asp.net mvc4 fundamental The ASP.NET WebAPI

Introduction

Hi. This is Scott Allen, and this module is a look at the web API features in ASP.NET MVC 4. The web API features allow you to rate web services that work very close to the medal of HTTP. I'll show you many familiar concepts in this module like routing and model binding and we'll see how these concepts are similar but also slightly different with the web API. We'll look at API controllers and build out a small web API to call from JAVA script and from C sharp code. ( Pause )

Web Services

Web services allow two computers to communicate and exchange data and I used the term computer loosely. You can have smartphones communicate with web servers or have web servers communicate with the mainframe, but for many years when someone talked about web services you generally assume they were talking about SOAP-- SOAP-based web services. If you've never heard of SOAP, that's okay, it's not important to know what it is for this module but generally speaking .NET developers have been using Windows Communication Foundation or WCF to implement SOAP-based web services over the last few years. And one of the goals of WCF was to abstract away the underlying transport technology used to exchange messages in a web service transaction. WCF and the SOAP specification didn't want you to know if you were using HTTP or MSMQ or sending binary data over TCP sockets. But over the last few years, some web service implementations have been moving away from SOAP and that transport agnosticism and instead building web services that embrace HTTP because the term HTTP is a flexible protocol that works very well on the internet where most of us communicate. HTTP messages can be cached. They can travel through firewalls. They're lightweight. They can be processed by processors on mobile devices. They can be encrypted and best of all nearly every programming environment in the world offers some capability to send and receive HTTP messages. SOAP-based web services generally required a tool kit and more processing power. HTTP is everywhere, it's lightweight and you can use it to exchange information with big servers as well as small devices like mobile phones. So faced with the shift in web services, the Windows Communication Foundation team at Microsoft began working on a new framework that is now part of ASP.NET and is called web API. The ideas that WCF is still a great technology when you're writing enterprise services and it can use toolkits but the web API is going to be more focused on being lightweight, highly interoperable, and using HTTP all the way.

Getting started

Now, we know that the web API is a framework for building web services on top of HTTP and web API is going to allow us to easily consume XML or JSON. We're also going to see it's easy to produce XML or JSON. The one thing you're going to notice about it is that it has very similar features and concepts to ASP.NET MVC. There's controllers, there's models, there's action filters but the MVC you are used to is really optimized for producing HTML by rendering a view. The web API is optimized for working with every other content type, JSON, XML, anything custom that you want to plug in. It's also very extensible like MVC. You also get started a little bit differently with web API because where as MVC is optimized for building web apps that run on a web server like IIS or IIS Express, web API is designed to run anywhere until you can start a new MVC project and make it a web API project that's one of the templates that you can select when you do found a new project or you can just start an MVC 4 web application, a regular one and web API can be a part of it or another option is to start and have any kind of project even a console mode program and host a web API inside. The easiest way to do this is to use NuGet to install Microsoft.aspnet.webapi.selfhost package that will give you all the assemblies and framework support that you need. We're going to start off by building an ASP.NET MVC 4 web applications, a regular web application like we have and working with and see what we can do with the web API inside of a web app. Here we are inside of visual studio and I'm going to start a new ASP.NET MVC 4 web application. The name of this project is going to be videos and let me go ahead and click OK and what you'll see is that web API is one of the project templates available in visual studio. If I click that I get an ASP.NET MVC project with a web API controller already installed. Well, let's go ahead and build just a regular ASP.NET MVC internet application and add an API controller to these project once it gets spun out because you can add web API into anything. Now, the new project is finally put together and let me just here control F5 to make sure everything is running properly before we start making changes and it looks like the home page appears properly. So now I'm ready to work with the web API and I can do that already because when you create a new MVC 4 project and internet application, it will provide you everything that you need to work with a web API and most notably it won't include the system.web.http assembly and that is actually done by installing the web API as a NuGet package. In fact, if I looked at the list of installed packages in this project, it includes everything from JQUERY to the Entity Framework to the web API which is over here on page two and this is actually a nice feature because currently I'm working with the web API, the release candidate version. At some point, there will be an update to this package and when there is, all they need to do is come into my NuGet packages, look at the updates that are available, click update and I'll be working with the latest version of the web API. That's great. But I have everything in this project that I need to work with the web API already and what I'd like to do is build a little web service using the web API that will allow someone to make calls to manage a list of videos and that means responding to HTTP request. And just like everything else in ASP.NET MVC if you want to respond to an HTTP request, you need to write a controller but in this case my controller is going to be slightly different. I'll right click on the controller's folder, say App controller. I'm going to call it a video controller and the template I'm going to select is API controller. So not a regular MVC controller that can render views but an API controller with empty read/write actions and even though I'm adding this to the controller's folder, there's no reason that I have to do that, I could put it anywhere inside of this project. A lot of people like adding API controllers to an API folder. They can also have them in the same folder as the other controllers or on a different project that's referenced by this web project. They can live anywhere. So with the controller being specified, the template specified, let me click add. And one of the reasons this is called an API controller is because its part of the web API and it derives from a base class known as API controller. So this is the base class for all of the web API controllers that need to respond to HTP messages. API controller is going to be somewhat familiar to you if you work with MVC controllers in the past because it still has a controller context property, it still has a model state property. You can get to the request, you can get to the current user property. And you might think that if I did a build at this point that I'd be able to reach this controller with a browser just by going to local host slash video slash get because that's the way the rest of MVC works. So if I have a home controller with an index action, then I go to slash home slash index and it appears. But API controllers, the routing rules are slightly different and that's what we're going to look at next even though I went to slash video slash get, I don't appear to be getting into this method, instead I'm getting a 4/4 error. So let's talk about routing with web API.

