

# TP N°6

Sécuriser une API avec JWT

**Enseignant :** Malek Zribi

1

# OBJECTIFS DU TP

- Gestion des utilisateurs et authentification dans une API rest avec ASP.NET Core Identity
- Génération de Token pour les utilisateurs et protection des Endpoint.
- Gestion des autorisations basée sur les rôles.

# JWT

## ▪ Qu'est-ce que le JWT ?

JSON Web Token (JWT) est une norme ouverte ( RFC 7519 ) qui définit un moyen compact et autonome de transmettre en toute sécurité des informations entre les parties en tant qu'objet JSON. Ces informations peuvent être vérifiées et fiables car elles sont signées numériquement. Les JWT peuvent être signés à l'aide d'un secret (avec l' algorithme HMAC ) ou d'une paire de clés publique/privée à l'aide de RSA ou ECDSA .

## ▪ Quelle est la structure du jeton Web JSON ?

Dans leur forme compacte, les jetons Web JSON se composent de trois parties séparées par des points qui sont :

- Entête (Header)
- Charge utile (Payload)
- Signature

Par conséquent, un JWT ressemble généralement à ce qui suit : xxxxx.yyyyyy.zzzzz

# JWT

Algorithm HS256 ▼

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

# JWT

- **Entête (Header):** L'en-tête se compose généralement de deux parties : le type du jeton, qui est JWT, et l'algorithme de signature utilisé, tel que HMAC SHA256 ou RSA.

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

Ensuite, ce JSON est codé en Base64Url pour former la première partie du JWT.

- **Charge utile (Payload):** La deuxième partie du jeton est la charge utile, qui contient les revendications. Les revendications sont des déclarations sur une entité (généralement l'utilisateur) et des données supplémentaires. Certains d'entre eux sont : iss (émetteur), exp (délai d'expiration), sub (sujet), aud (audience) et autres .

Un exemple de charge utile pourrait être :

La charge utile est ensuite codée en Base64Url pour former la deuxième partie du jeton Web JSON.

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

# JWT

- **Signature** : Pour créer la partie signature, vous devez prendre l'en-tête codé, la charge utile codée, un secret, l'algorithme spécifié dans l'en-tête, et le signer. Par exemple si vous souhaitez utiliser l'algorithme HMAC SHA256, la signature sera créée de la manière suivante :

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

La signature est utilisée pour vérifier que le message n'a pas été modifié en cours de route et, dans le cas de jetons signés avec une clé privée, elle peut également vérifier que l'expéditeur du JWT est bien celui qu'il prétend être.

# CRÉATION DU PROJET

- Créer un nouveau projet WEB API nommé SecureAPI\_JWT.
- Installer les packages Nuget suivants :

Install-Package Microsoft.AspNetCore.Authentication.JwtBearer  
Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore  
Install-Package Microsoft.EntityFrameworkCore  
Install-Package Microsoft.EntityFrameworkCore.Design  
Install-Package Microsoft.EntityFrameworkCore.SqlServer  
Install-Package Microsoft.EntityFrameworkCore.Tools  
Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design  
Install-Package System.IdentityModel.Tokens.Jwt

# CONFIGURATION DES DONNÉES JWT

- Ajouter dans le fichier appsettings.json la clé suivante contenant les informations JWT:

```
{,
  "JWT": {
    "Key": "sz8eI70dHBrjrIo8j9nTW/rQy010vY0pAQ2wDKQZw/0=",
    "Issuer": "SecureApi",
    "Audience": "SecureApiUser",
    "DurationInDays": 30
  }
}
```

Puis ajouter la classe suivante dans un dossier Helpers:

```
namespace SecureAPI_JWT.Helpers
{
    public class JWT
    {
        public string Key { get; set; }
        public string Issuer { get; set; }
        public string Audience { get; set; }
        public double DurationInDays { get; set; }
    }
}
```



# CLASSE APPLICATIONUSER

- Inscrire le service suivant dans program.cs pour mapper la classe JWT avec le paramètre de configuration « JWT ».

```
builder.Services.Configure<JWT>(builder.Configuration.GetSection("JWT"));
```

- Ajouter la classe ApplicationUser dans le dossier Models:

