# 4. Developing ASP.NET MVC 4 Controllers

A controller is a class that inherits from **System.Web.Mvc.Controller** They respond to user requests, and Actions respond to user requests and return an **ActionResult** object (view, json etc).

## Responding to User Requests

On receiving a request the following happens:

**MvcHandler** creates a controller factory, usually the **DefaultControllerFactory. The MVCHandler** chooses the controller factory based on the **RequestContext.**

The controller factory creates a Controller object and the **MVCHandler** calls the Execute method on that controller.

The **ControllerActionInvoker** examines the **RequestContext** and determines the action to call on the Controller

The **ControllerActionInvoker** uses a **ModelBinder** to determine the values to be passed to the action as parameters.

The **ControllerActionInvoker** runs the action.

## The User Request

Url typed in or a link. Urls can have route information specifying the name and action of the controller, and an optional Id parameter.

www.mysite.com/**photo/get**/*1?*deleted=false

**Bold** dealt with by **MvcHandlder** to determine which controller and action to use

*Italic* used by **ControllerActionInvoker** as parameter

Highlighted used by **ModelBinder** to look for parameters

You can modify this by creating routes to interpret routes differently.

Need to use the convention MyControllerNameController so that the **DefaultControllerFactory** creates the right controller, otherwise get a 404 error.

**Writing Controller Actions**

Controller hold user interaction logic in Action methods.

## Return Classes

Action methods usually have the ActionResult return type. Can return a range of classes:

* View - View()
* JsonResult – Json()
* HttpError
* FileContentResult – File()
* PartialViewResult
* RedirectToActionResult
* RedirectResult – to specific url
* ContentResult – return text, xml, csv etc

## Child Actions

When an action returns a complete view MVC sends a whole page to the browser. You may only want to send a piece of child content, you can use **ChildContentOnly** to mark that it should only be called from a view and prevent a user calling the view directly.

## Using Parameters

When users request pages they often request data other than page, e.g product to display. The **ControllerActionInvoker** and **DefaultModelBinder** obtain data from requests and pass them as parameters to the controller. The **DefaultModelBinder** looks at the *name* and *type* of the data in the request.

## Passing Information to Views

You can pass models to views using the **View()** helper method. In some cases you may want to augment the view with additional information. You can use ViewBag and ViewData.

**ViewBag** is a dynamic object, part of the base controller class.

**ViewData** Dictionary is used for backward compatibility, view bag is a dynamic wrapper around ViewData Dictionary.

## What are Controller Factories?

A **ControllerFactory** instantiates controllers. An **ActionInvoker** then calls a method on the controller and the **ModelBinder** passes parameters to it.

The **DefaultControllerFactory** class identifies controller by:

* The class scope must be public
* Must not be abstract
* Most not take generic parameters
* Must end with the name Controller

## Creating a Custom Controller Factory

May do this to modify the criteria for selecting controllers or to directly support dependency injection. Need to create a class that implements **IControllerFactory** (CreateController, GetControllerSessionBehaviour, ReleaseController).

Custom controller factories are registered in **Global.asax**

ControllerBuilder.Current.SetControllerFactory(new MyControllerFactory());

## Writing Action Filters

If you need to run code before or after controller action runs, then you can use an action filter. Use for cross cutting concerns (eg caching, logging etc).

Filter Types:

* **Authorize** : IAuthorizationFilter :AuthorizeAttrbuite – runs before other filters
* **Action** : IActionFilter : ActionFilterAttribute – runs before AND after code in action method
* **Result** : IResultFilter : ActionFilterAttribute – runs before And after result is returned
* **Exception** : IExceptionFilter : HandleErrorAttribute – only if method or filter errors

# 5. Developing Asp.NET MVC Views

## Creating Views with Razor Syntax

Views have models passed to them and generate the html.

## Adding Views

Each controller can have multiple views.

By convention and a MVC web application creates all view within the top-level Views folder. Within this folder there is a folder for each controller in the application.

## Differentiating Server Side Code from HTML

The razor view engine looks for the @ symbol to identify server side code. (Referred to as a **Razor Code Expression**)

## Modifying the Interpretation of Code and Content

You may need to change how razor renders, for example you can use @@ as an escape syntax. To fix parsing errors you can use @: which explicitly declares the line as content rather than code. If you want to declare several lines use the <text> tag.

