Building a Secure REST API with ASP.NET Core, Entity Framework Core, and JWT Authentication

Understanding Security & Authentication in ASP.NET Core

Before diving into the implementation, let's understand the key security concepts we'll be using in this project.

JWT (JSON Web Token) Authentication

JWT is a compact, URL-safe means of representing claims between two parties.

1. **Structure**: A JWT consists of three parts:
    - Header (algorithm & token type)
    - Payload (claims/data)
    - Signature (for verification)
2. **How it works**:
    - User provides credentials (username/password)
    - Server validates and generates a signed JWT
    - Client stores the token and sends it with subsequent requests
    - Server validates the token's signature and expiration
3. **Advantages**:
    - Stateless authentication
    - Can contain user data/roles
    - Cross-domain/CORS support
    - Scalable and efficient

ASP.NET Core Identity

While not used in this basic example, ASP.NET Core Identity is Microsoft's complete authentication and authorization system that provides:

1. **User Management**:
    - User registration and login
    - Password hashing and security
    - Account confirmation and recovery
    - Two-factor authentication
2. **Integration Capabilities**:
    - External login providers (Google, Facebook, etc.)
    - Custom user data storage
    - Customizable password policies
    - User lockout protection

Role-Based Access Control (RBAC)

RBAC is a security design that restricts system access based on roles:

1. **Core Concepts**:
    - Users are assigned roles
    - Roles contain permissions
    - Permissions define allowed operations
2. **Implementation in ASP.NET Core**:
    - Role-based authorization attributes
    - Policy-based authorization
    - Custom authorization handlers
    - Claims-based identity
3. **Benefits**:
    - Simplified access management
    - Hierarchical permission structure
    - Easy to audit and maintain
    - Scalable for large applications

Handling Secure APIs

Best practices for securing REST APIs:

1. **Transport Security**:
    - Always use HTTPS
    - Implement SSL/TLS
    - HTTP Strict Transport Security (HSTS)
2. **Authentication & Authorization**:
    - Token-based authentication
    - Short token expiration times
    - Secure token storage
    - Role-based access control
3. **Input Validation**:
    - Validate all inputs
    - Use strong data types
    - Implement model validation
    - Sanitize user input
4. **Security Headers**:
    - CORS policies
    - Content Security Policy

   - XSS Protection
   - Frame Options

5. **Rate Limiting**:
   - Implement request throttling
   - Prevent brute force attacks
   - API usage quotas
   - DDoS protection

6. **Error Handling**:
   - Don't expose sensitive information
   - Consistent error responses
   - Proper logging
   - Security exception handling

7. **Data Protection**:
   - Encrypt sensitive data
   - Secure password storage
   - Data access auditing
   - Principle of least privilege

Step-by-step guide for creating a secured REST endpoint using ASP.Net Core and Entity Framework Core

This guide will walk you through creating a secure Product Management REST API using ASP.NET Core Web API, Entity Framework Core with SQL Server, and JWT Authentication. This project demonstrates essential concepts like:

- REST API development
- Database integration with Entity Framework Core
- Authentication and Authorization with JWT
- Password hashing
- Secure endpoints
- Data validation

Prerequisites

1. Visual Studio 2022 (Community Edition or higher)
2. SQL Server Express or LocalDB
3. .NET 8.0 SDK
4. Basic understanding of C#, REST APIs, and HTTP methods

Step 1: Create the Project

1. Open Visual Studio 2022
2. Click "Create a new project"
3. Select "ASP.NET Core Web API"

4. Configure your new project:
    - Project name: ProductServiceWithEFAndSecurity
    - Location: Choose your preferred location
    - Solution name: ProductServiceWithEFAndSecurity
5. Click Next
6. Select:
    - Framework: .NET 8.0 (LTS)
    - Authentication type: None (we'll add it manually)
    - Configure for HTTPS: ✓
    - Enable OpenAPI support: ✓
    - Use controllers: ✓
    - Enable Docker: ✗
7. Click Create

Step 2: Install Required NuGet Packages

Install the following NuGet packages either through the Package Manager Console or the NuGet Package Manager UI:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 8.0.2
Install-Package Microsoft.AspNetCore.Authentication.JwtBearer -Version 8.0.2
Install-Package BCrypt.Net-Next -Version 4.0.3
```

Or using .NET CLI:

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 8.0.2
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 8.0.2
dotnet add package BCrypt.Net-Next --version 4.0.3
```

> **Note**: The versions specified above are compatible with .NET 8.0. If you're using a different .NET version, make sure to use compatible package versions.