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;

namespace SecureAPI_JWT.Models
{
    public class ApplicationUser : IdentityUser
    {
        [Required, MaxLength(50)]
        public string FirstName { get; set; }

        [Required, MaxLength(50)]
        public string LastName { get; set; }
    }
}
```

# CLASSE APPLICATIONDBCONTEXT

- Ajouter la classe ApplicationDbContext dans le dossier Models:

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace SecureAPI_JWT.Models
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {
        }
    }
}
```

# CONNECTIONSTRING

- Dans Appsettings.json ajouter la chaîne de connexion de la base de données sql server

<https://json.schemastore.org/appsettings.json>

```
1  {
2  "ConnectionStrings": {
3    "DefaultConnection": "Data Source=(localdb)\\ProjectsV13;Initial Catalog=TestApiJWT;Integrated Security=True"
4  },
5  "Logging": {
6    "LogLevel": {
7      "Default": "Information",
8      "Microsoft": "Warning",
9      "Microsoft.Extensions.Logging.ApplicationInsights": "Information"
10   },
11  },
12  "AllowedHosts": "*",
13  "JWT": {
14    "Key": "sz8eI70dHBrjrIo8j9nTW/rQy010vY0pAQ2wDKQZw/0=",
15    "Issuer": "SecureApi",
16    "Audience": "SecureApiUser",
17    "DurationInDays": 30
18  }
19 }
```

Microsoft Log level threshold.  
Type : string

# INSCRIPTION DES SERVICES ET MIGRATION

- Inscrire dans program.cs le service de context et le service Identity.

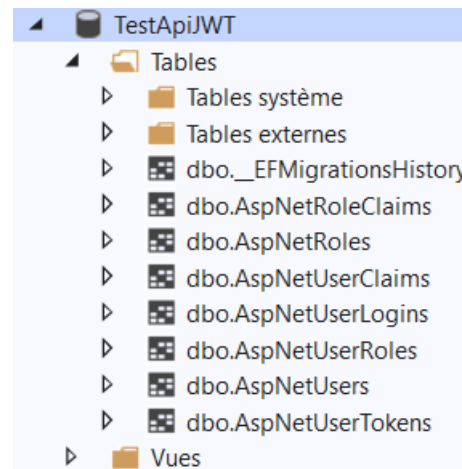
```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>().AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"))
);
```

- Puis lancer une Migration Entity Framework sous package manager console

PM> add-migration initialcreate

PM> update-database



# COUCHE DE SERVICE D'AUTHENTIFICATION

- Ajouter un dossier Services.
- Créer l'interface IAuthService et sa classe d'implémentation AuthService
- Inscrire le service dans program.cs



```
builder.Services.AddScoped<IAuthService, AuthService>();
```

- Ajouter le middleware app.UseAuthentication juste avant app.UseAuthorisation

```
app.UseAuthentication();
```

```
app.UseAuthorization();
```

# INSCRIPTION DU SERVICE D'AUTHENTIFICATION

- Ajouter le service d'authentification JWT dans program.cs

```
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})

    .AddJwtBearer(o =>
    {
        o.RequireHttpsMetadata = false;
        o.SaveToken = false;
        o.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidIssuer = builder.Configuration["JWT:Issuer"],
            ValidAudience = builder.Configuration["JWT:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWT:Key"]))
        };
    });
```

# LES MODELES D'AUTHENTIFICATION

- Dans le dossier Models, Ajouter les classes suivantes :

```
namespace SecureAPI_JWT.Models
{
    public class AuthModel
    {
        public string Message { get; set; }
        public bool IsAuthenticated { get; set; }
        public string Username { get; set; }
        public string Email { get; set; }
        public List<string> Roles { get; set; }
        public string Token { get; set; }
        public DateTime ExpiresOn { get; set; }
    }
}
```

```
using System.ComponentModel.DataAnnotations;
```

```
namespace SecureAPI_JWT.Models
{
    public class RegisterModel
    {
        [StringLength(100)]
        public string FirstName { get; set; }

        [StringLength(100)]
        public string LastName { get; set; }

        [StringLength(50)]
        public string Username { get; set; }

        [StringLength(128)]
        public string Email { get; set; }

        [StringLength(256)]
        public string Password { get; set; }
    }
}
```

