

# Server side Web Development

Lecture 1

Introduction, ASP.NET MVC

# Outline

- Introduction to course
  - The course content
  - Teaching and examination
  - Lab & Project
- About the web architecture
- Introduction to ASP.NET MVC

# Course contents from learning outcomes

- Web programming architectures, patterns, and tools
  - Web Forms, MVC
- Application servers and frameworks
  - IIS, .NET, node
- Current web technologies
  - ASP.NET, C#, JavaScript
- Security concerns in web applications
  - User management, SQL injections
- Server-side AJAX
  - Not really, REST
- Testing, debugging, and optimizing web applications
  - Yes

# Ping-pong

- Code for activating: TPWK16S1703
- How to:
  - 1. Find the course homepage: <http://pingpong.hj.se/>
  - 2. Use your Novell login to login to ping-pong
  - 3. Select Catalogue in Events
  - 4. In the list find " Server-side Web Development - TPWK16 - S17"
  - 5. Click **start** at the **right side**
  - 6. Use password: **TPWK16S1703** to activate the ping-pong activity

# Information documents

- Syllabus
- Course guide
- Lab work instructions

# Teachers

- Course coordinator: Anders Carstensen
  - Room E2420, Tel: 036-101586, email: [anders.carstensen@ju.se](mailto:anders.carstensen@ju.se)
- Additional teacher: Peter Larsson-Green
  - Room: - , Tel: 036-101735, email: [peter.larsson-green@ju.se](mailto:peter.larsson-green@ju.se)
- Additional lab assistant: Magnus Schoultz
  - Room: E3421, Tel: 036-101579, email: [magnus.schoultz@ju.se](mailto:magnus.schoultz@ju.se)
- ...and PLEASE DON'T use PIM use email

# Lectures

- Introduction to course and to ASP.NET Model View Controller (MVC)
- More about ASP.NET MVC
- Aspects on system development, testing, and deployment environments for web applications
- Data access the traditional way
- Data access using Entity Framework
- Security and user management - two lectures
- API for web applications
- Applications and frameworks

# Lab / Project work

- An introductory lab
  - A starting exercise
- Lab work with a common theme
  - Library case
  - ASP.NET-MVC



# Examination

- Lab and project work 4,5 credits
  - Short informal demonstration
  - Report, upload as pdf-file to pingpong
  - Code, upload as zip-file to pingpong
  - Peer assessment
- Written exam 4,5 credits
  - Pingpong exam 3hrs

# Course literature.

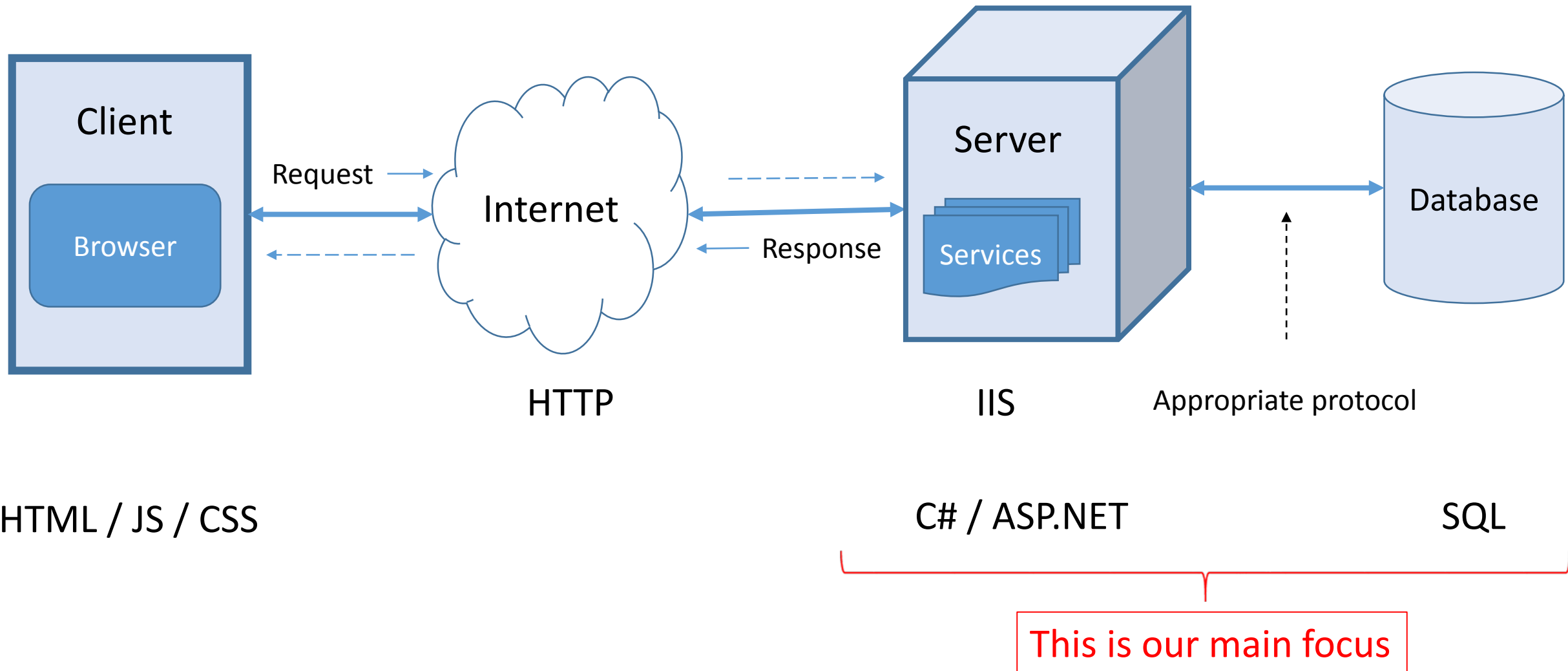
- Freeman, Adam
  - Pro ASP.NET MVC 5
  - ISBN: 978-1-4302-6529-0

Available as full-text from the library.

