ASP.NET Core : Sécurité

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

ASP.NET Core

But de la sécurité

Interdire, à un utilisateur, l'accès à une ressource à laquelle il n'a pas droit

ASP.NET Core

But de la sécurité

Interdire, à un utilisateur, l'accès à une ressource à laquelle il n'a pas droit

Deux étapes

- Qui veut accéder à la ressource ? (Authentification)
- A t-il le droit d'y accéder ? (Autorisation)

ASP.NET Core

Configuration de la sécurité

- En utilisant des données statiques (en mémoire, dans un fichier)
- En utilisant des données dynamiques (provenant d'une base de données)

ASP.NET Core

Configuration de la sécurité

- En utilisant des données statiques (en mémoire, dans un fichier)
- En utilisant des données dynamiques (provenant d'une base de données)

Dans **ASP.NET Core**

L'authentification est gérée par le `IAuthenticationService` : utilisé par lemiddlewaree d'authentification.

ASP.NET Core

JWT : **J**SON **W**eb **T**oken

- Librairie d'échange sécurisé d'informations
- Utilisant des algorithmes de cryptage comme **HMAC SHA256** ou **RSA**
- Utilisant les jetons (tokens)
- Un jeton est composé de trois parties séparées par un point :
  - entête (header) : objet **JSON** décrivant le jeton encodé en base 64
  - charge utile (payload) : objet **JSON** contenant les informations du jeton encodé en base 64
  - Une signature numérique = concaténation de deux éléments précédents séparés par un point + une clé secrète (le tout crypté par l'algorithme spécifié dans l'entête)
- Documentation officielle : `https://jwt.io/introduction/`

ASP.NET Core

Entête : exemple

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

ASP.NET Core

Entête : exemple

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

Charge utile : exemple

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

ASP.NET Core

Exemple de construction de signature en utilisant l'algorithme précisé dans l'entête

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

ASP.NET Core

Exemple de construction de signature en utilisant l'algorithme précisé dans l'entête

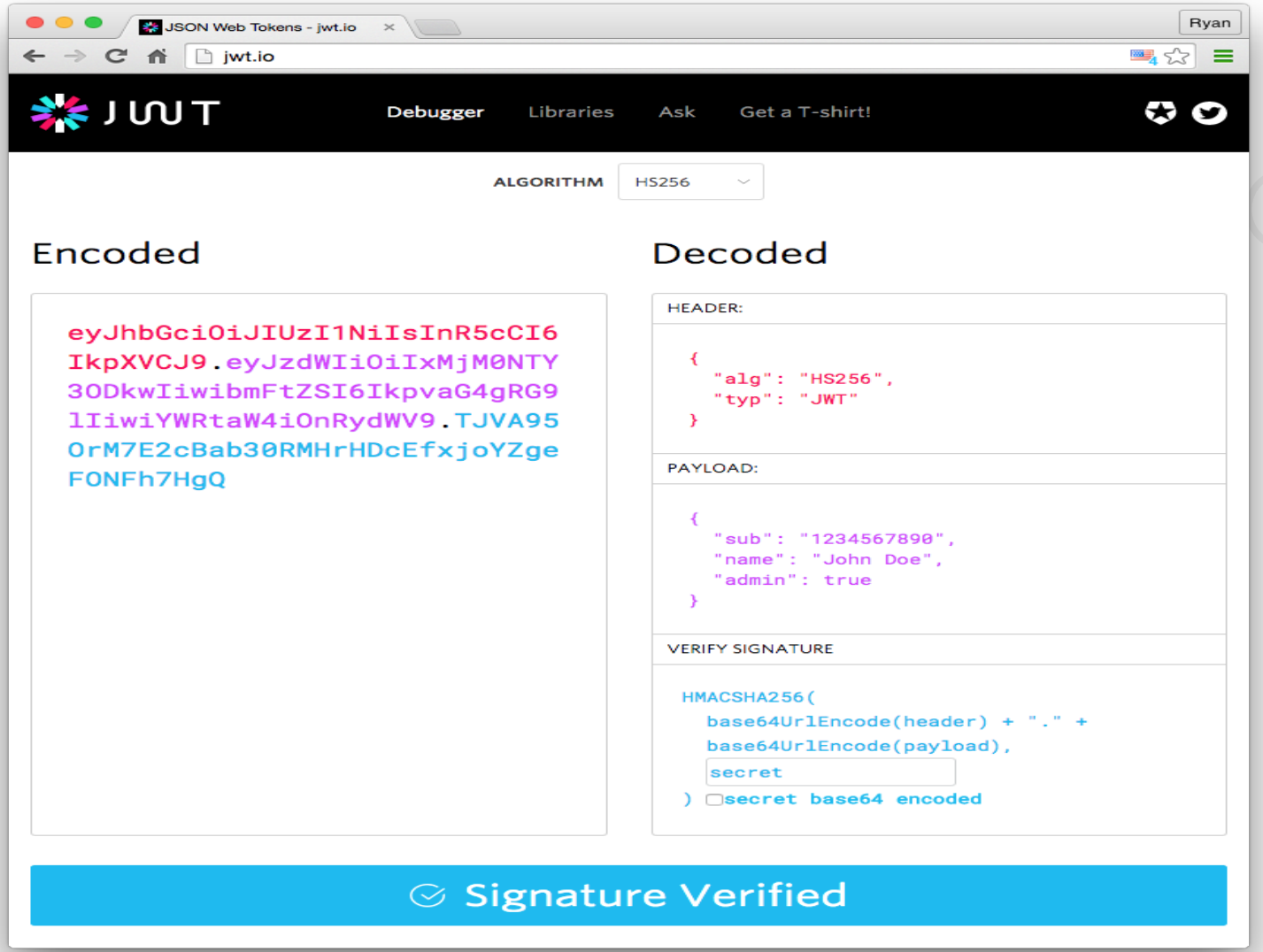
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Résultat

