Security in ASP.NET API

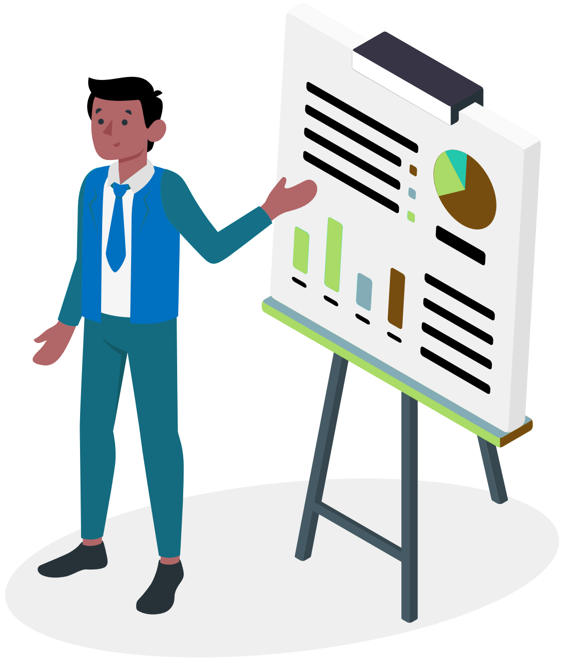
Agenda

1

- Introduction JWT

2

- Introduction Identity in ASP.NET API



Lesson Objectives

- ❖ *Authentication vs Authorization*
- ❖ *Traditional vs Token based authentication*
- ❖ *Entity Framework Core*
  - ➢ *Create tables, store user related data, get user related data*
- ❖ *Generating Access Tokens*

Section 1

Introduction JWT

Authentication vs Authorization

- **Authentication** and **Authorization** are crucial aspects of building secure ASP.NET APIs.
- Authentication verifies the **identity** of clients accessing the API.
- Authorization controls access to API resources based on **user roles and permissions**.
- Authentication ensures that only authenticated users can access the API.
- Common authentication mechanisms in ASP.NET API:
    - *Token-based authentication (JWT): Securely transmitting and verifying JSON Web Tokens.*
    - *OAuth 2.0: Delegating user authentication to trusted identity providers.*
    - *IdentityServer: Implementing OpenID Connect and OAuth 2.0 protocols.*

Authentication vs Authorization

- Authorization determines what actions users can perform within the API.
- **Role-based authorization**: Assigning users to predefined roles and granting permissions based on roles.
- **Claims-based authorization**: Assigning users specific claims and granting access based on those claims.
- **Attribute-based authorization**: Applying authorization rules directly to API endpoints using attributes.

Traditional Authenticate – Token based

The picture can't be displayed.

Why use Token-Based authentication

- Scalability
- Multiple device
- The Signature is used to verify the integrity of the token and ensure that it has not been tampered with.
- The Signature is created by combining the encoded Header, encoded Payload, and a secret key or private key.
- The Signature is typically generated using a cryptographic algorithm specified in the Header ("alg" claim).
- Verification of the Signature ensures that the token has not been modified or tampered with.