Routing

One of the significant differences in the web API compared to traditional ASP.NET MVC controllers, one of the differences is how the web API will route request into the action methods. With the web API, the HTTP method being used plays a role. The HTTP method is the verb used in the HTTP message and the common verbs are get, post, put, and delete. Those are the verbs that the web API will route by default. You an also handle additional verbs if you need to do something like webDAV. You can do that and handle additional verbs by using an except verbs attribute. What the web API will do is if there is a request for slash videos, the web API will look for a video controller and then look for a method on that controller starting with the word "get." So I could have an action called get videos and because this is an HTTP get message, the framework will invoke get videos. But I could also call it get whatever is just because it starts with the letters G-E-T that's why that particular action method will receive the request. The web API also registers these routes slightly differently since there is no action that's going to be in the URL for the routing engine to pick apart, it's using the verb instead. If you look at the default route configuration in a new MVC application and I'll show you where to find this in a minute, you'll see the default routing rule for a web API. It's done with an extension method map HTTP route. Notice there is no action in the URL template and it's also important to note that the path of this needs to start with API. After looking at this, I know the proper way to reach my video controller is to use slash API slash video as a URL on the browser. That will invoke the get method since we have a get request. Let's try it out. So, before we try to view the controller output and the browser again, let me go and then make one change to this controller. I'm going to do a re-factor rename and change the name to videos controller plural because it's really going to represent a collection of videos, not a single video, so it makes more sense for it to be plural. And now as far as the default routing rules, routings always configure during application start-up so of course that means you have to look in global.asax.cs inside of there is the application start event and here is the line of code that will call route config.register route. Route config is a class that is added to a project by default when you start a new MVC for project that goes into the app start directory. There's a number of things in here that configure the runtime during application start-up and if we look in route config, indeed there's the map HTTP route extension method being called to set-up the default routing for our web API controllers. And now I know, if I want to reach the videos controller, then I should do a build first and come into the browser and instead of doing slash video slash get, I should be doing slash API slash videos. And now, I do receive a response from a server. Internet Explorer doesn't want to show me that response, it wants me to save it or open it and that's because it realizes that the server return JSON. So I could open or save data or I could use a different browser that would actually show me the result in the browser window. I'm going to try a slightly different approach which is to go to a command line utility called curl or C-U-R-L. This is a utility that allows you to send off web request from the command line. So, for instance, I could request local host colon 1341, that's where the application lives and it shows me all the HTML that would be rendered by the home page. And it also means I should be able to get to slash API slash videos and by default, it sends a get request that invokes the get action and I can see value 1 and value 2. They're formatted into a JSON array and that's exactly what I expected to receive back based on the code that's inside of the get method. Let's actually run this with the debugger ones, so let me put an F9 here on these two different get actions and while I'm at it, let me rename those two get all videos just to give it a different name. And also I want to put a breakpoint on the delete and see how this works. And now, I'll do a build since I changed the action name, do an F5 to run it with the debugger, that will launch it in the browser but what I really want to do is come back to curl and say let's issue that same request again, slash API slash videos. So it's just trying to map the HTTP verb to the method name but it doesn't have to be an exact match, it just has to start with a get. And now, what about this other get that includes an ID? And actually, let me stop debugging once and actually when we send a respond back from here, I want to include the ID that we see in the response until I do a build and hit a file and let's come back to curl. And by the way with curl, you can use that capital X parameter to specify the method that you want to use. So I'll just make it explicit that this is a get, I will try that out one more time that arrives at the-- get action, hit F5 to continue and let's try with an ID. So I want to get the video with an ID of 5 and notice that arrives at the get method that take some N-parameter. So this is also something special when it comes to web API because the traditional MVC routing wouldn't distinguish between get-- two methods that have essentially the same name that just differ by the method signature, it would give up. But in the web API case, you can have a get that doesn't take any parameters and a get that does take a parameter. It doesn't really matter that I rename this to all videos, it's really just looking for a method that starts with get. But it's able to distinguish between those two because it does see an ID that arrived in the URL that got put into route data and got passed into us so we can see the 5 come in. That should be very a familiar if you've done MVC programming in the past. And if I hit F5 to continue, then we get value 5. This is working well. What happens if I try to delete the video with an ID F5 because I have a delete action in the background. And now I get back some sort of weird error message about the file or directory does not exist on the server. And that seems peculiar and this is something that you might run into occasionally with the web API and the problem is that not all web servers allow all the HTTP methods to come through. So pretty much anywhere observers going to allow a get or post because we all use them in day to day web programming with HTML but not every web servers configured to allow a put or delete, those aren't a common. And that's what happening in this case. This is running in IAS express which is a lightweight version of IAS that now shifts with Visual Studio in 2012 and it's the default host for any web applications that you build. What I mean to do is go into the configuration file for IAS express. That configuration file lives in my documents directory under a folder called IAS express slash config and it's inside of there where I can find application host. And it just so happens that I know I'm looking for an entry in here called extension list URL integrated 40 and right here we can see that the only verbs is gong to map to that particular handler or get ad post and debug, I also want to add put and delete so we handle those verbs as well and allow them to pass through our logic. I should just be able to save this and then we can come back up to curl and try to delete the video with an idea 5. And now that request arrived at our server. So, inside of these action methods, what we would essentially be doing is working with a database or some sort of persistent data store or the file system to look at videos, create videos, in this case, delete a video. But before we get to that, I want to talk about another important feature of the web API which is content negotiation.