# IMPLÉMENTATION DU SERVICE D'AUTHENTIFICATION

- Dans l'interface IAuthService Ajouter la méthode RegisterAsync :

```
using SecureAPI_JWT.Models;

namespace SecureAPI_JWT.Services
{
    public interface IAuthService
    {
        Task<AuthModel> RegisterAsync(RegisterModel model);
    }
}
```



# IMPLÉMENTATION DU SERVICE D'AUTHENTIFICATION

- Implémenter la méthode RegisterAsync :

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using SecureAPI_JWT.Helpers;
using SecureAPI_JWT.Models;

namespace SecureAPI_JWT.Services
{
    public class AuthService : IAuthService
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly JWT _jwt;

        public AuthService(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager, IOptions<JWT> jwt)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _jwt = jwt.Value;
        }
    }
}
```

# IMPLÉMENTATION DU SERVICE D'AUTHENTIFICATION

- Implémenter la méthode RegisterAsync :

```
public async Task<AuthModel> RegisterAsync(RegisterModel model)
{
    if (await _userManager.FindByEmailAsync(model.Email) is not null)
        return new AuthModel { Message = "Email is already registered!" };

    if (await _userManager.FindByNameAsync(model.Username) is not null)
        return new AuthModel { Message = "Username is already registered!" };

    var user = new ApplicationUser
    {
        UserName = model.Username,
        Email = model.Email,
        FirstName = model.FirstName,
        LastName = model.LastName
    };

    var result = await _userManager.CreateAsync(user, model.Password);

    if (!result.Succeeded)
    {
        var errors = string.Empty;

        foreach (var error in result.Errors)
            errors += $"{error.Description},";
    }
}
```

# IMPLÉMENTATION DU SERVICE D'AUTHENTIFICATION

```
        return new AuthModel { Message = errors };
    }

    await _userManager.AddToRoleAsync(user, "User");

    var jwtSecurityToken = await CreateJwtToken(user);

    return new AuthModel
    {
        Email = user.Email,
        ExpiresOn = jwtSecurityToken.ValidTo,
        IsAuthenticated = true,
        Roles = new List<string> { "User" },
        Token = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken),
        Username = user.UserName
    };
}
```

# IMPLÉMENTATION DU SERVICE D'AUTHENTIFICATION

- Implémenter la méthode `CreateJwtToken()` :

```
private async Task<JwtSecurityToken> CreateJwtToken(ApplicationUser user)
{
    var userClaims = await _userManager.GetClaimsAsync(user);
    var roles = await _userManager.GetRolesAsync(user);
    var roleClaims = new List<Claim>();

    foreach (var role in roles)
        roleClaims.Add(new Claim("roles", role));

    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.UserName),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new Claim(JwtRegisteredClaimNames.Email, user.Email),
        new Claim("uid", user.Id)
    }
    .Union(userClaims)
    .Union(roleClaims);

    var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwt.Key));
    var signingCredentials = new SigningCredentials(symmetricSecurityKey, SecurityAlgorithms.HmacSha256);

    var jwtSecurityToken = new JwtSecurityToken(
        issuer: _jwt.Issuer,
        audience: _jwt.Audience,
        claims: claims,
        expires: DateTime.Now.AddDays(_jwt.DurationInDays),
        signingCredentials: signingCredentials);

    return jwtSecurityToken;
}
```

# CREATION D'UN CONTRÔLEUR D'AUTHENTIFICATION

```
namespace SecureAPI_JWT.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthService _authService;

        public AuthController(IAuthService authService)
        {
            _authService = authService;
        }

        [HttpPost("register")]
        public async Task<IActionResult> RegisterAsync([FromBody] RegisterModel model)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);

            var result = await _authService.RegisterAsync(model);

            if (!result.IsAuthenticated)
                return BadRequest(result.Message);

            return Ok(result);
        }
    }
}
```

# TESTER LA MÉTHODE REGISTER

## Auth

POST /api/Auth/register

### Parameters

Cancel

Reset

No parameters

Request body

application/json

```
{  
  "firstName": "Ali",  
  "lastName": "triki",  
  "username": "userali",  
  "email": "ali@gmail.com",  
  "password": "Ali_triki2023"  
}
```

Execute

Clear

# TESTER LA MÉTHODE REGISTER

## Curl


