.NET 5 REST API Tutorial - Build From Scratch With C#

I. GETTING STARTED

--- 1. What will be learnt ?

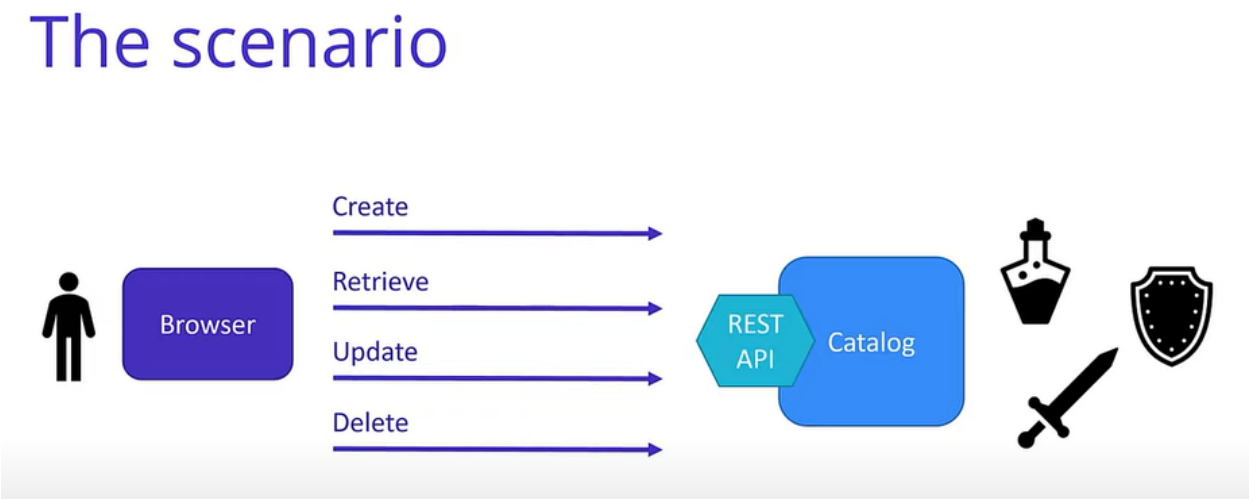
1. The scenario to be used across the tutorial
2. How to create a .NET 5 Web API project from scratch
3. How to use VS Code for building and debugging the project
4. How to trust the development certificate
5. How to use SwaggerUI to interact with the API

--- 2. Installing tools

1. I have to install version 5.0.101 as recommended here, otherwise I will get an error Unable to resolve 'Swashbuckle.AspNetCore (>= 5.6.3)' for 'net5.0' if I install the 5.0.401 (Latest version for the time being) and create the project using CLI
2. Visual Studio Code
3. Basic understanding of the C# language

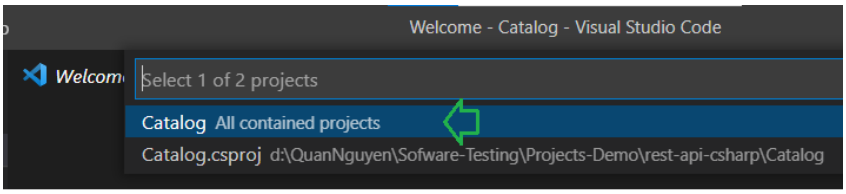
--- 3. The Scenario

- Catalog: danh mục
- Item: mặt hàng
- So in this sense: vì vậy theo ngữ nghĩa này
- Tôi giống như người chơi video game, tôi có thuốc, khiên, kiếm
- We want to manage these catalog items:
    - Create the catalog
    - Retrieve (truy/get) the list of item
    - Update the properties of the item
    - Delete the items in the catalog
- We don't have a way to expose the catalog to the internet → We are going to use REST API → We are going to build this REST API from scratch using dotnet.5

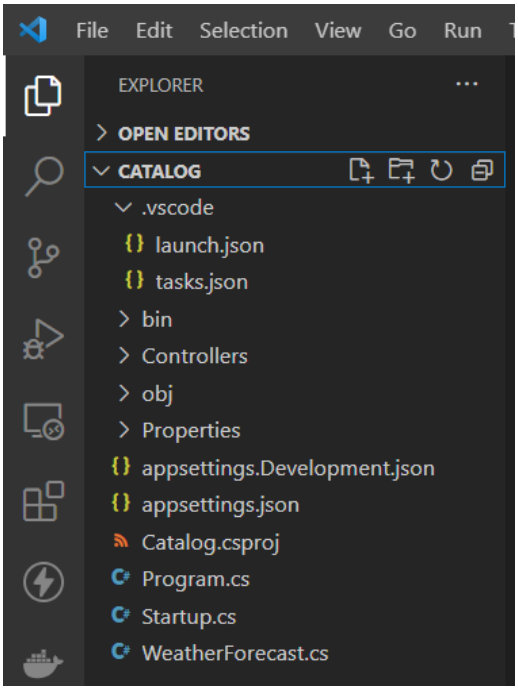


--- 4. Creating the Project

1. Open VS Code
2. **Make sure add extension: C#** for Visual Studio Code (powered by OmniSharp).
3. Restart VS Code → Terminal → New Terminal
   → cd D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp
   → dotnet new webapi -n Catalog
   **** Note: If you get the error: ...Catalog\Catalog.csproj : error NU1100: Unable to resolve 'Swashbuckle.AspNetCore (>= 5.6.3)' for 'net5.0'. → Uninstall .Net 5 and install this version*
4. File → Open Folder → Select 'Catalog' → Select Folder → Trust author of all files in rest-api-csharp → Yes → Select 'Catalog' All contained projects



Then It should load and auto popup '**Add assets to build and debug**' → **Yes** → Now you should see folder **bin** and **.vscode**



--- 5. Exploring the generated project files

1. Open **Catalog.csproj** file
   - Line: `<Project Sdk="Microsoft.NET.Sdk.Web">` → we're going to use the .NET.Sdk.Web to build the project, which includes a bunch of tools and tasks
   - Line: `<TargetFramework>net5.0</TargetFramework>` → The target framework defines the API's or what kind of API's are going to be available to the project. In this case net.5 is perfectly good for us.
   - A bunch of nuget `<PackageReference>`→ we're not going to be diving into right now. But those are just dependencies that we already acquired on this project

2. Open **Program.cs** file → it is the entry point of the application. It creates a HostBuilder and the hostBuilder calls **Startup**

3. Open **Startup.cs** file → it sets up configuration properties (environment variables, files, folders. It has a Startup constructor.
   a. Open **ConfigureServices** method → it is where you would register all the services that you're going to be using across your application.
   b. Open **Configure** method → it is request handling pipeline configuration. This will be run before your code is actually executed

4. Open **WeatherForecast.cs** file → it is auto generated as a sample with few very simple properties → Just watch it → then delete it.

5. Open folder Controllers/**WeatherForecastControllers.cs** file → it is a class that handles the routes. Yeah, pretty much the routes that your service exposes. We don't use it. → Just watch it → then delete it.

6. Open **appsettings.json** file → it is where you can declare configuration that you don't want to hardcode in everywhere else

7. Open **appsettings.Development.json** file → that is setup for Develop environment, you can also setup for Production, Staging, QA

8. Open **.vscode/tasks.json** → declare the tasks with key concepts of VS code. We will use label = build, command = dotnet (they are auto generated)

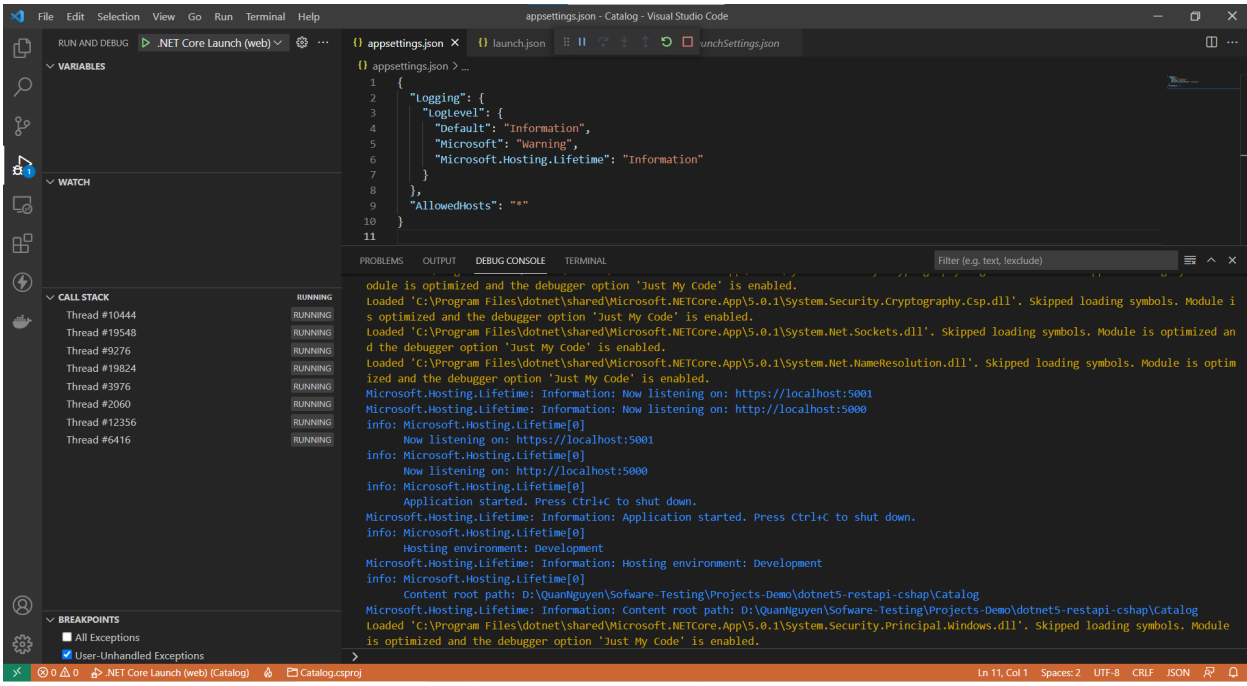
9. Open **.vscode/launch.json** → controls what is going to be launched or executed when we press f5 or start debugging

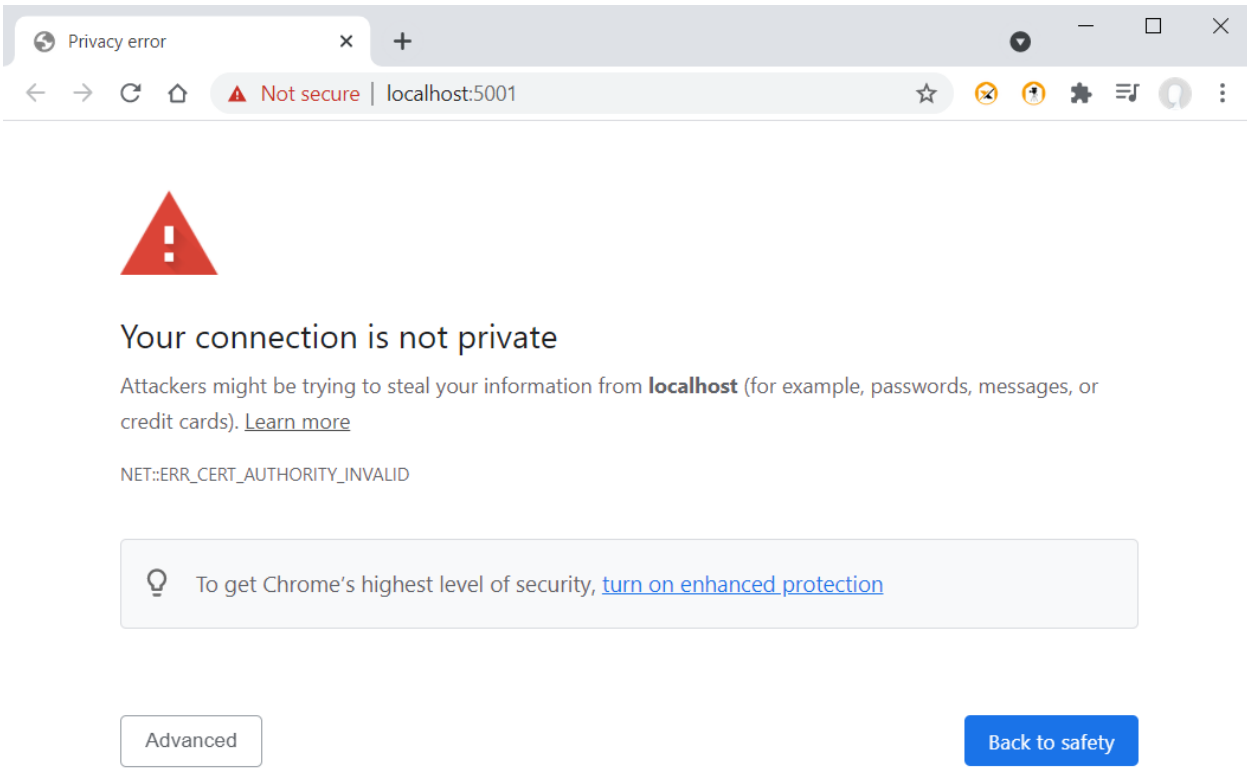
10. Open **Properties**/ **launchSettings.json** file → look and remember "applicationUrl": "
    - **http**://localhost:**5000**
    - **https**://localhost:**5001**

    Remember it is setting as → `"ASPNETCORE_ENVIRONMENT": "Development` and in **.vscode/launch.json** is same → `"ASPNETCORE_ENVIRONMENT": "Development"` → this is app environment when debugging

--- 6. Trusting the self-signed certificate

1. Run to make sure everything is working fine → Go to **Run and Debug** → Select **.NET Core Launch (web)** (← this is auto enabled when you select Yes in the popup 'Add assets to build and debug') → Run





2. It just means we don't have the self signed certificate → Ok fine → Close that browser → Stop running (Shift-F5)

3. Go to Terminal → + new → dotnet dev-certs https --trust → Yes

--- 7. Exploring the default Swagger UI page

1. Go VSCode, Run again (F5) → https://localhost:5001/ says **This localhost page can't be found** → No webpage was found for the web address: https://localhost:5001/ → Okay → Go to https://localhost:5001/swagger/index.html → A nice swagger appears

2. Click on the api **Get** /WeatherForecast → **Try it out** → **Execute** → **Yeah** → Response payload with 200 status code

localhost:5001/swagger/index.html

Curl

```
curl -X GET "https://localhost:5001/WeatherForecast" -H  "accept: text/plain"
```

Request URL

```
https://localhost:5001/WeatherForecast
```

Server response

Code	Details
200	

Response body


```
[
  {
    "date": "2021-09-17T16:10:12.8740741+07:00",
    "temperatureC": 26,
    "temperatureF": 78,
    "summary": "Balmy"
  },
  {
    "date": "2021-09-18T16:10:12.874292+07:00",
    "temperatureC": -7,
    "temperatureF": 20,
    "summary": "Mild"
  },
  {
    "date": "2021-09-19T16:10:12.8742936+07:00",
    "temperatureC": 38,
    "temperatureF": 100,
    "summary": "Chilly"
  },
  {
    "date": "2021-09-20T16:10:12.8742938+07:00",
    "temperatureC": 45,
    "temperatureF": 112,
    "summary": "Bracing"
  },
  {
    "date": "2021-09-21T16:10:12.8742939+07:00",
```


Response headers

```
content-type: application/json; charset=utf-8
date: Thu16 Sep 2021 09:10:12 GMT
server: Kestrel
```

Responses

--- 8. Configuring VSCode Settings

1. Now we want to whenever F5 → it doesn't need to re-launch the browser again and again, we want it to keep opening. What will we do? → Stop running VS → Go to **.vscode/launch.json** → Remove the **"serverReadyAction"**

```
{} launch.json ×
.vscode > {} launch.json > Launch Targets > {} .NET Core Launch (web)
12          // If you have changed target frameworks, make sure to update the program path.
13          "program": "${workspaceFolder}/bin/Debug/net5.0/Catalog.dll",
14          "args": [],
15          "cwd": "${workspaceFolder}",
16          "stopAtEntry": false,
17          // Enable launching a web browser when ASP.NET Core starts. For more information: https://aka.ms/VSCode-CS-Lau
18          "serverReadyAction": {
19              "action": "openExternally",
20              "pattern": "\\bNow listening on:\\s+(https?://\\S+)"
21          },
22          "env": {
23              "ASPNETCORE_ENVIRONMENT": "Development"
```

2. Now run again → It starts the host but no browser opens → Go to https://localhost:5001/swagger/index.html → A nice swagger appears, hell yeah! → Ok stop it.

3. Now, how to **build our project in VSCode** → Go to **.vscode/tasks.json** → Add this attribute into "tasks"

```
"group": {
    "kind": "build",
    "isDefault": true
}
```

→ Save (Ctrls - S) → Go to **Terminal** → **Run Build Task…** (Ctrl-Shift- B)

--- 9. Commit and push code

1. Add .gitIgnore file follow instruction with this content
2. Remove folders: bin and obj
3. Commit code to Git!

II. ENTITY, REPOSITORY, CONTROLLER GET

--- 1. What is Entity, Repository, Controller?

- The **entities** are used to represent the **items** in your **catalog**
- The **repository** class responsible for all items, **storage** related operations
- The **controller** will handle all the requests sent to a REST API.

Now, you will learn:

→ How to model an entity via C sharp record types.
→ How to implement an in-memory repository of resources
→ How to implement a controller with GET routes to retrieve resources.

--- 2. Adding an entity

Create a new folder **Entities**. (Some people call it Domain/Module whatever). In it, create a new file **item.cs** → Code as below:

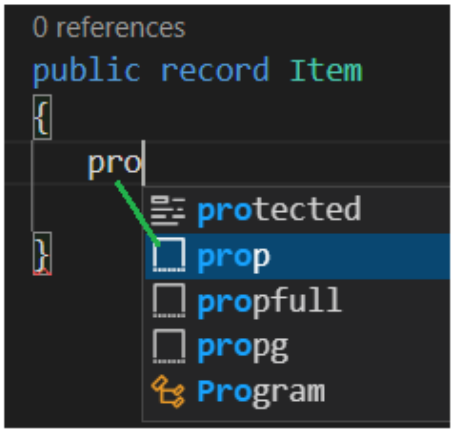
In C#9 and .NET5, we have **Record Types.**

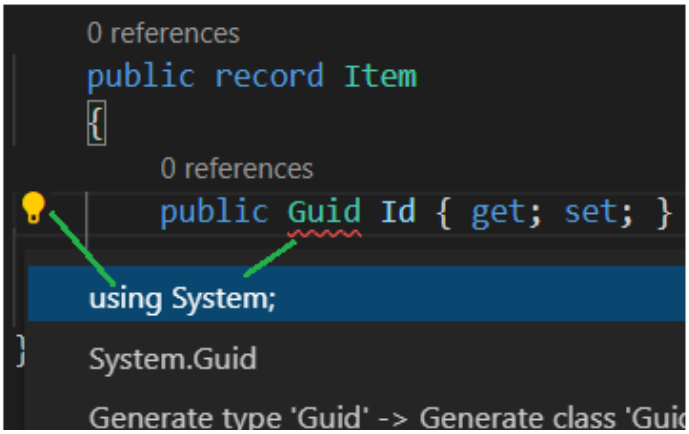
Record Types is very handy (rất là tiện)

- It is support for immutable object (đối tượng bất biến). Which means that once you get an instance of this object, you can't modify it.
- With-expressions support (diễn tả hỗ trợ)
- Value-based equality support → It means when you compare 2 instances of an item, those instances will be equal only if all of the properties of that instance are the same
    - E.g:
        - Item potion1 = new () {Name = "Potion", Price = 9};
        - Item potion2 = new () {Name = "Potion", Price = 9};
        - Item potion3 = potion1 with {Price = 15};

          → bool areEqual = potion1 == potion2; //**true**

          → bool areEqual = potion1 == potion3; //**false**
- Okay → let switch `public class Item` to `public record Item`

- Type short "prop" to add properties
- 



- Add Guid Id { get; set;} → (press **Ctrl .**) → using System;
- Change Guid Id { get; **set**;} to Guid Id { get; **init**;} Why? Because **init** is an annotation in C#9 and .Net5 and it is a great fix for property initializers.
- Ban đầu là Guid Id { get; **set**;}
- Sau đó vì object này là immutable nên → Guid Id { get; private **set**;} → để vậy khá challenge trong việc construct any set of these objects → Guid Id { get; **init**;}
- You can do this: Item item = new() { Id = Guid.NewGuid() };
- But not this: item.Id = Guid.NewGuid()
- Okay done for Items

```
Entities > Item.cs > {} Catalog.Entities
using System;

namespace Catalog.Entities
{
    0 references
    public record Item
    {
        0 references
        public Guid id { get; init; }
        0 references
        public string Name { get; init; }
        0 references
        public decimal Price { get; init; }
        0 references
        public DateTimeOffset CreatedDate { get; init; }
    }
}
```

--- 3. Adding a repository

Our repository, which is the class that is going to be in charge of storing the items in the system. To keep things simple For now, we will only use an **in-memory repository**. And a few episodes later in the tutorial, we will bring in a proper database.

Return an **IEnumerable** of item ← It is a basic interface that return a collection of items

```
namespace Catalog.Repositories
{
    0 references
    public class InMemItemsRepository
    {
        2 references
        private readonly List<Item> items = new()
        {
            new Item {Id = Guid.NewGuid(), Name = "Potion", Price = 9, CreatedDate = System.DateTimeOffset.Now },
            new Item {Id = Guid.NewGuid(), Name = "Iron Sword", Price = 20, CreatedDate = System.DateTimeOffset.Now },
            new Item {Id = Guid.NewGuid(), Name = "Bronze Shield", Price = 18, CreatedDate = System.DateTimeOffset.Now },
        };

        0 references
        public IEnumerable<Item> GetItems()
        {
            return items;
        }

        0 references
        public Item GetItem(Guid id)
        {
            return items.Where(item => item.Id == id).SingleOrDefault();
        }
    }
}
```

--- 4. Adding a controller

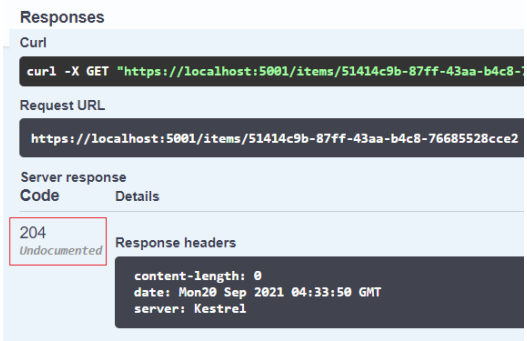
- Add ItemsController.cs kế thừa từ ControllerBase (import từ MS.AspNet.Core.Mvc)
- Mark the class as an [APIController] → it brings a bunch of additional default behaviors for your controller class, that will make our life easier.

- Declare [Route], the route defines which Http route the controller is going to be responding to. By default, we need to put name of controller. E.g just “items” → [Route(“items”)]

- Viết 2 methods như sau. Dễ hiểu mà
    - 1 cái để get toàn bộ list Item,
    - 1 cái là để get chỉ 1 item theo id thôi

```
namespace Catalog.Controllers
{

    [ApiController]
    [Route("items")]
    0 references
    public class ItemsController: ControllerBase
    {
        3 references
        private readonly InMemItemsRepository repository;

        0 references
        public ItemsController()
        {
            repository = new InMemItemsRepository();
        }

        //GET /items
        [HttpGet]
        0 references
        public IEnumerable<Item> GetItems()
        {
            var items = repository.GetItems();
            return items;
        }

        //GET /items/{id}
        [HttpGet("{id}")]
        0 references
        public Item GetItem (Guid id)
        {
            var items = repository.GetItem(id);
            return items;
        }
    }
}
```

- F5 chạy → Mở Swagger UI →
    - Try it out api List Item → Execute → 200 Success ok → Copy Id của 1 thằng
    - Try it out api get Item với Id copied → 204 Undocumented → sai sai



- Lí do: tại vì hiện tại cái ItemsController này lúc nào chạy nó cũng `repository = new InMemItemsRepository()` → nên lúc nào nó cũng sẽ tạo mới, mình copy thằng Id kia cho lần execute thứ 2 thì Id đó ko còn tồn tại nữa → null
- Vậy giờ mình muốn nếu Null thì nên trả về NotFound 404 chứ ko phải 204 Undocumented. Ok làm tiếp

--- 5. Debug

- Vẫn để F5 chạy bình thường như trên
- Đặt breakpoint trong đoạn code → bấm Execute trên Swagger của cái api get Item với Id copied hồi nãy → oh nó trả về Null → tại vì hiện tại cái ItemsController này lúc nào chạy nó cũng tạo mới `repository = new InMemItemsRepository()` → nên lúc nào nó cũng sẽ tạo mới, mình copy thằng Id kia cho lần execute thứ 2 thì nó ko còn tồn tại nữa → null

```
28
29          //GET /items/{id}
30          [HttpGet("{id}")]
            0 references
31          public Item GetItem (Guid id)
32          {                   null
33              var items = repository.GetItem(id);
34              return items;
35          }
```

- Fix bằng cách thêm code
    - Nếu items is null → trả về NotFound
    - Thay vì trả về mỗi 1 Item type → mình trả về ActionResult<Item> type để có thể trả về hơn 1 loại type, tức là có thể trả về NotFound type , hoặc trả về Item type

```
//GET /items/{id}
[HttpGet("{id}")]
0 references
public ActionResult<Item> GetItem (Guid id)
{
    var item = repository.GetItem(id);
    if (item is null)
    {
        return NotFound();
    }
    return item;
}
```

- F5 lại ok, lần này trả về đúng NotFound 404 rồi

```
Code        Details

404         Error:
Undocumented
            Response body
            {
              "type": "https://tools.ietf.
              "title": "Not Found",
              "status": 404,
              "traceId": "00-fa4393d3cf358
            }
```

- Còn việc thực sự Fix cái issue này ntn? Chúng ta sẽ dùng Dependency Injection học ở bài sau

III. DEPENDENCY, INJECTION, DTOs

You will learn...

