ÔN TẬP XÂY DỰNG PHẦN MỀM THEO MÔ HÌNH PHÂN LỚP

1. Trình bày ý nghĩa của 2 loại mô hình : sơ đồ lớp khái niệm và sơ đồ lớp thiết kế.
   - Nêu sự liên hệ giữa 2 mô hình này.
2. Trình bày ý nghĩa của 2 loại sơ đồ : sơ đồ hoạt động (activity diagram) và sơ đồ tuần tự (sequence diagram).

- Activity Diagram

- Biểu đồ hoạt động nắm bắt hành động và các kết quả của chúng.

- Biểu đồ hoạt động tập trung vào công việc được thực hiện trong khi thực thi một thủ tục (hàm), các hoạt động trong một lần thực thi một trường hợp sử dụng hoặc trong một đối tượng.

- Biểu đồ hoạt động là một biến thể của biểu đồ trạng thái và có một mục tiêu tương đối khác, đó là nắm bắt hành động (công việc và những hoạt động phải được thực hiện) cũng như kết quả của chúng theo sự biến đổi trạng thái. Các trạng thái trong biểu đồ hoạt động (được gọi là các trạng thái hành động) sẽ chuyển sang giai đoạn kế tiếp khi hành động trong trạng thái này đã được thực hiện xong (mà không xác định bất kỳ một sự kiện nào theo như nội dung của biểu đồ trạng thái).

- Một sự điểm phân biệt khác giữa biểu đồ hoạt động và biểu đồ trạng thái là các hành động của nó được định vị trong các *luồng* (*swimlane*). Một luồng sẽ gom nhóm các hoạt động, chú ý tới khái niệm người chịu trách nhiệm cho chúng hoặc chúng nằm ở đâu trong một tổ chức. Một biểu đồ hoạt động là một phương pháp bổ sung cho việc miêu tả tương tác, đi kèm với trách nhiệm thể hiện rõ các hành động xảy ra như thế nào, chúng làm gì (thay đổi trạng thái đối tượng), chúng xảy ra khi nào (chuỗi hành động), và chúng xảy ra ở đâu (luồng hành động).

  - *Biểu đồ hoạt động có thể được sử dụng cho nhiều mục đích khác nhau, ví dụ như:*

  - Để nắm bắt công việc (hành động) sẽ phải được thực thi khi một thủ tục được thực hiện. Đây là tác dụng thường gặp nhất và quan trọng nhất của biểu đồ hoạt động.

  - Để nắm bắt công việc nội bộ trong một đối tượng.

  - Để chỉ ra một nhóm hành động liên quan có thể được thực thi ra sao, và chúng sẽ ảnh hưởng đến những đối tượng nằm xung quanh chúng như thế nào.

  - Để chỉ ra một trường hợp sử dụng có thể được thực thể hóa như thế nào, theo khái niệm hành động và các sự biến đổi trạng thái của đối tượng.

- Để chỉ ra một doanh nghiệp hoạt động như thế nào theo các khái niệm công nhân (tác nhân), qui trình nghiệp vụ (workflow), hoặc tổ chức và đối tượng (các khía cạnh vật lý cũng như tri thức được sử dụng trong doanh nghiệp).

- Biểu đồ hoạt động có thể được coi là một loại Flow chart. Điểm khác biệt là Flow Chart bình thường ra chỉ được áp dụng đối với các qui trình tuần tự, biểu đồ hoạt động có thể xử lý cả các các qui trình song song.

Sequence Diagram

- Biểu đồ trình tự được sử dụng chủ yếu để thể hiện mối tương tác giữa các đối tượng trong thứ tự trình tự mà các mối tương quan này xảy ra. Giống như biểu đồ lớp, các chuyên viên phát triển thường nghĩ các biểu đồ trình tự là dành riêng đối với họ. Tuy nhiên, nhân viên kinh doanh của một tổ chức có thể tìm ra biểu đồ trình tự hiệu quả để trao đổi các công việc gần đây hoạt động thế nào bằng cách trình bày các đối tượng công việc đa dạng tác động với nhau thế nào. Bên cạnh tập hợp tài liệu các sự kiện của một tổ chức, một biểu đồ trình tự ở cấp độ kinh doanh có thể được sử dụng như là một tài liệu cần thiết để trao đổi các yêu cầu cho việc thực thi hệ thống trong tương lai. Trong giai đoạn các yêu cầu của một dự án, các chuyên viên phân tích có thể tận dụng các trường hợp tới mức độ cao hơn bằng cách đưa ra một mức cải tiến chính thức hơn. Khi xảy ra như vậy, sử dụng các tình huống được cải tiến vào một hoặc nhiều các biểu đồ trình tự.
- Nhân viên kỹ thuật của một tổ chức có thể tìm các biểu đồ trình tự hiệu quả qua việc chứng minh một hệ thống tương lai sẽ hoạt động thế nào. Trong giai đoạn thiết kế, các chuyên viên kiến trúc và chuyên viên phát triển có thể sử dụng biểu đồ để ép các mối tương quan đối tượng của hệ thống, do đó lọc ra các thiết kế hệ thống tổng thể.

- Một trong các sử dụng chính của biểu đồ trình tự là trong sự chuyển đổi từ các yêu cầu được thể hiện như là sử dụng các tình huống tới các cải tiến tiếp theo và mức chính thức hơn của sự cải tiến. Sử dụng các trường hợp được cải tiến thường xuyên hơn vào một hoặc nhiều biểu đồ trình tự. Ngoài các sử dụng trong việc thiết kế hệ thống mới, biểu đồ trình tự có thể được sử dụng để dẫn chứng các đối tượng trong một hệ thống đang tồn tại (gọi là "hợp lệ") gần đây tương tác thế nào. Sự dẫn chứng này rất hữu ích khi chuyển đổi một hệ thống tới nhân sự hoặc tổ chức khác.

3. Trình bày ý nghĩa và chức năng của mỗi lớp trong kiến trúc 3 lớp. Cho ví dụ minh họa.

- **GUI layer**: Lớp này làm nhiệm vụ giao tiếp với người dùng cuối để thu thập dữ liệu và hiển thị kết quả/dữ liệu thông qua các thành phần trong giao diện người sử dụng.

