**C# and Backend**

Table of Contents

[**1.** **LINQ** 2](#_Toc192621136)

[**2.** **SORT mongodb driver** 2](#_Toc192621137)

[**3.** **Sort có thể sort 1 hay nhiều fields** 2](#_Toc192621138)

[**4.** **Gộp nhiều câu filter với BsonDocument** 2](#_Toc192621139)

[**5.** **Webhook vs. API** 3](#_Toc192621140)

[**6.** **MVC pattern** 3](#_Toc192621141)

[**7.** **N-Layer Architecture** 3](#_Toc192621142)

[**8.** **Data Transfer Object (DTO)** 3](#_Toc192621143)

[**9.** **So sánh MVC, N-Tier và Three-Layer** 3](#_Toc192621144)

[**10.** **API là gì?** 4](#_Toc192621145)

[**11.** **OOP la gì?** 4](#_Toc192621146)

[**12.** **SOLID** 5](#_Toc192621147)

1. **LINQ**

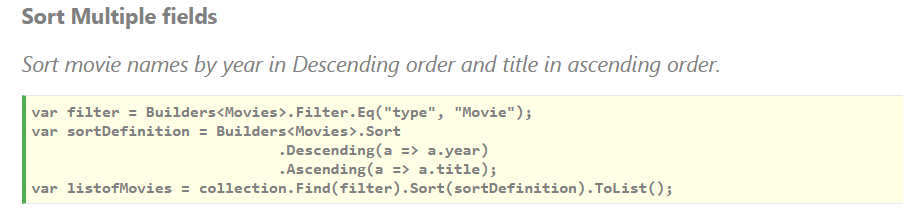
 var listOfNames = new List<string>()  
{  
    "John Doe",  
    "Jane Doe",  
    "Jenna Doe",  
    "Joe Doe"  
};

- The Query syntax (giống kiểu SQL)  
var qNames = from name in listOfNames where name.Length <= 8 select name;

- the Method syntax ( sử dụng lamda, giống arrow func và giống C# )

var mNames = listOfNames.Where(name => name.Length <= 8);

1. **SORT mongodb driver**



1. **Sort có thể sort 1 hay nhiều fields**

Filter 1 document có chứa 1 field là array

Có thể dùng Filter.ElemMatch

Filter.ElemMatch(param1, param2)

Filter.ElemMatch(\_ => \_.HandleableDepartments, Builders<MGTFExDepartmentHandlingModel>.Filter.In(h => h.Status, status));

Param1: **\_ => \_.HandleableDepartments** là field có chứa array

Param2: **Builders<MGTFExDepartmentHandlingModel>.Filter.In(h => h.Status, status)** là 1 filter, filter này sẽ cần 1 field trong array để so sánh

1. **Gộp nhiều câu filter với BsonDocument**



1. **Webhook vs. API**

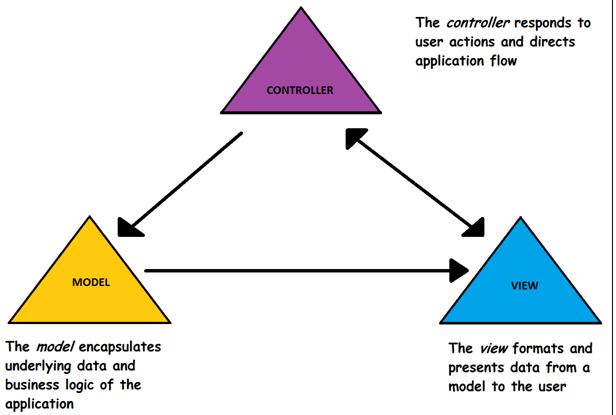
|  |  |
| --- | --- |
| **Webhooks** | **API** |
| POST to the webhook with some JSON data, and then it'll do something internally.  They are **API requests**, but they **help the server** instead of the client  Webhooks **take in data** and do internal work | API would be the team whose job it is to respond to inquiries from external parties  Clients make requests to an API from the client  API endpoints **return data** so that the client asking for that data can do something with it |

1. **MVC pattern**

MVC (Model-View-Controller) It separates an application into **three interconnected components**

* **M-Model**: manage the data, impliment the databasee such as store, get, update, delete data
* **V-View**: render, present the user interface
* **C-Controller**: handle actions, authorize the user, routing the View corresponding with data
* **Flow of Control in MVC**:

The user click a button => the controller handle that action and command the Model process the data => the view will update the UI from Model

* **Separation of Concerns**: seperate the logic, UI, data to make them easy to maintain and test
* **Triangular flow**: 

1. **N-Layer Architecture**

Presentation Layer (controller layer): handle request, intercept authorize user, routing to the correspond Business

Business Logic Layer (Business/ Service Layer): impliment data, modify data, call another services

Data Access Layer (Entity/Repository Layer): retrive, saved, update Data.

