Web Application development using ASP.NET MVC

Education, Training and Assessment

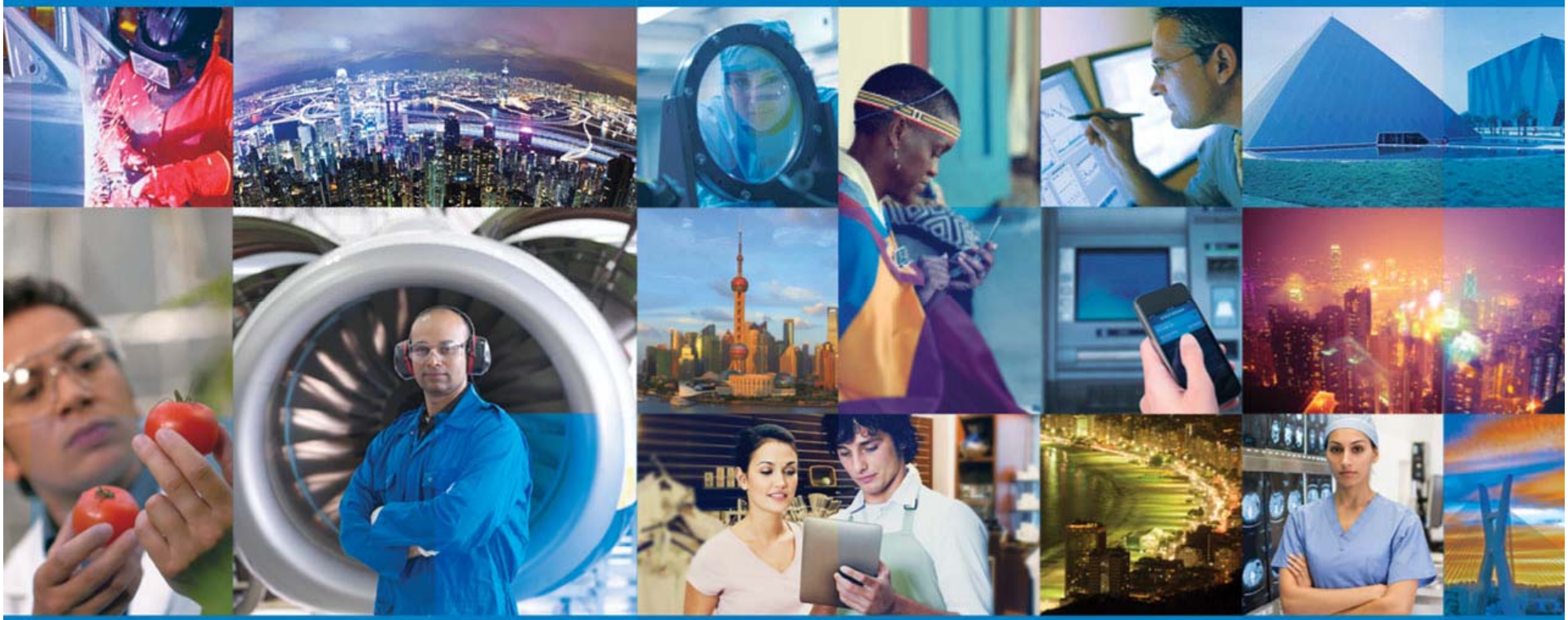
We enable you to leverage knowledge anytime, anywhere!

Usage Guidelines

Do not forward this document to any non-Infosys mail ID. Forwarding this document to a non-Infosys mail ID may lead to disciplinary action against you, including termination of employment.

Contents of this material cannot be used in any other internal or external document without explicit permission from ETA@infosys.com.

Web Application development using ASP.NET MVC



Education, Training and Assessment

We enable you to leverage knowledge anytime, anywhere!

Infosys® | Building Tomorrow's Enterprise

Course Information

Course Code: TMi2437

Course Name: Web Application development using ASP.NET MVC

Document Number:

Version Number: 1.0

Copyright Guideline

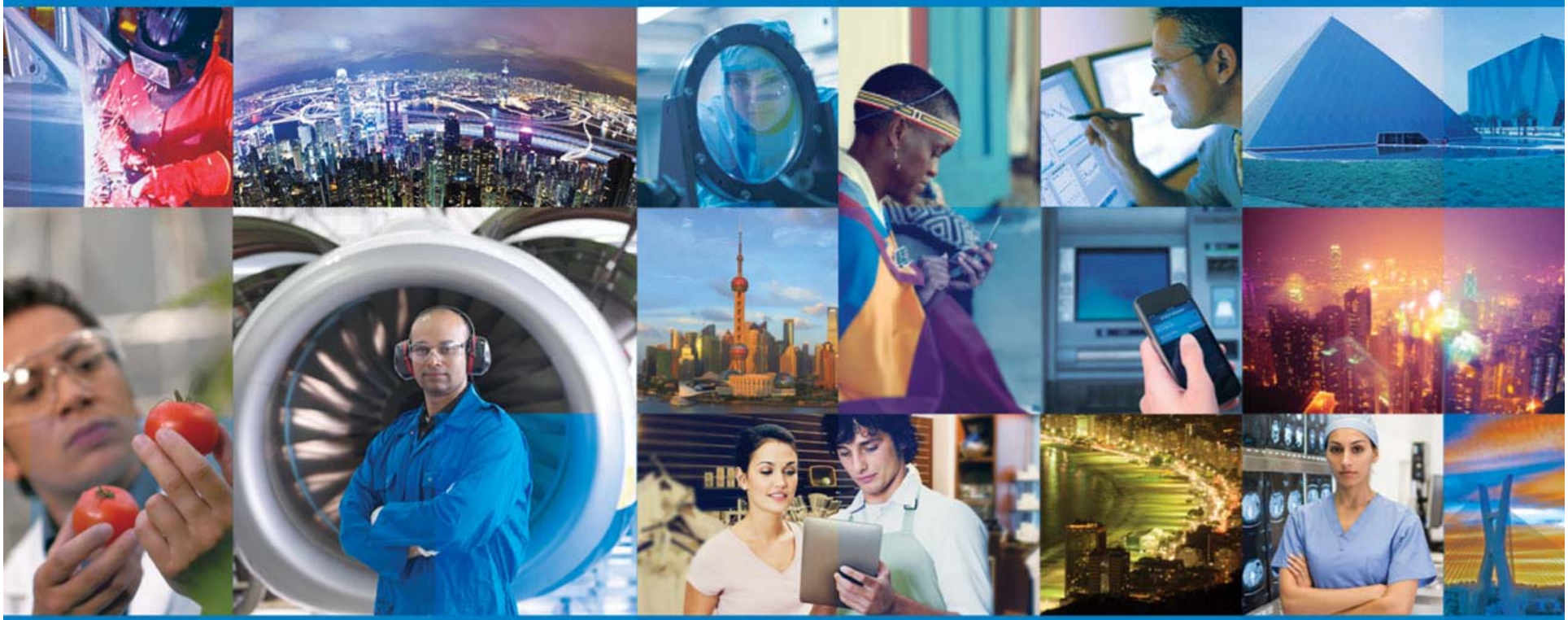
Session Plan (1/2)

- Introduction to ASP.NET MVC Framework
- Working with Controller
- Introduction to Routing
- Creating Views
- Introduction to Razor Engine
- Working with helper classes
- Working with Action Filters
- Introduction to Model
- Introducing data validation in Model

Session Plan (2/2)

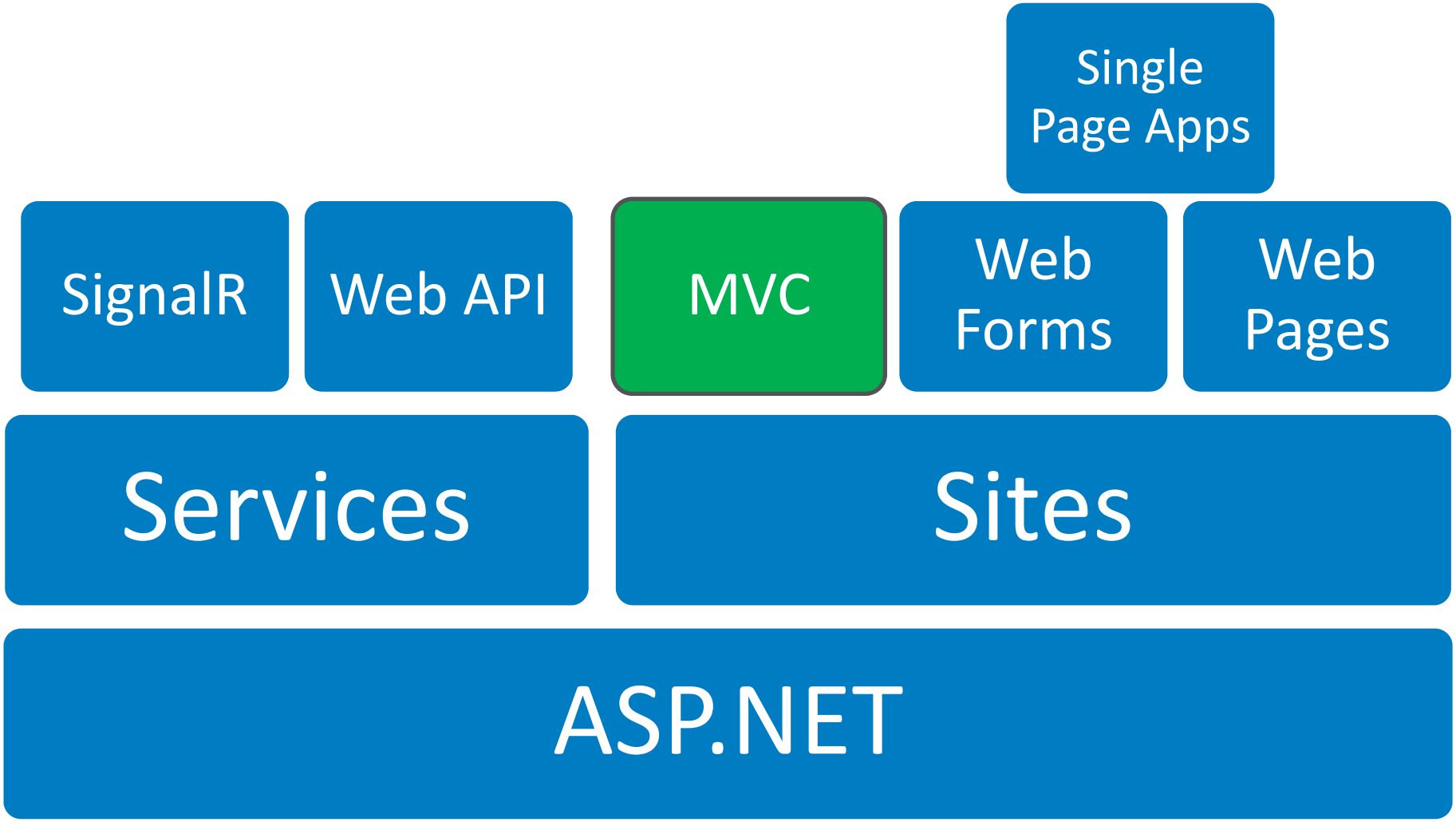
- Deploying ASP.NET MVC applications
- Unit testing MVC applications
- Web Optimization

Introduction to ASP.NET MVC



Infosys® | Building Tomorrow's Enterprise

ASP.NET Framework



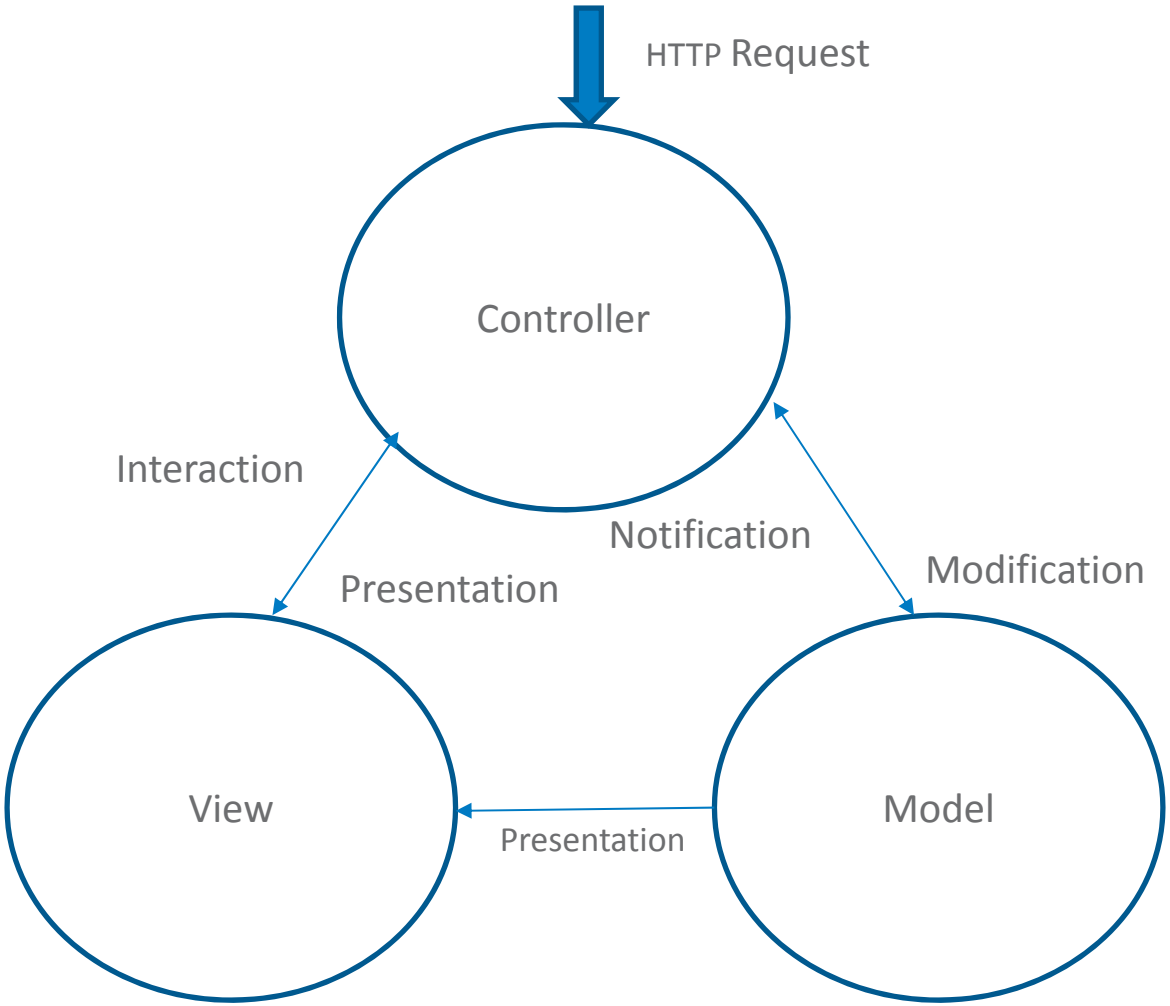
Introduction to Asp.Net MVC framework

- MVC stands for Model View Controller
- Asp.Net MVC is a Framework built on Microsoft .NET Framework to develop web application
- MVC application is separated into three core components:
  - The model
  - The view
  - The controller
- Is a lightweight, highly testable presentation framework
- Defined in the namespace System.Web.MVC

Model View Controller Pattern

- MVC is a pattern for developing applications such that each part has a responsibility that is different from another
- Model: The data of your application
- Views: The template files your application will use to dynamically generate HTML responses.
- Controllers: Classes that handle incoming URL requests to the application, retrieve model data, and then specify view templates that render a response back to the client