- **Business Logic Layer**: đây là layer xử lý chính các dữ liệu trước khi được đưa lên hiển thị trên màn hình hoặc xử lý các dữ liệu trước khi chuyển xuống Data Access Layer để lưu dữ liệu xuống cơ sở dữ liệu. Đây là nơi để kiểm tra ràng buộc các yếu tố nghiệp vụ, tính toán, xử lý các yêu cầu và lựa chọn kết quả trả về cho GUI Layers.
- **Data Access Layer:** Lớp này thực hiện các nghiệp vụ liên quan đến lưu trữ và truy xuất dữ liệu của ứng dụng như đọc, lưu, cập nhật cơ sở dữ liệu,..
    - **Ví dụ: Mô tả hoạt động của mô hình 3-layer với ứng dụng liệt kê danh sách điểm sinh viên:**

- Từ màn hình form quản lý sinh viên gồm có 1 combobox chọn lớp, 1 gridview để hiển thị danh sách sinh viên và 1 button để thực hiện lệnh liệt kê danh sách.

  (1) Người dùng chọn combobox lớp trên GUI và ấn button liệt kê

  (2) GUI kiểm tra yêu cầu chọn combobox hợp lệ và gởi mã lớp (**) vừa chọn sang BUS xử lý yêu cầu hiển thị danh sách điểm sinh viên

  (4) Tại BUS vì yêu cầu từ GUI khá đơn giản nên BUS sẽ không xử lý gì mà sẽ gởi mã lớp sang DAL lấy danh sách điểm.

  (5) Tại DAL sau khi đã nhận được yêu cầu lấy danh sách điểm từ mã lớp, DAL sẽ tương tác với hệ quản trị CSDL (6) qua các lệnh mở tập tin, kết nối, truy vấn,… để lấy được danh sách điểm (7) với mã số yêu cầu, DAL tiếp tục gởi danh sách (**) này sang BUS để xử lý (7)

  (8) Tại BUS sau khi nhận được danh sách điểm từ DAL gởi sang, BUS thực hiện nghiệp vụ của mình bằng cách tính điểm trung bình, kết luận đậu/rớt của từng sinh viên (tất cả xử lý về mặt nghiệp vụ), sau đó gởi danh sách điểm đã xử lý (**) sang GUI để hiển thị (9)

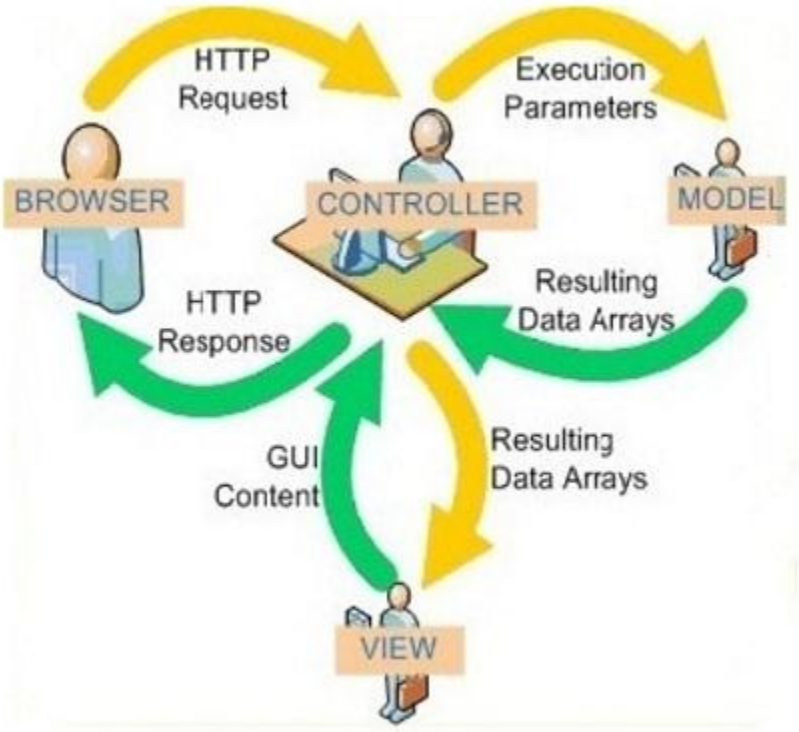
  (9) 1 lần nữa GUI có thể kiểm tra tính hợp lệ của dữ liệu và hiển thị thông tin và thông báo lên màn hình cho người dùng (10)

  (**) *Trong 1 số trường hợp vì lượng thông tin gởi nhiều, ví dụ như 1 sinh viên gồm nhiều thuộc tính như họ tên, tuổi, ngày sinh,… ta có thể dùng DTO để chuyển đối tượng hoặc danh sách đối tượng giữa các tầng với nhau cho tiện dụng.*

4. **Nêu các thành phần của hệ thống khi xây dựng cấu trúc đa giao diện.**
    - **Các công cụ để xây dựng hệ thống.**
5. **Trình bày các mô hình phân lớp mà bạn tìm hiểu (MVC, MVP, MVVM, …)**
    - **MVC**

- **Tổng quan:**
  Vào những năm 70 của thế kỷ 20, tại phòng thí nghiệm Xerox PARC ở Palo Alto. Sự ra đời của giao diện đồ họa (Graphical User Interface - GUI) và lập trình hướng đối tượng (Object Oriented Programming - OOP) cho phép lập trình viên làm việc với những thành phần đồ họa như những đối tượng đồ họa có thuộc tính và phương thức riêng của nó. Không dừng lại ở đó, những nhà nghiên cứu ở Xerox PARC còn đi xa hơn nữa khi họ cho ra đời cái gọi là kiến trúc MVC (viết tắt của Model – View – Controller).
  Kiến trúc này ngày càng được phát triển và hoàn thiện nhằm giải quyết các vấn đề phát sinh cũng như các giải pháp cho quá trình phát triển phần mềm.
- **Các thành phần trong MVC**
  Trong kiến trúc này, hệ thống được chia thành 3 tầng tương ứng đúng với tên gọi của nó (Model – View – Controller). Ở đó nhiệm vụ cụ thể của các tầng được phân chia như sau:

  1. **Model (Tầng dữ liệu):** là một đối tượng hoặc một tập hợp các đối tượng biểu diễn cho phần dữ liệu của chương trình. Nó được giao nhiệm vụ cung cấp dữ liệu cho cơ sở dữ liệu và lưu dữ liệu vào các kho chứa dữ liệu. Tất cả các nghiệp vụ logic được thực thi ở Model. Dữ liệu vào từ người dùng sẽ thông qua View đến Controller và được kiểm tra ở Model trước khi lưu vào cơ sở dữ liệu. Việc truy xuất, xác nhận, và lưu dữ liệu là một phần của Model



Hình 1: Luồng xử lý của mô hình MVC

2. **View (Tầng giao diện):** là phần giao diện với người dùng, bao gồm việc hiện dữ liệu ra màn hình, cung cấp các menu, nút bấm, hộp đối thoại, chọn lựa …, để người dùng có thể thêm, xóa. sửa, tìm kiếm và làm các thao tác khác đối với dữ liệu trong hệ thống.. Thông thường, các thông tin cần hiển thị được lấy từ thành phần Models.

