач звертається до цього маршруту, сервер повертає JSON-відповідь із повідомленням і токеном доступу (accessToken), збереженим у сесії користувача (req.session.access_token). Цей токен може використовуватися для доступу до персоналізованих даних або інших захищених ресурсів.

Третій маршрут, /public, є публічним і не вимагає аутентифікації. Коли будь-який користувач, незалежно від статусу входу в систему, звертається до цього маршруту, сервер відповідає JSON-даними, що містять повідомлення про публічний доступ. Цей маршрут призначений для ресурсів, доступних для всіх користувачів без обмежень.



```
Автоматичне форматування
{"message": "This is your profile data", "accessToken": "eyJhbGciOiJIU2R1IiwiaXNjaW9iaHR0cHMLY9tYXJpbmExMS51cy5hdXRoMC5jb20vIn0..VVnMH00zE1bzPoyE.fcHdt2LAHkrDe13acMzvk9ZV4UpJ-Zrhw5TlSG3TPQWteFHeJ0z0YXGQ3G01ZFc7Zv_SrtR6bM6du67XJf575LQ-jP99bn1Wk32572McP71ZdwAB0QLY31kxhavQ1e4AQEZ7JE13ueY0NTZiFe18RDLtveJy14MMKcC0tw5adpSrVDytrLHZK8ZEpAL1mxmSh61Gg4ce611wKopSnPR-B5S1eXm3LyqvWfReod8vWv1SNVnNDNkRqb98ho9NNZey_5U-226T1.kYnd-UNanBP393_l_hSF6w"}
```

Рисунок 1.11 – Успішний доступ до профілю



```
Автоматичне форматування
{"message": "This is a public route accessible to anyone."}
```

Рисунок 1.12 – Підтвердження доступності профілю

Висновок:

У цій роботі реалізовано систему аутентифікації за допомогою Auth0 у веб-додатку на Node.js, що використовує Express. Вона включає маршрути для авторизації користувачів та захищені ресурси, доступні лише для авторизованих осіб. Основна мета цього підходу полягає в забезпеченні безпеки даних і можливості контролю доступу до певних частин веб-додатку.

Завдяки цій роботі були здобуті навички налаштування аутентифікацію за допомогою сторонніх сервісів, таких як Auth0, а також формувати запити до API для отримання токенів доступу. Це важливо, оскільки дозволяє розробникам безпечно обробляти користувацькі дані та надавати доступ до чутливих ресурсів лише авторизованим особам.

Загалом, реалізація цієї функціональності підвищує рівень безпеки веб-додатків, забезпечуючи при цьому зручний процес авторизації для користувачів. Це критично важливо в умовах сучасних веб-сервісів, де захист персональних даних та конфіденційності є пріоритетом

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.eyJ1c2VybmFtZSI6IiVzZXJuYW11IiwiaWF0IjoxNzI5MTUwNjAyLCJleHAiOjE3MjkxNTQyMDJ9.u6tsuyLFc_Bs1Z36GzW1dhMUSuquPKRixZkfaIWVE78";
```

```
verifyToken(token);
```

redirect.js

```
const express = require('express');  
const session = require('express-session'); // Import session middleware  
const app = express();
```

```
// Set up session middleware
```

```
app.use(session({  
secret:  
'umMFiGfBGCMjywkI-qpg_vRUn8E3sEUaW6I65j58P2lSlymqfmT3gq1k5l7HFDO  
_', // Use a secure secret key  
resave: false,  
saveUninitialized: true  
}));
```

```
const requireAuth = (req, res, next) => {  
  if (!req.session.access_token) {  
    return res.status(401).json({ message: 'Unauthorized. Please login.' });  
  }  
  next();
```

```
};
```

```
// Маршрут для редіректу на сторінку логіну
```

```
app.get('/login', (req, res) => {  
  const authDomain = 'https://dev-hfzb6seuth5jesyp.eu.auth0.com/authorize';  
  const clientId = 'q0SmG7tFBjJIYxZ5NDhaz1umungwOAZ';  
  const redirectUri = 'http://localhost:3000/callback';  
  const responseType = 'code';  
  const responseMode = 'query';
```

```
// Створення URL для редіректу
```

```
    const loginUrl =  
`${authDomain}?client_id=${clientId}&redirect_uri=${encodeURIComponent(redirectUri)}&response_type=${responseType}&response_mode=${responseMode}`;
```

```
// Редірект на Auth0
```

```
  res.redirect(loginUrl);  
});
```

```
// Обробка callback після логіну
```

```
app.get('/callback', (req, res) => {  
  // Тут оброблятиметься відповідь з кодом авторизації  
  const code = req.query.code;  
  if (code) {  
    res.send(`Authorization code: ${code}`);  
    // Далі тут можна отримати access token через бекенд  
  } else {  
    res.send('Authorization code not found');  
  }  
}
```

```
});
```

```
app.get('/profile', requireAuth, (req, res) => {  
    res.json({ message: 'This is your profile data', accessToken:  
req.session.access_token });  
});
```

```
// Публічний маршрут
```

```
app.get('/public', (req, res) => {  
    res.json({ message: 'This is a public route accessible to anyone.' });  
});
```

```
app.listen(3000, () => {  
    console.log('Server is running on http://localhost:3000');  
});
```