## HTML Encoding

For security reasons when Razor runs server side code it encodes strings as html before rendering it. To render text without encoding use the Html.Raw() helper.

## Features of Razor Syntax

Razor Comments – you can use the @\* delimiter e.g. @\*Comment goes here \*@

Binding Views to Model Classes and Displaying Properties

Views that cannot be bound to a model class are called **dynamic views**. Views that can be bound to a model are called **strongly-typed views**. E.g.

@model MySite.Models.Product

You can also bind to enumerable lists:

@model IEnumerable<MySite.Models.Product>

## Using Dynamic Views

Sometimes you might want to create a view that displays more than one model class. A dynamic view does not include the @model declaration. You have to checked that properties exist before you use them.

## Rendering Accessible HTML

* Do not rely on colour difference to highlight content, e.g. links should be underlined
* Always provide alternative content e.g. alt tags on images
* Use markup and stylesheets to separate presentation from content
  + Avoid using tables to display content
  + Avoid using nested tables, this can become confusing when using text readers
  + Avoid using images that include important text, instead use markup

## Alternate View Engines

Common view engines:

* **Razor** – default view engine
* **ASPX** - web forms view engine
* **NHaml** - .Net version of Haml (ruby on rails) view engine
* **Spark** – similar to static html

## Creating Custom View Engines

You can create your own view engine:

* Create a view class, must implement **IView** and have a **Render()** method
* Create a view engine, should inherit the **VirtualPathProviderViewEngine** and include the **CreateView**() and **CreatePartialViewMethods**()
* Register the custom view engine in global.asax: **ViewEngines.Engines.Add()**

## Using HTML Helpers

Html helpers are method that usually return a string that the can be used to generate the completed web page.

* **@Html.ActionLink()** – can be used to create an a tag with the correct href parameter
* **@Html.Action()** – renders a url without with the a tag (e.g. source of image)
* **@Html.DisplayNameFor()** – renders the display name of the property
* **@Html.DisplayFor()** – renders the value of the property
* **@Html.BeginForm()** – renders for the form element and controller to send data to, use @using code block to ensure form tag is closed
* **@Html.LabelFor() –** similar to DisplayNameFor but uses label tag
* **@Html.EditorFor() –** renders appropriate input type for property
  + **Text box**
  + **Multi-line text box**
  + **Check box**
* **@Html.ValidationSummary()** – renders summary of invalid data
* **@Html.ValidationMessageFor()** – used to render validation message next to a control

## Creating Partial Views

Can use partial views to render similar html in different locations. When using the Add View dialog you select the **Create as Partial View** checkbox. By contention partial views are prefixed with a \_ character. They are often created in the Shared folder so they can be accessed by many controllers.

Partial views can be strongly typed or dynamic.

## Using Partial Views

Can reused partial views using @Html.Partial and @Html.Action

Use **Html.Partial** to render a partial view within another view file. You can pass a model – the same model that the parent view uses.

In some cases you may want to use a model object different from the parent, in such cases you can use **Html.Action()** this can create an instance of a model class by calling a controller.

6. Testing and Debugging ASP.Net MVC Applications

## Unit Testing MVC Components

A unit test instantiates a components you have written, calls one aspect and checks it responds correctly.

## Why Perform Unit Tests?

3 Types of test

* **Unit test** – tests small aspect of functionality
* **Integration tests** – test how two or more components work together
* **Acceptance test** – focus on requirements for stakeholders to accept system

## What is a unit test?

A procedure that verifies a specific aspect of functionality. You can assemble many unit tests into a test fixture, ie all tests for a class. 3 Phases to a unit test: **Arrange**, **Act**, **Assert**.

## How do unit tests help diagnose bugs?

Because they test a small aspect of code, it is easier to diagnose the problem when a test fails. Test should check the code you write, not the infrastructure you rely on e.g. Databases etc.

## Automated unit tests

Test should be able to rerun quickly and easily. Visual studio does this automatically when debugging.

## Principles of Test Driven Development

* Write the test
* Pass the test
* Refactor

## TDD Principles

* Write the tests before the code
* Move in small steps
* Only write enough code to pass the test

## Writing loosely coupled MVC Components

Loosely coupled components allow you to replace components with others. They are useful for unit testing as classes that deal with databases can be replaced with test doubles or fakes.