3. **Controller (Tầng điều khiển):** là phần điều khiển của ứng dụng, điều hướng các nhiệm vụ (task) đến đúng phương thức (method) có chức năng xử lý nhiệm vụ đó. Nó chịu trách nhiệm xử lý các tác động về mặt giao diện, các thao tác đối với models, và cuối cùng là chọn một view thích hợp để hiển thị ra màn hình.

- **<u>Ưu điểm và nhược điểm:</u>**

1. Ưu điểm:

Phát triển phần mềm: Có tính chuyên nghiệp hóa, có thể chia cho nhiều nhóm được đào tạo nhiều kỹ năng khác nhau, từ thiết kế mỹ thuật cho đến lập trình đến tổ chức database. Giúp phát triển ứng dụng nhanh, đơn giản, dễ nâng cấp..

Bảo trì: Với các lớp được phân chia theo như đã nói, thì các thành phần của một hệ thống dễ được thay đổi, nhưng sự thay đổi có thể được cô lập trong từng lớp, hoặc chỉ ảnh hưởng đến lớp ngay gần kề của nó, chứ không phát tán náo loạn trong cả chương trình.

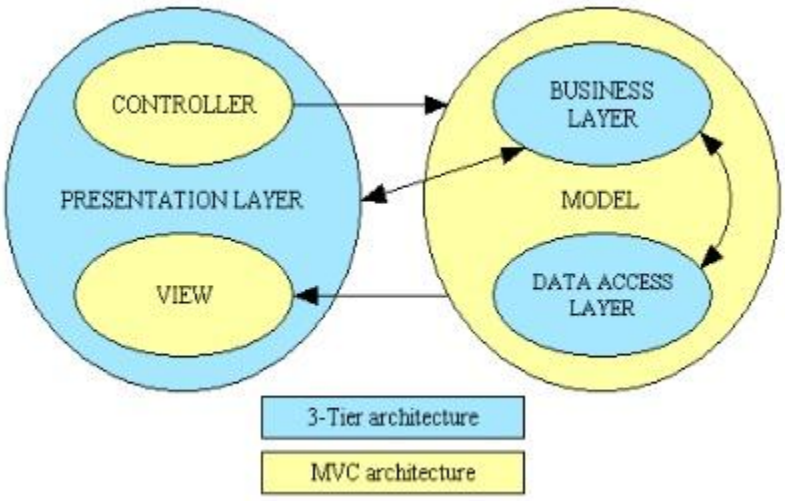
Mở rộng: Với các lớp được chia theo ba lớp như đã nói, việc thêm chức năng vào cho từng lớp sẽ dễ dàng hơn là phân chia theo cách khác.

2. Nhược điểm:

Đối với dự án nhỏ việc áp dụng mô hình MC gây cồng kềnh, tốn thời gian trong quá trình phát triể.

Tốn thời gian trung chuyển dữ liệu của các tầng

- **<u>So sánh mô hình MVC với mô hình 3 lớp</u>**



Hình 2: So sánh mô hình MVC với mô hình 3 lớp

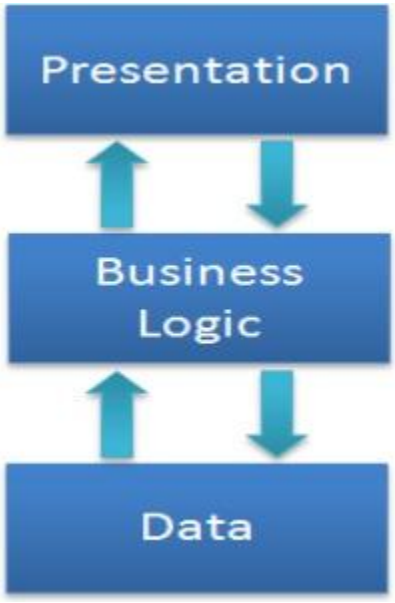
1. Điểm giống:

Tách rời programming core/business logic ra khỏi những phụ thuộc về tài nguyên và môi trường.

Presentation Layer (PL) thể hiện giống như chức năng của View và Controller. Business Layer (BL) và Data Access Layer (DL) thể hiện giống như chức năng của Model. Như thế nhìn ở góc độ này, thì MVC tương đương với 3-layer (tất nhiên có chồng chéo như hình vẽ)

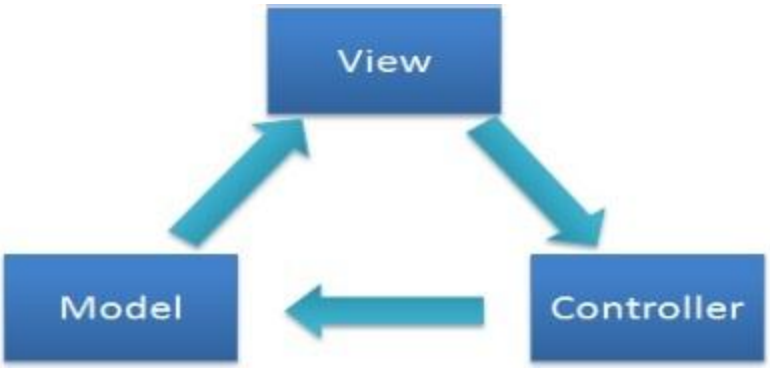
2. Điểm khác:

Trong mô hình 3 lớp, quá trình đi theo chiều dọc, bắt đầu từ PL, sang BL, rồi tới DL, và từ DL, chạy ngược lại BL rồi quay ra lại PL.