Content Negotiation

One of the core concepts in the HTTP specification is the concept of content negotiation. This is where a client asks for a resource like a list of videos or stock quote and when it request the resource, it also includes an HTTP header, the accept header to tell the server the representation that we prefer to see. There is also an accept the language header for the client to tell the server the language they will prefer like French or Swedish or English, and there are headers to specify the desired encoding and character set. But we're going to focus on the accept header for just now. The accept header that you see on the screen and the arrows and accept header is saying, I would like a PNG image. And the reason this is important is because the server might be holding a particular resource like the stock price for Microsoft, but that single resource can have multiple different representations. We could take that stock price resource and render it in an HTML view or we could render the stock price in XML or in Java Script object notation or maybe as an image. The idea of having a single resource with multiple different representations, that's a central theme in the web and it's one of the reasons the web is so interoperable because we can if we try hard enough, always give the client the representation they want to see, the representation that they know how to work with. The web API will take care of content negotiation for us. Essentially, there are different formatters that are plugged in to the web API runtime and each formatter knows how to handle a specific media type. Media type is the fancy term we use for house of represented resource, do you want a media type of JSON or XML or HTML or JPEG image. If you want more details on content negotiation and HTTP, see my HTTP course here on Pluralsight.com. For now, let's see how content negotiation works in our project. With the application running in debug mode and curl opened up, let's go back to where we were just requesting the get method for all videos. And with the breakpoint set there, I'm going to go into debug windows and open up the immediate window which you've never used this before to wonderful little interactive environment where we can play around with things. And I want to look at the global configuration object in ASP.NET MVC which exposes a static property global configuration.configuration and inside of here you can see that there are four matters that are registered in ASP.NET MVC by default. This is a collection that you can add in you own formatters, you can remove the default formatters but if we look at what formatters are, we can see that formatters are things that take an object that I return from web API method like get all videos. This is just a string array. It takes one of these objects and build a representation out of it in a specific media type, for instance, there's the JSON media type formatter and the XML media type formatter and then a couple formatters to deal with essentially formed data. And the default formatter that's going to be used if the web API doesn't know what to do with a particular request, it's just going to be the first formatter here, which is the JsonMediaTypeFormatter. If I just go ahead and say continue any application and then tell it to stop debugging because we don't need to look at that anymore and we come back into curl, this is a JavaScript object notation representation of that string array. It's something that is easily consumed by JavaScript and the reason it came out in JSON is because if I issue this request again with a dash V to say, make this robust then we'll see everything that went between the server and the client. And one of the things I can see is that when curl sent this request to the server, it sent it with an accept header of star slash star, basically give me anything, give me everything. So there wasn't any specific piece of information in there that the web API could use to say oh, they went XML or they want JSON. Well, we're not sure. So we'll just fall back to the default which is JSON, but I can with curl, using a dash capital H send off a specific header like an accept header and say please give me application slash JSON and I get the JSON back that I would expect and let me take the robust flag off of here. So we don't see everything anymore and now I want to change those to be, please give me XML. And the web API happily serializes that array of strings into XML. And presumably, I would ask for XML if I'm in an environment where it's easy to parse the XML that comes back. And let us try one more trick, let's say give me all the videos, but serialize them as image slash PNG or get back JSON. That's why this is content negotiation. I'm saying I would like an image, but since the web API didn't have a formatter register to take some arbitrary CLR object to make an image out of it, then I just get back with the default formatter, built for me which is the JSON. And hopefully, this demonstrates to you what content negotiation can do and the reason it is important is because now I have a web service that is very highly interoperable. If someone is programming in a JavaScript environment or a Ruby environment and they have a library or an API that makes it very easy to consume JSON, then they can walk up to this web service and send along an accept type that says please give me JSON back. But if I'm in an environment like C sharp where it might be easier to parse XML, then I can request the XML instead. ( Pause )

Parameter Binding

Now we've seen how content negotiation plays a role with the web API and of course that influences how we return data to the client. We no longer return action results or view results from these methods. We just return objects in the content negotiation algorithm in the formatters. They all step in and serialize the object into the media type the client wants. So that's a little different. And what's also different is how we map incoming data into parameters in an action method. I've already shown you that we can get a simple integer ID parameter into an action method and we did that when we went to slash API slash videos, slash five with the get request. The 5 represented an ID and we ended up in the action taking an ID parameter and the value was 5. So by default, the web API assumes that primitive types like integers and strings, it assumes they'll be somewhere in the request and it's going to go looking for them, but it's not going to look in the HTTP message body, otherwise it's like normal MVC model binding in the sense that the runtime will look in the query string and then look into route data that's where it found our ID parameter when we did slash video slash five. And it converts that value and passes it into the action, but it's not going to look in the message body. If you want to pass a primitive type in the message body, there is an attribute you need to use to tell the web API to look in the body and that is the from body attribute. Complex types on the other hand, the web API assumes they will be in the HTTP message body, like the body of a post request or the body of a put request. Post by the way is generally used to update a resource, if we want to update a video we could post to the video API controller while put is generally used to create a new resource and of course get is to read and delete is to destroy. I'll show you examples of all these in just a bit, but this is a significant difference from MVC where you could bind any sort of model. Web API expects a single complex model in the body and the content type of the body will decide on how to, to serialize the message body. So these formatters, the JSON formatter, the XML formatter. A form URL encoded formatter they will look in HTTP message body and when the client says "Hey I'm posting you JSON" they'll try to deserialize JSON, that JSON formatter will jump in and try to turn that into the object, the complex object that you expect in your method. Let's experiment with all this a bit. Let's use Curl to fire off a couple more request to this GetMethod that takes some ID parameter just to see how it behaves and we already know that this request is going to succeeds /API/Video/5. If I issue that we get back a response that basically says it found the value 5 to put the value in the response. And the reason that works it's because of our route configuration. We told the MVC runtime when you see something like API/Something/SomethingElse to treat that something else as the ID so that particular piece of the URL gets placed in route data and then when the MVC runtime sees that there is an action method available that needs an ID parameter, it's able to find it in route data, invert it into an integer and pass it into this. But this also works if I don't pass it in the URL but if I pass in the query string so there we get value 12. So it's looking in the query string, it's also looking in route data and that's another area of extensibility in the MVC framework if you want it to go off and look other places too. But I'd like to turn our attention now to the post method because eventually we're going to need to update videos. So it would be nice if we could post the video and make sure that we get the proper value back. We're going to try this with just a string first. So we come in Curl and say that the method now is post and this particular method right now doesn't need an ID parameter. So I'm going to take that away but I am going to pass data so that's done with a -D. And with data, typically you're providing form data. So form data usually looks like value equals Scott and pass that along but we got null value back and there're actually 2 reasons for that unfortunately. The first reason is that what the web API expect for single primitive parameter is just equals Scott that will get it in value but it's not going to look into the message body that equal Scott goes into the message body until I add that attribute on body and now let me do a build and let me send off the request and now you can see we pull that value out and let's look at it one more time with the -D and let me open up this window a little bit so that we can see that what Curl did-- let's first of all send off that post to /API/videos, said that the content type was application/X-form-encoded that means that the form URL encoded formatter jump into play that looked in the message body and pull that value out. Now you may think that that syntax is just a little bit worky the fact that I had to say equal Scott and not value equal Scott, but don't worry because when you're start working with actual complex tapes this works the way you would expect it. So just to do a real quick test, let's go ahead and create a video class and let's say that this class has an ID and let's say that every video also has a title and now let's go to the post method instead of taking a primitive type, we'll take a video parameter and I will return that video parameter just so we can see what comes out on Curl and put it here too. And I think I'll just need to make that a public class so that the C# compiler is happy. But I should be able to do a build. And now I can come down and let's do a post that says ID equals 1, and you can see we got back a JSON representation of the video with the ID property set to 1. So we got the ID through there and I can also say N title equals MVC4 and we got that video back. So hopefully this is giving you some idea of how a web API will work. When someone calls get, or get all videos, what we're going to do is just return objects, we're going to let the content negotiation work to determine the best format for the client, the best representation of that resource and we can also receive data the same way, like client wants to post a new video or post to update a video to our web service, we can just take a complex tape as a parameter, as a video and the formatter is in the pipeline in the web API will figure out how to take form encoded data or JSON encoded data or XML encoded data and turn that into an object force and then it's just up to us to do the work of saving things in the database and so forth. That's what we're going to take a look at next.