Understanding Model View Controller



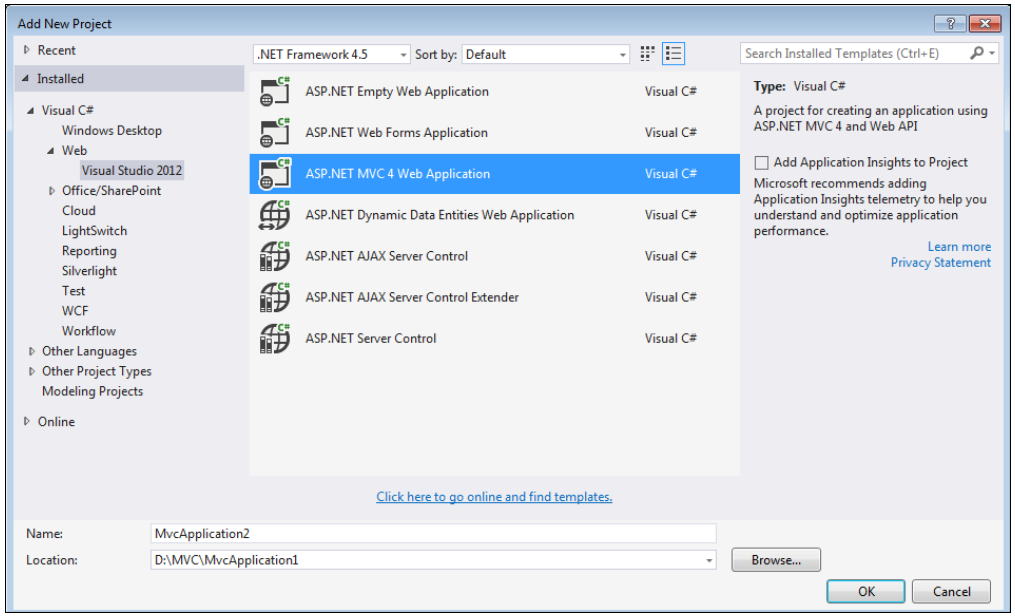
ASP.NET MVC Execution Process

- Receive first request for the application
- Perform routing
- Create MVC request handler
- Create controller
- Execute controller
- Invoke action
- Execute result

http://www.asp.net/mvc/overview/older-versions-1/overview/understanding-the-asp-net-mvc-execution-process

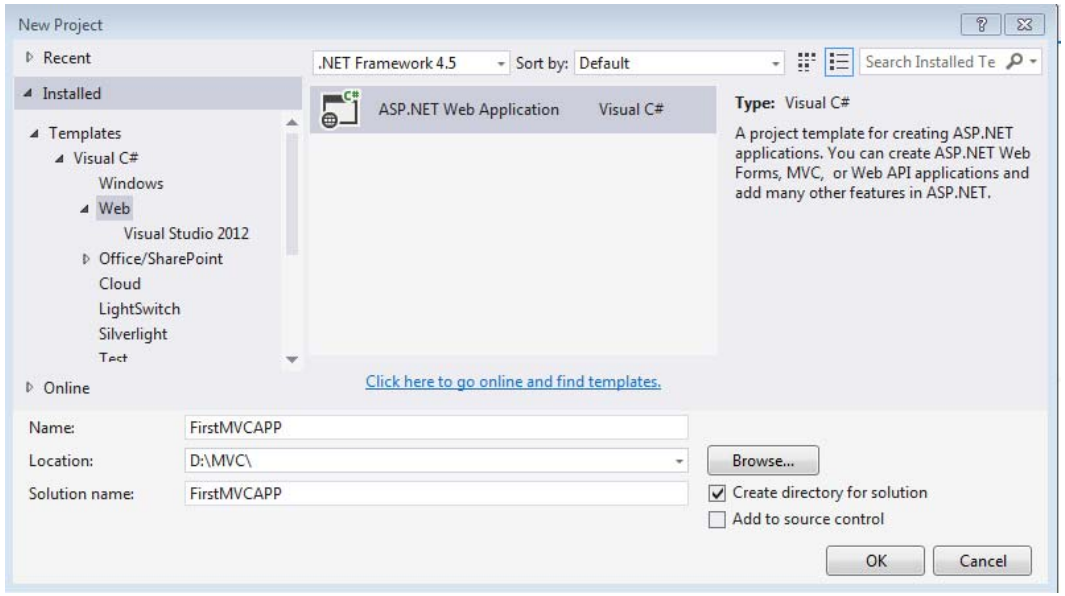
Get Started

- MVC4
- Installing Visual Studio 2012 includes ASP.NET MVC 4
- Visual Studio 2010 SP1 includes MVC 4
- For Visual Studio 2010, install the standalone ASP.NET MVC 4



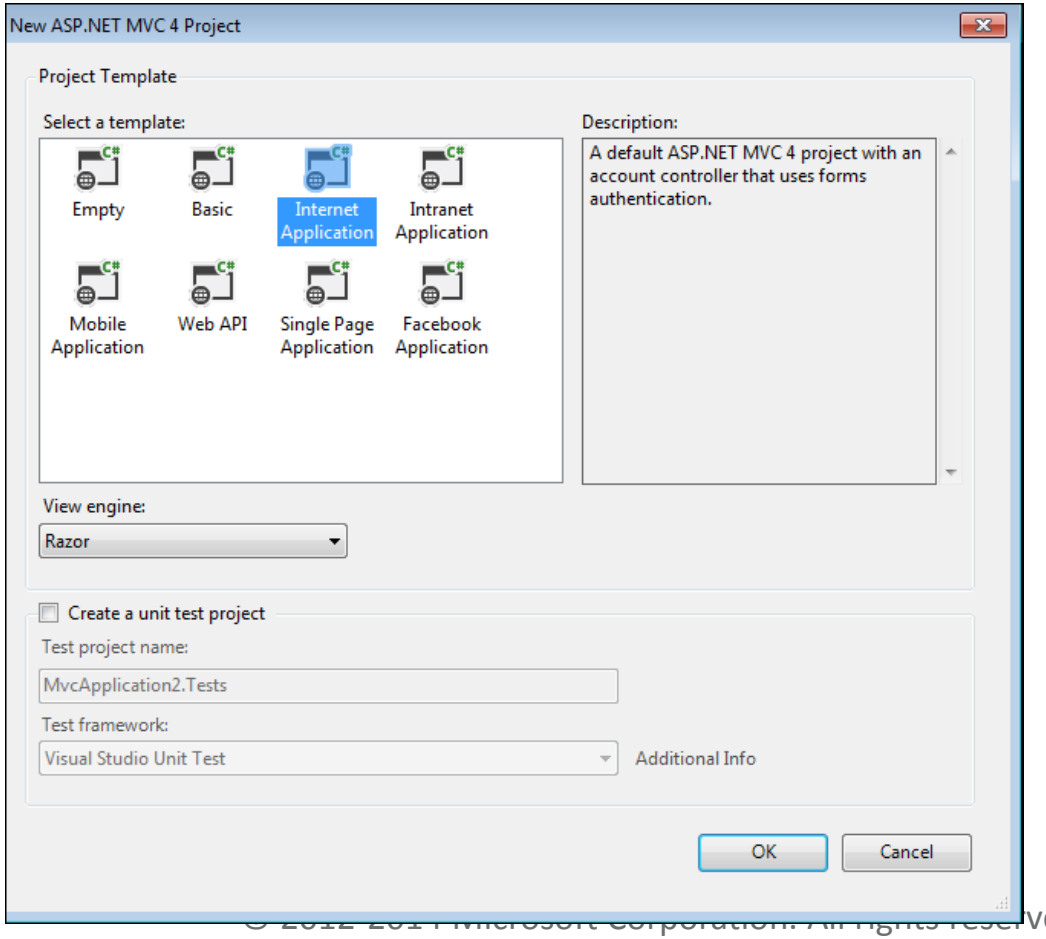
Get Started

- Installing MVC
  - Install ASP.NET MVC 5 support for Visual Studio 2012
  - Installing Visual Studio 2013 includes ASP.NET MVC5
- Creating First Application

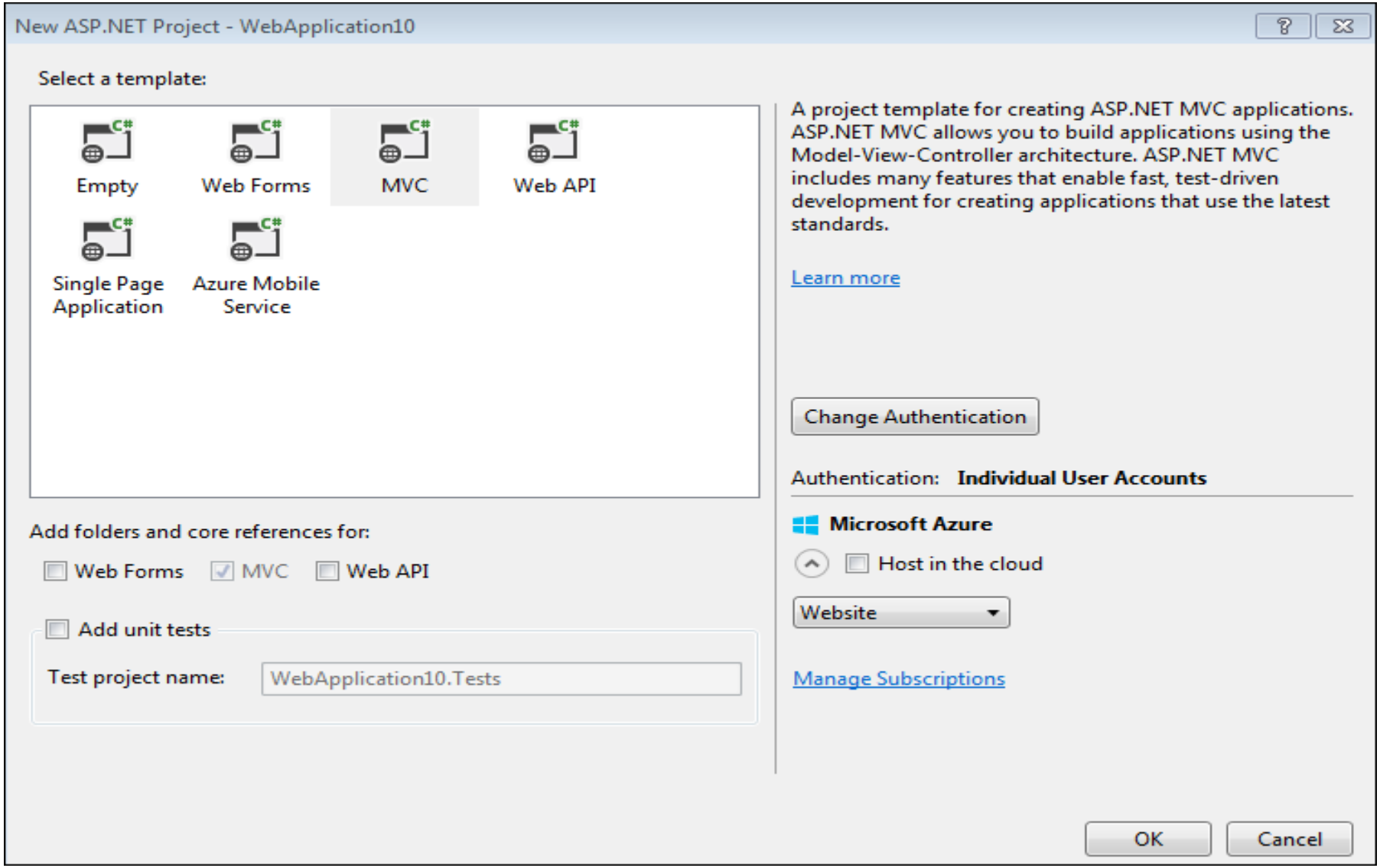


Choosing an ASP.NET MVC 4 Project Template

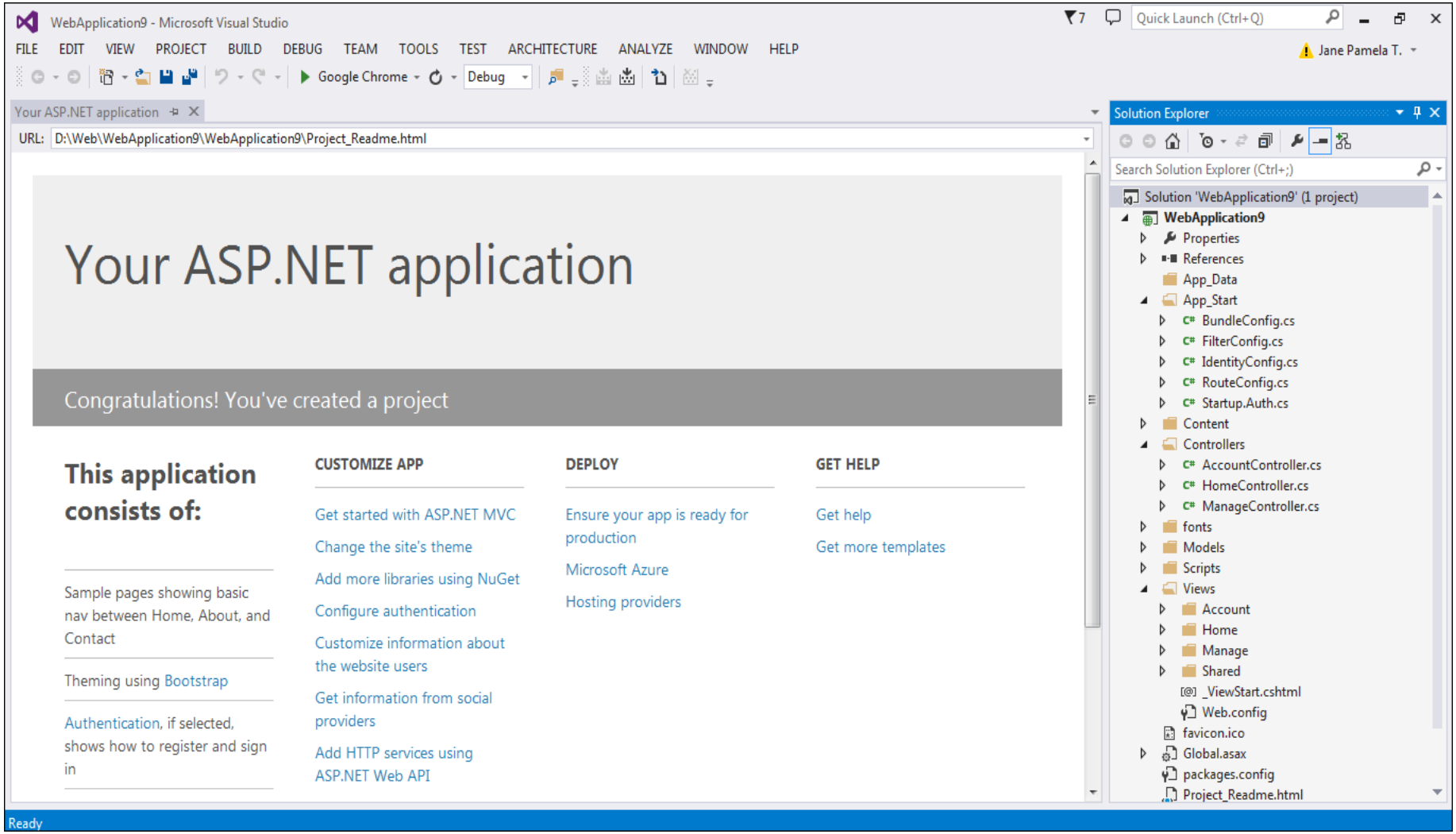
- ASP.NET MVC 4 comes with six built-in Visual Studio templates:
- Empty
- Basic
- Intranet Application
- Intranet Application
- Mobile Application
- Web API.