In addition there will be some links to read.

# Introduction to ASP.NET MVC

# Web architecture - Client / Server



# Web architecture – Basic server workflow

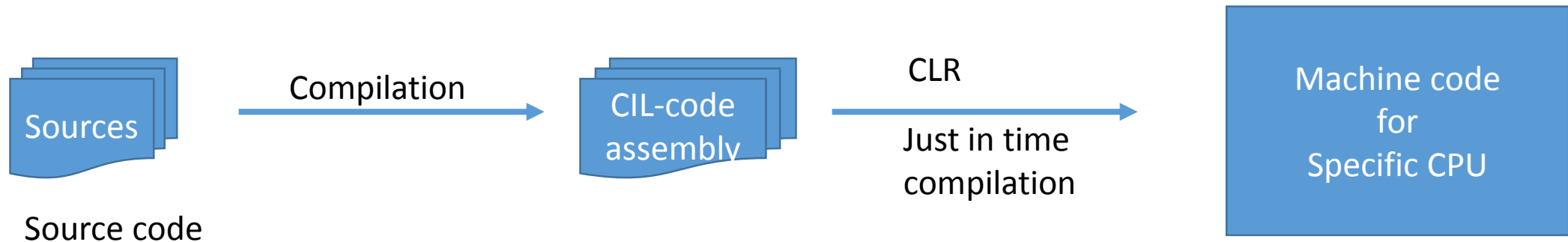
1. Wait for client request
2. On request
  - a. Route the request to the correct resource handler  
This is where we tap in
  - b. Get response from resource handler
  - c. Return response to client
3. Restart on 1

# ASP.NET

- ASP stand for **Active Server Pages**
- Intended for developing **dynamic web pages**
- Uses non-scripting languages
- Object oriented
  - Components
  - Event based

# ASP.NET - .NET

- Development framework supported by several languages, e.g.
  - C#
  - VB.NET
- Code is compiled to a Common Intermediate Language, CIL-code
  - Independent of source language
  - Independent of CPU



# ASP.NET

- Encourages reusability:
  - Rich set of components
  - Separates presentation and logic (Architectural style)
- Supports different technologies/design patterns:
  - Web forms
  - Model View Controller (MVC)