Implementing GET

Now that we know a little bit about the web API, let's actually build out a web API that talks to a database and saves and delete videos from that database. In order to do this, I've gone ahead and created a video plots. A video just has ID title and length properties, we're going to keep this very simple to start with. And I've also created a video DB. This derives from the entity frameworks, DB contacts. I've included a DB set of video and this is everything we need to get started with the entity framework in version 5, this is the code first approach to creating a database. And something else I can do with the entity framework version 5 is manage migrations of my database, schema migrations and DBL statements. I'm going to open up a package manager console window. So first I'll search for that up here in Visual Studio. And then type the command enable migrations. So this is automatically part of an MVC4 application when you create a new one. It adds a reference to the entity framework version 5 and you have these commands available in the package manager console. What enable migration will do is add a migrations folder to solution explorer. Inside of there is a configuration class that in figures the migrations for this project. I'm going to turn on automatic migrations just to keep things as easy as possible. And this also gives me a seed method. What I can do with the seed method is every time there's a schema change or when the entity framework first creates a database. It will invoke this method and I can pre-populate the database with stuff that I need in there for development. So populate look up tables. Or in this case, maybe what I want to do is just go ahead and add a few videos into the database, so that there're always some videos in the database. So the way I will do that is to walk up to the DB contacts drive plus that we have, the video DB and take context.videos.adderupdate. This expression, lambda expression that you see here, that's the way of telling the entity framework how to differentiate one video from another. So it's literally saying, look at the title property of the object that I'm trying to add and make sure that's not already in the database. Because the seed method will run every time there's migration. We can have multiple migrations and I don't want duplicate instances of MVC4 in there. I'm telling the entity framework, check the title. If you don't see an MVC4 record in the database, add this one. Add a link record and then cotext.savechanges will just make sure everything gets flushed the database. So with that in place and everything saved and built, I'm going to type update database. And this will go out and by default and store this database that I have and run the seed method. It's going to create up on local DB which is the new version of SQL server express here for Visual Studio 2012. And this gives us everything we need to get started. No connection strings. No other configuration necessary. And so with that in place, I can start to think about how to actually implement get all videos. And I'm thinking that every action inside of this controller will need access to the video DB. So I'm just go to go ahead and create a private build here to hold a reference to that, include the name space for videos.models. And then let's have a constructor here that will construct an instance of that and initialize it. So I could say, "DB equals new video DB." And I'm going to set an option that makes life a little bit easier for me later on. I'm going to turn off proxy creation. Proxy creation with the entity framework is really there to enable change tracking and lazy loading. But when you're in an API scenario, typically what you do is grab data from database and then immediately serialize it and send it to the client. Or you receive data from the client and you apply it to the database immediately. There's no need to do change tracking or lazy loading inside of an API controller. And that actually simplifies things a bit if I just turn off the proxy creation. And so with that in place what get all videos would look like. What I wanted to do is actually return an IEnumerable of all the about the videos that are in the database. And so that's as easy as saying, "Return DB.videos" and were finished. I can do a build. And now let's just test it out in a browser, before we write the webpage that consumes that. So local host/API/videos this will send off a get request and what the get request here returns is some XML that has these two videos that we see at the database with.