Choosing an ASP.NET MVC5 Project Template

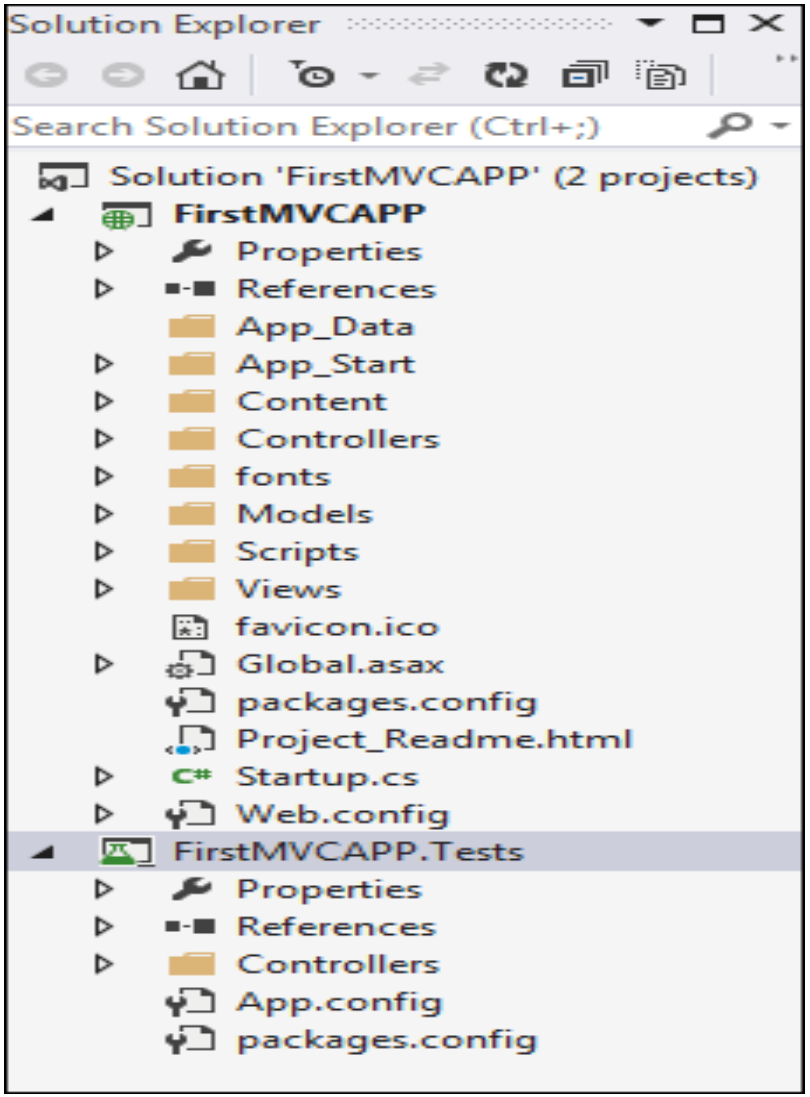


ASP.NET Application

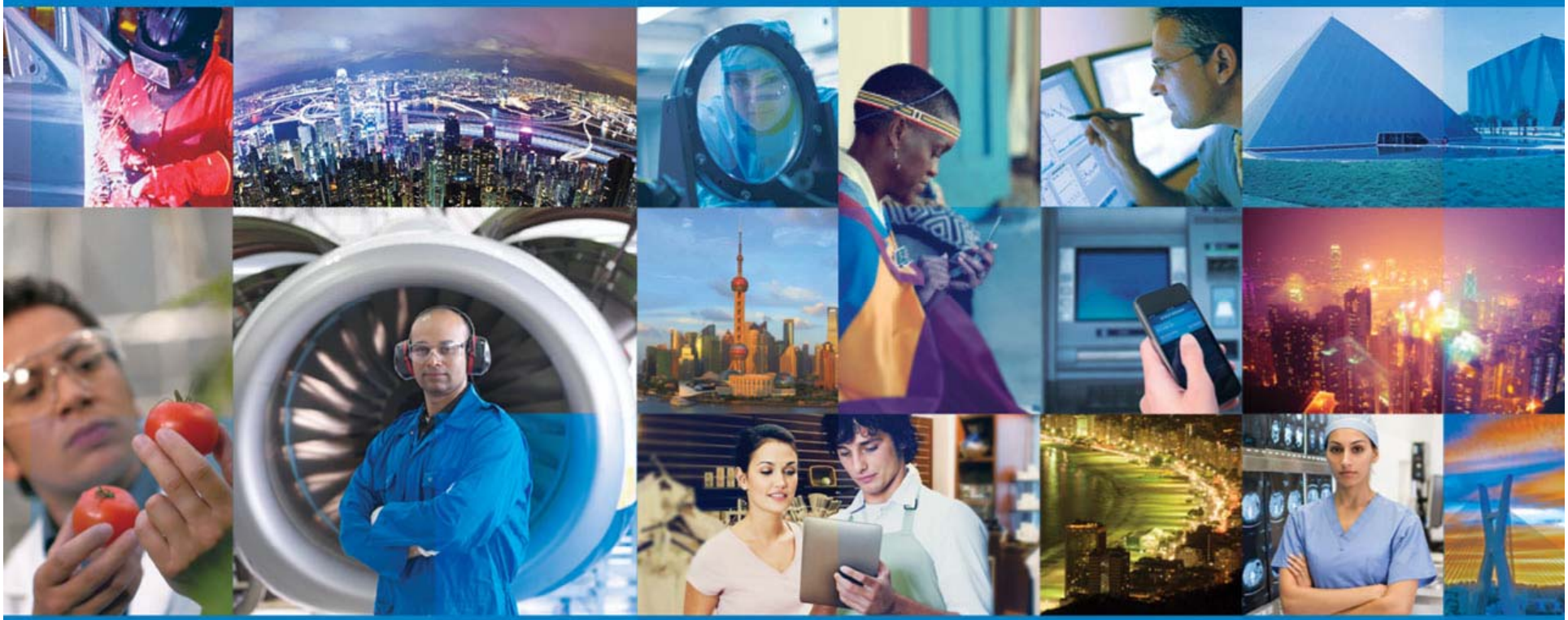


Understanding The MVC Project Structure

- MVC Project



Controllers

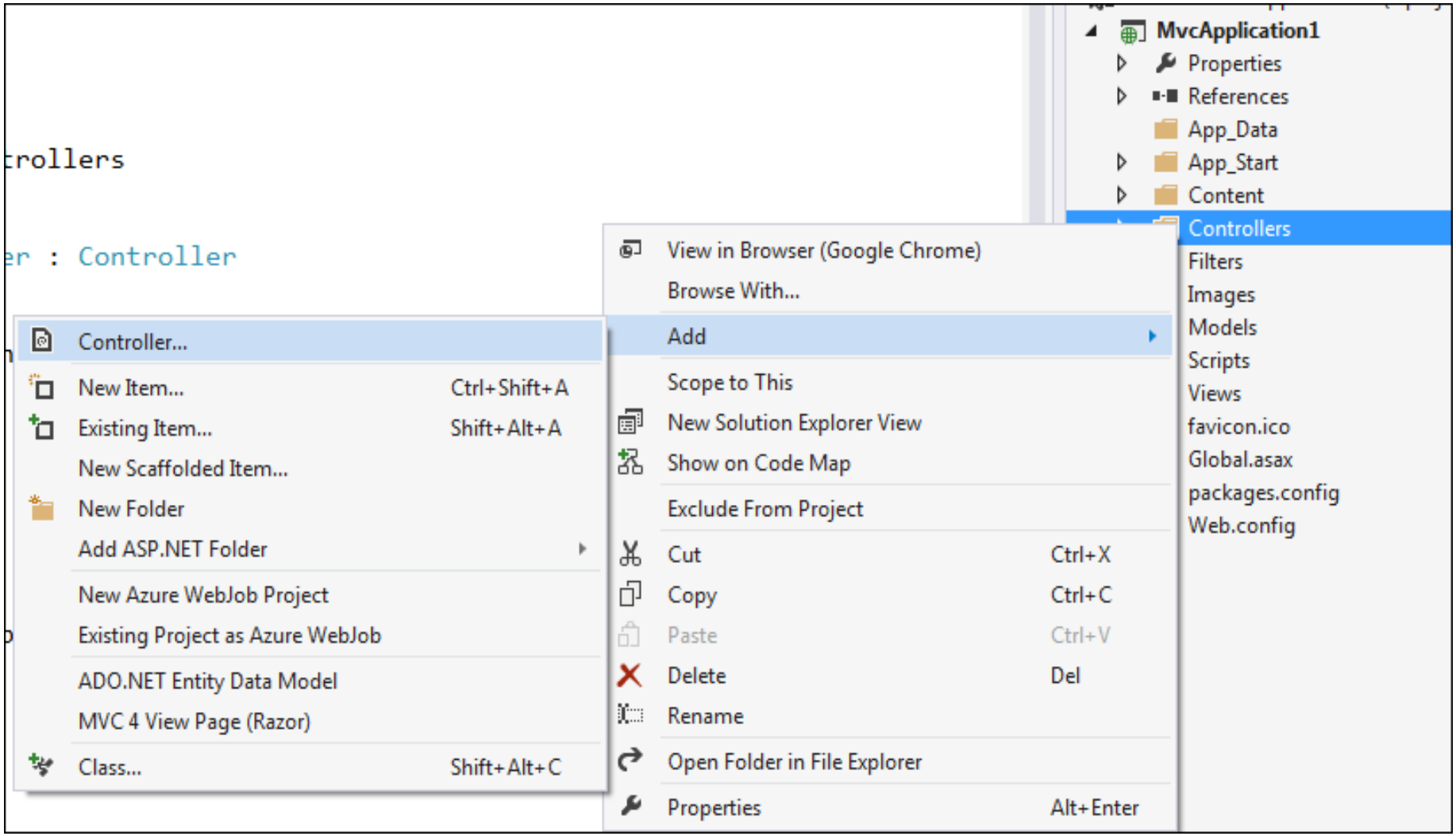


Infosys® | Building Tomorrow's Enterprise

Controller

- Understanding Controllers
  - MVC controllers take the responsibility for responding to the requests
  - Each request is mapped to a particular controller.
- The controller provides three roles in the MVC application.
  - It selects what view should be displayed.
  - It allows clean separation between the view and the model by acting as an intermediary between the two.
  - It processes data before it is passed along.

Creating a Controller



Writing Controller Action Methods

Writing a Controller action includes:

- Create public method
- Return a class that derives from ActionResult
- Add parameters to the method
- Insert code to perform the operation and return the result

Understanding Controller Actions

- A controller is a collection of controller actions.
- An action is a method on a controller that gets called when we enter a particular URL in your browser address bar
- A controller action
  - must be a public method of a controller class
  - cannot be overloaded.
  - cannot be a static method
  - cannot be an extension method.
  - cannot contain **ref** or **out** parameters

Creating Controller Actions with default view

```
public class SampleController : Controller
{
    0 references
    public ActionResult Index()
    {
        ViewBag.Message = "Hello from the Index method";
        return View();
    }
}
```

Passing data to Controller Actions

```
0 references
public string Hello(string id)
{
    return "<H1>Hello " + id + "</h1>";
}
```

```
0 references
public string Hello(string name)
{
    return "<H1>Hello " + name + "</h1>";
}
```

Controller Actions

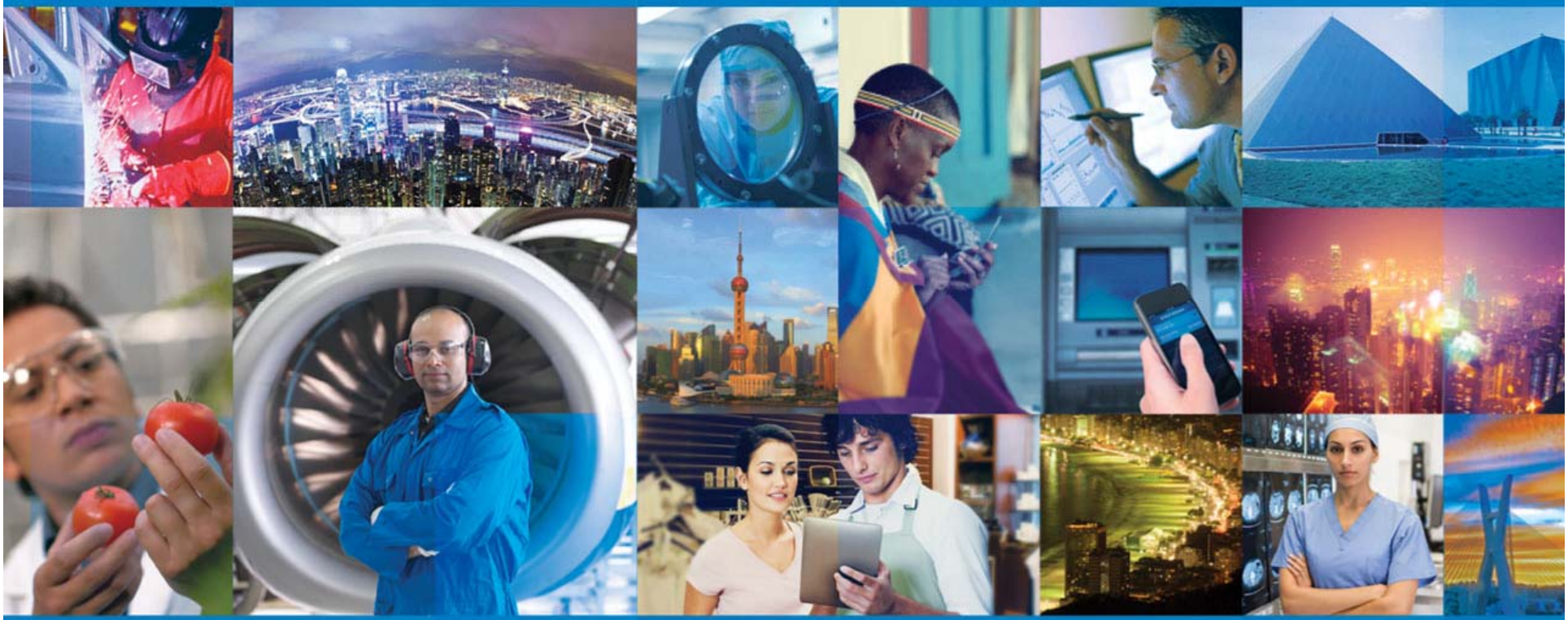
- Optional Parameter Feature in an Action Method

```
0 references
public ActionResult Search(string query = "all", int page = 1)
{
    // ...process request...
    return View();
}
[NonAction]
0 references
public string SomePublicMethod()
{
    return "Hello World";
}
```

Controller and Action methods

URL	Action
http://localhost	Executes default method defined in the default Controller specified in RouteConfig.cs
http://localhost/Employee	Executes default method defined in the Employee Controller
http://localhost/Employee/GetAllEmployees	Executes GetAllEmployees method defined the EmployeeController
http://localhost/Employee/GetEmployee/123	Executes GetEmployee method defined the EmployeeController and provides 123 as ID parameter value
http://localhost/Employee/GetEmployeeByName?name=Dan	Executes GetEmployeeByName method defined the EmployeeController and passes name to the parameter value

Action Results



Infosys® | Building Tomorrow's Enterprise

Action Results

- ViewResult
- PartialViewResult
- RedirectResult
- RedirctToRouteResult
- ContentResult
- JsonResult
- FileResult
- EmptyResult
- HttpNotFoundResult
- HttpUnauthorizedResult
- HttpStatusCodeResult

Action Results

- **return View()**
  - generates HTML to be displayed
  - sends the HTML to the browser.

```
public ActionResult Index()
{
    ViewBag.Message = "Hello from the Index method";
    return View();
}
```

