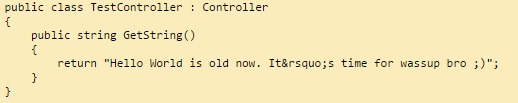
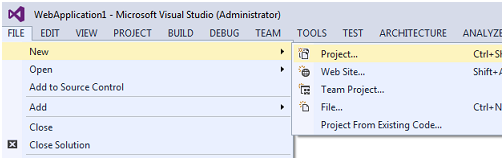
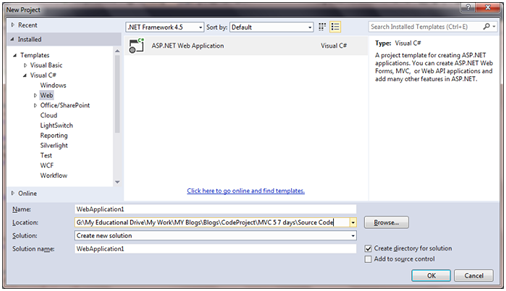
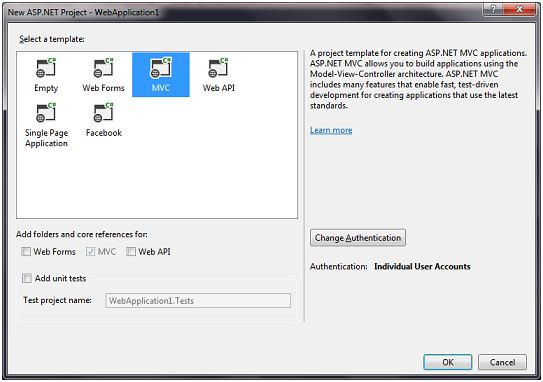
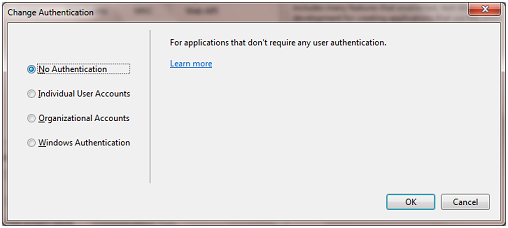
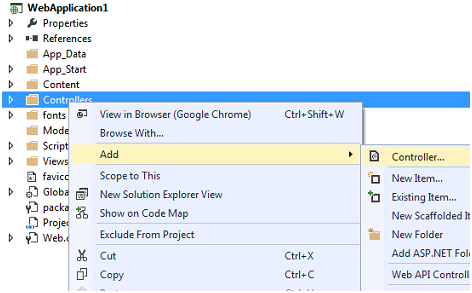
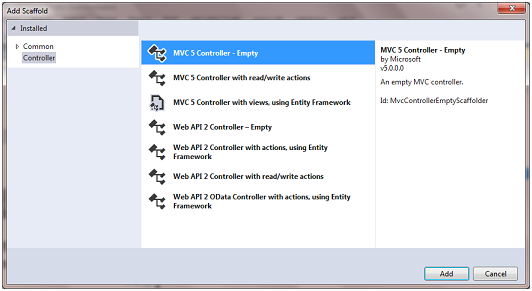
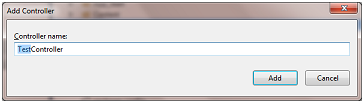
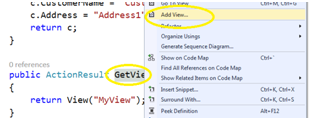
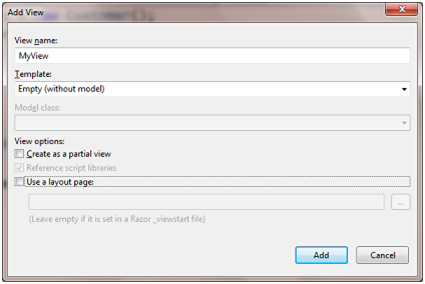
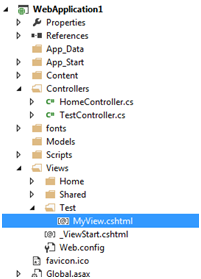
ASP.NET MVC 5 terminologies

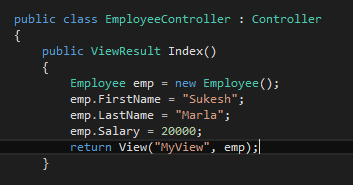
1. **M**: Model, business specific data. It will be created based on Business and Database structure.
2. **V**: View, pure UI. It should not contain any kind of logic
3. **C**: Controller, where logic of application happens
4. **Partial View**: combining with layout page, this one is different content per particular page.
5. **Layout Page**: the master page which make content between page will be consistent
6. **ActionResult**: is the abstract class, which is commonly return value in most of action methods in MVC 5. It has some descendants like below:
   1. ViewResult: renders View to the response by invoking the RenderView method of View.
   2. ContentResult: represents a scalar text response. It’s just like returning pure string
7. **Action Method**: Action method is simply a public method inside controller which accepts user’s request and returns some response. Eg:



