1. Use VS.NET to create a Web App, show what the wizard did in the project and in AAD
2. Use VS.NET to create a WebAPI, show as above
3. Ask what still needs to happen to have the WebAPP talk to WebAPI and start implementing these, if they miss an item, don’t do it – it will show up later as an error and I think it’s good for the class to experience debugging.
4. Use VS.NET Project->Connected Services to add ADAL to the Web App (check the ‘read AAD data’ button) and explain why we use this trick (easy way to get ADAL into the project even though we don’t really want to use AAD Graph but we need a token for the Web API)
5. Change the Startup.auth.cs code to ask for a token to the Web API instead of Graph
6. Run and show we are getting it (if we missed changing Web App’s permissions to call Web Api we will see an error – fix it).
7. Take the token we get in Startup and use Fiddler Composer to call the Web Api – show it works (or not 😊 – fix it).
8. Make a change to one of the methods in the Home controller (About) to call the Web Api. The easiest way is to make all the static properties in Startup.auth.cs public so we can use them in HomeController.cs, copy the ADAL code from Startup.auth.cs OnAutheorizationCodeReceived method to the About method and modify it to call AcquireTokenAsync(Startup.resourceId, clientCredentials, UserIdentity.AnyUser).
9. Do some more slides till you get to claims augmentation
10. Do the claims augmentation in the Web App. The code is in the slides.
11. I normally modify the Home.Contacts method to return current claims:

Response.Write(“<ul>”);

Foreach(var c in ClaimsPrincipal.Current.Claims)

   Response.Write($”<li>{c.Type}: {c:Value}</li>”);

Response.Write(“</ul>”);