Building a REST API with ASP.NET Core and Entity Framework Core

This guide will walk you through creating a Product Management REST API using ASP.NET Core Web API and Entity Framework Core with SQL Server.

Prerequisites

1. Visual Studio 2022 (Community Edition or higher)
2. SQL Server Express installed
3. Basic understanding of C# and REST APIs

Step 1: Create the Project

1. Open Visual Studio 2022
2. Click "Create a new project"
3. Select "ASP.NET Core Web API"
4. Set the following details:
    - Project name: ProductServiceWithEF
    - Location: Choose your preferred location
    - Solution name: ProductServiceWithEF
5. Click Next
6. Select:
    - Framework: .NET 8.0 (or latest LTS version)
    - Authentication type: None
    - Configure for HTTPS: Checked
    - Enable OpenAPI support: Checked
    - Use controllers: Checked
7. Click Create

Step 2: Install Required NuGet Packages

1. Right-click on the project in Solution Explorer
2. Select "Manage NuGet Packages"
3. Install these packages:
    - Microsoft.EntityFrameworkCore.SqlServer
    - Microsoft.EntityFrameworkCore.Tools

Alternatively, you can run these commands in Package Manager Console:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Step 3: Create the Product Model

1. Create a new folder called "Models"
2. Add a new class "Product.cs" in the Models folder
3. Add the following code:

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

namespace ProductServiceWithEF.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; }
    }
}
```

Step 4: Create the Database Context

1. Create a new folder called "Data"
2. Add a new class "ProductDbContext.cs" in the Data folder
3. Add the following code:

```
using Microsoft.EntityFrameworkCore;
using ProductServiceWithEF.Models;

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

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

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

            modelBuilder.Entity<Product>()
                .Property(p => p.Price)
                .HasColumnType("decimal(18,2)");
        }
    }
}
```

Step 5: Configure Database Connection

1. Open appsettings.json
2. Add the connection string inside the existing JSON:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=.\\SQLEXPRESS;Database=jecrc;Trusted_Connection=True;TrustServerCertificat
e=True;MultipleActiveResultSets=true"
  },
  // ... existing settings ...
}
```

Step 6: Create the Controller

1. In the Controllers folder, add a new class "ProductsController.cs"
2. Add the following code:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.EntityFrameworkCore;
using ProductServiceWithEF.Data;
using ProductServiceWithEF.Models;

namespace ProductServiceWithEF.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductsController : ControllerBase
    {
        private readonly ProductDbContext _context;

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

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Product>>> GetProducts()
        {
            return await _context.Products.ToListAsync();
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<Product>> GetProduct(Guid id)
        {
            var product = await _context.Products.FindAsync(id);

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

            return product;
        }

        [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);
        }

        [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();
        }

        [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: Update Program.cs

1. Open Program.cs
2. Replace its contents with:

```
using Microsoft.EntityFrameworkCore;
using ProductServiceWithEF.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddDbContext<ProductDbContext>(options =>
{

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"
));
    options.EnableSensitiveDataLogging()
           .EnableDetailedErrors()
           .LogTo(Console.WriteLine);
});

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

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

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

// Create the database if it doesn't exist
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var context = services.GetRequiredService<ProductDbContext>();
        context.Database.EnsureCreated();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"An error occurred while setting up the database:
{ex.Message}");
    }
}

app.Run();
```

Step 8: Run and Test the Application

1. Press F5 to run the application
2. The Swagger UI will open in your browser
3. Test the API endpoints:
    - POST /api/products - Create a new product
    - GET /api/products - List all products
    - GET /api/products/{id} - Get a specific product
    - PUT /api/products/{id} - Update a product
    - DELETE /api/products/{id} - Delete a product

Sample Product JSON for Testing

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

Verifying the Database

1. Open SQL Server Management Studio
2. Connect to your local SQL Express instance:
    - Server name: .\SQLEXPRESS
    - Authentication: Windows Authentication
3. You should see:
    - Database: jecrc
    - Table: Products

Common Issues and Solutions

1. **Connection String Error**: Make sure SQL Server Express is running and the connection string is correct
2. **Database Not Created**: Check if you have appropriate permissions to create databases
3. **Swagger Not Loading**: Ensure the Swagger middleware is properly configured in Program.cs

Next Steps

1. Add input validation
2. Implement sorting and filtering
3. Add authentication and authorization
4. Implement pagination
5. Add logging
6. Create unit tests

Additional Resources

- Entity Framework Core Documentation
- ASP.NET Core Documentation
- REST API Best Practices