1.Thứ tự thực hiện các từ khóa trong SQL:

FROM>ON>JOIN>WHERE>GROUP BY> HAVING> SELECT >DISTINCT >ORDER BY >TOP

2.Bundle: là tính năng để nối hoặc gom nhiều file vào 1 file. Ít file hơn có nghĩa là it HTTP Requests hơn và có thể cải thiện hiệu suất load trang

3. ATOM: Kiểu Symbol là một kiểu mới trong Javascript tiêu chuẩn ECMAScript 6. Mỗi Symbol là một giá trị sơ khai **đơn nhất** và **bất biến** và có thể được dùng như một khóa của một Object Trên một số ngôn ngữ lập trình, Symbol còn được gọi là "**atom**"

4. Global Filter chạy khi Application start, chạy trước khi chạy vào Route để lọc các lỗi.

5. Bắt đầu của HTTP Request sẽ là dòng Request-Line bao gồm 3 thông tin đó là:

* Method: là phương thức mà HTTP Request này sử dụng, thường là GET, POST, ngoài ra còn một số phương thức khác như PUT, DELETE
* URI: là địa chỉ định danh của tài nguyên. URI là / - tức request cho tài nguyên gốc, nếu request không yêu cầu một tài nguyên cụ thể, URI có thể là dấu \*.
* HTTP version: là phiên bản HTTP đang sử dụng, ở đây là HTTP 1.1.

Tiếp theo là các trường request-header, cho phép client gửi thêm các thông tin bổ sung về thông điệp HTTP request và về chính client. Một số trường thông dụng như:

* Accept: loại nội dung có thể nhận được từ thông điệp response. Ví dụ: text/plain, text/html…
* Accept-Encoding: các kiểu nén được chấp nhận. Ví dụ: gzip, xz, …
* Connection: tùy chọn điều khiển cho kết nối hiện thời. Ví dụ: Upgrade…
* Cookie: thông tin HTTP Cookie từ server.
* User-Agent: thông tin về user agent của người dùng.

Cuối cùng là phần Body có thể chứa message hoặc dữ liệu gửi lên

RESPONSE:

- Dòng Status line gồm 3 trường là phiên bản giao thức (HTTP version), mã trạng thái (Status code) và mô tả  trạng thái (Status text):

* + **Phiên bản giao thức (HTTP version)**: phiên bản của giao thức HTTP mà server hỗ trợ, thường là HTTP/1.0 hoặc HTTP/1.1
  + **Mã trạng thái (Status code)**: mô tả trạng thái kết nối dưới dạng số, mỗi trạng thái sẽ được biểu thị bởi một số nguyên. Ví dụ: 200, 404, 302,…
  + **Mô tả trạng thái (Status text)**: mô tả trạng thái kết nối dưới dạng văn bản một cách ngắn gọn, giúp người dùng dễ hiểu hơn so với mã trạng thái. Ví du: 200 OK, 404 Not Found, 403 Forbiden,…

- Các dòng Header line của gói tin response có chức năng tương tự như gói tin request, giúp server có thể truyền thêm các thông tin bổ sung đến client dưới dạng các cặp *“Name:Value”*.

- Phần Body là nơi đóng gói dữ liệu để trả về cho client, thông thường trong duyệt web thì dữ liệu trả về sẽ ở dưới dạng một trang HTML để trình duyệt có thể thông dịch được và hiển thị ra cho người dùng.

6.Route Table

|  |  |  |
| --- | --- | --- |
| Route1 | URL pattern 1 | Handler1 |
| Route2 | URL pattern 2 | Handler2 |
| Route3 | URL pattern 3 | Handler3 |

Route Engine sử dụng Route table để match request Url pattern với Url đã được định nghĩa trong route table. Có thể đăng kí một hoặc nhiều URL pattern trong Route table tại Application \_Start event. Khi tìm thấy Url pattern trong route table giống với Request, route engine sẽ chuyển tiếp đến controller và action tương ứng, nếu không thì sẽ trả về 404 Http status code.