Json Web Token

- JWT: an open standard(RFC 7519) that defines a compact and self-container way for securely transmitting information between parties as a json object.
- Structure of a JWT:
  - Header: {"alg": "HS256", "typ": "JWT"}
  - Payload: {"sub": "1234567890", "name": "John Doe", "admin": true, "exp": 1678934400}
  - Signature: Generated using the Header, Payload, and a secret key
- Example JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ik
pvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6y
JV_adQssw5c
```

Lesson Summary

- ❖*Authentication and Authorization*
- ❖*Traditional vs Token based Authentication*
- ❖*Json Web Token*

Section 2

Introduction Identity in ASP.NET API

Introduction Identity

- Identity is a feature in ASP.NET API Core that provides a robust framework for managing user authentication and authorization.
- Identity allows developers to easily integrate user management functionality into their applications.
- With Identity, you can handle user registration, login, password management, and role-based authorization.

Key Features of Identity in ASP.NET API

- **User Registration**: Provides APIs for creating new user accounts.
- **User Login**: Supports authentication using various methods such as passwords, social logins, or multi-factor authentication.
- **User Management**: Allows administrators to manage user accounts, including password resets, email confirmation, and account lockouts.
- **Role-Based Authorization**: Enables fine-grained access control by assigning roles to users and restricting access to certain resources based on roles.
- **Claims-Based Authorization**: Allows assigning custom claims to users and using those claims to control access to specific API endpoints.
- **Password Hashing**: Stores user passwords securely by hashing them using a strong cryptographic algorithm.

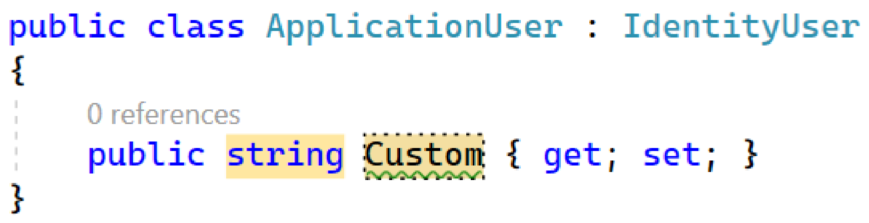
Integrating Identity into ASP.NET API

- Install the **Microsoft.AspNetCore.Identity.EntityFrameworkCore** NuGet package.
- Configure Identity services in the API startup class, including setting up the database context and configuring options.
- Customize the Identity models and data schema, if needed.
- Implement user registration, login, and password management endpoints using the Identity APIs.
- Apply role-based or claims-based authorization to API endpoints using attributes or middleware.

Adding Identity tables using EF

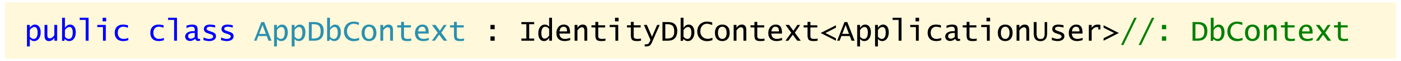
- Install nuget package: **Microsoft.AspNetCore.Identity.EntityFrameworkCore**

- Add class custom *IdentityUser*



```
public class ApplicationUser : IdentityUser
{
    0 references
    public string Custom { get; set; }
}
```

- Change class *AppDbContext* inheritance *IdentityDbContext*

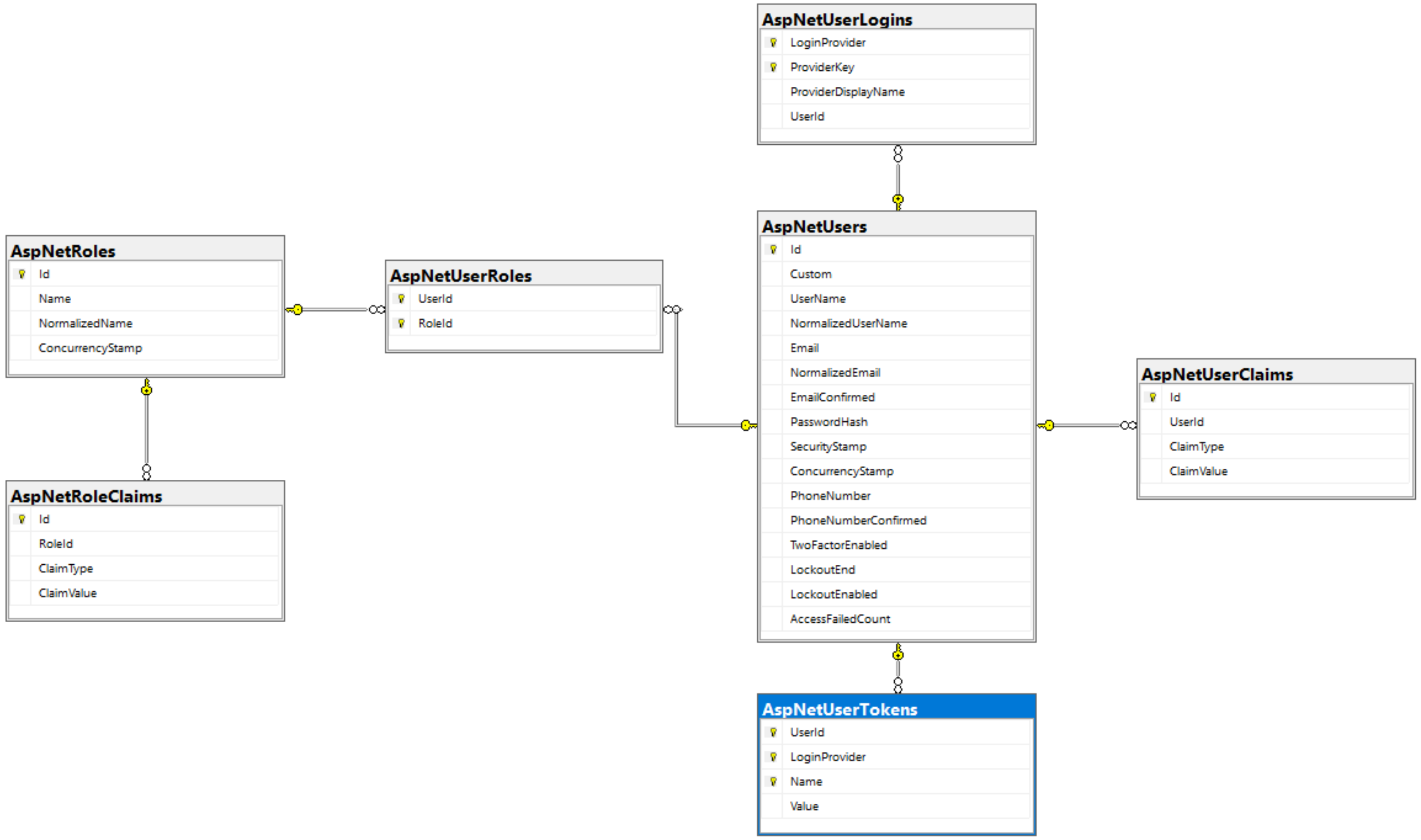


