|  |
| --- |
| 63734a0bd9a5160c7013baab3ecaed1e  **TỔNG HỢP KIẾN THỨC**  **TP.HCM, ngày 9 tháng 3 năm 2025** |

**MỤC LỤC**

[I. .NET Core 3](#_Toc23056)

[ .NET Core vs .NET Framework 3](#_Toc8701)

[1. C# (1) 3](#_Toc4536)

[ .Net framework 3](#_Toc29041)

[ C# 3](#_Toc30013)

[ Top-level statements 4](#_Toc12769)

[ OOP 4](#_Toc20744)

[2. C# (2) 8](#_Toc11465)

[ Biểu thức Lamda 8](#_Toc1173)

[ Linq 8](#_Toc14325)

[ Loading Strategies (Chiến lược nạp dữ liệu) 8](#_Toc14774)

[1. Lazy loading 8](#_Toc15428)

[2. Earger loading 8](#_Toc19162)

[3. Explicit Loading 9](#_Toc4933)

[ Queue / Stack 9](#_Toc30894)

[1. Hàng đợi Queue 9](#_Toc31463)

[2. Ngăn xếp Stack 10](#_Toc20875)

[ Collection - List 11](#_Toc6048)

[3. Networking 12](#_Toc400)

[3.1 Migration 12](#_Toc168)

[4. ASP.NET MVC 13](#_Toc14427)

[4.1 Model Binding (Ánh xạ dữ liệu) & Model Validation 13](#_Toc28481)

[ Binding 13](#_Toc31166)

[ Validation 14](#_Toc12588)

[ Dựng Validation 15](#_Toc6370)

[ Dựng Binding 16](#_Toc16303)

[ Syntax 16](#_Toc22963)

[5. ASP.NET CORE 17](#_Toc10197)

[5.1 Middleware 17](#_Toc21137)

[ Pipeline 17](#_Toc5601)

[ Thứ tự middleware 19](#_Toc14486)

[ MVC Endpoint (Endpoint Middleware) 19](#_Toc9468)

[ Custom middleware 19](#_Toc21227)

[ Thêm phương thức mở rộng 21](#_Toc23974)

[ Truyền dữ liệu giữa các Middleware 22](#_Toc30222)

[5.2 Security and Identity 23](#_Toc24699)

[ Authentication 23](#_Toc11762)

[1. Authentication với JWT 23](#_Toc26692)

[II Mô hình phát triển phần mềm 26](#_Toc13913)

[1. Waterfall (Mô hình thác nước) 26](#_Toc25967)

[2. Mô hình Scum 27](#_Toc6880)

[ Ghi chú 28](#_Toc18015)

**I. .NET Core**

* **.NET Core vs .NET Framework**

|  |  |  |
| --- | --- | --- |
| **BASED ON** | **.NET core** | **.NET Framework** |
| **Open Source** | √ | × |
| **Cross-Platform** | Windows,Linux, Mac OS | Only windows |
| **Application Models** | Focus on Web,windows mobile, windows store | Desktop,web applications,windows forms, WPF |
| **Installation** | Packaged and installed independently,Cross-platform | A single package |
| **Support** | Microserives, Rest APIs | Rest APIs |
| **WCF Services** | × | √ |
| **Performance and Scalability** | high | less |
| **Android Development** | iOS, Android, Windows phones | × |
| **Packaging and Shipping** | .Net core is shipped as a collection of Nugget packages | All the libraries of .NET Framework are packaged and shipped together |
| **Deployment Model** | Updated instantly on one machine at at a time => flexible | Deployed on the Internet Information Server only |
| **Compatibility** | Windows,Linux, Mac OS | Only windows |
| **CLI Tools** |  |  |
| **Security** | × | √ |

1. **C# (1)**

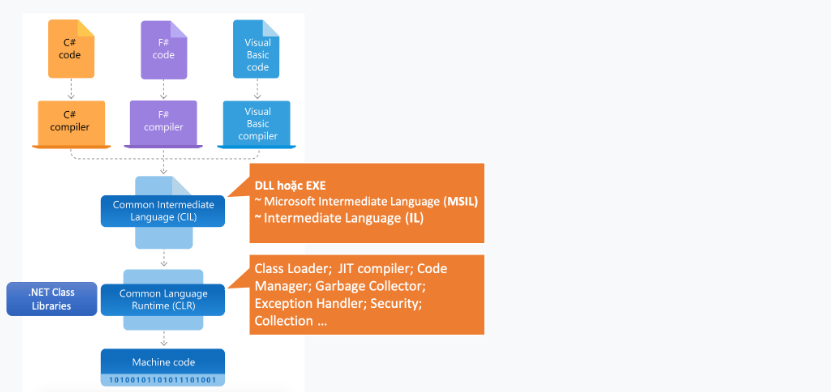
* **.Net framework**

-Là nền tảng phần mềm được phát triển bởi Microsoft, phiên bản 1.0 giới thiệu vào năm 2002, ban đầu .NET chỉ hỗ trợ trên các hệ điều hành Windows và mã nguồn .Net đóng

-Năm 2016, .Net core 1.0 phát hành trên đa nền tảng (Windows, Linux, MacOs), các ứng dụng .Net core có hiệu suất cao

-.Net bao gồm thành phần CLR (Common Language Runtime) → security,memory management,exception handling,… → “managed code” và các lớp thư viện Framework Class Library (FCL)

-Thư viện lớp (FCL) là tập hợp class,interface,… thực thi các tác vụ, từ các FCL → phát triển, mở rộng các chức năng khác nhau với C#.



* **C#**

Là ngôn ngữ lập trình hướng đối tượng, dùng để phát triển nhiều loại ứng dụng (Web,Windows application, Linux, MacOs,…) hiệu năng cao và bảo mật

Một số đặc điểm:

- Cú pháp đơn giản

- Các phương thức, thuộc tính bắt buộc phải khai báo trong lớp

- Biến được khai báo với kiểu dữ liệu

- …

* **Top-level statements**

Top-Level Statements (Câu lệnh cấp cao) trong C# cho phép viết chương trình mà không cần khai báo một lớp hoặc phương thức Main() , giúp đơn giản hóa mã nguồn, chủ yếu cho các ứng dụng nhỏ hoặc script (từ C# 9.0)

Cách hoạt động:

- Nếu file chứa Top-level statements, không thể khai báo lớp Main()

- Chỉ được phép có 1 file duy nhất chứa Top-level statements trong dự án

- Tất cả dòng lệnh trong file sẽ được đặt trong một phương thức Main() do trình biên dịch tạo ra

* **OOP**

Lập trình hướng đối tượng (**Object-Oriented Programming - OOP**) là một mô hình lập trình dựa trên việc sử dụng đối tượng (**objects**) để thiết kế và xây dựng ứng dụng. C# là một ngôn ngữ lập trình hướng đối tượng thuần túy.

1. **Bốn nguyên tắc chính**

* **Đóng gói (Encapsulation)**:

Đóng gói dữ liệu và ẩn thông tin, cho phép truy cập thông qua các phương thức công khai (public methods :public,private,protected,internal, protectet internal).