Use interfaces for loose coupling.

## Writing Unit Tests for MVC Components

Separation of concerns helps with unit testing. You can add a unit test project template to your solution and add references to your MVC project, you also need a reference to Microsoft.Web.MVC.

[TestClass] and [TestMethod] annotations.

Repository interface and fake repositories

## Using IoC Containers

Can use an IoC container to setup which implementations are used.

## Using Mocking Frameworks

# 6. Testing and Debugging

## Logging exceptions

Exceptions during development can be investigated during debugging. In the real word exceptions can occur and you may want to log them so that can be investigated and fixed. Consider where to log exceptions, xml, database, emails, etc.

## Where to write error logging code

Should choose an approach that will be written once and work across the application. Instead you could create a custom base controller, write the error logging code in an overridden **OnException** method.

## Using 3rd Party Logging Tools

Common tool is Elmah can log to xml or databases and web pages where you can view exceptions.

## Health Monitoring

You can use health monitoring to log exceptions and record other events such as application starts and stops, logon etc. You can customise the category of events that are recorded:

* **Application Lifetime** – start, stop
* **Audit Events** – Login events
* **Error Events**
* **Request Processing Events** – as controllers etc receive requests
* **Heartbeats** – confirm the application is running

Providers:

* EventLogWebEventProvider
* SQLWebEventProvider
* WMIWebEventProvider
* SimpleMailWebEventProvider
* TemlatedWebEventProbider
* TraceWebEventProvider

Can configure health monitoring by using the health monitoring element in the web.config

If you want to use the sql provider you can use the asp\_regsql command to run scripts to create the table.

# 7. Structuring ASP.Net MVC4 Web Applications

## Analysing Information Architecture

Models impose a logical structure on an application, however you also need to consider users who may expect a hierarchy.

## What is IA?

When the information your application manages is complex you need to ensure the application will be useable. IA is a logical structure for the objects your web application manages, so that users can find content quickly.

## Presenting a hierarchy in navigation controls

* Site menus
* Tree views
* Breadcrumb trails

## Presenting a hierarchy in URLS

Many applications have long URLS with large query strings, URLS can be plain and comprehensible. URLs in MVC follow a default pattern: controller/action/(id)

URLs can be made to work with the information architecture by configuring routes.

## What is Search Engine Optimisation

Users tend to find sites by search engines, SEO ensures that more people visit your application. Best practice:

* Ensure each webpage has a meaningful title
* Use metaname keywords in the header
* Use metaname description
* Choose a domain name that includes one of your keywords
* Ensure that navigation controls allow web bots to search the whole application
* Ensure URLS do not include guids and long text

## Configuring Routes

A route is an object that is passed a URL and determines the controller that must be called (incoming routes). Outgoing routes – used by html helpers to create links etc.

Asp.Net routes are used to pass the url from the browser and to formulate urls in web page links – incoming urls. E.g. Html.ActionLink and Html.Action - outgoing urls.

## Default route

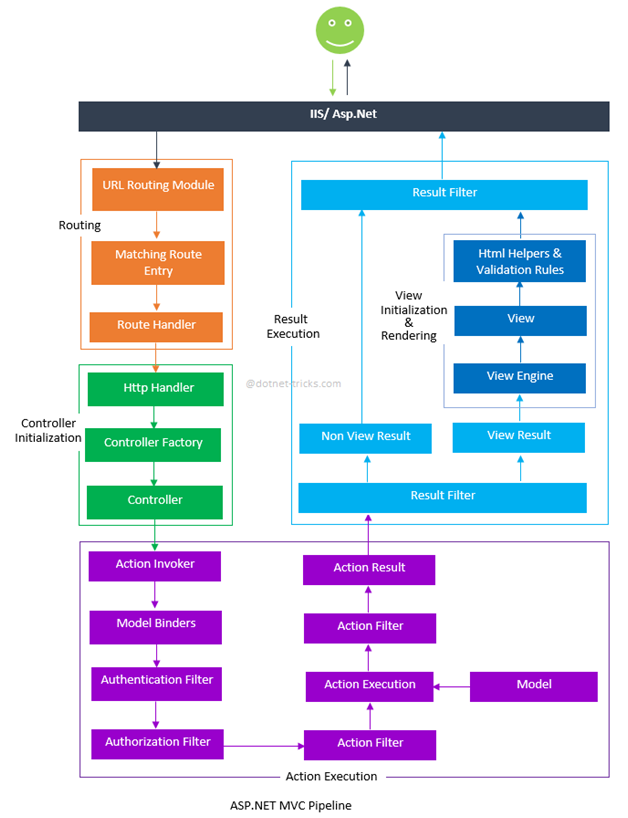
/controller/action/(id)

## Custom routes

Developers add custom routes to make urls easier for users to understand. You should consider what users know to create routes to accept the information.