Action Results

- **return RedirectToAction()**
  - redirects the specified action in place of rendering HTML.
  - Similar to Response.Redirect() in Asp.NET Web Form.

```
public ActionResult About()
{
    ViewBag.Message = "Your application description page.";
    return RedirectToAction("Index", "Home");
}
```

```
public ActionResult About(string name)
{
    ViewBag.Message = "Your application description page.";
    return RedirectToAction("Index", "Home", new { UserName = name });

}
```

Action Results

- **return RedirectToRoute()**
  - Looks up the specified route into the Route table which is defined in RouteConfig.cs
  - Redirect to that action or controller and action defined in that route.
  - Similar to RedirectToAction().

```
1 reference
public ActionResult About(string name)
{
    ViewBag.Message = "Your application description page.";
    return RedirectToRoute("Default", new { controller="Home", action="Contact"});
}
```

Action Results

- **return Redirect()**
  - Redirects the specified URL instead of rendering HTML.
  - Similar to Response.Redirect() in Asp.NET Web Form.

```
public ActionResult Hello(string name)
{
    if (string.IsNullOrEmpty(name))
        return Redirect("http://www.infosys.com");
        //return RedirectToAction("Index","Home");
    return Content( "<H1>Hello " + name + "</h1>");
}
```

Action Results

- Return Json()

```
1 reference
public ActionResult About(string name)
{
    ViewBag.Message = "Your application description page.";
    return Json(new{Message="Hello",name="Josh"},JsonRequestBehavior.AllowGet) ;
}
```

- Return File()

```
public ActionResult About(string name)
{
    ViewBag.Message = "Your application description page.";
    return File(Server.MapPath("~/Content/site.css"),"text/css");
}
```

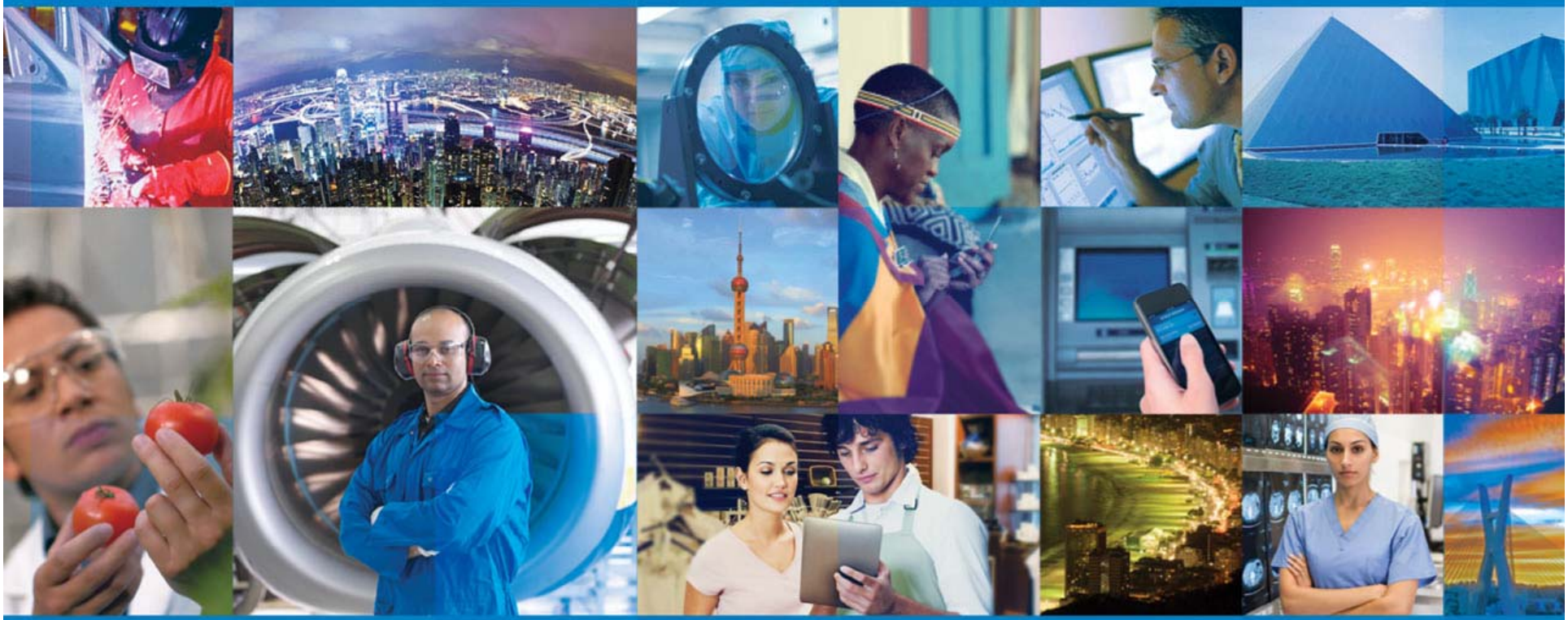
Action Selectors

- ActionName
- ActionVerbs
  - HttpPost
  - HttpGet

```
[ActionName("Reachus")]
1 reference
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

```
[HttpGet ]
0 references
public ActionResult Search()
{
    return Content("Search");
}
[HttpPost]
0 references
public ActionResult Search(string token)
{
    return Content("Search Post");
}
```

Routing in ASP.NET MVC



Infosys® | Building Tomorrow's Enterprise

Routing in Asp.Net MVC

- Is a pattern matching which detects the incoming request and find out what to do with that request.
- At runtime, Routing engine use the Route table for matching the incoming request's URL pattern
- We can register one or more URL patterns to the Route table at Application_Start event using RouteConfig.cs

Routing in Asp.Net MVC

```
public class MvcApplication : System.Web.HttpApplication
{
    0 references
    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

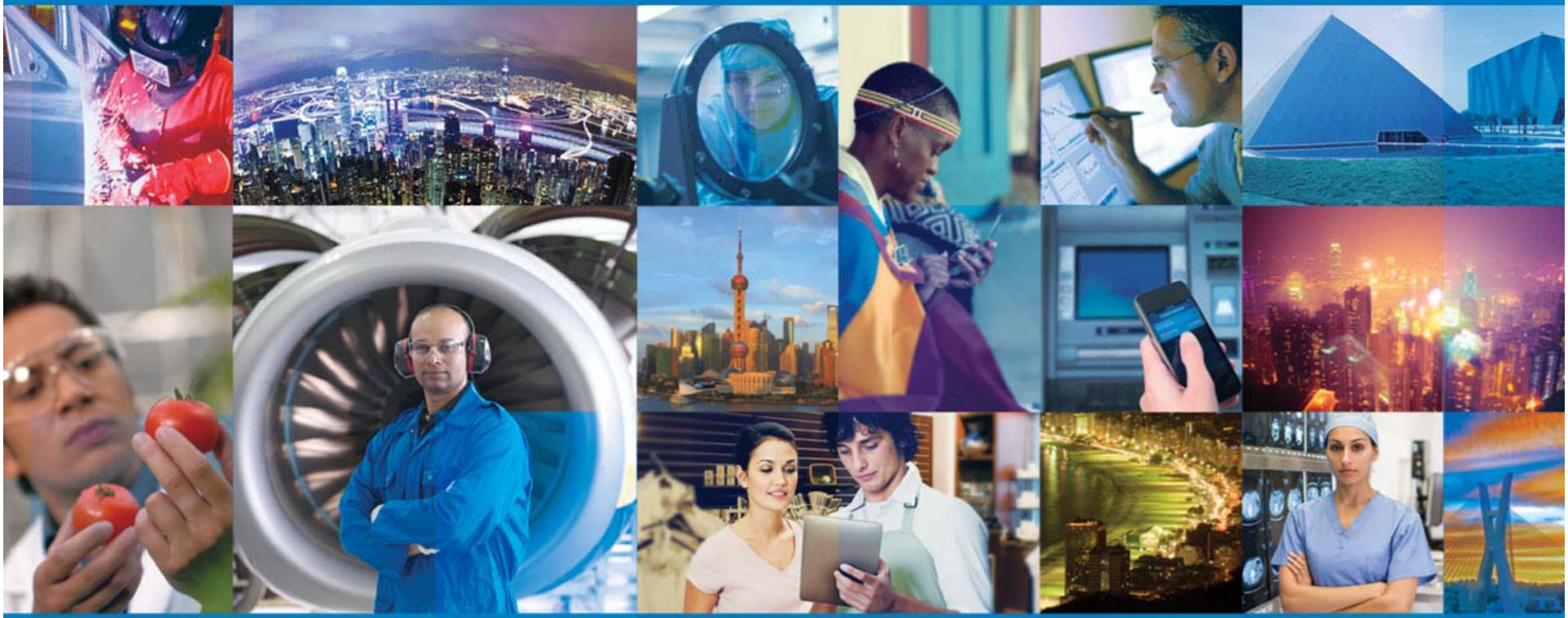
```
public class RouteConfig
{
    1 reference
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

Defining Routing Pattern

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Reports",
        url: "Report/{year}/{month}",
        defaults: new
        {
            controller = "SalesData",
            action = "Reports",
            year = 0,
            month = 0
        }
    );
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}
        defaults: new { controller
            id = UrlParameter.Optio
    );
```

```
public class SalesDataController : Controller
{
    //
    // GET: /SalesData/
    0 references
    public ActionResult Reports(int year,int month)
    {
        ViewBag.Year = year;
        ViewBag.Month = month;
        return View();
    }
}
```

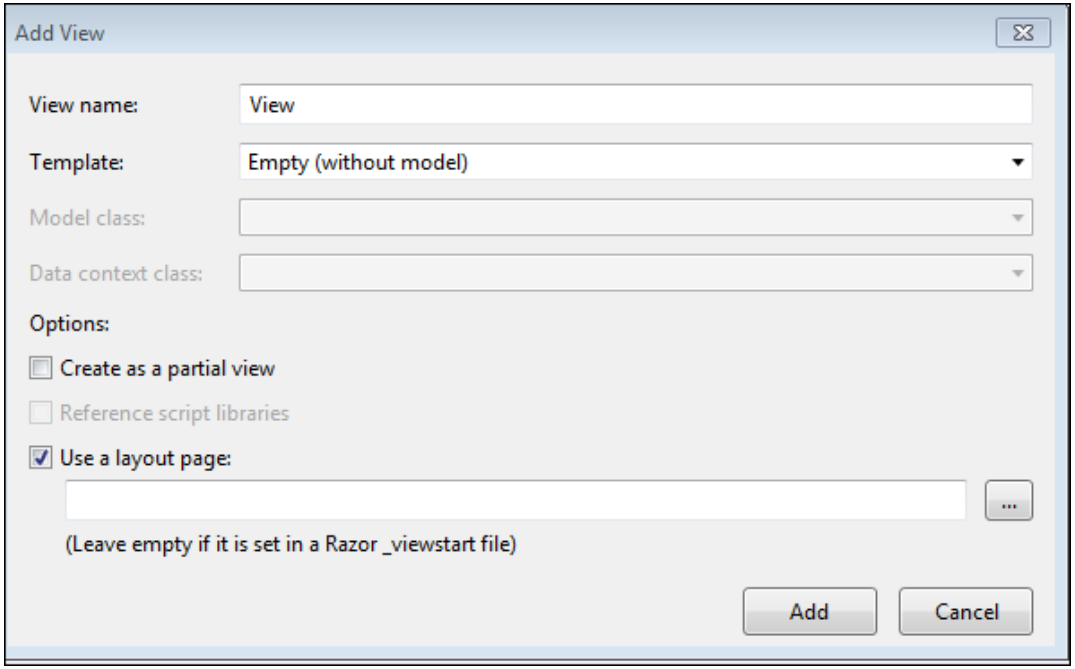
Working with Views

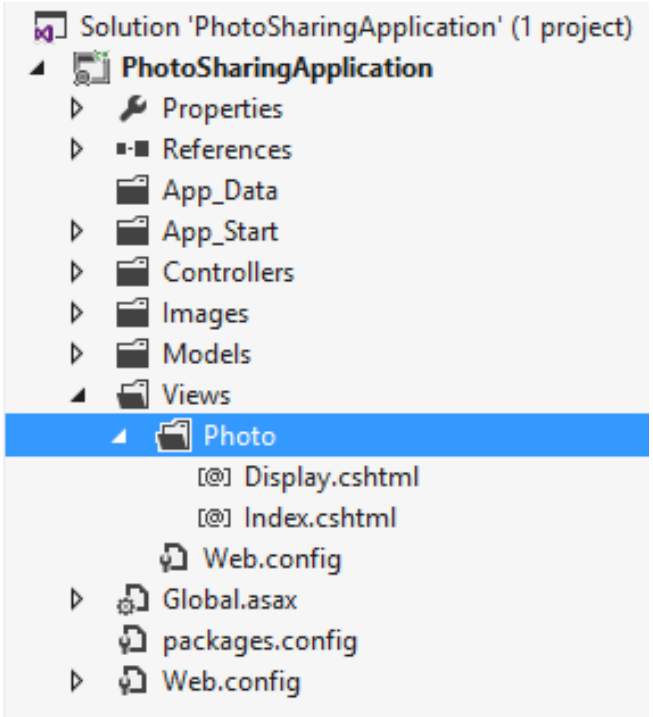


Infosys® | Building Tomorrow's Enterprise

Adding Views

```
6 references
public class HomeController : Controller
{
    1 reference | 0/1 passing
    public ActionResult Index()
    {
        return View();
    }
```





Razor Views

- Template(HTML) + Data(.NET Code)= output
- Creating a view

```
<div>
    @Model.Name
</div>
<div>
    @Model.City, @Model.Country
</div>
```

Features of Razor Syntax

```
@* Razor examples *@
<span>Price including Sale Tax: @Model.Price * 1.3 </span>
<span>Price including Sale Tax: @(Model.Price * 1.1) </span>


@if (Model.Count > 3){
 <ol>
   @foreach(var item in Model)   {
      <li>@item.Name</li>
   } </ol>}
```

Differentiating Server Side Code from HTML

- Razor identifies server-side code by looking for the @ symbol.
- In Razor syntax, the @ symbol has various uses. You can:
  - Use @ to identify server-side C# code
  - Use @@ to render an @ symbol in an HTML page.
  - Use @**:** to explicitly declare a line of text as content and not code.
  - Use **<text>**to explicitly declare several lines of text as content and not code.
- To render text without HTML encoding, you can use the **Html.Raw()** helper.

Controller Action with specific view

```
0 references
public ActionResult Index()
{
    ViewBag.Message = "Hello from the Index method";
    return View("MyView");
}
0 references
public ViewResult Index()
{
    return View("~/Views/Other/Index.cshtml");
}
```

Receiving Data from an Action Method to a View

```
public ViewResult TimeIndex()
{
    DateTime date = DateTime.Now;
    return View(date);
}
```

TimeIndex.cshtml* HomeController.cs

```
@model System.DateTime

@{
    ViewBag.Title = "TimeIndex";
}

<h2>TimeIndex</h2>
The day is: @Model.DayOfWeek
```

Understanding ViewModel

- ViewModel
  - A class which contains the fields which are represented in the strongly-typed view.
  - Is used to pass data from controller to strongly-typed view.
  - Contains fields that are represented in the view
  - Can have specific validation rules using data annotations or IDataErrorInfo
  - Can have multiple entities or objects from different data models or data source.
  - Put only those fields/data that to display on the view/page.
- View represents the properties of the ViewModel, so it is easy for rendering and maintenance. .