```
public class AppDbContext : IdentityDbContext<ApplicationUser>//: DbContext
```

- Update method *OnModelCreating add: base.OnModelCreating(modelBuilder);*

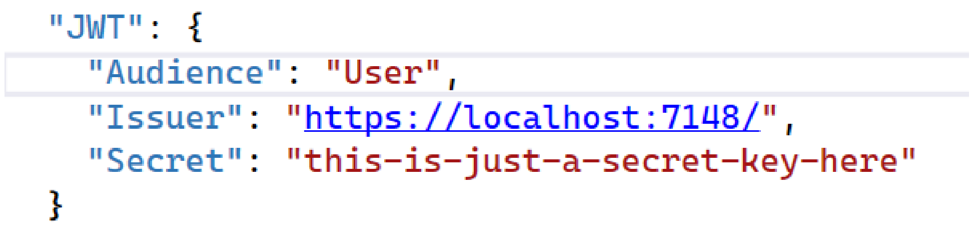
- Add Migration and update database

Diagram Identity Tables



Configuring JWT in service - 1

- Install Nuget package: **Microsoft.AspNetCore.Authentication.JwtBearer**
- Add key JWT in *appsettings.json* file:



```json
"JWT": {
  "Audience": "User",
  "Issuer": "https://localhost:7148/",
  "Secret": "this-is-just-a-secret-key-here"
}
```

- Add Identity Service:

```csharp
//Add Identity
builder.Services
    .AddIdentity<ApplicationUser, IdentityRole>()
    .AddEntityFrameworkStores<AppDbContext>()
    .AddDefaultTokenProviders();
```

Configuring JWT in service - 2

- Add authenticate and JwtBearer:

```
//Add Authenticate
builder.Services
    .AddAuthentication(config =>
    {
        config.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        config.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        config.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;

    })
    //add Jwtbearer
    .AddJwtBearer(options =>
    {
        options.SaveToken = true;
        options.RequireHttpsMetadata= false;
        options.TokenValidationParameters = new TokenValidationParameters()
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding
                                                        .ASCII
                                                        .GetBytes(builder.Configuration["JWT:Secret"]
                                                        .ToString())),
            ValidateIssuer = true,
            ValidIssuer = builder.Configuration["JWT:Issuer"],
            ValidateAudience= true,
            ValidAudience= builder.Configuration["JWT:Audience"],
        };
    });
```

Setting up Authentication Controller

- Add AuthenticationController and inject service:

```
[Route("api/[controller]")]
    [ApiController]
    public class AuthenticationController : ControllerBase
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly RoleManager<IdentityRole> _roleManager;
        private readonly AppDbContext _context;
        private readonly IConfiguration _configuration;

        public AuthenticationController(UserManager<ApplicationUser> userManager,
            RoleManager<IdentityRole> roleManager,
            AppDbContext context,
            IConfiguration configuration)
        {
            _userManager = userManager;
            _roleManager = roleManager;
            _context = context;
            _configuration = configuration;
        }
    }
```

Registering a new User - 1

- Add class **RegisterVm**:

```
public class RegisterVm
{
    [Required(ErrorMessage = "Username is required!")]
    public string UserName { get; set; }
    [Required(ErrorMessage = "Email is required!")]
    public string Email { get; set; }
    [Required(ErrorMessage = "Password is required!")]
    public string Password { get; set; }
}
```

Registering a new User - 2

- Add method **register post**:

```
[HttpPost("register-user")]
public async Task<IActionResult> Register([FromBody] RegisterVm registerVm)
{
    var userExists = await _userManager.FindByEmailAsync(registerVm.Email);

    if (userExists != null)
    {
        return BadRequest($"User {registerVm.Email} already exists!");
    }

    var newUser = new ApplicationUser()
    {
        UserName = registerVm.UserName,
        Email = registerVm.Email,
        Custom = registerVm.UserName,
        SecurityStamp = new Guid().ToString()
    };

    var result = await _userManager.CreateAsync(newUser, registerVm.Password);

    if (!result.Succeeded)
    {
        return BadRequest("User could not be create!");
    }

    return Created(nameof(Register), $"User {registerVm.Email} created!");
}
```

Add RefreshToken table to Db

- Add class **RefreshToken** and create relationship n-1 to **ApplicationUser**:

```
public class RefreshToken
   {
       public int Id { get; set; }

       public string UserId { get; set; }
       public string Token { get; set; }
       public string JwtId { get; set; }

       public bool IsRevoked { get; set; }
       public DateTime DateAdded { get; set; }
       public DateTime DateExpire { get; set; }

       [ForeignKey("UserId")]
       public ApplicationUser User { get; set; }
   }
```

- Add DbSet<RefreshToken> to AppDbContext

- Add-migration and update database

Generate JwtToken - 1

- Add class **AuthResultVM** and **LoginVM**:

```
public class AuthResultVM
  {
      public string Token { get; set; }
      public string RefreshToken { get; set; }
      public DateTime ExpiresAt { get; set; }
  }
```

```
public class LoginVM
  {
      [Required(ErrorMessage = "Email is required")]
      public string Email { get; set; }

      [Required(ErrorMessage = "Password is required")]
      public string Password { get; set; }
  }
```

Generate JwtToken - 2

- Add method **GenerateJwtToken** :

```
private async Task<AuthResultVM> GenerateJwtToken(ApplicationUser user)
{
    var authClaims = new List<Claim>()
    {
        new Claim(ClaimTypes.Name, user.UserName),
        new Claim(ClaimTypes.NameIdentifier, user.Id),
        new Claim(JwtRegisteredClaimNames.Email, user.Email),
        new Claim(JwtRegisteredClaimNames.Sub, user.Email),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };

    var authSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(_configuration["JWT:Secret"]));

    var token = new JwtSecurityToken(
        issuer: _configuration["JWT:Issuer"],
        audience: _configuration["JWT:Audience"],
        expires: DateTime.UtcNow.AddMinutes(10), // 5 - 10mins
        claims: authClaims,
        signingCredentials: new SigningCredentials(authSigningKey, SecurityAlgorithms.HmacSha256)
        );

    var jwtToken = new JwtSecurityTokenHandler().WriteToken(token);
```

Generate JwtToken - 3

```
var refreshToken = new RefreshToken()
            {
                JwtId = token.Id,
                IsRevoked = false,
                UserId = user.Id,
                DateAdded = DateTime.UtcNow,
                DateExpire = DateTime.UtcNow.AddMonths(6),
                Token = Guid.NewGuid().ToString() + "-" + Guid.NewGuid().ToString()
            };

            await _context.RefreshTokens.AddAsync(refreshToken);
            await _context.SaveChangesAsync();

            var response = new AuthResultVM()
            {
                Token = jwtToken,
                RefreshToken = refreshToken.Token,
                ExpiresAt = token.ValidTo
            };

