* Introduction
  + Understanding MVC pipeline
  + ASP.NET Life Cycle
    - Begin request
    - Resolve request cache
    - Map request handler
    - Acquire request state
    - Request handler execute(controllers & actions)
    - Update request cache
    - Log request
    - End request
  + MVC is supported by ASP.NET platform
* Who is this course for
  + Working knowledge of MVC
  + Concept more than code
  + Existing developers looking for deeper understanding of MVC
  + Developers migrating from another
  + Developers who started their .NET career with MVC
  + Never exposed to the larger ASP.NET platform
* Defining the Request Life Cycle
  + Step/events used to handle request or change in application
  + MVC life cycles
    - Application life cycle
    - Request life cycle
  + Application life cycle
    - Application starts running till it stops
    - Application start and end events in startup file
  + Request life cycle
    - https request is handled by application
* Overview of the MVC Request …
  + Request -> Routing -> Controller Initialization -> Action Execution -> Result Execution -> View Engine -> Result Execution -> Response
  + Routing
    - URL routing module: matching incoming url to routes we define in app
    - MVC Route Handler: retrieves instance of mvc httphandler
    - MVC HttpHandler: initializing and executing controller
  + Controller initialization
    - Controller Factory
    - Activator
    - Dependency Resolution
  + Action Execution
    - Model Binding
    - Action Filters(Action Executing)
    - Action Execution
    - Action Filters(Action Executed)
    - Action Result
  + Result Execution
    - Result filter(result executing)
    - Invoke action result
  + View Engine
    - Find and render view
    - will be skipped if not returning view
  + Result Execution
    - Result filter(result executed)
  + Modules are dot net components that can hook into application life cycle and add functionality
* Webforms and MVC
  + ASP.NET webforms has its own lifecycle
  + Handling a request
    - To a web form, general map to file on disk
    - For MVC, maps to controller action method
  + Both implemented through Http Handler
  + ASP.NET Platform
    - One platform, multiple implementations
    - Http Handler
      * MVC Framework
      * Web forms
      * Custom Implementation
* Summary
* Introduction
* Understanding Application Sta..
  + MVCApplication in Global.asax files
    - Inherits from HttpApplication class
    - Life cycle events get inherited
  + Application start
    - An event fires when first request is received
    - Can be used to run initial configuration code
    - Registers routes
    - Ex
    - protected void Application\_Start()
    - {
    - AreaRegistration.RegisterAllAreas();
    - ….
    - RouteConfig.RegisterRoutes(RouteTable.Routes);
    - …
    - }
  + Registering Routes
    - Adds routes we define to static collection on route table class
    - Collection of routes url routing module will try to match to incoming urls
    - Each routes need an associated route handler class
      * Ex) defaults: new { controller = “Home”, action = “Index”, id = UrlParameter.Optional }
    - Route handler provide asp.net with an http handler that will process incoming request after being matched to a route
    - Routes are registered before any other lifecycle event happens
    - Ex)
    - public static void RegisterRoutes(RouteCollection routes)
    - {
    - routes.MapRoute(
    - name: “Default”,
    - url: “{controller}/{action}/{id}”
    - defaults: new { controller = “Home”, action = “Index”, id = UrlParameter.Optional }
    - );
    - }
  + Application end
    - Event that fires when application ends
    - Not guaranteed to fire when application crashes
    - Not a great option for handling application errors
* Demo - Application Start and E..
  + MapRoute(): assign routes to route table collection
    - Route table collection will be examined by url routing module
    - Behind the scenes mvc creates a new route and mvc handler and adds them to the collection
  + Every route needs an associated route handler class to go with it
    - Purpose is to retrieve the right http handler for request
  + Http handler
    - Is what will execute to generate response
  + MVC is an open source project, you can check the source code online
  + You can do it yourself
  + Ex instead of using routes.MapRoute())
    - Route myRoute = new Route(“{controller}/{action}/{id}”, new RouteValueDictionary{ {“controller”, “Home”}, {“action”, “Index”}, {“id”: “1” }},
    - New MvcRouteHandler());
    - routes.Add(myRoute);
* Configurations with the PreAp..
  + PreApplicationStart
    - Another option for running initial configuration code
    - Applied through an attribute
    - Often used to register modules
* Demo-PreApplicationStartMe..