STRONGLY TYPED VIEWS

```
public ActionResult Index()
{
    var model =from r in reviews
               orderby r.Rating
                   select r;

    return View(model );
}
```

```
@model IEnumerable<FirstMVCAPP.Models.BlogReview>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Owner)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Category)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Comments)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Rating)
        </th>
        <th></th>
    </tr>
```

ViewData

- Is a dictionary object
- Derived from ViewDataDictionary class
- Is a property of ControllerBase class.
- Is used to pass data from controller to corresponding view.
- It’s life lies during the current request.
- Required typecasting for getting data.

ViewBag

- Is a dynamic property.
- Is a wrapper around the ViewData
- Used to pass data from controller to corresponding view.
- Is a property of ControllerBase class.
- Life lies during the current request.
- It doesn’t required typecasting for getting data.

TempData

- is a dictionary object derived from TempDataDictionary class
- is a property of ControllerBase class
- is used to pass data from current request to subsequent request
- It's life is very short
- It's required typecasting for getting data
- It is used to store only one time messages

Session

Session

- Is an object derived from HttpSessionState class
- Is a property of HttpContext class
- Is used to pass data within the ASP.NET MVC application
- Is valid for all requests by a user
- Is required typecasting for getting data

Receiving Data from an Action Method to a View

```
public ActionResult Index()
{
    var featuredProduct = new Product {
        Name = "Special Muffin cake Assortment!",
        Description = "Delectable Strawberry and chocolate Muffins",
        CreationDate = DateTime.Today,
        ExpirationDate = DateTime.Today.AddDays(7),
        ImageName = "Muffin.jpg",
        Price = 7.99M,
        QtyOnHand = 12 };
    ViewData["FeaturedProduct"] = featuredProduct;
    ViewBag.Product = featuredProduct;
    TempData["FeaturedProduct"] = featuredProduct;
    return View();
}
```

Receiving Data from an Action Method to a View

```
@using WebApplication5.Models;
@{
    var viewDataProduct = ViewData["FeaturedProduct"] as Product;
    var tempDataProduct = TempData["FeaturedProduct"] as Product;
}
<h4>@ViewBag.FeaturedProduct.Name</h4>
<h3> @viewDataProduct.Name</h3>
<h2>@tempDataProduct.Name</h2>
```

Understanding Views

- Views consists of
  - Views pages
  - Layout Pages
  - Html Helpers
  - Partial Views
- View engine has three main functional components.
  - View engine class
  - View class
  - Template parsing engine

Layouts

- Layouts
  - used to maintain a consistent look and feel across multiple views
  - Layouts are like as Master Pages
- Section
  - a region of content within a layout.
  - expects one parameter which is the name of the section.

Organization and Consistency

Layout methods

- RenderBody() -Renders anything in a view not in a section
  - To render child page/view.
  - only one RenderBody method.
- RenderSection(name, required) -Allow views to add specific sections
  - Scripts
  - Banners
  - Sidebars
  - Use @section name to create section in view
    - Note the casing

Rendering layouts

- Layout contains common CSS, jQuery files across the multiple Views and one or more placeholders for which Views provide content
- We can render the layout by the following ways
  - using _ViewStart file
  - Return Layout from ActionResult
  - Define Layout with in each view
  - Adding _ViewStart file in each of the directories

Using _ViewStart file

```
@{
    var current = HttpContext.Current.Request.RequestContext;
    var controller = current.RouteData.Values["Controller"].ToString();
    string layout = "";
    if (controller == "Admin")
    {
        layout = "~/Views/Shared/_AdminLayout.cshtml";
    }
    else
    {
        layout = "~/Views/Shared/_Layout.cshtml";
    }
    Layout = layout;
}
```

Layout

- Return Layout from ActionResult

```
0 references
public ActionResult Index()
{
    RegisterModel model = new RegisterModel();
    //TO DO:
    return View("Index", "_AdminLayout", model);
}
```

- Define Layout with in each view
- Adding _ViewStart file in each of the directories

```
@{
    Layout = "~/Views/Shared/_AdminLayout.cshtml";
}
```

Render Body

```
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    </footer>
</div>
```

Partial View

- Is like as user control in Web forms.
- To reduce code duplication.
- Reusable

Creating Partial Views

You can use partial views to render the same HTML content in different locations in your web application

- Creating and Naming Partial Views:
  - Create a partial view by using the **Add View** dialog
  - Name partial views with an underscore prefix to keep to convention
- Strongly-typed and dynamic partial views:
  - Create strongly-typed partial views if you are certain that the partial view will always display the same model class.
  - Create dynamic partial views if you are not sure if the partial view will always display the same model class.

Using Partial Views

- Using HTML helpers, we can include partial views within other views in a web application:
  - To pass the same model object to a partial view from the parent view, use Html.Partial()
  - To pass a model object to a partial view, which is different from the parent view or of a different model class, use Html.Action()
- Use the ViewBag and ViewData collections to share data between the controller action, parent view, and partial view

Html.RenderPartial

- Directly written to the HTTP response stream
- Returns void.
- Simple to use and no need to create any action.
- Useful when the displaying data in the partial view is already in the corresponding view model.
- Faster than Partial method since its result is directly written to the response stream

```
@{Html.RenderPartial("_Comments");}
```

Html.Partial

- Renders as an HTML-encoded string.
- Result can be stored in a variable
- Simple to use
- No need to create any action.
- Partial method is useful when the displaying data in the partial view is already in the corresponding view model

@Html.Partial("_Comments")

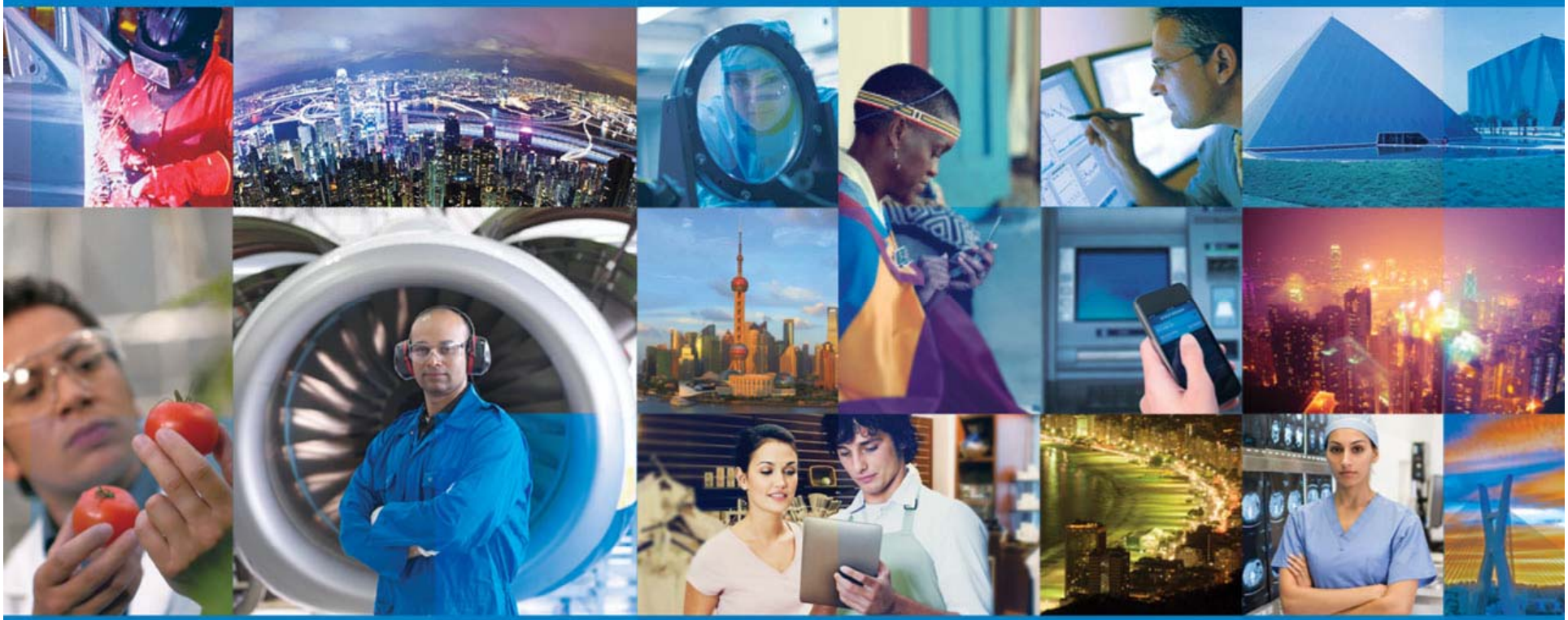
Partial View

- Returning Partial View using Html.Action

```
@Html.Action("GetProduct","Welcome")
```

```
[ChildActionOnly]
0 references
public ActionResult GetProduct()
{
    return PartialView("_Partial");
}
```

Using Helpers



Infosys® | Building Tomorrow's Enterprise

Understanding HTML Helpers in ASP.NET MVC

- HTML helpers are to invoke on the Html property of a view.
- Inline HTML Helpers
- Built-In HTML Helpers
  - Standard HTML Helpers
  - Strongly Typed HTML Helpers
  - Templated HTML Helpers
- Custom HTML Helpers

Inline Html Helpers

```
@helper ListingItems(string[] items)
{
    <ol>
        @foreach (string item in items)
        {
            <li>@item</li>
        }
    </ol>
    }
<h3>Programming Languages:</h3>
@ListingItems(new string[] { "C++", "Java", "C#" })
<h3>Book List:</h3>
@ListingItems(new string[] { "Let us C++", "Let us Java", "Let us C#" })
```

Standard Html Helpers

- Html is a property of the ViewPage Base class
  - Create links

```
<li>@Html.ActionLink("Home", "Index", "Home")</li>
```

  - Create inputs
  - Create forms

```
@using (Html.BeginForm()) {
        @Html.ValidationSummary(true)
            <div class="editor-label">
                @Html.LabelFor(model => model.FirstName)
            </div>
            <div class="editor-field">
                @Html.EditorFor(model => model.FirstName)
                @Html.ValidationMessageFor(model => model.FirstName)
            </div>
```

Using Action Helpers

- **Html.ActionLink()**

```
@Html.ActionLink("Click here to view Item 1", "Display", new { id = 1 })
```

```
<a href="/Home/Display/1">Click here to view Item 1</a>
```

- **Url.Action()**

```
<img alt="This picture came from an action"  src="@Url.Action("GetPicture", new { id = 1 })" />
```

```
<img alt="This picture came from an action"
src="/Home/GetPicture/1" />
```

Html.RenderAction

- Directly written to the HTTP response stream
- Need to create a child action for the rendering the partial view
- Useful when the displaying data in the partial view is independent from corresponding view model
- Is the best choice when we want to cache
- Is faster than Action method

```
@{Html.RenderAction("Category","Home");}
```

Html.Action

- Renders as an HtmlString .
- Result can be stored in a variable, since it returns string type value.
- Is useful when the displaying data in the partial view is independent from corresponding view model.
- Is also the best choice when you want to cache.

```
@{Html.Action("Category","Home");}
```

Using Display Helpers

- **Html.DisplayNameFor()**

```
@Html.DisplayNameFor(model =>
model.CreatedDate)
```

Created Date

- **Html.DisplayFor()**

```
@Html.DisplayFor(model =>
model.CreatedDate)
```

03/12/2012

The Begin Form Helper

- Html.BeginForm()

```
@using (Html.BeginForm("Create", "Employee",
   FormMethod.Post,
   new { enctype = "multipart/form-data" }))
{
   @* Place input controls here *@
}
```

```
<form action="/employee/Create"
method="post"  enctype="multipart/form-
data">
</form>
```

Using Editor Helpers

- **Html.LabelFor()**
- **Html.EditorFor()**

```
@Html.LabelFor(model => model.ContactMe)
```

```
<label for="ContactMe">
  Contact Me
</label>
```

```
@Html.EditorFor(model => model.ContactMe)
```

```
<input type="checkbox"
   name="Description">
```

Using Validation Helpers

- **Html.ValidationSummary()**
- **Html.ValidationMessageFor()**

```
@Html.ValidationSummary()
```

```
<ul>
  <li>Please enter your last name</li>
  <li>Please enter a valid email address</li>
</ul>
```

```
@Html.ValidationMessageFor(model =>
model.Email)
```

Please enter a valid email address

Custom Helper Methods

- We can create our own custom helper methods
  - by an extension method on the HtmlHelper class
  - by static methods with in a utility class

URL Helpers in MVC

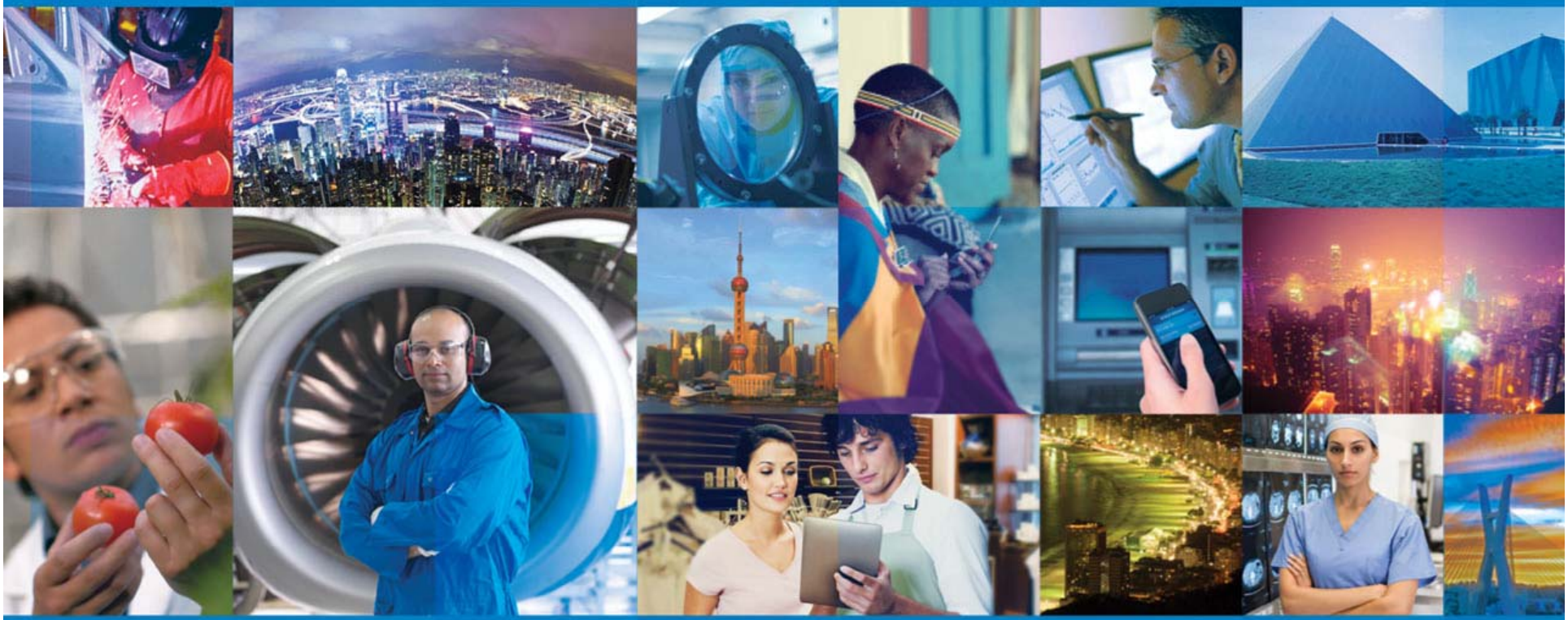
- build URLs and return the URLs as strings

Types	example
Action	 @Url.Action("Browse", "Store", new { genre = "Jazz" }, null)
Content	<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
RouteUrl	

AJAX helpers in MVC

- Are used to create AJAX enabled elements which performs request asynchronously.
- Are extension methods of AJAXHelper class from System.Web.Mvc namespace.
- @Ajax.ActionLink("Items List", "GetItems", new AjaxOptions {UpdateTargetId = “Items-container", HttpMethod = "GET" })
- ASP.NET MVC supports unobtrusive Ajax which is based on jQuery