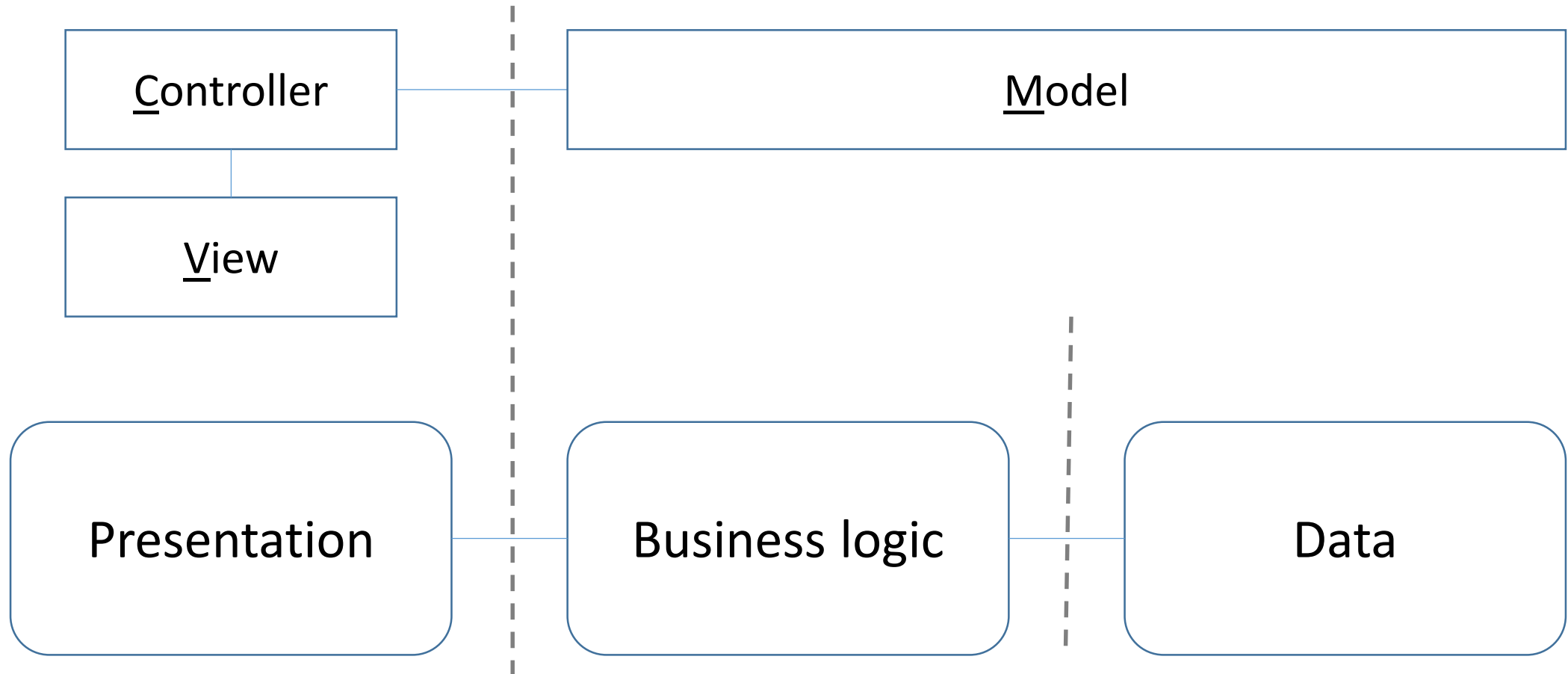


# Application layers

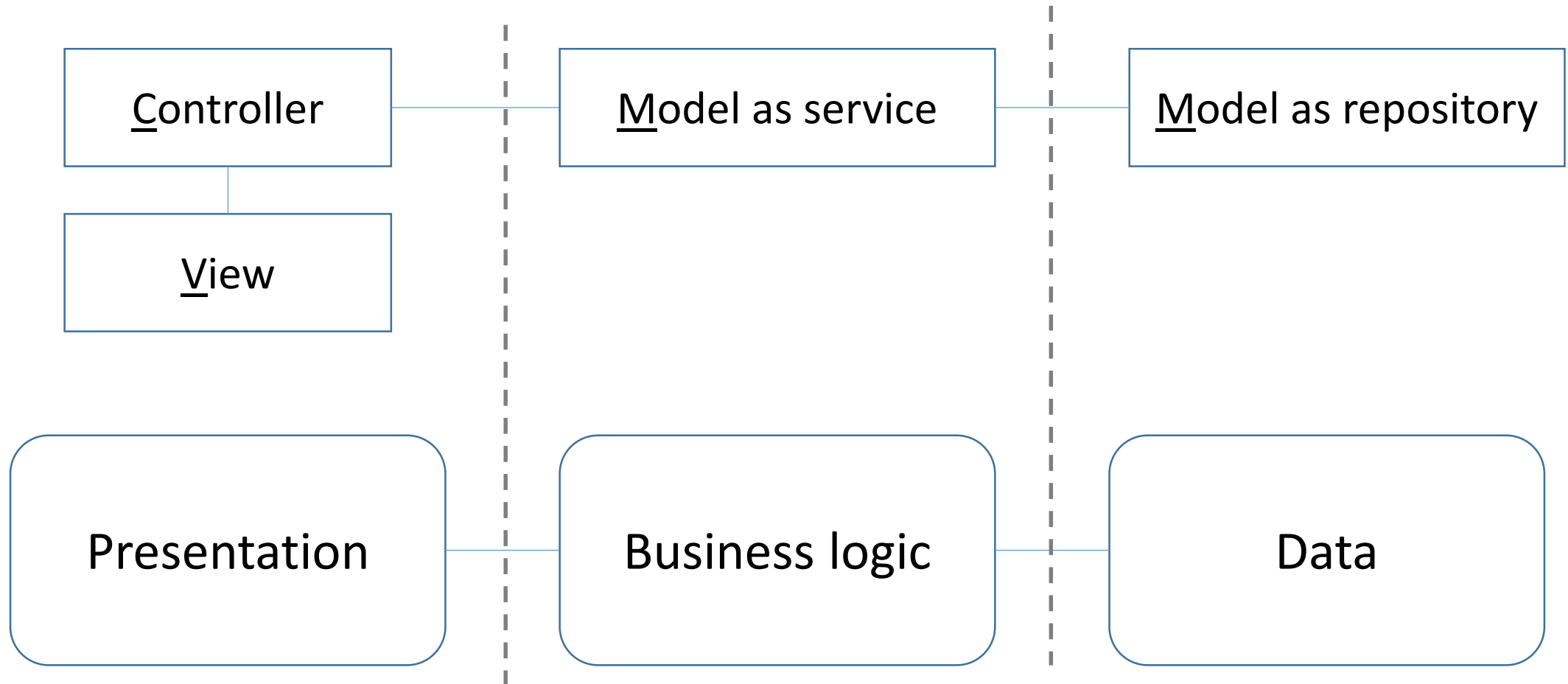


- Presentation: the user interface
- Business logic: rules for managing the application context
- Data: persistent storage of data, often in database

# Applying the MVC architecture – 2 layers



# Applying the MVC architecture – 3 layers



# Request/Response using MVC architecture

1 Client sends request, ***routed*** to an ***action method*** in a controller class

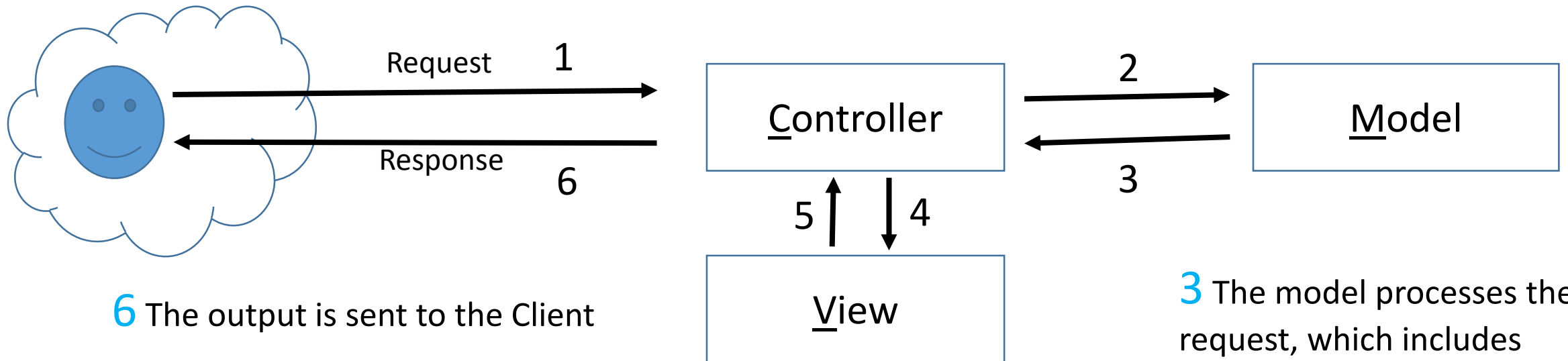
2 The controller calls methods in the model classes, implementing the business logic