## ControllerFactory and Routes

ControllerFactory creates the controller and ActionInvoker calls the right method, both use the routing table.



## Adding and Configuring Routes

Routes are stored in the **route table object**. You can add routes by calling the **MapRoot** method. RouteConfig is called in the **ApplicationStart** method in the **global.asax**

Properties of a route:

* **Name** – string name of the route, not used in matching
* **Url** – url pattern compared to a request, can specifiy segment variables by using braces
* **Constraints** – can use regular expressions
* **Defaults** – can assign default values, used when request does not specify segment variables

## Creating Custom Routes

Routes are allocated by the routing engine in order they are added, if the route does not match it evaluates the next. Add the most specific routes first, routes with constraints should be added before routes without constraints.

## Using routes to pass parameters

Segment values with names other than controller and action have no special names and are passed to actions and can be used by the routedata collection or by a model binder.

Optional parameters

## Unit tests and routes

Create a HttpContext test double, then use this in tests.

* **Arrange** – set the **request url** for the test, create a new route collection and call **RouteConfig.RegisterRoutes()**
* **Act** – tests the routes by calling **GetRouteData()** on the route collection
* **Assert** – use the **RouteData.Values** collection to check that controller, action and other values are assigned correctly.

## Creating a Navigation Structure

Navigation controls should have a clear context and obvious next step.

## MVC Sitemap Provider

This is a component that stores the hierarchy of the application. The MVCSiteMap provider is designed to work with controllers and applications and is configured by xml, it also providers helpers that can be used to render menus (3rd party component).

# 8. Adding Styles to ASP.NET MVC4 Web Applications

Can usea layout e.g. **Layout.cshtml** to define layout (template view) for the whole application,to link a layout you use the Layout directive in a view:

@{

Layout = “~/Views/Shared/Layout.cshtml”

}

You can use **@RenderBody()** to render where the content of a view goes.

On the layout you can use **@RenderSection()** which takes a param require e.g.

@RenderSection(“MySection”, false)

Then on the view you use the section directive:

@Section MySection

{

<!-- Stuff here -->

}

You can define the layout for an application or section by using the **\_ViewStart** file, which runs before other views.

## Applying CSS Styles

External stylesheets, style blocks and inline css.

## Expression Blend

Blah blah blah graphical interface for designers

## Importing Styles

.css files link in html. Using css classes, id selectors.

## Creating an adaptive User Interface

Website adapts depending on device. Html5 viewport attribute helps eliminate need to reduce size of layout.

<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=2.0, user-scalable=yes" />

Has width and height attributes.

## Media Queries

Html5 includes css media queries that begin with **@media** e.g.

@media only screen and (max-width: 500px)

{

/\*css here\*/

}

You can also apply media queries in the link element:

<link … media=” only screen and (max-width: 500px)” />

Properties on media queries:

* Width – inc browser
* Height – inc browser
* Device-width – width of whole screen device
* Device-height – height of whole screen device
* Orientation – landscape, portrait based on device width and height
* Aspect-ratio
* Device-aspect-ratio
* Color
* Color-Index
* Monochrome
* Resolution
* Scan
* Grid

## MVC4 templates and mobile specific views

You can use a different set of view files for mobiles and use a different view, using the convention: myView.mobile.cshtml you can also support different mobile devices e.g. myView.ios.cshtml

## JQuery Mobile

Helps to add responsive elements to your web application and to ensure that older devices see usable controls.

# 9. Building Responsive Pages

## Using AJAX and partial page updates

You can update sections without reloading the entire using **Ajax.ActionLink**

Need to return a **PartialView** from the controller being used.

## Ajax.ActionLink Helper

Can replace or insert code e.g.