Bundling and Minification



Infosys® | Building Tomorrow's Enterprise

Performance Optimization with Bundling & Minification

- Bundling and Minification techniques reduce
  - The number of request to the server
  - The size of CSS and JavaScript library
  - Improves page loading time
- System.web.optimization class offers the bundling and minification techniques.
- Bundle
  - Is a logical group of files that is loaded with a single HTTP request.
  - Style bundle for CSS
  - Script bundle for java scripts

Bundling

- Creating Style bundle

```
bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));
```

- Creating Script bundle

```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
            "~/Scripts/jquery-{version}.js"));
```

- Creating Bundle using the "*" Wildcard Character

```
bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
            "~/Scripts/jquery.unobtrusive*",
            "~/Scripts/jquery.validate*"));
```

- Registering Bundle

```
protected void Application_Start()
{

    BundleConfig.RegisterBundles(BundleTable.Bundles);
}
```

Bundling

- Adding to Layout Page

```
@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr")
```

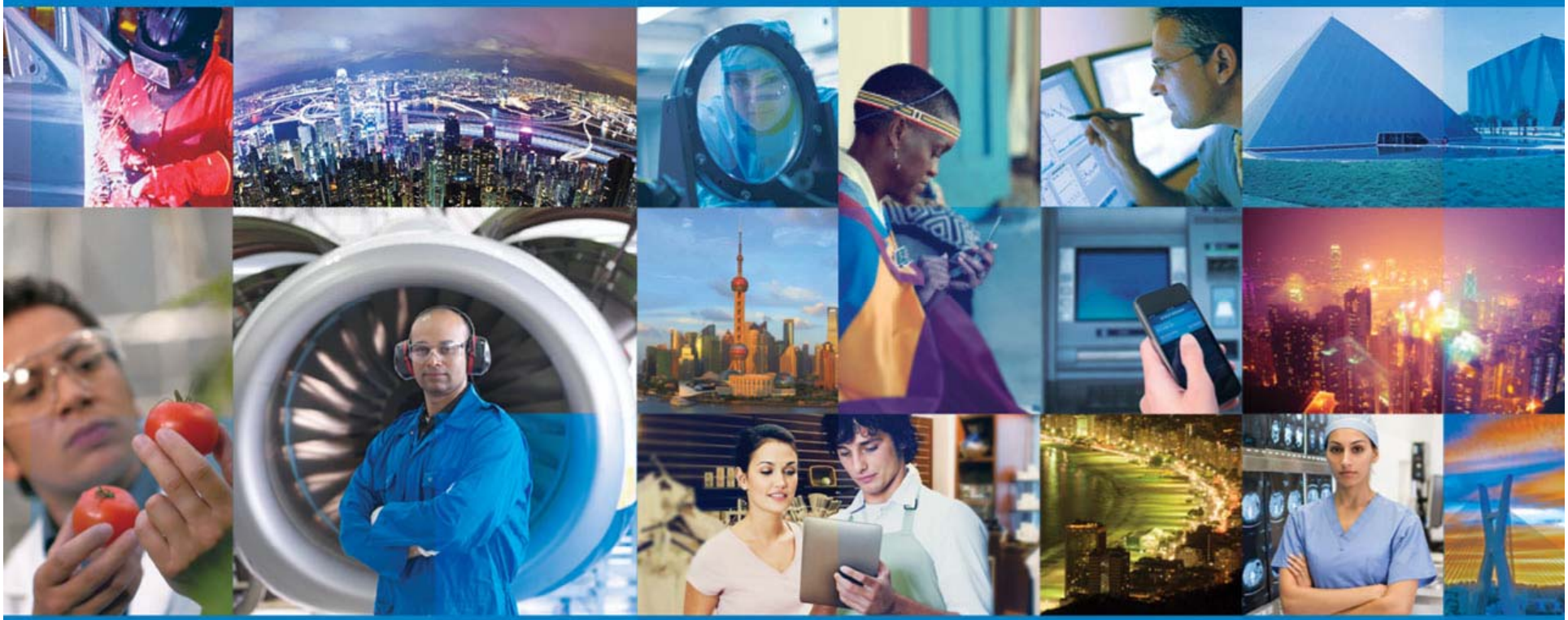
- Enabling in debug mode

```
protected void Application_Start()
{
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    BundleTable.EnableOptimizations = true;
}
```

Minification

- Technique for removing unnecessary characters (like white space, newline, tab) and comments
- From JavaScript and CSS files
- Reduces the size which cause improved load times of a webpage
- There are so many tools for minifying css and js files
- Tools example : JSMin and YUI compressor

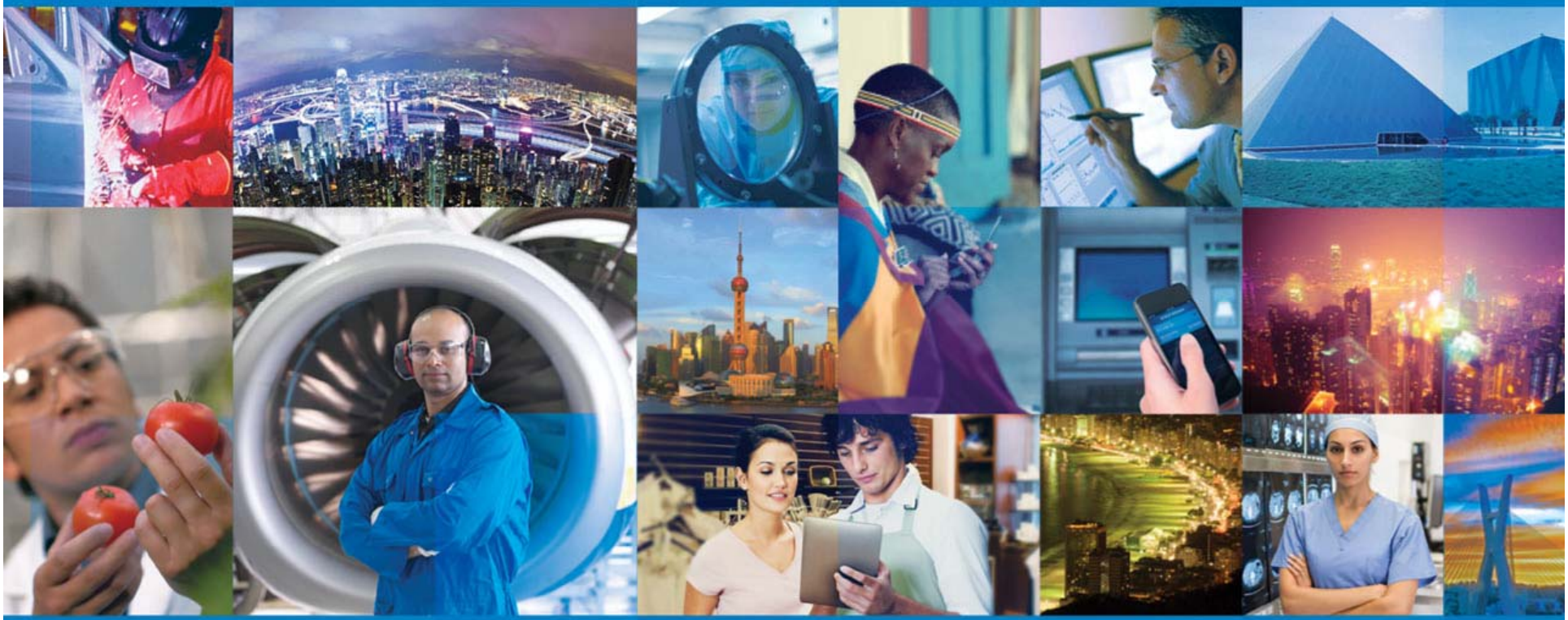
MVC Areas



MVC Areas

- Allow us to organize models, views, and controllers into separate functional sections of the application, such as administration, billing, customer support, and so on
- This is very helpful in a large web application, where all the controllers, views, and models have a single set of folders and that become difficult to manage
- Each MVC area has its own folder structure which allow us to keep separate controllers, views, and models
- This also helps the multiple developers to work on the same web application without interfere to one another.

Action Filters

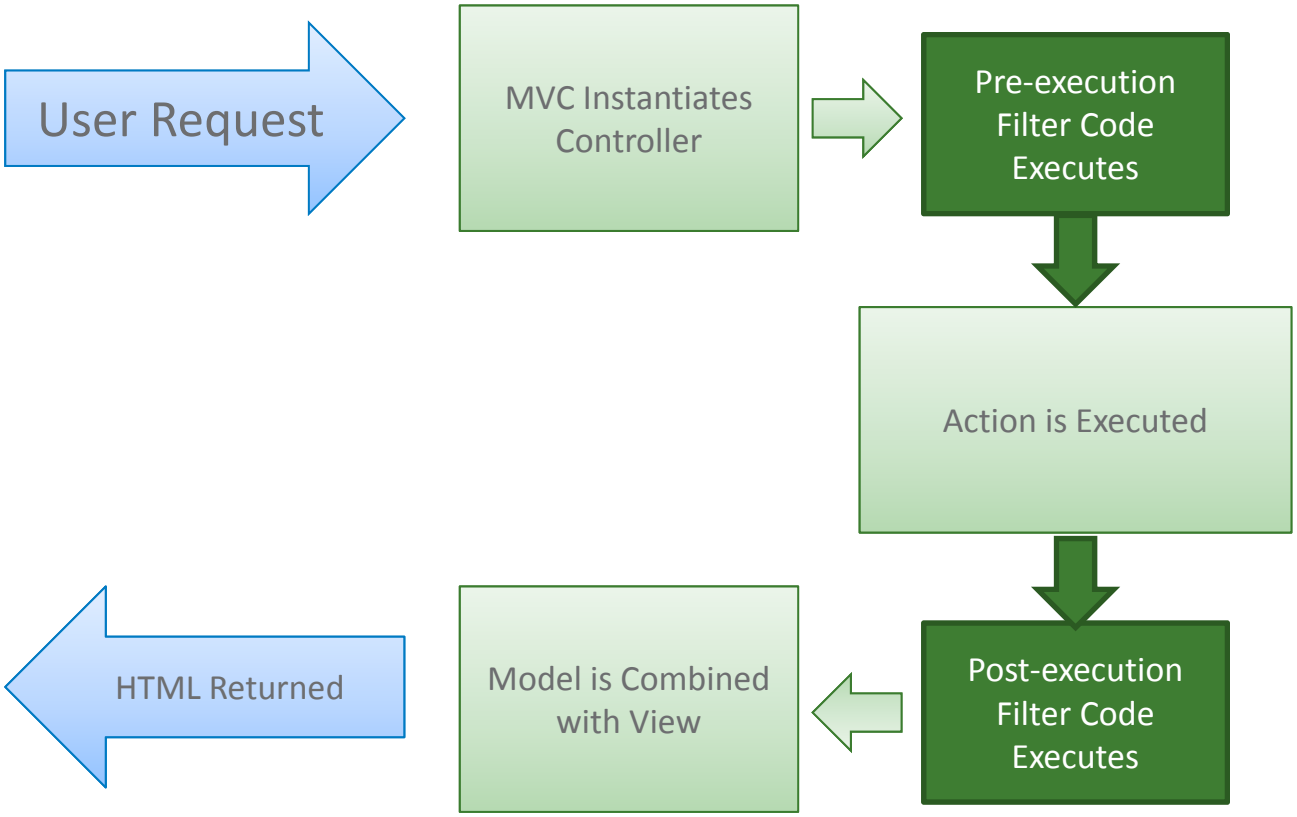


Infosys® | Building Tomorrow's Enterprise

Action Filters

- Easy way to insert a piece of code or logic either before or after an action is executed.
- An action filter is an attribute
- Some of the common functionalities than can be implemented:
  - Custom Authentication
  - Custom Authorization(User based or Role based)
  - Error handling or logging
  - User Activity Logging
  - Data Compression
  - Data Caching

Actions with Filters



Types of Filters and Execution Order

- Authentication filters
- Authorization filters
- Action filters
- Result filters
- Exception filters
- Output Cache
- ValidateAntiForgeryToken
- ValidateInput

Higher

Lower

Configuring Filters

- **Global level**

```
protected void Application_Start()
{
FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
}
```

- **Controller level**

```
[Authorize(Roles="Admin1")]
public class AdminController : Controller
{
//
}
```

Configuring Filters

- **Action level**

```
public class UserController : Controller
{
[Authorize(Users="User_1,User_2")]
public ActionResult LoginIndex(string provider)
{
// TODO:
return View();
}
}
```

Authorization Attribute

- Used to restrict access to an action method
- Require Https
- Authorize
  - Users
  - Roles

The Base ActionFilterAttribute Class

- The following methods to override:
- OnActionExecuting – This method is called before a controller action is executed.
- OnActionExecuted – This method is called after a controller action is executed.
- OnResultExecuting – This method is called before a controller action result is executed.
- OnResultExecuted – This method is called after a controller action result is executed

Output Cache Filter

```
public class DataController : Controller
    {
        [OutputCache(Duration=10)]
        public string Index()
        {
            return DateTime.Now.ToString("T");
        }
    }
```

Exception Handling

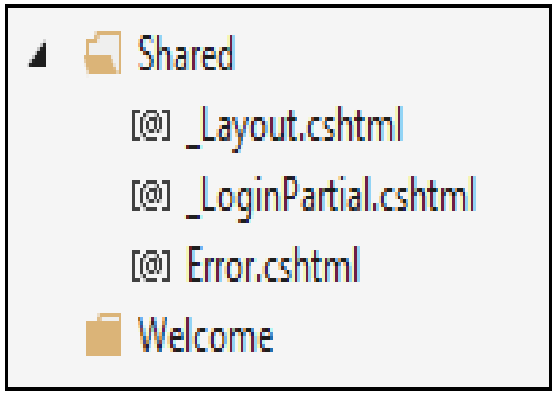
- Bulit-in exception filter - HandleError
- HandleError Attribute can be used to handle at
  - Action Method level, Controller level and Global level
- Limitation of HandleError
  - No support to log the exceptions as it suppress the error once it is handled.
  - Only catch 500 Http error and doesn't catch other HTTP errors like 404,401 etc.
  - Doesn‘t catch the errors occurs in model.