7.View hiển thị dữ liệu từ model cho người dùng xem, đồng thời cũng cho phép người dùng modify dữ liệu.Các phương thức của các single controller khác nhau có thể render các view khác nhau, vì vậy thư mục View chứa các folder riêng biệt cho từng controller với tên cùng tên với controller,.

Razor view engine: có thể trộn giữa các thẻ html và code xử lý phía server trong razor view. Razor sử dụng ký tự @ cho code phía server. Có thể sử dụng code C# để viết code server trong razor view. Razor view engine tối đa hóa tốc đọ viết code bằng cách tối tiểu hóa số lượng ký tự required khi viêt 1 view. Razor view thường có đuôi .cshtml

Có thể chuyển model data từ controller đến view bằng nhiều cách như View Bag,ViewData,TempData.

-----------------------------------------------------------------------

**VIEW DATA**: khai báo view data như sau:

Public ViewDataDictionary ViewData {get;set;}

Lớp ViewDataDictionary implements interface IDIctionnary. Vì thế có thể nói rằng ViewData trong ASP.NET là “a weakly typed dictionary object”. Vì là một loại đối tượng từ điển(a dictionay object) nên nó sẽ lấy dữ liệu trong form bằng cặp key-value, mỗi key phải là 1 string và value được chuyển đến dictionary sẽ được store trong form của một object type.

Vì nó store dữ liệu trong một form của một object vì thế trong khi lấy dữ liệu từ View Data type ép kiểu là bắt buộc. Nếu truy cập string data từ View DataDictionay, thì nó sẽ không yêu câu ép kiểu View Data thành kiều string. Nhưng bắt buộc phải ép kiểu nếu truy cập dữ liệu khác với string type từ View Data.

VD:Truyền dữ liệu từ controller: ViewData[“KEY”]=Value. Không cần truyền tham số khi return tức là return View().

Khi nhận ở View :@{var data= ViewData[“KEY”] as ‘type’} dùng as để ép kiểu. Sau đó dùng @ data.”thuộc tính” để lấy dữ liệu

View Data không cung cấp một bộ compile-time error checking. Ví dụ nếu gõ sai Key thì ta sẽ không thấy bất kỳ lỗi compile-time thay vào đó ta biết có lỗi tại runtime

ViewData trong ASP.NET MVC thì chỉ có thể chuyển dữ liệu từ 1 controller action đến 1 view. Điều đó có nghĩa là nó chỉ hợp lệ trong request hiện tại.

**VIEW BAG:** trong ASP.NET MVC là một cơ chế để chuyển dữ liệu từ controller đến view . Nếu đi đến class ControllerBase sẽ thấy khai báo thuộc tính View Bag:

public dynamic ViewBag { get; }

ViewBag đang hoạt động trên kiểu dữ liệu động(“the new dynamic data type”). Lợi ích là ta không cần phải ép kiểu khi truy cập dữ liệu từ ViewBag.

ViewBag là dynamic property cái mà được xử lý khi runtime giống như ViewData. Nó cũng không kiểm tra lỗi trong conpile-time thay vào đó ta sẽ biết lỗi tại runtime

VD: Controller: ViewBag.Data=Value

View : @{var data= ViewBag.Data}. Sau đó @data.”thuộc tính”

Giống và khác giữa ViewData và ViewBag trong asp.net mvc:

* + Có thể truyền dữ liệu bằng cả ViewData và ViewBag
  + ViewData là một dictionary object trong khi ViewBag là dynamic property.
  + ViewData ta sử dụng string như là key để lưu trữ và lấy dữ liệu trong khi đó trong ViewBag ta sử dụng thuộc tính động để lưu trữ và lấy dữ liệu
  + ViewData yêu cầu ép kiểu với dữ liệu có kiểu phức tạp cũng như kiểm tra các giá trị null để tránh xảy ra lỗi trong khi đó ViewBag không yêu cầu bất kỳ loại ép kiểu nào cho loại dữ liệu phức tạp
  + Cả ViewData keys và ViewBag dynamic property đều chỉ được xử lý tại runtime. Do đó, cả 2 đều không cung cấp công cụ kiểm tra lỗi ở compile-time.
  + Vì vậy nếu ta gõ sai tên key hoặc dynamic property names thì ta sẽ không thấy bất kỳ lỗi ở compile-time thay vào đó ta sẽ chỉ nhận thấy lỗi ở runtime. Đây là lý do ViewBag và ViewData ít được sử dụng.

