* Introduction
* What is OWIN
  + Open Web Interface for .NET(OWIN)
  + Standard interface between .NET web servers and web applications
  + Specification that defines how you can abstract away web server in a efficient way
  + Build web applications without caring about how it will be hosted
  + .NET + Web = IIS
  + Abstract away web server using a single delegate and dictionary
  + Ex)
    - using AppFunc = Func<IDictionary<string, object>, Task>;
    - var f = new AppFunc(environment = > { return Task.FromResult(null); });
  + IDictioanry<string,object> contains all information passed to server in http request
    - Also any functionality that web server offers to web application
* The Parts of OWIN
  + Owin defines parts/actors that are involved in handling the incoming request and return response
  + Host
    - Some process that host all other parts
    - Starting everything up
  + Server
    - Responsible for accepting incoming request and sending back responses
  + Middleware pipeline
    - Piece of code that request passes on the way to and from the application
    - Can be used to inspect and modify incoming request and outgoing response
  + Application
    - Responsible for generating response
  + Use web framework to abstract away middleware and application
* The Flow
* Basic Keys in the Environment …
  + A lot of keys, prefix with owin.
  + Ex) owin.RequestPath
* Project Katana
  + Microsoft implementation of OWIN specification
  + Start new ASP.NET web application
  + Choose MVC template
  + Change authentication(individual user accounts)
    - Will include owin packages
* Summary
  + OWIN is specification for webserver abstraction
  + Open Web Interface for .NET(OWIN)
  + Katana is Microsoft implementation of OWIN
* Introduction
  + Building a simple OWIN Pipeline
* Creating an OWIN Based Appli…
  + Create an empty project
  + Install package ‘Microsoft.Owin.Host.SystemWeb’
    - Microsoft.Owin and Owin will also get installed
    - Microsoft.Owin contains katana
  + Add class Startup
    - Create entry point
    - Ex)
    - public class Startup
    - {
    - public static void Configuration(IAppBuilder app)
    - {
    - app.Use(async (ctx, next) => { await ctx.Response.WriteAsync(“Hello World”});
    - }
    - }
  + IAppBuilder is used to add middleware to OWIN pipeline
  + app.Use((IOwinContext, Func<Task>)
* Adding Multiple Middlewares t…
  + Need to make sure rest of pipeline executes
  + Ex)
    - app.Use(async (ctx, next) => {
    - Debug.WriteLine(ctx.Request.Path);
    - await next();
    - });
  + Next references to the next middle ware in the pipeline
* Summary
  + NuGet the required host
  + Add public class Startup
  + Add Configuration(IAppBuilder app)
  + Add middlewares
* Introduction
* Creating the Middleware Class
  + Add folder ‘Middleware’
  + Add class
    - If you inherit from OwinMiddleware you will only be able to be used for owin
    - Declare AppFunc
    - Add constructor
    - Ex
    - using AppFunc = System.Func<System.Collections.Generic.IDictionary<string, object>, System.Threading.Tasks.Task
    - …
    - public class DebugMiddleware
    - {
    - AppFunc \_next;
    - public DebugMiddleware(AppFunc next)
    - {
    - \_next = next;
    - }
    - }
  + Then add public method to invoke middleware in middle class
  + Ex)
    - public async Task Invoke(IDictionary<string, object> environment)
    - {
    - var ctx = new OwinContext(environment);
    - Debug.WriteLine(“Incoming request: “ + ctx.Request.Path);
    - await \_next(environment);
    - Debug.WriteLine(“Outgoing request:” + ctx.Request.Path);
    - }
  + Instead of using new OwinContext(environment)
    - You can use var path = (string)environment[“own.RequestPath”];
  + Then in Startup
    - Use the DebugMiddleware
    - Ex) app.Use<DebugMiddleware>();
* Adding Middleware Configurati…
  + To be able to configure middleware to do different things in different situations, use Options class
  + Name of options of class is [name of middleware] + Options
    - Properties corresponding to things we want to configure
  + Ex
    - public class DebugMiddlewareoptions
    - {
    - public Action<IOwinContext> OnIncomingRequest { get; set; }
    - public Action<IOwinContext> OnOutgoingRequest { get; set; }
    - }
  + Then in middleware class add options to constructor
  + Ex
    - public DebugMiddleware(AppFunc next, DebugMiddlewareOptions options)
    - {
    - \_next = next;
    - \_options = options;
    - If(\_options.OnIncomingRequest == null)
    - \_options.OnIncomingRequest = (ctx) = > { Debug.WriteLine(….)};
    - If(\_options.OnOutgoingRequest == null)
    - \_options.OnOutgoingRequest = (ctx) => { Debug.WriteLine(…)};
    - }
  + Then in invoke method call methods on options
  + Ex
    - public async Task Invoke(IDictionary<string, object> environment)
    - {
    - var ctx = new OwinContext(environment);
    - \_options.OnIncomingRequest(ctx);
    - await \_next(environment);
    - \_options.OnOutgoingRequest(ctx);
    - }
  + Then in Startup
    - Add DebugMiddlewareOptions instance as a parameter to where we ad DebugMiddleware to owin pipeline
  + Ex)
    - app.Use<DebugMiddleware>(new DebugMiddlewareOptions());
* Create Add Method for the Mid..
