Tài Liệu Tìm Hiểu Về Môn Nhập Môn Công Nghệ Phần Mềm

**1. Mô Hình MVC**

**MVC** là từ viết tắt bởi 3 từ **Model – View – Controller**. Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm. Mô hình source code thành 3 phần, tương ứng mỗi từ. Mỗi từ tương ứng với một hoạt động tách biệt trong một mô hình.

|  |
| --- |
|  |

Các thành phần trong mô hình.

**Model (M):**

Là bộ phận có chức năng lưu trữ toàn bộ dữ liệu của ứng dụng. Bộ phận này là một cầu nối giữa 2 thành phần bên dưới là View và Controller. Model thể hiện dưới hình thức là một cơ sở dữ liệu hoặc có khi chỉ đơn giản là một [file XML](https://monamedia.co/file-xml-la-gi/) bình thường. Model thể hiện rõ các thao tác với cơ sở dữ liệu như cho phép xem, truy xuất, xử lý dữ liệu,…

**View (V):**

Đây là phần giao diện (theme) dành cho người sử dụng. Nơi mà người dùng có thể lấy được thông tin dữ liệu của MVC thông qua các thao tác truy vấn như tìm kiếm hoặc sử dụng thông qua các website.

Thông thường, các ứng dụng web sử dụng MVC View như một phần của  hệ thống, nơi các thành phần [HTML](https://monamedia.co/html-va-css-la-gi/) được tạo ra. Bên cạnh đó, View cũng có chức năng ghi nhận hoạt động của  người dùng để tương tác với Controller. Tuy nhiên, View không có mối quan hệ trực tiếp với Controller, cũng không được lấy dữ liệu từ Controller mà chỉ hiển thị yêu cầu chuyển cho Controller mà thôi.

Ví dụ: Nút “delete” được tạo bởi View khi người dùng nhấn vào nút đó sẽ có một hành động trong Controller.

|  |
| --- |
|  |

**Controller (C):**

Bộ phận có nhiệm vụ xử lý các yêu cầu người dùng đưa đến thông qua view. Từ đó, C đưa ra dữ liệu phù hợp với người dùng. Bên cạnh đó, Controller còn có chức năng kết nối với model.

Một ví dụ cụ thể, khi bạn đăng một bài lên website:

* View hiển thị ra những phần để nhập tiêu đề bài viết và nội dung bài viết.
* Controller lấy dữ liệu từ 2 trường vừa mới nhập vào và gửi đến Model.
* Model nhận dữ liệu từ Controller để lưu trữ trong cơ sở dữ liệu.

Mỗi bộ phận thực hiện chức năng nhất định, nhưng chúng có sự thống nhất, liên kết với nhau tạo nên **mô hình MVC**. Mô hình này tương đối nhẹ. Nó có thể tích hợp được nhiều tính năng có trong ASP.NET hiện giờ. Ví dụ như authentication (quá trình xác thực).

**Luồng đi trong mô hình MVC như thế nào.**

Bạn có thể hình dung, khi  một yêu cầu từ máy client gửi tới server, Controller sẽ thực hiện nhiệm vụ của mình đó là tiếp nhận và xử lý yêu cầu. Trong trường hợp cần thiết, nó có thể liên hệ Model – bộ phận làm việc với database để hỗ trợ.

Khi xử lý xong yêu cầu, kết quả sẽ được trả về View. Tại View sẽ tạo mã HTML thành giao diện và trả về hiển thị trên trình duyệt.

**Ưu và nhược điểm của mô hình MVC**

Nhẹ, tiết kiệm băng thông: MVC không sử dụng viewstate nên khá tiết kiệm diện tích băng thông. Khi sử dụng, người dùng có thể sử dụng ứng dụng trên web cần tương tác gửi và nhận dữ liệu một cách liên tục. Do đó, việc giảm băng thông giúp cho website hoạt động tốt và ổn định hơn.

|  |
| --- |
|  |

Kiểm tra dễ dàng: Với **MVC**, bạn có thể dễ dàng kiểm tra, rà soát lỗi phần mềm trước khi tới tay người tiêu dùng, đảm bảo chất lượng và độ uy tín cao hơn.