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4

ASP.NET Core

Exemple complet (pour tester `https://jwt.io/#debugger-io`)



ASP.NET Core

Pour décoder les deux premières parties d'un jeton

`http://calebb.net/`

ASP.NET Core

Remarque

Pour tester, utilisons le projet `CoursWebApi` créé dans le chapitre précédent.

ASP.NET Core

Remarque

Pour tester, utilisons le projet `CoursWebApi` créé dans le chapitre précédent.

Assemblies à installer

- `Microsoft.AspNetCore.Authentication.JwtBearer` (version 5.0.13)
- `Microsoft.IdentityModel.Tokens` (version 6.7.1)

ASP.NET Core

Rappel

Pour intégrer des nouveaux assemblies dans le projet, on utilise **NuGet**.

ASP.NET Core

Rappel

Pour intégrer des nouveaux assemblies dans le projet, on utilise **NuGet**.

NuGet

- Gestionnaire de paquets, par défaut, pour **.NET**
- Open-source et gratuit
- Inclus dans *Visual Studio* depuis 2012
- Utilisable aussi en ligne de commande

ASP.NET Core

Utiliser **NuGet** pour installer les assemblies

- Faire un clic droit sur `Dépendances` dans l'`Explorateur de solution`
- Choisir `Gérer les packages NuGet`
- Aller dans l'onglet `Parcourir` et chercher `Microsoft.AspNetCore.Authentication.JwtBearer`
- Choisir la version (version 5.0.13) et installer
- Accepter, attendre la fin de l'installation
- Vérifier l'ajout du package installé dans `Dépendances/packages`

ASP.NET Core

Utiliser **NuGet** pour installer les assemblies

- Faire un clic droit sur `Dépendances` dans l'`Explorateur de solution`
- Choisir `Gérer les packages NuGet`
- Aller dans l'onglet `Parcourir` et chercher `Microsoft.AspNetCore.Authentication.JwtBearer`
- Choisir la version (version 5.0.13) et installer
- Accepter, attendre la fin de l'installation
- Vérifier l'ajout du package installé dans `Dépendances/packages`

Refaire la même chose pour `Microsoft.IdentityModel.Tokens` (version 6.7.1).

ASP.NET Core

Deux classes modèles

- `ConnectionData` : une classe qui nous permet de récupérer les identifiants de l'utilisateur non-connecté (enregistré ou non-enregistré dans notre système).
- `User` : une classe (entité) qui contient les données d'un l'utilisateur connecté (enregistré dans notre système).

ASP.NET Core

Commençons par créer une classe `ConnectionData` qui va nous permettre de récupérer les identifiants de l'utilisateur qui souhaite se connecter

```
using System.ComponentModel.DataAnnotations;

namespace CoursWebApi.Models
{
    public class ConnectionData
    {
        [Required]
        public string Username { get; set; }
        [Required]
        public string Password { get; set; }
    }
}
```

ASP.NET Core

Commençons par créer une classe `ConnectionData` qui va nous permettre de récupérer les identifiants de l'utilisateur qui souhaite se connecter

```
using System.ComponentModel.DataAnnotations;

namespace CoursWebApi.Models
{
    public class ConnectionData
    {
        [Required]
        public string Username { get; set; }
        [Required]
        public string Password { get; set; }
    }
}
```

Les deux propriétés sont obligatoires pour la connexion.

