Project Question Problem Statement: Create a RESTful API to manage a collection of products. The API should allow clients to retrieve a list of products, retrieve a single product by ID, filter products by category using route parameters and query string parameters, and create new products. Project Structure: Controller: ProductsController This controller will handle all the HTTP requests related to product management. It will include methods to retrieve a list of products, retrieve a single product by ID, filter products by category, and create new products. Create a folder named Data and have the ApplicationDbContext.cs file to set up the database context for the application with the DbSet property Products. Model: Product Create a folder named Models and create the model for Product. Represents a product in the system. Properties: Id: Integer - The unique identifier of the product (auto-increments by 1) Name: String - The name of the product. Category: String - The category of the product. Price: Decimal - The price of the product. Stock: Integer - The available stock of the product. Controller Methods: 1. GET /api/products: Retrieves a list of all products. Status: 200 Route: /api/products Method: public ActionResult<IEnumerable<Product>> GetProducts() Response: List of Product objects 2. GET /api/products/{id} Retrieves a single product by its ID. Status: 200 Route: /api/products/{id} Method: public ActionResult<Product> GetProductById(int id) Parameters:id: Integer (Route parameter) Response: Product object with the specified ID 3. GET /api/products/filter?category={categoryName} Retrieves a list of products filtered by category. Route: /api/products/filter Status: 200 Method: public ActionResult<IEnumerable<Product>> GetProductsByCategory([FromQuery] string category) Query String Parameter:category: String Response: List of Product objects filtered by the specified category 4. POST /api/products Creates a new product. Route: /api/products Status: 201 Method: public ActionResult<Product> CreateProduct(Product product) Request Body: Product object (JSON) Response: Product object representing the newly created product Example Request: 1. Retrieve all products: Request: GET /api/products Response: List of all products in the system. 2. Retrieve a product by ID: Request: GET /api/products/1 Response: Details of the product with ID 1. 3. Retrieve products filtered by category: Request: GET /api/products/filter?category=electronics Response: List of products belonging to the "electronics" category. 4. Create a new product: Request: POST /api/products Content-Type: application/json { "Name": "New Product", "Category": "electronics", "Price": 99.99, "Stock": 10 } Response: Details of the newly created product. Note: Make sure that, before running the test cases or submitting the project, the project is running on the port 8080 Link to view Swagger/idex.html page - https://8080-\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.premiumproject.examly.io/swagger/index.html Commands to Run the Project: cd dotnetapp - Select the dotnet project folder dotnet restore - This command will restore all the required packages to run the application. dotnet run - To run the application in port 8080 (The settings preloaded click 8080 Port to View) dotnet build - To build and check for errors dotnet clean - If the same error persists clean the project and build again Install EF using the following commands : dotnet new tool-manifest dotnet tool install --local dotnet-ef --version 6.0.6 --Then use dotnet dotnet-ef instead of dotnet-ef. dotnet dotnet-ef --To check the EF installed or not dotnet dotnet-ef migrations add "InitialSetup" --command to setup initial creating of tables mentioned in DBContext dotnet dotnet-ef database update --command to update the database To Work with SQLServer: (Open a New Terminal) type the below commands sqlcmd -U sa password: examlyMssql@123 1> create database DBName 2> go >use DBName >go 1> create table TableName(id int identity(1,1),........) 2> go Note: Use the below sample connection string to connect the MsSql Server private string connectionString = "User ID=sa;password=examlyMssql@123; server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False"; Endpoint Screenshot: Note : The project will not be submitted if "Submit Project" is not done atleast once

**ChatGPT said:**

Here’s a complete guide and sample implementation for your **Products RESTful API** in ASP.NET Core, following your project requirements. I’ve structured it step by step.

**1. Project Structure**

dotnetapp/

│

├── Controllers/

│ └── ProductsController.cs

│

├── Models/

│ └── Product.cs

│

├── Data/

│ └── ApplicationDbContext.cs

│

├── Program.cs

└── appsettings.json

**2. Model: Product**

Create Models/Product.cs:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace dotnetapp.Models

{

public class Product

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

public string Category { get; set; }

[Column(TypeName = "decimal(18,2)")]

public decimal Price { get; set; }

public int Stock { get; set; }

}

}

**3. Database Context: ApplicationDbContext**

Create Data/ApplicationDbContext.cs:

using Microsoft.EntityFrameworkCore;

using dotnetapp.Models;

namespace dotnetapp.Data

{

public class ApplicationDbContext : DbContext

{

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

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

}

}

**4. Controller: ProductsController**

Create Controllers/ProductsController.cs:

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using dotnetapp.Data;

using dotnetapp.Models;

namespace dotnetapp.Controllers

{

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

[ApiController]

public class ProductsController : ControllerBase

{

private readonly ApplicationDbContext \_context;

public ProductsController(ApplicationDbContext context)

{

\_context = context;

}

// GET: /api/products

[HttpGet]

public async Task<ActionResult<IEnumerable<Product>>> GetProducts()

{

return await \_context.Products.ToListAsync();

}

// GET: /api/products/{id}

[HttpGet("{id}")]

public async Task<ActionResult<Product>> GetProductById(int id)

{

var product = await \_context.Products.FindAsync(id);

if (product == null)

return NotFound();

return product;

}

// GET: /api/products/filter?category={categoryName}

[HttpGet("filter")]

public async Task<ActionResult<IEnumerable<Product>>> GetProductsByCategory([FromQuery] string category)

{

var products = await \_context.Products

.Where(p => p.Category.ToLower() == category.ToLower())

.ToListAsync();

return products;

}

// POST: /api/products

[HttpPost]

public async Task<ActionResult<Product>> CreateProduct([FromBody] Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

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

}

}

}

**5. Program.cs**

Ensure EF Core and SQL Server connection are configured:

using Microsoft.EntityFrameworkCore;

using dotnetapp.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Configure SQL Server connection

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer("User ID=sa;password=examlyMssql@123;server=localhost;Database=appdb;trusted\_connection=false;Persist Security Info=False;Encrypt=False"));

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**6. EF Core Commands**

Install EF Core locally:

dotnet new tool-manifest

dotnet tool install --local dotnet-ef --version 6.0.6

dotnet dotnet-ef

Create initial migration and update database:

dotnet dotnet-ef migrations add InitialSetup

dotnet dotnet-ef database update

**7. Run the Project**

cd dotnetapp

dotnet restore

dotnet run

Ensure it runs on **port 8080** (set via launchSettings.json if needed).  
Swagger URL: https://localhost:8080/swagger/index.html

**8. Sample Requests**

1. **Get all products**  
   GET /api/products
2. **Get product by ID**  
   GET /api/products/1
3. **Filter products by category**  
   GET /api/products/filter?category=electronics
4. **Create a new product**  
   POST /api/products  
   Body:
5. {
6. "Name": "New Product",
7. "Category": "electronics",
8. "Price": 99.99,
9. "Stock": 10
10. }

This setup covers **all your project requireme**