3 The model processes the request, which includes fetching data. Then sends back the result to the controller.

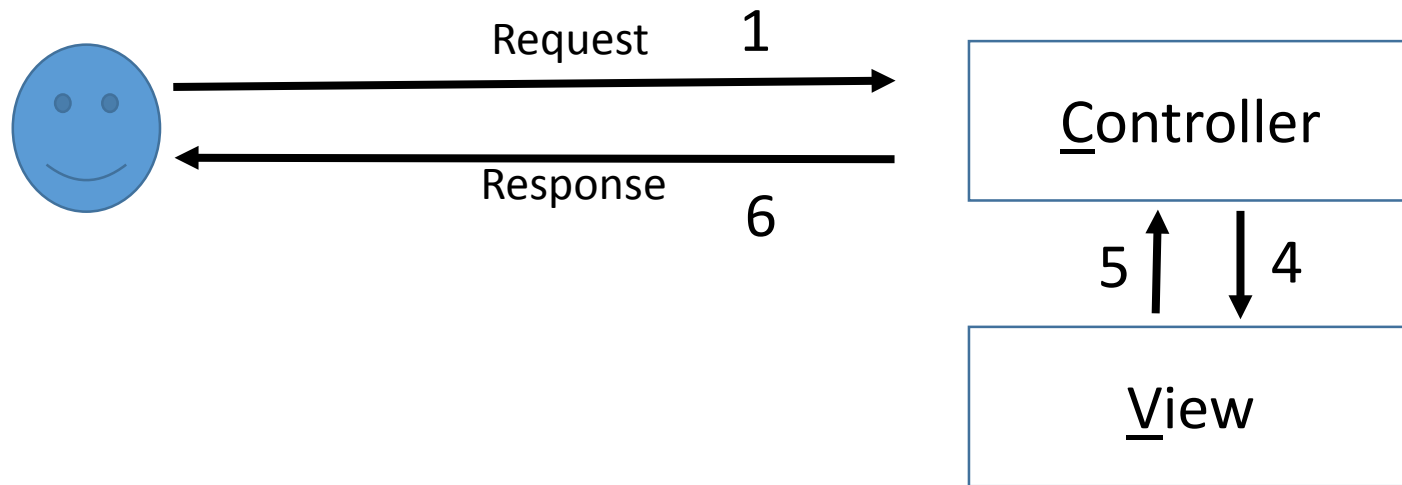
4 The controller sends the result to the appropriate view

5 The view prepares the output by mixing static and dynamic HTML

6 The output is sent to the Client



# Controller and view – our first focus



# Creating a MVC project in Visual Studio (VS)

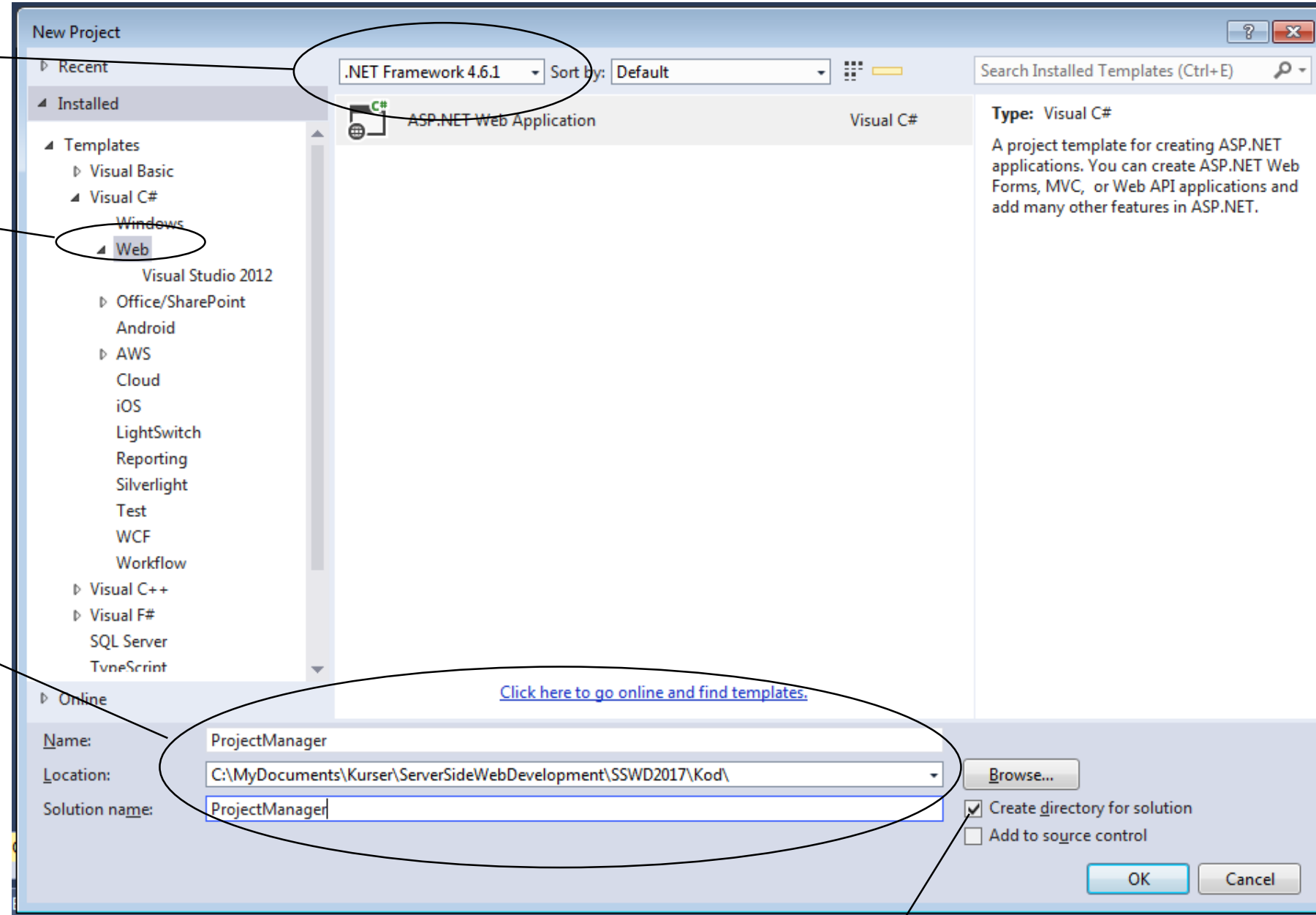
- We will use VS 2013 in this course
- The next four slides will show how to create a MS MVC project
- The steps will be:
  - File -> New -> Project
  - (In C# template) select “Web” and “ASP.NET Web Application”
    - (If we are creating a project from scratch) Mark “Create directory for solution”
    - Fill in name of project and name of solution
  - In window “New ASP.NET Project - <Name of project>”
    - Select template “Empty”
    - For configuring check MVC