Step 3: Create the Models

Create Product Model

Create a new folder called "Models" and add a new class "Product.cs":

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

namespace ProductServiceWithEFAndSecurity.Models
{
    public class Product
    {
```

```
        [Key]
        public Guid Id { get; set; }

        [Required(ErrorMessage = "Name is required")]
        [StringLength(100)]
        public string Name { get; set; }

        [Required(ErrorMessage = "Description is required")]
        [StringLength(500)]
        public string Description { get; set; }

        [Required]
        [Column(TypeName = "decimal(18,2)")]
        [Range(0.01, double.MaxValue, ErrorMessage = "Price must be greater than
0")]
        public decimal Price { get; set; }

        [Required(ErrorMessage = "Units is required")]
        [StringLength(20)]
        public string Units { get; set; }

        [Url(ErrorMessage = "Please provide a valid URL for the picture")]
        public string Picture { get; set; }

        [Required]
        [Range(0, int.MaxValue, ErrorMessage = "Units in stock must be 0 or
greater")]
        public int UnitsInStock { get; set; }
    }
}
```

Create User Model

In the same Models folder, create "User.cs":

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

namespace ProductServiceWithEFAndSecurity.Models
{
    public class User
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        public string Username { get; set; }

        [Required]
        public string Password { get; set; }
```

```
    }

    public class LoginModel
    {
        [Required]
        public string Username { get; set; }

        [Required]
        public string Password { get; set; }
    }
}
```

Step 4: Create the Database Context

Create a new folder called "Data" and add "ProductDbContext.cs":

```
using Microsoft.EntityFrameworkCore;
using ProductServiceWithEFAndSecurity.Models;

namespace ProductServiceWithEFAndSecurity.Data
{
    public class ProductDbContext : DbContext
    {
        public ProductDbContext(DbContextOptions<ProductDbContext> options)
            : base(options)
        {
        }

        public DbSet<Product> Products { get; set; }
        public DbSet<User> Users { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Configure Product entity
            modelBuilder.Entity<Product>()
                .Property(p => p.Price)
                .HasColumnType("decimal(18,2)");

            // Configure User entity
            modelBuilder.Entity<User>()
                .HasIndex(u => u.Username)
                .IsUnique();
        }
    }
}
```

Step 5: Create the Authentication Service

Create a new folder called "Services" and add "AuthService.cs":

```
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using ProductServiceWithEFAndSecurity.Data;
using ProductServiceWithEFAndSecurity.Models;

namespace ProductServiceWithEFAndSecurity.Services
{
    public interface IAuthService
    {
        Task<string> AuthenticateAsync(string username, string password);
        Task<bool> RegisterAsync(string username, string password);
    }

    public class AuthService : IAuthService
    {
        private readonly string _jwtKey;
        private readonly ProductDbContext _context;

        public AuthService(ProductDbContext context, string jwtKey)
        {
            _context = context;
            _jwtKey = jwtKey;
        }

        public async Task<string> AuthenticateAsync(string username, string
password)
        {
            var user = await _context.Users.FirstOrDefaultAsync(u => u.Username ==
username);

            if (user == null || !BCrypt.Net.BCrypt.Verify(password,
user.Password))
            {
                return null;
            }

            // Generate JWT token
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_jwtKey);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                    new Claim(ClaimTypes.Name, username),
                    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())
                }),
                Expires = DateTime.UtcNow.AddDays(7),
```

```
                SigningCredentials = new SigningCredentials(
                    new SymmetricSecurityKey(key),
                    SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }

        public async Task<bool> RegisterAsync(string username, string password)
        {
            if (await _context.Users.AnyAsync(u => u.Username == username))
            {
                return false; // Username already exists
            }

            var user = new User
            {
                Username = username,
                Password = BCrypt.Net.BCrypt.HashPassword(password)
            };

            _context.Users.Add(user);
            await _context.SaveChangesAsync();
            return true;
        }
    }
}
```