            return response;
        }
```

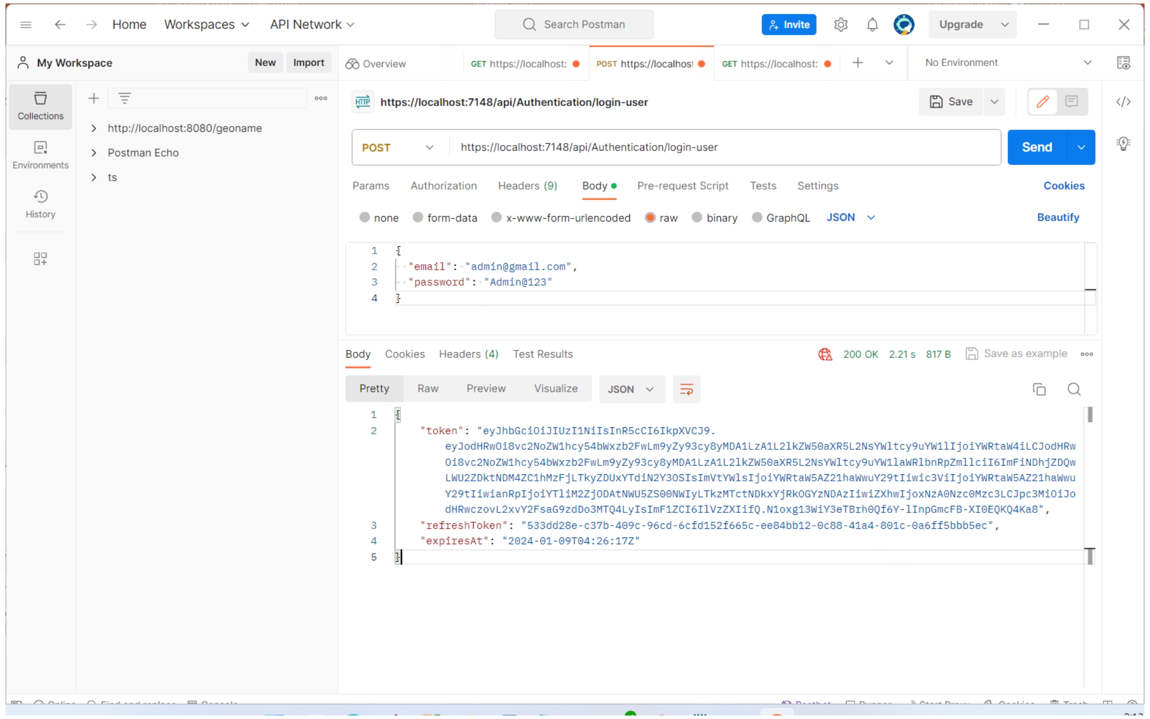
Logging in and Authorizing user

- Add method **Login [HttpPost]**

```
[HttpPost("login-user")]
public async Task<IActionResult> Login([FromBody] LoginVM payload)
{
    if (!ModelState.IsValid)
    {
        return BadRequest("Please, provide all required fields");
    }

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

    if (user != null && await _userManager.CheckPasswordAsync(user, payload.Password))
    {
        var tokenValue = await GenerateJwtToken(user);

