ASP.NET Training Materials

# Claims-based identity

<https://msdn.microsoft.com/en-us/library/hh873308.aspx>

* the user identity is represented in your application as a set of claims
* an external identity is configured as a trusted source
* single sign-on is easier to achieve
* your application is no longer responsible for authenticating users, storing passwords, etc.

# Identity

* Based on Windows identity foundation (**WIF**)

Claim

* Claims vs. attributes: attribs are more commonly used in enterprise directories, and an application would normally lookup a user attrib
* In this case, a user delivers or issues a claim to your application, which then examines the claim
* WIF contains a Claim type, and an Issuer property

# Interview Questions

## What is MVC

An architecture pattern which separates representation and user interaction:

* Model
  + Represents the real world object and provides data to the view
* View
  + All the UI and front-end components of the application
* Controller
  + Acts as an interface between Model and View
  + It handles the user request, and loads the appropriate model and view

## Explain the MVC Life Cycle

Request Object:

1. **Fill the route –** On the first request, fill out the route table in global.asax
   1. public static void RegisterRoutes(RouteCollection routes)
2. **Fetch the route** – “UrlRoutingModule” searches the route table to create the “RouteData” object, which contains the **controller and action** to invoke.
3. **Create Request Context**– The RouteData object is used to create the ***RequestContext*** object
4. **Create Controller instance**– the request object is sent to the MvcHandler instance. MvcHandler creates the controller and passes the a *ControllerContext*, and executes.
5. **Action Execution -** the controller’s ***ActionInvoker*** determines which action to call
6. **View Result –** The action method receives input, prepares the response data, then executes the result by returning a result type (RouteData object, RedirectToRouteResult, ***RedirectResult***, ContentResult, ***JsonResult***, FileResult, EmptyResult)
7. **View Engine –** Select the view engine to render the result (Razor and WebForm view engines by default). It is handled by the ***IViewEngine*** interface.

## How to pass data from controller to the view?

(see ViewData details below)

* Use ViewData (only from controller to view); derived from ***ViewDataDictionary***

## How to pass data between controllers ?

* Use TempData
* Derived from TempDataDictionary, and stored short-term in the session
* Available until target is fully loaded

## What is the difference between each version of MVC 2, 3 , 4, 5 and 6?

* MVC 6
  + MVC and Api has been merged
  + DI is built in
  + Side by side – deploy the runtime and framework with your application
  + No need to recompile changes each time, save and refresh browser
* MVC 5
  + Introduced One Asp.Net, combining Web Forms, MVC, Web Api, etc.
  + Attribute based routing
  + Identity
  + Bootstrap with mvc template
  + Auth filter
  + Filter overrides
* MVC 4
  + Web Api

## Can we map multiple URL’s to the same action?

Yes, you can, you just need to make two entries with different key names and specify the same controller and action.

## Explain Attribute based routing

We can define URI in the controller, using the URI resource name – ex/ api/Users/about

We can decorate the controllers with the Route[] attribute:

[Route("Users/about")]

public ActionResult GotoAbout()

{

return View();

}

\*\*\* Previously (MVC 4) routing rules would be set in RouteConfig.cs and use the standard conventions (i.e. {controller}/{action}/{id} ) .

## How to pass strongly-typed data between controller and view ?

Change the @ Page declaration of the view: <%@ Page Inherits="ViewPage<Product>" %>

## Where is the route mapping code written?

In RouteConfig.cs, and registered in global.asax app start event

(<http://www.webdevelopmenthelp.net/2014/03/routing-in-asp-net-mvc.html> )

*Global.asax* file.

*protected void Application\_Start()*  
*{*  
*01.    AreaRegistration.RegisterAllAreas();*  
*02.    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);*  
***03.   RouteConfig.RegisterRoutes(RouteTable.Routes);***  
*04.    BundleConfig.RegisterBundles(BundleTable.Bundles);*  
*}*

*RouteConfig.cs*

*public static void RegisterRoutes(RouteCollection routes)*  
*{*  
*01.   routes.IgnoreRoute(“{resource}.axd/{\*pathInfo}”);*  
*02.****03.******routes.MapRoute(***  
***04.      name: “Default”,***  
***05.      url: “{controller}/{action}/{id}”,***  
***06.      defaults: new { controller = “Home”, action = “Index”, id = UrlParameter.Optional }***  
***07.      );***  
*}*

# Glossary of terms

## ActionFilter

* Used to implement logic pre-action/post-action behaviour to a controller action
* 4 types (executed in this order)
  + Authorization filter
  + Action filter
    - [OutputCache(Duration=20)] (20 seconds)
    - [HandlerError]
    - [Authorize]
  + Result filter
    - Implements the IResultFilter and wraps execution of ActionResult object.
    - Two methods: OnResultExecuting (runs before ActionResult is executed), OnResultExecuted (runs after)
  + Exception filter

## Exception

* The OnException method can be overridden
* OnException is always invoked when an ***unhandled*** exception occurrs

protected override void **OnException**(ExceptionContext filterContext)

{

   ...

}

## HTML Helpers

* Standard helpers
  + Html.AntiForgeryToken()
    - Prevents against CSRF (cross-site req forgery)
    - Stores cryptographic random key in hidden form field
    - Stored in cookie and compared against hidden form field each time
    - Random key changes with each http request [ValidateAntiForgeryToken]
* Custom Html Helpers (two ways)
  + Extend the ***HtmlHelper*** class
    - Create a new namespace and class, then add the namespace to web.config
  + Create a static method

## Partial View

* Allows you to render a view inside a parent view
* It is a reusable view (like a user control) which is embedded inside another view
* View ex: 🡪 <% Html.RenderPartial("MyView"); %>
* Controller ex: 🡪 return PartialView(“..\myPartial.cshtml”, myModel);

## Session

* Property of Controller class, type is HttpSessionStateBase
* Used to pass data within the application, and has a default expire time of 20 min
* Session is valid for all request
* Also requires typecasting

## TempData

* Used to transfer data from one request to another (redirects)
* A dictionary derived from TempDataDictionary, and stored short-term in the session
* A property of ControllerBase class
* Valid only until the target view is fully loaded
* Type casting is required to get data
* Best to store one-time validation and error messages; to store data, call the keep() method within current action method

ViewData["FeaturedProduct"] = featuredProduct;

ViewBag.Product = featuredProduct;

TempData["FeaturedProduct"] = featuredProduct;

## ViewBag

See above: “How to pass data from controller to the view?”

* Also used to pass data **from controller to view**
* Takes advantage of new dynamic features of c# 4.0
* A *wrapper* around ViewData
* It’s a property of ControllerBase
* ***Does not require typecasting for getting data (\*\*\* different from ViewData)***
* Only valid for current request; becomes null on redirect

## ViewData

* **Pass data from controller to view**
* It is a ***dynamic object*** (no predefined properties)
* ViewData is a dictionary which contain key-value pairs

public ActionResult Index()

{

IList<Student> studentList = new List<Student>();

studentList.Add(new Student(){ StudentName = "Bill" });

studentList.Add(new Student(){ StudentName = "Steve" });

studentList.Add(new Student(){ StudentName = "Ram" });

***ViewData["students"]*** = studentList;

return View();

}

* ViewData and ViewBag use the same dictionary internally, so you cannot use the same keys in both (mvc throws an exception)
* You must type cast the value
* Valid only during the current request; it becomes null at a redirect
* derived from ***ViewDataDictionary***
* It’s a property of ControllerBase class