ASP.NET Core

Créons aussi une classe `User` qui contiendra les données d'un utilisateur connecté

```
using System.Text.Json.Serialization;

namespace CoursWebApi.Models
{
    public class User
    {
        public int? Id { get; set; }
        public string Username { get; set; }
        [JsonIgnore]
        public string Password { get; set; }
        public string? Nom { get; set; }
        public string? Prenom { get; set; }
        public string? Email { get; set; }
    }
}
```

ASP.NET Core

Dans `appsettings.json`, ajoutons la clé qui va nous permettre de décoder et encoder le jeton

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Jwt": {
    "Key": "UtiliseUneClePlusSecurisee",
  }
}
```

ASP.NET Core

Dans `Interfaces`, créons l'interface `IUserService`

```
using CoursWebApi.Models;

namespace CoursWebApi.Interfaces
{
    public interface IUserService
    {
        User? CheckUser(ConnectionData data);
        string GenerateJWT(User user);
        User Add(User user);
        User? GetOneById(int id);
    }
}
```

ASP.NET Core

Dans `Service`, créons une classe `UserService` qui implémente l'interface `IUserService`

```
using CoursWebApi.Interfaces;

namespace CoursWebApi.Services
{
    public class UserService : IUserService
    {
    }
}
```

ASP.NET Core

Commençons par injecter pour récupérer les propriétés définies dans `appsettings.json`

```
using CoursWebApi.Interfaces;

namespace CoursWebApi.Services
{
    public class UserService : IUserService
    {
        private IConfiguration _configuration = null;

        public UserService(IConfiguration configuration)
        {
            _configuration = configuration;
        }
    }
}
```

ASP.NET Core

Définissons notre tableau d'utilisateurs

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;

namespace CoursWebApi.Services
{
    public class UserService : IUserService
    {
        private IConfiguration _configuration = null;

        public List<User> Users { get; set; }

        public UserService(IConfiguration configuration)
        {
            _configuration = configuration;
            Users = new List<User>() {
                new User() { Id = 1, Nom = "Wick", Prenom = "John", Username = "Wick",
                  Email = "w@w.fr", Password = "John" },
                new User() { Id = 2, Nom = "Dalton", Prenom = "Jack", Username = "
                  Dalton", Email = "d@d.fr", Password = "Jack" }
            };
        }
    }
}
```

Commençons par implémenter la méthode `CheckUser`

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;

namespace CoursWebApi.Services
{
    public class UserService : IUserService
    {
        private IConfiguration _configuration = null;
        public List<User> Users { get; set; }

        public UserService(IConfiguration configuration)
        {
            _configuration = configuration;
            Users = new List<User>() {
                new User() { Id = 1, Nom = "Wick", Prenom = "John", Username = "Wick", Email =
                   "w@w.fr", Password = "John" },
                new User() { Id = 2, Nom = "Dalton", Prenom = "Jack", Username = "Dalton",
                   Email = "d@d.fr", Password = "Jack" }
            };
        }
        public User? CheckUser(ConnectionData data)
        {
            return Users.Find(elt => elt.Username == data.Username && elt.Password == data.
              Password);
        }
    }
}
```

ASP.NET Core

Ensuite les méthodes `Add` et `GetOneById`

```
public User Add(User user)
{
    Users.Add(user);
    return user;
}

public User? GetOneById(int id)
{
    return Users.Find(elt => elt.Id == id);
}
```

Et enfin la méthode `GenerateJWT`

```
public string GenerateJWT(User user)
{

    var key = _configuration["Jwt:Key"];
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
    var token = new JwtSecurityToken(
         signingCredentials: credentials
    );
    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Et enfin la méthode `GenerateJWT`

```
public string GenerateJWT(User user)
{

    var key = _configuration["Jwt:Key"];
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
    var token = new JwtSecurityToken(
         signingCredentials: credentials
    );
    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Explication

- On commence par récupérer la clé et ensuite on construit un objet `SymmetricSecurityKey` pour la création et la validation du jeton.
- On spécifie l'algorithme de hachage
- On choisit ce qu'il faut ajouter dans le jeton
- On crée et on retourne le jeton

ASP.NET Core

Dans `Program.cs`, définissons ce qu'il faut injecter pour `IUserService`