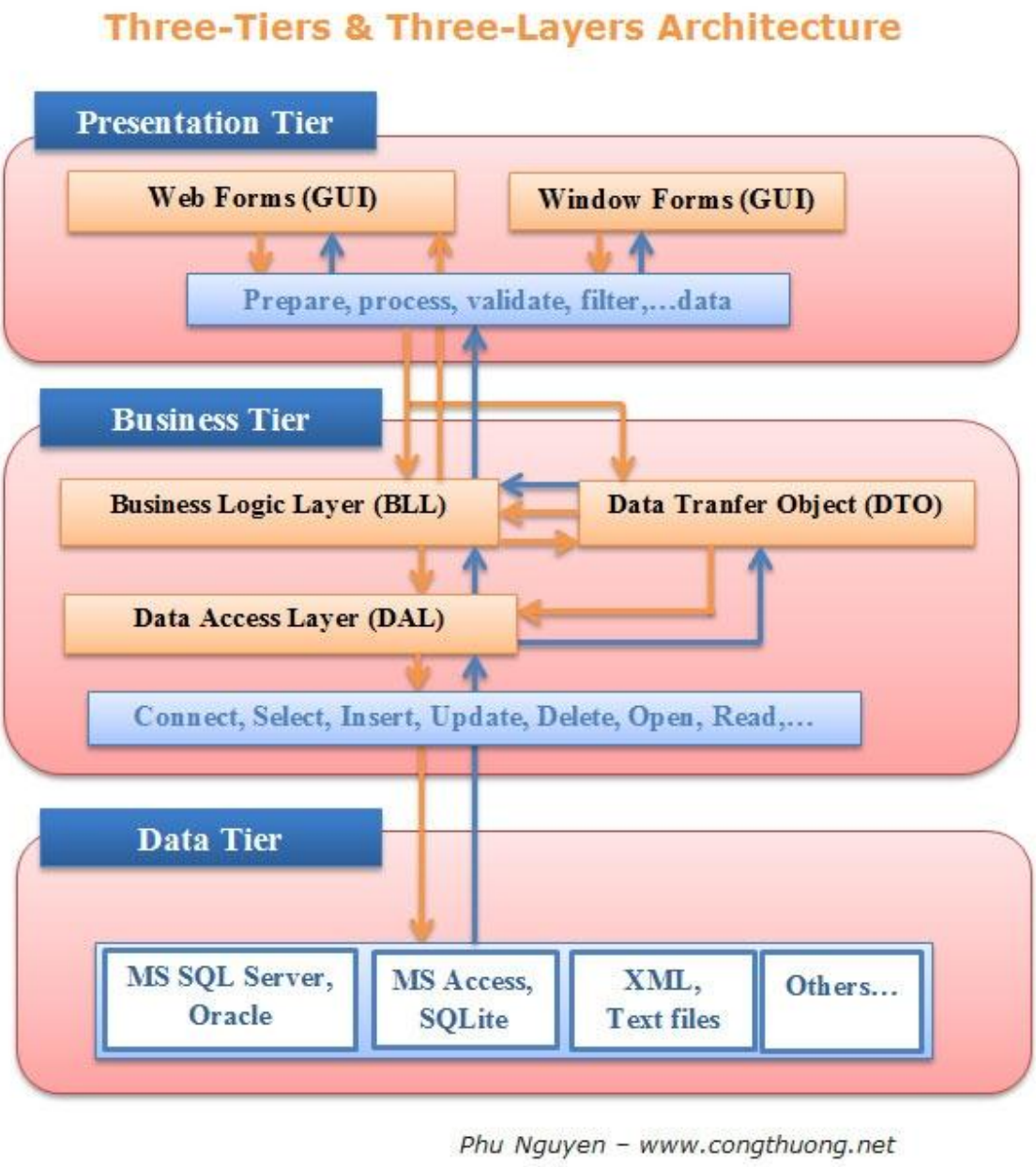
Hình 3: Mô hình 3 lớp

Còn trong mô hình MVC, dữ liệu được nhận bởi View, View sẽ chuyển cho Controller cập nhật vào Model, rồi sau đó dữ liệu trong Model sẽ được đưa lại cho View mà không thông qua Controller, do vậy luồng xử lý này có hình tam giác.



Hình 4: Mô hình MVC

- ☞ Thông thường khi áp dụng thì người ta kết hợp cả 2: MVC được áp dụng bên phía Client. Sau đó 3 tiers được áp dụng như bình thường trên hệ thống client-server. Việc có sử dụng hay là có **bọc lẫn nhau giữa MVC và 3-tier** là không hoàn toàn bắt buộc, ta có thể sử dụng MVC mà không cần đến 3-tier (với những ứng dụng nhỏ)
- ☞ 3-tiers 3-layers

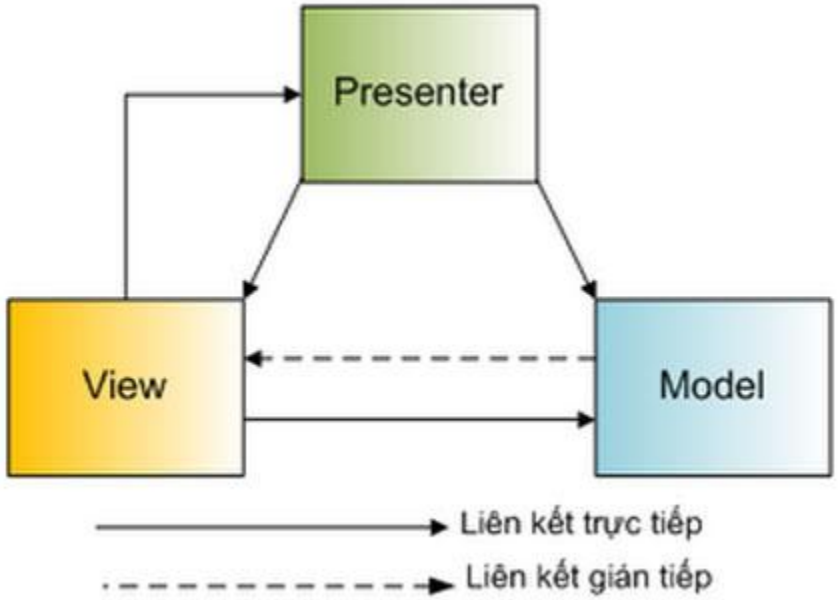


MVP

- **Tổng quan**
  Mẫu kiến trúc Dolphin Smalltalk Model-View-Presenter chia ứng dụng thành các phần dữ liệu (data), trình bày (presentation) và các xử lý logic thuộc phần trình bày (presentation logic) thành những thành phần riêng biệt.
- **Lịch sử**
  Phiên bản Dolphin Smalltalk Model-View-Presenter (gọi tắt là Dolphin MVP) là phiên bản của MVP được xây dựng dựa trên phiên bản Taligent MVP xuất hiện

trước đó. Dolphin MVP được xây dựng về cơ bản bên ngoài tương tự như MVC cổ điển nhưng khác nhau ở vai trò của Controller và Presenter.

- **Cấu trúc**



- **Các thành phần**
  Model chứa dữ liệu và các tính toán xử lý logic để giải quyết vấn đề mà phần mềm hướng tới (business logic).
  View là thành phần đảm nhận trình bày từ những dữ liệu của Model. View bao gồm những gì thể hiện trên màn hình như các control, form, widget,…
  Presenter là thành phần đảm nhận các xử lý về trình bày mà nó cần đến sự tương tác trên dữ liệu.
- **Phối hợp các thành phần**
  Trong khi vai trò của Model không thay đổi so với MVC cổ điển. Thành phần View của Dolphin MVP cũng thực hiện việc trình bày từ nội dung của Model và gắn kết với Model thông qua Observer Pattern. Tuy nhiên, nó thực hiện bắt các sự kiện xảy ra ngay bên trong nó. Một vài trường hợp đơn giản như TextView, dữ liệu nhập được xử lý trực tiếp và cập nhật thay đổi trực tiếp vào Model còn hầu hết các trường hợp khác, việc xử lý được chuyển giao cho Presenter đảm trách khi cần thao tác đến Model.