Step 6: Create the Controllers

Create AuthController

In the Controllers folder, add "AuthController.cs":

```
using Microsoft.AspNetCore.Mvc;
using ProductServiceWithEFAndSecurity.Models;
using ProductServiceWithEFAndSecurity.Services;

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

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

```
        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] LoginModel model)
        {
            var token = await _authService.AuthenticateAsync(model.Username,
model.Password);

            if (token == null)
                return Unauthorized(new { message = "Invalid username or password"
});

            return Ok(new { token });
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] LoginModel model)
        {
            if (string.IsNullOrWhiteSpace(model.Username) ||
string.IsNullOrWhiteSpace(model.Password))
                return BadRequest(new { message = "Username and password are
required" });

            var result = await _authService.RegisterAsync(model.Username,
model.Password);

            if (!result)
                return BadRequest(new { message = "Username already exists" });

            return Ok(new { message = "User registered successfully" });
        }
    }
}
```

Create ProductsController

Add "ProductsController.cs":

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ProductServiceWithEFAndSecurity.Data;
using ProductServiceWithEFAndSecurity.Models;

namespace ProductServiceWithEFAndSecurity.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductsController : ControllerBase
```

```
    {
        private readonly ProductDbContext _context;

        public ProductsController(ProductDbContext context)
        {
            _context = context;
        }

        // GET: api/products - No authentication required
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Product>>> GetProducts()
        {
            return await _context.Products.ToListAsync();
        }

        // GET: api/products/{id} - No authentication required
        [HttpGet("{id}")]
        public async Task<ActionResult<Product>> GetProduct(Guid id)
        {
            var product = await _context.Products.FindAsync(id);

            if (product == null)
            {
                return NotFound();
            }

            return product;
        }

        // POST: api/products - Requires authentication
        [Authorize]
        [HttpPost]
        public async Task<ActionResult<Product>> CreateProduct(Product product)
        {
            if (product.Id == Guid.Empty)
            {
                product.Id = Guid.NewGuid();
            }

            _context.Products.Add(product);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(GetProduct), new { id = product.Id },
product);
        }

        // PUT: api/products/{id} - Requires authentication
        [Authorize]
        [HttpPut("{id}")]
        public async Task<IActionResult> UpdateProduct(Guid id, Product product)
        {
            if (id != product.Id)
            {
                return BadRequest();
```

```
            }

            _context.Entry(product).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!await ProductExists(id))
                {
                    return NotFound();
                }
                throw;
            }

            return NoContent();
        }

        // DELETE: api/products/{id} - Requires authentication
        [Authorize]
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteProduct(Guid id)
        {
            var product = await _context.Products.FindAsync(id);
            if (product == null)
            {
                return NotFound();
            }

            _context.Products.Remove(product);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private async Task<bool> ProductExists(Guid id)
        {
            return await _context.Products.AnyAsync(e => e.Id == id);
        }
    }
}
```

Step 7: Configure appsettings.json

Update appsettings.json with the connection string and JWT settings:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
```

```
"Server=.\\SQLEXPRESS;Database=jecrc_secure;Trusted_Connection=True;TrustServerCer
tificate=True;MultipleActiveResultSets=true"
  },
  "JwtSettings": {
    "Key": "your-super-secret-key-with-at-least-32-characters",
    "ExpiryInDays": 7
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Step 8: Configure Program.cs