  + LogModule hooks into log event of request life cycle(near end of lifecycle)
  + Module needs to be registered with application in Global.asax
  + PreApplicationStart attribute has to be defined at the assembly level
  + Ex
    - using …
    - [assembly: PreApplicationStartMethod(typeof(MvcApplication), “Register”)]
    - namespace HealthAssist
    - {
    - public class MvcApplication : System.Web.HttpApplication
    - {
    - …
    - }
  + Then add Register() method to register log module
  + Ex
    - public static void Register()
    - {
    - HttpApplication.RegisterModule(typeof(LogModule));
    - }
    - protected void Application\_Start()
    - {
    - …
    - }
* Exploring the Request Life Cyc..
  + BeginRequest
  + AuthenticateRequest
  + AuthorizeRequest
  + ResolveRequestCache
    - URL Routing Module responds to PostResolveRequestCache, when it selects http handler for request
    - Choose handler
  + MapRequstHandler
    - Officiates handler is about to be chosen
  + AcquireRequestState
  + RequestHandlerExecute
    - MVC handler executes
    - MVC generates response
  + UpdateRequestCache
  + LogRequest
  + EndRequest
* Demo - The Request Life Cyc…
  + Application\_PostRequestHandlerExecute()
    - Where most MVC will execute(action methods, etc)
* Working with Events Across Fr..
  + Application life cycle is framework agnostic
    - Can access events in an asp.net application, mvc or not
* Demo - Events Across Framew..
* Summary
  + Pipeline beings with Application\_Start
  + PReApplicationStartMethod allows early configurations
  + Events, HttpsHandlers, and HttpModules support and enable execution of the MVC framework
  + Some of these components can be shared across requests and even frameworks
* Introduction
* Understanding HttpHandlers
  + Generate response
  + Classes that implement IHttpHandler and generate a response to HttpRequst
  + MapRequestHandler
    - ASP.NET HttpHandler Selecltion
  + RequestHandlerExecute
    - HttpHandler executes
  + Creating an HttpHandler
    - Create a class that implements the IHttpHandler interface
    - Register the HttpHandler through code or config file
    - IHttpHandler exposes two members
      * IsReusable
      * ProcessRequest() //main execution method, generates a response
* Demo - Building a Custom Http…
  + Common uses
    - Customizing behavior of existing frameworks
    - Building your own framework
  + Request intercepted by custom handler will not be executed by mvc handler
  + Add new class the implements IHttpHandler interface
  + Implement ProcessRequest(HttpContext context)
  + Ex
    - public void ProcessRequest(HttpContext context)
    - {
    - context.Response.Write(“<p>This was generated by the sample handler. </p>”);
    - }
  + HttpContext is a resource provided by the larger asp.net pipeline
  + Then need to register handler with application using code or web config
  + In RouteConfig.cs
    - Add route to SampleHanlder
    - Ex)
    - routes.Add(new Route(“home/about”, new SampleRouteHandler()));
  + each route needs an associated route handler
  + ex
    - public class SampleRouteHandler : IRouteHandler
    - {
    - public IHttpHandler GetHttpHandler(RequestContext requestContext)
    - {
    - return new SampleHandler();
    - }
    - }
  + HttpHandlers are what generate response to browser
  + Only one can run for every request
  + When url is requested, whatever route matches request first will have its associated handler exeutes
* Working with HttpModules
  + Classes that implement IHttpModule
  + Designed to respond to Life Cycle Events
  + Can be use manipulate the request
  + Many modules can act upon a single request
  + Each module can hook into many life cycle events
  + Advantages of HttpModule
    - Event handler code can be abstracted into a reusable container
    - Reusable HttpModule
  + Creating an HttpModule
    - Create a class that implements the IHttpModule Interface
    - Register the HttpModule through code or config file
  + IHttpModule exposes two members
    - Init() //called when an httpmodule is instantiated, used to register methods that will handle the asp.net life cycle events
    - Dispose() //release resources
  + HttpContext
    - Used by HttpModule to provide information and services to executing framework like mvc
  + Modules can populate properties on HttpContext objects
    - Those can be accessed by through handlers or MVC Framework
    - Modules can act on request before framework if you want them too
* Demo - Creating a Custom Htt..