Chức năng control: [HTML](https://wiki.tino.org/html-css-javascript-la-gi/), CSS và JavaScript tạo nên một nền tảng vững chắc. MVC sẽ giúp bạn control ưu việt hơn trên nền tảng các ngôn ngữ mới với nhiều hình thức khác nhau.

View và size: View sẽ là nơi lưu trữ các dữ liệu. Càng nhiều yêu cầu được thực hiện thì kích thước càng tệp càng lớn. Khi  đó, đường truyền mạng cũng giảm tốc độ load. Việc sử dụng mô hình MVC sẽ giúp bạn tiết kiệm được diện tích băng thông một cách tối ưu

Chức năng Soc (Separation of Concern): Chức năng này cho phép bạn phân tách rõ ràng các phần như Model, giao diện, data, nghiệp vụ.

Tính kết hợp: Việc tích hợp ở **mô hình MVC** cho phép bạn thoải mái viết code trên nền tảng website. Khi đó, server của bạn sẽ được giảm tải khá nhiều.

Đơn giản: Đây là một mô hình với kết cấu tương đối đơn giản. Dù bạn không có quá nhiều chuyên môn cũng có thể sử dụng được.

**2.Các quan hệ association, aggregation, composition**

Sự khác nhau giữa Class ( lớp ) vs Object ( đối tượng ) với Instance ( thể hiện ): Class ( lớp ) như bản thiết kết của một Object ( đối tượng ). Instance ( thể hiện ) là một đơn vị độc lập của class

Ví dụ: lớp HocSinh sẽ có thuộc tính maHocSinh, tenHocSinh là bản thiết kế của đối tượng học sinh. **hocSinh1**, **hocSinh2** là 2 thể hiện của lớp học Sinh.

**Association**

Sự liên kết giữa 2 lớp mà không ai sở hữu ai.

|  |
| --- |
|  |

Vòng đời các thể hiện của 2 lớp thì độc lập nhau và không có mối quan hệ sở hữu nào ở đây cả.

Ví dụ: Một học sinh có thể liên kết ( associate ) với nhiều giáo viên. Học sinh **khoa**có 2 giáo viên là **teacher1**, **teacher2**

Những thể hiện (Instance) của class teacher được tạo ra ngoài class student. Nêm khi thằng Khoa bị huỷ thì 2 ông giáo viên **teacher1, teacher2**, cũng không bị ảnh hưởng gì.

Các học sinh khác cũng thế nên ta nói là **Student** không có quan hệ sở hữu (Ownship) với **Teacher** hết.

Khi vẽ quan hệ **Association**, ta dùng ký hiệu như sau.

|  |
| --- |
|  |

**Aggregation**

Aggregation cũng giống như Association, nhưng khác là Aggregation có mối quan hệ sở hữu ( ownership ) giữa các instance

Vậy **Aggregation** khác với **Asociation** như thế nào.

Có tính sở hữu (Ownership) giữa 2 class. **Part-whole relationship.** Vòng đời của **part**  không phụ thuộc vào **whole**. Ví dụ điện thoại(cellphone) và pin(Pin) có mối liên hệ là **Aggregation**. Điện thoại cần có một cục pin để hoạt động. Khi điện thoại hư có thể đem cục pin này sang điện thoại khác hoạt động.

[UML](https://tapchithethao.com/uml-la-gi/) dùng mũi tên sau để vẽ:

|  |
| --- |
|  |

**Composition**

**Composition** tương tự như **Aggregation** nhưng khác là vòng đời của thằng **Part** sẽ bị phụ thuộc vào thằng **Whole.**

Ví dụ: Khi một Hotel (Whole) bi huỷ thì Rom1, Rom2, cũng bi huỷ theo. Nói cách khác là vòng đời của thàng **Part** bị phụ thuộc vào vòng đời của **whole trong quan hệ Part-whole ralationship.**

Ký hiệu:

|  |
| --- |
|  |

**Summary**

|  |
| --- |
|  |

**Dependency**

Khi một class A phụ thuộc và một class B, những thay đổi ở B có thể ảnh hưởng đến A.

Khi lưu Order ta cần phải biết CustomerId.

Nếu thay đôi **id** ở Customer từ kiểu Int sang String thì Order cũng phải thay đổi theo. Ta gọi mối quan hệ này là **Dependency**