  Trong khi View đảm nhận trình bày thì Presenter đảm trách cách Model được thao tác và thay đổi như thế nào bởi giao diện người dùng. Presenter là nơi chứa các xử lý đặc trưng của ứng dụng (application logic so với business logic của Model). Một điểm đáng chú ý khác là Presenter có khả năng thao tác trực tiếp lên View mà nó gắn kết, điều này khác biệt với phiên bản Taligent MVP xuất hiện trước đó.

- **So sánh Dolphin MVP và MVC cổ điển**
  Điểm khác biệt cơ bản của Dolphin MVP và MVC cổ điển là sự khác nhau về vai trò của Presenter và Controller, nó cũng dẫn đến sự khác nhau về vai trò giữa hai thành phần View. Trong Dolphin MVP, sự hiện diện của Controller bị loại bỏ, thay vào đó, việc xử lý các dữ liệu input được View đảm nhận và được chuyển cho Presenter khi có yêu cầu tương tác đến Model.

    MVVM
- **View**: Tương tự như trong mô hình MVC, View là phần giao diện của ứng dụng để hiển thị dữ liệu và nhận tương tác của người dùng. Một điểm khác biệt so với các ứng dụng truyền thống là View trong mô hình này tích cực hơn. Nó có khả năng thực hiện các hành vi và phản hồi lại người dùng thông qua tính năng binding, command.
- **Model**: Cũng tương tự như trong mô hình MVC. Model là các đối tượng giúp truy xuất và thao tác trên dữ liệu thực sự.
- **ViewModel**: Lớp trung gian giữa View và Model. ViewModel có thể được xem là thành phần thay thế cho Controller trong mô hình MVC. Nó chứa các mã lệnh cần thiết để thực hiện data binding, command.
  Một điểm cần lưu ý là trong mô hình MVVM, các tầng bên dưới sẽ không biết được các thông tin gì về tầng bên trên nó. Như hình minh họa dưới đây:

Mô hình 3 tầng (3-tiers) là gì?

Theo wikipedia:

Trích dẫn:"3-tiers là một kiến trúc kiểu client/server mà trong đó giao diện người dùng (UI-user interface), các quy tắc xử lý (BR-business rule hay BL-business logic), và việc lưu trữ dữ liệu được phát triển như những module độc lập, và hầu hết là được duy trì trên các nền tảng độc lập, và mô hình 3 tầng (3-tiers) được coi là một kiến trúc phần mềm và là một mẫu thiết kế." (dịch lại từ wikipedia tiếng Anh).

3-Tiers có tính vật lý (physical): là mô hình client-server (mỗi tier có thể đặt chung 1 nơi hoặc nhiều nơi, kết nối với nhau qua Web services, WCF, Remoting...). Như hình vẽ ta thấy 3 tầng rõ rệt 3 tầng:

+ Presentation tier bao gồm các thành phần phần xử lý giao diện Graphic User Interface (GUI)
+ Business tier gồm các thành phần Business Logic Layer (BLL), Data Access Layer (DAL) và Data Tranfer Object (DTO): xem thêm phần 3-layers
+ Data tier lưu trữ dữ liệu, là các hệ quản trị CSDL như MS SQL Server, Oracle, SQLite, MS Access, XML files, text files,...

Tuy nhiên bạn cần chú ý những ưu và nhược điểm sau đây để áp dụng nó một cách đúng đắn.
Ưu điểm:
- Dễ dàng mở rộng, thay đổi quy mô của hệ thống: Khi cần tải lớn, người quản trị có thể dễ dàng thêm các máy chủ vào nhóm, hoặc lấy bớt ra trong trường hợp ngược lại.
Nhược điểm:
- Việc truyền dữ liệu giữa các tầng sẽ chậm hơn vì phải truyền giữa các tiến trình khác nhau (IPC), dữ liệu cần phải được đóng gói -> truyền đi -> mở gói trước khi có thể dùng được.
- Việc phát triển ứng dụng phức tạp hơn.

2. Mô hình 3 lớp (3-layers) là gì?

Không như 3-Tiers có tính vật lý, 3-Layers có tính logic (mỗi layer có 1 công việc) và là 1 thành phần của 3-Tiers. Gồm 3 lớp chính:
+ Graphic User Interface (GUI): Thành phần giao diện, là các form của chương trình tương tác với người sử dụng.
+ Business Logic Layer (BLL): Xử lý các nghiệp vụ của chương trình như tính toán, xử lý hợp lệ và toàn vẹn về mặt dữ liệu.
+ Data Access Layer (DAL): Tầng giao tiếp với các hệ quản trị CSDL

Trong 1 số trường hợp vì lượng thông tin gởi nhiều ta có thể dùng Data Tranfer Object (DTO) để chuyển đối tượng hoặc danh sách đối tượng giữa các tầng với nhau cho tiện dụng.

SỰ KHÁC BIỆT GIỮA ASP.NET MVC VÀ ASP.NET WEBFORM

Trước khi nêu ra sự khác biệt ASP.net MVC và ASP.net WebFrom ta cần biết một số khái niệm cơ bản về mô hình MVC.

1. Mô hình MVC cơ bản:

MVC là viết tắt chữ cái đầu của Models, Views, Controllers. MVC chia giao diện UI (User Interface) thành 3 phần tương ứng, đầu vào của các controller là các điều khiển thông qua HTTP request,

model chứa các miền logic, view là những thứ được sinh ra trả về cho trình duyệt. Sau đây là một vài chi tiết trong 3 thành phần của MVC:

- Model: Được giao nhiệm vụ cung cấp dữ liệu cho cơ sở dữ liệu và lưu dữ liệu vào các kho chứa dữ liệu. Tất cả các nghiệp vụ logic được thực thi ở Model. Dữ liệu vào từ người dùng sẽ thông qua View để kiểm tra ở Model trước khi lưu vào cơ sở dữ liệu. Việc truy xuất, xác nhận và lưu dữ liệu là một phần của Model.

- View: Hiển thị các thông tin cho người dùng của ứng dụng và được giao nhiệm vụ cho việc nhận các dữ liệu vào từ người dùng, gởi đi các yêu cầu đến bộ điều khiển, sau đó là nhận lại các phản hồi từ bộ điề khiển và hiển thị kết quả cho người dùng. Các trang HTML, JSP, các thư viện thể và các file nguồn là một phần của View.