- What is dependency injection?
- How to register and inject dependencies in .NET 5
- How to implement Data Transfer Objects (DTOs)
- How to map entities to DTOs

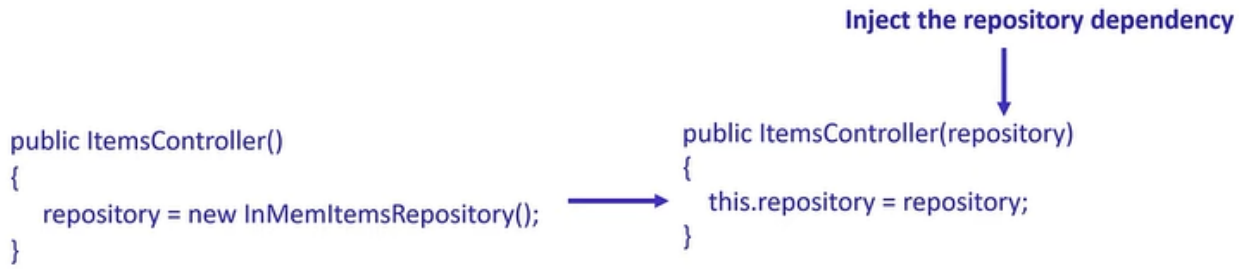
--- 1. What is Dependency?

- Ta có 1 Class, class này muốn dùng OtherClass → Ta gọi Class phụ thuộc vào OtherClass .
Vd: class ItemsController (là một contructor) nó đang tạo ra 1 instance (thực thể) InMemItemsRepository. Vậy InMemItemsRepository phải có trước cho ItemsController dùng. Vậy ta nói ItemsController phụ thuộc vào (depend on) InMemItemsRepository

```
public ItemsController()
{
    repository = new InMemItemsRepository();
}
```

--- 2. What is Dependency Injection?

- Giờ để dùng Dependency Injection, thì mình chỉnh ở trên một chút.
→ Thay vì để cho lớp ItemsController instructs cái instance đó, mình sẽ truyền Repository vào trong class constructor ItemsController, rồi sau đó tham chiếu (reference) vào chính cái class constructor đó luôn

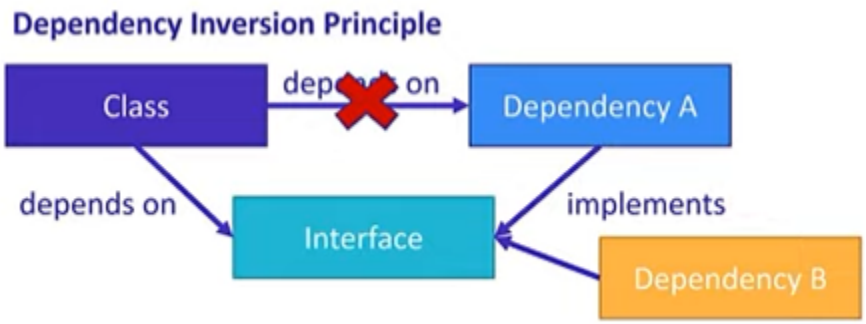


--- 3. What is the Dependency Inversion Principle?

Inversion = sự đảo ngược → công thức đảo ngược sự phụ thuộc.

- Ta có “Class” phụ thuộc vào “Dependency A” (ko có “Dependency A” là “Class” mệt mỏi)

- Nhưng mà giờ mình ko chơi cách dependency như ở trên nữa, tức là “Class” không còn phụ thuộc vào “Dependency A” nữa
- Thay vào đó “Class” sẽ phụ thuộc vào 1 abstraction (trừu tượng) và cụ thể ở đây là 1 Interface Và cái Interface đó là do “Dependency A”, “Dependency B”, “Dependency N” implements (thực hiện/triển khai)



--- 4. Why does it need?

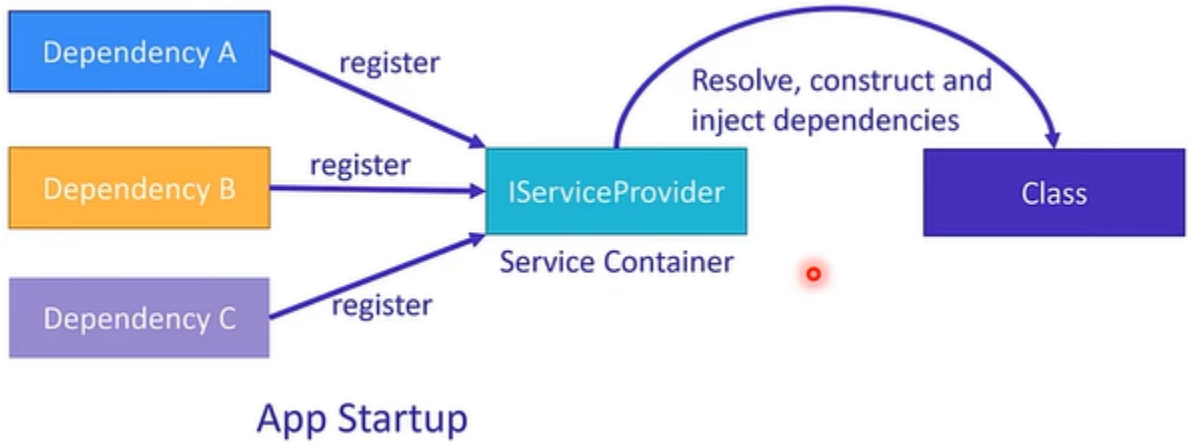
→ tách (decoupling) nhỏ sự phụ thuộc

→ code dễ đọc, dễ sử dụng, dễ bảo trì, dễ test hơn

Why?

- By having our code depend upon abstractions we are decoupling implementations from each other
- Code is cleaner, easier to modify and easier to reuse

--- 5. How to construct ?



1. -Go to class `InMemItemsRepository`
2. Bấm vào bóng đèn → Extract interface cho thằng InMemItemsRepository

```
namespace Catalog.Repositories
{
    2 references
    public class InMemItemsRepository

    Generate overrides...
    Generate Equals(object)...
    Generate Equals and GetHashCode...
    Generate constructor 'InMemItemsRepository()'
    Extract interface...
    Generate constructor...

new()

Name = "Potion", Price = 9, CreatedDate = D
Name = "Iron Sword", Price = 20, CreatedDat
Name = "Bronze Shield", Price = 18, Created
```

3. Nó sẽ generate ra cái này

```
namespace Catalog.Repositories
{
    1 reference
    public interface IInMemItemsRepository
    {
        1 reference
        Item GetItem(Guid id);
        1 reference
        IEnumerable<Item> GetItems();
    }

    2 references
    public class InMemItemsRepository : IInMemItemsRepository
    {
```

4. Tạo file IItemsRepository.cs trong folder Repositories
5. Copy cái interface generate ở 3, paste vào trong IItemsRepository.cs, nhớ using mấy cái lib thích hợp.
6. Go to ItemsController.cs đổi cái này

```
public class ItemsController : ControllerBase
{
    3 references
    private readonly InMemItemsRepository repository;

    0 references
    public ItemsController()
    {
        repository = new InMemItemsRepository();
    }
```

Thành cái này

```
public class ItemsController : ControllerBase
{
    3 references
    private readonly IItemsRepository repository;

    0 references
    public ItemsController(IItemsRepository repository)
    {
        this.repository = repository;
    }
```

7. Ok cuối cùng ta hoàn thành Dependency Injection theo mô hình sau

1. Interface

```
namespace Catalog.Repositories
{
    0 references
    public interface IItemsRepository
    {
        0 references
        Item GetItem(Guid id);
        0 references
        IEnumerable<Item> GetItems();
    }
}
```

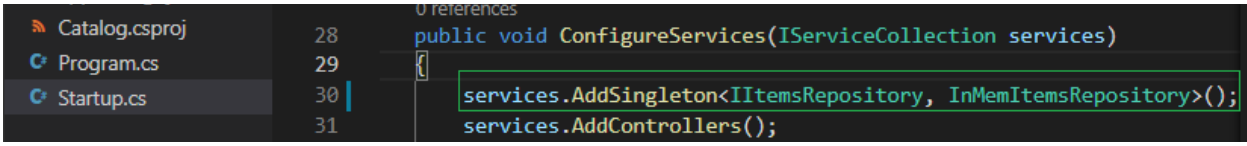
```
namespace Catalog.Repositories {

    0 references
    public class InMemItemsRepository : IItemsRepository
    {
        2 references
        private readonly List<Item> items = new()
        {
            new Item { Id = Guid.NewGuid(), Name = "Potion", Price =
            new Item { Id = Guid.NewGuid(), Name = "Iron Sword", Pric
            new Item { Id = Guid.NewGuid(), Name = "Bronze Shield", P
        };
```

2. Repository Implement Interface

```
namespace Catalog.Controllers
{

    [ApiController]
    [Route("items")]
    0 references
    public class ItemsController: ControllerBase
    {
        3 references
        private readonly IItemsRepository repository;

        0 references
        public ItemsController(IItemsRepository repository)
        {
            this.repository = repository;
        }
```

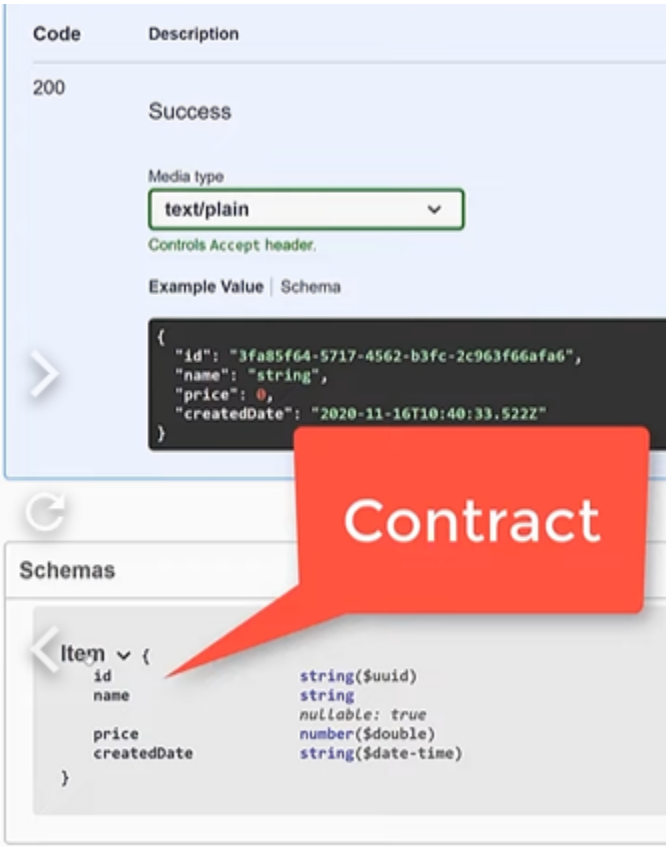
3. Controller injects the dependency through the interface calls to the repository

8. Ok giờ mình config cái Service Container như hình ở trên trên. Chỗ add startup á
    a. Singleton là gì? → Là nothing else than having one copy of the instance (of a type) across the entire lifetime of our service. So only once will be created. It will be reused whenever wherever it is need. Nó được khởi tạo 1 lần và được tái sử dụng mọi lúc mọi nơi mỗi khi cần → giải quyết vấn đề api getItem by Id trả về 404 Not found
    b. Go to Go to Startup.cs add dòng này vô



9. Ok, giờ F5 chạy lại 2 api get kia → ok hết bị, Code 200 return!

10. Tuy nhiên bây giờ có 1 vấn đề nữa là mình đang để lộ thông tin Item Entity ra bên ngoài, điều này nguy hiểm và đang vi phạm Contract. Khi mình build REST API, mình phải thiết lập (establish) 1 cái contract (hợp đồng) với clients như thế nào đó, để khi mình thêm xóa, sửa, field trong repository, những field đó ko được để lộ ra bên ngoài.



→ Do đó cần thiết lập DTOs

```
// GET /items
[HttpGet]
0 references
public IEnumerable<Item> GetItems()
{
    var items = repository.GetItems();
    return items;
}

// GET /items/{id}
[HttpGet("{id}")]
0 references
public ActionResult<Item> GetItem(Guid id)
{
    var item = repository.GetItem(id);

    if (item is null)
    {
        return NotFound();
```

Data Transfer Object (DTO)

--- 6. What is DTO?

- **Data Transfer Object** is nothing else other than the the actual **contract** that's going to be **enabled** between the client and our service
- Tạo Thư mục Dtos → Add file ItemDto.cs → copy class record Item qua như sau

```
using System;

namespace Catalog.Dtos
{
    0 references
    public record ItemDto
    {
        0 references
        public Guid Id { get; init; }
        0 references
        public string Name { get; init; }
        0 references
        public decimal Price { get; init; }
        0 references
        public DateTimeOffset CreatedDate { get; set; }
    }
}
```

- Nhìn thì có vẻ dư thừa, nhưng benefit của nó thì về sau sẽ rất rõ ràng khi mà mình chỉnh sửa database. Mình không phải đụng vào contract, mình có thể sẽ rất cẩn thận với contract or supposed to be breaking our clients anytime.
- Ok sau đó mình vào ItemsController.cs update lại api get như sau

```
//GET /items
[HttpGet]
0 references
public IEnumerable<Item> GetItems()
{
    var items = repository.GetItems();
    return items;
}
```

```
//GET /items/{id}
[HttpGet("{id}")]
0 references
public ActionResult<Item> GetItem (Guid id)
{
    var item = repository.GetItem(id);
    if (item is null)
    {
        return NotFound();
    }
    return item;
}
```

```
//GET /items
[HttpGet]
0 references
public IEnumerable<ItemDto> GetItems()
{
    var items = repository.GetItems().Select(item => new ItemDto
    {
        Id = item.Id,
        Name = item.Name,
        Price = item.Price,
        CreatedDate = item.CreatedDate
    });
    return items;
}
```

- Giờ tất nhiên phải update cho api thứ 2 rồi, mà làm từng cái vậy cực quá, giờ viết 1 cái extension luôn nha
- Tạo file Extension.cs như sau, nhớ Extension luôn đi với Static class

```
Extension.cs > {} Catalog
using Catalog.Dtos;
using Catalog.Entities;

namespace Catalog
{
    0 references
    public static class Extensions
    {
        0 references
        public static ItemDto AsDto(this Item item )
        {
            return new ItemDto
            {
                Id = item.Id,
                Name = item.Name,
                Price = item.Price,
                CreatedDate = item.CreatedDate
            };
        }
    }
}
```

Method AsDto này sẽ nhận 1 cái biến tên là item với type là Item được truyền vào, chữ **this** có nghĩa là cái item được truyền vào nhờ method AsDto đó trả về type ItemDto luôn

- Sau đó 2 api get kia được updated lại với AsDto như sau

```
//GET /items
[HttpGet]
0 references
public IEnumerable<Item> GetItems()
{
    var items = repository.GetItems();
    return items;
}

//GET /items/{id}
[HttpGet("{id}")]
0 references
public ActionResult<Item> GetItem (Guid id)
{
    var item = repository.GetItem(id);
    if (item is null)
    {
        return NotFound();
    }
    return item;
}
```

```
//GET /items
[HttpGet]
0 references
public IEnumerable<ItemDto> GetItems()
{
    var items = repository.GetItems().Select(item => item.AsDto()
    return items;
}

//GET /items/{id}
[HttpGet("{id}")]
0 references
public ActionResult<ItemDto> GetItem (Guid id)
{
    var item = repository.GetItem(id);
    if (item is null)
        return NotFound();
    return item.AsDto();
}
```

- Ok, giờ F5 chạy lại 2 api get kia → ok giờ Schemas (lược đồ) không còn lộ **Item** nữa, mà là **ItemDto**, ủa chỉ là đổi tên thôi hả ??? → Người giảng nói là “The contract we're exposing to our consumers is no longer an item but it is itemDto with the properties posted right here.”, xong hết bài ← chỗ này hơi thiệt là khó hiểu
-

Code	Description
200	Success

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
{
  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "name": "string",
  "price": 0,
  "createdDate": "2021-09-20T10:25:10.105Z"
}
```

Schemas

```
ItemDto ⌄ {
    id                  string($uuid)
    name                string
                        nullable: true
    price               number($double)
    createdDate         string($date-time)
}
```

IV. POST, PUT, DELETE

You will learn...

- How to create resources with POST
- How to validate the values of DTO properties
- How to update resources with PUT

--- 1. Create Post, Put, Delete

- Go to interface `IItemsRepository`, add the code for CreateItem and UpdateItem

```
public interface IItemsRepository
{
    1 reference
    Item GetItem(Guid id);
    1 reference
    IEnumerable<Item> GetItems();
    1 reference
    void CreateItem(Item item);
    0 references
    void UpdateItem(Item item);
}
```

- Go to interface `InMemItemsRepository`, implement interface

```
namespace Catalog.Repositories {

    1 reference
    public class InMemItemsRepository : IItemsRepository
    {
        2 references
        private readonly List<Item> item
        {
```

Implement interface
Implement all members explicitly
Implement remaining members explicitly

it generates this code

```
0 references
public void CreateItem(Item item)
{
    throw new NotImplementedException();
}
```

- Update the methods CreateItem, UpdateItem to

```
1 reference
public void CreateItem(Item item)
{
    items.Add(item) ;
}

1 reference
public void UpdateItem(Item item)
{
    var index = items.FindIndex(existingItem => existingItem.Id == item.Id);
    items[index] = item;

}
```

- In Dtos folder, create **CreateItemDto**.cs, **UpdateItemDto**.cs

```
namespace Catalog.Dtos
{
    1 reference
    public record CreateItemDto
    {
        [Required]
        1 reference
        public string Name { get; init; }

        [Required]
        [Range (1, 1000)]
        1 reference
        public decimal Price { get; init; }

    }
}
```

```
namespace Catalog.Dtos
{
    1 reference
    public record UpdateItemDto
    {
        [Required]
        1 reference
        public string Name { get; init; }

        [Required]
        [Range (1, 1000)]
        1 reference
        public decimal Price { get; init; }

    }
}
```

- Go to Controller `ItemsController`, add **Post** and **Put** routes, remember again, we use **ActionResult** because it can return more than 1 type

```
//GET /items/
[HttpPost]
0 references
public ActionResult<ItemDto> CreateItem (CreateItemDto itemDto)
{
    Item item = new() {
        Id = Guid.NewGuid(),
        Name = itemDto.Name,
        Price = itemDto.Price,
        CreatedDate = DateTimeOffset.UtcNow
    };
    repository.CreateItem(item);
    return CreatedAtAction(nameof(GetItem), new {id = item.Id}, item.AsDto());
}

//PUT /items/{id}
[HttpPut("{id}")]
0 references
public ActionResult<ItemDto> UpdateItem(Guid id, UpdateItemDto itemDto)
{
    var existingItem = repository.GetItem(id);
    if(existingItem is null)
    {
        return NotFound();
    }
    Item updatedItem = existingItem with
    {
        Name = itemDto.Name,
        Price = itemDto.Price
    };
    repository.UpdateItem(updatedItem);
    return NoContent();
}
```

With - expression: Method **UpdateItem** ở trên đơn giản là mình khai báo 1 Item mới tên là **updatedItem** với bản chất là copy từ **existingItem** rồi sau đó update thuộc tính Name, Price của nó

- Add Delete apis

IItemsRepository

```
4 references
public interface IItemsRepository
{
    2 references
    Item GetItem(Guid id);
    1 reference
    IEnumerable<Item> GetItems();
    1 reference
    void CreateItem(Item item);
    1 reference
    void UpdateItem(Item item);
    0 references
    void DeleteItem(Guid id);
}
```

InMemItemsRepository

```
1 reference
public void DeleteItem(Guid id)
{
    var index = items.FindIndex(existingItem => existingItem.Id == id);
    items.RemoveAt(index);
}
```

ItemsController

```
//DELETE /items/{id}
[HttpDelete("{id}")]
0 references
public ActionResult DeleteItem(Guid id) {
    var existingItem = repository.GetItem(id);
    if(existingItem is null)
    {
        return NotFound();
    }
    repository.DeleteItem(id);
    return NoContent();
}
```

- F5, mở Swagger UI chạy **Post**, **Update**, **Delete** → Ok

V. PERSISTING ENTITIES WITH MONGODB

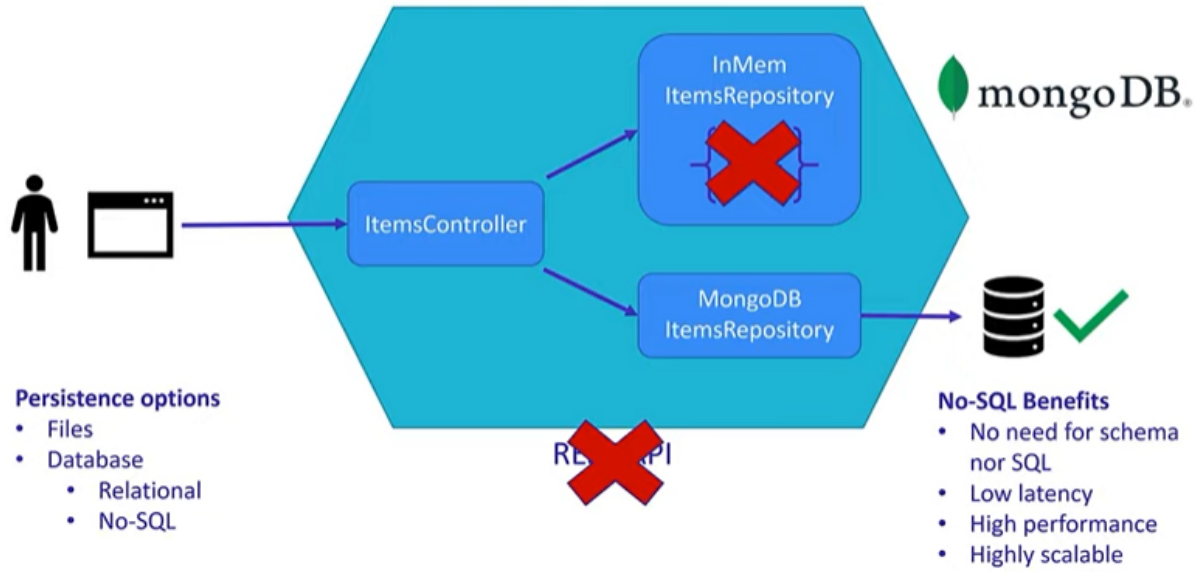
You will learn...

- How to implement a simple MongoDB repository
- How to run MongoDB as a docker container
- How to use Postman to interact with a REST API

You will need...

- Docker (https://docs.docker.com/get-docker)
- Postman (https://www.postman.com/downloads)

Adding Persistence



--- 1. Use Postman

Open postman, when VS F5 → Microsoft.Hosting

- → URL: https://localhost:5001/items
- → URL: http://localhost:5000/items

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\5.0.1\System.Net.Name
enabled.
Microsoft.Hosting.Lifetime: Information: Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
Microsoft.Hosting.Lifetime: Information: Now listening on: http://localhost:5000
```

```
namespace Catalog.Controllers
{
    [ApiController]
    [Route("items")]
    0 references
    public class ItemsController: Control
    {
```

GET http://localhost:5000/items

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (4) Test Results Status: 200 OK Time: 80 ms Size: 527 B

Pretty Raw Preview Visualize JSON

```
1  [
2      {
3          "id": "e840817f-e954-4c93-b1c1-d040c5466fb9",
4          "name": "Potion",
5          "price": 9,
6          "createdDate": "2021-09-21T15:53:17.397599+07:00"
7      },
```