Select latest version

Select C# - "Web"

Fill in:  
Name of project  
Suitable storage location  
Name of solution space

If a new project:  
Check the checkbox

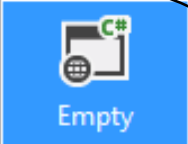







Select template  
"Empty"

Check the "MVC"  
configuration

New ASP.NET Project - ProjectManager

Select a template:

 Empty  Web Forms  MVC  Web API

 Single Page Application  Facebook

Add folders and core references for:

☐ Web Forms ☒ MVC ☐ Web API

☐ Add unit tests

Test project name:

An empty project template for creating ASP.NET applications. This template does not have any content in it.

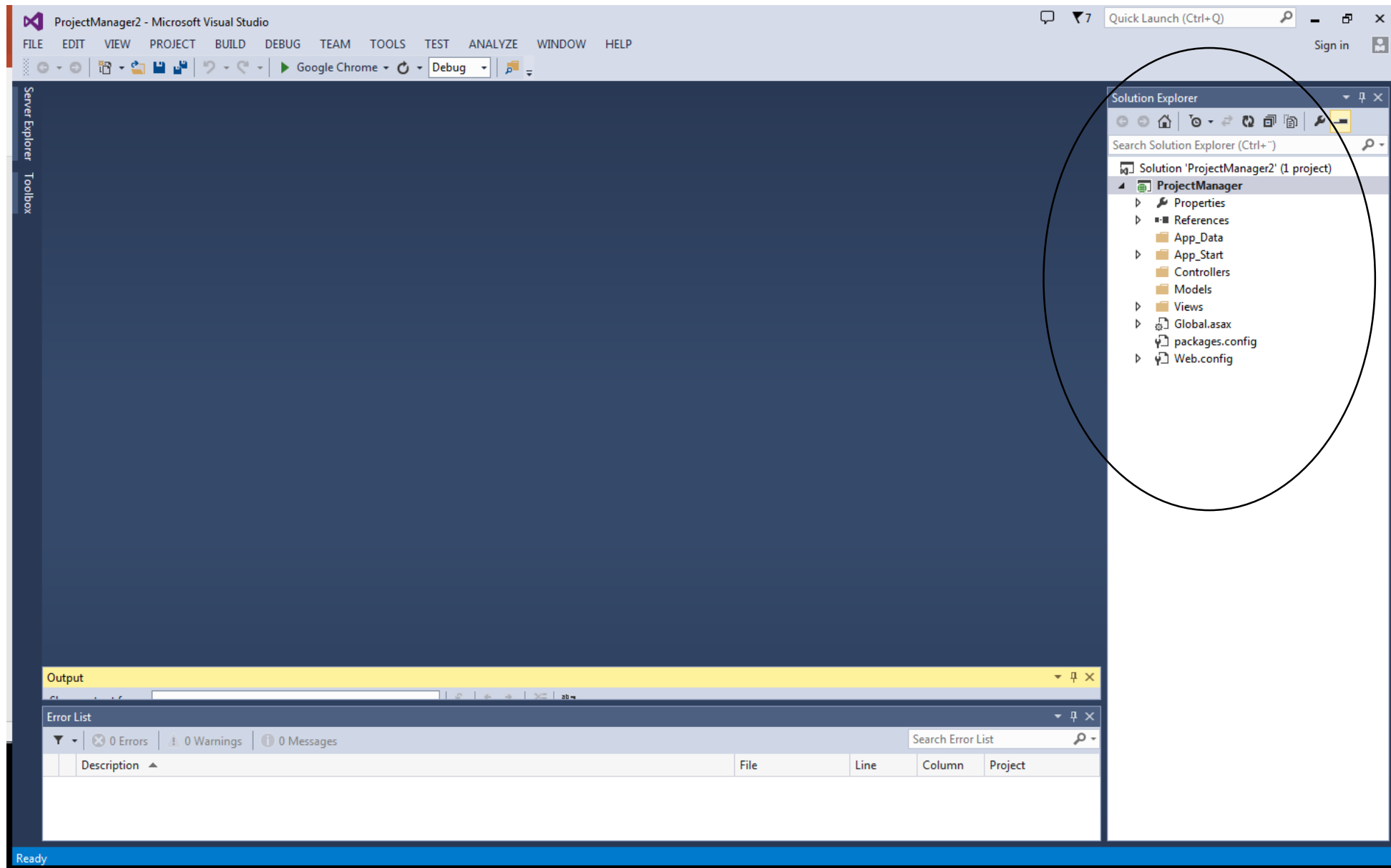
[Learn more](#)

