* 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
  + Install nuget package Microsoft.aspnet.mvc
  + Add class to controllers folder
    - Use template MVC 5 controller-empty
  + Create view in for action method
    - In View folder -> [controller] folder
    - Name view [action method]
    - Have @inherits System.Web.Mvc.WebViewPage
  + When you manually added aspnet mvc nuget package there is missing a few web config settings you normally get in project from template
    - To fix this add @inherits System.Web.Mvc.WebViewPage
  + Create a global application class
    - Add route for controller and action
  + In Application\_Start()
    - Ex)
    - protected void Application\_Start(object sender, EventArgs e)
    - {
    - RouteTable.Routes.MapRoute(name: “Default”,
    - url: “{controller}/{action}”,
    - defaults: new { controller = “Home”, action = “Index” });
    - }
  + If there is no middleware in pipeline that send back response, request is passed on to ASP.NET MVC to handle
* Summary
* Introduction
  + Use katana to add security/authentication into web app
* Adding Cookie Authentication
  + Add [Authorize] attribute to controller
  + Ex)
    - [Authorize]
    - public class SecretController : Controller
    - {
    - public ActionResult Index(0
    - {
    - return View();
    - }
    - }
  + [Authorize] tells mvc that only authenticate user can use controller
  + Use cookies to authenticate
  + Install nuget package Microsoft.owin.security.cookies
  + In Startup.cs add cookie authentication middleware
  + Ex
    - app.UseCookieAuthentication(new Microsoft.Owin.Security.Cookies.CookieAuthenticationOptions {
    - AuthenticationType = “ApplicationCookie”,
    - LoginPath = new Microsoft.Owin.PathString(“/Auth/Login”)
    - });
* Logging in Users
  + Add Folder Models
  + Add class for login
    - Add username property
    - Add password property
  + Add controller
    - Add action called Login(from the LoginPath you set in startup.cs)
  + Add view for action
    - Include @inherits System.Web.Mvc.WebViewPage<Owin.Demo.Models.LoginModel>
    - Include @using System.Web.Mvc.Html
    - Then add form for input
  + Ex
    - @using (var form = Html.BeginForm())
    - {
    - <div>
    - @Html.LabelFor(x => x.UserName)
    - @Html.TextBoxFor(x => x.UserName)
    - </div>
    - <div>
    - @Html.LabelFor(x => x.Password)
    - @Html.PasswordFor(x => x.Password)
    - </div>
    - <div>
    - <input type=”submit” value=”Log in” />
    - </div>
    - }
  + Then in controller add action to handle post
  + Ex)
    - [HttPost]
    - public ActionResult Login(LoginModel model)
    - {
    - if(model.UserName.Equals(“chris”, StringComparison.OrdinalIgnoreCase) && model.Password == “password”)
    - {
    - var identity = new ClaimsIdentity(“ApplicationCookie”);
    - identity.AddClaims(new List<Claim>
    - {
    - new Claim(ClaimTypes.NameIdentifier, model.UserName),
    - new Claim(ClaimTypes.Name, model.UserName)
    - });
    - HttpContext.GetOwinContext().Authentication.SiginIn(identity);
    - }
    - return View(model);
    - }
  + Then remap nancy
    - app.Map(“/nancy”, mappedApp => { mappedApp.UseNancy(); });
* Interacting with the User and L…
  + Create action method to log out
  + Ex
    - public ActionResult Logout()
    - {
    - HttpContext.GetOwinContext().Authentication.SignOut();
    - return Redirect(“/”);
    - }
  + To add authentication with nancyfx install nuget package nancy.msowinsecurity
  + In NancyDemoModule
    - Add this.RequiresMSOwinAuthentication();
  + Ex
    - public NancyDemoModule()
    - {
    - this.RequiresMSOwinAuthentication();
    - Get(“/nancy”, x => …);
    - }
  + Integrate authentication in middle ware
  + Ex in Startup.cs
    - app.Use(async (ctx, next) => {
    - if(ctx.Authentication.User.Identity.IsAuthenticated)
    - Debug.WriteLine(“User: “ + ctx.Authentication.User.Identity.Name);
    - else
    - Debug.WriteLine(“User Not Authenticated”);
    - await next();
    - }
* Summary
  + Add CookieAuthenticationMiddleware
  + Authenticate User
  + Use the user and the user’s claim
  + Log out user
* Introduction
* Setting up the Development En…
  + Twitter for example doesn’t not accept application to be hosted on local host
  + Fake a url by changing host file on machine
  + Open c:\windows\system32\drivers\etc
    - Change file type to All Files(\*.\*)
    - Add 127.0.0.1 [www.owin.demo](http://www.owin.demo)
  + Open IIS Manager
    - Add new website
    - Set physical path to path of project
    - Set host name to [www.owin.demo](http://www.owin.demo)