```
builder.Services.AddSingleton<IUserService, UserService>();
```

ASP.NET Core

Dans `Controllers`**, créons un contrôleur** `UsersController`

```
namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UsersController : ControllerBase
    {
    }
}
```

ASP.NET Core

Dans `UsersController`, injectons l'interface `UsersController`

```
namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UsersController : ControllerBase
    {
        private readonly IUserService userService;

        public UsersController(IUserService userService)
        {
            this.userService = userService;
        }
    }
}
```

Ajoutons maintenant la méthode d'authentification

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UsersController : ControllerBase
    {
        private readonly IUserService userService;

        public UsersController(IUserService userService)
        {
            this.userService = userService;
        }
        [AllowAnonymous]
        [HttpPost]
        public IActionResult Login([FromBody] ConnectionData data)
        {
            var user = userService.CheckUser(data);
            if (user != null)
            {
                var tokenString = userService.GenerateJWT(user);
                return Ok(tokenString);
            }
            return Unauthorized();
        }
    }
}
```

On peut aussi retourner le jeton dans un objet JSON

```
using CoursWebApi.Interfaces;
using CoursWebApi.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace CoursWebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UsersController : ControllerBase
    {
        private readonly IUserService userService;

        public UsersController(IUserService userService)
        {
            this.userService = userService;
        }
        [AllowAnonymous]
        [HttpPost]
        public IActionResult Login([FromBody] ConnectionData data)
        {
            var user = userService.CheckUser(data);
            if (user != null)
            {
                var tokenString = userService.GenerateJWT(user);
                return Ok(new { token = tokenString });
            }
            return Unauthorized();
        }
    }
}
```

ASP.NET Core

Dans `Program.cs`, préparons les paramètres de validation de tokens (reçus)

```
var key = Encoding.ASCII.GetBytes(builder.Configuration["Jwt:Key"]);

var tokenParams = new TokenValidationParameters()
{
    ValidateIssuer = false,
    ValidateAudience = false,
    ValidateLifetime = false,
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = new SymmetricSecurityKey(key),
};
```

ASP.NET Core

Dans `Program.cs`, précisons maintenant l'utilisation de `JWT` et chargeons les paramètres de validation

```
builder.Services.AddAuthentication(JwtBearerDefaults.
   AuthenticationScheme)
     .AddJwtBearer(options =>
     {
         options.RequireHttpsMetadata = true;
         options.TokenValidationParameters = tokenParams;
     });
```

ASP.NET Core

Dans `Program.cs`, veillons à bien respecter l'ordre de chargement de nos middlewares

```
app.UseHttpsRedirection();
app.UseCors("CorsPolicy");
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
```

ASP.NET Core

Remarque 1

Avant de tester, décorons le contrôleur `PersonnesController` par l'attribut `[Authorize]` pour exigé que l'utilisateur soit authentifié et autorisé pour y accéder.

ASP.NET Core

Remarque 1

Avant de tester, décorons le contrôleur `PersonnesController` par l'attribut `[Authorize]` pour exigé que l'utilisateur soit authentifié et autorisé pour y accéder.

Remarque 2

Vérifiez que toutes les actions du contrôleur `PersonnesController` ne sont plus accessibles : **401 Unauthorized**.

ASP.NET Core

Pour obtenir le jeton avec **Postman**

- Dans la liste déroulante, choisir `POST` puis saisir l'url vers le web service `http://localhost:<port>/api/Users`
- Dans `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Ensuite cliquer sur `Body`, cocher `raw`, choisir `JSON (application/json)` et saisir des données sous format `JSON` correspondant à l'objet personne à ajouter

```
{
    "Username": "Wick",
    "Password": "John",
}
```

- Cliquer sur `Send` puis copier le jeton

ASP.NET Core

Pour obtenir la liste des personnes avec **Postman**

- Dans la liste déroulante, choisir `GET` puis saisir l'url vers notre web service `http://localhost:<port>/api/personnes`
- Dans `Headers`, saisir `Content-Type` comme `Key` et `application/json` comme `Value`
- Dans `Authorization`, cliquer sur `Type`, choisir `Bearer Token` et coller le token
- Cliquer sur `Send`