GET http://localhost:50... GET https://localhost:5...

https://localhost:5001/items

GET https://localhost:5001/items

--- 2. Connect MongoDB

- Create file MongoDbItemsRepository → Ctrl . → implement interface

```
using System;
using System.Collections.Generic;
using Catalog.Entities;

namespace Catalog.Repositories
{
    0 references
    public class MongoDbItemsRepository : IItemsRepository
    {
        1 reference
        public void CreateItem(Item item)
        {
            throw new NotImplementedException();
        }

        1 reference
        public void DeleteItem(Guid id)
        {
            throw new NotImplementedException();
        }

        3 references
        public Item GetItem(Guid id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public IEnumerable<Item> GetItems()
        {
            throw new NotImplementedException();
        }

        1 reference
        public void UpdateItem(Item item)
        {
            throw new NotImplementedException();
        }
    }

}
```

Repositories
- IItemsRepository.cs
- InMemItemsRepository.cs
- MongoDbItemsRepository.cs

- Add Nuget package **MongoDB.Driver** →

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csh
arp\Catalog> dotnet add package MongoDB.Driver
```

- See it → OK

```
Catalog.csproj
1   <Project Sdk="Microsoft.NET.Sdk.Web">
2
3     <PropertyGroup>
4       <TargetFramework>net5.0</TargetFramework>
5     </PropertyGroup>
6
7     <ItemGroup>
8       <PackageReference Include="Microsoft.AspNetCore.Authentic
9       <PackageReference Include="Microsoft.AspNetCore.Authentic
10      <PackageReference Include="MongoDB.Driver" Version="2.13.
11      <PackageReference Include="Swashbuckle.AspNetCore" Versic
12    </ItemGroup>
13
14  </Project>
15
```

- Then we need DB Name and Collection Name. Usually, all documents are going to be grouped into collections. And we can have 1 or more collections in DB

```
private const string databaseName = "catalog";
private const string collectionName = "items";
```

- Code 1 tí cho CreateItem như sau:

```
namespace Catalog.Repositories
{
    1 reference
    public class MongoDbItemsRepository : IItemsRepository
    {
        1 reference
        private const string databaseName = "catalog";
        1 reference
        private const string collectionName = "items";
        2 references
        private readonly IMongoCollection<Item> itemsCollection;

        0 references
        public MongoDbItemsRepository(IMongoClient mongoClient)
        {
            IMongoDatabase database = mongoClient.GetDatabase(databaseName);
            itemsCollection = database.GetCollection<Item>(collectionName);
        }

        1 reference
        public void CreateItem(Item item)
        {
            itemsCollection.InsertOne(item);
        }

        1 reference
        public void DeleteItem(Guid id)
        {
            throw new NotImplementedException();
        }

        3 references
        public Item GetItem(Guid id)
        {
            throw new NotImplementedException();
        }

        1 reference
        public IEnumerable<Item> GetItems()
        {
            throw new NotImplementedException();
        }

        1 reference
        public void UpdateItem(Item item)
        {
```

- Bây giờ Làm sao để kết nối MongoDB vào VS của mình ?
    - Cách 1: Install the DB via the MongoDB Installer ← ko đi hướng này
    - Cách 2: Run the BD as a part of a Docker container ← we go this way
- Nhắc lại về Docker Image → là 1 standalone software including everything needed to run an application (MongoBD) → Nói cách khác là everything is packaged in this Docker Image → When we run/execute this Docker Image → it becomes what we call a Docker Container → so is running instance of a Docker Image → that docker container is going to run in the Docker engine

- Làm sao để kết nối Docker Engine vào VS của mình ?
    - Cài Docker → xong
    - Bật Docker Desktop lên → nhớ bật thì mới execute lệnh dưới được
    - Terminal → run : **docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db mongo**
        - rm là remove, nghĩa là container sẽ bị destroyed after we close the process
        - -p: nghĩa là mình open 1 cái port ở local machine của mình
        - Port bên trái **27017** là port ở local machine của mình, khai báo để nó can be mapped into the MongoDB port inside the Docker container. Mình có thể chọn any other port nào cũng được
        - Port bên phải **27017**: là port của MongoBD, bắt buộc phải là **27017**
        - **-v mongodbdata:** v nghĩa là Volume, so that you don't lose the data that has been stored in MongoDB when you stop the docker container. Nếu ko khai báo thì sẽ mất all data as you start getting the content. **/data/db** is the usual location where MongoDB is going to store the data inside the container. Tức là mình sẽ nói nó là ê, tao sẽ map cái location tên là **mongodbdata** from our local machine into thư mục **/data/db** directory inside the Docker container.
        - **Mongo:** là tên của the image

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db mongo
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
35807b77a593: Downloading [================================>                  ]  19.11MB/28.57MB
```

mongo	IN USE	latest	ccf4b4ee3bee	about 12 hours ago	684.93 MB

- Xong rồi thì kiểm tra bằng: **docker ps**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db mongo
3145095b6d673146d127040429d98fea4f1249942dd9c44809ec0b49adca6e0f
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                           NAMES
3145095b6d67   mongo     "docker-entrypoint.s…"   9 seconds ago    Up 6 seconds    0.0.0.0:27017->27017/tcp, :::27017->27017/tcp   mongo
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

- Ok, bây giờ thì mình config một tí để chuyển database từ InmemRepo to MongoDB

- Vào appsetting.json add Host and Port

```
appsettings.json > ...
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft": "Warning",
6        "Microsoft.Hosting.Lifetime": "Information"
7      }
8    },
9    "AllowedHosts": "*",
10   "MongoDbSettings" : {
11     "Host" : "localhost",
12     "Port" : "27017"
13   }
14 }
15
```

- Tạo thư mục Settings → tạo file MongoDbSettings.cs

```
namespace Catalog.Settings
{
    0 references
    public class MongoDbSettings
    {
        1 reference
        public string Host { get; set; }
        1 reference
        public int Port { get; set; }

        0 references
        public string ConnectionString
        {
            get
            {
                return $"mongodb://{Host}:{Port}";
            }
        }
    }
}
```

- Go to Startup.cs, update

- Một điểm cần lưu ý là cái entity Item

```
22 references
public record Item
{
    10 references
    public Guid Id { get; init; }

    6 references
    public string Name { get; init; }

    6 references
    public decimal Price { get; init; }

    5 references
    public DateTimeOffset CreatedDate { get; init; }

}
```

- mình khai báo Id và DateTimeOffset như trên, ở Inmem thì nó work bình thường, nhưng với MongoDB thì mình cần phải serialize (sắp đặt tuần tự), và phải chỉnh sửa 1 chút, nếu ko thì MongoDB nó sẽ trả về nhìn ko có human friendly lắm → dùng BsonSerializer của MongoDB

```
public void ConfigureServices(IServiceCollection services)
{
    BsonSerializer.RegisterSerializer(new GuidSerializer(BsonType.String));
    BsonSerializer.RegisterSerializer(new DateTimeOffsetSerializer(BsonType.String));
    services.AddSingleton<IMongoClient>(serviceProvider =>
    {
        var settings = Configuration.GetSection(nameof(MongoDbSettings)).Get<MongoDbSettings>();
        return new MongoClient(settings.ConnectionString);

    });
    // We don't use InMemItemsRepository, so comment this
    // services.AddSingleton<IItemsRepository, InMemItemsRepository>();

    // We switch to MongoDb
    services.AddSingleton<IItemsRepository, MongoDbItemsRepository>();

    services.AddControllers();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "Catalog", Version = "v1" });
    });
}
```

- F5 → Mở Postman chạy thử
- POST → tự điền Jason → 201 Created

POST https://localhost:5001/items

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "name": "Búa",
3     "price" : 50
4 }
```

Body Cookies Headers (5) Test Results Status: 201 Created

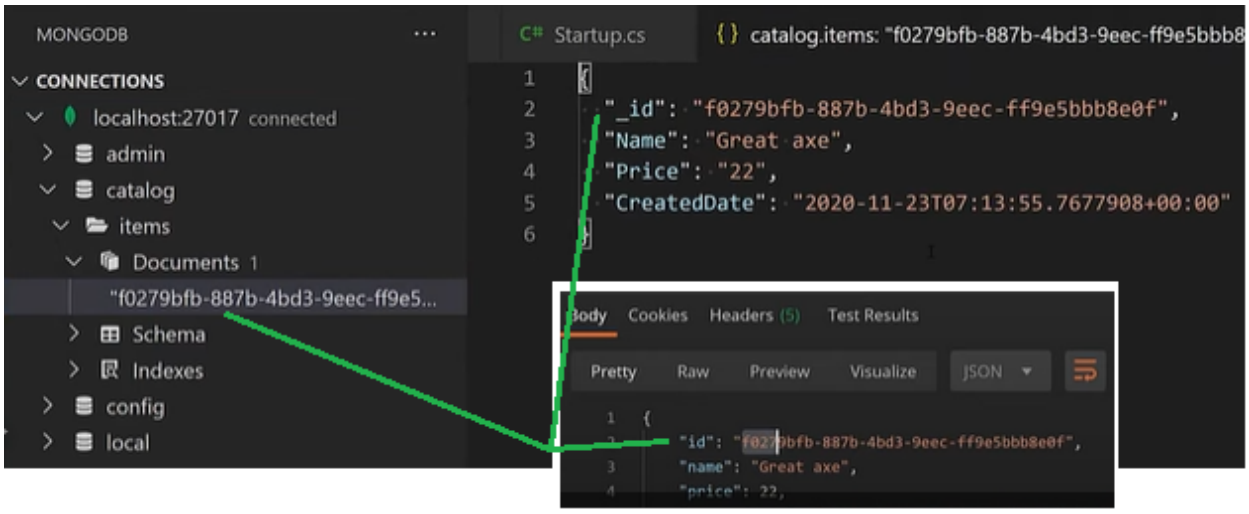
Pretty Raw Preview Visualize JSON

```
1 {
2     "id": "d05ac86b-5176-433d-a7b1-fc5161918d35",
3     "name": "Búa",
4     "price": 50,
5     "createdDate": "2021-09-22T01:51:22.5691273+00:00"
6 }
```

Header

Location	https://localhost:5001/items/d05ad35

- Ko biết cái item trên acutally looks like trong database như thế nào nhỉ → Qua VS → stop F5 → cài MongoDB for VS Code extension → Sẽ thấy chiếc lá ở menu của MongoDB hình chiếc lá ở bên trái → bấm vô, chọn Advanced Connenction → hostname = localhost, port = 27017 → bấm connect --> sẽ thấy Connections = localhost: 27017 → nếu connection failed thì right click → remove connection → add connection lại → hostname = localhost, port = 27017 → lúc này chưa dùng Authen → cứ để None → connect → xong check id của cái item mới tạo trung với id trong document là ok!



- Ok, vậy là xong **CreateItem** trong file **MongoDbItemsRepository**
- Bây giờ là tới **GetItem**
    - **Get list** → Code như sau → F5 → Postman → OK

```
1 reference
public IEnumerable<Item> GetItems()
{
    return itemsCollection.Find(new BsonDocument()).ToList();
}
```

- **Get theo ID** → Đối với MongoDB thì mình phải dùng **Filter Definition Builder** → để filter những cái items mà mình muốn collection trả về → Khai báo như sau

```
private readonly FilterDefinitionBuilder<Item> filterBuilder = Builders<Item>.Filter;
```

```
public Item GetItem(Guid id)
{
    var filter = filterBuilder.Eq(item => item.Id, id);
    return itemsCollection.Find(filter).SingleOrDefault();
}
```

- F5 → chạy PM ok nha

- Bây giờ là tới **UpdateItem**

```
public void UpdateItem(Item item)
{
    var filter = filterBuilder.Eq(exsitingItem => exsitingItem.Id, item.Id);
    itemsCollection.ReplaceOne(filter, item);
}
```

- F5 → chạy PM → PUT → ok nha
- Nhớ là với PUT đồng nghĩa với việc mình XÓA cái item đó và TẠO LẠI → đòi hỏi phải khai táo toàn bộ field của nó trong Json
- PATCH là vá từng field của item đó, mình ko cần phải khai báo toàn bộ field như PUT
- Bây giờ là tới **DeleteItem**

```
public void DeleteItem(Guid id)
{
    var filter = filterBuilder.Eq(item => item.Id, id);
    itemsCollection.DeleteOne(filter);
}
```

- F5 → chạy PM → DELETE→ ok nha
- Lời Kết:
  - We don’t need to touch our items controller at all.
  - The only things that we really need here:
    - Adding a few configuration
    - Registering the MongoDB Client
    - Create a MongoDbItemsRepository that was plugged into the service and that is able to by itself do all the logic of interacting with MongoDB, the

rest of the service has not changed at all → That's a great benefit that we get from Dependency Injection and in this case, the repository pattern

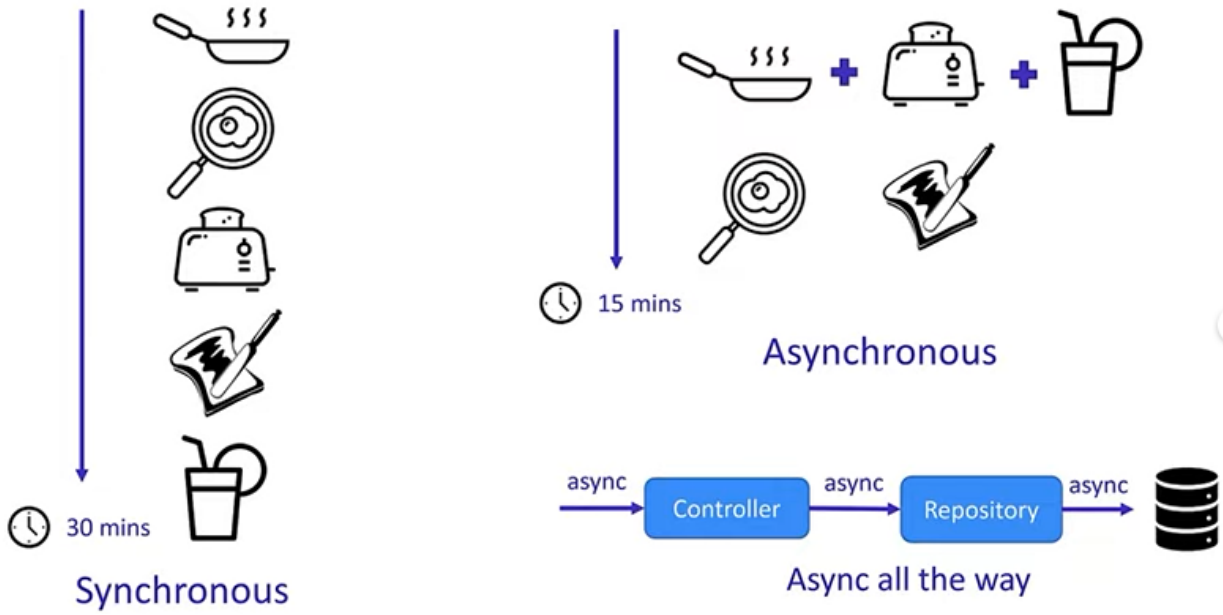
VI. TASKS, ASYNC AND WAIT

You will learn...

- What is the asynchronous programming model
- How to use tasks, async and await to add asynchronous programming to your REST API

  - Vd: You prepare your breakfast → you're going to do a bunch of tasks
  - Instead of do that bunch of tasks sequentially (take you 30min), you will do them in parallel order (take you 15min)
  - Synchronous (adj) = (syn-crô-nợt) = khởi phát đồng thời = đồng bô

Asynchronous programming



- Async: mình không phải chờ mỗi every single task to complete before starting another one → mình sẽ start when we can, then go back to previous task when it's time to do so.
- **Async all the way** provide a lot of performance and efficiency (ef-fíc-sần-sì = hiệu quả) to the execution of your code

<u>Giải thích Async all the way như trên hình:</u>

- Bây giờ mình có 1 **MongoDB** database ← mình có 1 **Repository** class tương tác với cái DB đó. {*Tuy nhiên, việc tương tác với database là một expensive operation bởi vì mình phải perform input, output. Mình have to go over-the-wire (thành ngữ gì đó ko hiểu) and talk to the database, that database may take time to give you back results depend on where it is, how much load at that point in time. → You DON'T have to wait for the database to finish the work → what you have to do IS instead of doing a* ***Synchronous*** *call to the database. You are going to do* ***Asynchronous*** *call to the database.*}
- Rồi mình có **Controller** talking to the **Repository**, rồi mình có whatever talking to **Controller** cũng đều áp dụng ***Asynchronous***
-

--- 1. Refactor to use the Async

- Open ItemRepository.cs, rename symbol (F2) bằng cách add suffix (hậu tố) Async cho tất cả method
- Make sure each of the methods return TASK. This is because that is a way to signal that that is not going to be a ***Synchronous*** method anymore. But it is going to be an ***Asynchronous*** method. Task represents (đại diện cho) an ***Asynchronous*** operation that eventually is going to return an item whenever we have finished retrieving that item from the Database
- From

```
namespace Catalog.Repositories
{
    5 references
    public interface IItemsRepository
    {
        3 references
        Item GetItem(Guid id);
        1 reference
        IEnumerable<Item> GetItems();
        1 reference
        void CreateItem(Item item);
        1 reference
        void UpdateItem(Item item);
        1 reference
        void DeleteItem(Guid id);
    }
}
```

- To

```
public interface IItemsRepository
{
    3 references
    Task<Item> GetItemAsync(Guid id);
    1 reference
    Task<IEnumerable<Item>> GetItemsAsync();
    1 reference
    Task CreateItemAsync(Item item);
    1 reference
    Task UpdateItemAsync(Item item);
    1 reference
    Task DeleteItemAsync(Guid id);
}
```

- Now, mở file **MongoDbItemsRepository,** update các method **void** → **Task, đổi InsertOne** → **InsertOneAsync,** và thêm keywords **async** và **await** như sau

```
public void CreateItemAsync(Item item)
{
    itemsCollection.InsertOne(item);
}
```

```
public async Task CreateItemAsync(Item item)
{
    await itemsCollection.InsertOneAsync(item);
}
```

Update toàn bộ method luôn nha, ta có

```
public async Task CreateItemAsync(Item item)
{
    await itemsCollection.InsertOneAsync(item);
}

1 reference
public async Task DeleteItemAsync(Guid id)
{
    var filter = filterBuilder.Eq(item => item.Id, id);
    await itemsCollection.DeleteOneAsync(filter);
}

3 references
public async Task<Item> GetItemAsync(Guid id)
{
    var filter = filterBuilder.Eq(item => item.Id, id);
    return await itemsCollection.Find(filter).SingleOrDefaultAsync();
}

1 reference
public async Task <IEnumerable<Item>> GetItemsAsync()
{
    return await itemsCollection.Find(new BsonDocument()).ToListAsync();
}

1 reference
public async Task UpdateItemAsync(Item item)
{
    var filter = filterBuilder.Eq(exsitingItem => exsitingItem.Id, item.Id);
    await itemsCollection.ReplaceOneAsync(filter, item);
}
```

- Now, mở file **InMemItemsRepository,** file này có thể xóa vì giờ mình đi hướng MongoDB rồi, nhưng giữ lại để tham khảo cũng được, bây giờ mình tương tự cũng update Task, async, await cho nó.

```
public async Task<IEnumerable<Item>> GetItemsAsync()
{
    return await Task.FromResult(items);
}

3 references
public async Task<Item> GetItemAsync(Guid id)
{
    var item = items.Where(item => item.Id == id).SingleOrDefault();
    return await Task.FromResult(item);
}

1 reference
public async Task CreateItemAsync(Item item)
{
    items.Add(item) ;
    await Task.CompletedTask;
}

1 reference
public async Task UpdateItemAsync(Item item)
{
    var index = items.FindIndex(existingItem => existingItem.Id == item.Id);
    items[index] = item;
    await Task.CompletedTask;
}

1 reference
public async Task DeleteItemAsync(Guid id)
{
    var index = items.FindIndex(existingItem => existingItem.Id == id);
    items.RemoveAt(index);
    await Task.CompletedTask;
}
```

- Now, mở file **ItemsController,** tương tự cũng update Task, async, await cho nó.

```
//GET /items
[HttpGet]
0 references
public async Task<IEnumerable<ItemDto>> GetItemsAsync()
{
    var items = (await repository.GetItemsAsync())
        .Select(item => item.AsDto());
    return items;
}
```

```
//GET /items/{id}
[HttpGet("{id}")]
1 reference
public async Task<ActionResult<ItemDto>> GetItemAsync (Guid id)
{
    var item = await repository.GetItemAsync(id);
    if (item is null)
        return NotFound();
    return item.AsDto();
}
```

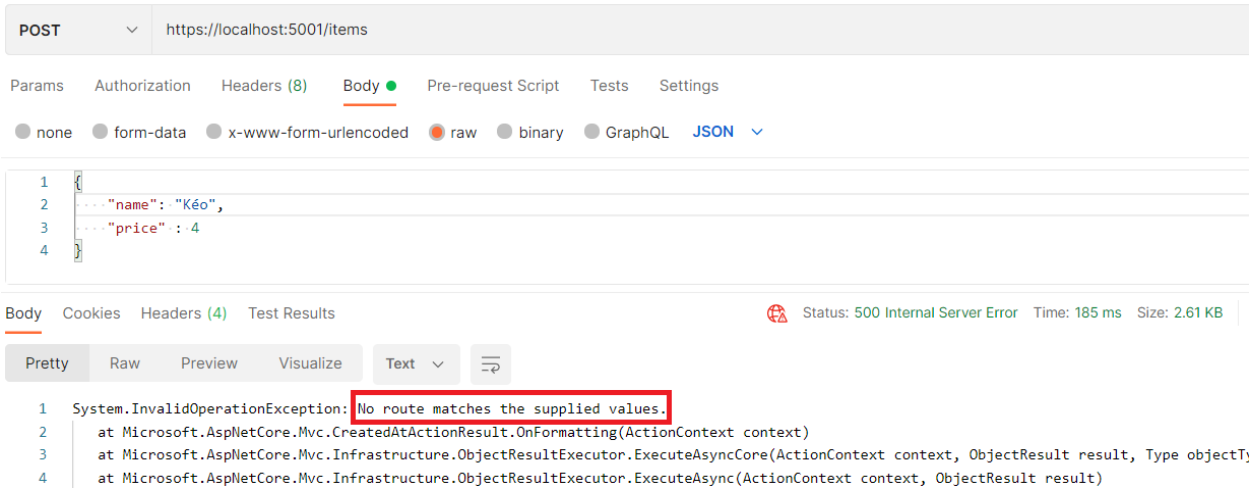
```
//POST /items/
[HttpPost]
0 references
public async Task<ActionResult<ItemDto>> CreateItemAsync (CreateItemDto itemDto)
{
    Item item = new() {
        Id = Guid.NewGuid(),
        Name = itemDto.Name,
        Price = itemDto.Price,
        CreatedDate = DateTimeOffset.UtcNow
    };
    await repository.CreateItemAsync(item);
    return CreatedAtAction(nameof(GetItemAsync), new {id = item.Id}, item.AsDto());
}
```

```
//PUT /items/{id}
[HttpPut("{id}")]
0 references
public async Task<ActionResult<ItemDto>> UpdateItemAsync(Guid id, UpdateItemDto itemDto)
{
    var existingItem = await repository.GetItemAsync(id);
    if(existingItem is null)
    {
        return NotFound();
    }
    Item updatedItem = existingItem with
    {
        Name = itemDto.Name,
        Price = itemDto.Price
    };
    await repository.UpdateItemAsync(updatedItem);
    return NoContent();
}
```

```
//DELETE /items/{id}
[HttpDelete("{id}")]
0 references
public async Task<ActionResult> DeleteItemAsync(Guid id) {
    var existingItem = await repository.GetItemAsync(id);
    if(existingItem is null)
    {
        return NotFound();
    }
    await repository.DeleteItemAsync(id);
    return NoContent();
}
```

--- 2. Testing async methods in Postman

- Now, mở Postman,
    - Run api GET items → Ok
    - Run api POST → Lỗi



- Lí do là method CreateItemAsync gọi nameof(GetItemAsync) nó ngược với GetItem lúc trước khi update

```
// POST /items
[HttpPost]
0 references
public async Task<ActionResult<ItemDto>> CreateItemAsync(CreateItemDto itemDto)
{
    Item item = new()
    {
        Id = Guid.NewGuid(),
        Name = itemDto.Name,
        Price = itemDto.Price,
        CreatedDate = DateTimeOffset.UtcNow
    };

    await repository.CreateItemAsync(item);

    return CreatedAtAction(nameof(GetItemAsync), new { id = item.Id }, item.AsDto
}
```

- Fix bằng cách, mở VS, mở Startup file, update dòng code này

