
# AKADEMIA HANDLOWA NAUK STOSOWANYCH W RADOMIU

## PRACA ZALICZENIOWA

## ZAGROŻENIA BEZPIECZEŃSTWA SYSTEMÓW

## Laboratoria

Wykładowca: magister inżynier Bartłomiej Kicior

DAWID DULA

NR. ALBUMU: 20238

STUDIA NIESTACJONARNE 2024/2025

III ROK : INFORMATYKA

# Pytanie #1

Sprawdź czym są tokeny JWT, jaka jest ich rola w aplikacji, do czego można je wykorzystać. Znajdź lub wygeneruj i opisz przykładowy token, z czego się składa oraz co można w nim zapisać? Sprawdź jakie paczki nuget można wykorzystać do generowania tokenów w aplikacji .NET, wypisz te rozwiązania i opisz czym się charakteryzują. Odpowiedzi zapisz w dokumencie, z którego później wygenerujesz plik pdf, zapisz informacje do, którego z punktów ćwiczenia się odnoszą.

**Odpowiedź :**

Tokeny JWT ( JSON Web Token ) to standardowy format tokenów uwierzytelniających, wykorzystywany do przekazywania informacji między stronami w sposób bezpieczny. Są one często używane do autoryzacji użytkowników w aplikacjach webowych oraz API.

Rola JWT w aplikacji :

- **Autoryzacja** - użytkownik po poprawnym zalogowaniu otrzymuje token JWT, który jest następnie przesyłany w nagłówkach HTTP jako dowód tożsamości.
- **Bezpieczeństwo** - tokeny JWT mogą być podpisane i zaszyfrowane, aby zabezpieczyć integralność oraz poufność przesyłanych danych.
- **Bezstanowość** - serwer nie musi przechowywać sesji użytkownika, ponieważ wszystkie informacje są zawarte w tokenie.
- **Skalowalność** - dzięki bezstanowości JWT świetnie nadają się do aplikacji rozproszonych i mikroserwisowych.

Przykładowy JWT:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkphbiBEb2UiLCJhZG1pbiI6dHJ1ZX0.KhJvr3XIDohH9fpTirX0A_J6YgAsl2tGwZCQQP1Sw1U

Token JWT składa się z trzech części oddzielonych kropkami :

- **Nagłówek ( Header )** - zawiera informacje o algorytmie podpisu i typie tokenu.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- **Ładunek ( Payload ) -** przechowuje dane użytkownika i dodatkowe informacje ( np. role, uprawnienia ).

```
{
  "sub": "1234567890",
  "name": "Jan Doe",
  "admin": true
}
```

- **Podpis ( Signature )** - zapewnia integralność tokenu, generowany na podstawie nagłówka, ładunku i klucza tajnego.

W tokenie można przechowywać np. :

- ID użytkownika ( sub )
- Nazwę użytkownika ( name )
- Role ( admin, user )
- Datę ważności ( exp )
- Adres e-mail ( email )

**Paczki NuGet do generowania JWT w .NET**

W .NET istnieje kilka popularnych bibliotek do generowania i obsługi tokenów JWT:

I. **System.IdentityModel.Tokens.Jwt**

- Oficjalna biblioteka Microsoftu do pracy z tokenami JWT w .NET.
- Pozwala na generowanie, podpisywanie i walidację tokenów.
- Przykładowe użycie :

```
var tokenHandler = new JwtSecurityTokenHandler();
var key = Encoding.UTF8.GetBytes("superSecretKey123");
var tokenDescriptor = new SecurityTokenDescriptor
{
   Subject = new ClaimsIdentity(new[] { new Claim("sub", "123") }),
   Expires = DateTime.UtcNow.AddMinutes(30),
   SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
SecurityAlgorithms.HmacSha256Signature)
};
var token = tokenHandler.CreateToken(tokenDescriptor);
string jwt = tokenHandler.WriteToken(token);
```

II. **Microsoft.AspNetCore.Authentication.JwtBearer**

- Używana głównie do obsługi JWT w aplikacjach ASP.NET Core.
- Integruje autoryzację na podstawie JWT z ASP.NET Core Identity.
- Konfiguracja w Program.cs :

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
  .AddJwtBearer(options =>
  {
    options.TokenValidationParameters = new TokenValidationParameters
    {
      ValidateIssuer = true,
      ValidateAudience = true,
      ValidateLifetime = true,
      ValidateIssuerSigningKey = true,
      IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey123"))
    };
  });
```

III. **Jose.JWT**

- Lekka i wydajna biblioteka do generowania i dekodowania JWT.
- Obsługuje różne algorytmy podpisywania (HS256, RS256, ES256).
- Przykładowe użycie:

```
var payload = new { sub = "1234567890", name = "Jan Doe" };
string token = Jose.JWT.Encode(payload, Encoding.UTF8.GetBytes("superSecretKey123"),
JwsAlgorithm.HS256);
```

IV. **IdentityServer4**

- Framework do zarządzania uwierzytelnianiem i autoryzacją.
- Obsługuje OAuth 2.0 oraz OpenID Connect.
- Używany w aplikacjach wymagających rozbudowanej kontroli tożsamości i uprawnień.

# Pytanie #2

Zastanów się i zapisz jakie jeszcze informacje można zawrzeć w tokenie JWT, sprawdź i opisz jakie algorytmy bezpieczeństwa można zastosować do kodowania credentiali ( klasa: Microsoft.IdentityModel.Tokens.SecurityAlgorithms ), wypisz trzy oraz zdefiniuj różnice pomiędzy nimi.

**Odpowiedź :**

Token JWT można wzbogacić o różne dodatkowe dane, które pomagają w autoryzacji i zarządzaniu użytkownikami. Przykłady :

- **exp ( Expiration Time )** - określa czas ważności tokenu.
- **iat ( Issued At )** - znacznik czasu wskazujący, kiedy token został wygenerowany.
- **nbf ( Not Before )** - token nie będzie ważny przed tą datą.
- **aud ( Audience )** - określa odbiorcę tokenu (np. „WeatherCheckerUsers").
- **iss ( Issuer )** - informacja o podmiocie, który wygenerował token.
- **role ( Role )** - role użytkownika, np. admin, user.
- **permissions ( Uprawnienia )** - lista uprawnień przypisanych do użytkownika.
- **custom claims** - można dodawać własne pola, np. department, subscriptionLevel.

Np.

```
{
 "sub": "1234567890",
 "name": "Jan Kowalski",
 "email": "jan.kowalski@example.com",
 "role": "admin",
 "permissions": ["read", "write", "delete"],
 "iat": 1711558652,
 "exp": 1711562252
}
```

**Algorytmy bezpieczeństwa do kodowania credentiali ( SecurityAlgorithms )**
W .NET do podpisywania tokenów JWT można wykorzystać różne algorytmy dostępne w klasie Microsoft.IdentityModel.Tokens.SecurityAlgorithms. Poniżej opis trzech popularnych algorytmów :

- **1. HMACSHA256 ( HmacSha256Signature )**
  - Jest to algorytm symetryczny, co oznacza, że ten sam klucz służy do podpisywania i weryfikacji tokenu.
  - Używa funkcji skrótu SHA-256 ( Secure Hash Algorithm ) oraz klucza tajnego do generowania podpisu.

- Jest szybki i wydajny, ale wymaga bezpiecznego przechowywania klucza na serwerze.

Np.

```
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("superSecretKey123"));
var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
```

➢ **2. RS256 ( RsaSha256 )**

- Jest to algorytm asymetryczny, co oznacza, że używa dwóch różnych kluczy :
  - **Prywatny klucz** do podpisywania tokenu.
  - **Publiczny klucz** do weryfikacji tokenu.
- Stosowany w systemach wymagających większego bezpieczeństwa, np. w OpenID Connect.
- Wolniejszy od HMAC, ale bardziej bezpieczny w systemach rozproszonych.

Np.

```
using (var rsa = RSA.Create())
{
   var key = new RsaSecurityKey(rsa);
   var credentials = new SigningCredentials(key, SecurityAlgorithms.RsaSha256);
}
```

➢ **ES256** ( **EcdsaSha256** )

- Wykorzystuje krzywe eliptyczne ( ECDSA ) do podpisywania tokenów.
- Jest bardziej efektywny niż RSA pod względem długości klucza i wydajności.
- Często używany w nowoczesnych systemach wymagających wysokiego poziomu bezpieczeństwa.

Np.

```
var ecdsa = ECDsa.Create();
var key = new ECDsaSecurityKey(ecdsa);
var credentials = new SigningCredentials(key, SecurityAlgorithms.EcdsaSha256);
```

**Różnice między algorytmami**

| Algorytm | Typ | Bezpieczeństwo | Wydajność | Klucz |
|---|---|---|---|---|
| **HMACSHA256** | Symetryczny | Średnie | Wysoka | Jeden wspólny klucz |
| **RS256** | Asymetryczny | Wysokie | Średnia | Para kluczy ( prywatny/publiczny) |
| **ES256** | Asymetryczny | Bardzo wysokie | Wysoka | Para kluczy ( prywatny/publiczny ) |