* **Kế thừa (Inheritance)**:

Kế thừa thuộc tính/phương thức từ lớp cha

* **Đa hình (Polymorphism)**:

Phương thức có nhiều hình thức khác nhau (**Overloading , Overriding**)

**- Overloading** (Nạp chồng phương thức) cùng phương thức khác tham số.

**- Overriding** (Ghi đè phương thức) lớp con sử dụng (ghi đè) phương thức cha bằng từ khóa **override**,

* **Trừu tượng (Abstraction)**:

Ẩn đi chi tiết không cần thiết, chỉ cung cấp giao diện bên ngoài (Abstract class, Interface)

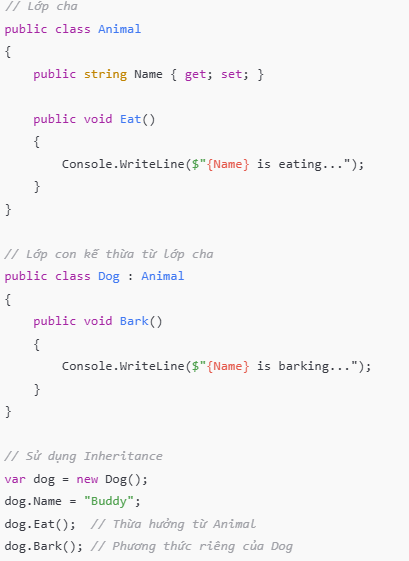
|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| Có hoặc không có thân hàm | Chỉ liệt kê hoạt động (tên hàm, không có thân hàm) |
| Lớp con kế thừa 1 abstract class | Lớp con kế thừa nhiều interface |
| Thể hiện tính trừu tượng | |
| Đều được kế thừa bởi lớp con | |

1. **Ví dụ**

* **Tính đóng gói**

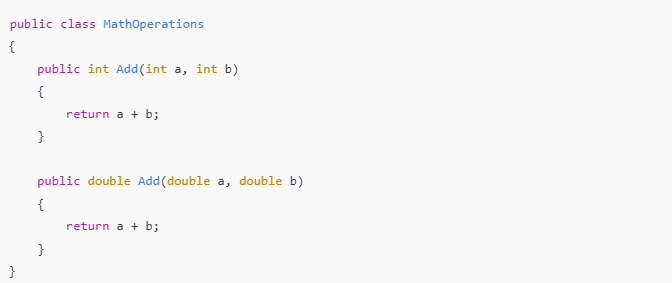


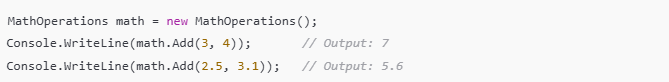
* **Tính kế thừa**



* **Tính đa hình**

**Overloading()**





**Overriding()**



* **Tính trừu tượng**

**Abtract class**



**Interface**



1. **C# (2)**

* **Biểu thức Lamda**

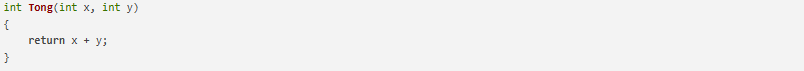
**Biểu thức Lamda - biểu thức nặc danh (Anoymous),** một biểu thức khai báo giống phương thức (hàm) nhưng thiếu tên, Cú pháp sử dụng toán tử **“ => ” .**

Ví dụ 1



Ví dụ 2





* **Linq**

LINQ (Language Integrated Query) - ngôn ngữ truy vấn tích hợp, tích hợp cú pháp truy vấn (gần giống với các câu lệnh sql) vào bên trong ngôn ngữ lập trình C# giúp truy vấn nhiều nguồn dữ liệu (SQL DB, XML, List,..) với cùng cú pháp

|  |  |
| --- | --- |
| **Các mệnh đề** | **Ý nghĩa** |
| Select |  |
| Where |  |
| Include |  |
| ThenInclude |  |
| Groupby |  |
| Join |  |
| GroupJoin |  |
| Orderby |  |

* **Loading Strategies (Chiến lược nạp dữ liệu)**

1. **Lazy loading**

- Các đối tượng liên quan (đối tượng con) không được tải tự động với đối tượng mẹ của nó cho đến khi chúng được yêu cầu

- Sử dụng Lazy Loading khi bạn đang sử dụng bộ sưu tập một-nhiều.

- Sử dụng Lazy Loading khi chắc chắn rằng không sử dụng các thực thể liên quan ngay lập tức.



1. **Earger loading**

- Eager Loading giúp tải tất cả các thực thể cần thiết cùng một lúc. tức là các đối tượng liên quan (đối tượng con) được tải tự động với đối tượng mẹ của nó.

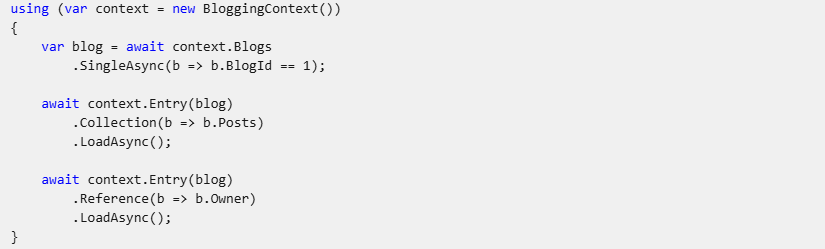
- Sử dụng Eager Loading khi các mối quan hệ không quá nhiều. Do đó, Eager Loading là một phương pháp tốt để giảm các truy vấn tiếp theo trên Server.

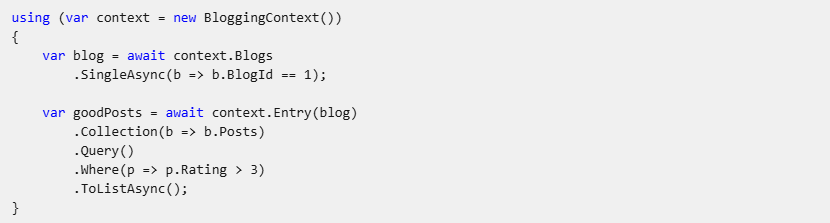
- Sử dụng Eager Loading khi bạn chắc chắn rằng sẽ sử dụng các thực thể liên quan với thực thể chính ở mọi nơi.



1. **Explicit Loading**

**- Explicit Loading** (tải dữ liệu tường minh) là kỹ thuật trong EF, giúp tải dữ liệu liên quan thủ công sau khi thực thể chính được truy vấn.





* **Queue / Stack**

1. **Hàng đợi Queue**

là mô hình **FIFO**( First In Fisrt Out - vào trước ra trước).

<https://learn.microsoft.com/en-us/dotnet/api/system.collections.queue?view=net-6.0>



|  |  |
| --- | --- |
| **Các phương thức, thuộc tính** | |
| Count | Lấy tổng số phần tử |
| Enqueue | Đưa phần tử vào cuối hàng đợi |
| Dequeue | Đọc - loại phần tử đầu (không có phần tử => lỗi) |
| Peek | Đọc phần tử đầu hàng đợi |
| Clone |  |
| Clear |  |
| …. |  |

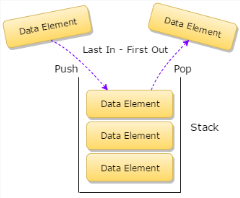
Ví dụ:



1. **Ngăn xếp Stack**

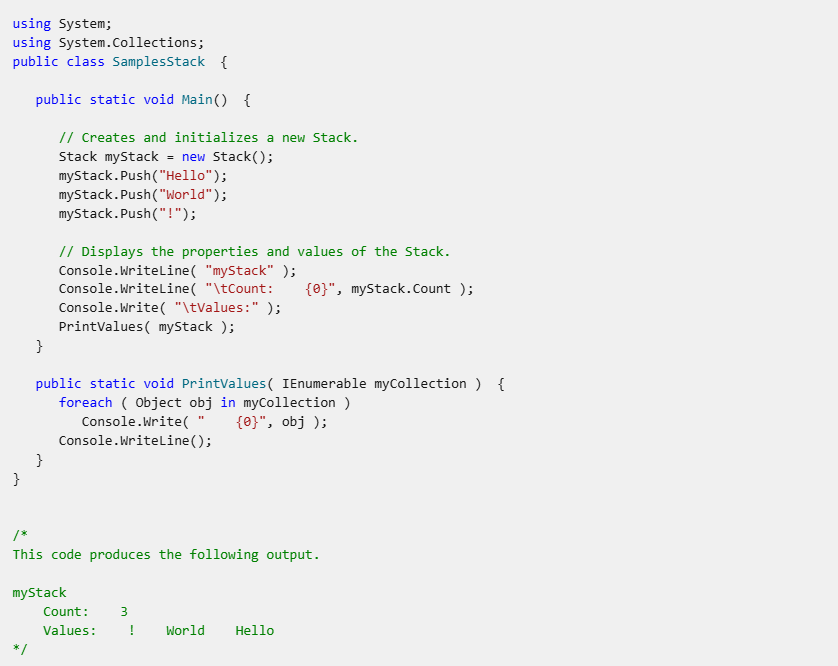
là mô hình **LIFO** (Last In First Out - vào sau ra trước)

<https://learn.microsoft.com/en-us/dotnet/api/system.collections.stack?view=net-6.0>



|  |  |
| --- | --- |
| **Các phương thức và thuộc tính** | |
| Count | Lấy tổng số phần tử |
| Push | Thêm một phần tử vào đỉnh Stack |
| Pop | Đọc - xóa phần tử đỉnh Stack |
| Peak | Đọc từ đỉnh Stack |
| Contains | Kiểm tra phần tử có trong Stack không |
| Clear |  |
| Clone |  |
| …. |  |

Ví dụ



* Collection - List

**- Collection** (bộ, tập hợp) là một nhóm đối tượng có sự liên quan với nhau. Có nhiều loại Collection, chúng tập hợp ở namspace **System.Collections.**

- .NET cung cấp một số lớp Collection kiểu Generic như:

List<T>,Dictionary<TKey,TValue>, Stack<T>,…

|  |  |
| --- | --- |
| **Các interface về collection thường dùng** | |
| **IEnumerable<T>** | Triển khai nó nếu muốn duyệt phần tử bằng **foreach**, nó định nghĩa phương thức GetEnumerator trả về một enumerator |
| **ICollection<T>** | Giao diện này được triển khai bở các generic collection. Với nó lấy tổng phần tử bằng thuộc tính **Count**, copy các phần tử vào mảng bằng **CopyTo**, thêm bớt phần tử với **Add**, **Remove**, **Clear** |
| **IList<T>** | Giao diện này kế thừa ICollection<T> là một danh sách các phần tử truy cập được theo vị trí của nó. Nó có indexer, phương thức để chèn phần tử xóa phần tử **Insert RemoveAt**. |
| **IDictionary<TKey,TValue>** | Giao diện để triển khai loại dữ liệu lưu trữ theo cặp key, value |
| **IComparer<TKey,TValue>** | Giao diện để triển khai cho phép so sánh, sắp xếp Collection |
| **….** |  |

|  |  |
| --- | --- |
| **Một số phương thức của List** | |
| **IndexOf(obj)** | Tìm index của đối tượng trong list |
| **LastIndexOf(obj)** | Tìm index của phần tử cuối cùng có giá trị bằng obj trong List |
| **FindIndex()** | Tìm kiểm trả về index |
| **FindLastIndex()** | Tìm kiếm trả về index cuối |
| **Find(Predicate)** | Tìm kiểm phần tử trả về |
| **FindAll(Predicate)** | Tìm kiếm trả về danh sách phần tử |
| **FindLast ()** | Tìm kiếm trả về phần tử cuối cùng |
| **Add(), AddRange()** | Thêm phần tử |
| **Insert(index,object), InsertRange(index,object)** | Chèn phần tử |
| **RemoveAt(index), RemoveRange(index,count),Clear() , RemoveAll()** | Xóa phần tử |
| **Contains(obj)** | Kiểm tra có chứa phần tử obj |
| **Reverse()** | Đảo thứ tự danh sách |
| **ToArray()** | Copy các phần tử qua mảng |
| **…** |  |

**3. Networking**

**3.1 Migration**

- Migration là kỹ thuật trong việc tương tác với cơ sở dữ liệu, khi thay đổi CSDL ở code sẽ cập nhật lên CSDL, đảm bảo toàn vẹn dữ liệu và lịch sử thay đổi.

- Khi sử dụng EF làm việc với DB, có các cách tạo dựng database:

+ Database first (Có database trước sau đó viết code)

+ Code first (Viết code tạo database)

+ Model first (Tạo model, sau đó code, tạo database sau)

<https://www.dammio.com/2017/04/20/entity-framework-phan-5-tao-mo-hinh-truoc-model-first>

|  |  |
| --- | --- |
| **Lệnh** | **Ý nghĩa** |
| Add-migration[name] | Thêm mới migration với tên name |
| Update-database | Cập nhật migration cuối cùng |
| Update-database -migration/-m [name] | Cập nhật database theo tên migration |
| Remove-migration [migration-name] | Xóa migration theo tên |
| Drop-database | Xóa database |

**4. ASP.NET MVC**

**4.1 Model Binding (Ánh xạ dữ liệu) & Model Validation**

* Binding

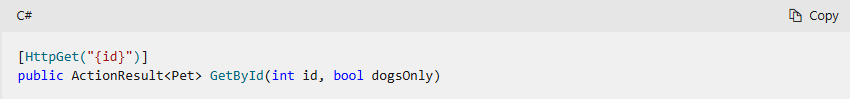
- Controller và Razor pages làm việc với dữ liệu bằng HTTP requests, ví dụ, route data gửi record key, posts form fileds gửi giá trị **properties** model

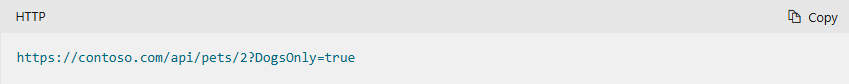
- Truy xuất dữ liệu (**Retrives data**) từ nhiều nguồn: route data, form flelds, and query strings.

- Chuyển dữ liệu từ controllers và Razor pages vào các thông số (**parameters**) và **public properies**

- Chuyển đổi chuỗi dữ liệu sang kiểu .NET

- Cập nhật thuộc tính (**properties**) của các kiểu phức tạp (**complex types**)



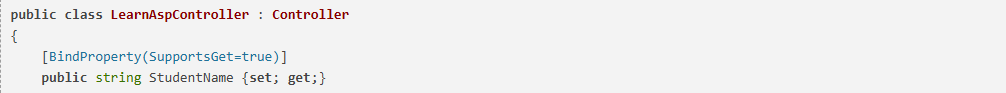


→ {“DogsOnly”= true; “id”=2 }

→ Khi hệ thống tìm từ nguồn đến, nếu không có sẽ thiết lập giá trị mặc định cho tham số của phương thức

→ Lỗi khi binding hoặc validation sẽ lưu trong **ControllerBase.ModelState** (dùng **ModelState.IsValid** để kiểm tra ánh xạ dữ liệu.