- Controller: là tầng trung gian giữa Model và View. Controller được giao nhiệm vụ nhận các yêu cầu từ phía máy khách. Một yêu cầu được nhận từ máy khách được thực hiện bởi một chức năng logic thích hợp từ thành phần Model và sau đó sinh ra các kết quả cho người dùng và được thành phần View hiển thị. ActionServlet, Action, ActionForm, struts-config.xml là các thành phần của Controller.

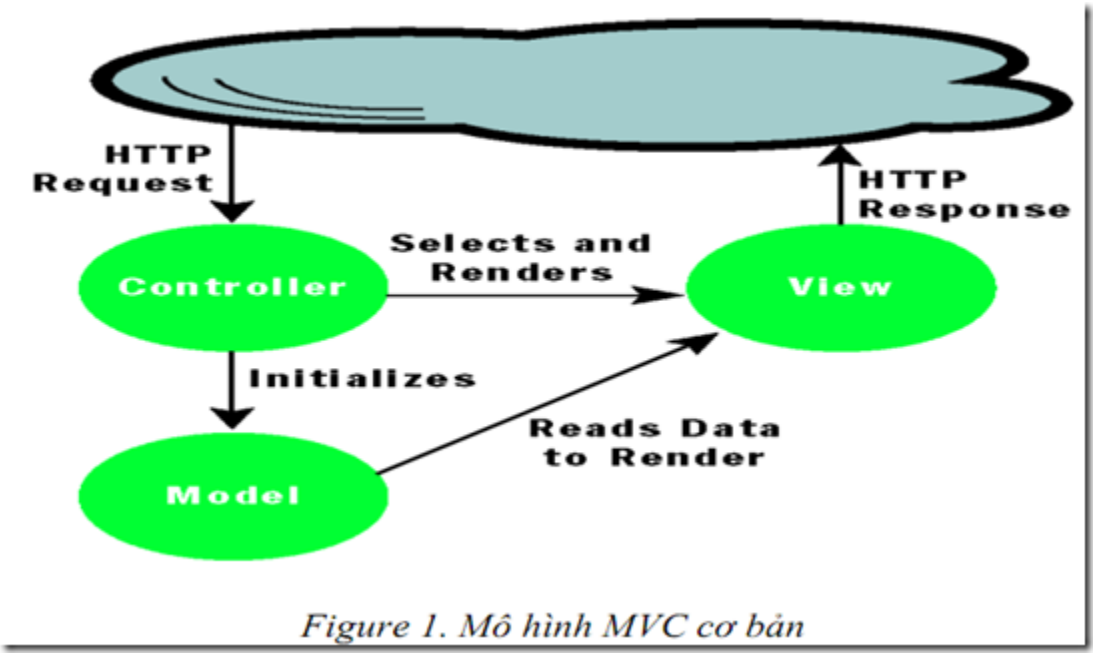


Figure 1. Mô hình MVC cơ bản

Lợi ích của việc dùng phương pháp MVC là sự phân đoạn rõ ràng giữa Models, Views, Controllers bên trong ứng dụng. Một cấu trúc sạch sẽ giúp cho việc kiểm tra ứng dụng của bạn dễ dàng hơn.

2. Một vài đặc tính trong ASP.net MVC.

- Tách rõ ràng các mối liên quan, mở khả năng test TDD (Test Driven Developer). Có thể test unit trong ứng dụng mà không cần phải chạy Controllers cùng với tiến trình của ASP.net và có thể dùng bất kỳ một unit testing framework như NUnit, MBUnit, MS Test…

- Có khả năng mở rộng, mọi thứ trong MVC được thiết kế dễ dàng thay thế/ tùy biến (ví dụ có thể lựa chọn engine view riêng routing policy, parameter serialization, …).

- Bao gồm ánh xạ URL mạnh mẽ, cho phép xây dựng ứng dụng với những URL sạch, các URL không cần cs mở rộng (ví dụ có thể ánh xạ địa chỉ /Products/Edit/4 để thực hiện hành động edit của lớp điều khiển ProductControllers hoặc ánh xạ địa chỉ Blog/SomeTopic để thực hiện hành động "Display Topic" của lớp điều khiển BlogEngineController).

- ASP.net MVC Framework cũng hỗ trợ file ASP.net như .ASPX .ASCX và .Master đánh dấu các tập tin này như một "view template" (có thể dễ dàng sử dụng các tính năng của ASP.net như lồng các trang Master, <%=%> snippets, mô tả server controls, template, data-binding, localization…). Tuy nhiên sẽ không còn postback và interactive back server và thay vào đó là interactive end-user với một controller class (không còn viewstate, page lifecycle).

- ASP.net MVC Framework hỗ trợ đầy đủ các tính năng bảo mật của ASP.net như Form/ Windows authenticate, URL authorization, membership/roles, output và data caching, section/ profile state, configuration system, provider architecture, …

3. Sự khác biệt Giữa MVC và WebForm:

- ASP.net WebForm sử dụng ViewState để quản lý, các trang ASP.net đều có lifecycle, postback và dùng các web controls, các event để thực hiện các hành động cho UI (User Interface) khi có sự tương tác với người dùng nên hầu hết ASP.net WebForm xử lý chậm.

- ASP.net MVC chia ra làm 3 phần: Models, View, Controller. Mọi tương tác của người dùng với Views sẽ được thực hiện hành động trong Controllers, không còn postback, lifecycel và events.

- Việc kiểm tra (test), gỡ lỗi (debug) với ASP.net WebForm đều phải chạy tất cả các tiến trình của ASP.net, và sự thay đổi ID của bất kỳ Controls nào cũng ảnh hưởng đến ứng dụng. Đối với MVC thì việc đó có thể sử dụng các unit test có thể thẩm định rất dễ dàng các Controllers thực hiện như thế nào.

- Sau đây là bảng so sánh các tính năng của ASP.net WebForm với ASP.net MVC

Các tính năng	ASP.net WebForm	ASP.net MVC
Kiến trúc chương trình	Kiến trúc mô hình WebForm –> Bussiness –> Database	Kiến trúc sử dụng việc phân chia chương trình thành: Models, Views, Controllers
Cú pháp chương trình	Sử dụng cú pháp của WebForm, tất cả các sự kiện và controls do server quản lý	Các sự kiện được điều khiển bởi controllers, các controls không do server quản lý.
Truy cập dữ liệu	Sử dụng hầu hết các công nghệ truy cập dữ liệu trong ứng dụng	Phần lớn dùng LINQ và SQL class để tạo mô hình truy cập đối tượng.
Debug	Debug phải thực hiện tất cả bao gồm các lớp truy cập dữ liệu, sự hiển thị, điều khiển các controls.	Debug có thể sử dụng các unit test để kiểm tra các phương thức trong controllers.
Tốc độ phân tải	Tốc độ phân tải chậm khi trong trang có quá nhiều các controls vì ViewState quá lớn	Phân tải nhanh hơn do không phải quản lý ViewState để quản lý các controls trong trang.
Tương tác với JavaScript	Tương tác với JavaScript khó khăn vì các controls được điều	Tương tác với JavaScript dễ dàng vì các đối tượng không do