```
services.AddControllers();
```

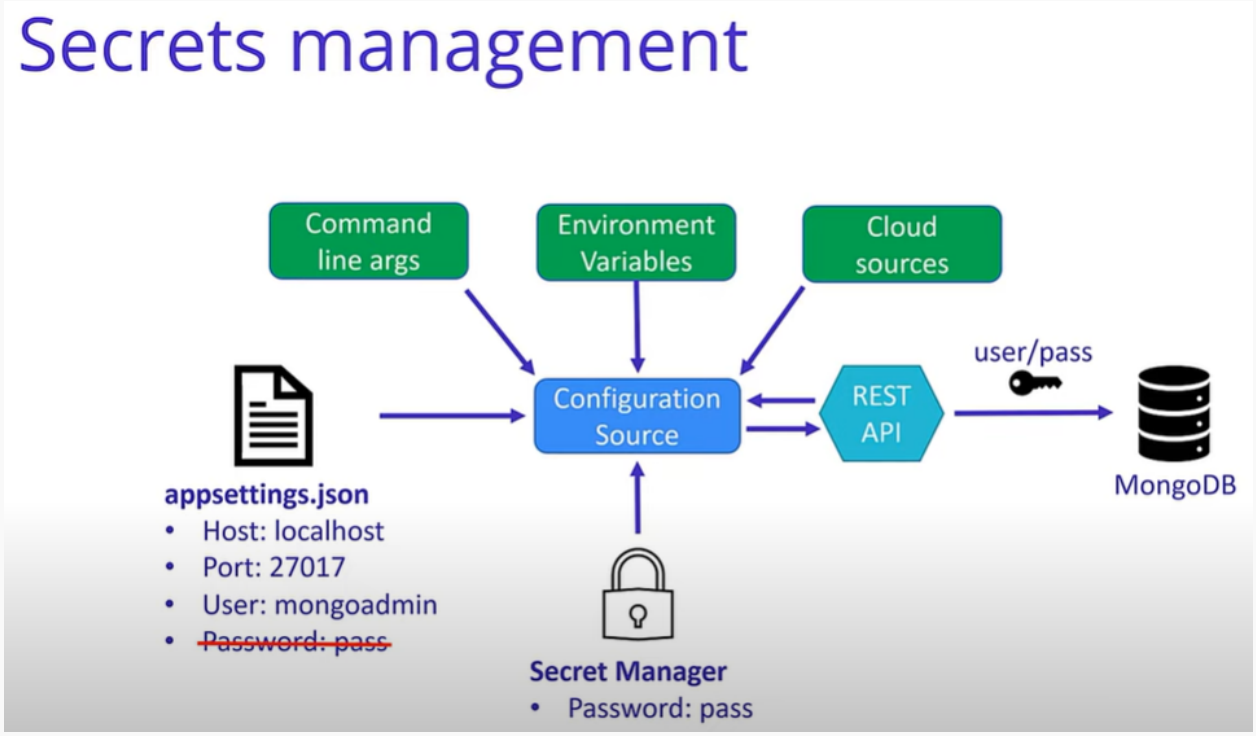
```
services.AddControllers(options =>
{
    options.SuppressAsyncSuffixInActionNames = false;
});
```

- Now, F5 → mở Postman,
    - Run api GET items → Ok
    - Run api POST → Ok
    - Run api GET item by Id → Ok
    - Run api PUT item by Id → Ok
    - Run api DELETE item by Id → Ok
- → trong VS, mở MongoDB chiếc lá, refresh document → items show ok!

VII. SECRETS AND HEALTH CHECKS

You will learn...

- How to store and use secrets via the .NET Secret Manager
- How to use health checks to report the health of the rest API and its dependencies



--- 1. Re-create MongoDB with authentication

- Remember we do this way on **development** database for learning purposes, we will **delete the volume** where Mongo container currently storing all the data ← DON'T DO THIS IN PROD. This allow us to start a new container with a brand new Docker volume with authentication enabled
- 1. Đầu tiên mình phải remove cái Database cũ, vì nó ko có authentication
    - Check container is ready or not → terminal → run: **docker ps**
    - If it is on → turn it off → run: **docker stop mongo**
    - List out volumes that docker using → run: **docker volume ls** → thấy 1 thằng **mongodbdata**
    - Delete it by running: **docker volume rm mongodbdata**
- 2. Giờ tạo lại Database mới với authentication
    - run: **docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db -e MONGO_INITDB_ROOT_USERNAME=mongoadmin -e MONGO_INITDB_ROOT_PASSWORD=Pass#word1 mongo** → thành công thì nó trả về một bộ gồm **3770ea239d5c4d0a596230ef7cf040cbf9431d626ff0cb56514f053c0d06f499** và **mongodbdata**
    - -e: mean environment variable
    - Ok, giờ chạy trên postman là bị đòi authentication

--- 2. Install .NET Secret Manager

-

```
Body  Cookies  Headers (4)  Test Results        Status: 500 Internal Server Error   Time: 821 ms
Pretty  Raw  Preview  Visualize  Text
1  MongoDB.Driver.MongoCommandException: Command find failed: command find requires authentication.
```

1. Declare a configuration setting for user

- Cài **.NET Secret Manager** → Terminal run: **dotnet user-secrets init**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> dotnet user-secrets init
Set UserSecretsId to '2b1088ca-f3dd-4d7b-9693-949bbab17f0c' for MSBuild project 'D:\QuanNguyen\Sofware-Testing\
```

- Mở file Catalog.csproject → thấy UserSecretsId

```
Catalog.csproj
1   <Project Sdk="Microsoft.NET.Sdk.Web">
2
3     <PropertyGroup>
4       <TargetFramework>net5.0</TargetFramework>
5       <UserSecretsId>2b1088ca-f3dd-4d7b-9693-949bbab17f0c</UserSecretsId>
```

- Mở file **appsettings.json**
  → khai báo User = “mongoadmin”,
  → khai báo password bằng cmd như sau: **dotnet user-secrets set MongoDbSettings:Password Pass#word1** → successfully

```
10  "MongoDbSettings": {
11    "Host": "localhost",
12    "Port": "27017",
13    "User": "mongoadmin"
14  }
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS                    4: pwsh
PS D:\projects\Catalog> dotnet user-secrets set MongoDbSettings:Password Pass#word1
```

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> dotnet user-secrets set MongoDbSettings:Password Pass#word1
Successfully saved MongoDbSettings:Password = Pass#word1 to the secret store.
```

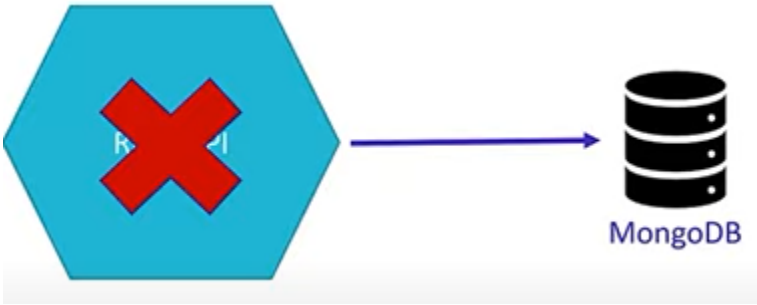
- Refresh lại VS MongoDB chiếc lá → thấy DB được refresh là ok!
- Mở file Settings/**MongoDbSettings,** add properties User, Password và đổi ConnectionString thành kiểu có authentication như bên dưới → đặt Debug → F5 → Mở Postman → Chạy thử GetItem và quay lại VS kiểm tra debug trả về password như hình dưới → OK đúng

```
1 reference
public string User { get; set; }
1 reference
public string Password { get; set; }

1 reference
public string ConnectionString
{
    get
    {                                   "Pass#word1"
        return $"mongodb://{User}:{Password}@{Host}:{Port}";
    }
}
```

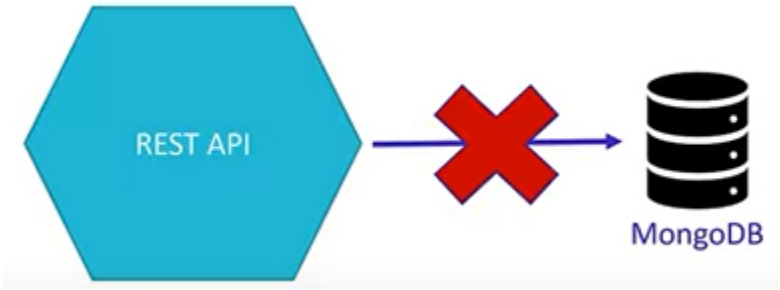
- → Mở Postman → Chạy hết API → OK đúng như thường

--- 3. Enabling HealthCheck for MongoDB

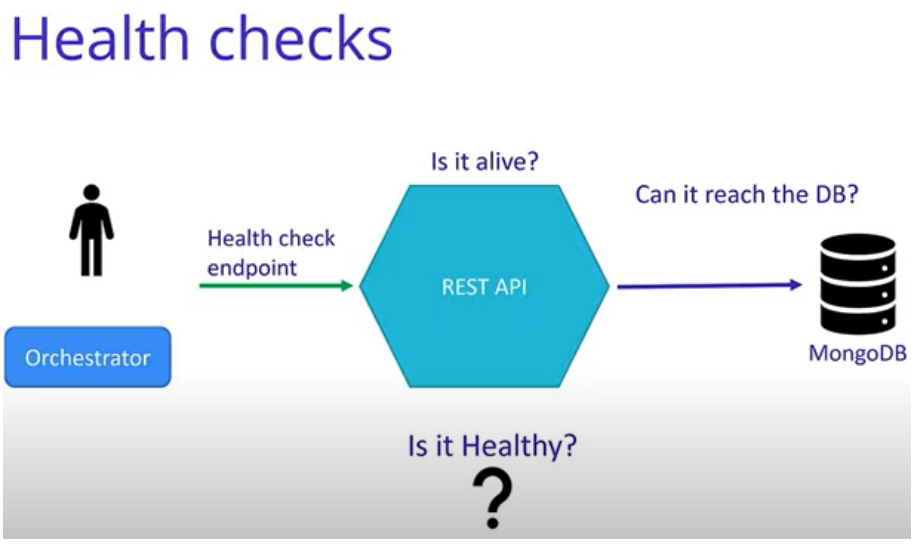
- For the **REST API** which is talking to MongoDB. The face that REST API can go down for a variety of reasons e.g temporarily deploying to cloud server



- For the **Connection.** It could have communication issues such as lose connection



- So the question is is our REST API alive? Then can we reach the DataBase? Is that connection in a good state? → **Is our API healthy**? → the answer is **enable health check endpoint** for the Orchestrator = (o'-kis-strây-đờ) = người trưởng dàn nhạc → who in charge of monitoring the system



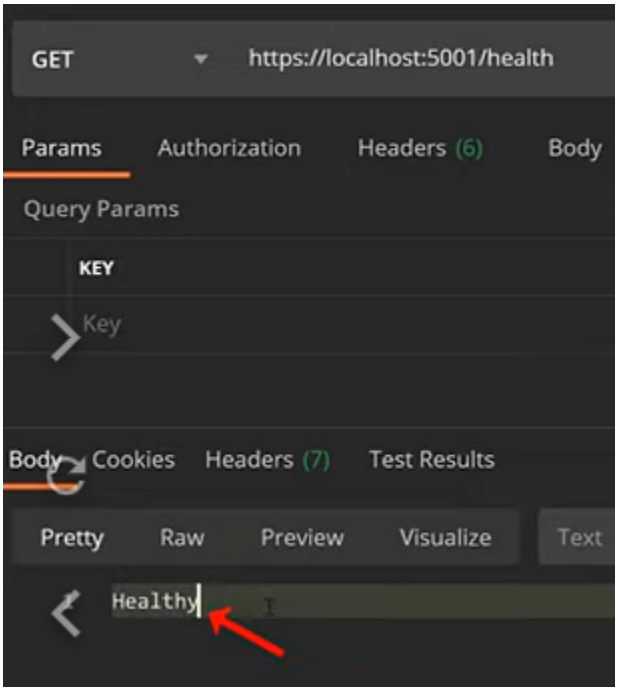
- <u>1. Add healthcheck in Startup file</u>
    - **ConfigureServices** method → add this

```
services.AddHealthChecks();
```

    - **Configure** method → add the endpoint.MapHealthChecks

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapHealthChecks("/health");
});
```

    - **F5** → Postman, run this → OK (but not really helpful, right?)



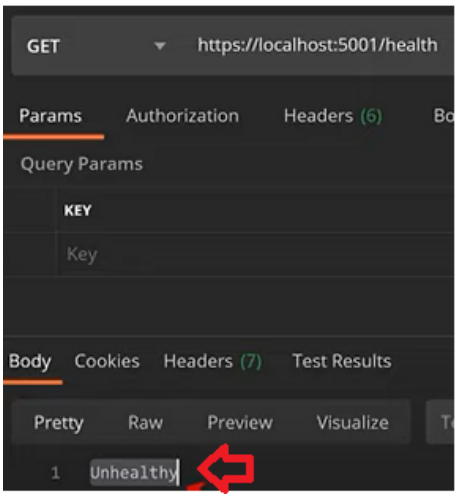
- <u>2. Add nuget package HealChecks.MongoDB</u>
    - Run: **dotnet add package AspNetCore.HealthChecks.MongoDb**
    - This package is open source, not from .Net but very handy.

- <u>3. Update the ConfigureServices</u>

```
var mongoDbSettings = Configuration.GetSection(nameof(MongoDbSettings)).Get<MongoDbSettings>();
services.AddHealthChecks().AddMongoDb(mongoDbSettings.ConnectionString, name: "mongodb", timeout: TimeSpan.FromSeconds(3));
```

- <u>4. **F5** → Postman, run this → OK → **docker stop mongo** → Postman run → Unhealthy</u>

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog
> docker stop mongo
mongo
```



- <u>5. More scenarios</u>
    - We may want to have more than 1 endpoint to verify. Check if it receive the request or not?
        - <u>Ready endpoint</u>: tell us is the service ready to receive the requests? Is the database up and running? Ready to go?
        - <u>Live endpoint</u>: tell us is the service alive?
        - Predicate is the way that we can filter out which health checks we want to include in the endpoint

```
services.AddHealthChecks()
.AddMongoDb(
    mongoDbSettings.ConnectionString,
    name: "mongodb",
    timeout: TimeSpan.FromSeconds(3),
    tags: new[] {"ready"}
    );
```

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapHealthChecks("/health/ready", new HealthCheckOptions{
        Predicate = (check) => check.Tags.Contains("ready")
    });
    endpoints.MapHealthChecks("/health/live", new HealthCheckOptions{
        Predicate = (_) => false
    });
});
```

        - Endpoint ready → check

- Endpoint live → don't check. False means we exclude every single health including the MongoDB one, so that it would just come back to us as long as the Rest API is alive
- Now, we are still having Mongo stopped → F5 → PM
  - https://localhost:5001/health/ready → after 3s → Unhealthy
  - https://localhost:5001/health/live → Healthy
- Now, we start Mongo
  - Run → **docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db -e MONGO_INITDB_ROOT_USERNAME=mongoadmin -e MONGO_INITDB_ROOT_PASSWORD=Pass#word1 mongo**
  - https://localhost:5001/health/ready → Healthy
  - https://localhost:5001/health/live → Healthy

- 6. Giờ configure để nó trả về message cụ thể hơn là (un)healthy như sau:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapHealthChecks("/health/ready", new HealthCheckOptions{
        Predicate = (check) => check.Tags.Contains("ready"),
        ResponseWriter = async(context, report) =>
        {
            var result = JsonSerializer.Serialize(
                new {
                    status = report.Status.ToString(),
                    checks = report.Entries.Select(entry => new {
                        name = entry.Key,
                        status = entry.Value.Status.ToString(),
                        exception = entry.Value.Exception != null ? entry.Value.Exception.Message: "none",
                        duration = entry.Value.Duration.ToString()
                    })
                }
            );
            context.Response.ContentType = MediaTypeNames.Application.Json;
            await context.Response.WriteAsync(result);
        }
    });
     endpoints.MapHealthChecks("/health/live", new HealthCheckOptions{
        Predicate = (_) => false
    });
});
```

F5 → Postman → GET https://localhost:5001/health/ready

GET https://localhost:5001/health/ready

Body Cookies Headers (7) Test Results Status: 200 OK Time: 2.18 s

Pretty Raw Preview Visualize JSON


```
{
    "status": "Healthy",
    "checks": [
        {
            "name": "mongodb",
            "status": "Healthy",
            "exception": "none",
            "duration": "00:00:00.7474864"
        }
    ]
}
```


Run: **docker stop mongo** → Postman → GET
https://localhost:5001/health/ready

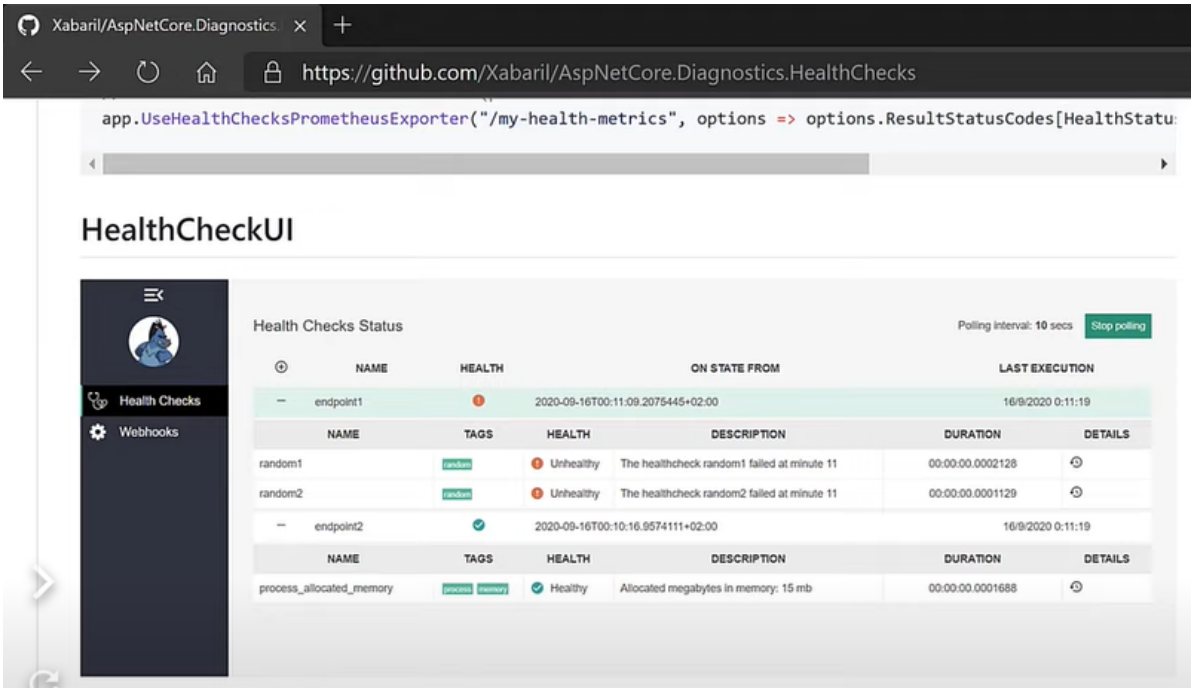
GET https://localhost:5001/health/ready

Body Cookies Headers (7) Test Results Status: 503 Service Unavailable Time: 3.07 s

Pretty Raw Preview Visualize JSON


```
{
    "status": "Unhealthy",
    "checks": [
        {
            "name": "mongodb",
            "status": "Unhealthy",
            "exception": "The operation was canceled.",
            "duration": "00:00:03.0278885"
        }
    ]
}
```


Thông tin thêm về Health Check, xem thêm tại đây
https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks

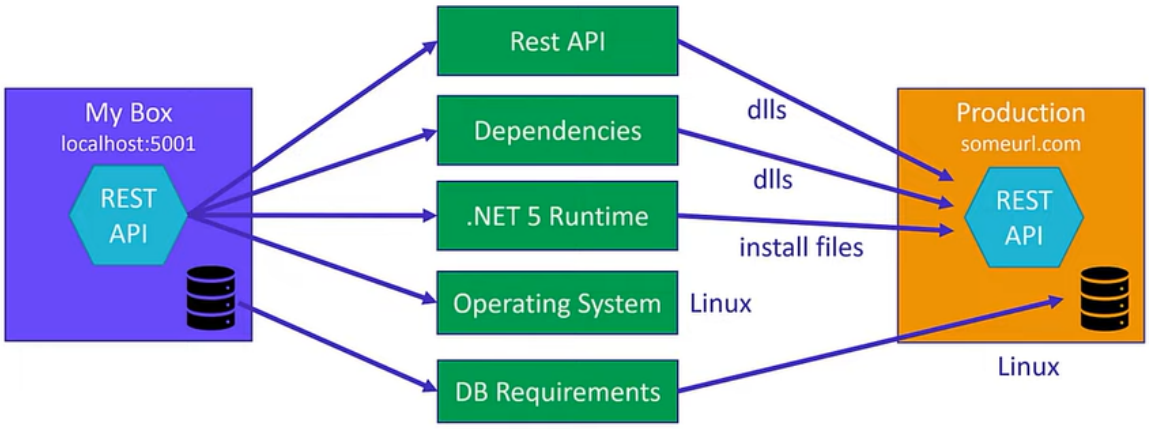


VIII. DOCKER

You will learn...

- The challenges of deployment
- How docker works and why you should use it
- How to turn your REST API into a docker image
- How to run your REST API as a docker container

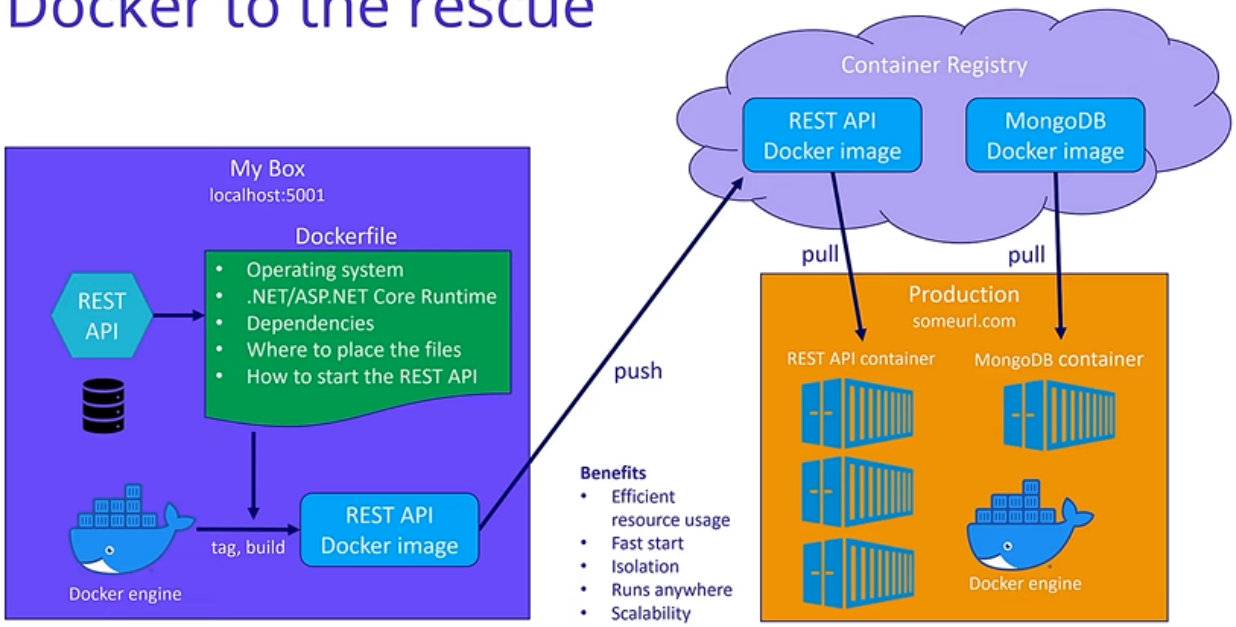
Preparing for deployment



- Prepare a box
  - Physical machine or virtual machine
  - Linux or Windows, pick the correct OS version!
- How to take the files to the production machine?
- What if DB requires different version of OS or dependencies?
- What if we want to move to a new version of .NET?
- How do we quickly start the REST API on the machine?
- What if one instance is not enough to handle the load?

Rescue = giải thoát

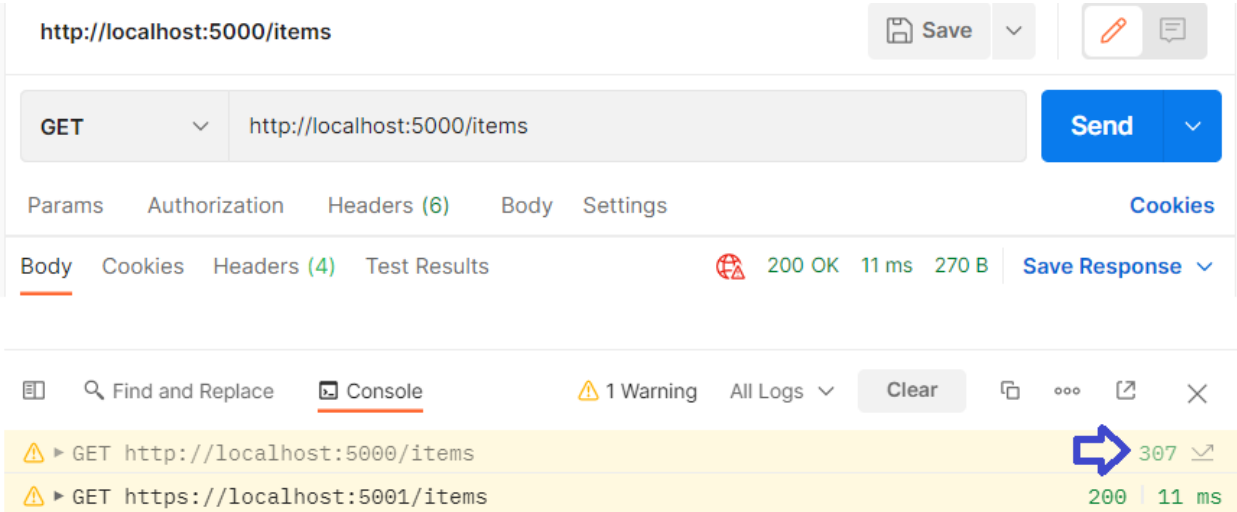
Docker to the rescue



--- 1. Create Docker Image for REST API

1. <u>Don't redirect HTTP to HTTPS</u>

a. To keep it simple, we need to update the REST API so that it no longer performs HTTPS direction and allows the use of HTTP addresses only. The use of HTTPS from here on is a topic out of the scope of this tutorial
b. Hiện tại nếu gọi HTTP → nó sẽ auto redirect qua HTTPS (Code trả về **307** tức là this request was triggered on redirect) → lúc nào cũng về HTTPs



c. Làm sao? → Go to Startup.cs → Update method Configure chỗ này thành:

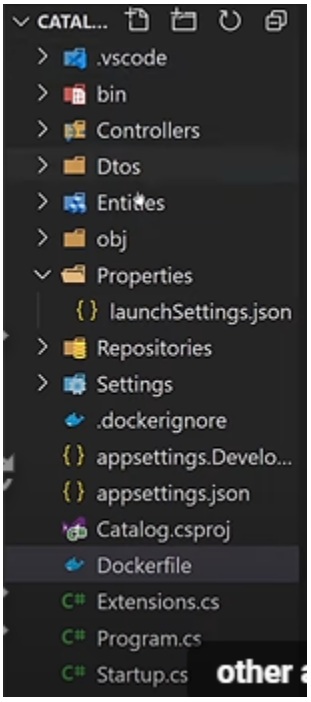
```
if (env.IsDevelopment())
{
    app.UseHttpsRedirection();
}
```

Lúc nào mình setup Docker container xong nó sẽ work.

2. <u>Create Corresponding Docker file</u>
   a. Install Docker Extension
   b. Go to View → Command Palette → **"Docker: Add Docker Files to Workspace"** → **.NET: ASP.NET Core** → select OS of the docker container is **Linux** → Select Port **80** → **NO** generate Docker Compose file → Then you see ***Dockerfile*** and ***.dockerignore*** files is generated
3. <u>Giải thích DockerFile (chưa hiểu thì nên xem lại nhiều lần)</u>