        return Ok(tokenValue);
    }

    return Unauthorized();
}
```

- Add attribute `[Authorize]` to BooksController

Using Postman to test JWT

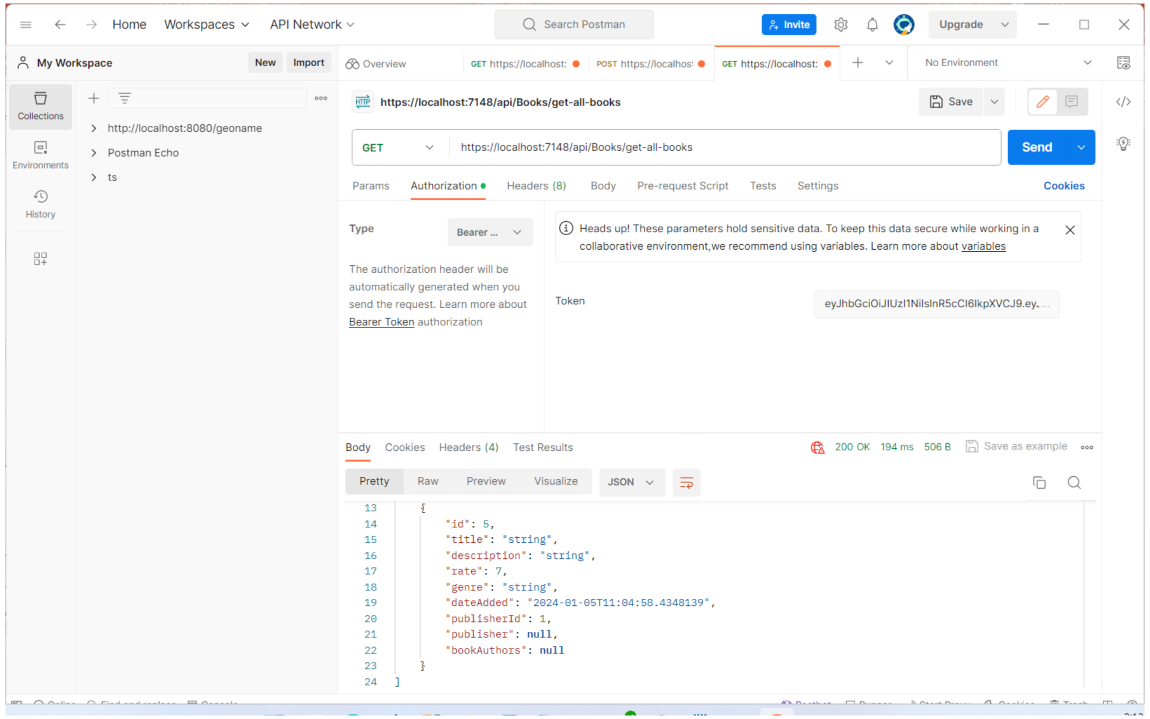


POST https://localhost:7148/api/Authentication/login-user

```
{
  "email": "admin@gmail.com",
  "password": "Admin@123"
}
```

Response: 200 OK 2.21 s 817 B

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiYWRtaW4iLCJodHRw
        Oi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6ImFiNDhjZDQw
        LWU2ZDktNDM4ZC1hMzFjLTkyZDUxYTdiN2Y3OSIsImVtYWlsIjoiYWRtaW5AZ21haWwuY29tIiwic3ViIjoiYWRtaW5AZ21haWwu
        Y29tIiwianRpIjoiYTliM2ZjODAtNWU5ZS00NWIyLTkzMTctNDkxYjRkOGYzNDAzIiwiZXhwIjoxNzA0Nzc0Mzc3LCJpc3MiOiJo
        dHRwczovL2xvY2FsaG9zdDo3MTQ4LyIsImF1ZCI6IlVzZXIifQ.N1oxg13WiY3eTBrh0Qf6Y-lInpGmcFB-XI0EQKQ4Ka8",
    "refreshToken": "533dd28e-c37b-409c-96cd-6cfd152f665c-ee84bb12-0c88-41a4-801c-0a6ff5bbb5ec",
    "expiresAt": "2024-01-09T04:26:17Z"
}
```



GET https://localhost:7148/api/Books/get-all-books

Authorization — Type: Bearer ...

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...