Change Authentication

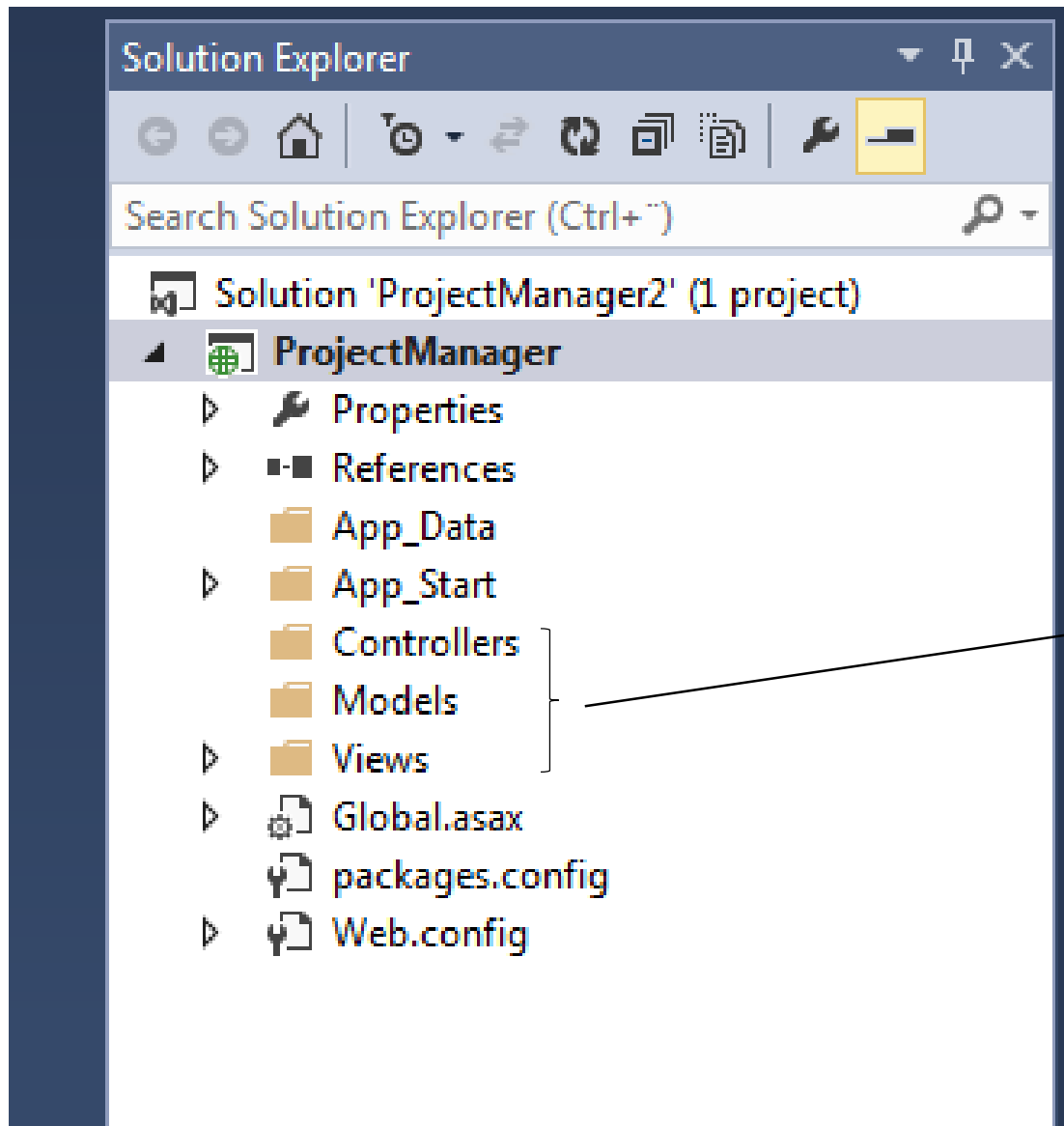
Authentication: **No Authentication**

OK Cancel





On next slide



Folder for controller classes with action methods

Folder for model classes, later on we will use separate project for this instead

Folder for creating views

# The controller

- Consists of one or several controller classes
  - Inherits from `System.Web.Mvc.Controller`
- The controller receives the request
  - Invokes the model for processing the request
  - Selects the appropriate view for preparing the response
  - Passes the response to the client
- The controller has properties for storing information about the current request managed by the controller
  - E.g. `HttpContext`, `Request`, `Response`, `RouteData`, `Session`, `TempData`
- A controller class contains one or several action-methods

# The controller – action method

- An action-method contains the code for
  - Calling methods in model-classes
  - Passing results to views
- An action-method is invoked through a routing process
  - The routing process is invoked by a URL passed from the client, e.g.

`http://www.server.com/Home/Index`

The diagram illustrates the components of the URL `http://www.server.com/Home/Index`. Brackets are placed under each part of the URL: `http://` is bracketed and labeled 'Protocol'; `www.server.com` is bracketed and labeled 'Server'; `Home` is bracketed and labeled 'Controller' (with an upward-pointing arrow from the label below); and `Index` is bracketed and labeled 'Action'.

Protocol      Server      Controller      Action

# Action-method example

The controller configured in VS to be default

Class in namespace System.Web.Mvc

```
public class HomeController : Controller
{
    //
    // GET: /Home/
    public ActionResult Index()
    {
        return View();
    }
}
```

The action method created by default

The returnvalue from the action method

The return clause calls a view with the same name as the action-method. The view process the HTML-code which is returned as a response to the client

# ActionResult

- The return value from the action method is an object of the class ActionResult
  - This is not directly the response object sent to the client
  - It describes what the response will be
  - The MVC Framework will process the ActionResult object and produce the response
- There are several sub-classes which describes more specifically the intended response
  - The controller class has helper methods that render these more specific ActionResult objects

Class	Controller method	Description
ViewResult	View	HTML from specified or default view (.cshtml)
PartialViewResult	PartialView	HTML defined in a partial view
RedirectResult	Redirect	Redirect to a URL
RedirectToRouteResult	RedirectToRoute	Redirects to a URL, generated by the routing system
FileResult	File	Returns file-data directly into the response to the browser
ContentResult	Content	Returns a text-string into the response to the browser
JsonResult	Json	Serializes a .NET object in Json format into the response
EmptyResult	null	Does not return anything

# Action-method example

```
public class HomeController : Controller
{
    //
    // GET: /Home/
    public ActionResult Index()
    {
        return View();
    }
}
```

Since the View-method returns a result of type ActionResult it makes sense to change the return-value of the action-method to ActionResult

# The View

- Is where HTML-code is written
- The View-file is given the extension .cshtml (or .vbhtml)
- Most views belong to a certain controller
  - Placed in the Views\<controller name> - folder
- Some views may be shared by several controllers
  - Placed in Views\Shared – folder
- Can be full views or partial views
  - A full view has all parts of an HTML page
  - A partial view has parts of HTML and is included in a full view or another partial view



# View - example

```
@{  
    Layout = null;  
}
```

Most of it looks like an ordinary empty HTML page.