**=>Cách tốt nhất để truyền dữ liệu từ Controller sang View là sử dụng A Strongly Typed Model.**

**STRONG TYPED VIEW:**

Để tạo strong typed view trong ứng dụng ASP.NET MVC , ta cần gửi một object model như một tham số thông qua hàm View() trong controller. Class ControllerBase cung cấp cho ta 4 loại hàm View để truyền dữ liệu từ controller đến View:

//

// Summary:

// Creates a System.Web.Mvc.ViewResult object that renders the specified IView object.

//

// Parameters:

// viewName:

// The view that is rendered to the response.

//

// model:

// The model that is rendered by the view.

//

// Returns:

// The view result.

protected internal ViewResult View(string viewName, object model);

//

// Summary:

// Creates a System.Web.Mvc.ViewResult object using the view name and master-page

// name that renders a view to the response.

//

// Parameters:

// viewName:

// The name of the view that is rendered to the response.

//

// masterName:

// The name of the master page or template to use when the view is rendered.

//

// Returns:

// The view result.

protected internal ViewResult View(string viewName, string masterName);

//

// Summary:

// Creates a System.Web.Mvc.ViewResult object that renders the specified IView object.

//

// Parameters:

// view:

// The view that is rendered to the response.

//

// Returns:

// The view result.

protected internal ViewResult View(IView view);

//

// Summary:

// Creates a System.Web.Mvc.ViewResult object by using the model that renders a

// view to the response.

//

// Parameters:

// model:

// The model that is rendered by the view.

//

// Returns:

// The view result.

protected internal ViewResult View(object model);

Ta sẽ dùng hàm View cuối cùng với input chỉ là object model.

Để tạo strongly typed view trong ASP.Net MVC ta cần chỉ định model view bằng cách sử dụng @model. Sau đó có thể gọi các thuộc tính một cách đơn giản bằng cách sử dụng @Model.”thuộc tính”

Lợi ích: - Strongly Typed View trong ASP.NET MVC cung cấp bộ kiểm tra lỗi ở compile-time như một bộ hỗ trợ thông minh.

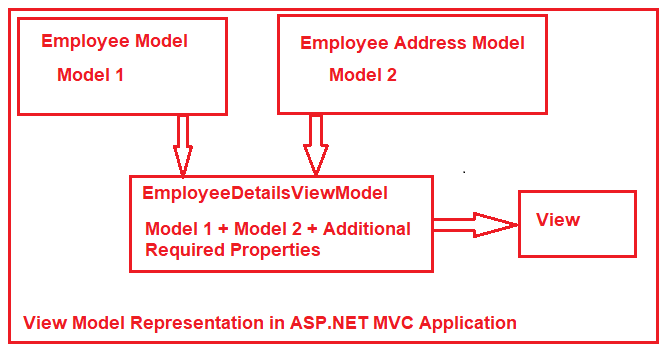
* + Nếu viết sai tên thuộc tính thì nó sẽ biết ở compile-time thay vì runtime.

**VIEWMODEL:**

Trong một ứng dụng ASP.NET MVC, một đối tượng model không thể chưa toàn bộ thông tin cần thiết cho 1 view. VÍ dụ , một view yêu cầu nhiều model data. Vì thế, ta cần đến ViewModel.

ViewModel trong ứng dụng ASP.NET MVC là một model mà chứa nhiều hơn 1 model data được yêu cầu bởi một view cụ thể.

Ta có ví dụ :



Ta cần hiển thị thông tin nhân viên chi tiết trong trang web. Và trong ứng dụng, ta có 2 model chứa dữ liệu nhân viên. Employee model chứa thông tin cơ bản về nhân viên trong khi Employee Address chứa thông tin khác về nhân viên. Cùng với thông tin trong 2 model trên, cần truyền một vài thông tin tĩnh như page header, tiêu đề của View. Để làm được việc đó ta cần tạo EmployeeDetailsViewModel .