```
curl -X 'POST' \
  'https://localhost:7005/api/Auth/register' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "firstName": "Ali",
    "lastName": "triki",
    "username": "userali",
    "email": "ali@gmail.com",
    "password": "Ali_triki2023"
  }'
```



## Request URL

**https://localhost:7005/api/Auth/register**

### Server response

Code	Details
200	<div><div>Response body</div><div><pre>{  "message": null,  "isAuthenticated": true,  "username": "userali",  "email": "ali@gmail.com",  "roles": [    "User"  ],  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIj1c2VyYWxpIiwianRpIjoibG1MTE5NjEtNWVmMi00OD11LWlwZjAtYTczZmM5MmMwYTQ5IiwiaW1haWwiOiJhbG1AZ21haWwY29tIiwidWlkIjoibG1MTE5NjEtNWVmMi00OD11LWlwZjAtYTczZmM5MmMwYTQ5IiwiaWF0Ij01Nj00OTYtOGQzZS00OD1zLWJjOGYtNTA4Y2EwNGQ4N2QwIiwicm9sZXMiOiIj1c2VyIiwiaXNjaXNzAGNDM5MjI3LCJpc3MiOiIjTZW1cmVhcGkiLCJhdWQiOiIjTZW1cmVhcG1Vc2VyIn0.gCwdujY0Xb2F-XVSAu-kuTZPTnMDYMHQp7rI1_u4XRo",  "expiresOn": "2024-01-05T07:20:27Z"}</pre></div><div><div> Download</div></div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Wed, 06 Dec 2023 07:20:27 GMT server: Kestrel</pre></div></div>

# IMPLÉMENTATION USER LOGIN

- Dans le dossier Models ajouter la classe TokenRequestModel qui représente le modèle de Login.
- Dans l'interface de service IAuthService Ajouter la signature de la méthode GetTokenAsync.

```
namespace SecureAPI_JWT.Models
{
    public class TokenRequestModel
    {
        public string Email { get; set; }

        public string Password { get; set; }
    }
}
```

```
using SecureAPI_JWT.Models;

namespace SecureAPI_JWT.Services
{
    public interface IAuthService
    {
        Task<AuthModel> RegisterAsync(RegisterModel model);

        Task<AuthModel> GetTokenAsync(TokenRequestModel model);
    }
}
```



# IMPLÉMENTATION USER LOGIN

- Ajouter l'implémentation de la méthode GetTokenAsync dans la classe d'implémentation AuthService

```
public async Task<AuthModel> GetTokenAsync(TokenRequestModel model)
{
    var authModel = new AuthModel();

    var user = await _userManager.FindByEmailAsync(model.Email);

    if (user is null || !await _userManager.CheckPasswordAsync(user, model.Password))
    {
        authModel.Message = "Email or Password is incorrect!";
        return authModel;
    }

    var jwtSecurityToken = await CreateJwtToken(user);
    var rolesList = await _userManager.GetRolesAsync(user);

    authModel.IsAuthenticated = true;
    authModel.Token = new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken);
    authModel.Email = user.Email;
    authModel.Username = user.UserName;
    authModel.ExpiresOn = jwtSecurityToken.ValidTo;
    authModel.Roles = rolesList.ToList();

    return authModel;
}
```

# IMPLÉMENTATION USER LOGIN

- Dans AuthController Ajouter la méthode suivante pour la fonctionnalité Login :

```
[HttpPost("token")]
public async Task<IActionResult> GetTokenAsync([FromBody] TokenRequestModel model)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var result = await _authService.GetTokenAsync(model);

    if (!result.IsAuthenticated)
        return BadRequest(result.Message);

    return Ok(result);
}
```

# TESTER LOGIN

POST /api/Auth/token

Parameters

Cancel

Reset

No parameters

Request body

application/json

```
{  
  "email": "ali@gmail.com",  
  "password": "Ali_triki2023"  
}
```

Execute

Clear

# TESTER LOGIN

## Curl

```
curl -X 'POST' \
  'https://localhost:7005/api/Auth/token' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "email": "ali@gmail.com",
    "password": "Ali_triki2023"
  }'
```



## Request URL

**https://localhost:7005/api/Auth/token**

### Server response


## Code

### Details

200

## Response body