```
Dockerfile > ...
1 FROM mcr.microsoft.com/dotnet/aspnet:5.0-focal AS base
  WORKDIR /app
  EXPOSE 80

2 ENV ASPNETCORE_URLS=http://+:80

3 FROM mcr.microsoft.com/dotnet/sdk:5.0-focal AS build
  WORKDIR /src
  COPY ["Catalog.csproj", "./"]
  RUN dotnet restore "Catalog.csproj"
  COPY . .
  WORKDIR "/src/."
  RUN dotnet build "Catalog.csproj" -c Release -o /app/build

4 FROM build AS publish
  RUN dotnet publish "Catalog.csproj" -c Release -o /app/publish

5 FROM base AS final
  WORKDIR /app
  COPY --from=publish /app/publish .
  ENTRYPOINT ["dotnet", "Catalog.dll"]
```

1. The 1st stage of building the container we called as **base**. We'll go from an asp.net version 5.0. **/app** means everything happens after this will happen in the /app directory. Expose via port **80 as we** declared at the step before
2. We define an environment variable URL
3. The 2nd stage is **build.** It says ok, let copy the **Catalog.csproj**, the one that defines our project → (into) the root of the current location (**./**). **Copy . .** means copy every other files that's needed for a folder up including all the files in the left side except few sections



4. Mình sẽ remove dòng 12 **Workdir "/src/."** vì nó giống với dòng **Workdir /src**
5. Dòng 13 nghĩa là mình sẽ chạy dotnet build, release the version of the app to optimize it for the production environment. And the results of that build will go into the **/app/build** directory
6. Optimal 1 chút để bỏ những dòng dư thừa thành

```
Dockerfile > ...
1   FROM mcr.microsoft.com/dotnet/aspnet:5.0-focal AS base
2   WORKDIR /app
3   EXPOSE 80
4
5   ENV ASPNETCORE_URLS=http://+:80
6
7   FROM mcr.microsoft.com/dotnet/sdk:5.0-focal AS build
8   WORKDIR /src
9   COPY ["Catalog.csproj", "./"]
10  RUN dotnet restore "Catalog.csproj"
11  COPY . .
12  RUN dotnet publish "Catalog.csproj" -c Release -o /app/publish
13
14  FROM base AS final
15  WORKDIR /app
16  COPY --from=build /app/publish .
17  ENTRYPOINT ["dotnet", "Catalog.dll"]
```

Dòng 15 tức là mình switch again về lại /app directory
Dòng 16 copy from ở build stage, copy toàn bộ những gì có trong thư mục **/app/publish** vào trong thư mục **/app** (ở dòng 15)

4. <u>Build Docker image</u>
   a. Terminal run: **docker build -t catalog:v1 .**
   b. **-t** là tag, **v1** là tên của tag, dấu **.** nghĩa là current directory
   c. Nếu mình ko có gì thay đổi mà chạy lại câu lệnh, thì nó sẽ ko có gì thay đổi do nó có cache lại. Nếu có thay đổi thì nó chạy lại vài file thay đổi

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catal
[+] Building 1.0s (18/18) FINISHED
 => [internal] load build definition from Dockerfile          0.0s
 => [internal] load .dockerignore                             0.0s
 => => transferring context: 35B                              0.0s
 => [internal] load metadata for mcr.microsoft.com/dotnet/sdk  0.8s
 => [internal] load metadata for mcr.microsoft.com/dotnet/asp  0.8s
 => [build 1/7] FROM mcr.microsoft.com/dotnet/sdk:5.0-focal@s  0.0s
 => [base 1/2] FROM mcr.microsoft.com/dotnet/aspnet:5.0-focal  0.0s
 => [internal] load build context                             0.1s
 => => transferring context: 851B                             0.0s
 => CACHED [base 2/2] WORKDIR /app                            0.0s
 => CACHED [final 1/2] WORKDIR /app                           0.0s
 => CACHED [build 2/7] WORKDIR /src                           0.0s
 => CACHED [build 3/7] COPY [Catalog.csproj, ./]              0.0s
 => CACHED [build 4/7] RUN dotnet restore "Catalog.csproj"    0.0s
 => CACHED [build 5/7] COPY . .                               0.0s
 => CACHED [build 6/7] WORKDIR /src/.                         0.0s
 => CACHED [build 7/7] RUN dotnet build "Catalog.csproj" -c R  0.0s
 => CACHED [publish 1/1] RUN dotnet publish "Catalog.csproj"  0.0s
 => CACHED [final 2/2] COPY --from=publish /app/publish .     0.0s
 => exporting to image                                        0.0s
 => => exporting layers                                       0.0s
 => => writing image sha256:a36f75b5eb94ac19339384b8f90413b58  0.0s
 => => naming to docker.io/library/catalog:v1                 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabi
lities and learn how to fix them
```

5. <u>Create a Docker Network</u>
   a. Terminal run: **docker network create net5tutorial**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catal
og> docker network create net5tutorial
7aef5226ef92adbaca41571ca441964e935a4b2f2eb5ba85bd18b3600f86cb89
```

   b. Check : **docker network ls**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catal
og> docker network ls
NETWORK ID     NAME                 DRIVER    SCOPE
29cd074e50d9   bridge               bridge    local
c0379677edd1   for-mobile_default   bridge    local
ab4b5a89312b   host                 host      local
7aef5226ef92   net5tutorial         bridge    local
c3488e699ac2   none                 null      local
```

--- 2. Join the MongoBD container to the created network above

   a. Stop MongoDB container if any → **docker ps** → thấy mongo đang running → **docker stop mongo**
   b. Thay vì chạy câu lệnh như hồi trước:
      docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db -e MONGO_INITDB_ROOT_USERNAME=mongoadmin -e MONGO_INITDB_ROOT_PASSWORD=Pass#word1 mongo
   c. Giờ chạy lệnh đó với thêm 1 thuộc tính màu tím:
      docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/data/db -e MONGO_INITDB_ROOT_USERNAME=mongoadmin -e

MONGO_INITDB_ROOT_PASSWORD=Pass#word1 --network=net5tutorial mongo

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catal
og> docker run -d --rm --name mongo -p 27017:27017 -v mongodbdata:/d
ata/db -e MONGO_INITDB_ROOT_USERNAME=mongoadmin -e MONGO_INITDB_ROOT
_PASSWORD=Pass#word1 --network=net5tutorial mongo
bc578dd95e39020d500c6ca17a3a8ddd5a967e6118e65c664f5aa1c300b805c4
```

d. **docker image** → thấy **catalog** và **mongo** → ok

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker images
REPOSITORY                    TAG          IMAGE ID       CREATED          SIZE
catalog                       v1           a36f75b5eb94   13 minutes ago   225MB
mongo                         latest       ccf4b4ee3bee   6 days ago       685MB
```

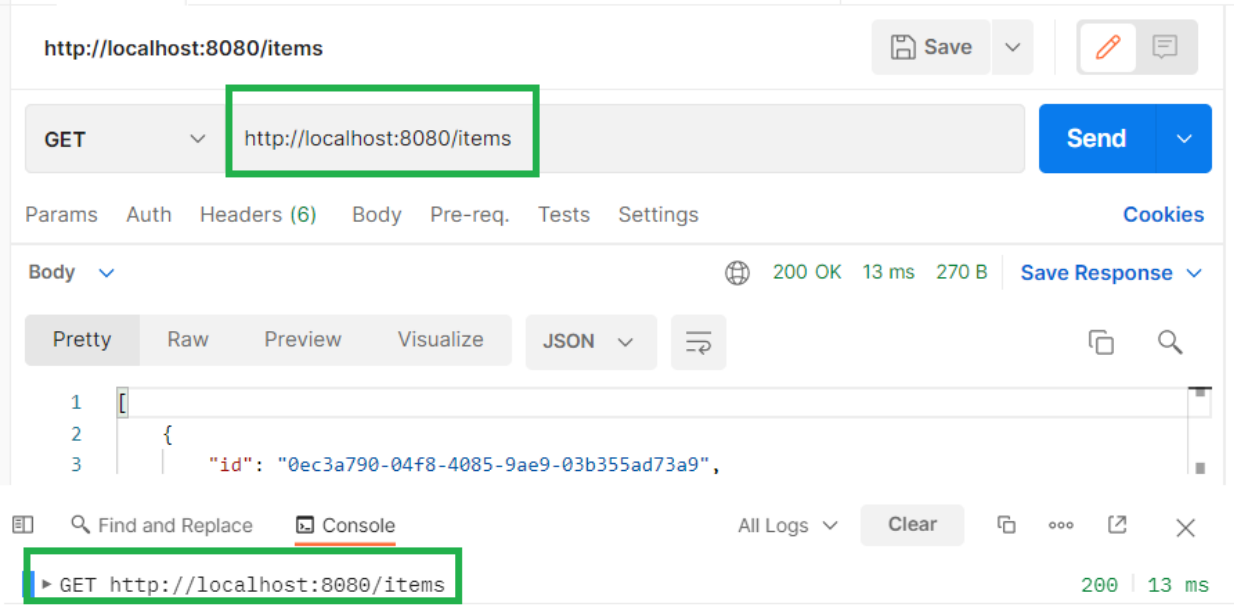
--- 3. Start Docker Container

a. **docker run -it --rm -p 8080:80 -e MongoDbSettings:Host=mongo -e MongoDbSettings:Password=Pass#word1 --network=net5tutorial catalog:v1**

```
PS Ddocker run -it --rm -p 8080:80 -e MongoDbSettings:Host=mongo -e MongoDbSettings:Password=Pass#word1 --netw
ork=net5tutorial catalog:v1
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```

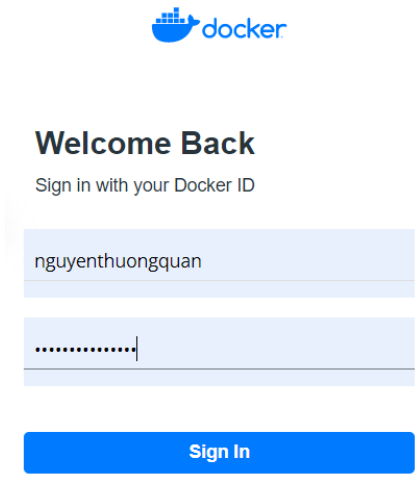
Host environment is changed from Development to Production

b. Postman → Port 8080 Ok → Nolonger redirect to Https



--- 4. How could we share the Docker image to somebody else?

a. Login docker by cmd:
   - Stop docker image → VS → Ctrl C
   - Make sure you can login to Docker with your account vd: nguyenthuongquan / P@....99#



   - Login docker by cmd → **docker login**
   - Username: **nguyenthuongquan**
   - Password: **P@....99#**

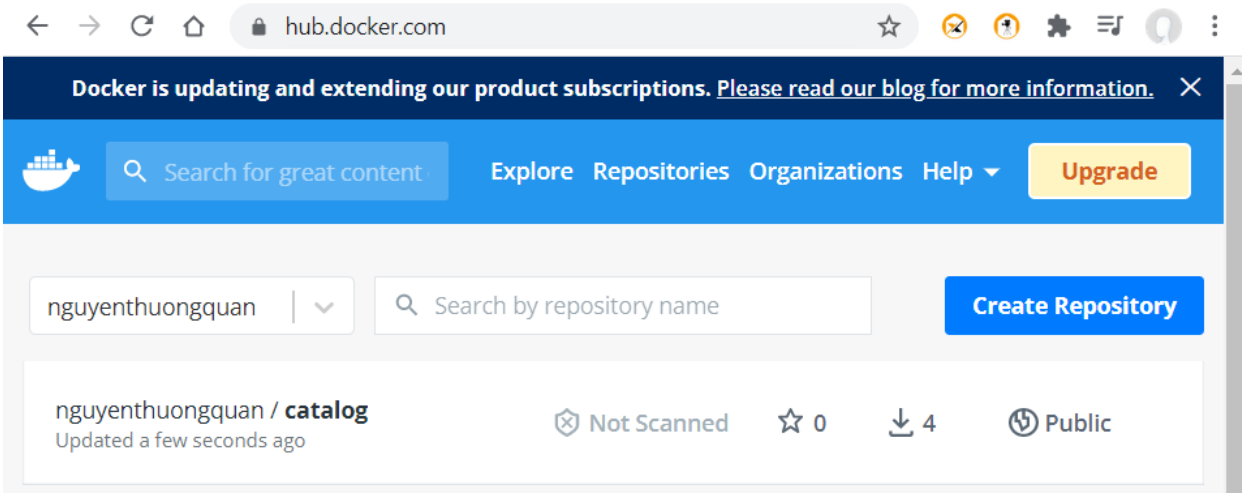
```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to
 https://hub.docker.com to create one.
Username: nguyenthuongquan
Password:
Login Succeeded
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

b. Retag the docker image with credential: from **catalog** to **nguyenthuongquan/catalog**
   - **docker images**
   - **docker tag catalog:v1 nguyenthuongquan/catalog:v1** → same Id

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker tag catalog:v1 nguyenthuongquan
/catalog:v1
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
nguyenthuongquan/catalog      v1        a36f75b5eb94   38 minutes ago   225MB
catalog                       v1        a36f75b5eb94   38 minutes ago   225MB
mongo                         latest    ccf4b4ee3bee   6 days ago       685MB
```

   - **docker push nguyenthuongquan/catalog:v1** → publish into Docker hub → so anybody else got in the future, just pull that image and start using

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker push nguyenthuongquan/catalog:v
1
The push refers to repository [docker.io/nguyenthuongquan/catalog]
34449e305472: Pushed
5f70bf18a086: Pushed
97231560cb7b: Pushed
42291a3a402d: Pushed
799412614616: Pushed
a50d23118da7: Pushed
9ccfd7bd4396: Pushed
4942a1abcbfa: Mounted from library/mongo
v1: digest: sha256:2dd1e936af183cc3755aead48bb86a28830d44b07d1b5fffaa61a5e413e6955a size: 1995
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```



--- 5. Get that new published Docker image and use

a. Remove the 2 existing docker images (same Id) →
   - **docker rmi nguyenthuongquan/catalog:v1**
   - **docker rmi catalog:v1**
   - **docker images** → ok the catalog image is removed
   - **docker logout**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker rmi nguyenthuongquan/catalog:v1

Untagged: nguyenthuongquan/catalog:v1
Untagged: nguyenthuongquan/catalog@sha256:2dd1e936af183cc3755aead48bb86a28830d44b07d1b5fffaa61a5e413e6955a
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker rmi catalog:v1
Untagged: catalog:v1
Deleted: sha256:a36f75b5eb94ac19339384b8f90413b58bab3f170eada46fc2ffe741134271d0
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker images
REPOSITORY                          TAG             IMAGE ID       CREATED        SIZE
mongo                               latest          ccf4b4ee3bee   6 days ago     685MB
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker logout
Removing login credentials for https://index.docker.io/v1/
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

At this point, I'm somebody who has never had access to Catalog.

b. Pull the nguyenthuongquan/catalog image and use it for the 1st time.
   - **docker run -it --rm -p 8080:80 -e MongoDbSettings:Host=mongo -e MongoDbSettings:Password=Pass#word1 --network=net5tutorial nguyenthuongquan/catalog:v1**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> docker run -it --rm -p 8080:80
oDbSettings:Host=mongo -e MongoDbSettings:Password=Pass#word1 --network=net5tutorial nguyenthuongquan/c
v1
Unable to find image 'nguyenthuongquan/catalog:v1' locally
v1: Pulling from nguyenthuongquan/catalog
35807b77a593: Already exists
e8381c8661cf: Already exists
da6c06131f06: Already exists
005d8a0c9128: Already exists
a0a7917bab44: Already exists
e75173d462a2: Already exists
4f4fb700ef54: Already exists
e4853e749778: Already exists
Digest: sha256:2dd1e936af183cc3755aead48bb86a28830d44b07d1b5fffaa61a5e413e6955a
Status: Downloaded newer image for nguyenthuongquan/catalog:v1
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
```

- Postman → OK

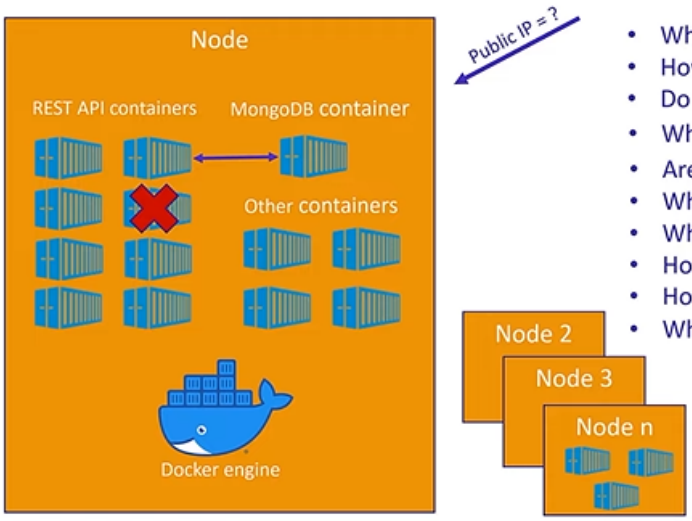
IX. KUBERNETES

You will learn...

- Why a container orchestrator is needed
- What is Kubernetes and which are its basic components
- How Kubernetes enables resilient distributed systems
- How to stand up a basic Kubernetes cluster in your box
- How to deploy your REST API (and MongoDB) to Kubernetes
- How to scale a Kubernetes deployment

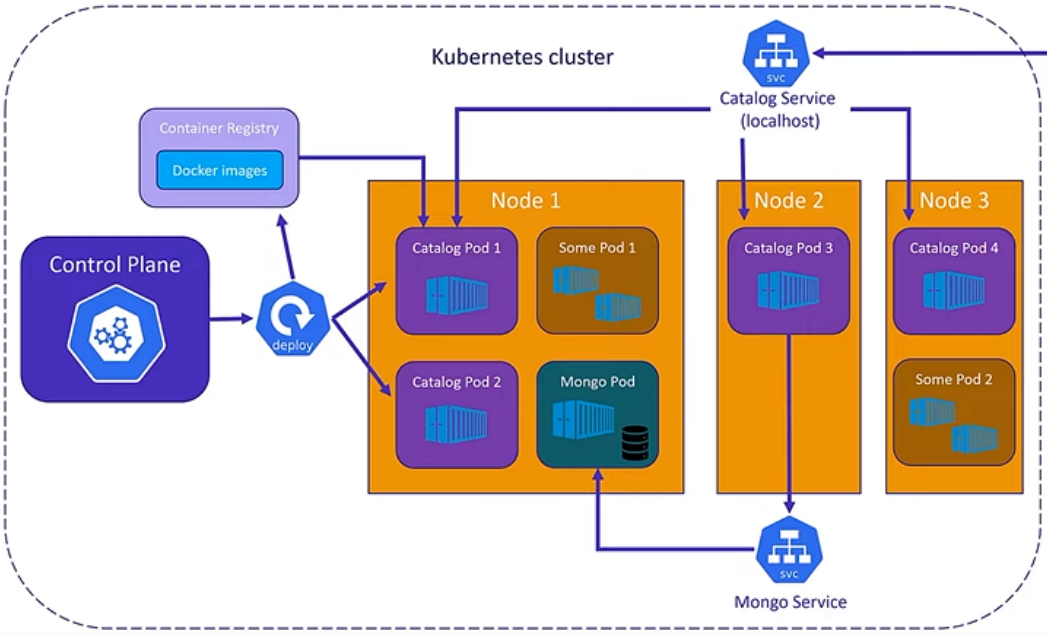
Orchestrator = người nhạc trưởng
Resilient = đàn hồi
Cluster = cụm

Why Orchestration?



- Who takes care of pulling and running the containers?
- How to run the containers?
- Do we have enough container instances?
- Which is the best node to place the containers?
- Are the containers healthy? What if one crashes?
- Where to store the secrets?
- Where to store the database files?
- How to enable containers to talk to each other?
- How to reach the containers from the outside?
- Which of all instances should serve a request?