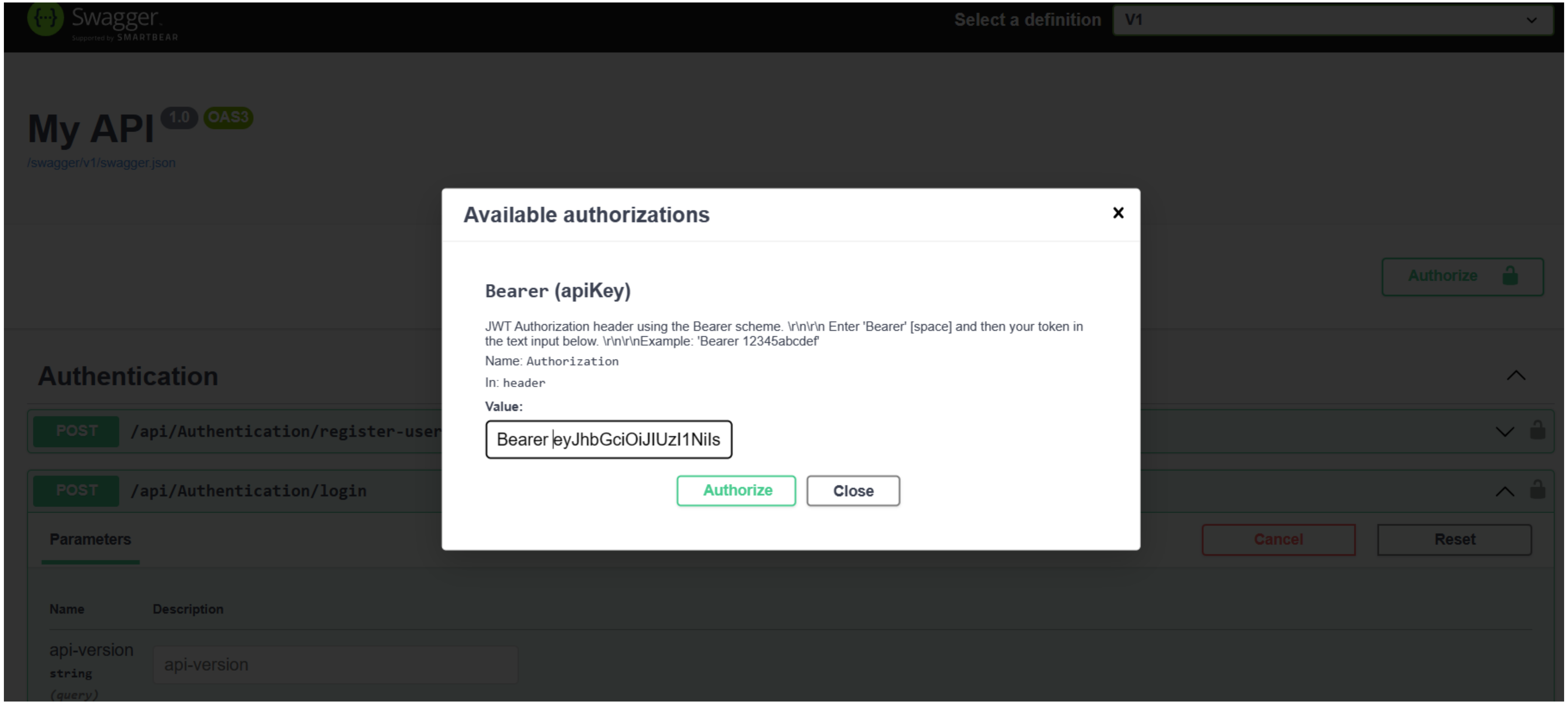
Response: 200 OK 194 ms 506 B

```
    {
        "id": 5,
        "title": "string",
        "description": "string",
        "rate": 7,
        "genre": "string",
        "dateAdded": "2024-01-05T11:04:58.4348139",
        "publisherId": 1,
        "publisher": null,
        "bookAuthors": null
    }
]
```

Config swagger to test authentication

```
builder.Services.AddSwaggerGen(c =>
{
    var provider = builder.Services.BuildServiceProvider().GetRequiredService<IApiVersionDescriptionProvider>();

    foreach (var description in provider.ApiVersionDescriptions)
    {
      c.SwaggerDoc(description.GroupName, new OpenApiInfo { Title = "My API", Version = description.ApiVersion.ToString()
});
    }

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Description = @"JWT Authorization header using the Bearer scheme. \r\n\r\n
                      Enter 'Bearer' [space] and then your token in the text input below.
                      \r\n\r\nExample: 'Bearer 12345abcdef'",
        Name = "Authorization",        BearerFormat = "JWT", In = ParameterLocation.Header,
        Type = SecuritySchemeType.ApiKey, Scheme = "Bearer"
    });

    c.AddSecurityRequirement(new OpenApiSecurityRequirement
                {                    {
                          new OpenApiSecurityScheme
                          {
                              Reference = new OpenApiReference
                              {
                                  Type = ReferenceType.SecurityScheme,
                                  Id = "Bearer"
                              }
                          },
                         new string[] {}
                    }
                });
});
```

Using Swagger to test JWT



Available authorizations

Bearer (apiKey)

JWT Authorization header using the Bearer scheme. \r\n\r\n Enter 'Bearer' [space] and then your token in the text input below. \r\n\r\nExample: 'Bearer 12345abcdef'

Name: `Authorization`

In: `header`

Value:

Bearer eyJhbGciOiJIUzI1NiIs

Authorize | Close

Lesson Summary

- *Using Entity Framework core to create tables to store the user related data*
- *Register User and Login*
- *Generating Access Tokens and authorize controller*

THANK YOU!