ASP.NET Core

Pour ajouter des données concernant l'utilisateur dans le jeton, on utilise la section `claims`

```
public string GenerateJWT(User user)
{
  var key = _configuration["Jwt:Key"];
  var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
  var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
  Claim userClaim = new Claim("id", user.Id.ToString());
  Claim roleClaim = new Claim("role", user.Role);
  var token = new JwtSecurityToken(
    claims: new Claim[] { userClaim },
    signingCredentials: credentials
  );
  return new JwtSecurityTokenHandler().WriteToken(token);
}
```

ASP.NET Core

Pour tester

- Utiliser **Postman** pour vous connecter avec les identifiants suivants

```
{
    "Username": "Wick",
    "Password": "John",
}
```

- Copier le jeton
- Aller à `http://calebb.net/`
- Coller le jeton et vérifier la présence de `"Id": 1` dans la deuxième partie.

ASP.NET Core

Pour renforcer la sécurité de notre jeton, utilisons `issuer` et `audience` pour les inclure dans la génération et la validation du token (contenu de `appsettings.json`)

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Jwt": {
    "Key": "UtiliseUneClePlusSecurisee",
    "Issuer": "https://localhost:7288",
    "Audience": "http://localhost:4200"
  }
}
```

ASP.NET Core

Incluons `issuer` et `audience` dans la génération du token

```
public string GenerateJWT(User user)
{
  var key = _configuration["Jwt:Key"];
  var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
  var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
  Claim userClaim = new Claim("id", user.Id.ToString());
  Claim roleClaim = new Claim("role", user.Role);
  var token = new JwtSecurityToken(
    issuer: _configuration["Jwt:Issuer"],
    audience: _configuration["Jwt:Audience"],
    claims: new Claim[] { userClaim, roleClaim },
    signingCredentials: credentials
  );
  return new JwtSecurityTokenHandler().WriteToken(token);
}
```

ASP.NET Core

Dans `Program.cs`, modifions les paramètres de validation de tokens

```
var key = Encoding.ASCII.GetBytes(builder.Configuration["Jwt:Key"]);

var tokenParams = new TokenValidationParameters()
{
    ValidIssuer = builder.Configuration["Jwt:Issuer"],
    ValidAudience = builder.Configuration["Jwt:Audience"],
    ValidateIssuer = true,
    ValidateAudience = true,
    ValidateLifetime = false,
    ValidateIssuerSigningKey = true,
    IssuerSigningKey = new SymmetricSecurityKey(key),
};
```

ASP.NET Core

Remarque

Initialement, **Swagger** ne permet pas l'envoi de jetons.

ASP.NET Core

Remarque

Initialement, **Swagger** ne permet pas l'envoi de jetons.

Solution

Le reconfigurer

ASP.NET Core

Dans `Program.cs`, remplaçons la ligne suivante

```
builder.Services.AddSwaggerGen();
```

ASP.NET Core

Par le code suivant

```
builder.Services.AddSwaggerGen(swagger =>
{
    var jwtSecurityScheme = new OpenApiSecurityScheme
    {
        Scheme = "bearer",
        BearerFormat = "JWT",
        Name = "JWT Authentication",
        In = ParameterLocation.Header,
        Type = SecuritySchemeType.Http,
        Description = "Coller ici votre jeton",
        Reference = new OpenApiReference
        {
            Id = JwtBearerDefaults.AuthenticationScheme,
            Type = ReferenceType.SecurityScheme
        }
    };

    swagger.AddSecurityDefinition(jwtSecurityScheme.Reference.Id, jwtSecurityScheme);

    swagger.AddSecurityRequirement(new OpenApiSecurityRequirement
    {
        { jwtSecurityScheme, Array.Empty<string>() }
    });
});
```

ASP.NET Core

Relancez le projet et vérifiez la présence de la nouvelle section **Authorization**.

ASP.NET Core

Rôle

- Propriété qui permet d'autoriser ou d'interdire, un utilisateur connecté, à une ressource.
- Peut être spécifié dans le décorateur `[Authorize]`.

ASP.NET Core

Dans `Models`, commençons par définir une classe statique `Role`

```
namespace CoursWebApi.Models
{
    public static class Role
    {
        public const string Admin = "Admin";
        public const string User = "User";
    }
}
```

ASP.NET Core

Dans `User`, ajoutons une propriété `Role`