  + Redirect Actions can be expensive
    - both request have to travel through mvc pipeline
  + with and HTTPModule we can skip the mvc framework entirely
  + in web config add <redirects> section
  + ex
    - <redirects>
    - <add Title=”Marketing” Old=”/Home/About” New=”/Home/Contact”/>
    - <add Title=”ContactChange” Old=”/Home/Services” New=”/Home/Contact”/>
    - </redirects>
  + Add a Configuration folder to project
  + then add classes to work with managing web config entries
  + Redirect.cs
    - have c# property for each attribute on redirect add element in webconfig
  + RedirectCollection.cs
    - collection of redirect items
  + RedirectSection.cs
    - represent overall redirect configuration section in webconfig
  + in WebConfig.cs
    - add <configSections> that maps RedirectSection class to <configSection>
  + add HttpModule class
    - implements IHttpModule interface
    - add RedirectUrls() method
  + RedirectUrls()
    - get reference to all redirect entries in webconfig
    - loop through them, compare incoming url to what was entered in webconfig
    - if it matches, call response and pass in new entry
  + then register module in Web.Config
    - <system.WebServer>, <modules>
  + we took full control of request using http module and redirect it to another url
  + module handle event before mvc handler executes, framework never even process the request
* Comparing HttpHandlers and …
  + Modules
    - many HttpModules can service on request
    - modify and support requests through services
    - Implemented through IHttpModule interface
    - Registered through code or the webconfig file
    - Designed to integrate with any of the Life Cycle Events
  + Handlers
    - only one HttpHandler can service a request
    - generate the response that is sent back to the browser
    - Implemented through IHttpHandler interface
    - Registered through code or the webconfig file
    - Generally only concerned with the events related to handler mapping and execution
* Exploring the MVCRouteHandl…
  + UrlRoutingModule
    - match url to Route defined in MVC app
    - get the route’s MvcRouteHandler
    - call GetHttpHandler on MvcRouteHandler
    - call ProcessRequest on returned MvcHandler
* Summary
  + HttpHandlers generate a response for a request
  + HttpModules respond to Life Cycle Events to provide services and manipulate the request
  + MVcHandler is the MVC frameworks implementation of IHttpHandler
* Introduction
  + controllers and request pipeline
  + controller factory
  + dependency resolver
* Controllers and the Request Lif..
  + controllers are responsible for orchestrating relationship between view and model
  + implements IController interface
    - Execute() method
  + controller initialization happens in Mvc Handler ProcessRequest()
  + ProcessRequest()
    - call ProcessRequestInit()
    - which call controller factory, select appropriate controller class using supplied route data
    - factory use controller activator to create instance of that class
    - controller activator uses dependency resolver
    - dependency resolver calls Controller.Execute()
* Introducing the Controller Fact…
  + provide appropriate type of controller to service the request
  + IControllerFactor interface
    - IController CreateController()
      * returns instance of IController
    - SessionStateBehavior GetControllerSessionStateBehavior()
      * determines how sessions is handled
    - Void ReleaseController()
  + There is a DefaultControllerFactory
    - convention based approach to creating controllers
    - maintain awareness of what controller is available
    - access to dependency resolver
* Demo - Building a Custom Con…
  + Add controller that implements IController
  + Implement Execute()
  + Ex)
    - public void Execute(RequestContext requestContext)
    - {
    - HttpContext.Current.Response.Write(“This was generated by our custom IController class”);
    - }
  + Takes a lot of work to make Execute do something, usually just better to implement Controller instead of IController
  + By default MVC 5 framework cannot create constructors that include parameters
    - Most common reason to modify default controller factory’s behavior
  + Add new class to project that implements IControllerFactory
  + Ex)
    - public class ParameterControllerFactory : IControllerFactory
    - {}
  + Implement IController CreateController
  + Ex
    - public IController CreateController(System.Web.Routing.RequestContext requestContext, string controllerName)
    - {
    - if(controllerName == “contact”)
    - {
    - return new ContactController();
    - }
    - }
  + Add class that implements ILoggingService
    - public class Logger : ILoggingService {}
  + implement SessionStateBehavior GetControllerSessionBehavior(Requestcontext context, string controllerName)
    - public SessionStateBehavior GetControllerSessionBehavior(RequestContext requestContext, string controllerName)  
      {
    - return SessionStateBehavior.Default;
    - }
  + Then in Global.asax, register ParameterControllerFactory as factory application should use
    - Protected void Application\_Start()
    - {
    - ControllerBuilder.Current.SetControllerFactory(new ParameterControllerFactory());