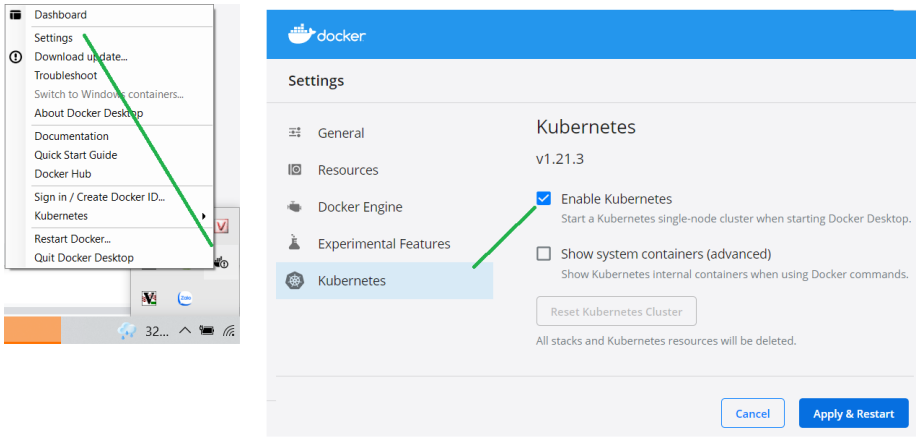
Enter Kubernetes

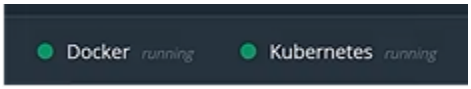


- Turns desired state into actual state
- Selects nodes to run pods
- Self-heals by creating and killing pods
- Stores configuration and secrets
- Provides service discovery and load balancing
- Ensures no downtime
- Can auto scale
- Auto mounts storage
- Provides gradual rollout and rollback

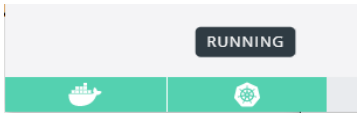
--- 1. Enable Kubernetes and create yaml file

a. <u>Enable Kubernetes cluster.</u>



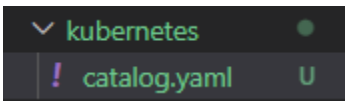


or



b. <u>Connect to the correct Kubernetes cluster.</u>
   - Run: **kubectl config current-context** → 'docker-desktop' is the name of the cluster installed by Docker when we enable Kubernetes

c. <u>Creating and declaring the components into Kubernetes</u>
   - Install **Kubernetes extension**
   - In VS create folder kubernetes with catalog.yaml file



   - Type **deploy** and select **'Kubernetes Deployment'** → automatically generate basic code for the deployment resource.

```
io.k8s.api.apps.v1.Deployment (v1@deployment.json)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: <Image>
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: <Port>
```

Giải thích

- **Metadata**: defines name for these deployment → catalog-deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: catalog-deployment
```

- **Selector**: specify how this deployment is going to select the pods that it is going to manage. In this case → manage all the pods that have **app** label = **catalog**, will be managed by this deployment

```
spec:
  selector:
    matchLabels:
      app: catalog
```

- **Template**: declare all the containers that is going to be included in the deployment. Declare the 1st thing is the label for the pods. In this case below → assign a ‘catalog’ label to all pods. Remember

the label in line #12 has to match the label in line #8 so that the deployment can actually manage these pods

```
9   template:
10    metadata:
11      labels:
12        app: catalog
```

- **Containers:** declare the list of containers that are going to be included in this deployment. Give it a name e.g: catalog. → Declare image that we are going to pull down into Kubernetes to deploy e.g: nguyenthuongquan/catalog:v1

```
13    spec:
14      containers:
15      - name: catalog
16        image: nguyenthuongquan/catalog:v1
```

- **Resources**: declare what resources specifically in terms of memory and CPU. In this case, what resources are needed from the Kubernetes node, in order to run your REST API.
    - 1Mi (Mebibyte) = 1024 Kibibytes = 1,048,576 Bytes
    - 500 millicpu = 0.5 CPU =. In this case we assign half CPU to this container, definitely enough for us.

```
17        resources:
18          limits:
19            memory: "128Mi"
20            cpu: "500m"
```

- **ContainerPort**: declare which port that our container is exposing (lộ ra) so we can communicate to, in order to access the REST api. Here we declare **80**

```
21        ports:
22        - containerPort: 80
```

- **Một số env variable cần khai báo thêm**:

```
23        env:
24          - name: MongoDbSettings__Host
25            value: mongodb-service
```

- **Create the secret for password**
    - Ctrl J: to open last terminal
    - Run: **kubectl create secret generic catalog-secrets --from-literal=mongodb-password='Pass#word1'**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo
\rest-api-csharp\Catalog> kubectl create secre
t generic catalog-secrets --from-literal=mongo
db-password='Pass#word1'
secret/catalog-secrets created
```

- Sau đó mình declare password tiếp theo dòng 25 ở trên

```
26          - name: MongoDbSettings__Password
27            valueFrom:
28             secretKeyRef:
29               name: catalog-secrets
30               key: mongodb-password
```

- **Create the Health Probes**

```
31          livenessProbe:
32           httpGet:
33             path: /health/live
34             port: 80
35          readinessProbe:
36           httpGet:
37             path: /health/ready
38             port: 80
```

- **Enable Service, so anybody from outside can reach into this container**
  - Thay vì tách ra 2 yaml file, thì mình dùng --- để tách file.
  - Sau đó gõ **service** and select **'Kubernetes service** → automatically generate basic code (lần trước mình gõ **deploy**)
  - **Service is an abstract way to expose an application running on a set of Pods as a network service**

```
40  ---
41  apiVersion: v1
42  kind: Service
43  metadata:
44    name: myapp
45  spec:
```

Service

An abstract way to expose an application running on a set of Pods as a network service

- **Type**: **LoadBalancer** is the one that allows Kubernetes to request a way to open a window to the outside so that people can reach out to the Rest API from the outside. Lát chạy sẽ rõ

```yaml
40  ---
41    apiVersion: v1
42    kind: Service
43    metadata:
44      name: catalog-service
45    spec:
46      type: LoadBalancer
```

- **Selector**: app = catalog, phải giống với **label** trong mục **template.** This service is going to be able to target any of the pods that have the label = **app** with be value = **catalog.** That how we connect the service with the pods

```yaml
47      selector:
48        app: catalog
```

- **Port**: ở dòng 50. We are going to be using **80** just because it's the default port for HTTP. They will go the port **80**
- **TargetPort**: ở dòng 51. Has to be the port that we have specified for the container **80.** Scroll up up, it has to be = containerPort = **80** → **So it maps 80 to 80** → but we also can map **8080 to 80** that fine but it is more usual to use a port **80**

```yaml
49      ports:
50      - port: 80
51        targetPort: 80
```

→ NOW, it ready to start deploying both **deployment** and **service**

--- 2. How to deploy this into Kubernetes

a. Ctrl J: type cd kube [tab] → **cd .\kubernetes\ → kubectl apply -f .\catalog.yaml → kubectl get deployments**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> cd .\kubernetes\
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes> kubectl apply -f .\catalog.yaml
deployment.apps/catalog-deployment created
service/catalog-service created
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes> kubectl get deployments
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
catalog-deployment   0/1     1            0           7s
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes>
```

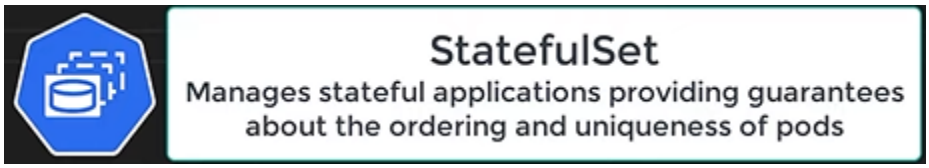
It is saying 0/1 pod is ready to go → it's not ready yet → check by running:

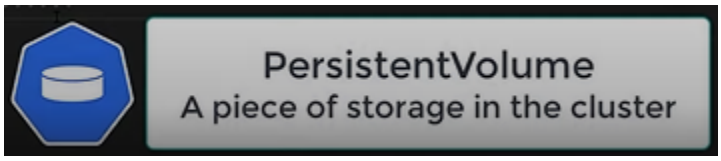
- **kubectl get pods** → copy the name →
- **kubectl logs catalog-deployment-6fc68c644f-dqg4q** → saying Unhealthy

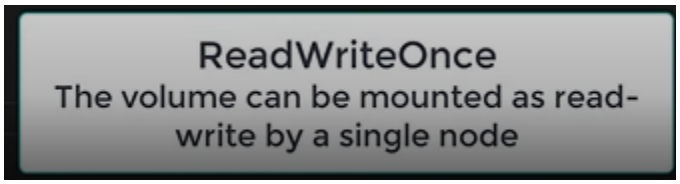
```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes> kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-dqg4q   0/1     Running   0          5m26s
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes> kubectl logs catalog-deployment-6fc68c644f-dqg4q
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
fail: Microsoft.Extensions.Diagnostics.HealthChecks.DefaultHealthCheckService[103]
      Health check mongodb completed after 875.8616ms with status Unhealthy and description '(null)'
      System.OperationCanceledException: The operation was canceled.
         at System.Threading.CancellationToken.ThrowOperationCanceledException()
         at System.Threading.CancellationToken.ThrowIfCancellationRequested()
         at MongoDB.Driver.Core.Clusters.Cluster.WaitForDescriptionChangedHelper.HandleCompletedTask(Task completedTask)
         at MongoDB.Driver.Core.Clusters.Cluster.WaitForDescriptionChangedAsync(IServerSelector selector, ClusterDescription descrip
```

Ok, close it !

b. Under folder kubernetes, create **mongodb.yaml,** this just for the deployment of the mongodb database. Now the type of resource that we are going to create for MongoDB is actually called a **stateful-set.** **state→ Kubernetes StatefulSet**







```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb-statefulset
spec:
  serviceName: "mongodb-service"
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 27017
        env:
          - name: MONGO_INITDB_ROOT_USERNAME
            value: "mongoadmin"
          - name: MONGO_INITDB_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: catalog-secrets
                key: mongodb-password
        volumeMounts:
          - mountPath: /data/db
            name: data
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi

---
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  clusterIP: None
  selector:
    app: mongodb
  ports:
  - port: 27017
    targetPort: 27017
```

c. Ctrl J: type cd kube [tab] → **cd .\kubernetes\ → kubectl apply -f .\mongodb.yaml → kubectl get statefulsets**

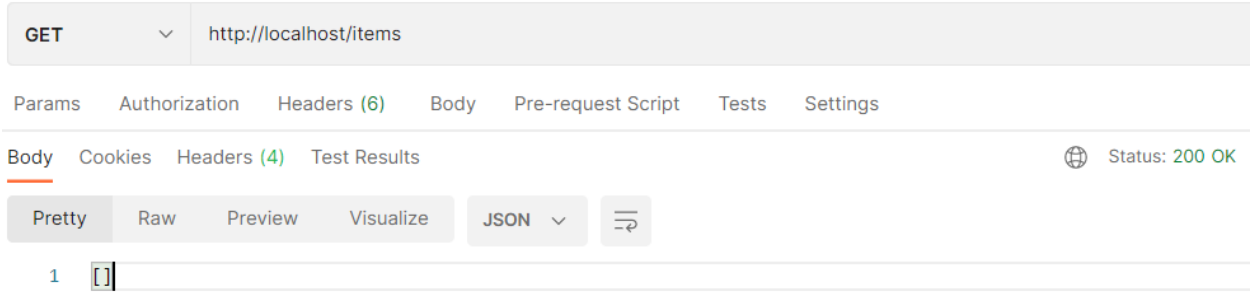
```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog
\kubernetes> kubectl apply -f .\mongodb.yaml
statefulset.apps/mongodb-statefulset created
service/mongodb-service created
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog
\kubernetes> kubectl get statefulsets
NAME                 READY   AGE
mongodb-statefulset  1/1     38s
```

→ **kubectl get pods** → **ok**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubern
etes> kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-dqg4q   1/1     Running   0          47m
mongodb-statefulset-0                 1/1     Running   0          2m11s
```

d. Test with Postman

Now we can call api on port 80, but 80 is the default so we can remove it. The call below work fine with 200 OK status (http://localhost:80/items or http://localhost/items)



Run all api again. (change https to http, remove port 8080) → Ok

--- 3. Exploring the Kubernetes self-healing capability

Split the panel to have 2 terminal side by side:

a. <u>Test delete the catalog-deployment would it come back?</u>

- TerLeft: **kubectl get pods -w** → that let me **watch** changes that happen across the current existing pods

- TerRight: **kubectl delete pod catalog-deployment-6fc68c644f-dqg4q** → that will delete a pod → then on TerLeft, right away. As soon as it starts terminating that pod, it immediately spins up a brand new container. The Kubernetes brings it back.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl get po
ds -w
NAME                                    READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-dqg4q     1/1     Running   0          73m
mongodb-statefulset-0                   1/1     Running   0          28m
catalog-deployment-6fc68c644f-dqg4q     1/1     Terminating   0          79m
catalog-deployment-6fc68c644f-mvl6z     0/1     Pending       0          0s
catalog-deployment-6fc68c644f-mvl6z     0/1     Pending       0          0s
catalog-deployment-6fc68c644f-mvl6z     0/1     ContainerCreating   0          0s
catalog-deployment-6fc68c644f-dqg4q     0/1     Terminating         0          79m
catalog-deployment-6fc68c644f-mvl6z     0/1     Running             0          3s
catalog-deployment-6fc68c644f-dqg4q     0/1     Terminating         0          79m
catalog-deployment-6fc68c644f-dqg4q     0/1     Terminating         0          79m
catalog-deployment-6fc68c644f-mvl6z     1/1     Running             0          11s
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl delete
 pod catalog-deployment-6fc68c644f-dqg4q
pod "catalog-deployment-6fc68c644f-dqg4q" deleted
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

- TerLeft: **Ctrl - C** → **kubectl get pods** → good again. It's not the same identity (mấy số cuối) as before. The old deleted one ends with **dqg4q**, the new one **mv16z**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl get po
ds
NAME                                    READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-mvl6z     1/1     Running   0          2m26s
mongodb-statefulset-0                   1/1     Running   0          36m
```

b. <u>Test delete the mongodb-statefulset, would it come back?</u>

- TerLeft: **kubectl get pods -w** → that let me **watch** changes that happen across the current existing pods

- TerRight: **kubectl delete pod mongodb-statefulset-0** → that will delete a pod → then on TerLeft, right away. As soon as it starts terminating that pod, it immediately spins up a **SAME.** So the data is still there, still exists. That means our persistent volume got created and the data is being stored outside of the container and into the host. And that is enabled by our stateful-set. → Go to Postman → Get → Data is still there.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl get po
ds -w
NAME                                    READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-mvl6z     1/1     Running   0          7m55s
mongodb-statefulset-0                   1/1     Running   0          41m
mongodb-statefulset-0                   1/1     Terminating   0          43m
mongodb-statefulset-0                   0/1     Terminating   0          43m
mongodb-statefulset-0                   0/1     Terminating   0          43m
mongodb-statefulset-0                   0/1     Terminating   0          43m
mongodb-statefulset-0                   0/1     Pending       0          0s
mongodb-statefulset-0                   0/1     Pending       0          0s
mongodb-statefulset-0                   0/1     ContainerCreating   0          0s
mongodb-statefulset-0                   1/1     Running             0          6s
catalog-deployment-6fc68c644f-mvl6z     0/1     Running             0          10m
catalog-deployment-6fc68c644f-mvl6z     1/1     Running             0          10m
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl get po
ds
NAME                                    READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-mvl6z     1/1     Running   0          11m
mongodb-statefulset-0                   1/1     Running   0          72s
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl delete
 pod mongodb-statefulset-0
pod "mongodb-statefulset-0" deleted
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

--- 4. Scaling Kubernetes pods

Split the panel to have 2 terminal side by side:

- TerLeft: **kubectl get pods -w** → that let me **watch** changes that happen across the current existing pods

- TerRight: **kubectl scale deployments/catalog-deployment --replicas=3** → that will immediately start provisioning a bunch of new pods to enforce this new decided state, which happens blazing fast. (Replica = bản chép lại)

- TerLeft: **Ctrl - C** → **kubectl get pods** → we had 1 catalog-deployment just seconds ago, now wwe have 3 pods → now we see the power of combination of Docker containers, and Kubenetes, now we can bring in a much more instances of our containers of our REST API. Just 1 line command, now we have 3 copies of the container running. The whole point of having these data replicas of the pod is so that we can do some good load balancing between them. So when a request come, it should land in one or another of these pods

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
catalog-deployment-6fc68c644f-hvrg9   1/1     Running   0          32s
catalog-deployment-6fc68c644f-mvl6z   1/1     Running   1          14h
catalog-deployment-6fc68c644f-tddfv   1/1     Running   0          32s
mongodb-statefulset-0                 1/1     Running   1          14h
```

--- 5. Adding logs via ILogger

Open **Controllers/ItemsControllers** and add some code as below

```
public class ItemsController: ControllerBase
{
    8 references
    private readonly IItemsRepository repository;
    2 references
    private readonly ILogger<ItemsController> logger;

    0 references
    public ItemsController(IItemsRepository repository, ILogger<ItemsController> logger)
    {
        this.repository = repository;
        this.logger = logger;
    }

    //GET /items
    [HttpGet]
    0 references
    public async Task<IEnumerable<ItemDto>> GetItemsAsync()
    {
        var items = (await repository.GetItemsAsync())
            .Select(item => item.AsDto());
        logger.LogInformation($"{DateTime.UtcNow.ToString("hh:mm:ss")}: Retrived {items.Count()} items");
        return items;
    }
```

Terminal: **docker build -t nguyenthuongquan/catalog:v2 .** → this is the new version of the image, we should be bumping it to v2, this is necessary, so that Kubernetes can later tell that this is a new image, or that needs to pull it down from Docker

Terminal: **docker login → nguyenthuongquan → P@ssw0rd230899#**

Terminal: **docker push nguyenthuongquan/catalog:v2**

Open **kubernetes/catalog.yaml** and update v1 image to: *nguyenthuongquan/catalog:v2*

Terminal: **cd .\kubernetes**

Terminal: **kubectl apply -f .\catalog.yaml**

Terminal: **kubectl get pods -w** → then you see the old container, the old pods are getting destroyed. And new pods are getting immediately installed.

Terminal: Ctrl-C → **kubectl get pods** → then this ok

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes> kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
catalog-deployment-7bd968f76-5ndv5    1/1     Running   0          92s
catalog-deployment-7bd968f76-sdbxc    1/1     Running   0          106s
catalog-deployment-7bd968f76-vbchz    1/1     Running   0          82s
mongodb-statefulset-0                 1/1     Running   0          38s
```

--- 6. Load balancing requests across pods

Split the panel to have 3 terminal side by side:

Ter1: **kubectl get pods** → see 3 catalog-deployment images. Copy the Name of 1st one

Ter1: **kubectl logs catalog-deployment-7bd968f76-5ndv5 -f** →

Ter2: **kubectl logs catalog-deployment-7bd968f76-sdbxc -f** →

Ter3: **kubectl logs catalog-deployment-7bd968f76-vbchz -f** →

Logs should be seen as below. If we met issue like this

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
fail: Microsoft.Extensions.Diagnostics.HealthChecks.DefaultHealthCheckService[103]
      Health check mongodb completed after 875.8616ms with status Unhealthy and description '(null)'
      System.OperationCanceledException: The operation was canceled.
```

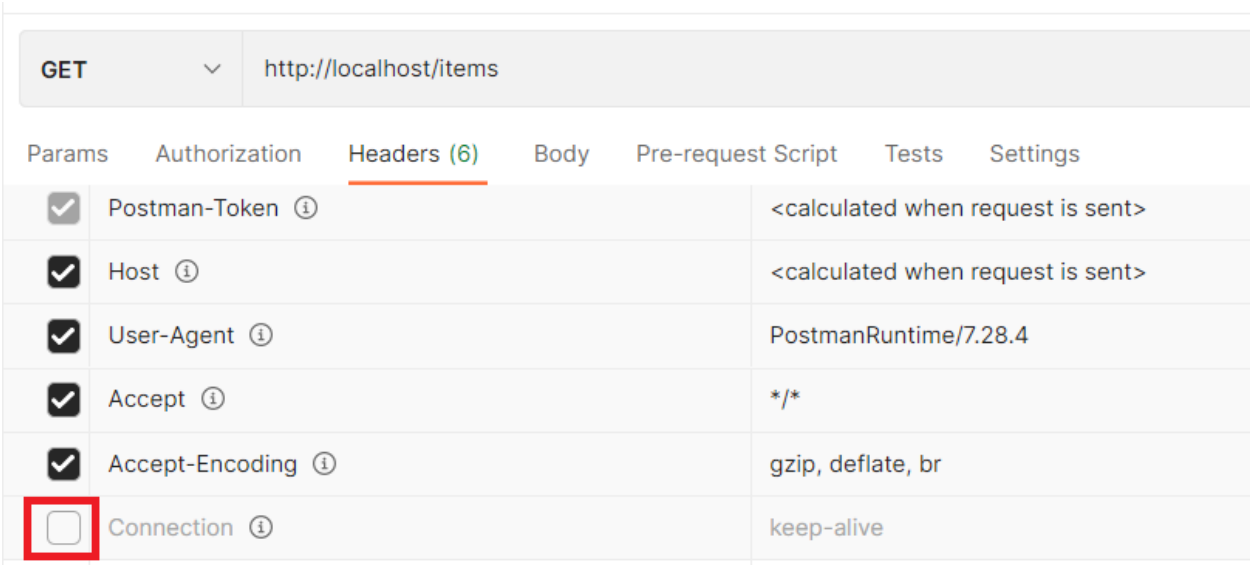
→ Workaround: go to Docker → Remove the images → Then they will automatically regenerated new one → Try again then you don't see fail Unhealthy

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes>  kubectl logs catalog-deployment-7bd968f76-1f9pb
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\kubernetes>

PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl logs catalog-deployment-7bd968f76-mjhf8
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>

PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> kubectl logs catalog-deployment-7bd968f76-qg51h
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:80
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /app
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog>
```

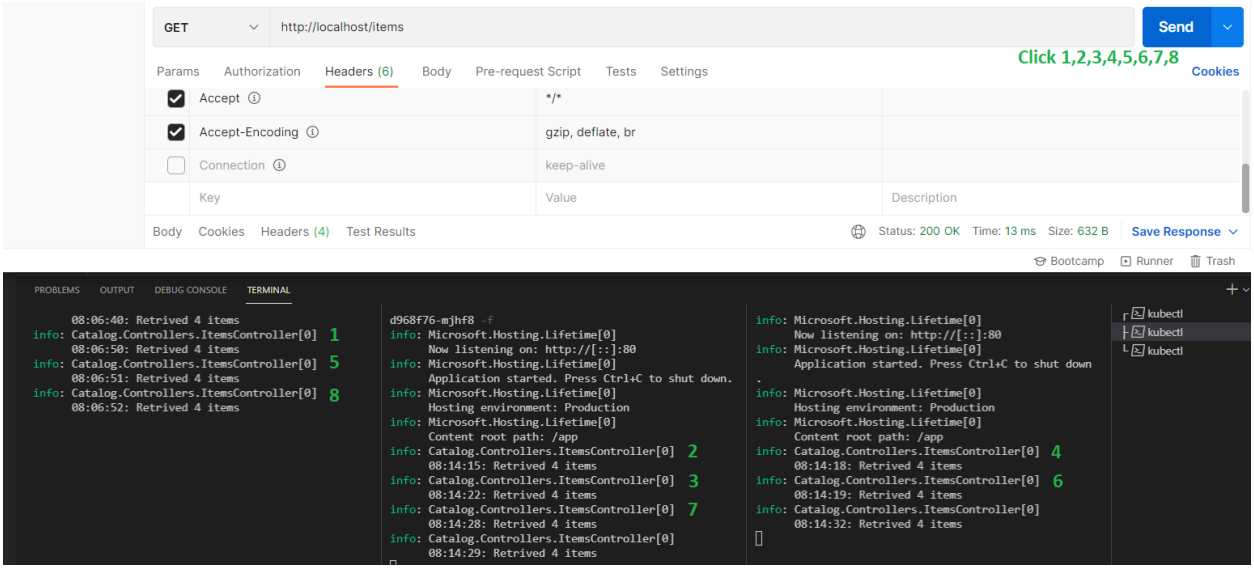
Open PostMan → Header → Uncheck the Connection, why ?

- persistent (liên tục)
- subsequent (tiếp theo)

Because if Connection = keep-alive → that is going to actually setup a persistent TCP connection between postman and our box → will prevent our little exercise from allowing different subsequent connections to go into different bots. So to see how thing work we will need to **uncheck** the connection = keep-alive as below



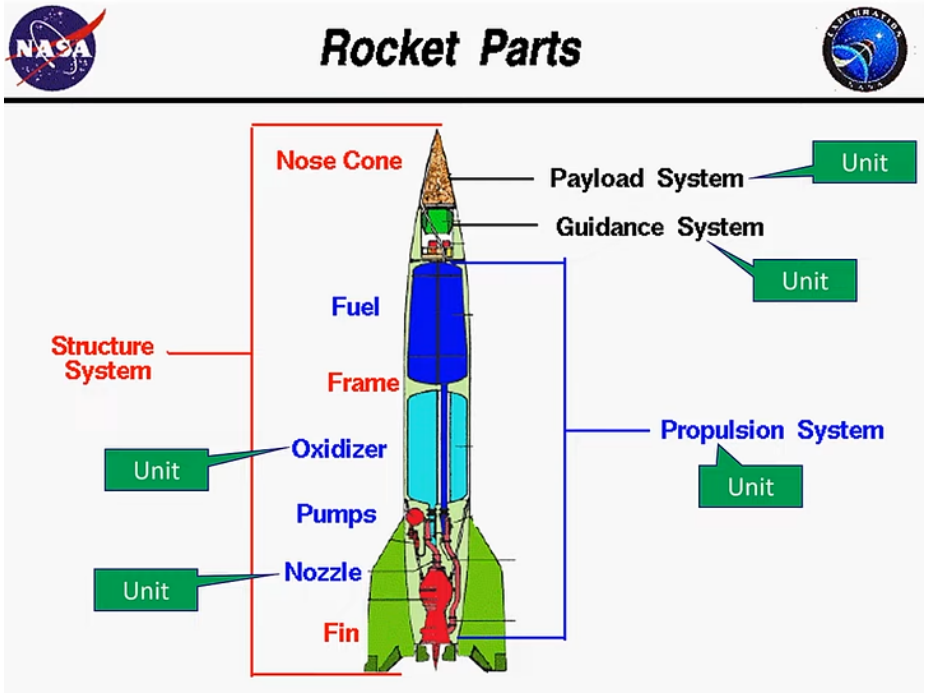
Then from PM, we click Send request 1,2,3,4,5,6,7,8 times, and we see it lands on the 1st part, 2nd part, 3rd part. A little bit random. The algorithm that's supposed to be using is **Round Robin** and that how the LoadBalancing in Kubernetes work.



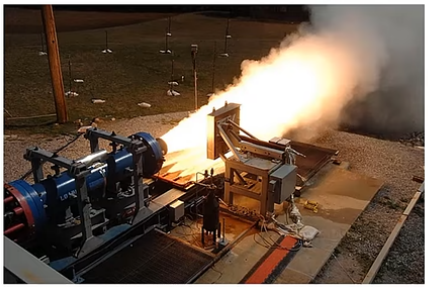
IX. UNIT TESTING AND TDD

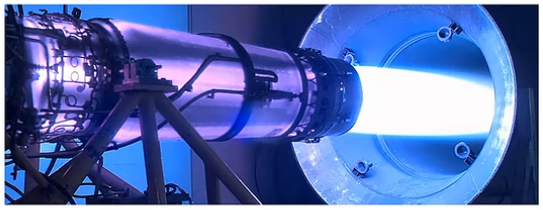
You will learn...

- What is unit testing
- What is Test Driven Development (TDD)
- Unit test a REST API controller via xUnit
- Mock dependencies via Moq
- Write better assertions via FluentAssertions
- TDD in practice



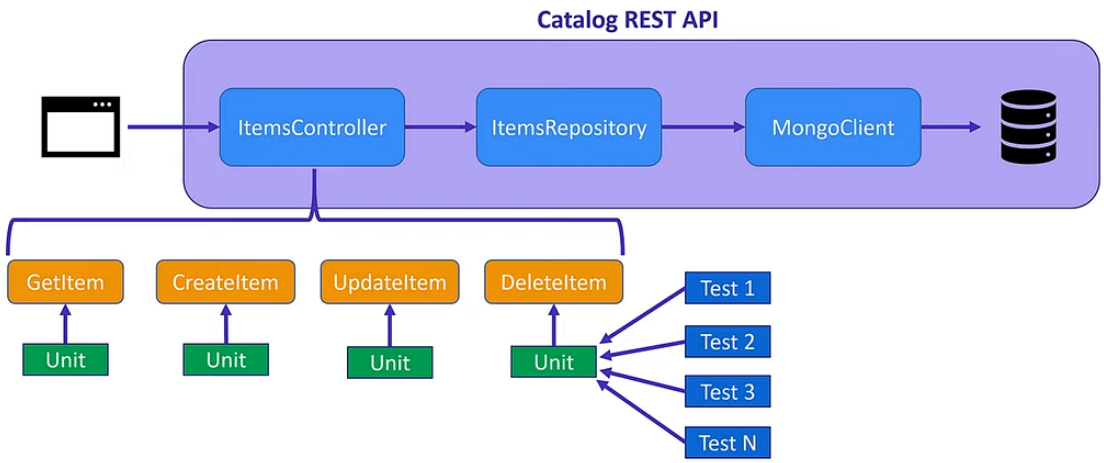
Testing one unit in isolation





Unit testing is a way to test every piece of code in isolation without external dependencies

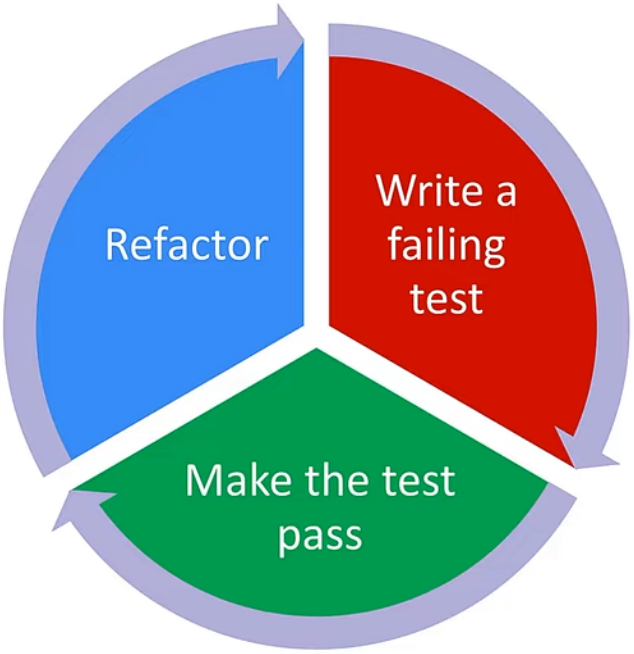
Testing the units of the REST API

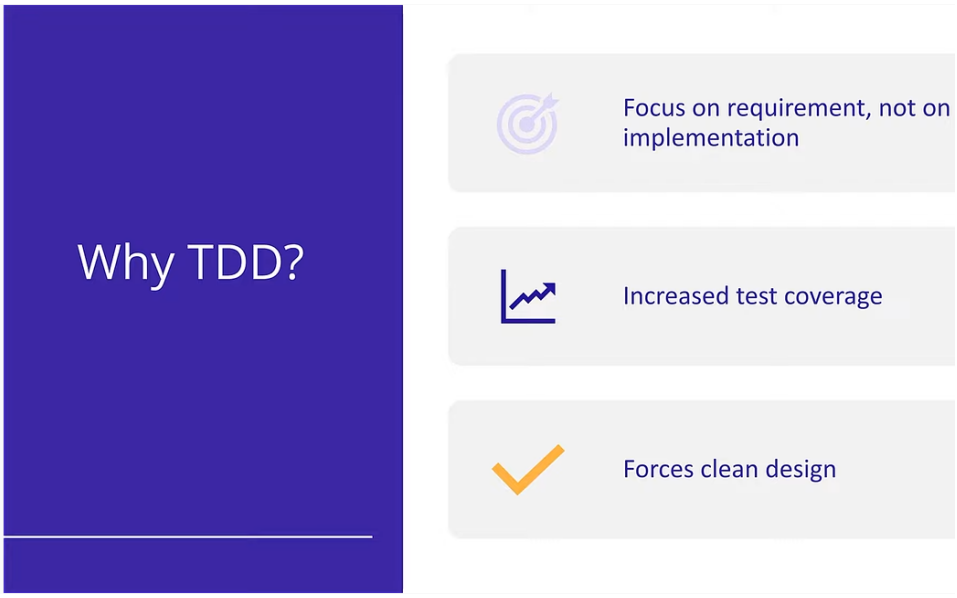


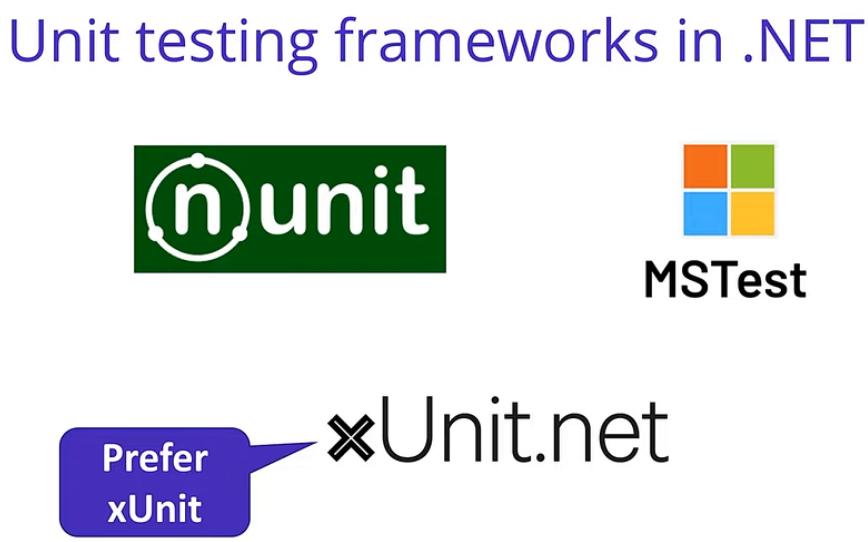
Why unit test?

- Quickly validate your code
- Make changes without fear
- Reduce the cost of bug fixes
- Provide live documentation

TDD is a software development approach where you write a test before you write just enough production code to make the failing test pass

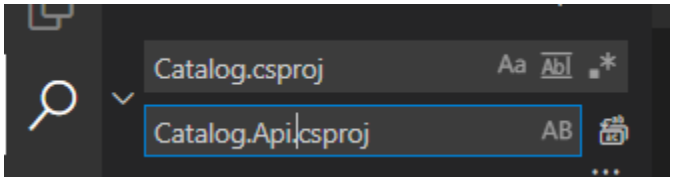




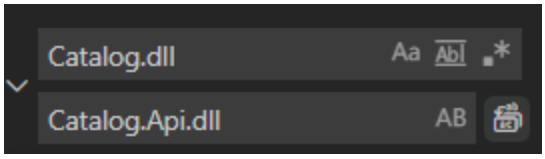


--- 1. Restructuring files and directories

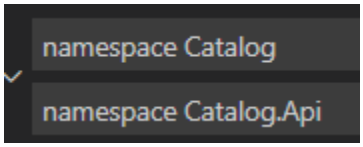
- Create new folder **Catalog.Api**
- Move all files and folder except folder {.vscode, bin, obj) into Catalog.Api
- Delete folders **bin** and **obj**
- Rename **Catalog.csproj** to **Catalog.Api.csproj**
- Search and replace all **Catalog.csproj** to **Catalog.Api.csproj**



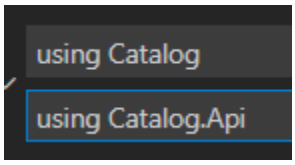
- Search and replace all **Catalog.dll** to **Catalog.Api.dll**



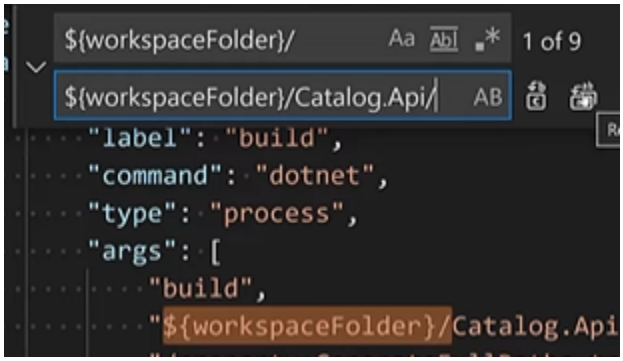
- Search and replace all **namespace Catalog** to **namespace Catalog.Api**



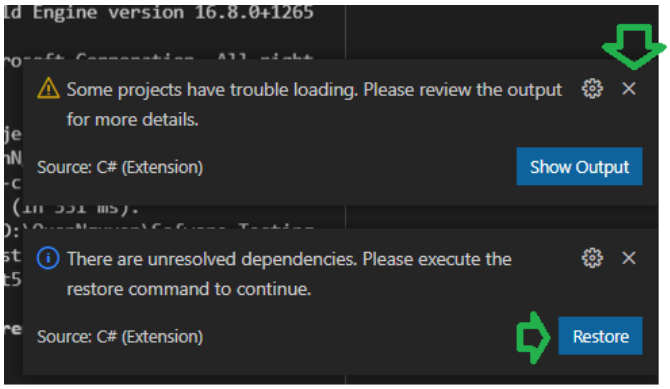
- Search and replace all **using Catalog** to **using Catalog.Api**



- Open file .vscode/**tasks.json** and replace all **${workspaceFolder}/** to **${workspaceFolder}/Catalog.Api/**



- Open file .vscode/**launch.json** and replace all **${workspaceFolder}/** to **${workspaceFolder}/Catalog.Api/**
- Press **Ctrl-Shift-B** → Close the 1st pop-up, and Restore for the 2nd pop-up



- Close terminal
- Open brand new terminal **cd catalog.api** → **docker build -t nguyenthuongquan/catalog:v3 .** → **docker images**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\catalog.api> docker images
REPOSITORY                          TAG                                                     IMAGE ID       CR
EATED          SIZE
nguyenthuongquan/catalog            v3                                                      e64539b9b7a0   6
seconds ago    225MB
nguyenthuognquan/catalog            v2                                                      c296fcdfd317   8
hours ago      225MB
nguyenthuongquan/catalog            v2                                                      c296fcdfd317   8
hours ago      225MB
nguyenthuongquan/catalog            v1                                                      a36f75b5eb94   29
```

--- 2. Creating the xUnit test project

- Go up to the parent directory **cd..**
- **→ dotnet new xunit -n Catalog.UnitTests**

```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> dotnet new x
unit -n Catalog.UnitTests
The template "xUnit Test Project" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on Catalog.UnitTests\Catalog.UnitTests.csproj...
  Determining projects to restore...
  Restored D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\Catal
og.UnitTests\Catalog.UnitTests.csproj (in 50.06 sec).
Restore succeeded.
```

--- 3. Building multiple projects in VS Code

- Bây giờ mình cần build, ko hẳn chỉ là 1, nhưng mà là 2 project. Nhưng giờ chưa config yet → Mở .vscode, lúc này mới only Catalog.Api thôi

```
.vscode > {} tasks.json > [ ] tasks > {} 0 > [ ] args > 1
1
2   "version": "2.0.0",
3   "tasks": [
4       {
5           "label": "build",
6           "command": "dotnet",
7           "type": "process",
8           "args": [
9               "build",
10              "${workspaceFolder}/Catalog.Api/Catalog.Api.csproj",
11              "/property:GenerateFullPaths=true",
12              "/consoleloggerparameters:NoSummary"
13          ],
```

- Giờ làm sao → tạo file **build.proj** in root

```
build.proj
1  <Project Sdk="Microsoft.Build.Traversal/3.0.3">
2      <ItemGroup>
3          <ProjectReference Include="**\*.*proj" />
4      </ItemGroup>
5  </Project>
```

- Update dòng 10 ở trên thành

```
10                "${workspaceFolder}/build.proj",
```

- Press **Ctrl-Shift-B → Ok**

```
  Determining projects to restore...
  All projects are up-to-date for restore.
  Catalog.Api -> D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\Catalog.Api\bin\Debug\net5.0\Catalog.Api.dll
  Catalog.UnitTests -> D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\Catalog.UnitTests\bin\Debug\net5.0\Catalog.UnitTests.dll
```

- Now we need to add reference to our API, because we are going to test the API controller. So we need the reference.
- Open Terminal (PowerShell) → **dir**

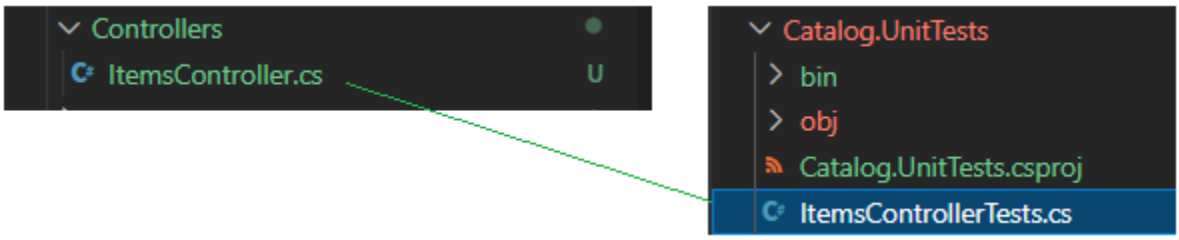
```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog> dir


    Directory: D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----         9/17/2021   5:49 PM                .vscode
d-----         9/28/2021   4:32 PM                Catalog.Api
d-----         9/28/2021   5:02 PM                Catalog.UnitTests
d-----         9/28/2021   5:02 PM                obj
-a----         9/28/2021   5:03 PM            145 build.proj
```

- Terminal → **cd catalog.unittests**
- Terminal → **dotnet add reference ..\Catalog.Api\Catalog.Api.csproj**
- Now we have the reference

```
Catalog.UnitTests > Catalog.UnitTests.csproj
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>

    <IsPackable>false</IsPackable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.7.1" />
    <PackageReference Include="xunit" Version="2.4.1" />
    <PackageReference Include="xunit.runner.visualstudio" Version="2.4.3">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="coverlet.collector" Version="1.3.0">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Catalog.Api\Catalog.Api.csproj" />
  </ItemGroup>

</Project>

```

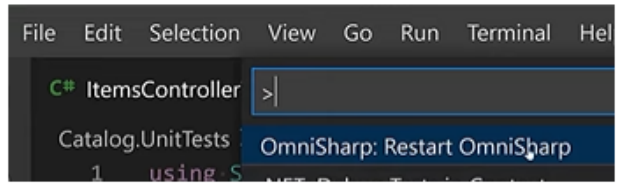
--- 4. Adding NuGet packages for unit testing

- Now we need couple of additional nuget packages
- Terminal → **dotnet add package Microsoft.Extensions.Logging.Abstractions** → this is because we are trying to test our controller class and in ItemsController.cs we are using ILogger, and for that we need that to get package **Logging Abstractions** we just added
- Terminal → **dotnet add package moq** → this is a little framework that can help you actually mock your classes, your classes are using in your controller, so that you can test only the pieces that you care about in controller but you don't have to worry about external dependencies
- Rename **UnitTest1.cs** to **ItemsControllerTests.cs** ← đó là quy tắc, vì mình có **ItemsControllers.cs** → test của nó sẽ là ItemsControllerTests.cs



- Có 1 lỗi màu đỏ → cách fix là **Ctrl Shift P** → **OmniSharp: Restart OmniSharp** → hết

```
using System;
using Xunit;

namespace Catalog.UnitTests
{
    0 references
    public class ItemsControllerTests
    {
        [Fact]
        0 references
        public void Test1()
```



- **[Fact]** là dấu hiệu nhận biết Method đó là test
- Name of a test need to follow convention:
  `[UnitOfWork]_[StateUnderTest]_[ExpectedBehavior]`

```
namespace Catalog.UnitTests
{
    0 references | Run All Tests | Debug All Tests
    public class ItemsControllerTests
    {
        [Fact]
        0 references | Run Test | Debug Test
        public void UnitOfWork_StateUnderTest_ExpectedBehavior()
        {

        }
    }
}
```

--- 5. Testing GetItemAsync unexisting item

- [UnitOfWork] = name of the method = **GetItemAsync** như của bên Controller
- [StateUnderTest] = cases/condition = **WithUnexistingItem**
- [ExpectedBehavior] = **ReturnsNotFound**
  → **GetItemAsync_WithUnexistingItem_ReturnsNotFound**

--- 6. Using the AAA pattern

- **//Arrange** → setup everything to be ready to execute the test {mocks, variables, inputs}
- **//Act** → execute the test
- **//Assert** → verify about the execution of the action

--- 7. Stubbing dependencies via Moq

Stub is a kind of a fake instance, or a fake version of the items/classes are going to be used just for the purpose of the test.

var **repositoryStub** = new Mock<IItemsRepository>();

- When Stub, you will not verify anything on this object itself.
- When Mock, you will in //Assert section, verify something

```
namespace Catalog.UnitTests
{
    0 references | Run All Tests | Debug All Tests
    public class ItemsControllerTests
    {
        [Fact]
        0 references | Run Test | Debug Test
        public async Task GetItemAsync_WithUnexistingItem_ReturnsNotFound()
        {
            //Arrange
            var repositoryStub = new Mock<IItemsRepository>();
            repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
                .ReturnsAsync((Item)null);

            var loggerStub = new Mock<ILogger<ItemsController>>();

            var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

            //Act
            var result = await controller.GetItemAsync(Guid.NewGuid());

            //Assert
            Assert.IsType<NotFoundResult>(result.Result);
        }
    }
}
```

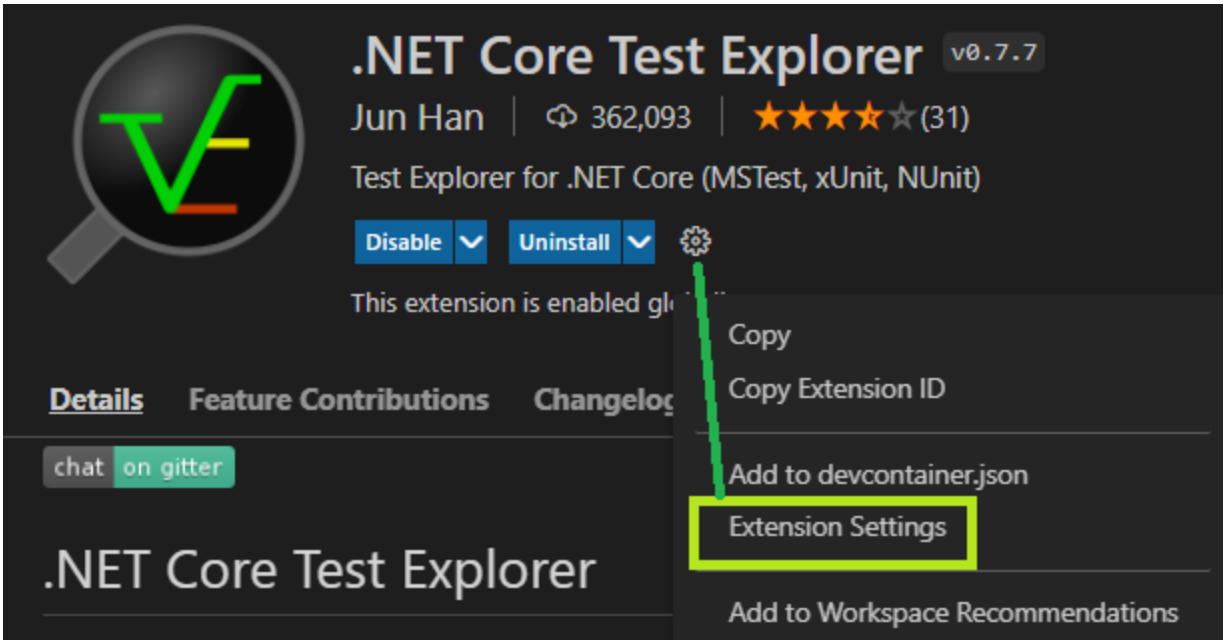
--- 8. Running tests in VS Code

1. Cách 1: Click 'Run Test'
2. Cách 2: Cmd **cd .\Catalog.UnitTests** → **dotnet test**

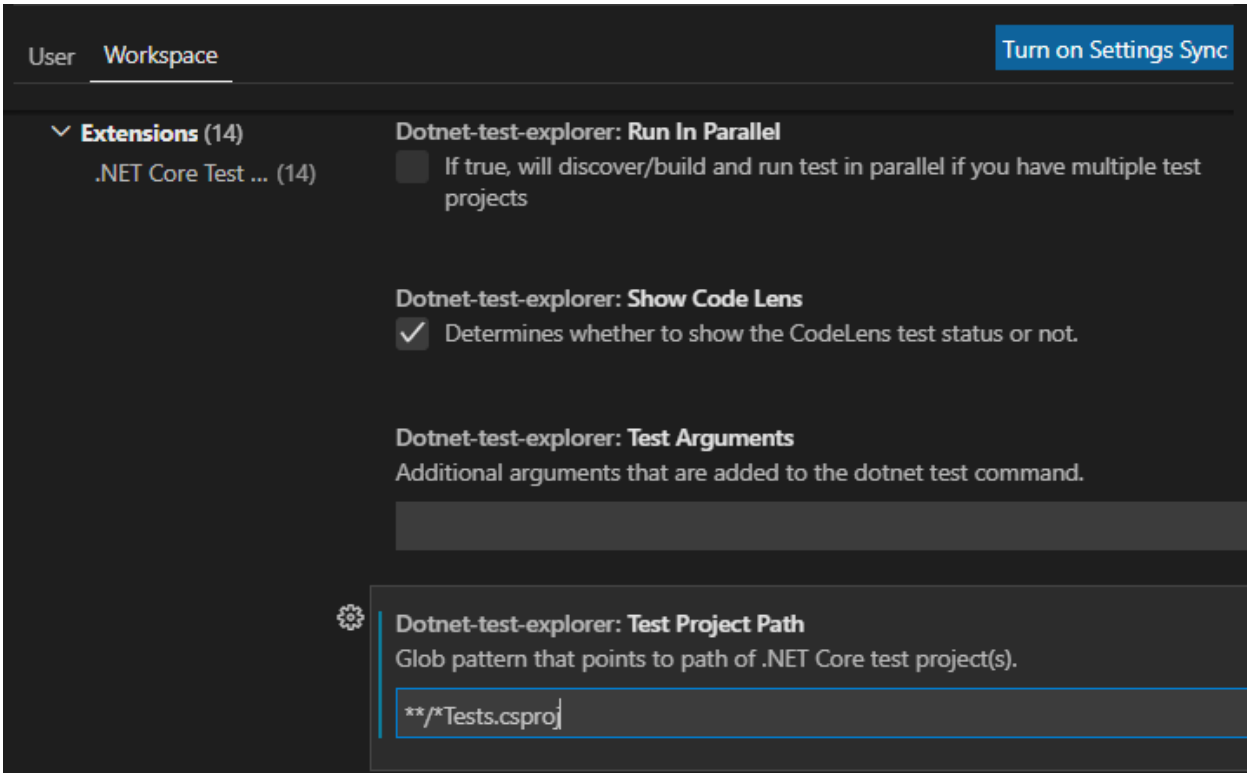
```
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:     1, Skipped:     0, Total:     1, Duration: 95 ms - Catalog.UnitTests.dll
 (net5.0)
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\Catalog.UnitTests>
```