  + Click on properties of project
    - Click on web
    - Switch to Local IIS
    - Set project url to <http://www.owin.demo>
* Adding Facebook Authentication
  + Add app on facebook
  + Click on app
  + Set up redirect uri for oauth flow
  + Click on settings
    - Click on advance
  + In “Valid OAuth redirect URIs”
    - Add <http://www.owin.demo/signin-facebook>
  + Save changes
  + In visual studio
  + Install nuget package Microsoft.owin.security.facebook
  + In startup
    - app.UseFacebookAuthentication(new Microsoft.Owin.Security.Facebook.FacebookAuthenticationOptions {
    - AppId = “1498507743777935”,
    - AppSecret = “ba49d8db081c6c6e1c1decc6519c94f”,
    - SiginInAsAuthenticationType = “ApplicationCookie”
    - });
  + Get AppId from facebook app settings -> basic
  + Get App secret from facebook as well
  + In production, you will want to have user sign in with separate cookie
    - Then transfer to them being proper web application user
  + Add action
  + Ex
    - public ActionResult LoginFacebook()
    - {
    - HttpContext.GetOwinContext().Authentication.Challenge(new Microsoft.Owin.Security.AuthenticationProperties {
    - RedirectUri = “/secret”
    - }, “Facebook”);
    - return new HttpUnauthorizedResult();
    - }
* Adding Twitter Authentication
  + Create new twitter app
  + Add website url <http://www.owin.demo>
  + Callback url <http://www.owin.demo/signin-twitter>
  + Install nuget packet Microsoft.owin.security.twitter
  + In start up
    - app.UseTwitterAuthentication(new Microsoft.Owin.Security.Twitter.TwitterOptions
    - {
    - ConsumerKey = “”,
    - ConsumerSecret = “”,
    - SignInAsAuthenticationType = “ApplicationCookie”
    - BackchannelCertificateValidator = “”
    - });
  + Use LoginModel to pass list of authentication providers to view
  + Have view render link to each one dynamically
  + Ex)
    - var providers = HttpContext.GetOwinContext().Authentication.GetAuthenticationTypes(x => !string.IsNullOrEmpty(x.Caption)).ToList();
    - model.AuthProviders = providers;
  + then in view
  + ex)
    - @foreach(var provider in Model.AuthProviders)
    - {
    - <div>@Html.ActionLink(“Log in with “ + provider.Caption, “SocialLogin”, new { id = provider.AuthenticationType});
  + Then create action method
    - public ActionResult SocialLogin(string id)
    - {
    - HttpContext.GetOwinContext().Authentication.Challenge(new Microsoft.Owin.Security.AuthenticationProperties {
    - RedirectUri = “/secret”
    - }, id);
    - }
* Summary
  + Integrate social media authentication using katana and middlewares
  + Configure authentication provider
    - Set up app with them
  + Get nuget package for selected provider
  + Add middlware to pipeline
  + Configure middleware with id and secret
  + Create login link
  + Create challenge
  + Return HTTP 401 Unauthorized after creating challenge otherwise cookie authentication middle wont handle challenge and redirect the user
* Introduction
  + Hosting an OWIN pipeline outside of IIS
* Self-Hosting OWIN
  + asp net mvc cannot be hosted in owin pipeline
  + create a new project, Console application
  + install nuget package Microsoft.owin.selfhost
  + create a start up class
    - make it public
    - add method Configuration(IAppBuilder app)
  + ex)
    - public static void Configuration(IAppBuilder app)
    - {
    - app.Use(async (ctx, next) => {
    - await ctx.Response.WriteAsync(“Hello World”);
    - });
    - }
  + in program.cs add in Main
  + ex)
    - static void Main(string [] argos)
    - {
    - using (WebApp.Start<Startup>(“<http://localhost:12345>”))
    - {
    - Console.ReadLine();
    - }
    - }
  + serving up static content with IIS will handle that for you
  + when you self host you will have to server static file on your own
  + install nuget package microsoft.owin.staticfiles
  + then in startup class add it to owin pipeline
    - ex) app.UseStaticFiles();
  + add static file to project
    - to make sure it ends up in same folder as console application
    - click on properties of file
    - change ‘Copy to Output Directory’ to Copy if newer
* Testing the OWIN Pipeline
  + add new project to solution
  + choose test -> unit test project
  + right click on test project references
    - add reference to the hosting project
  + in Test class
    - [TestClass]
    - public class OwinTests
    - {
    - [TestMethod]
    - public async Task Owin\_returns\_200\_on\_request\_to\_root()
    - {
    - }
    - }
  + install nuget package microsoft.owin.testing