Ajax.ActionLink(“Name”, “ActionName”, new AjaxOptions { HttpMethod = “POST”, UpdateTargetId= “someDiv”, InserionMode = InsertionMode.Replace}

## Implementing a Caching Strategy

Can use caching to reduce the calls made to the database and make the application more responsive, reduces load on server. Before implementing caching you should consider if relevant to your application.

* Output cache
* Data cache
* Http cache

Content from a cache may be outdated from that held in the database.

## Output Cache

Allow ASP.NET to store the rendered content of a page in memory. This avoids the execution of code on the server. Use the **[OutputCache]** attribute:

**[OutputCache(duration)]**

The duration controls the period of time in **seconds**. By default it only stores one copy of a rendered view, you can add the **VaryByParam** property:

**[OutputCache(60, VaryByParam = “Id”)]**

You can also use the **VaryByCustom** property eg **browser**.

## DataCache

Web applications usually depend on content in the database. You can implement a data cache to avoid loading data for every request using **MemoryCache**. Using the memory cache:

System.Data.DataTable dtCustomer = System.Runtime.MemoryCache.Default.AddOrGetExisting(“CustomerData”, this.GetCustomerData(), System.DateTime.Now.AddHours(1));

You can use AddOrGetData to use a cache with one line of code.

## HttpCache

HttpCacheing can be implemented in browser cache and proxy cache. Storing data in local cache helps reduce calls to the server. If the content is updated in the server the browser will download the updated content.

Preventing caching. Caching can sometimes cause issues, you can use a **cache control header** to tell the browser how to handle the cache using HttpCachePolicy.SetCacheability e.g.

Response.Cache.SetCacheability(HttpCacheability.Private);

To prevent cacheing you can use HttpCacheability.NoCache

## 10. Using Javascript and jQuery for responsive MVC 4 Web Applications

Blah blah javscript files.

## Bundling and Minification

Bundling helps combine multiple javascript files into a single file. Minificiation reduces the size of the file:

var bundles = new BundlesCollection;

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

In the view you need **@Scripts.Render**(“~/bundles/jquery”)

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

In the web.config NuGet adds the namespace.

## Calling a web service using jQuery

$.ajax({

type: “POST”,

dataType: “json”,

url: “Service.asmx”,

data: {id : 1},

contentType: “application/json; charset=utf-8”,

success: function(data){

//doStuff

},

failure: function(){

//oh noes

}

});

## Introduction to jQuery UI

Accordion, AutoComplete, etc.

Effects .

jQuery Utility – helps align an element in relation to another.

# 11. Controlling Access

## Local Authentication Providers

* **ActiveDirectoryMembershipProvider** – use Active Directory as membership and role repository
* **SqlMembershipProvider** – works with specific tables schema, can generate using **aspnet\_regdb.exe**
* **SimpleMembershipProvider –** works with various version of SQL, requires only 3 parms table name, user Id and user name
* **UniversalProvider –** works with any database that EntityFramework supports, only works with schema designed by Microsoft.

The **SimpleMemebershipProvider** and **UniversalProvider** support OAuth.

## Claims-Based Authentication

Model that facilitates single sign on (SSO), receive a claim when you login to a centralized authentication. The claim helps authentication system identify users. Claims based facilitates:

* Authenticating users
* Storing user account information and passwords
* Checking enterprise directories for user information

## Federated Authentication

Federations rely on claims based authentication. Windows Identity Framework supports federations by the **WSFederationAuthenticationModule**.

FederatedPassiveSignIn - can embed in the login page

PassiveRedirect – when you want to redirect to **Secure Token Service** without using a login page, need to add **WSFederationAuthenticationModule** in the HTTP pipline.

## Restricting Access to Resources

Using **[Authorize]** attribute.

## Assigning Roles and Membership

Roles providers in ASP.Net 4.5

* **ActiveDirectoryRoleProvider** – use active directory as management for roles
* **SQLRoleProvider** – provides roles using a specific table schema
* **WindowsTokenRoleProvider** – uses windows authentication token
* **SimpleRoleProvider** – new generation that works with various versions of sql, allows table to be defined by developer
* **UniversalProviders** – database agonistic, works with any database that entity framework supports

## Adding user accounts to roles

You can load data into the table or using the addUserToRole function.