Replace the contents of Program.cs with:

```
using System.Text;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using ProductServiceWithEFAndSecurity.Data;
using ProductServiceWithEFAndSecurity.Services;
using ProductServiceWithEFAndSecurity.Models;

var builder = WebApplication.CreateBuilder(args);

// Database
builder.Services.AddDbContext<ProductDbContext>(options =>
{

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"
));
    if (builder.Environment.IsDevelopment())
    {
        options.EnableSensitiveDataLogging()
               .EnableDetailedErrors();
    }
});

// JWT Configuration
var jwtKey = builder.Configuration["JwtSettings:Key"] ??
    "your-super-secret-key-with-at-least-32-characters"; // Fallback key for
development
var key = Encoding.ASCII.GetBytes(jwtKey);

// Register AuthService with DbContext
builder.Services.AddScoped<IAuthService>(provider =>
```

```
    new AuthService(provider.GetRequiredService<ProductDbContext>(), jwtKey));

// JWT Authentication
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(x =>
{
    x.RequireHttpsMetadata = false; // Set to true in production
    x.SaveToken = true;
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(key),
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true
    };
});

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Create and seed the database
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var context = services.GetRequiredService<ProductDbContext>();

        // First ensure database is deleted to start fresh
        context.Database.EnsureDeleted();

        // Then create the database and tables
        context.Database.EnsureCreated();

        // Now seed the admin user
        var adminUser = new User
        {
            Username = "admin",
            Password = BCrypt.Net.BCrypt.HashPassword("admin123")
        };
        context.Users.Add(adminUser);
        context.SaveChanges();

        Console.WriteLine("Database created and seeded successfully!");
    }
    catch (Exception ex)
```

```
    {
        var logger = services.GetRequiredService<ILogger<Program>>();
        logger.LogError(ex, "An error occurred while initializing the database.");
        throw;
    }
}

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

Step 9: Testing the API

1. Run the application
2. Open Swagger UI (https://localhost:5277/swagger)

Authentication Flow:

1. First, register a new user (optional, as admin is already seeded):

```
POST /api/auth/register
Content-Type: application/json

{
    "username": "newuser",
    "password": "password123"
}
```

2. Login to get JWT token:

```
POST /api/auth/login
Content-Type: application/json

{
    "username": "admin",
    "password": "admin123"
}
```

3. Use the token for authenticated endpoints:

- Click "Authorize" button in Swagger
- Enter: Bearer your-token-here
- Or in HTTP requests, add header: Authorization: Bearer your-token-here

Testing Products API:

1. Get all products (no auth required):

```
GET /api/products
```

2. Create a product (requires auth):

```
POST /api/products
Content-Type: application/json
Authorization: Bearer your-token-here

{
    "name": "Basmati Rice",
    "description": "Premium long-grain basmati rice",
    "price": 99.99,
    "units": "1 kg",
    "picture": "https://example.com/rice.jpg",
    "unitsInStock": 100
}
```

Understanding the Security Features

1. **Password Security**:
   - Passwords are hashed using BCrypt
   - Original passwords are never stored in the database
   - BCrypt automatically handles salt generation and storage
2. **JWT Authentication**:
   - Tokens are signed with a secret key
   - Tokens include user identity claims
   - Tokens expire after 7 days
   - Protected endpoints require valid tokens
3. **Database Security**:
   - Uses Windows Authentication for SQL Server
   - Unique constraint on username
   - Auto-generated IDs for security
4. **API Security**:

- HTTPS enabled
- Authentication required for modifying data
- Input validation using Data Annotations
- SQL injection protection via Entity Framework

Common Issues and Solutions

1. **Database Connection**:
   - Ensure SQL Server is running
   - Check connection string in appsettings.json
   - Verify Windows Authentication is enabled
2. **Authentication Issues**:
   - Check token format (Bearer prefix)
   - Verify token hasn't expired
   - Ensure username/password are correct
3. **API Errors**:
   - 401: Not authenticated
   - 403: Not authorized
   - 400: Invalid input
   - 404: Resource not found

Next Steps for Enhancement

1. Add refresh tokens
2. Implement role-based authorization
3. Add email verification
4. Implement password reset
5. Add request rate limiting
6. Enhance input validation
7. Add logging and monitoring
8. Implement CORS policy

Additional Resources

- ASP.NET Core Documentation
- Entity Framework Core
- JWT Authentication
- Security Best Practices