--- 9. Using the .NET Core Test Explorer extension

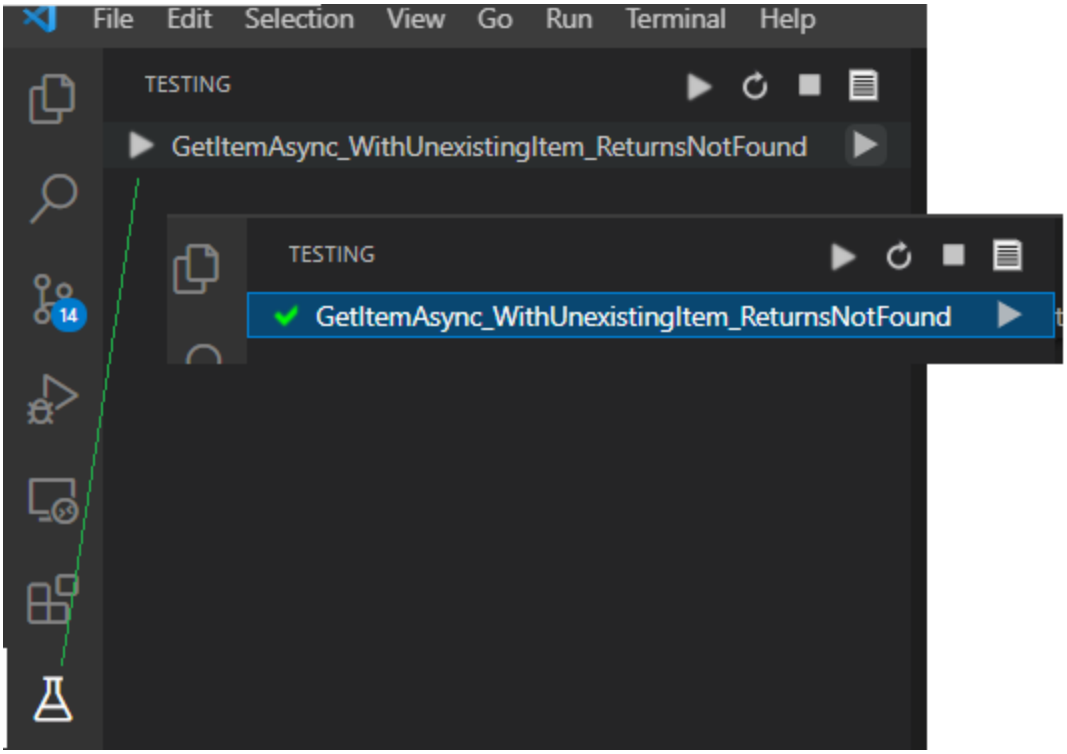
- Install extension **.net core test explorer**
- Go to Extension Settings



- Go to **Workspace** → Define **Test Project Path = **/*Tests.csproj**



- Now you can **see** the test, and you can **run** the Test and it is **green** like this



--- 10. Testing GetItemAsync existing item

- Ok, đang viết như bên dưới

--- 11. Using FluentAssertions

- Nhưng viết kiểu này Assert nhiều quá ko hay

```
[Fact]
0 references | Run Test | Debug Test
public async Task GetItemAsync_WithExistingItem_ReturnsNotFound()
{
    //Arrange
    var expectedItem =  CreateRandomItem();
    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(expectedItem);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.GetItemAsync(Guid.NewGuid());

    //Assert
    Assert.IsType<ItemDto>(result.Value);
    var dto = (result as ActionResult<ItemDto>).Value;
    Assert.Equal(expectedItem.Id, dto.Id);
    Assert.Equal(expectedItem.Name, dto.Name);

}
```

- Phải dùng **Fluent Assertion**
- Ctrl J → PowerShell → Make sure đang ở **\Catalog.UnitTests** → **dotnet add package FluentAssertions**

- Xong dùng cái này thể thay nguyên cục //Assert ở trên (nhớ **using FluentAssertions**)

```
//Assert
result.Value.Should().BeEquivalentTo(
    expectedItem,
    options => options.ComparingByMembers<Item>()
);
```

- Đổi cái thằng ở trên bằng thằng ở dưới luôn, dùng FluentAssertions.

```
//Assert
// Assert.IsType<NotFoundResult>(result.Result);
result.Result.Should().BeOfType<NotFoundResult>();
```

--- 12. Testing GetItemsAsync

```
private readonly Mock<IItemsRepository> repositoryStub = new();
6 references
private readonly Mock<ILogger<ItemsController>> loggerStub = new();
2 references
private readonly Random rand = new();
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task GetItemAsync_WithUnexistingItem_ReturnsNotFound()
{
    //Arrange
    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync((Item)null);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.GetItemAsync(Guid.NewGuid());

    //Assert
    result.Result.Should().BeOfType<NotFoundResult>();
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task GetItemAsync_WithExistingItem_ReturnsExpectedItem()
{
    //Arrange
    var expectedItem =  CreateRandomItem();

    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(expectedItem);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.GetItemAsync(Guid.NewGuid());

    //Assert
    result.Value.Should().BeEquivalentTo(
        expectedItem,
        options => options.ComparingByMembers<Item>()
    );
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task GetItemsAsync_WithExistingItems_ReturnsAllItems()
{
    //Arrange
    var expectedItems = new[] {CreateRandomItem(), CreateRandomItem(), CreateRandomItem()};

    repositoryStub.Setup(repo => repo.GetItemsAsync())
        .ReturnsAsync(expectedItems);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var actualsItems = await controller.GetItemsAsync();

    //Assert
    actualsItems.Should().BeEquivalentTo(
        expectedItems,
        options => options.ComparingByMembers<Item>()
    );
}
```

--- 13. Testing CreateItemAsync

```
private Item CreateRandomItem()
{
    return new()
    {
        Id = Guid.NewGuid(),
        Name = Guid.NewGuid().ToString(),
        Price = rand.Next(1000),
        CreatedDate = DateTimeOffset.UtcNow
    };
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task CreateItemAsync_WithItemToCreate_ReturnsCreatedItem()
{
    //Arrange
    var itemToCreate = new CreateItemDto() {
        Name = Guid.NewGuid().ToString(),
        Price = rand.Next(1000)
    };

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.CreateItemAsync(itemToCreate);

    //Assert
    var createdItem = (result.Result as CreatedAtActionResult).Value as ItemDto;
    itemToCreate.Should().BeEquivalentTo(
        createdItem,
        options => options.ComparingByMembers<ItemDto>().ExcludingMissingMembers()
    );
    createdItem.Id.Should().NotBeEmpty();
    createdItem.CreatedDate.Should().BeCloseTo(DateTimeOffset.UtcNow, TimeSpan.FromMilliseconds(1000));
}
```

--- 14. Testing UpdateItemAsync

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task UpdateItemAsync_WithExistingItem_ReturnsNoContent()
{
    //Arrange
   var existingItem =  CreateRandomItem();

    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(existingItem);

    var itemId = existingItem.Id;
    var itemToUpdate = new UpdateItemDto() {
        Name = Guid.NewGuid().ToString(),
        Price = existingItem.Price + 3
    };

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.UpdateItemAsync(itemId, itemToUpdate);

    //Assert
    result.Should().BeOfType<NoContentResult>();
}
```

--- 15. Testing DeleteItemAsync

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task DeleteItemAsync_WithExistingItem_ReturnsNoContent()
{
    //Arrange
   var existingItem =  CreateRandomItem();

    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(existingItem);


    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.DeleteItemAsync(existingItem.Id);

    //Assert
    result.Should().BeOfType<NoContentResult>();
}
```

--- 16. Refactoring and catching regressions

Bây giờ mình sẽ make 2 big changes để xem, Unit Test có prevent breaking anything ko nha.

1. Vào Entities/Items.cs đổi Type Record sang Class → Update code
2. Assert cho Type = Class thì mình có thể đổi lại cho gọn như sau. Ko cần option, options là cho Type = Record

Type = Record

```
//Assert
actualsItems.Should().BeEquivalentTo(
    expectedItems,
    options => options.ComparingByMembers<Item>()
);
```

Type = Class

```
//Assert
result.Value.Should().BeEquivalentTo(expectedItem);
```

Ok xem lại nha:

```
11 references
private readonly Mock<IItemsRepository> repositoryStub = new();
6 references
private readonly Mock<ILogger<ItemsController>> loggerStub = new();
2 references
private readonly Random rand = new();
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task GetItemAsync_WithUnexistingItem_ReturnsNotFound()
{
    //Arrange
    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync((Item)null);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.GetItemAsync(Guid.NewGuid());

    //Assert
    result.Result.Should().BeOfType<NotFoundResult>();
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task GetItemAsync_WithExistingItem_ReturnsExpectedItem()
{
    //Arrange
    var expectedItem =  CreateRandomItem();

    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(expectedItem);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.GetItemAsync(Guid.NewGuid());

    //Assert
    result.Value.Should().BeEquivalentTo(expectedItem);
}
```

```
private Item CreateRandomItem()
{
    return new()
    {
        Id = Guid.NewGuid(),
        Name = Guid.NewGuid().ToString(),
        Description = Guid.NewGuid().ToString(),
        Price = rand.Next(1000),
        CreatedDate = DateTimeOffset.UtcNow
    };
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task GetItemsAsync_WithExistingItems_ReturnsAllItems()
{
    //Arrange
    var expectedItems = new[] {CreateRandomItem(), CreateRandomItem(), CreateRandomItem()};

    repositoryStub.Setup(repo => repo.GetItemsAsync())
        .ReturnsAsync(expectedItems);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var actualsItems = await controller.GetItemsAsync();

    //Assert
    actualsItems.Should().BeEquivalentTo(expectedItems);
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task CreateItemAsync_WithItemToCreate_ReturnsCreatedItem()
{
    //Arrange
    var itemToCreate = new CreateItemDto(
        Guid.NewGuid().ToString(),
        Guid.NewGuid().ToString(),
        rand.Next(1000)
    );

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.CreateItemAsync(itemToCreate);

    //Assert
    var createdItem = (result.Result as CreatedAtActionResult).Value as ItemDto;
    itemToCreate.Should().BeEquivalentTo(
        createdItem,
        options => options.ComparingByMembers<ItemDto>().ExcludingMissingMembers()
    );
    createdItem.Id.Should().NotBeEmpty();
    createdItem.CreatedDate.Should().BeCloseTo(DateTimeOffset.UtcNow, TimeSpan.FromMilliseconds(1000));
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task UpdateItemAsync_WithExistingItem_ReturnsNoContent()
{
    //Arrange
   var existingItem =  CreateRandomItem();

    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(existingItem);

    var itemId = existingItem.Id;

    var itemToUpdate = new UpdateItemDto(
        Guid.NewGuid().ToString(),
        Guid.NewGuid().ToString(),
        existingItem.Price + 3
    );

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.UpdateItemAsync(itemId, itemToUpdate);

    //Assert
    result.Should().BeOfType<NoContentResult>();
}
```

```
[Fact]
✓ | 0 references | Run Test | Debug Test
public async Task DeleteItemAsync_WithExistingItem_ReturnsNoContent()
{
    //Arrange
   var existingItem =  CreateRandomItem();

    repositoryStub.Setup(repo => repo.GetItemAsync(It.IsAny<Guid>()))
        .ReturnsAsync(existingItem);


    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    var result = await controller.DeleteItemAsync(existingItem.Id);

    //Assert
    result.Should().BeOfType<NoContentResult>();
}
```

--- 17. Using TDD to test a yet to be created method

<u>Scenario:</u> New requirement: we need an api which returns all the items of which name contains the keyword. E.g: “potion” →

- High potion
- Low potion
- Super potion

Step1: Make the test first. Xong build Ctrl-Shift-B → build fail as expected → Red phase of TDD

```
[Fact]
0 references | Run Test | Debug Test
public async Task GetItemAsync_WithMatchingItems_ReturnsMatchingItems()
{
    //Arrange
    var allItems = new[]
    {
        new Item() { Name = "Potion"},
        new Item() { Name = "Antidote"},
        new Item() { Name = "Hi-Potion"}
    };

    var nameToMatch = "Potion";

    repositoryStub.Setup(repo => repo.GetItemsAsync())
        .ReturnsAsync(allItems);

    var controller = new ItemsController(repositoryStub.Object, loggerStub.Object);

    //Act
    IEnumerable<ItemDto> foundItems = await controller.GetItemsAsync(nameToMatch);

    //Assert
    foundItems.Should().OnlyContain(
        item => item.Name == allItems[0].Name || item.Name == allItems[2].Name
    );
}
```

Step2:

```
r.GetItemsAsync(nameToMatch);

    Add parameter to 'ItemsController.GetItemsAsync()'
    Generate method 'ItemsController.GetItemsAsync'
.N  Change 'GetItemsAsync' to 'GetItemAsync'.
```

--- 18. Going back to green by fixing the failing test

Step3: No need step2, just go refactor the GetItemsAsync() adding attribute which can be null.

```
//GET /items
[HttpGet]
2 references
public async Task<IEnumerable<ItemDto>> GetItemsAsync()
{
    var items = (await repository.GetItemsAsync())
        .Select(item => item.AsDto());
    logger.LogInformation($"{DateTime.UtcNow.ToString("hh:mm:ss")}: Retrived {items.Count()} items");
    return items;
}
```

```
//GET /items
[HttpGet]
2 references
public async Task<IEnumerable<ItemDto>> GetItemsAsync(string name = null)
{
    var items = (await repository.GetItemsAsync())
        .Select(item => item.AsDto());

    if (!string.IsNullOrWhiteSpace(name))
    {
        items = items.Where(item => item.Name.Contains(name, StringComparison.OrdinalIgnoreCase));
    }

    logger.LogInformation($"{DateTime.UtcNow.ToString("hh:mm:ss")}: Retrived {items.Count()} items");

    return items;
}
```

Step4: → Run → Passed all

```
TESTING
✔ CreateItemAsync_WithItemToCreate_ReturnsCreatedItem
✔ DeleteItemAsync_WithExistingItem_ReturnsNoContent
✔ GetItemAsync_WithExistingItem_ReturnsExpectedItem
✔ GetItemAsync_WithMatchingItems_ReturnsMatchingItems
✔ GetItemAsync_WithUnexistingItem_ReturnsNotFound
✔ GetItemsAsync_WithExistingItems_ReturnsAllItems
✔ UpdateItemAsync_WithExistingItem_ReturnsNoContent
```

--- 19. Testing the new controller method in Postman

1. **DEPLOY API**
    a. Open **Catalog.Api/kubernetes/catalog.yaml** → update nguyenthuongquan/catalog:**v4**
    b. **cd .\Catalog.Api\kubernetes**
    c. **kubectl apply -f .\catalog.yaml**
    d. **kubectl get deployments**
    e. **kubectl get pods**

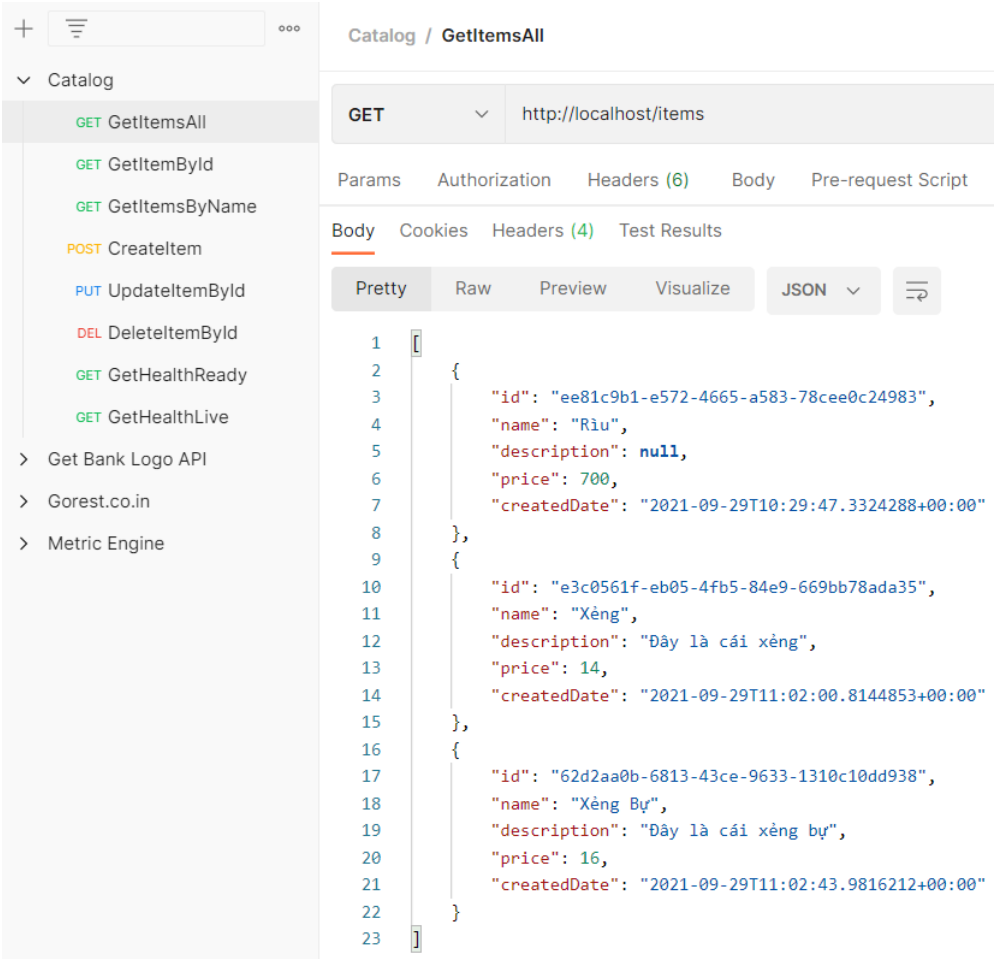
```
NAME                                  READY   STATUS             RESTARTS
catalog-deployment-7bd968f76-lf9pb    1/1     Running            2
catalog-deployment-7bd968f76-mjhf8    1/1     Running            2
catalog-deployment-7bd968f76-qg5lh    1/1     Running            2
catalog-deployment-cd768ffc8-h7rtw    0/1     ImagePullBackOff   0
mongodb-statefulset-0                 1/1     Running            1
```

f. **cd ..** → để trở ra ngoài Catalog.Api (thoát kubernetes)
g. **docker build -t nguyenthuongquan/catalog:v4** → build èn èn
h. **docker login**
i. **docker push nguyenthuongquan/catalog:v4**
j. **cd .\kubernetes**
k. **Kubectl get pods** → mới tinh

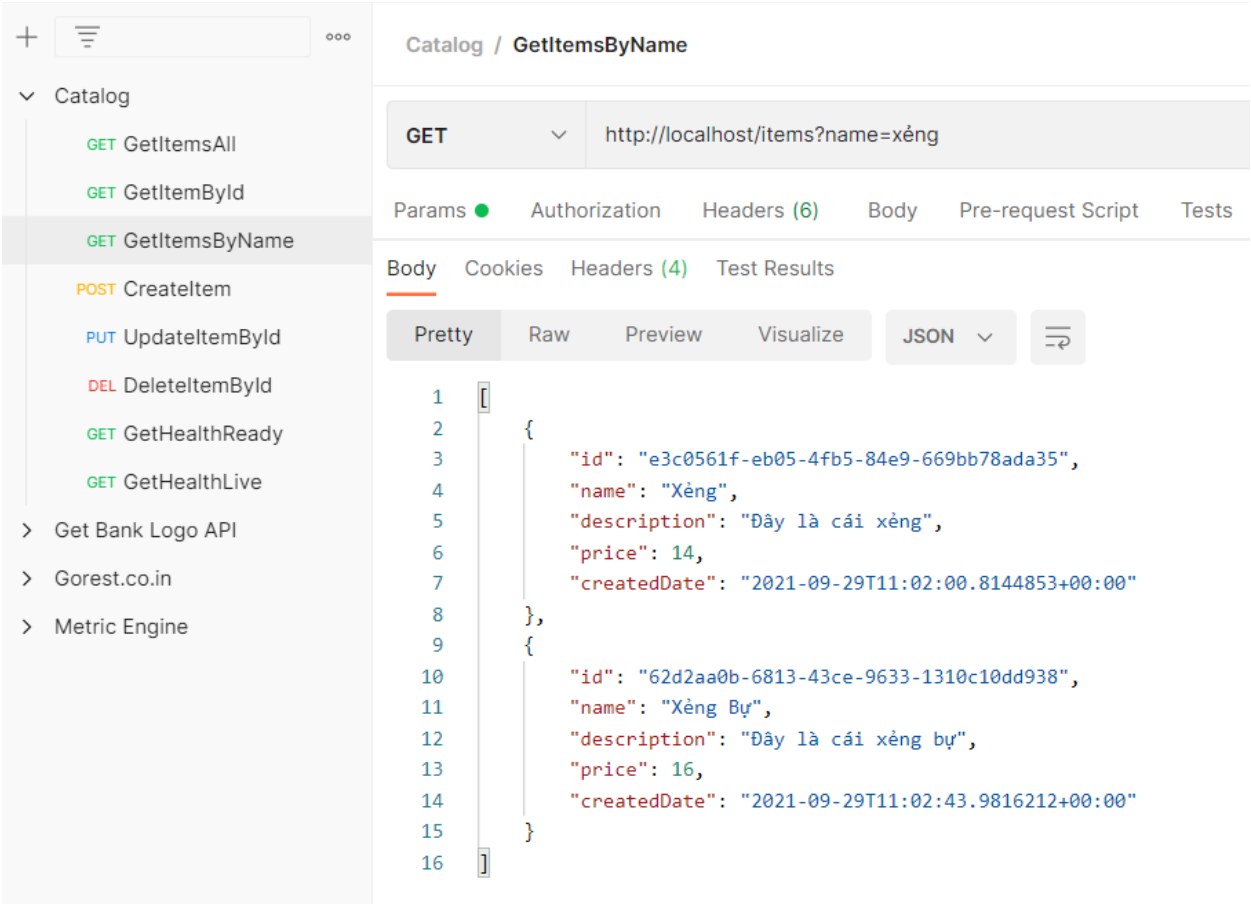
```
PS D:\QuanNguyen\Sofware-Testing\Projects-Demo\rest-api-csharp\Catalog\Catalog.Api\kubernetes> kubectl
NAME                                  READY   STATUS    RESTARTS   AGE
catalog-deployment-cd768ffc8-h7rtw    1/1     Running   0          16m
catalog-deployment-cd768ffc8-hft4w    1/1     Running   0          11m
catalog-deployment-cd768ffc8-vl8rb    1/1     Running   0          11m
mongodb-statefulset-0                 1/1     Running   1          27h
```

2. TEST POSTMAN

Open Postman, execute all the api below → Good!



GetItem by Name → Good!


Catalog
GET GetItemsAll
GET GetItemById
GET GetItemsByName
POST CreateItem
PUT UpdateItemById
DEL DeleteItemById
GET GetHealthReady
GET GetHealthLive
Get Bank Logo API
Gorest.co.in
Metric Engine
Catalog / GetItemsByName
GET
http://localhost/items?name=xẻng
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": "e3c0561f-eb05-4fb5-84e9-669bb78ada35",
"name": "Xẻng",
"description": "Đây là cái xẻng",
"price": 14,
"createdDate": "2021-09-29T11:02:00.8144853+00:00"
},
{
"id": "62d2aa0b-6813-43ce-9633-1310c10dd938",
"name": "Xẻng Bự",
"description": "Đây là cái xẻng bự",
"price": 16,
"createdDate": "2021-09-29T11:02:43.9816212+00:00"
}
]

---------- THE END ----------