**1. Create the ASP.NET Core Web API Project**

# Create a new Web API project

dotnet new webapi -n dotnetapp

cd dotnetapp

# Build project to check for errors

dotnet build dotnetapp.csproj

**Configure Application URL to Port 8080**

Edit Properties/launchSettings.json and set:

"applicationUrl": "http://0.0.0.0:8080"

**2. Add the Product Model**

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public string Description { get; set; }

}

**3. Create Repository Interface**

public interface IProductRepository

{

IEnumerable<Product> GetProducts();

Product GetProduct(int productId);

Product CreateProduct(Product product);

Product UpdateProduct(Product product);

bool DeleteProduct(int productId);

}

**4. Implement ProductController**

[Route("api/[controller]")]

[ApiController]

public class ProductController : ControllerBase

{

private readonly IProductRepository \_repository;

public ProductController(IProductRepository repository)

{

\_repository = repository;

}

[HttpGet]

public IActionResult GetAll()

{

return Ok(\_repository.GetProducts());

}

[HttpGet("{id}")]

public IActionResult Get(int id)

{

var product = \_repository.GetProduct(id);

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

return Ok(product);

}

[HttpPost]

public IActionResult Create(Product product)

{

var created = \_repository.CreateProduct(product);

return CreatedAtAction(nameof(Get), new { id = created.ProductId }, created);

}

[HttpPut("{id}")]

public IActionResult Update(int id, Product product)

{

if (id != product.ProductId) return BadRequest();

var updated = \_repository.UpdateProduct(product);

if (updated == null) return NotFound();

return Ok(updated);

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var deleted = \_repository.DeleteProduct(id);

if (!deleted) return NotFound();

return NoContent();

}

}

**5. Setup Entity Framework Core (Optional)**

dotnet add package Microsoft.EntityFrameworkCore --version 6.0

dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 6.0

dotnet add package Microsoft.EntityFrameworkCore.Tools --version 6.0

Sample **DbContext**:

public class AppDbContext : DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

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

}

**6. Unit Testing Setup**

**Create NUnit Test Project**

cd ..

dotnet new sln -n dotnetapp

dotnet new nunit -n MyTestNUnitProject

dotnet sln add dotnetapp/dotnetapp.csproj

dotnet sln add MyTestNUnitProject/MyTestNUnitProject.csproj

dotnet add MyTestNUnitProject/MyTestNUnitProject.csproj reference dotnetapp/dotnetapp.csproj

dotnet add package Moq --version 4.18.4

**7. Writing Unit Tests with NUnit and Moq**

**Example: ProductController Tests**

using Moq;

using NUnit.Framework;

using dotnetapp.Controllers;

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc;

[TestFixture]

public class ProductControllerTests

{

private Mock<IProductRepository> \_mockRepo;

private ProductController \_controller;

[SetUp]

public void Setup()

{

\_mockRepo = new Mock<IProductRepository>();

\_controller = new ProductController(\_mockRepo.Object);

}

[Test]

public void GetAll\_ReturnsAllProducts()

{

\_mockRepo.Setup(repo => repo.GetProducts())

.Returns(new List<Product> { new Product { ProductId = 1, Name = "Test", Price = 10 } });

var result = \_controller.GetAll() as OkObjectResult;

Assert.IsNotNull(result);

Assert.AreEqual(200, result.StatusCode);

}

[Test]

public void Get\_ProductNotFound\_ReturnsNotFound()

{

\_mockRepo.Setup(repo => repo.GetProduct(1)).Returns((Product)null);

var result = \_controller.Get(1);

Assert.IsInstanceOf<NotFoundResult>(result);

}

[Test]

public void Create\_ValidProduct\_ReturnsCreatedProduct()

{

var product = new Product { ProductId = 1, Name = "New", Price = 50 };

\_mockRepo.Setup(r => r.CreateProduct(product)).Returns(product);

var result = \_controller.Create(product) as CreatedAtActionResult;

Assert.IsNotNull(result);

Assert.AreEqual(201, result.StatusCode);

}

}

**Tips:**

* Mock the repository using **Moq** to isolate controller logic.
* Write tests for **positive** and **negative scenarios**.
* Use Assert.IsTrue, Assert.IsFalse, Assert.AreEqual, etc., for validations.

**8. Run Your Application and Tests**

# Run API

cd dotnetapp

dotnet restore

dotnet run

# Run Tests

cd ../MyTestNUnitProject

dotnet test