    - AreaRegistration.RegisterAllAreas();
    - …
    - }
  + MVC handler ProccesRequest() triggers Controller Factory
    - Called between Pre and Post RequestHandlerExecute()
* Exploring the Dependency Res…
  + Used by MVC internally to resolve types
  + Not limited to providing controller instances
  + IDependencyResolver Interface
    - Has two members
    - Object GetService(Type serviceType)
      * Used by controller factory to obtain a controller instance to handle response
    - IEnumerable<object> GetServies(Type serviceType)
* Demo - A Customer Dependency …
  + Add Ninject nuget package
  + Ninject replaces need to write custom controller factory
    - Uses custom dependency resolver
    - NinjectWebCommon.cs
    - Set custom dependency resolver for our application
  + Can influence behavior of controller factory by changing depending resolver it relies on
* Summary
  + MVCHandler retrieves a controller from the ControllerFactory
  + MVCHandler calls the Execute() method of the acquired Controller
  + DefaultControllerFactory can use a Dependency Resolver to acquire controller instances
  + Dependency Resolver is extensible
* Introduction
  + Action methods and the life cycle
  + Understanding the Action Invoker
  + Making decisions with action selectors
  + Influencing the request with filters
  + The model binding process
* Action Methods and the Life C…
  + Controller Execute()
    - Action invoker selects method
      * Selects and execute best method to handle request
      * Using route data and method names
    - Authentication Filters
      * Failure results in challenge
    - Authorization Filters
      * Failure results in challenge
    - Model Binding
      * Takes data from request and use to create parameters for method
    - Action Filters
      * OnActionExecuting fires before action method
      * OnActionExecuted fires after action method
    - Action Method
      * Returns Action Result which renders response for request
    - Action Filters
* The Action Invoker
  + Used by controller to locate and execute action method
  + IActionInvoker Interface
    - bool InvokeAction(ControllerContext controllerContext, string actionName);
  + MVC framework provides the powerful ControllerActionInvoker
  + Can customize default behavior using extension points
* Action Method Selectors
  + Public, non static, non special are excluded
  + If there are multiple qualified methods
    - Checks if they have action selectors
  + If there is still multiple
    - Chose one with action selector
  + If there is still multiple return error
  + Use Action Selector for get and post of methods with same name
  + Can create custom action selectors
    - Class that inherits ActionMethodSelectorAttribute
    - Override method bool IsValidForRequest
    - If selector for method returns false is no longer a candidate for selection by action invoker
* Demo - Decision Making with A..
  + Can write custom action selectors
  + Add folder to project, called extensions
  + Add class to folder
  + Inherits from ActionMethodSelectorAttribute
  + Override IsValidForRequest
  + Ex)
    - public class IsMobile : ActionMethodSelectorAttribute
    - {
    - public override bool IsValidForRequest(ControllerContext controllerContext, methodInfo)
    - {
    - return controllerContext.HttpContext.Request.Browser.IsMobileDevice;
    - }
    - }
  + The in a controller
  + Ex)
    - [IsMobile]
    - public ActionResult Register() {}
* The Model Binding Process
  + Process that maps data from request to parameters on action method
  + Retrieves data from value providers
  + Value providers are classes that provide information from various sources related to current request
    - Provide 4 default providers that collect data from common places
    - From Data
    - Route Data
    - Query String
    - Files
    - Custom sources
  + IModelBinder Interface
    - Exposes one method
    - public object BindModel(ControllerContext controllerContext, ModelBindingContext bindingContext) { }
  + can create custom model binder
* Working with MVC Filters
  + Runs at multiple points in life cycle
  + Filter types
    - Authentication Filters -> IAuthenticationFilter
    - Authorization Filters -> IAuthorizationFilter
    - Action Filters -> IActionFilter
      * Run before and after action methods
    - Result Filters -> IResultFilter
      * Run before and after action results
    - Exception Filters -> IExceptionFilter
      * Run if there is an error in action execution pipeline
  + Filter Scope
    - Can be applied to individual action methods
    - Can be applied at controller level, applies to all action in that class
    - Can be applied globally across entire application for all controllers
  + Filter Execution Process
    - Can execute in any given order unless told otherwise
    - Order only applies to OnActionExecuting method in reverse
* Demo - MVC Filter Execution O…
  + Create class
  + Ex
    - public class ActionFilter1: FilterAttribute, IActionFilter
    - {
    - public void OnActionExecuted(ActionExecutedContext filterContext)
    - {
    - }
    - public void OnActionExecuting(ActionExecutingContext filterContext)
    - {
    - }
    - }
    - public class AuthorizationFilter : AuthorizeAttribute