Showing Data

The next step I want to do is change the homepage of this application to display videos by writing some JavaScript that calls get all videos and putting that information on the screen. Since I want that to happen on the home page that means I'll need to go into the views folder and open up the index view of the home controller. I've currently removed all the HTML and Razor code that was in here so that's we've just have a blank homepage. What I want to do is add some script to display our videos on this home page, but not do it through a Razor view, but through a call to the web API. And I'm going to need some help from some other JavaScript libraries to pull this off. The first thing I'm going to do is come into the package manager console and say update package jQuery. I just want to make sure that I'm running with the latest version which should be 1.7.2. You might have a different version when you do a file new MVC4 project, but if you run an update package jQuery, you'll have the latest version and the other thing that wants to do is install a package. I want to install a package called Handlebars.js. This is templating library for JavaScript. There're lots of different templating libraries out there. Pick the one that you like the best. I like Handlebars because I like Handlebars syntax is the best. But you might find one that suits your needs a little bit better. With Handlebars in placed, I know I need to include that script. I need to write some script for this homepage. There're a couple of different approaches I could use to do that. I could create an extra JS file that gets included in here. But just to keep things simple, I'm going to keep all of our markup and all of our JavaScript in the same file which is index.CSHTML. And I have an ideal place where I can place scripts because the layout view for an MVC4 application includes at the bottom of the page, a render section for an optional sectional scripts. This means that you can override that and say that I want to create some content for the section called scripts and I know I can include as much script code as I went inside of here or include as many script tags as I need and I'll go to the bottom of the page where they load very efficiently. So let's me create a script the tag where I will write my JavaScript. But before I forget, let me also go in make sure that Handlebars is available. I'll just drag up from the Solution Explorer and drop it here. We can get rid of the type attribute that's not needed in HTML5 and just say source equals tilde slash scripts slash Handlebars.js. MVC4 will make sure it will resolve that to an app relative path. Okay, so with Handlebars in placed, I think what I want is an abstraction that represent my server, the thing that I call into to get videos, to update a video, to delete a video. So you think that JavaScript module pattern, I'm going to declare a video serve object and that's going to be set equal to the following function evaluation. One of the things that I'll need to know when I'm trying to reach the server is what URL to go to. So let me paste in a piece of code that will build a video API URL. This is going to be done using a Razor expression and calling into the URL Helper, calling the route URL method. Well, route URL can do is when you give it the name of a route and some parameters it will build a URL for you, like /API/videos. So what we have to tell route URL is the name of the route that's registered in the system that we want to reach or that we want to use. So if you remember in AppStart when he had a route config, we had a HTTP route defined as default API. That's the route that we want to use. So in order to use to that, we really have to do two things. First of all, specify default API as parameter here and number two, what they don't tell you in the documentation it that you have to pass in these anonymously typed parameters object, a property on that object called HTTP route and it doesn't have to set to any particular value. It just has to be present as a property in order for the MVC runtime to correctly resolve a route to the video controller to produce something that is /API/videos. So this is the call, we want to reach those controller using this route, that should give us the nice variable there. And the other thing that I'm going to setup right now globally is some sort of Ajax Air handler. So if anything goes wrong on any call to the server, I can display in this case an alert that shows me the status of HTML HTTP request that is literally the HTTP response code whether it would be a 404, or 400, or 500, and also the status text that came back from the server like found, not found, or invalid method. Yes, we should give the user a nicer user interface than just an alert box, but this will give us a good start. It's going to keep things simple will allows us to immediately see if anything fails So with that all in place, I can add a method here that allows me too call the API controller, do a method called get videos. All that's going to do is use jQuery's dollar sign dot AJAX method to say let's kick off an XML each to be a request to this URL. If anything goes wrong, this is the global error handler for any jQuery, AJAX request that has an error code that returns-- we're just going to kick that off here and return that object, I'll show you how you can use the jQuery promises that are behind dollar sign dot AJAX and dollar sign dot get JSON, and dollar sign dot post, they all return promises that we can build on. And finally, since this is a module pattern or a revealing module, in this case, I need to actually return an object that will get set to this video server variable and that object will have a get videos member that points to my private implementation of get videos. So, that's all set up and ready to use. Now, I just need to write some code that will call that method, process the data, and update the UI. So, that, I'm also going to place into a module, this one doesn't have to be assigned to a variable, everything is going to be private inside of here. And it's inside of here, one of the techniques I use is declare a module and inside of that module, have the code that you use to handle the DOM ready event. So, this is writing up the DOM ready event and jQuery, I'm saying, when the DOM is ready, call a method named the refresh videos, what would refresh videos look like? Well, that could be a private method inside of this module that might look like this. Refresh videos is a function that calls video server, that's the object that we built up here, call into video server, call the get videos method that will return dollar sign dot AJAX, which is a jQuery promise. One of the things you can do with a promise is to say when you are dot done, when there's an error or when you failed, when you succeeded, you can register all sorts of call backs into it. The one I'm just particularly interested in is when you are successfully done. And when you're done, I want to show all the videos. So, we need to take the data that comes back from the server hopefully in JSON format because we're calling it using JavaScript. I want to take all the videos that came back and get them onto the screen somehow. And this is where handlebars comes in because with handlebars, you can define a template using HTML with little commands inside of it and tell handlebars to take that template, combine it with some data, and create some HTML for you. So, back up here in the view outside of the script section, I'm actually going to define a script that could go into the side of the script section too, but it really doesn't matter because this isn't a real script that's going to have executable JavaScript code inside, and that's why the type is set to text/HTML. This is not script, this is not the HTML that I want to display, this is really just a template that I need to give to handlebars so that it can create the table with videos inside. These are the handlebar instruction saying, "You give me an object and what I'll do is for each video in that object," so it expects a collection of videos, "I'll write out a table row. I'll give each row a data-ID attribute with the video's ID property. I'll create a table cell with the video's title, create a table cell with the videos length, and add two buttons to each row, and I'll do that for each video that I found." So, that's the template that we're going to use. And one of the things that you do with handlebars is you compile these templates. To-- back down here inside of my module, I'm going to create a variable to hold all of my compiled templates because I know I'm going to need, I think, at least two. There's going to be a template to display a list of videos and there's going to be a template to update a video or edit a video. Something will perform an input so I put a video inside of. The other piece that I'm going to add before I forget about it is that when I render this template and it produces HTML, where do I want to put that? I'll create a div here called video table output, that's where I'll place the output of this template. All right, so back down here, we have a variable to hold our compiled templates, what I'll do is I'll add a function that will compile templates for me, I mean, add that here. What compile templates will do is add a video table property to this templates object. And that video table property is set to handlebars.compile in my script. So, video table, if you remember when I scroll up, is the template that we have here. We're going to take that template, compile it, put the result into this video table member, that will be a function that I can invoke and pass in the videos that I want to render with that template. So what I'll have to do is when the DOM is ready, make sure that we compile the templates first and then refresh the videos. When we refresh the videos, we'll get the videos from the video server and then we'll show them. Show all videos now is a pretty easy implementation that I have this-- now that I have this templates in place. Show all videos is a function that we'll get in both-- we'll get the data from the XML-- HTTP request object, that'll just be the array of videos. I'll show up to that video table the compiled template that we created, tell it to render that template using the data that we got back from the server. That will produce some output, with HTML output so we walk up to that div, the video table output div, put that HTML inside of it. Quite a bit of script that we've written so far, just to do one simple thing, but we will be able to keep building on top of this, so other operations will be a little bit easier. I'm just going to save everything and build to make sure everything is in sync and then run the application with my fingers cross to see what happens. And there we get our two videos from the database on the screen. Behind the scenes are JavaScript, made a call to that web API controller and produce some JSON with the videos inside. We run it through a Handlebar's template and put it on to the screen.