The first part beginning with @ is different and is example of *razor syntax* for writing C# - code (or alternatively VB-code)

```
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    <div>  
    </div>  
</body>  
</html>
```

# Call on the view

- When an action-method calls a view it can do this in two ways:
  - Let the routing system find the view with the same name as the action- method has (locating by naming convention)
  - Explicitly call a view by passing the name of the view as a parameter to the View method

# The View – adding HTML dynamically

- A dynamic response is the main purpose
  - Dynamic HTML-code is created with the help of ***razor syntax***
  - Razor syntax let us use C#-code (or VB-code) to manipulate the HTML content
- It is the controllers job to collect data which then is passed to the view
  - An action-method may pass data to a view using
    - A viewbag
    - A model object passed as a parameter of a view-method
- A view (used by a client) may pass information to an action-method
  - Using a form, submitted either as GET or POST
  - Using an ActionLink - helper method provided in the MVC-framework - GET

# ViewBag

- A data dictionary to which a property and its data can be added dynamically
- Share's data between controller method and its view
  - It can only store data during the current HTTP-request
- Example: Controller method -----> View

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Greeting = "Hello world!";
        return View();
    }
}
```

```
@{
    Layout = null;
}
...
<body>
    <div>
        @ViewBag.Greeting
    </div>
</body>
```

# ViewBag

- In the controller method

```
ViewBag.<property> = <Data object>;
```

- In the associated view

```
@ViewBag.<property>[.<Part of data object>]
```

# ActionLink

- Html - helper method for creating an <a> - tag
- Has several overloaded methods
- Common usage

```
@Html.ActionLink(<Displayed text>,<action-method>,<controller>,  
    <Object with GET parameters>,<additional html attributes>)
```

# ActionLink

- Example

```
@Html.ActionLink("List departments", "ListDepartments","Home",new { id = "1" },null);
```

Will correspond to

```
<a href="/Home/ListDepartments?id=1">List departments</a>
```

And generate a URL

```
http://<Default host>/Home/ListDepartments/1
```

or

```
http://<Default host>/Home/ListDepartments?id=1
```

# MVC - Model

- Creates the ***Domain Model*** in the application
  - A set of classes
  - Represents the data
- Provides the logic for managing data from storage
- Storage is not part of the framework
  - Usually provided by a database



# Razor code

- Is Written in the View
- Is only ment to unfold the dynamic data content into the view
- Is NOT ment to perform business logic or manipulate the domain model
- All razor code begins with the @ character

# Razor code - ViewBag

- A viewbag is a global object
- Properties may be added on the fly in code
- Transfer object value from controller method to a corresponding view
- In the view access the viewbag through razor code
  - @ViewBag.propertyName

# Razor code – The ordinary construction elements

- Selection
  - @switch
  - @if
- Iteration
  - @while
  - @for
  - @foreach

# Razor code – the razor code block

- Allows to include C# statements in a view
- The razor code block is delimited by @{ and }
- The statements inside the code block are evaluated when the View is rendered
  - C# - code need not be preceded with the @-character inside a code block
- An "empty" View will initially have

```
@{  
    Layout = null;  
}
```

# Razor - @model

- Declares the type of ***model object*** passed to the View
  - The view needs to be strongly typed
- Allows the View to use the methods, fields and properties of the model object
- E.g. A domain model *Product* with a property *Name* can be accessed in the razor code in a View by

@Model.Name

- Notice when accessing it a capital M in Model is used

# Creating the domain model

- In the Models folder add Classes for representing the data
- Mostly these classes consist of properties and methods
- E.g. an Employee - class