1. **ViewData**: object will hold dictionary of data from Controller to View. Controller manipulate and assign its output to ViewData as key/value. This one can be accessed from View via that key and perform rendering.  
   Eg: ViewData[“Student”] = new Student(“12”, “9A”, “FU”);
2. **ViewBag**: is just a syntactic sugar for ViewData. Because ViewBag don’t store data as key/value. It has dynamic binding which affect at runtime when assign a value to a (dynamic) its properties.  
   Eg: ViewBag.Student = new Student(“12”, “9A”, “FU”); // Student is dynamic property here.
3. **ViewModel**: It fits between Model and View and act as data container for View. Controller will create and initializes ViewModel object based on one or more Model object (Specific data will be fetched from Models which use for displaying data in View).
4. **Strongly typed view**: means a view just need only one model to render. See example at section below
5. **Entity Framework**: is Object Relational Mapping tool which create for us database logic layer and mapping database data to object-oriented data or vice versa. See example at section below
6. **Model Binder**: is a feature of ASP.NET MVC:
   1. Model Binder will executes automatically whenever a request is made to an action method containing parameter.
   2. Model binder will iterate though all primitive parameters of a method and then it will compare name of the parameter with each key in the incoming data (Incoming data means either posted data or query string).When match is found, corresponding incoming data will be assigned to the parameter.
   3. After that Model binder will iterate through each and every property of each and every class parameter and compare each property name with each key in incoming data.When match is found, corresponding incoming value will be assigned to the parameter.
   4. See example at section below.

How to? Examples

1. **Create MVC 5 project**
   1. Open Visual studio 2013(or higher). Click on File>>New>>Project.   
      
   2. Select Web Application. Put Name. Put Location and say ok.   
      
   3. Select MVC template   
      
   4. Click on Change Authentication. Select “No Authentication” from “Change Authentication” dialog box and click ok.   
      
   5. Click ok.
2. **Create Controller**
   1. In the solution explorer, right click the controller folder and select Add>>Controller   
      
   2. Select “MVC 5 Controller – Empty” and click Add   
      
   3. Put controller name as “TestController” and click Add.  
        
      One very important point to note at this step is do not delete the word controller. For now you can think it’s like a reserved keyword.   
      
3. **Create View**
   1. Right click the above action method and select “Add View”.  
      
   2. In the “Add View” dialog box put view name as “MyView”, uncheck “use a layout” checkbox and click “Add”.  
        
      Notice:   
      If you want to create a partial view, tick on checkbox: “**Create as a partial view**”  
      If you want to specify this view will use which existed layout, tick on checkbox “**Use a layout page**” and choose your desired layout.
   3. It will add a new view inside “Views/Test” folder in solution explored  
      
4. **Make strongly typed view**
   1. Inside action method of the Controller, pass model through the ViewResult

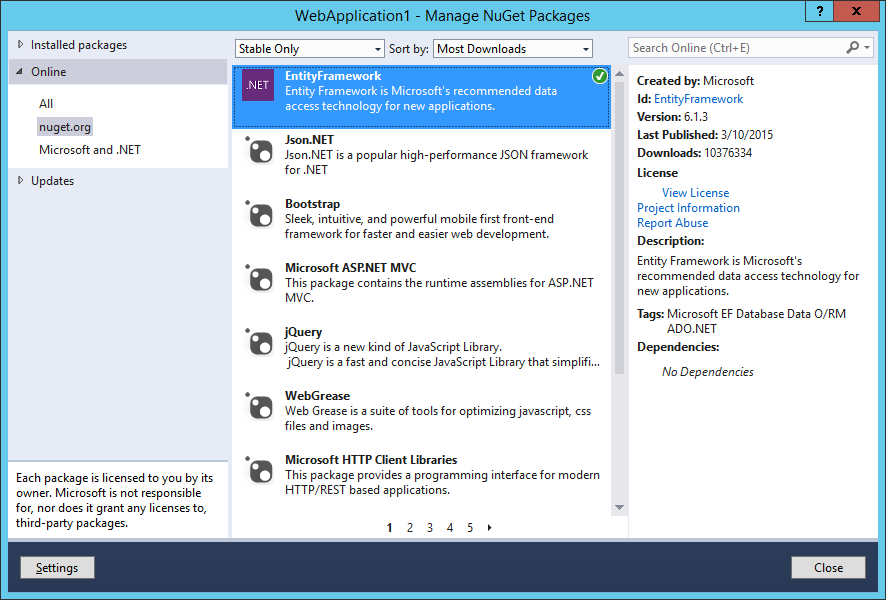
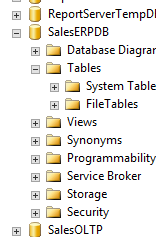


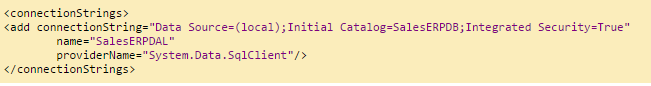
* 1. Using namespace of model and add model in top of the View.