Employee(EmployeeID,Name,Gender,Department,Salary,AddressId)

Address(AddressId,Country,State,City,Pin)

Nên tạo ViewModels trong thư mục mới tên là ViewModels

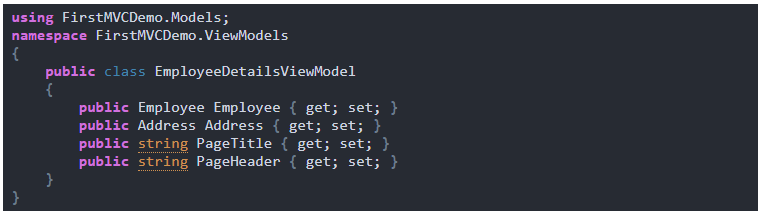
Tên EmployeeDetailsViewModel có nghĩa là:

+ Employee đại diện cho tên Controller

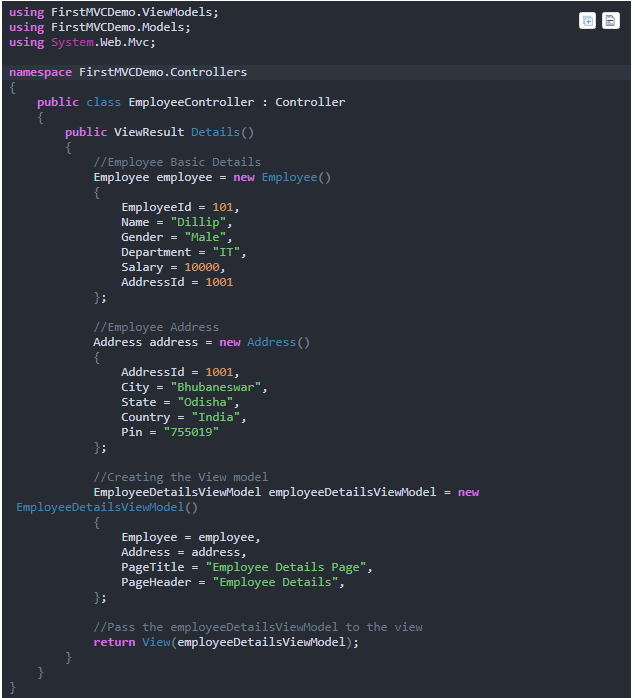
+Details đại diện cho tên action

+Vì nó là view model nên ta thêm hậu tố ViewModel

Cách đặt tên này không bắt buộc nhưng được khuyên nên dùng



Tạo controller



Từ đây có thể truy cập các thuộc tính từ View như trong Strong type view

Trả ra html:

Khi gọi đến hàm protected internal System.Web.Mvc.ViewResult View (object model); trong controller nó sẽ tạo ra đối tượng System.Web.Mvc.ViewResult sử dụng model để render view.

Trong **ViewResult Class** có hàmExecuteResult khi được gọi bởi action invoker, nó render ra view

# HtmlHelper Class(System.Web.Mvc) hỗ trợ render ra các thẻ HTML.

View được thể hiện bởi interface IView(System.Web.Mvc.IView) và được render bằng View Engine thích hợp. Quá trình render được xử lý bằng interface IViewEngine (System.Web.Mvc.IVuewEngine) của view engine. Mặc định, ASP.NET MVC cung cấp WebForm và Razor view engine. Có thể tự tạo engine bằng cách sử dụng IViewEngine interface và đăng ký nó trong ứng dụng ASP.NET MVC như sau:

Html Helpers được sử dụng để viết các trường input, tạo link dựa vào route, Form có AJAX,….Html Helpers là extension methods của lớp HtmlHelper và có thể kế thừa dễ dàng. Ở một trường hợp phức tạp hơn, nó có thể render 1 form với client side validation kết hợp với JavaScript hoặc JQuery