**[BindProperty] :** Để nguồn dữ liệu qua **HTTP requests** (Form, route data, url request,…)tự động convert dữ liệu và gán vào thuộc tính của Controller



**- [FormQuery] :** dữ liệu trích xuất từ Url query (/abc/?key=value)

**- [FormRoute] :** dữ liệu lấy từ giá trị trong tham số của Route

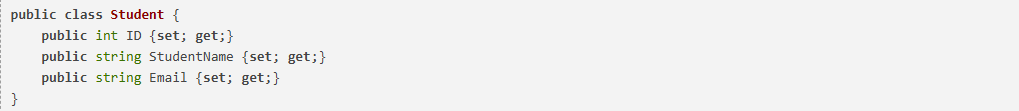
**- [FormForm] :** dữ liệu lấy từ Form

**- [FormBody]:** dữ liệu lấy từ Body của Http Request

**- [FormHeader] :** dữ liệu lấy từ Header của Http Request

**Name trong [BindingProperty], [FromQuery], [FromRoute] … :** khi binding dữ liệu mà tên ở nguồn đến và tên thuộc tính Controller khác nhau dùng thuộc tính Name







→ prefix.property\_name (tên key được tổ hợp từ tiền tố thuộc tính và tên thuộc tính).

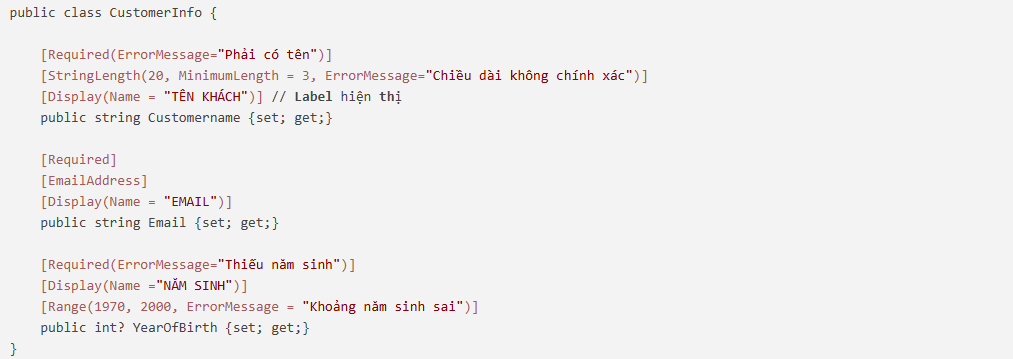
Khi các cơ chế binding,validation mặc định không đáp ứng nhu cầu nào đó có thể xây dựng 1 binding,validation riêng.

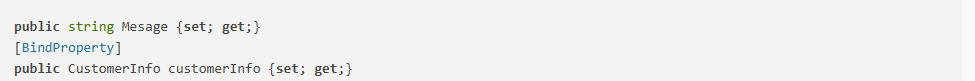
* **Validation**

Để đảm bảo dữ liệu chính xác, cần kiểm tra trước khi xử lý, .Net core hỗ trợ DataAnnotation Attribute với mô hình binding.

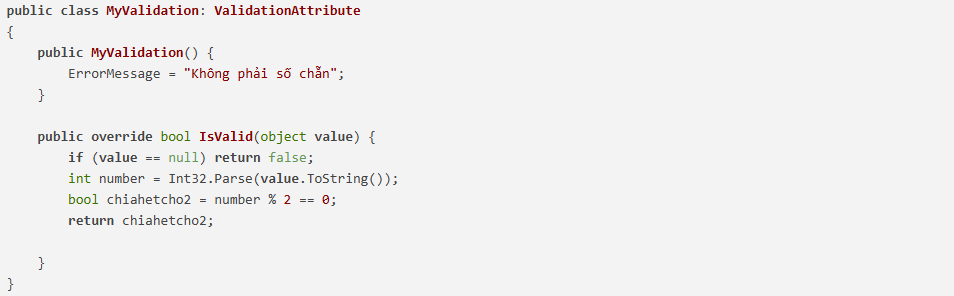
|  |  |
| --- | --- |
| **Attribute** | **Sử dụng** |
| Required | Thuộc tính cần phải có giá trị |
| StringLength | Chiều dài chuỗi (tối thiểu,tối đa) |
| Range | Số nằm trong khoảng |
| RegularExpression | Gía trị phù hợp với biểu thức chính quy |
| CustomValidation | Thuộc tính sử validation tự tạo |
| EmailAddress | Là địa chỉ Email |
| FileExtention | Phần mở rộng file |
| MaxLength | Dài tối đa của mảng, chuỗi |
| MinLength | Dài tối thiểu |
| Phone | Là số điện thoại |

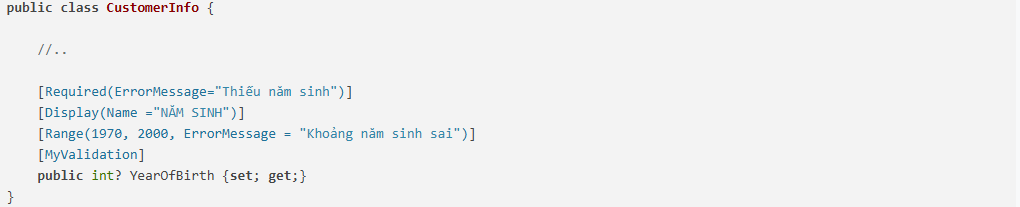
Ví dụ:



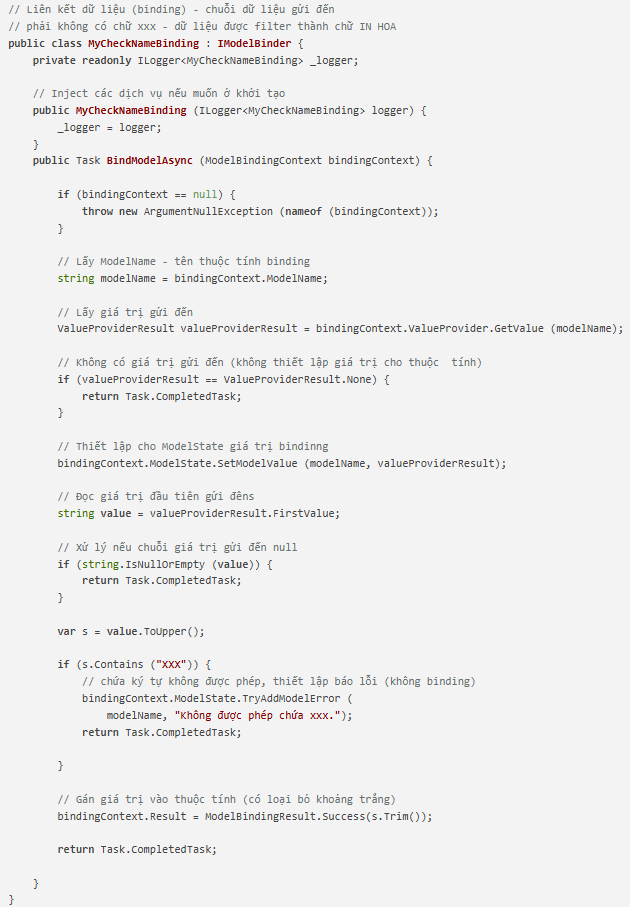


* **Dựng Validation**





* **Dựng Binding**



* **Syntax**



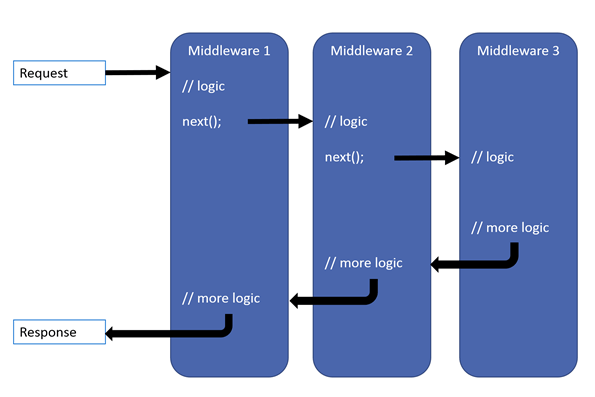


1. **ASP.NET CORE**

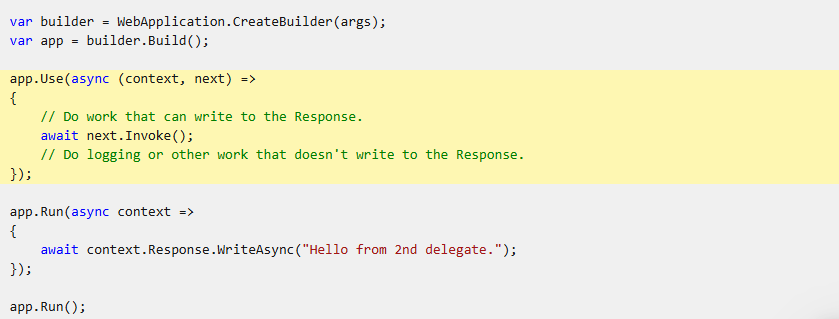
**5.1 Middleware**

* **Pipeline**

Trong ứng dụng .net core , các middleware kết nối với nhau thành xích, chuỗi các middleware theo thứ tự như hình gọi là pipeline.



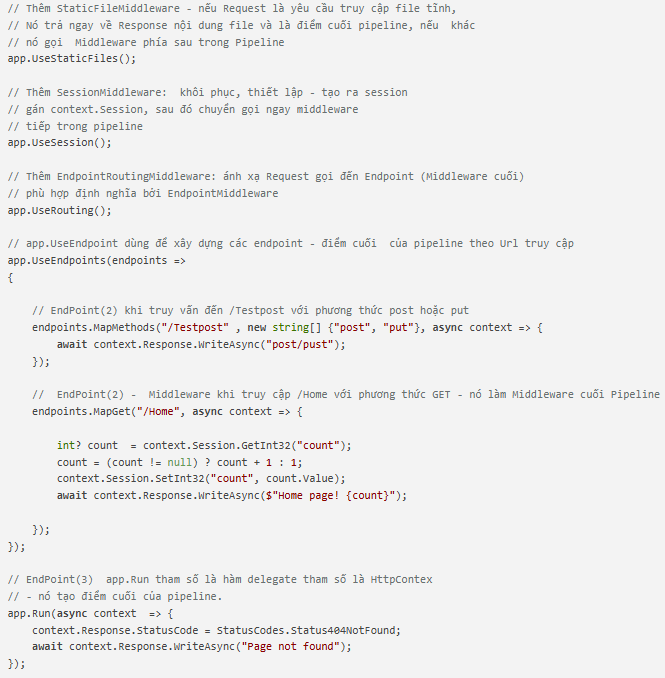
Các middleware là các dịch vụ nhỏ, đăng ký vào ứng dụng bằng cách sử dụng đối tượng IapplicationBuilder, sau đó ứng dụng sẽ tạo luồng xử lý (pipeline) cho các truy vấn gửi lên.



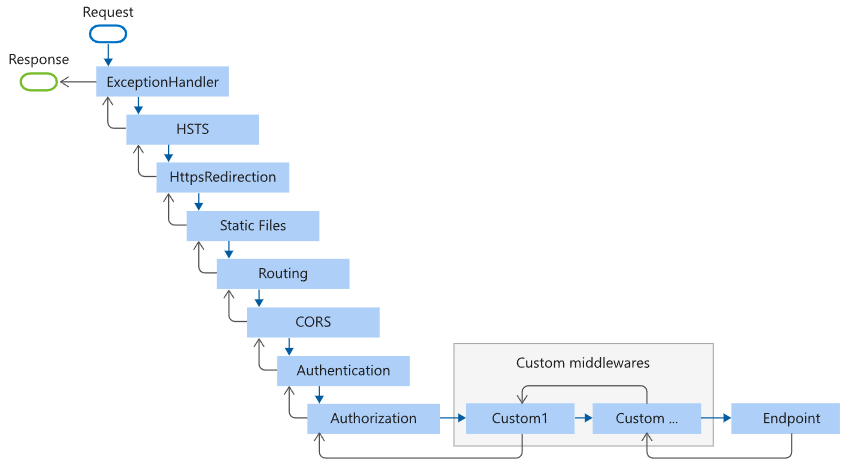
Các middleware trong pipeline xử lý các HttpMessage có dạng:

- Đầu vào là tham số kiểu HttpContext (chứa HttpRequest và HttpResponse), thi hành code và sau đó chuyển đến middleware tiếp theo.

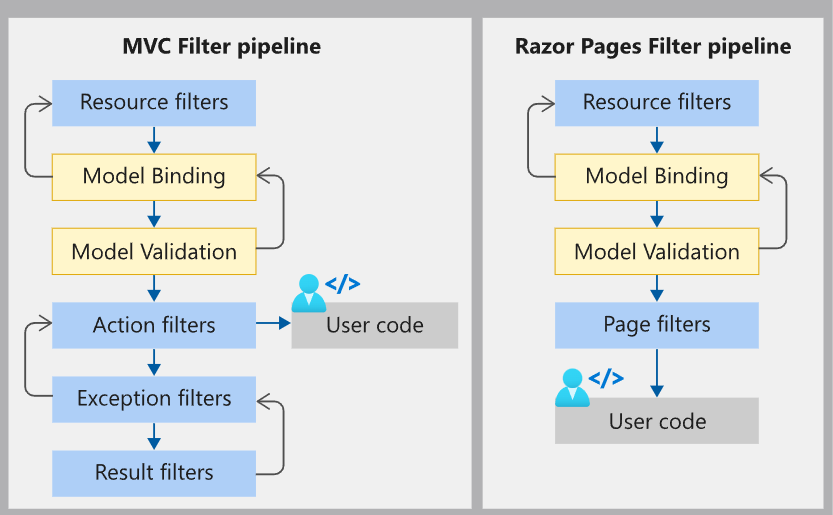
- app.Run() tham số (sử dụng phương thức overload) là hàm delegate tham số là HttpContext - **là điểm cuối của pipeline**



* **Thứ tự middleware**



* **MVC Endpoint (Endpoint Middleware)**



* **Custom middleware**

Cấu trúc middleware trong ASP.NET:

- Có một phương thức khởi tạo public với tham số thứ nhất kiểu **RequestDelegate,** các tham số tiếp theo phải inject được từ DI của hệ thống.

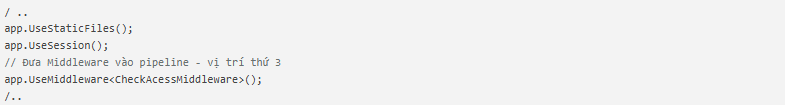
- Tối thiểu 1 phương thức Invoke, InvokeAsync với tham số là HttpContext, trả về Task (**InvokeAsync**).