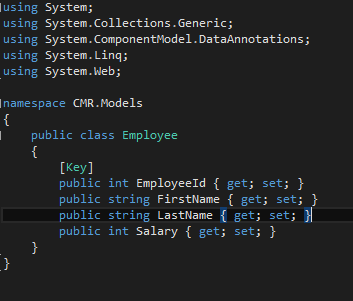
* 1. Inside the View, type model and its attribute at where you want to display value



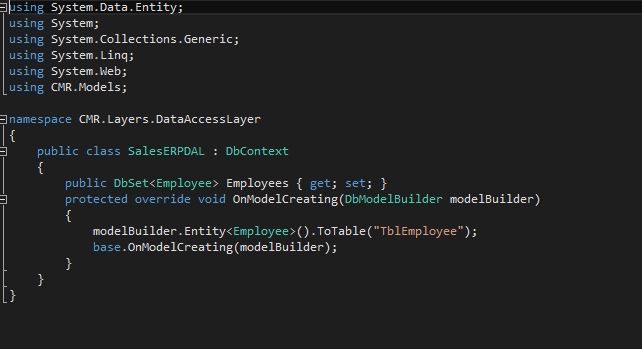
1. **Setup Entity Framework with database**
   1. Assume you are able to know ASP.NET basic (routing/navigating from ActionResult to specific View)
   2. Right click the project >> Manage Nuget packages. Search for Entity Framework and click install.  
      
   3. Connect to the Sql Server and create new Database called “SalesERPDB”.  
      
   4. Create ConnectionString  
      Open Web.config file and inside Configuration section add following section



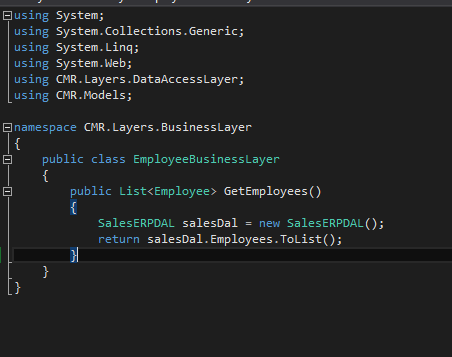
* 1. Create Entity named **Employee** like below (Model)  
       
       
     +) **Key** annotation means this field is primary key in the table



* 1. Create a class named **SalesERPDAL** and its content like below (Model\_DAL)  
       
       
     +) In above code snippet “TblEmployee” represents the table name. It automatically get created in runtime.  
     +) DbSet will represent all the employees that can be queried from the database.  
     +) This class will define mapping between entity and database table

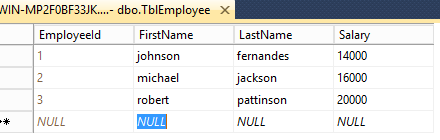
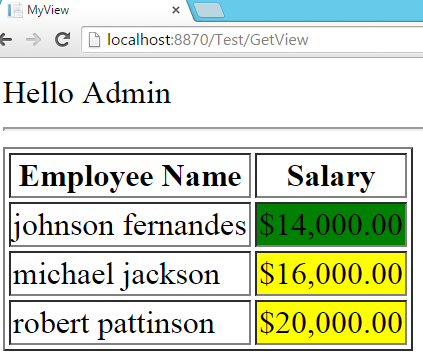


* 1. Create a class named **EmployeeBusinessLayer** and its content like below (Model\_BL)



* 1. Create View like below (View)



* 1. Insert Test Data  
     Add some dummy data to TblEmployee table.  
     
  2. Execute and test the application  
     Here we go ☺  
     

1. **Model Binder example**
   1. At client, I has a file index.html like below



* 1. And at server side, I have action method named SaveEmployee accepting 1 parameter Employee e
  2. **What happened when I click submit from client side?**

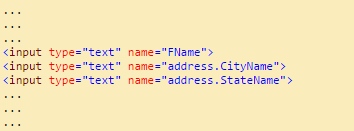
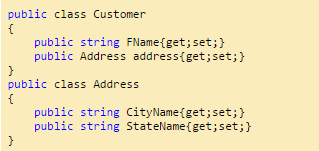


The parameter **e** will be filled FirstName = Chung, LastName = Nguyen, Salary = 10000 by Model Binder hierarchy, it map attribute **name of input** to **properties of Employee** and perform assigning value.