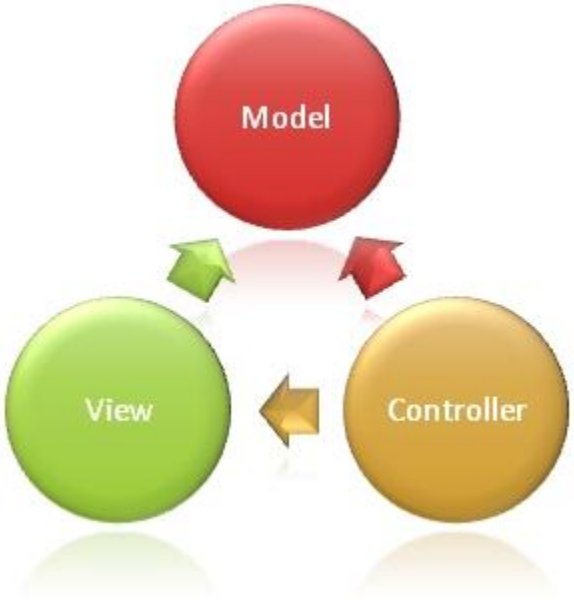
	khiển bởi server	server quản lý điều khiển không khó
URL address	Cấu trúc địa chỉ URL có dạng: <filename>.aspx?&<các tham số>	Cấu trúc địa chỉ rành mạch theo dạng Controllers/Action/ID

Tổng quan về ASP.NET MVC

Mẫu kiến trúc Model – View – Controller được sử dụng nhằm chi ứng dụng thành ba thành phần chính: model, view và controller. Nền tảng ASP.NET MVC giúp cho chúng ta có thể tạo được các ứng dụng web áp dụng mô hình MVC thay vì tạo ứng dụng theo mẫu ASP.NET Web Forsm. Nền tảng ASP.NET MVC có đặc điểm nổi bật là nhẹ (lighweigt), dễ kiểm thử phần giao diện (so với ứng dụng Web Forms), tích hợp các tính năng có sẵn của ASP.NET. Nền tảng ASP.NET MVC được định nghĩa trong namespace System.Web.Mvc và là một phần của name space System.Web.

MVC là một mẫu thiết kế (design pattern) chuẩn mà nhiều lập trình viên đã quen thuộc. Một số loại ứng dụng web sẽ thích hợp với kiến trúc MVC. Một số khác vẫn thích hợp với ASP.NET Web Forms và cơ chế postbacks. Đôi khi có những ứng dụng kết hợp cả hai kiến trúc trên.

Nền tảng MVC bao gồm các thành phần dưới đây:



***Hình 01**: Mẫu Model – View – Controller*

Models: Các đối tượng Models là một phần của ứng dụng, các đối tượng này thiết lập logic của phần dữ liệu của ứng dụng. Thông thường, các đối tượng model lấy và lưu trạng thái của model trong CSDL. Ví dụ như, một đối tượng Product (sản phẩm) sẽ lấy dữ liệu từ CSDL, thao tác trên dữ liệu và sẽ cập nhật dữ liệu trở lại vào bảng Products ở SQL Server.

Trong các ứng dụng nhỏ, model thường là chỉ là một khái niệm nhằm phân biệt hơn là được cài đặt thực thụ, ví dụ, nếu ứng dụng chỉ đọc dữ liệu từ CSDL và gởi chúng đến view, ứng dụng khong cần phải có tầng model và các lớp lien quan. Trong trường hợp này, dữ liệu được lấy như là một đối

tượng model (hơn là tầng model).

Views: Views là các thành phần dùng để hiển thị giao diện người dùng (UI). Thông thường, view được tạo dựa vào thông tin dữ liệu model. Ví dụ như, view dùng để cập nhật bảng Products sẽ hiển thị các hộp văn bản, drop-down list, và các check box dựa trên trạng thái hiện tại của một đối tượng Product.

Controllers: Controller là các thành phần dùng để quản lý tương tác người dùng, làm việc với model và chọn view để hiển thị giao diện người dùng. Trong một ứng dụng MVC, view chỉ được dùng để hiển thị thông tin, controller chịu trách nhiệm quản lý và đáp trả nội dung người dùng nhập và tương tác với người dùng. Ví dụ, controller sẽ quản lý các dữ liệu người dùng gởi lên (query-string values) và gởi các giá trị đó đến model, model sẽ lấy dữ liệu từ CSDL nhờ vào các giá trị này.

Mẫu MVC giúp bạn tạo được các ứng dụng mà chúng phân tách rạch ròi các khía cạnh của ứng dụng (logic về nhập liệu, logic xử lý tác vụ và logic về giao diện). Mẫu MVC chỉ ra mỗi loại logic kể trên nên được thiếp lập ở đâu trên ứng dụng. Logic giao diện (UI logic) thuộc về views. Logic nhập liệu (input logic) thuộc về controller. Và logic tác vụ (Business logic – là logic xử lý thông tin, mục đích chính của ứng dụng) thuộc về model. Sự phân chia này giúp bạn giảm bớt được sự phức tạp của ứng dụng và chỉ tập trung vào mỗi khía cạnh cần được cài đặt ở mỗi thời điểm. Ví dụ như bạn chỉ cần tập trung vào giao diện (views) mà không phải quan tâm đến logic xử lý thông tin của ứng dụng.

Để quản lý sự phức tạp của ứng dụng, mẫu MVC giúp cho chúng ta có thể kiểm thử ứng dụng dễ dàng hơn hẳn so với khi áp dụng mẫu Web Forms. Ví dụ, trong một ứng dụng ASP.NET Web Forms, một lớp thường được sử dụng để hiển thị thông tin xuất ra cho người dùng và đồng thời xử lý thông tin người dùng nhập. Việc xây dựng các bộ test tự động cho ứng dụng Web Forms là rất phức tạp, bởi để kiểm thử mỗi trang web, bạn phải khởi tạo đối tượng trang, khởi tạo tất cả các control được sử dụng trong trang và các lớp phụ thuộc trong ứng dụng. Và bởi vì có quá nhiều lớp cần được khởi tạo để chạy được trang, thật khó để có thể viết các test chỉ tập trung vào một khía cạnh nào đó của ứng dụng. Và vì thế, kiểm thử đối với các ứng dụng dựa trên nền tảng Web Forms sẽ khó khăn hơn nhiều so với khi áp dụng trên ứng dụng MVC. Hơn thế nữa, việc kiểm thử trên nền tảng Web Forms yêu cầu phải sử dụng đến web server. Nền tảng MVC phân tách các thành phần và sử dụng các interface (khái niệm giao diện trong lập trình hướng đối tượng), và nhờ đó có thể kiểm thử các thành phần riêng biệt trong tình trạng phân lập với các yếu tố còn lại của ứng dụng.