Implementing PUT

For our next scenario, let's implement all the functionality we need so that when the user clicks on edit a video, they can change some of the properties of that video, save it back to the server and see it refreshed in this list. This is going to require a bit more code but we're starting to form a base so that when we implement delete, and ultimately I also want to do create a new video, those pieces are going to be easy. So when we edit the video, if I think through what has to happen, the first thing I might want to do when someone clicks on edit, this MBC video is I might want to go back to the server and get the latest version of that video. Just in case something has changed in the database, but primarily because I want to show you how to implement the get video, controller action, that takes an ID. Obviously now we're looking for a video and not some string. So I'm going to paste in some codes that represents getting a video. The first thing we'll do is find that video in our database, find it using the find API of the DB context. That's a really easy way to find an entity by its primary key value. And then if the video is null, what do we do? That's means we didn't find it in the database. Well, this is one of those cases where we want to have a little more control over what the clients gets back. We just don't want to send them an empty video. That's not really a good indication of what happened on the server, and this is where we can start taking control little more over the HTTP message that gets returned to the client. It turns out that HTTP already has a perfect way to represent what happens when you try to get something and it's not there and that's to return an HTPP status code of not filmed. That's the typical 404 error. And the way I'll do that is by throwing an exception. Something went wrong, you asked for a video that's not there. You obviously had some sort of identifier on the client that shouldn't have existed. So we will throw an HTTP response exception that will generate an HTTP 404 message ultimately to the client that would raise that Ajax error handler on the client and tell the user that they had a 404 or Not Found error. But if we did find the video we'll just hand it back and let the formatter serialize it into whatever format that the client requested. So with that in place, what I'd like is some sort of representation here in my video server of a method that would allow me to get a video and it's going to be very similar to get videos in the sense that we doesn't need to use dollar sign dot AJAX to use that video API URL. But not we need to include the ID in that URL. So it should be something like slash API slash videos slash UI, is the ID. And I'll just expose that method. Here in the object that we assigned the video server, so I should be able to say videoserver.getvideo, and pass in an ID. So now I have to come back and think about what's happening in UI. When the user clicks on that edit button, first of all I need to know about it. And I'm going to need to know when they click on edit or delete. So I'm going to create, during start up time, or during DOM ready time. Just before we compile the template, I'll call into a method called, wire events which we haven't implemented yet but I know I need to wire up all these different events. And for right now I'll start off just by wiring up when someone clicks on an edit video button. So, if you remember in our mark up, if I scroll up here to the top in our template, this edit button, if I click on that, it has a class of edit video. And in jQuery, there's a really great way to subscribe to events on elements that don't even exist yet, the elements that may appear in the future, and that's what's going to happen when we create that table of videos. One or more edit buttons are suddenly going to appear. A great way to handle that with jQuery is to use the on method. What's basically I'm saying, anywhere in the document, if someone clicks on something that has a class of edit video invoked this method, edit video. And now, what's that going to do? I would expect if I click on this, a little bit of extra UI will appear down here that has inputs of the video title and the video length and that means, I'll probably need another template so that given a video, the latest version of that video that I get back from the server, I display that UI. Let me go ahead and add another template here called video edit. And so again, it's a script with the type of text/HTML. That means it will not be rendered as HTML. It will not be interpreted as a script. Instead, it's more of just a resource that's sitting here, waiting for me to template it with Handlebars. And when I do template it and output it, I want it to appear beneath the table of videos that we have. So, this is where the output would be directed. And this is a Handlebar template which means, it should be complied. Fortunately, back down here in my module, I already have a method that will compile all my templates. So, I'll just add another function here that will compile that video edit template and make it available through that template's object. So now, the template is ready. What I need to do when someone clicks on the edit button, is I need to find out the ID of the video that they're trying to edit and then ultimately, grab it from the server and use that template to render it. That template by the way, it has inputs for the video title and the length of the video. And it's using the Handlebar mark up to insert a proper ID value into a hidden input, a proper title and the proper length of the video if the user had a chance to edit it. I'm also using some HTML5 attributes like required and min and max to constrain the input and provide some validation support at least that the user's browsers supports it. You could always add some additional scripts to add those features and if the browser doesn't support it natively. But the current problem is, they click on the edit button, we need to be able find out the ID of the video that they want to edit. Fortunately, that's going to be a relatively easy because when we render that table of videos, we put a data-attribute in here with the video ID. So, I just need to be able to navigate from this button up to this TR element and pull up that data ID. And that's relatively easy with jQuery. I can just go ahead write a helper function right now down here in this module called getID where you give me the element that was clicked on and I will go up and look through the parents of that element to find the TR element and pull out the data ID attribute. That will return the ID value. And that means, we just about have everything in place that we need to implement this edit video. Edit video will look something like this. When you want to edit a video, first I'm going to get the ID. This will represent the button that was clicked on. So, the getID function was implemented so that we should be able to find the ID given that child element. Once we have the ID, we can get through the video server, get the current data of that particular video, and when that call is done, we can show that video for edit. Show video for edit is going to look very much like show all videos because we need to use a template to generate the output that we need. So, give me the video edit template, render it using the video that we get back from the sever. When that produces some HTML output, we'll walk up to this element. Instead, it's HTML equal to that output. So, we're not saving the video yet, we're hopefully just getting it on the screen. Let me run this to see if it works. We'll do it quick. Control F5 to run without the debugger. This homepage will hopefully load. We'll get some videos on the screen. And now, I want to edit the link course and there, my input has popped down. So, I should be able to type and change the title or change the length. And now, we need to wire up this submit button and actually have it save changes to the server. So this Submit button was created inside of this template. It has an ID of save video. That's another event that I want to wire out using jQuery's on method because this template could appear and disappear throughout the lifetime of this page. So back down in my wire events, in addition to wiring up click events on Edit button, I'll also wire up click event on anything with an ID of save video and call this method, save video. Inside of save video what I would have to do is go out to the forum that holds the ID, the title and the length. Those would be input in that edit template, dig out all those values and you could use a library like knockout to do automatic data binding for you so you don't have to do this manually, but I'm just going to have it manually here to try to keep things a little bit simple. We'll construct a video out of what the user has typed into this input forum including the hidden ID input and then we'll tell the video server to update that video. That's not a method that we have implemented yet but once the video is updated we'll refresh our list of videos to get any changes from the server and clear out that edit template so that it disappears. That's a very easy function to write. I'll go ahead and add that now. Clear out the video edit output so that it disappears after the user click submit and of course that's only done if it's successful. We can add all sorts of additional error handling in here for the user to make it work nice for them if something goes wrong on the server. So what does video server dot update video look like? We'll work this from JavaScript back to the controller. Well, if I'm updating a resource, that typically means I send a put request to the server. So let me go up to my video server and in addition to get videos and get video, we'll have an update video. Update video will use that video API URL concatenated with the ID and do a put to the server passing along the video as data. What jQuery will automatically do when it does dollar sign dot ajax when you pass along some data is it will form encode that particular object into the HTTP request message and form encoding is something that the web API should be able to handle and deserialize that video. So now, I just need to make sure that I expose that update video method on the public interface that we're exposing from this. And now it's time to come over to the video's controller and look at implementing the put operation here. So the put operation is a little bit different from some of the operations that we've done in the past. What I want to do is not take an ID in a string value, what I want to do as I paste this in here is take an ID and a video and have the web API formatters just automatically deserialize a video from the request body. Remember complex types, it always looks in the request body. Primitive types like INT, it will always look in the URL. So the ID will need to be in the URL like api/video/5 and then form encoded into the request will be the video. The first thing we'll do is check to see if that video-- if it's valid, if the model state is valid. This is very similar to model binding in regular ASP.NET MVC. The model binder knows that the video might have certain required properties or non-nullable properties or apply regular expressions to some of the string values that are on that video. And any of those validation checks fail, model state will not be valid so we don't want to save it into the database. But if model state is valid this is how with the entity framework code first the API, we can attach an existing object that's already in the database and tell the entity framework just to update it. So it's as easy as doing tv.entry on that video, in other words, here's a video that's already in the database but I'm telling you about it and I'm telling you that it's modified. Now please go out and save the changes. So if save changes works, what we'll do is we'll create a response that says that everything worked, this is essentially an ATP status code of 200, everything worked and by the way it will give you the updated video that we have. And you'll notice this method no longer returns a video like get methods did, this returns an HTTP response message. This is another level of control that you can use in the web API to carefully craft the response that you send back to the client. In this case we want to send back that everything worked and by the way here's some serialized content to pass along to the client. If there were some sort of concurrency exception when we updated the database we can create a response that essentially said that's a 404 error. That might not be the best choice of a response period in this situation but that will certainly tell the client that something went wrong. And if model state was invalid we can pick a particular HTTP status code to say sorry, you gave us a bad request, something was missing or something was not properly thought out. And it's up to the client to interpret these different codes so you want to make sure you pick a code that's descriptive of what went wrong and bad request would be a pretty good code to return in this scenario. So let's try this out. We might have everything in place where I can be able to click edit updates and attributes, click submit and actually have it saved on the database on the server. The homepage appears, my video has come up, I have to edit the link video and just change the length a little bit, change it to 198 minutes and submit and you can see that that refreshers up here. We can change the MVC video, maybe make this MVC 4.1, that changed too. So everything is working so far. And now we really have some code that we can build on, the delete and the create scenarios are going to be very easy compared to what we've done so far.