* 1. **What will happen when two parameters are specified, one as “Employee e” and second as “string FirstName”?**  
       
     FirstName will be updated in both primitive FirstName variable and e.FirstName property.
  2. **Will Model Binder workwith composition relationship?**Yes it will, but in that case name of the control should be given accordingly.

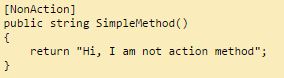
Example

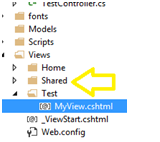
Let say we have Customer class and Address class as follows  
  
  
In this case Html should look like this



Q&A

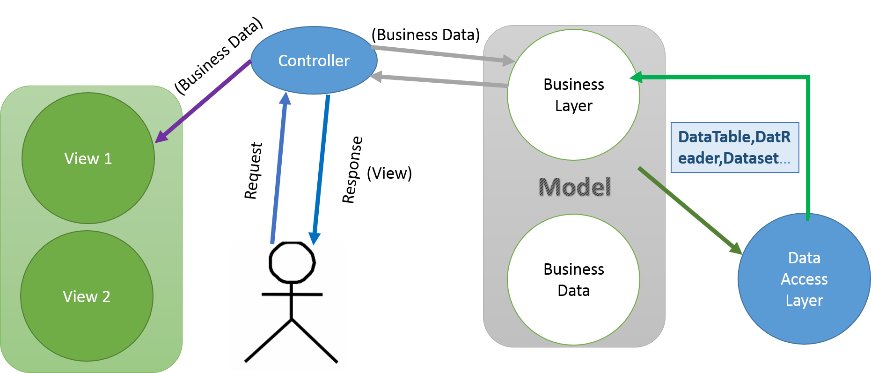
1. **What will happen if we try to return an object from an action method?**  
   When return type is some object like ‘customer’, it will return ‘ToString()’ implementation of that object.By default ‘ToString()’ method returns fully qualified name of the class which is “NameSpace.ClassName”;
2. **Is it must to decorate action methods with public access modifier?**  
   Yes, every public method will become action methods automatically.
3. **What about non-public action methods?**  
   They are simply methods of a class and not available publicly . In simple words these methods can not be invoked from the web.
4. **What if we want a method to be public but not action method?**  
   Just add annotation **[NonAction]** above of method.



1. **Can’t we reuse some of the views across multiple controllers?**  
   Yes, we can. For that we will keep those files inside a special folder called “Shared”.  
   

Views located inside this Shared folder will be available to all the controllers.

1. **What is the difference between writing Razor code with brace brackets (that is “{“ and “}”) and without brace brackets?**  
   +) @ with brace brackets: means you can put some code snippets on it  
   +) @ without brace brackets: simply display the value of variable or expression.
2. **What about the Database Logic/ Data Access Layer and Business Layer?**  
   +) Data Access Layer is one of the unspoken layer in Asp.Net MVC. It’s always there but never included in MVC definition.

+) Business layer is a part of Model.  


1. **Difference between Model and ViewModel?**

Model is Business specific data. It will be created based on Business and Database structure. ViewModel is View specific data. It will be created based on the View.

1. **How View and ViewModel will be connected?**

View is going to be a strongly typed view of type ViewModel.

1. **How Model and ViewModel will be connected?**

Model and ViewModel should be independent of each other. Controller will create and initialises ViewModel object based on one or more Model object.

1. **Every model will have one View Model?**

No, Every View will have its corresponding ViewModel.

1. **Is it a good practice to have some relationship between Model and ViewModel?**

No, as a best practice Model and ViewModel should be independent to each other.

1. **Can we make View a strongly typed view of List?**

Yes, we can.

1. **Why didn’t we made View a strongly typed view of type List<T>? (T means any type)?**  
   If we use List<T> directly instead of T then there will be two problems.

Managing future presentation logic.

Particular property. Eg: UserName (Logging User) is not associated with individual employees (Data user want to see). It is associated with complete View.

References

This document is brief of some very first actions needed to setup environment. For full guidance, I recommend you take a look at for further actions in ASP.NET MVC.  
<http://www.codeproject.com/Articles/866143/Learn-MVC-step-by-step-in-days-Day>

Class structure

1. **App\_Start**: Contains all config about filters, routes
2. **Content**: Contains css/image file
3. **Controller**: Contains all controllers
4. **Layers**: Contains BL and DAL layers
5. **Scripts**: Contains all javascript files
6. **ViewModels**: Contains all ViewModel classes
7. **Views**: Contains all Views  
   Notice:   
   +) Name of folder must match with name of controller:  
   Eg: TestController will map with views in **Views/Test/** folder  
   +) Shared folder contains all views which can used by any controller