- Trong invoke, InvokeAsync viết code xử lý middleware,nếu muốn chuyển đến middleware khác bằng cách gọi RequestDelegate đã truyền trong hàm tạo, nếu không cần chuyển middleware tiếp theo thì cần đảm bảo HttpResponse trong HttpContext phù hợp

Ví dụ 1:



→ Đưa middleware vào pipeline



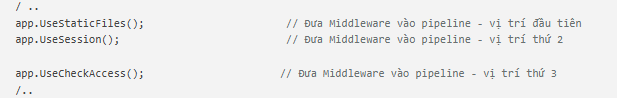
Ví dụ 2:



* **Thêm phương thức mở rộng**

Ví dụ 1:





Ví dụ 2:



Program.cs



#Xây dựng middleware triển khai từ giao diện IMiddleware



→ Để sử dụng middleware này, phải đăng ký vào DI container (Dependency Injection)



→ Sau đó thêm vào pipeline



* **Truyền dữ liệu giữa các Middleware**

Truyền dữ liệu giữa các middleware sử dụng **HttpContext.Items**

5.2 Security and Identity

* **Authentication**

1. **Authentication với JWT**

**Token** là gì **?**

**- Token** là một thông tin rất cần thiết hay còn gọi là công cụ để truy cập giao diện tài nguyên.

- **Token** bao gồm: uuid (danh tính duy nhất của người dùng), thời gian, ký hiệu (chữ ký, một vài chữ số đầu tiên của token được nén thành một chuỗi Hexadecimal - thập lục phân có độ dài nhất định bằng thuật toán **Hashing** - hàm băm).

**Tại sao sử dụng?**

**- Server không trạng thái - Stateless[[1]](#endnote-0)** và khả năng mở rộng tốt

- Hỗ trợ cho thiết bị di động rất tốt về get resourse

- Bảo mật

- Hỗ trợ các cuộc gọi chéo tên miền hay application

- Mỗi yêu cầu cần mang token và token được đặt trong header HTTP

- Xác thực người dựa trên token - stateless. Server không cần lưu dữ liệu của token, giảm áp lực cho server và giảm query liên tục và thường xuyên

- Token được quản lý hoàn toàn bởi ứng dụng, vì vậy có thể bỏ qua CORS

**Token được lưu ở đâu?**

- Khi client được return về token thì sẽ lưu ở Cookie và localStorage, ngoài ra còn có sessionStorage, indexDB,…

**Token được gửi đến client bằng cách nào?**

Cách 1: Đặt trên header mỗi request

Cách 2: Đặt token trong phần body data như phương thức POST

Cách 3: Chuyển qua URL

**Refresh Token?**

**Refresh token** cũng là một token, có nhiệm vụ duy nhất là để lấy token mới, nếu Access token cho user hết hạn. Refresh token được cấp cùng lúc với Access token thời gian có thể lâu hơn, Refresh token được lưu ở database phía server.

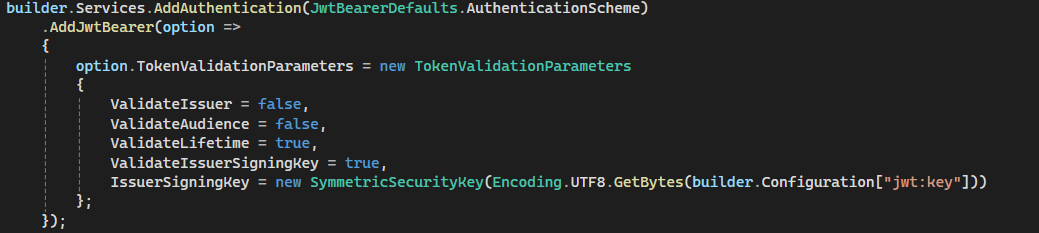
Cấu hình sử dụng token và refresh token,Ví dụ:

1. **Tạo Key Secret ở appsettings.json**



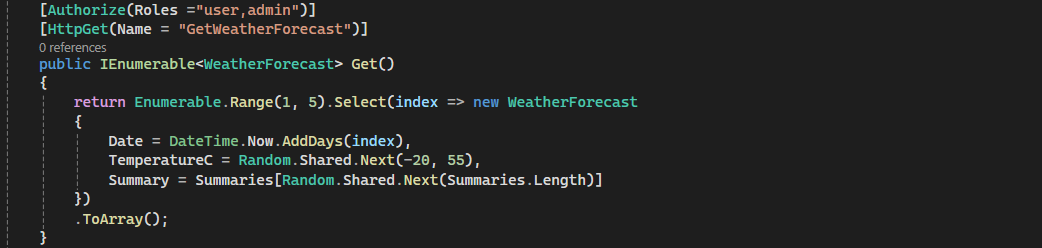
1. **Cấu hình ở Program.cs**

Cấu hình dịch vụ



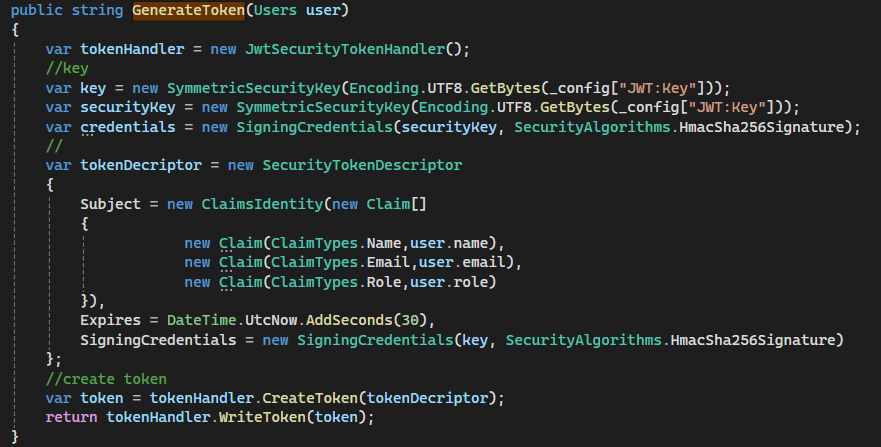
1. **Thêm thuộc tính Authorize**

Để sử dụng kiểm tra phần quyền

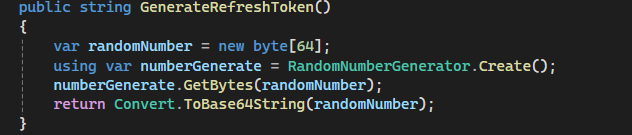


1. **Tạo và sử dụng token và refresh token**

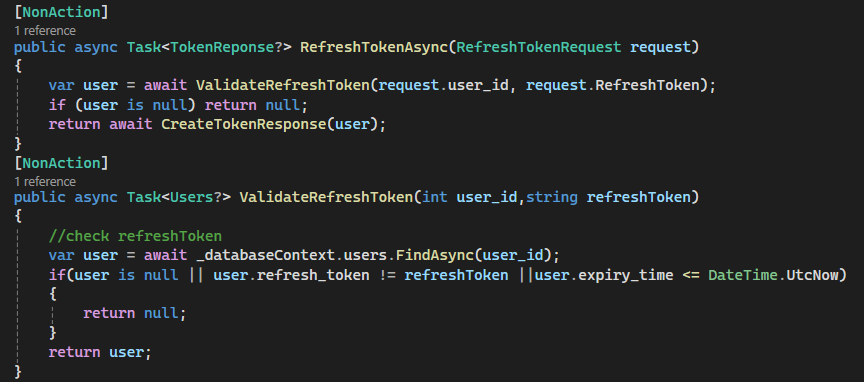
Tạo token

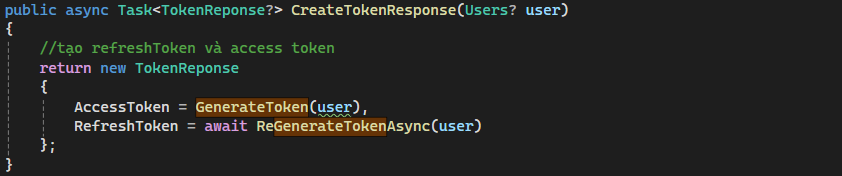


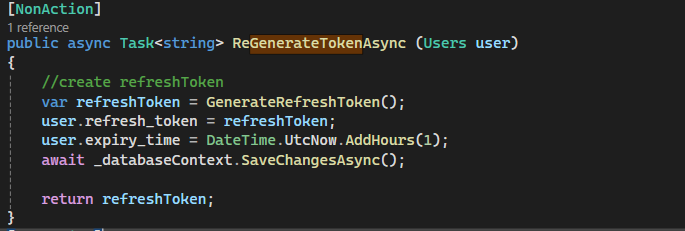
Tạo refresh token



Tạo mới token khi access token hết hạn







**5.3 Performance**

* **Caching**

Caching là một trong những công cụ quan trọng giúp tăng cường hiệu suất và giảm tải cho các ứng dụng web, cho phép lưu trữ dữ liệu tạm thời để sử dụng lại trong các yêu cầu tương tự. Việc sử dụng caching có thể giảm thiểu thời gian truy vấn cơ sở dữ liệu, tải trang nhanh hơn và cải thiện trải nghiệm người dùng.