Implementing POST

Now let's see how it easy it would be to create a new video using the web API in JavaScript. Create usually comes to the post method and since I'm already in the video's controller let me just go ahead and replace this post method with a post method that will actually save a video on the database because it's going to take a video as a parameter, it's going to return in HTTP response message because we want to have some low level control over what's going to be in that response. I'll show you why in just a second but if everything about the video is valid and model state is valid, we'll tell the entity framework to add that video to its videos collection. We'll tell the entity framework to save changes and that should issue an insert statement that saves that video on the database. It will also retrieve the new identity value that was generated for that video and propagate it back and put it into the video. And that's important because of these next 3 lines of code. These next 3 lines of code are here because the HTTP specification actually says if someone does a post to your server and that post operation creates a new resource, what you should return is an HTTP status code of 201. That's the universal status code for hey, you've created something and that tells the client that something was created. And in addition to using that code instead of just a regular 200 everything was okay, in addition to that, you should also include a location header that tells the client where they can go to get to that newly created resource. So literally, a URL that they can use and issue in a get request to get that resource in the future and that's exactly what these 3 lines of codes are doing. They're building a response that uses an HTTP status code of 201 created, also formatting the video into the response body. And then setting the location by generating a URL using url.link, generating a link that goes to that default API route and telling it what the ID will be, and that literally will come out to be something like location equals /api/videos/5 and tells the client exactly where to go to get to this new video. That's the response that we will return and that's why we're using HTTP response message here instead of just video. So we can control the status code, control some of the headers that go in that response. If model state wasn't valid for some reason we'll return a response that says bad request and if something catastrophic happens inside of here like save changes there was an exception, that would generate an HTTP status code of 500 which also tells the client that something went very wrong on the server. That should be all the server side logic we need. What we have to add to the UI is some sort of button that I can click on to say I want to create a new video. So if I look in the index video, let me add a button directly underneath that table of videos called the Create Video button. So obviously, I'll need to wire up a click event on that button so let me come down to where we wire up all our events on this page and in addition to wiring up click events on clicking edit and clicking save, we'll also wire up a click event on the create video that will just call a function called create video that will be very similar to the edit video function that we wrote earlier. In fact, we could probably figure out a way to share the code here but on create video, what I'll do is instead of going out and getting a video from the server, just create a new empty video with an empty title 0 link, 0 ID and call and show video for edit. That's a function that we've already implemented that pops down that little bit of UI that includes a submit button and when the user clicks on that submit button, it comes back to save video. This is a method that would have to adjust a little bit because right now it seems that we always to update an existing video. But now we might need to add a new video instead of updating an existing video. So let me add some code here that will just check that the video has an ID or not. If the video does have an ID greater than 0 or it's not equal to 0, we'll assume that you want to update an existing video. Otherwise, if video ID is 0, we'll assume you're trying to add a brand new video. And regardless of which operation you perform when it's finish we need to refresh all the videos in the list and also clear out that edit box. So we're nearly done. We just need to add something to the server object up here that will allow you to call add video and I'll place that here after update video. Add video is just going to take a video and put it in the data in a post request to the video API URL. Add the semi colon here and add this to the public object that we exposed, so add video we'll go to that function we just wrote-- called add video. Now, I should have everything in place if I save everything and do a build and we bring up the browser and refresh it. Hopefully the create video button will appear. And shortly after that the videos will appear. This is all good. Let's create a video-- a C sharp video and just set the link to something like 32, click submit, and there we just created a video. Create one more, let's call this one entity framework 180 and click submit, that seems to work too. So not much we have to add there, delete is going to be even easier.

