Building a REST API with ASP.NET Core (In-Memory Storage)

This guide will walk you through creating a Product Management REST API using ASP.NET Core Web API with in-memory storage. This is a simpler version that doesn't require a database, making it perfect for learning and prototyping.

Prerequisites

1. Visual Studio 2022 (Community Edition or higher)
2. 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: ProductService
   - Location: Choose your preferred location
   - Solution name: ProductService
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: 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;

namespace ProductService.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]
        [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 3: Create the Product Service

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using ProductService.Models;

namespace ProductService.Services
{
    public interface IProductService
    {
        IEnumerable<Product> GetAll();
        Product GetById(Guid id);
        Product Add(Product product);
        Product Update(Product product);
        bool Delete(Guid id);
    }

    public class ProductService : IProductService
    {
        private readonly List<Product> _products;

        public ProductService()
        {
            _products = new List<Product>();
        }

        public IEnumerable<Product> GetAll()
        {
            return _products;
        }

        public Product GetById(Guid id)
        {
            return _products.FirstOrDefault(p => p.Id == id);
        }

        public Product Add(Product product)
        {
            if (product.Id == Guid.Empty)
            {
                product.Id = Guid.NewGuid();
            }
            _products.Add(product);
            return product;
        }

        public Product Update(Product product)
        {
            var existingProduct = _products.FirstOrDefault(p => p.Id ==
product.Id);
            if (existingProduct == null)
            {
                return null;
            }

            _products.Remove(existingProduct);
            _products.Add(product);
            return product;
        }

        public bool Delete(Guid id)
        {
            var product = _products.FirstOrDefault(p => p.Id == id);
            if (product == null)
            {
                return false;
            }

            return _products.Remove(product);
        }
    }
}
```

Step 4: 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 Microsoft.AspNetCore.Mvc;
using ProductService.Models;
using ProductService.Services;

namespace ProductService.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductsController : ControllerBase
    {
        private readonly IProductService _productService;

        public ProductsController(IProductService productService)
        {
            _productService = productService;
        }

        [HttpGet]
        public ActionResult<IEnumerable<Product>> GetProducts()
        {
            return Ok(_productService.GetAll());
        }

        [HttpGet("{id}")]
        public ActionResult<Product> GetProduct(Guid id)
        {
            var product = _productService.GetById(id);

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

            return Ok(product);
        }

        [HttpPost]
        public ActionResult<Product> CreateProduct(Product product)
        {
            var createdProduct = _productService.Add(product);
            return CreatedAtAction(nameof(GetProduct), new { id =
createdProduct.Id }, createdProduct);
        }
        [HttpPut("{id}")]
        public IActionResult UpdateProduct(Guid id, Product product)
        {
            if (id != product.Id)
            {
                return BadRequest();
            }

            var updatedProduct = _productService.Update(product);
            if (updatedProduct == null)
            {
                return NotFound();
            }

            return NoContent();
        }

        [HttpDelete("{id}")]
        public IActionResult DeleteProduct(Guid id)
        {
            var result = _productService.Delete(id);
            if (!result)
            {
                return NotFound();
            }

            return NoContent();
        }
    }
}
```

Step 5: Update Program.cs

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

```
using ProductService.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddSingleton<IProductService,
ProductService.Services.ProductService>();
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();

app.Run();
```

Step 6: 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
}
```

Important Notes

1. This implementation uses in-memory storage (List)
2. Data will be lost when the application restarts
3. The service is registered as a Singleton to maintain data during the application's lifetime

Key Differences from the EF Core Version

1. No database configuration required
2. Uses List instead of DbContext
3. Synchronous operations (no async/await)
4. Data is volatile (lost on application restart)
5. Simpler setup and deployment
6. Perfect for prototyping and learning

Common Issues and Solutions

1. **Data Loss After Restart**: This is expected behavior with in-memory storage
2. **Swagger Not Loading**: Ensure the Swagger middleware is properly configured in Program.cs
3. **Duplicate IDs**: The service automatically generates new GUIDs for products without IDs

Next Steps

1. Add input validation
2. Implement sorting and filtering
3. Add authentication and authorization
4. Add logging
5. Create unit tests
6. Consider upgrading to database storage (see the ProductServiceWithEF project)

Additional Resources

- ASP.NET Core Documentation
- REST API Best Practices
- C# In-Memory Collections