  + Convention when adding middle, use custom method “Use[Middleware]”
  + Ex)
    - app.UseDebugMiddleware();
  + create a static class,
  + changes namespace to just Owin
  + ex)
    - public static class DebugMiddlewareExtensions
    - {
    - public static void UseDebugMiddleware(this IAppBuilder app, DebugMiddlewareOptions options = null)
    - {
    - If(options == null)
    - options = new DebugMiddlewareOptions();
    - app.Use<DebugMiddleware>(options);
    - }
    - }
* Summary
  + Create a middleware class in which we put all the functionality we wanted
  + Create middleware options class used to configure middleware
  + Create a middleware extension class in which we create extension method to IApp builder, making it easy to register middleware in OWIN pipeline
* Introduction
  + Integrate third party framework instead of building everything on our own
* Third Party Framework
  + Make sure framework can exist as middleware in OWIN pipeline
* Adding NancyFx
  + Install nuget package Nancy.owin
  + Create Nancy modules and inside of modules define what paths they should respond to
  + Create folder
  + Create class
    - Add using Nancy, Nancy.Owin
    - Inherit from NancyModule
  + Ex
    - public class NancyDemoModule : NancyModule
    - {
    - public NancyDemoModule()
    - {
    - Get(“/nancy”, x => {
    - var env = Context.GetOwinEnvironment();
    - return “Hello from Nancy! You requested: “ + env[“owin.RequestPath”];
    - });
    - }
    - }
  + Inside NancyModule there is dictionary for each Http Verbs
  + Add path as key and pass delegate as a value
    - Delegate will be called every time there is the Http verb request with path that correspond to key
  + The in start up add Nancy to pipeline
  + Ex
    - ..
    - app.UseDebugMiddleware(…)
    - app.UseNancy();
    - app.Use(…)
* Configuring NancyFx for Passt..
  + Nancyfx will hog pipeline
  + Several way to fix this
  + app.Map() define specific path that will have its own configuration
  + ex)
    - app.Map(“/nancy", mappedApp = > { mappedApp.UseNancy(); });
  + problem is nancy does routing on request path but ignores request path base
  + better to use Nancy.Owin setting
    - tell it pass through nancy if response code matches predefined one
  + ex)
    - using Nancy.Owin
    - ….
    - app.UseNancy(config => {
    - config.PassThroughWhenStatusCodeAre(HttpStatusCode.NotFound);
    - });
  + When status code is not found it should pass through to next middle ware
* Adding ASP.NET Web API
  + Install nuget package Microsoft.AspNet.WebApi.Owin
  + Add controller folder
  + Add class
  + Inherit from ApiController
  + Add attributes
  + Create an action that returns IHttpActionResult
  + Ex)
    - [RoutePrefix(“api”)]
    - public class HelloWorldApiController : ApiController
    - {
    - [Route(“hello”)]
    - [HttpGet]
    - public IHttpActionResult HelloWorld()
    - {
    - return Content(System.Net.HttpStatusCode.OK, “Hello from Web Api”)
    - }
    - }
  + To call action …/api/hello
  + Then in Startup register WebApi in OWIN pipeline
  + Ex)
    - …
    - var config = new HttpConfiguration();
    - config.MapHttpAttributeRoutes(); //go through app and find api controller attributes
    - app.UseWebApi(config);
  + web api will default to return json to browser
* Adding ASP.NET MVC