```
{
  "message": null,
  "isAuthenticated": true,
  "username": "userali",
  "email": "ali@gmail.com",
  "roles": [
    "user"
  ],
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyYWxpIiwianRpIjoIIm2JhZGQxNmMtYmRlNS00ZTEyLTgyNmUtNjE1YzI0ZDk5IiwiaWF0IjoiJHbG1AZ21haWwY29tIiwidWlkIjoIINTZjYjM0OTYtOGQzZS00OjEzLWJjOGYtNTA4Y2EwNGQ4N2QwIiwicm9sZXMiOiJ1c2VyIiwiaXhwaXoXNzA0NDQzMtM3L3Cjpc3MiOiJ1c2VWN1cmVBCGkiLCJhdWQiOiJ1c2VWN1cmVBCG1Vc2VYIn0.5L15WnH9HjDT9ND1r7xwDoF_QJRr4MHwBTudjSw1HhQ",
  "expiresOn": "2024-01-05T08:26:37Z"
}
```


[Download](#)



Download

## Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 06 Dec 2023 08:26:36 GMT
server: Kestrel
```

# AJOUTER DES ROLES AUX UTILISATEURS

- Ajouter la classe AddRoleModel suivante au dossier Models
- Puis ajouter la méthode AddRoleAsync à l'interface IAuthService

```
namespace SecureAPI_JWT.Models
{
    public class AddRoleModel
    {
        public string UserId { get; set; }

        public string Role { get; set; }
    }
}
```

```
using SecureAPI_JWT.Models;

namespace SecureAPI_JWT.Services
{
    public interface IAuthService
    {
        Task<AuthModel> RegisterAsync(RegisterModel model);

        Task<AuthModel> GetTokenAsync(TokenRequestModel model);

        Task<string> AddRoleAsync(AddRoleModel model);
    }
}
```

# AJOUTER DES ROLES AUX UTILISATEURS

- Ajouter l'implémentation de la méthode dans la classe AuthService

```
public async Task<string> AddRoleAsync(AddRoleModel model)
{
    var user = await _userManager.FindByIdAsync(model.UserId);

    if (user is null || !await _roleManager.RoleExistsAsync(model.Role))
        return "Invalid user ID or Role";

    if (await _userManager.IsInRoleAsync(user, model.Role))
        return "User already assigned to this role";

    var result = await _userManager.AddToRoleAsync(user, model.Role);

    return result.Succeeded ? string.Empty : "Something went wrong";
}
```

# AJOUTER DES ROLES AUX UTILISATEURS

- Dans AuthController ajouter la méthode AddRoleAsync :

```
[HttpPost("addrole")]
public async Task<IActionResult> AddRoleAsync([FromBody] AddRoleModel model)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    var result = await _authService.AddRoleAsync(model);

    if (!string.IsNullOrEmpty(result))
        return BadRequest(result);

    return Ok(model);
}
```

# TEST AVEC SWAGGER

POST /api/Auth/addrole



Parameters

Cancel

Reset

No parameters

Request body

application/json



```
{
  "userId": "abf212b9-33c4-485e-8015-4d880c9a02f5",
  "role": "admin"
}
```



# TEST AVEC SWACGER

## Curl

```
curl -X 'POST' \
  'https://localhost:7005/api/Auth/addrole' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "userId": "abf212b9-33c4-485e-8015-4d880c9a02f5",
    "role": "admin"
  }'
```

## Request URL

`https://localhost:7005/api/Auth/addrole`

## Server response

### Code

### Details

200

### Response body

```
{
  "userId": "abf212b9-33c4-485e-8015-4d880c9a02f5",
  "role": "admin"
}
```

### Response headers

```
content-type: application/json; charset=utf-8
date: Wed, 06 Dec 2023 14:58:35 GMT
server: Kestrel
```

## Responses

Code

Description

# TESTER LES AUTORISATIONS

- Tester avec WeatherForecastController, en ajoutant l'attribut Authorize sans rôle puis avec le rôle admin

```
[Authorize(Roles = "admin")]
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
    private static readonly string[] Summaries = new[]
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "!"
    };

    private readonly ILogger<WeatherForecastController> _logger;

    public WeatherForecastController(ILogger<WeatherForecastController> logger)
    {
    }
```