## Building a Custom Role Provider

You can implement your own provider, that inherits from the **RoleProvider** class. Need to implement the GetRolesForUser function:

Public override string[] GetRolesForUser(string username)

{

//do stuff

}

You then need to apply the custom role provider to the web.config.

## Providing Membership Services

Building a Custom Membership provider, inherit from **SimpleMembershipProvider** then override the ValidateUser(string username, string password).

# 12. Building a Resilient ASP.NET MVC4 Application

## Cross Site Scripting XSS

Malicious insertion of scripts using authentication of user without their knowledge. Use **Ajax.JavaScriptStringEncode** to prevent xss where data coming in from the query string.

## Using the AntiXss Library

Can bring this library in to increase protected by encoding and decoding html. **@Encode.JavascriptEncode**

## Cross-Site Request Forgery CSRF

Attack where you open a URL in a web browser without knowing you are allowing hackers to make changes while you are logged in.

* Don’t use GET for state modification
* Ensure request does not replay if attacker uses javascript
* Use **[ValidateAntiForgory]** and **@Html.AntiForgeryToken**

## SQL Injection Attacks

User input used to create dynamic SQL. You should:

* Validate user input
* Avoid string concatenation when writing queries
* Store sensitive data in encrypted format
* Ensure the application does not used or access the database with administrative privileges

## Disabling Attack Protection

**Request Validation** is an ASP.NET feature that detects dangerous content. You may need to disable this in some scenarios, for example where html editors are used. You can disable request validation in the web.config or on a page.

Site – web.config:

**<httpRuntime requestValidationMode=”2.0”>**

Controller:

**[ValidateInput(false)]**

Model:

**[AllowHtml]**

## Secure Socket Layers

Protected content trasmistted between server and client and reassure user that site is valid. You can use **[RequireHttps]** on controllers/actions it redirects users to the secure link if using http.

## State Management

You may need to retain information across requests e.g. work on an order relevant to a customer. Http is stateless so need other techniques.

## Client-side session management:

* Hidden fields
* Cookies
* Query string

## Server-side techniques:

* Application state
* Session state
* TempData
* Profile properties
* Database support

**TempData** – uses the **session** variable to store data.

Session state element in the web.config to define where session state data should be stored e.g.

**InProc** – in the asp.net process

**StateServer** – on a dedicated state server (not in IIS)

**SQLServer** – use database

**Custom**

**Off**

## StateServer mode

Need to install the asp.net state service on the server (**aspnet\_state.exe**) before using the statesever mode. In the mode of the web.config set the session state **mode** to **StateServer** then set the **StateConnectionString** to tcp= <<servername>>:42424

## SqlServer mode

Set mode to SqlServer set the StateConnectionString to your SQL Server. On the SQL server run **aspnet\_regsql.exe**

## Scaling State Storage

State server and SQLMode allow multiple servers to access state storage. You can use partitioning to have multiple state servers to improve scaling. You need to implement **IPartitionResolver** to apply logic for partitioning and configure this in web.config:

<sessionState mode=”SQLServer” partitionResolverType=”MyPartitionResolverClass” />

## 13. Using Windows Azure Web Services

Azure is cool.

## Azure Services

Execution models:

* Virtual Machines
* Web applications (web sites)
* Cloud services

Identity Services

* Windows Azure Active Directory

Data Management

* SQLAzure
* Table storage
* Blob storage

Business Analytics

* SSRS Reports
* Hadoop

Networking Services

Messaging

* Service Bus

Media Management

* CDN

## Benefits of hosting *services* on Azure:

* Can run applications as services (do not require admin access, don’t need interaction with windows etc)

**Web Role** and **Worker Role**. Configuration file to configure the application. Do not retain state data, can use other services for storing state data.

## Benefits of hosting *applications* on Azure

Similar to hosting services, however applications provide full access to IIS instance. Can host asp.net and other applications.

* Automates process of deploying
* Simplifies scaling process

## Windows Azure Storage Services

* Azure (Blob) storage
* SQL Azure
  + Increase availability
  + Reduce hardware costs

## The Life Cycle of a Service

**RoleEntryPoint** class – add logic to services

* OnStart
* Run
* OnStop

## Deploying a Web Service

* ServiceDefinition. cscfg – define roles
* ServiceConfiguration.cscfg – configure instance e.g. number of instances

Visual studio creates a package .cspkg and .csfg for deployment.