```
public class Employee
{
    public int empid { get; set; }
    public string name { get; set; }
    public int depid { get; set; }
    public int salary { get; set; }
}
```

# Creating the domain model

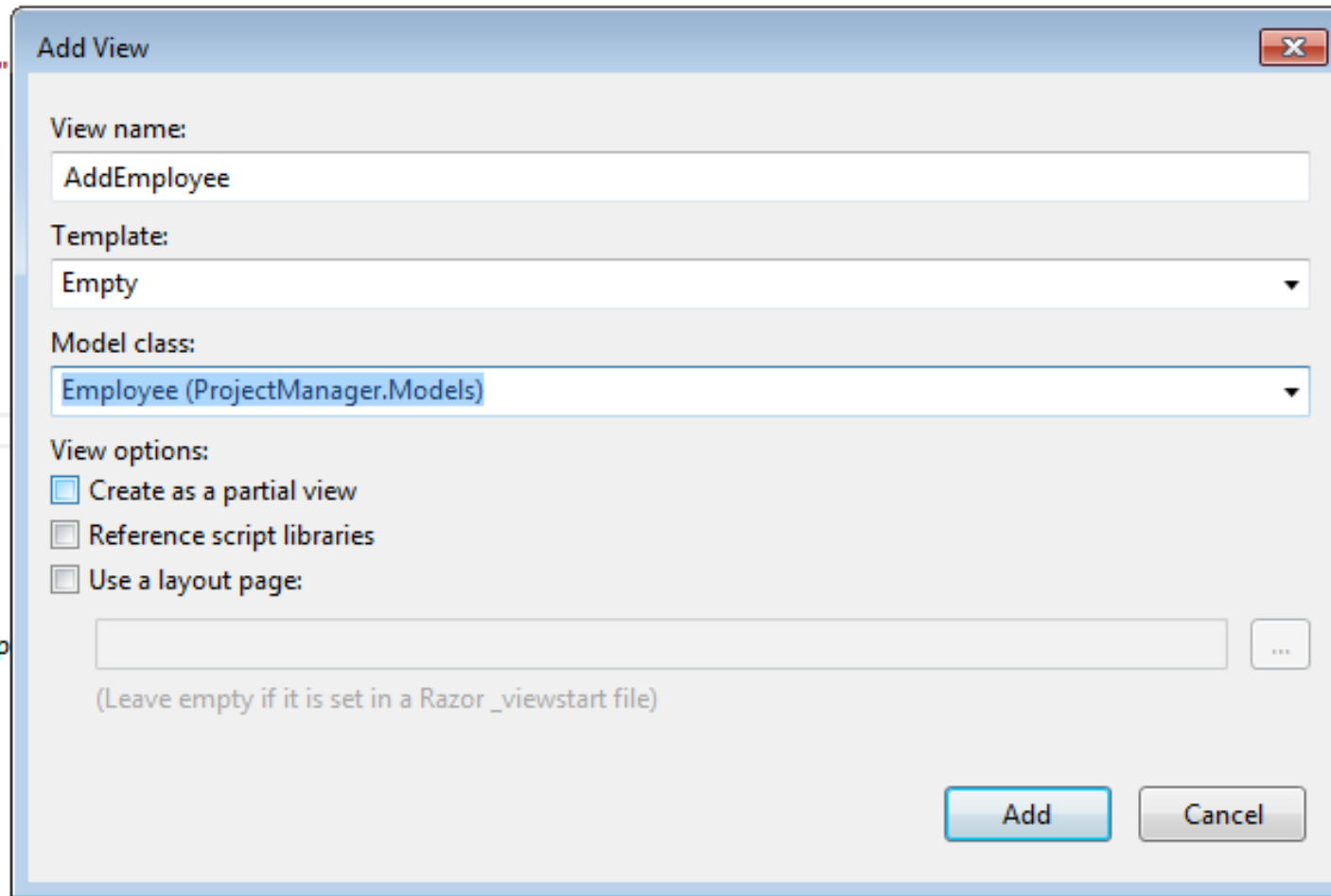
- One important concern for the MVC framework is to separate Presentation, Business logic and Data storage
- This is done by introducing several layers (Service and Repository)
  - Often placing these layers in separate projects
  - We will work more with this in coming lectures
- In the initial phase of development it can be useful to do a mockup
  - i.e. to fake data by
  - Adding classes (perhaps not fully outlined) in the Models folder
  - Instantiate objects from these classes and hard-code some data
  - This way you may still work with coding the presentation layer without dealing with the complexity of the database connection

# Strongly typed views

- The views need to now how data is represented in different objects
- A strongly typed view uses objects defined by specific class in the models folder
- An action-method associated to this view may pass objects of the class to the view
- The view uses razor syntax to unfold the data from the passed object
- The model object may be also be used to pass data from the view to the controllers action method



# Creating a strongly typed view



The screenshot shows the 'Add View' dialog box with the following configuration:

- View name:** AddEmployee
- Template:** Empty
- Model class:** Employee (ProjectManager.Models)
- View options:**
  - ☒ Create as a partial view
  - ☐ Reference script libraries
  - ☐ Use a layout page:

At the bottom, there is an empty text box for a layout page and an 'Add' button. A note below the text box says: (Leave empty if it is set in a Razor \_viewstart file).

First add classes to Models folder

Build the solution

Then when adding a view:

- Change Template to "Empty"
- Select the appropriate Models class

# Creating a strongly typed view

```
@model ProjectManager.Models.Employee
```

```
@{  
    Layout = null;  
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <meta name="viewport" content="width=device-width" />
```

```
    <title>ListEmployees</title>
```

```
</head>
```

```
<body>
```

```
    <div>
```

```
    </div>
```

```
</body>
```

```
</html>
```

The first row in the strongly typed view shows the model binding

# Form handling

MVC includes some helpers for our formhandling:

- `BeginForm()` - creates the HTML form
- `TextBox()` - creates an input type text
- `Password()` - creates an input type password
- `RadioButton()` - creates a radiobutton
- ... for any HTML form input element

# Form Handling

*@Html.BeginForm("ActionName", "ControllerName", [Method], [RouteValue])*

Use @Html.EndForm() to end the form element

or put the BeginForm in a using statement:

```
@using(Html.BeginForm("Create", "Author")){  
    //The form  
}
```

It is possible to mix traditional HTML coding with MVC helper methods

```
@model ProjectManager.Models.Employee
```

```
...
```

```
<body>
```

```
<div>
```

```
<h1>Add new employee</h1>
```

```
@using(Html.BeginForm())
```

```
{
```

```
@:Employee id: <input type="text" name="empid" value="@Model.empid" />
```

```
@:Name: <input type="text" name="name" value="@Model.name" />
```

```
@:Department id: <input type="text" name="depid" value="@Model.depid" />
```

```
@:Salary: @Html.TextBox("salary",Model.salary)
```

```
<input type="submit" value="Add"/>
```

```
}
```

```
</div>
```

```
</body>
```

```
</html>
```

A traditional <input>

Plain HTML-text follows

Html helper method for a  
textbox

# Form handling

- The form will by default use a POST message for sending the information
- To capture the posted information an action-method with the same name as the method calling the view should be created
  - The method should be preceded with the attribute [HttpPost]

# Form handling

```
public ActionResult AddEmployee()  
{  
    Employee empObj = new Employee();  
    return View(empObj);  
}
```

By default the action method is designed for handling a GET request. It is possible to add the attribute [HttpGet], but not necessary

```
[HttpPost]  
public ActionResult AddEmployee(Employee empObj)  
{  
    //Some code for storing the data  
    return View("ListEmployees", empObj);  
}
```

Add the attribute [HttpPost] to capture the POSTED information from the form

# Reading

- Pro ASP.NET MVC 5
  - Chapter 1 – Putting ASP.NET in context
  - Chapter 2 – Your First MVC application
- Links:
  - <http://rachelappel.com/2014/01/02/when-to-use-viewbag-viewdata-or-tempdata-in-asp-net-mvc-3-applications/>
  - <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>