Handle Error Filter

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
}
```

```
public ActionResult Search()
{
    throw new Exception("Something terrible has happened");
    return View();
}
```

```
<customErrors mode="On"/>
```



ValidateAntiForgeryToken

- Generates a hidden form field (anti-forgery token) that is validated when the form is submitted.
- Used to help protect your application against cross-site request forgery

```
[HttpPost]
[Authorize(Roles = "Admins")]
[ValidateAntiForgeryToken()]
public ActionResult Edit(ProductDetails productdetails)
{
    if (ModelState.IsValid)
    {
        db.Entry(productdetails).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(productdetails);
}
```

ValidateInput Filter

- Used to mark action methods whose input must be validated
- Filters when user sends HTML values to the controller
- ValidateInput(false) allows to send HTML values

```
[HttpPost]
[ValidateInput(true)]
0 references
public ActionResult GetTest(string name)
{
    return Content("Post Test Method");
}
```

Filter Overrides

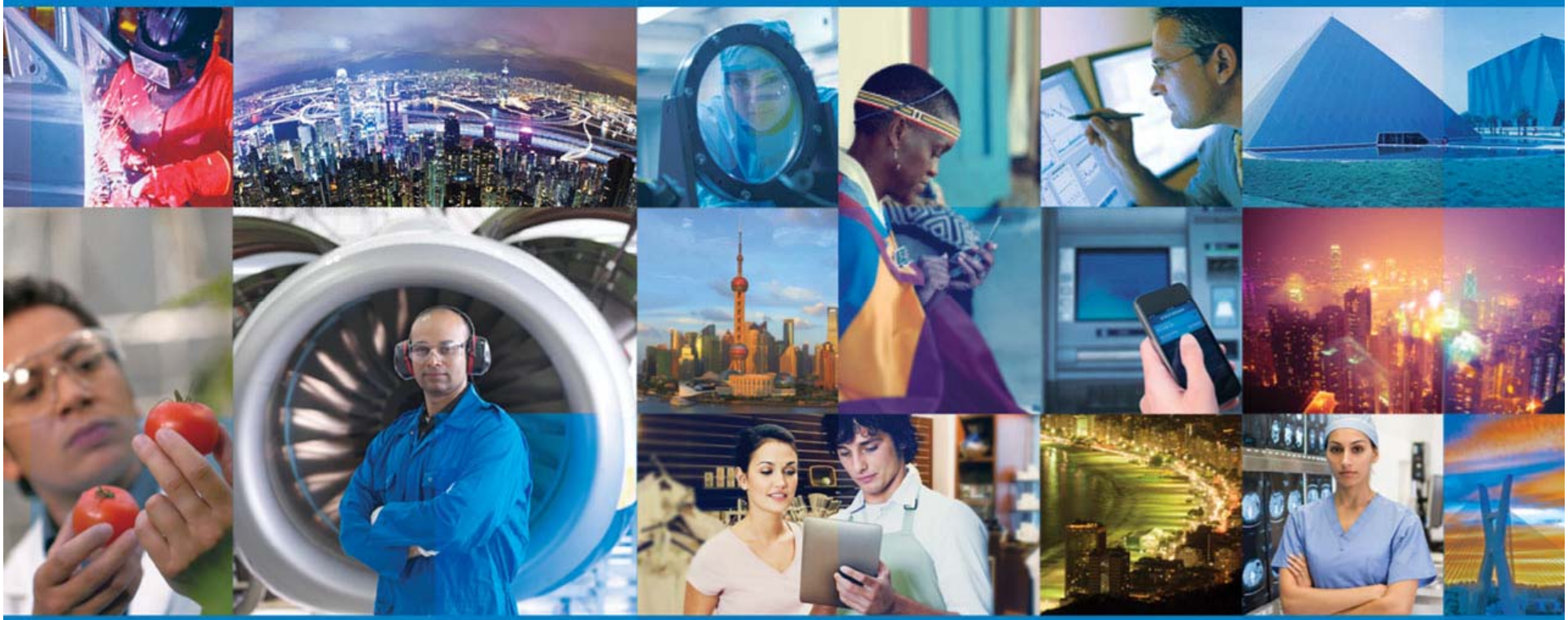
- We can exclude
  - A specific action method from a controller filter
  - A controller from the global filter
- Filter override types
  - OverrideAuthenticationAttribute
  - OverrideAuthorizationAttribute
  - OverrideActionFiltersAttribute
  - OverrideResultAttribute
  - OverrideExceptionAttribute

Filter Overrides

```
[Authorize(Users = "SuperAdmin")]
6 references
public class HomeController : Controller
{
    1 reference
    public ActionResult Index()
    {
        ViewBag.Message = "Welcome to ASP.NET MVC!";
        return View();
    }
    [OverrideAuthorization]
    1 reference
    public ActionResult About()
    {
        return View();
    }
}
```

```
[Authorize(Users = "SuperAdmin")]
6 references
public class HomeController : Controller
{
    [OverrideAuthorization]
    [Authorize(Users = "Dan")]
    1 reference | 0/1 passing
    public ActionResult About()
    {
        return View();
    }
}
```

Working with Models



Infosys® | Building Tomorrow's Enterprise

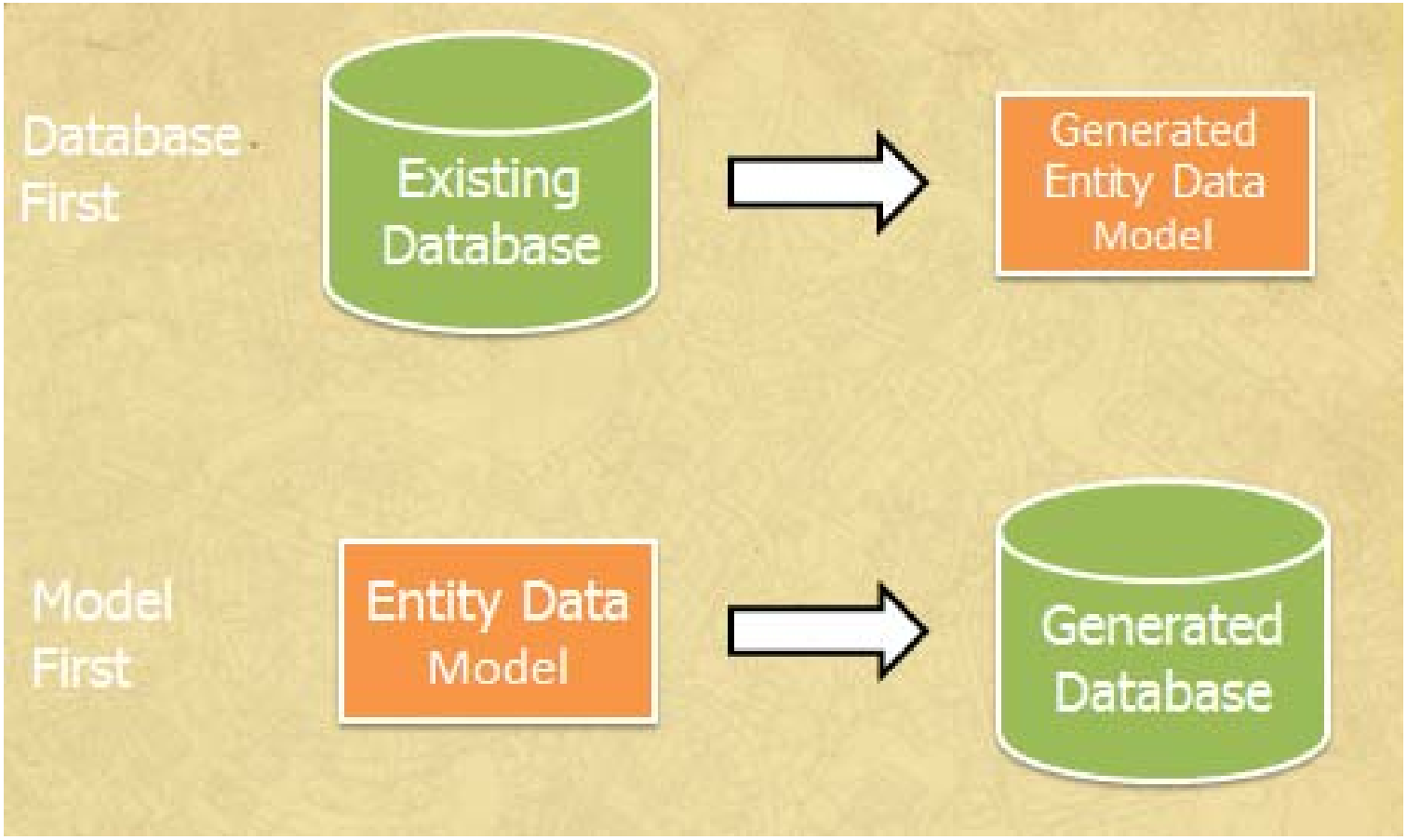
The Entity Framework

Types of Entity Framework Workflows

- Database First
- Model First
- Code First

- Access to relational database
- Using strongly typed LINQ queries

Model



Working with Entity Framework

- Install Entity Framework

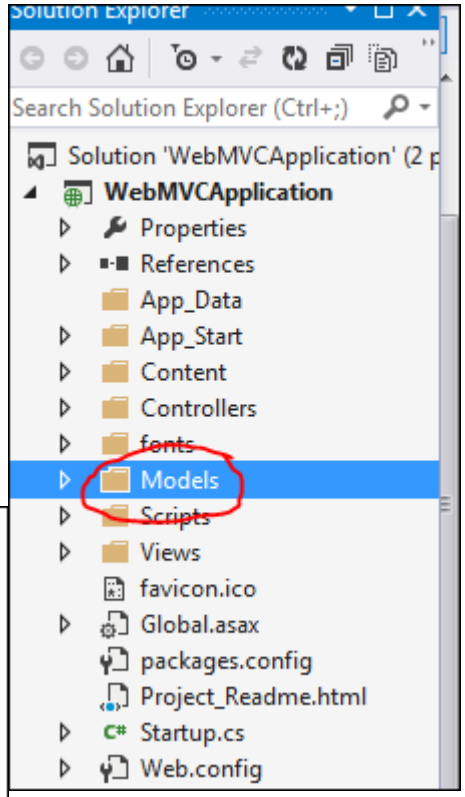
```
Install-Package EntityFramework
```

- Create the Data Model
- Create the Database Context
  - Specifying entity sets
- Set up EF to initialize the database with test data
- Set up EF to use a SQL Server
- Creating Controller and Views
- View the Database

What is a Model

- A Class
- MVC Model
  - Input controls
  - Display formatting
  - Validation

```
namespace WebMVCApplication.Models
{
    1 reference
    public class Album
    {
        0 references
        public int AlbumId { get; set; }
        0 references
        public string Title { get; set; }
        0 references
        public Artist Artist{get;set;}
    }
}
```



Developing Models


```
public class Photo
{
   public int PhotoID { get; set; }
   public string Title { get; set; }
   public byte[] PhotoFile { get; set; }
   public string Description { get; set; }
   public DateTime CreatedDate { get; set; }
   public string Owner { get; set; }
   public virtual List<Comment> Comments { get;
set; }
}
```

Preparing Models

- Attributes
  - “Decorate” properties
- Available Attributes
  - DataTypeAttribute
  - DisplayAttribute
  - Validation
    - RequiredAttribute
    - StringLengthAttribute
    - RegularExpressionAttribute
    - CompareAttribute

Data Types

- Available Data Types
  - CreditCard
  - Currency
  - EmailAddress
  - Password
  - Url

Prompts

- Display Attribute
  - Name
    - Display prompt and header

DisplayFormat

- Uses a .NET format string
  - {0:C} Currency
  - {0:0.00}
    - Always use two decimal places
      - .42 displays as 0.42
      - 42 displays as 42.00
  - {0:d-m-yy}
    - Single digit day & month, two digit year
      - June 10th, 2014 displays as 10-6-14

Adding Validation

- Attributes
  - Required
  - StringLength
    - MinLength
    - MaxLength
  - RegularExpression
  - Range
- Error Message
  - {0} will use the display name Ex: "{0} must be provided"

Display and Edit Data Annotations

```
public class Photo
{
   // other properties excluded
   [DisplayName("Picture")]
   public byte[] PhotoFile { get; set; }

   [DataType(DataType.MultilineText)]
   public string Description { get; set; }

   [DataType(DataType.DateTime)]
   [DisplayName("Created Date")]
   [DisplayFormat(DataFormatString = "{0:dd/MM/yy}"]
   public DateTime CreatedDate { get; set; }
}
```

Validating User Input with Data Annotations

```
public class Person
{
   public int PersonID { get; set; }

   [Required(ErrorMessage="Please enter a name.")]
   public string Name { get; set; }

   [Range(0, 400)]
   public int Height { get; set; }

   [Required]
   [DataType(DataType.EmailAddress)]
   public string EmailAddress { get; set; }
}
```

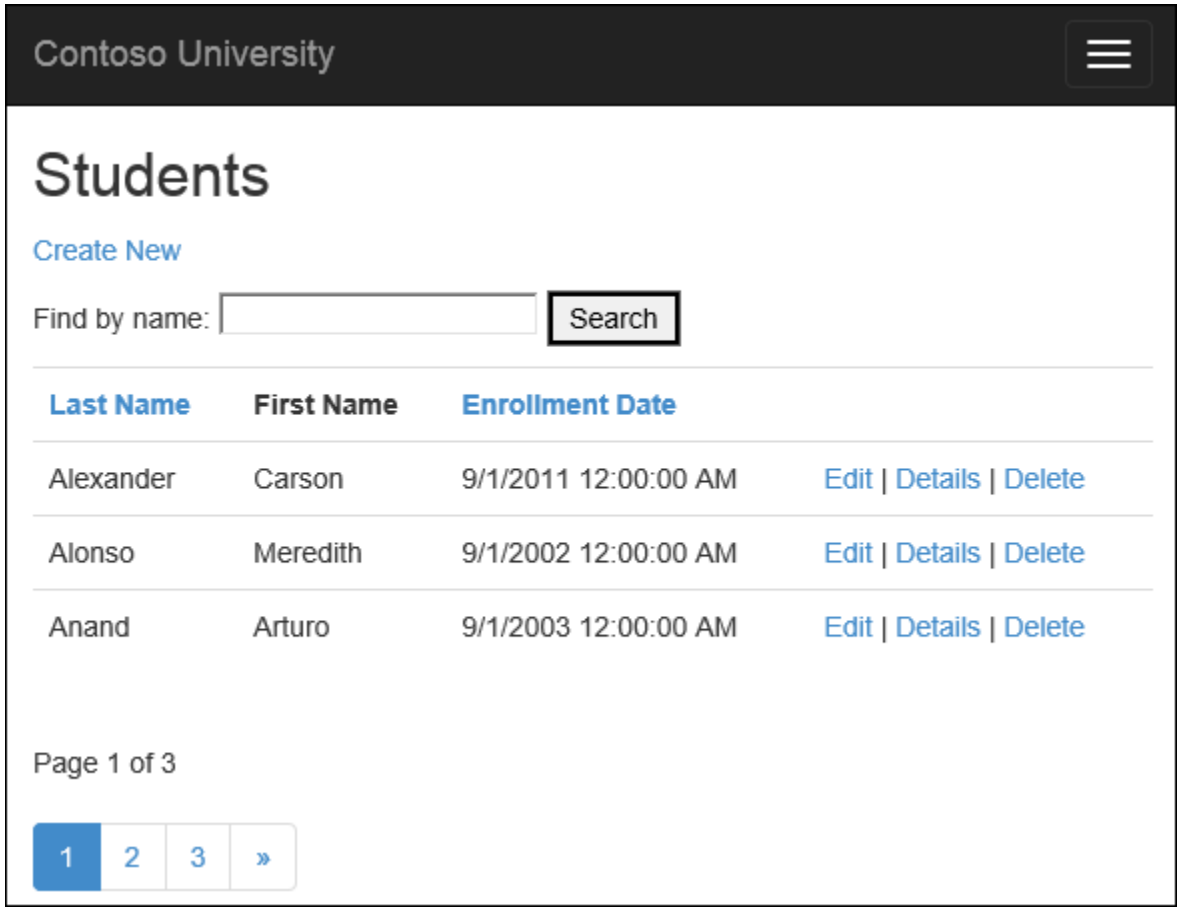
Customize the Data Model by Using Attributes

- The StringLengthAttribute
  - [StringLength(50, ErrorMessage = "First name cannot be longer than 50 characters.")] To ensure the user don't enter more than 50 characters.
- The Column Attribute
  - using System.ComponentModel.DataAnnotations.Schema;
  - [Column("FirstName")]

    public string FirstMidName { get; set; }

Working with Entity Framework

- Basic CRUD Functionality
  - Create
  - Edit / Delete
  - Detail
  - List
- Sorting
- Filtering
- Paging



Where Does Validation Occur?

- On the server
  - ModelState.IsValid property
- On the client
  - Requires jQuery unobtrusive validation

Server-side model validation

- Server side validations are required for ensuring that the received data is accurate and valid
- In MVC Razor, we can validate a model server side by two ways:
  - Explicit Model Validation
  - Model Validation with Data Annotations

Adding Validation to the Model

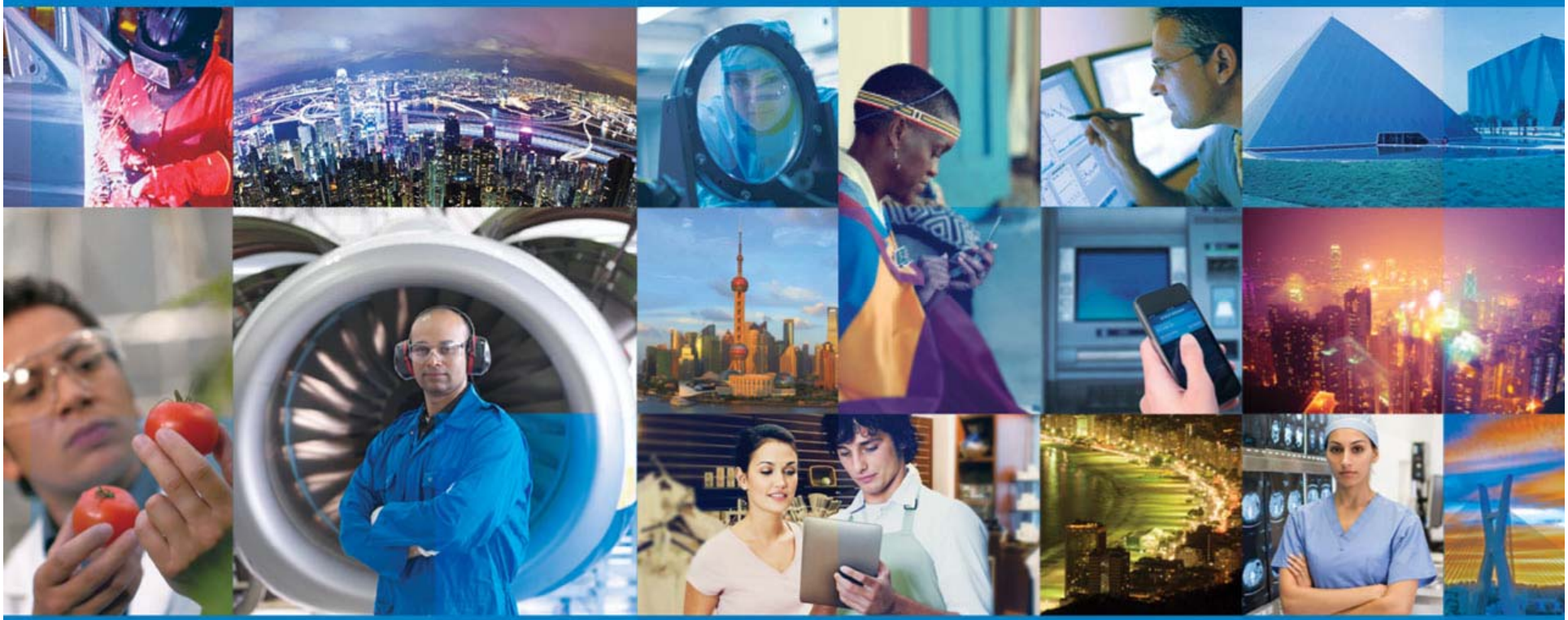
- Adding validation to Model class
- Adding Client-Side Validation