## Debugging Windows Azure

* **Diagonistic log** – import moduleName=”Diagnoistics”
* **IntelliTrace**
* **RemoteDesktop**

# 14. Implementing Web APIs in WEP.Net MVC4 Web Applications

## What is a Web API?

Implement REST to solve interoperability issues. Help external systems use business logic in your application. APIs use urls in requests e.g. api/Customers/1

* Reduce processing power needed to create requests
* Reduce the amount of data sent

## Routing in Web API

When creating a new project using Web API template it has a default routing rule:

routes.MapRoute(

name: “API Default”,

routeTemplate: “api/{controller}/{id}”,

defaults: new { id = RouteParameter.Optional }

)

**HttpVerbs**

* GET
* POST
* PUT
* DELETE

Have attributes for these verbs so you can override the action naming convention.

**[AcceptVerbs]** Allows you to specify verbs to actions.

**[ActionName]** allows you to specify the action name to be used in routing

## Media Formatters

Used to serialize information. You can create your own media formatter by inheriting from **BufferedMediaTypeFormatter**.

## Supporting Multiple Operations

You may want to support different version of your api. E.g. v1 and v2.

**[NonAction]** stop method being called via api

## Calling Web API from Server Side code

WebApiClient NuGet package

Windows Phone – use JSON.Net library for calling WebAPI.

# 15. Handling Requests in ASP.Net MVC

## What is a HttpModule?

Program that runs application login on a webpage before asp.net renders a web page. Requests passed to isapi\_aspnet.dll

Then request passes through **HttpApplication** and **HttpModule** library

Modules provide:

* Security
* Logging
* Custom headers and footers

.e.g OutputCache, Session, WindowsAuthentication, FormsAuthetication, PassportAuthentication, UrlAuthorization, FileAuthorization, DefaultAuthorization.

Modules apply to all http requests received.

## Creating Http Modules

Add the a custom module that inherits from **IHttpModule** need to implement **ModuleName** and **Init**

After you complete the development of the module you need to register it in the **web.config**

<system.WebServer>

<httpModules>

<add name=”CustomModule” type=”CustomModule” />

## What is a Http Handler?

Can handle ashx files from a specific server side based page:

* Asp.net web page handler aspx
* Web service handler .asmx
* Generic web handler.ashx default handler that does not have a user interface includes @WebHandler directive
* Trace event handler .axd

## Using Web Sockets

Http does not cater for realtime updates from the server.

Websockets library:

In web api e.g.

HttpContext.Current.AcceptWebSocketRequest(new ChatWebSocketHandler());

Then class:

Class ChatWebSocketHandler : WebSocketHandler

{

public override void OnOpen()

{

}

public override void OnMessage(string message)

{

}

}

You need to add to the **web.config**:

<appSettings>

<add key=”aspnet:UseTaskFriendlySynchorizationContext” value=”true” />

On client:

websocket = new WebSocket(‘url);

websocket.onopen = function(){};

websokcet.onmessage = function(){};

websocket.send(‘hello’);

websocket.close();

## SinglarR

# 16. Deploying a Web Application

Asp.Net MVC 4 Dependencies

* ASP.Net 4.0 Common Runtime Language CLR
* Asp.Net MVC4 Runtime
* Database Server
* Entity Framework
* Membership Providers

## Deploying to web servers

Create and configure IIS application.

## IIS Application Pools

In IIS applications are run within a context known as an application pool. Applications that run the same pool share the same resources:

* Worker process
* Memory
* Identity

To improve reliability use a separate application pool for each application.

## Deploying

You need to copy the application files to the server, using Visual Studio or other tools such as FTP.

## Deploying Web Applications to Multi-Server Farms

Group of two or more servers.

* Create app pools of each server
* Create a matching machine key in web.config

## Deploying Web Applications on Windows Azure

* Download publishing profile
* Publish using visual studio

## Windows Azure Web Applications and Windows Azure Cloud Services

Can deploy application as cloud service roles, need Windows Azure SDK.

## Deploying MVC4 Application

* Remove debug attribute from web.config from web.release.config

## Using Bin Deploy

**Add deployable dependencies –** deploy dependencies on server

## Using Visual Studio 2012 Deployment Tools

* File share
* Ftp
* Web Deploy