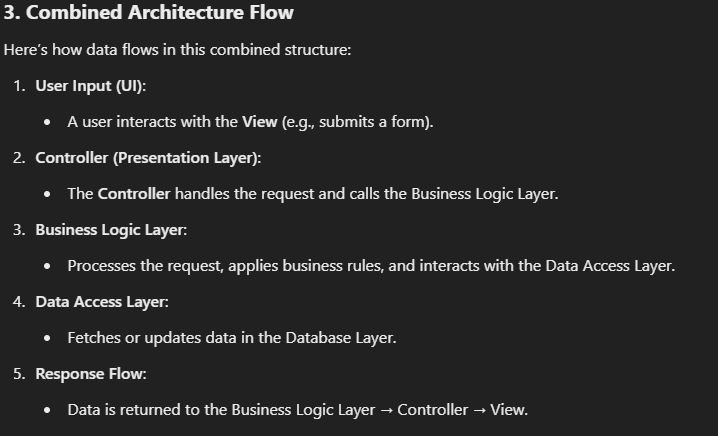
1. **Data Transfer Object (DTO)**

* Dùng để giấu cấu trúc của data, encapsulate data
* lọc ra các field cần thiết để trả về cho client,
* validate field data

1. **So sánh MVC, N-Tier và Three-Layer**

Một điểm quan trọng của layered architecture là các layer này phải **xếp chồng lên nhau**. Nguyên tắc là layer trên gọi được xuống (hoặc phụ thuộc vào) các layer dưới mà không phải ngược lại. Ví dụ controller có thể gọi service, service gọi cho repository, nhưng repository không thể gọi cho 2 thằng kia được. Và cũng không bắt buộc phải là 3 layer, có thể thêm các layer khác tùy nhu cầu.

Còn MVC chỉ là **architectural pattern**, tuy cũng chia 3 phần nhưng **không có tính xếp lớp** như layered architecture, mà thiên về kiểu tách chức năng hơn (separation of concerns). Do đó các **pattern này có thể dùng chung với layered architecture** được. Trong các app winform hay web dạng SSR, thì phần model trong MVC vẫn có thể access đến các layer khác như service hoặc repository chẳng hạn.



1. **API là gì?**

* Stand for Application Programming Interface
* Dùng để giao tiếp giữa ứng dụng và những nguồn khác ( hệ thống, người dùng, ứng dụng khác)
* Dùng để định nghĩa các thành phần có thể giao tiếp với ứng dụng
* Types of APIs. API ko chỉ có HTTP endpoints mà còn có các loại khác vd:
  + Web API: là những http endpoint
  + Library API: là các API mà thư việc cung cấp, Reactjs cho phép sử dụng các hook UseState, useEffect
  + Command-line API: giúp cho phép tương tác với API thông qua các dòng lệnh, vd: git command

1. **OOP la gì?**

* Stand for Object Oriented Programming
* Là Mô hình lập trình hướng đối tượng. Các đối tượng chứa 2 phần chính là Properties(thuộc tính) và Methods(phương thức)
* Trong OOP sẽ có Class và Object.
* Class là 1 khung mẫu (template )hoặc 1 cấu trúc (structure) để xây dựng đối tượng.
* Properties: định nghĩa các thông tin, đặc điểm, đặc tính
* Methods: các hành vi, các hành động của đối tượng đó
* Object là 1 instance của Class. Và có đầy đủ Properties và Methods- phương thức của Class đó khi khởi tạo

Có 4 tính chất trong OOP

**Tính đóng gói (encapsulation)**

* Để bảo vệ và giữ bí mật các thành phần bên trong của Object.
* Tránh thay đổi trực tiếp các thuộc tính trong Obj
* Mà phải thông qua các phương thức public của Object.
* Như vậy để object có thể kiểm soát, validate dữ liệu trước khi thay đổi thuộc tính trên trong

**Tính Trừu Tượng (Abstraction)**

* Đơn giản hóa đi các vận hành, cấu trúc của đối tượng
* Chỉ đưa ra, show ra những chức năng cần thiết cho bên ngoài

**Tính Kế Thừa (Inheritance)**

* Cho phép 1 class con kế thừa, thừa hưởng các thuộc tính và phương thức của class cha
* Ko cần định nghĩa lại các thuộc tính và phương thức và giảm sự lặp lại, có tính tái sử dụng

**Tính Đa Hình (Polymorphism)**

* Có thể ghi đè override lại phương thức của lớp cha, để cho phép có hành vi khác với lớp cha

1. **SOLID**

* **Single Responsibility Principle: Nguyên tắc chỉ nhận 1 trách nhiệm**
  + 1 class chỉ nên có 1 trách nhiệm để thay đổi và đảm nhận
  + Vd: class lưu thông tin thì chỉ nên có phương thức save, ko nên chứa mấy chức khác như update, delete
* **Open/Closed Principle: Nguyên tắc mở rộng, và ko được sửa đổi class cũ**
  + Lúc đầu có 1 class chỉ có 1 phương thức thanh toán, trong phương thức thanh toán chỉ có 1 loại thanh toán. Sau đó ta lại thêm 1 loại thanh toán mới trong class cũ thì vi phạm
  + Nên viết class mới mở rộng từ class cũ
* **Liskov Subtitution Principle: các lớp con có thể thay thế lớp cha mà ko thay đổi tính đúng đắn lớp con**
  + Ta có class Person, person sẽ có các đặc tính như 1 con người( ăn, uống, đi, nói). Nếu ta tạo ra 1 class con là 1 robot (robot ko biết ăn, ko biết uống ) thì bị vi phạm
* **Interface segregation Principle: dùng nhiều interface để phục vụ các mục đích nhỏ**
  + Nếu có 1 interface quá dài thì 1 class không thực thi hết các methods trong đó, dẫn đến sự dư thừa
* **Dependancy Inversion Principle: module cấp cao ko phụ thuộc vào module cấp thấp, mà giao tiếp thông qua abstraction(interface). Các class giao tiếp với nhau thông qua interface ko phải implementation**
  + Tôi chọn mua 1 cái màn hình trước, rôi sẽ mua dây kết nối sau.
  + Chứ ko mua dây kết nối trước rồi mới chọn màn hình

ORM