```
<script src="/Scripts/MicrosoftAjax.js" type="text/javascript">
</script>

 <script src="/Scripts/MicrosoftMvcValidation.js"
type="text/javascript"> </script>

Html.EnableClientValidation();
```

Client-Side Validation in MVC

- MVC supports unobtrusive client-side validation.
- Validation rules are
  - Defined using attributes added to the generated HTML elements.
  - Interpreted by the included JavaScript library
  - Uses the attribute values to configure the jQuery Validation library which does the actual validation work

Razor Helpers



Infosys® | Building Tomorrow's Enterprise

Razor helpers

- WebGrid helper
  - Automatically sets up an HTML table to display data
  - Supports different options for formatting
  - Supports paging through data
  - Supports sorting by clicking on column headings
- Chart Helper
  - can display data from arrays , from databases, or from files
- WebMail Helper
  - provides functions for sending email messages using SMTP

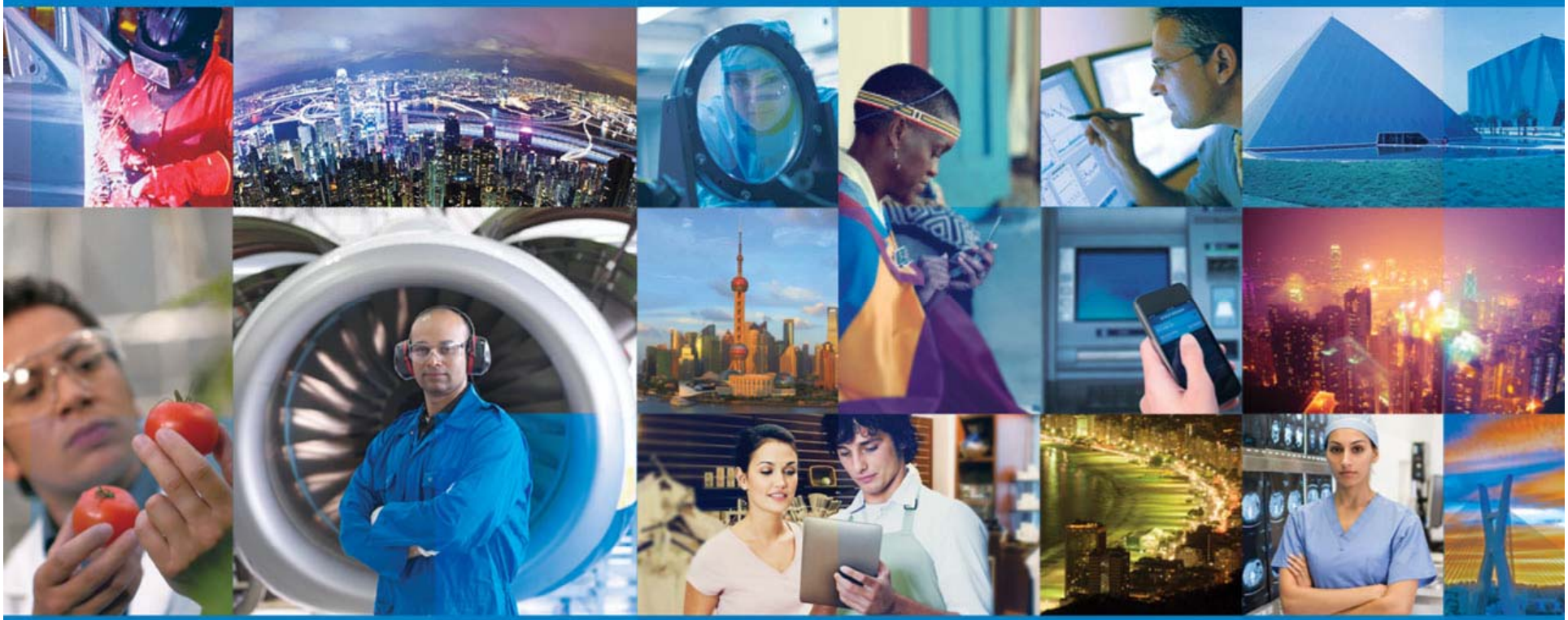
Working with Images

- Using WebImage
  - provides functionality to manage images in a web page
  - lets flip and rotate images
  - Add a Watermark to an Image

Creating Web Grid

- Describe WebGrid to display data on a web page using an HTML table element.
- Renders tabular data in a very simple manner
- Supports for custom formatting of columns, paging, sorting, and asynchronous updates via AJAX
- Properties:
  - Source
  - DefaultSort, RowsPerPage, SelectedFieldName
  - CanPage, CanSort

Unit Testing and Debugging



Infosys® | Building Tomorrow's Enterprise

Testing and Debugging

FirstMVCAPP - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Run
Debug
Playlist
Test Settings
Analyze Code Coverage
Profile Test
Windows

Selected Tests
All Tests Ctrl+R, A
Failed Tests
Not Run Tests
Passed Tests
Repeat Last Run Ctrl+R, L

HomeControllerTest.cs | Index.cshtml | WelcomeController.cs

FirstMVCAPP.Tests.Controllers.HomeControllerTest

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Mvc;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using FirstMVCAPP;
using FirstMVCAPP.Controllers;

namespace FirstMVCAPP.Tests.Controllers
{
    [TestClass]
    0 references
    public class HomeControllerTest
    {
        [TestMethod]
        0 references
        public void Index()
        {
            // Arrange
            HomeController controller = new HomeController();

            // Act
            ViewResult result = controller.Index() as ViewResult;

            // Assert
            Assert.IsNotNull(result);
        }

        [TestMethod]
        0 references
        public void About()
        {
            // Arrange
           // HomeController controller = new HomeController();
```

Solution Explorer

Solution 'FirstMVCAPP' (2 projects)
- FirstMVCAPP
- FirstMVCAPP.Tests
  - Properties
  - References
  - Controllers
    - HomeControllerTest.cs
  - App.config
  - packages.config

Ready Ln 1 Col 1 Ch 1 INS

Unit Testing

FirstMVCAPP - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

HomeControllerTest.cs | Index.cshtml | WelcomeController.cs

FirstMVCAPP.Tests.Controllers.HomeControllerTest | Index()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Mvc;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using FirstMVCAPP;
using FirstMVCAPP.Controllers;

namespace FirstMVCAPP.Tests.Controllers
{
    [TestClass]
    0 references
    public class HomeControllerTest
    {
        [TestMethod]
        0 references
        public void Index()
        {
            // Arrange
            HomeController controller = new HomeController();

            // Act
            ViewResult result = controller.Index() as ViewResult;

            // Assert
            Assert.IsNotNull(result);
        }

        [TestMethod]
        0 references
        public void About()
        {
            // Arrange
           // HomeController controller = new HomeController();
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'FirstMVCAPP' (2 projects)
  - FirstMVCAPP
  - FirstMVCAPP.Tests
    - Properties
    - References
    - ...trollers
    - HomeControllerTest.cs
    - ...config
    - ...kages.config

New Item...	Ctrl+Shift+A
Existing Item...	Shift+Alt+A
New Folder	
Unit Test...	
Load Test...	
Web Performance Test...	
Ordered Test	
Generic Test	
Windows Form...	
User Control...	
Component...	
Class...	Shift+Alt+C

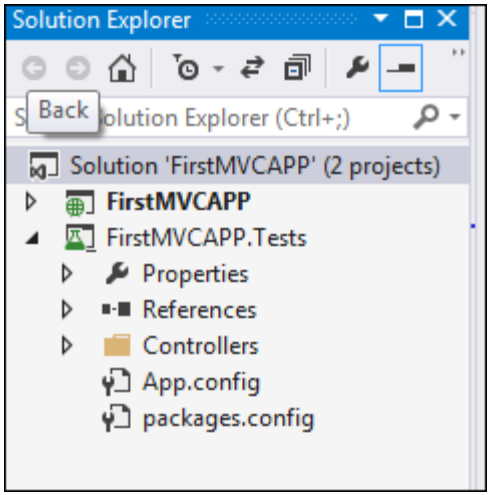
Add	
Scope to This	
New Solution Explorer View	
Show on Code Map	
Exclude From Project	
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Del
Rename	
Open Folder in File Explorer	
Properties	Alt+Enter

This item does not support previewing | Ln 1 | Col 1 | Ch 1 | INS

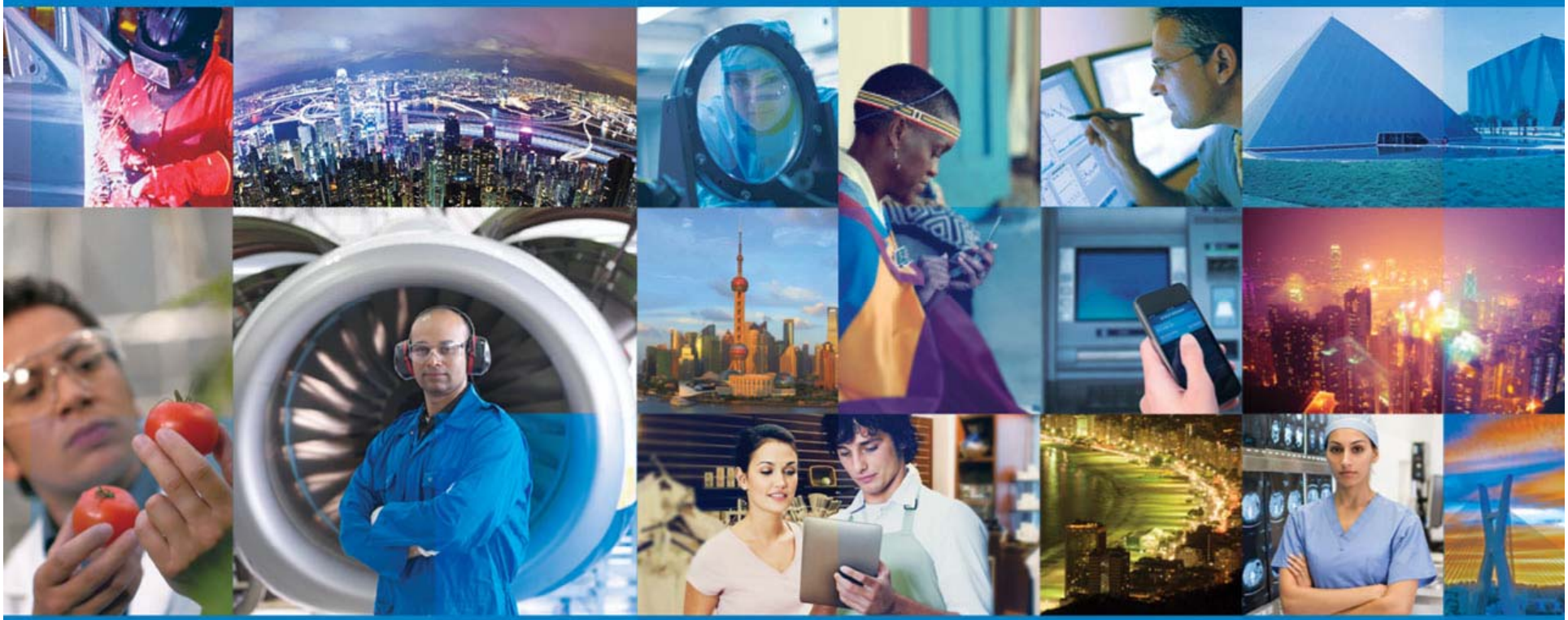
9:06 AM 6/11/2015

Unit Tests for ASP.NET MVC Applications

- Creating the Controller under Test
- Testing the View returned by a Controller
- Testing the View Data returned by a Controller
- Testing the Action Result returned by a Controller



Best Practices



Infosys® | Building Tomorrow's Enterprise

Best Practices

- Create a ViewModel for each view
- ViewModel should not contain presentation logic
- Decorate the action methods with appropriate verbs like Get or Post as applicable.
- Decorate the most used action methods with OutputCache attribute
- Try to keep away domain logic from controller. Controller should only be responsible for
  - Input validation and sanitization.
  - Get view related data from the model.
  - Return the appropriate view or redirect to another appropriate action method

Summary

- We have discussed on
- Introduction to ASP.NET MVC Framework
- Working with Controller
- Introduction to Routing
- Creating Views
- Introduction to Razor Engine
- Working with helper classes
- Working with Action Filters
- Introduction to Model
- Introducing data validation in Model

Summary - Contd

- Deploying ASP.NET MVC applications
- Unit testing MVC applications
- Web Optimization

Thank You

Infosys® | Building Tomorrow's Enterprise