[https://learn.microsoft.com/en-us/aspnet/core/performance/](https://learn.microsoft.com/en-us/aspnet/core/performance/caching/response?view=aspnetcore-9.0)caching/response?view=aspnetcore-9.0

1. **Response caching**

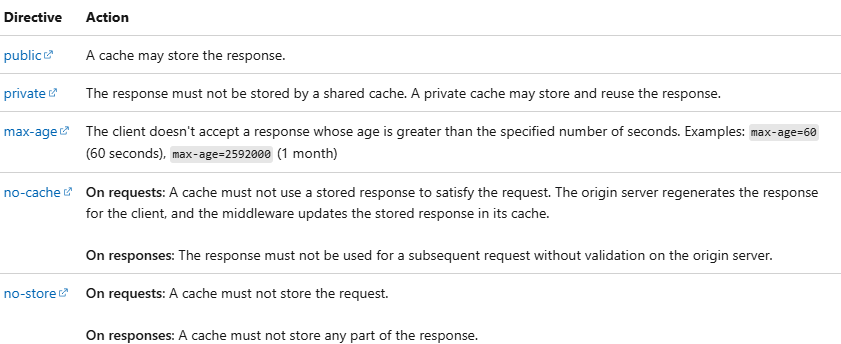
- Response caching giúp giảm số lần gửi request tử client hoặc proxy đến web server

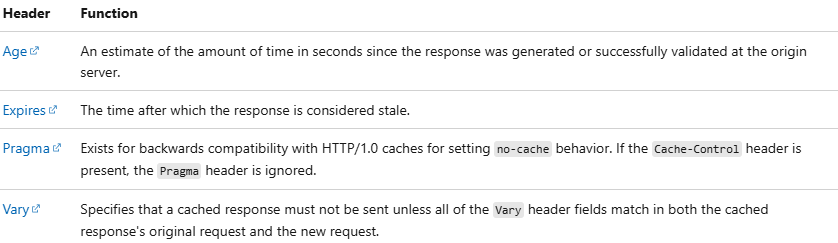
- Giảm số lần làm việc của web server thực hiện tạo response

- Response caching được đặt tại header.

- Các thuộc tính: **Vary**, **Nostore**, **Location**, **Duration,..**

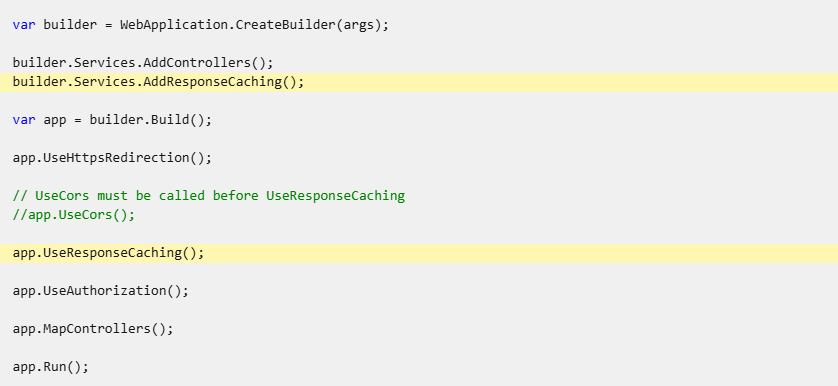
**Cache-Control[[2]](#endnote-1)**





Ví dụ:

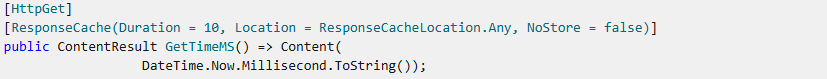
Cấu hình dịch vụ

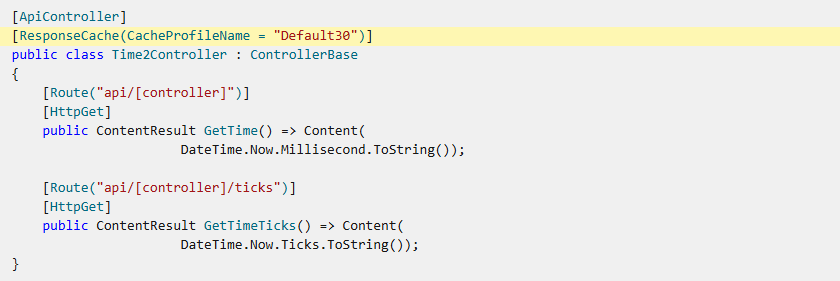


Cache profile



Sử dụng cache







1. **Memory caching**

Là kỹ thuật lưu dữ liệu tạm thời trong bộ nhớ Ram để tăng tốc độ truy xuất giảm tải cho cơ sở dữ liệu

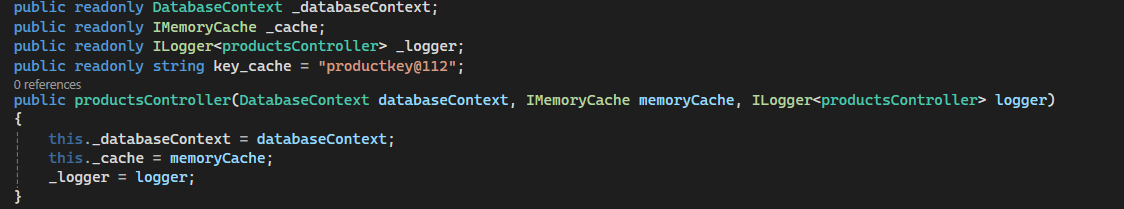
|  |  |
| --- | --- |
| **Các thuộc tính** | |
| **SetSlidingExpiration()** | Thời gian cache sau request cuối |
| **SetAbsoluteExpiration()** | Thời gian cache hết hạn sau khi tạo |
| **SetPriority(CacheItemPriority.[Normal])** | Độ ưu tiên của cache |
| **SetSize()** | Giới hạn cache |
| **RegisterPostEvictionCallback()** | Callback khi cache bị xóa |
| **Remove(key\_name)** | Xóa cache với tên là key\_name |
| **Compact()** | Giảm bộ nhớ cache |

Ví dụ:

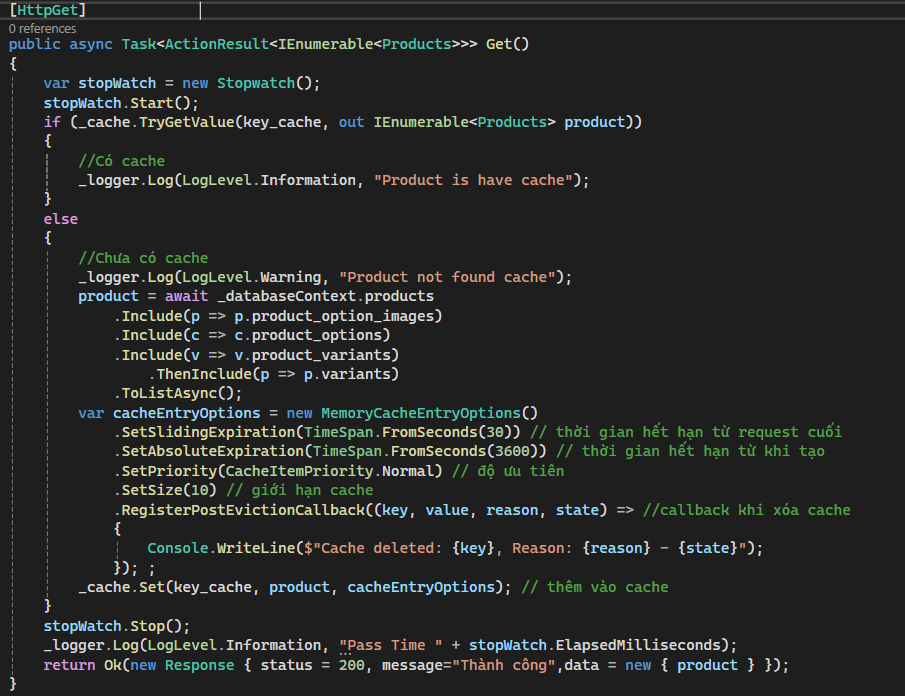
Cấu hình dịch vụ sử dụng memory cache và đặt giới hạn toàn bộ cache



Dùng DI khai báo dịch vụ dùng memory cache và tạo contructor



Sử dụng Cache



Output



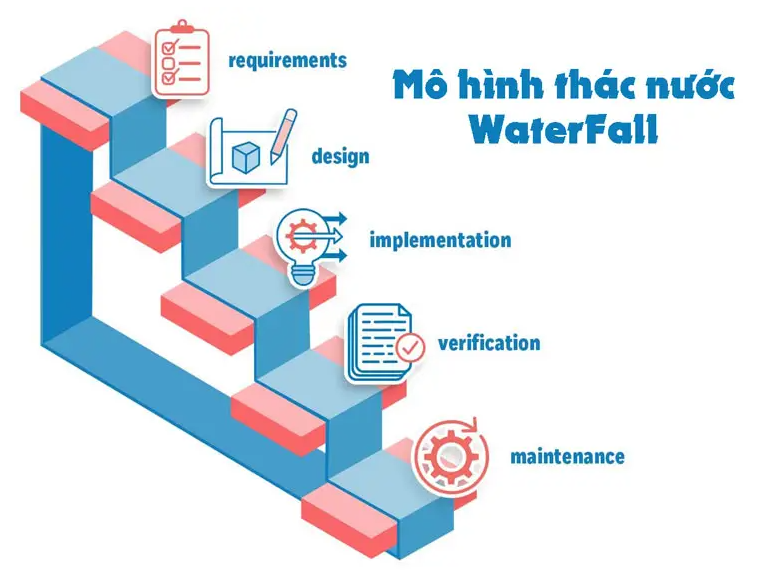


**II Mô hình phát triển phần mềm**

<https://testerpro.vn/cac-mo-hinh-phat-trien-phan-mem/>

Mô hình phát triển phần mềm hay quy trình phát triển phần mềm là tập hợp các kỹ thuật và hệ thống tổ chức để tạo ra phần mềm máy tính. Mục tiêu xây dựng các chức năng chương trình một cách hiệu quả nhất

1. **Waterfall (Mô hình thác nước)**



Là mô hình, trong đó các giai đoạn phát triển phần mềm được sắp xếp một cách chiến lược, thể hiện sự tương tác giữa khách hàng với phần mềm và môi trường mà nó thực hiện.

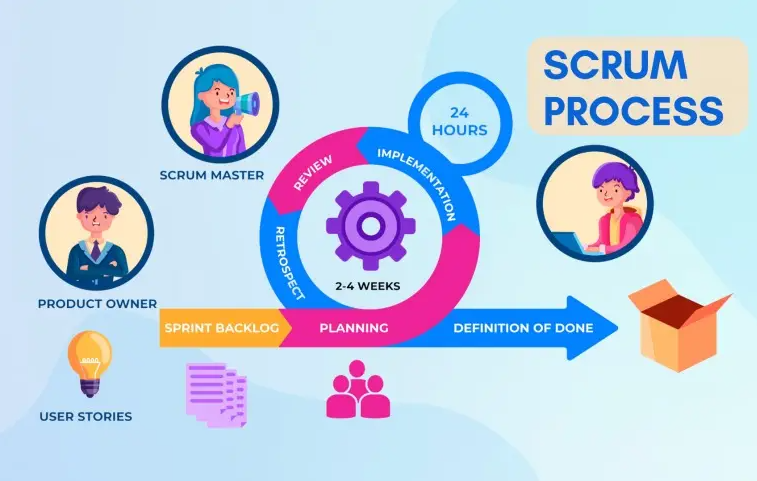
**Các giai đoạn của mô hình thác nước:**

* **Analysis**: Lên kế hoạch, phân tích và đặc tả yêu cầu
* **Design**: Thiết kế và đặc tả hệ thống
* **Implementation**: Lập trình và kiểm thử đơn vị
* **Verification**:kiểm thử hệ thống và tích hợp
* **Deployment**: Triển khai hệ thống
* **Maintance**: Giao hàng, bảo trì, cải tiến

**Trường hợp sử dụng:**

* Khi có một ý tưởng rõ ràng về những gì bạn muốn kết quả cuối cùng như thế nào
* Khi khách hàng không thể thay đổi phạm vi của một dự án khi nó đã bắt đầu
* Khi nói đến thành công, khái niệm và định nghĩa là rất quan trọng
* Khi không còn nghi ngờ về những gì phải làm

1. **Mô hình Scum**



Các hoạt động của dự án được hoàn thành trong các chu kỳ ngắn - chạy nước rút, tương đối dễ quản lý và theo dõi tiến độ

**Các giai đoạn của mô hình Scrum**

* **Product Backlog**: Khi các nhiệm vụ ưu tiên được xác định kỹ lưỡng về dự án sẽ được tạo ra được thu thập.
* **Sprint**: là nhịp tim của quy trình scrum, khung thời gian một tháng trong đó diễn ra việc tạo ra một sản phẩm có thể giao hàng được.
* **Burn Down**: Là giai đoạn đo lường tiến độ của một dự án scrum. Khi mỗi lần chạy nước rút hoàn thành, scrum master sẽ chịu trách nhiệm cập nhật hình ảnh.

**Khi nào sử dụng mô hình Scrum**

* Cách tiếp cận này được sử dụng trong các tình huống cần có kết quả ngay lập tức.
* Trong những trường hợp khi có nhiều sự mơ hồ và các nhiệm vụ không được xác định rõ ràng.
* Khi khách hàng yêu cầu phương pháp phát triển tùy biến cao cho một sản phẩm nhất định
* Ghi chú

1. **Stateless**: chính là không có bộ nhớ để xử lý giao dịch và phía server không lưu bất kỳ thông tin phiên nào mỗi khi client request và server response. [↑](#endnote-ref-0)
2. **Cache-Control:** [↑](#endnote-ref-1)