    - {
    - public override void OnAuthorization(AuthorizationContext filterContext)
    - { }
    - }
  + FilterAttribute allows them to be applied as attribute inside of controllers
  + Can addition order parameter to filter to order their execution
  + Ex)
    - [ActionFilter1(Order=1)]
    - [ActionFilter2(Order=2)]
    - public ActionResult test () {}
  + Authorization will run first regardless of order
  + Add filter on controller to apply to all methods in controller
  + Ex
    - [ActionFilter2]
    - public class HomeController : Controller {}
  + to apply filter globally, open FilterConfig.cs in the App\_Start folder
  + ex)
    - public class FilterConfig
    - {
    - public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    - {
    - filters.Add(new HandleErrorAttribute());
    - filters.Add(new ActionFilter3());
    - }
    - }
  + Filters fire in order from the most global level to the specific
* Summary
  + Controller uses the Action Invoker to find and execute Action Methods
  + Invoker selects a method based on name and Action Selectors
  + Filters can inject security logic and other custom code
  + Model Binder populates Action Methods parameters for the Action Invoker
* Introduction
  + View Results and everything else
  + Actions results in the life cycle
  + Building a custom action result
  + View results and the view engine
  + Extending view engine functionality
* Action Results and the Life Cyc…
  + Before action result executes, result filters are run first
    - OnResultExecuting
    - Action Result - ExecuteResult()
  + If ActionResult is a View or PartialView
    - View Engine and render view
  + Other ActionResults, handles response itself
  + Action Result Types
    - Several types: ContentResult, ViewResult, RedirectAction, etc
    - All inherit from ActionResult and implement ExecuteResult()
* Demo - Working with Action R…
  + Json.NET provides more features to default built Javascript serializer
  + add class to extensions folder
    - inherit from ActionResult
    - public property for object that will be passed in for serialization
    - override ExecuteResult()
    - ex)
    - public class JsonNETResult : ActionResult
    - {
    - public object Data { get; set; }
    - public override void ExecuteResult(ControllerContext context)
    - {
    - var response = context.HttpContext.Response;
    - response.ContentType = “application/json”;
    - response.Write(JsonConvert.SerializeObject(Data));
    - }
    - }
  + JsonConvert is a useful class provided by Json.NET that helps with serialization
* View Results and the View Eng…
  + View Results uses View Engine
  + MVC provides two engines by default
    - Razor View Engine
    - Legacy View Engine
  + Action Invoker
    - ViewResultBase - ExecuteResult()
    - ViewResult - FindView()
    - ViewEngine - FIndView()
    - ViewEngineResult
  + IViewEngine Interface
    - ViewEngineResult FindView(ControllerContext controllerContext, string viewName, string masterName, boolUseCache);
    - ViewEngineResult FindPartialView(ControllerContext controllerContext, string partialViewName, bool useCache);
    - Void ReleaseView(ControllerContext controllerContext, IView view);
  + Default Razor Search Locations
    - ~/Views/{1}/{0}.cshtml
    - ~/Views/Shared/{0}.cshtml
    - ~/Areas/{2}/Views/{1}/{0}.cshtml
    - ~/Areas/{2}/Views/Shared/{0}.cshtml
    - {0} is Action name
    - {1} is Controller name
    - {2} is Area name
  + Can change location where engine searches
    - Extend existing one
    - Build custom one
  + View Engine Search Order
    - Multiple implementations can be registered at the same time
    - Utilized in order they were added to view engine collection
    - As soon as one finds a view the search if finished,
* Demo - Customizing View Engi..
  + Add folder
  + Add new class
  + Add new locations to look for views in constructor
  + Exposes two properties
    - View and Partial view formats
    - Array of strings that define search location for search engine
  + Ex)
    - public class ThemeViewEngine : RazorViewEngine
    - {
    - ViewLocationFormats = new string [] { “~/Theme/{1}/{0}.cshtml” }’
    - PartialViewLocationFormats = new string [] { “~/Theme/{1}/{0}.cshtml” };
    - }
  + Copy \_ViewStart.cshtml and Web.config in Views to folder Theme
    - Web.config makes sure view is served up properly and cannot be access as files directly
    - \_ViewStart.cshtml will bring in the \_Layout.cshtml
    - In production you will want to override \_Layout.cshtml
  + Then register view engine with application, in Global.asax
    - protected void Application\_Start()
    - {
    - ….
    - ViewEngines.Engines.Add(new ThemeViewEngine());
    - }
  + Default search location is used first since ThemeViewEngine was added after
  + ViewEngines.Engines.Add() adds the view engine at the end of the collection
  + ViewEngines.Engines.Insert(index, view engine) allows you to specify a index
* Summary
  + Action Results generate a response for the request
  + Different action results types have different execution implementations
    - Through ExecuteResult()
  + Custom action results can be built for specific scenarios
  + View results rely on the View engine to find and parse views