Sự phân tách rạch ròi ba thành phần của ứng dụng MVC còn giúp cho việc lập trình diễn ra song song. Ví dụ như một lập trình viên làm việc với view, lập trình viên thứ hai lo cài đặt logic của controller và lập trình viên thứ ba có thể tập trung vào logic tác vụ của model tại cùng một thời điểm.

Lựa chọn áp dụng MVC trong xây dựng ứng dụng

Bạn cần phải xem xét kỹ càng việc áp dụng mô hình ASP.NET MVC hay mô hình ASP.NET Web Forms khi xây dựng một ứng dụng. Mô hình MVC không phải là mô hình thay thế cho Web Forms, bạn có thể dùng một trong hai mô hình.

Trước khi quyết định sử dụng MVC hay Web Forms cho một web site cụ thể, bạn cần phải phân tích lợi ích khi chọn một trong hai hướng.

Lợi ích của ứng dụng web dựa trên mô hình MVC

Nền tảng ASP.NET MVC mang lại những lợi ích sau:

- Dễ dàng quản lý sự phức tạp của ứng dụng bằng cách chia ứng dụng thành ba thành phần model, view, controller
- Nó không sử dụng view state hoặc server-based form. Điều này tốt cho những lập trình viên muốn quản lý hết các khía cạnh của một ứng dụng.
- Nó sử dụng mẫu Front Controller, mẫu này giúp quản lý các requests (yêu cầu) chỉ thông qua một Controller. Nhờ đó bạn có thể thiết kế một hạ tầng quản lý định tuyến. Để có nhiều thông tin hơn, bạn nên xem phần Front Controller trên web site MSDN
- Hỗ trợ tốt hơn cho mô hình phát triển ứng dụng hướng kiểm thử (TDD)
- Nó hỗ trợ tốt cho các ứng dụng được xây dựng bởi những đội có nhiều lập trình viên và thiết kế mà vẫn quản lý được tính năng của ứng dụng

Lợi ích của ứng dụng được xây dựng trên nền tảng Web Forms

- Nó hỗ trợ cách lập trình hướng sự kiện, quản lý trạng thái trên giao thức HTTP, tiện dụng cho việc phát triển các ứng dụng Web phục vụ kinh doanh. Các ứng dụng trên nền tảng Web Forms cung cấp hàng tá các sự kiện được hỗ trợ bởi hàng trăm các server controls.
- Sử dụng mẫu Page Controller. Xem thêm ở mục Page Controller trên MSDN
- Mô hình này sử dụng view state hoặc server-based form, nhờ đó sẽ giúp cho việc quản lý trạng thái các trang web dễ dàng.
- Nó rất phù hợp với các nhóm lập trình viên quy mô nhỏ và các thiết kế, những người muốn tận dụng các thành phần giúp xây dựng ứng dụng một cách nhanh chóng.
- Nói tóm lại, áp dụng Web Forms giúp giảm bớt sự phức tạp trong xây dựng ứng dụng, bởi vì các thành phần (lớp Page, controls,…) được tích hợp chặc chẽ và thường thì giúp bạn viết ít code hơn là áp dụng theo mô hình MVC.

Các tính năng của nền tảng ASP.NET MVC

- Tách bạch các tác vụ của ứng dụng (logic nhập liệu, business logic, và logic giao diện), dễ dàng kiểm thử và mặc định áp dụng hướng phát triển TDD. Tất cả các tính năng chính của mô hình MVC được cài đặt dựa trên interface và được kiểm thử bằng cách sử dụng các đối tượng mocks, mock object là các đối tượng mô phỏng các tính năng của những đối tượng thực sự trong ứng dụng. Bạn có thể kiểm thử unit-test cho ứng dụng mà không cần chạy controller trong tiến trình ASP.NET, và điều đó giúp unit test được áp dụng nhanh chóng và tiện dụng. Bạn có thể sử dụng bất kỳ nền tảng unit-testing nào tương thích với nền tảng .NET.
- MVC là một nền tảng khả mở rộng (extensible) & khả nhúng (pluggable). Các thành phần của ASP.NET MVC được thiết kế để chúng có thể được thay thế một cách dễ dàng hoặc dễ dàng tùy chỉnh. Bạn có thể nhúng thêm view engine, cơ chế định tuyến cho URL, cách kết xuất tham số của action-method và các thành phần khác. ASP.NET MVC cũng hỗ trợ việc sử dụng Dependency Injection (DI) và Inversion of Control (IoC). DI cho phép bạn gắn các đối tượng vào một lớp cho lớp đó sử dụng thay vì buộc lớp đó phải tự mình khởi tạo các đối tượng. IoC quy định rằng, nếu một đối tượng yêu cầu một đối tượng khác, đối tượng đầu sẽ lấy đối tượng thứ hai từ một nguồn bên ngoài, ví dụ như từ tập tin cấu hình. Và nhờ vậy, việc sử dụng DI và IoC sẽ giúp kiểm thử dễ dàng hơn.
- ASP.NET MVC có thành phần ánh xạ URL mạnh mẽ cho phép bạn xây dựng những ứng dụng có các địa chỉ URL xúc tích và dễ tìm kiếm. Các địa chỉ URL không cần phải có phần mở rộng của tên tập tin và được thiết kế để hỗ trợ các mẫu định dạng tên phù hợp với việc tối ưu hóa tìm kiếm (URL) và phù hợp với lập địa chỉ theo kiểu REST.
- Hỗ trợ sử dụng đặc tả (các thẻ) của các trang ASP.NET(.aspx), điều khiển người dùng (.ascx) và trang master page (.marter). Bạn có thể sử dụng các tính năng có sẵn của ASP.NET như là sử dụng lồng các trang master page, sử dụng in-line expression (<%= %>), sử dụng server controls, mẫu, data-binding, địa phương hóa (localization) và hơn thế nữa.

- Hỗ trợ các tính năng có sẵn của ASP.NET như cơ chế xác thực người dùng, quản lý thành viên, quyền, output caching và data caching, seession và profile, quản lý tình trạng ứng dụng, hệ thống cấu hình…
- ASP.NET MVC 3 còn bổ sung một view engine mới là Razor View Engine cho phép thiết lập các view nhanh chóng, dễ dàng và tốn ít công sức hơn so với việc sử dụng Web Forms view engine.