Implementing DELETE

Now finally, what implement the functionality is that when I click the delete button I can delete the entity framework. I know that wouldn't make a lot of people happy. Obviously the first thing I need to do is wire up a click event on that delete button. We can do that down here where we wire up the other events-- the edit events and the save events, I'll just add another call to the on method saying if anyone clicks on something that has the class delete video, I'm going to call this method delete video. That method is a pretty straight forward implementation, very similar to what we've done before. We just need to go out and get the ID of the video where user click delete, and tell the video server to delete that video. When it's all done, we'll refresh the videos that are on the screen. But we do not have this method yet on our video server, we need to implement delete video. All of these methods are pretty straight forward and simple. They're all just variations on dollar sign dot AJAX. This one need to take the ID, formulate the URL to the video API, and issue a request with an HTTP method of delete. That's all we need to do there. And lastly just add delete video as part of the public API. That should gives use everything that we need on the client. We haven't implemented delete on the server yet so let's flip over to the video's controller. Find the delete method which I believe is down here at the bottom and replace this implementation with on one that takes some ID. Again, return some HTTP response message. What we'll do is we use the entity framework to find the video that you want to delete. If it's null, return an HTTP status code that says "sorry, we're poor we didn't find this." Otherwise, tell the entity framework to remove the video, save changes, that should issue a delete statement. And then assuming everything went okay, we'll return an HTTP status code of 200, include the representation of the video when we do that. If there was some sort of concurrency exception which we done have concurrency checking turned on with the entity framework model that we're using. But if there was an exception we have it set up to create a not bound status code. That should be everything we need for delete, very simple now that we have most of the code written. So let me refresh the page, our videos will show up and we will delete the entity framework or delete C sharp too. So there we have basic CRUD functionality, implemented in a web API controller. And we build a page that is using that API to allow user to create, read, update, and delete videos. Now if that was all there was to that you might say "But Scott, I couldn't have done that with a regular MVC controller." Nothing really special about web API that we shown here but we're not quite finish yet.

XML Client

( Pause ) One of the distinguishing features of the web API when compared to traditional MVC controllers is the fact that web API does content negotiation. We talked about this topic a little earlier in the module but I wanted to make it a little more concrete by building an application that use the web API and asks for XML data. So the client I'm building is a console mode application. I created those just by going to the solution and saying I have a new project, selecting a console mode application. And then another great feature of the web API is the fact that you really don't need any specific toolset or framework to get information from the web API. If you can make an HTTP request, you can get data from a web API. But since we are in dot NET and using C sharp, I'm going to use a web API client that ships with the web API that makes it very easy to make HTTP request and process HTTP responses. This client, the HTTP client is a part of the web API. You can install it into any application by saying Install-Package Microsoft.AspNet.WebApi.Client that'll add all the assemblies that you need. And I do want to point out, even though this HTTP client is part of web API, it doesn't mean that it can only call into something that was built with the web API. It's really of a generic library in API for making HTTP request to anything and processing an HTTP response. So once that is installed, I can instantiate an HTTP client, I can say that, that client should always add an accept header, saying that we are requesting XML. And at that point, I can make a call to the server using GetAsync or PostAsync or PutAsync. They're all async methods that will send off a request to the server. I'm going to block the thread here and wait for the result. I can only get away with that because I'm in a console mode application and we can just force the user to wait for the result. That's not something you'd want to do if you're building a responsive UI, you'd want to await that result. But we're going to check to see if the result that came back had a status code indicating that the request was okay. If so, we'll take the content in that result and read it as a stream, load that into an XDocument which is part of linked to XML, create a namespace because the XML elements that are in there by default are created by the data contract serializer and they'll be inside of a namespace that contains sort of the C sharp namespace that the particular model object is a part of. So we'll create that namespace then issue a query looking for all the titles of the videos that are out there and writing them out. And so what does this look like when I run it? Let's just go to debug, start newInstance. I can get back the two videos that are in the database, I can do that from a console application that has no references or dependencies on that video's web application. It's strictly making HTTP calls requesting XML, parsing that XML, and displaying the results.

Summary

In this module, we had a basic introduction to the web API. We learned that it really embraces HTTP. It allows you to build web services that embrace HTTP. It allows you to build web services that are scaleable and highly interoperable, scalable because you could take advantage of using HTTP GET which can include caching, interoperable because you don't depend on any particular specification or framework or toolset when you're building your service. And it's also just as flexible as ASP.NET MVC. It has many of the same extension points. You can plug in your own custom model binders, your own custom formatters, action filters, and all the rest. We also took a quick look at building a client using web API and the HTTP client class. And one thing that I didn't have time to show you is the ability to self-host a web API. That's not something you can really do with ASP.NET MVC, it's a bit tied to the ASP.NET runtime. But web API is something that you can take and host inside of a Windows service or a console mode application or a WPF desktop application. And you can have clients that call into those applications.