```
using System.Text.Json.Serialization;

namespace CoursWebApi.Models
{
    public class User
    {
        public int? Id { get; set; }
        public string Username { get; set; }
        [JsonIgnore]
        public string Password { get; set; }
        public string? Nom { get; set; }
        public string? Prenom { get; set; }
        public string? Email { get; set; }
        public string Role { get; set; }
    }
}
```

ASP.NET Core

Dans `UserService`, ajoutons deux rôles différents à nos deux utilisateurs

```
namespace CoursWebApi.Services
{
    public class UserService : IUserService
    {
        private IConfiguration _configuration = null;
        public List<User> Users { get; set; }
        public UserService(IConfiguration configuration)
        {
            _configuration = configuration;
            Users = new List<User>() {
                new User() { Id = 1, Nom = "Wick", Prenom = "John", Username = "Wick",
                  Email = "w@w.fr", Password = "John", Role = Role.Admin },
                new User() { Id = 2, Nom = "Dalton", Prenom = "Jack", Username = "
                  Dalton", Email = "d@d.fr", Password = "Jack",Role = Role.User }
            };
        }

        // + le contenu précédent
    }
}
```

ASP.NET Core

Il faut intégrer les rôles dans le token

```
public string GenerateJWT(User user)
{
  var key = _configuration["Jwt:Key"];
  var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
  var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
  Claim userClaim = new Claim("id", user.Id.ToString());
  Claim roleClaim = new Claim("role", user.Role);
  var token = new JwtSecurityToken(
    issuer: _configuration["Jwt:Issuer"],
    audience: _configuration["Jwt:Audience"],
    claims: new Claim[] { userClaim, roleClaim },
    signingCredentials: credentials
  );
  return new JwtSecurityTokenHandler().WriteToken(token);
}
```

ASP.NET Core

Pour tester

- Utiliser **Postman** pour vous connecter avec les identifiants suivants

```
{
    "Username": "Wick",
    "Password": "John",
}
```

- Copier le jeton
- Aller à `http://calebb.net/`
- Coller le jeton et vérifier la présence de `"role": Admin` dans la deuxième partie.

ASP.NET Core

Dans `PersonnesController`, ajoutons les décorateurs `[Authorize]`) aux deux actions `Get`

```
[Authorize(Roles = Role.Admin)]
[HttpGet]
public IEnumerable<Personne> Get()
{
    return personneService.GetAll();
}

[Authorize(Roles = Role.User)]
[HttpGet("{id}.{format?}")]
public IActionResult Get(int id)
{
    var personne = personneService.GetOneById(id);
    if (personne == null)
    {
        return NotFound();
    }
    return Ok(personne);
}
```

ASP.NET Core

Pour tester

- Connectez-vous avec `(Wick, John)` et vérifiez
  - que vous pouvez accéder à la route `/api/personnes`
  - qu'un message `403 Forbidden` s'affiche lorsque vous essayez d'accéder à la route `/api/personnes/1`
- Connectez-vous avec `(Dalton, Jack)` et vérifiez
  - que vous pouvez accéder à la route `/api/personnes/1`
  - qu'un message `403 Forbidden` s'affiche lorsque vous essayez d'accéder à la route `/api/personnes`

ASP.NET Core

Remarque

Pour hacher le mot de passe, on peut utiliser `Bcrypt`.

ASP.NET Core

Remarque

Pour hacher le mot de passe, on peut utiliser `Bcrypt`.

Assembly à installer

`Bcrypt.Net-core` (version 1.6.0)

ASP.NET Core

Dans `UserService`, hechons les mots de passe dans le constructeur et utilisons la méthode `Verify` pour comparer les mots de passe saisis avec les mots de passe hachés

```
namespace CoursWebApi.Services
{
    public class UserService : IUserService
    {
        private IConfiguration _configuration = null;
        public List<User> Users { get; set; }
        public UserService(IConfiguration configuration)
        {
            _configuration = configuration;
            Users = new List<User>() {
                new User() { Id = 1, Nom = "Wick", Prenom = "John", Username = "Wick",
                  Email = "w@w.fr", Password = BCrypt.Net.BCrypt.HashPassword("John"),
                  Role = Role.Admin },
                new User() { Id = 2, Nom = "Dalton", Prenom = "Jack", Username = "
                  Dalton", Email = "d@d.fr", Password  = BCrypt.Net.BCrypt.HashPassword
                  ("Jack"),Role = Role.User }
            };
        }
        public User? CheckUser(ConnectionData data)
        {
            return Users.Find(elt => elt.Username == data.Username && BCrypt.Net.BCrypt
              .Verify(data.Password, elt.Password));
        }
        // + le contenu précédent
    }
}
```