Hello World

Putting It All Together

# Introduction

Last week we created a simple layout for the UI of our Web app. We have three elements:

* The greeting
* The locale ID
* An image (this is a placeholder)

Our API has greetings in four different languages: U.S. English, French, Spanish and German.

Now we'll take the final step and connect our application to our API and pull our Web page elements from the API instead.

# Modifying the API

Before beginning, add two additional Greeting objects to ValuesController (Spanish and German) if you haven't already.

Our plan is to replace the generic image in our interface with the flag of the country whose greeting we are displaying. For example, if we display "Hello World!", our locale is en-us (United States English), then we'll display an image of the American flag.

There are a number of ways of serving binary files via a Web API, but we're going for the simplest.

* Install the NuGet Package Manager extension
* Press F1 and enter Nuget in the command search field
* Select Add Package
* Search for StaticFiles
* Select Microsoft.AspNetCore.StaticFiles

|  |
| --- |
| **NOTE**: You will see multiple versions available. Examine the csproj file of your Web application (it's a text file) and use the package version number you find there. |

* When you save your changes, you'll be prompted to update your dependencies. Once that's done, you can continue.
* In Startup.cs, add

|  |
| --- |
| app.UseStaticFiles(); |

to the Configure() method.

* Add a folder under wwwroot called images
* Extract the contents of flags.zip to the images folder under wwwroot.

Now we need to modify our data model to include a new field, banner, that will contain the name of the flag image file for that greeting.

* In Greeting.cs, add the following property to your Greeting class:

|  |
| --- |
| public string Banner { get; set; } |

* Save your changes and open ValuesController.cs.
* Modify the contents of the greetings array to add the Banner field. For example, the US greeting will now look like this:

|  |
| --- |
| new Greeting { Id = 1, Salute = "Hello, World!", Locale = "en-us", Banner = "en-us.jpg"}, |

* Save your changes.
* Build and test locally to make sure your API still delivers random greeting data.
* To test the flag images, use the URL

http://localhost:5000/images/en-us.jpg

You should see an American flag in your browser.

* Once you've confirmed that your code is still working, commit your changes with an appropriate message.
* Push your new code up to Azure.
* Log in to the Azure portal.
* Select your API.
* In Application->Settings, click on CORS. (This stands for Cross Origin Resource Sharing).
* In the text field, add an asterisk (\*).
* Click on Save.
* Your API is now accessible to the public.

# Connecting the App to the API

* Open your Hello World app project in VS Code.

To keep things simple, we will add a script to our app that will query our API and place the elements on our page.

* Open Index.cshtml.
* Below the code for your button, add the following:

|  |
| --- |
| <script>  function loadGreeting() {  var xhttp = new XMLHttpRequest();  xhttp.onreadystatechange = function() {  if (xhttp.readyState == 4 && xhttp.status == 200) {  // Convert the text response to a JSON object  var obj = JSON.parse(xhttp.responseText);  }  };  xhttp.open("GET", "http://howdy-api.azurewebsites.net/api/values", true);  // NOTE: Substitute your own API URL for the above.  xhttp.send();  };  </script> |

Let's break down this script:

* <script>...</script> tags - this tells the browser not to display this but instead run the script and display the results.
* var xhttp = new XMLHttpRequest(); - The XMLHttpRequest class has methods that let you transfer data between a Web client and a server.
* onreadystatechange - this method waits for the server to respond, then runs the code defined by function().
* var obj = JSON.parse(xhttp.responseText); - At this stage, the only thing function() does with the server response is to convert it into a JSON object. (We'll do more later.)
* xhttp.readyState == 4 - The server request is finished and the response is ready.
* xhttp.status == 200 - The return status is OK. (no errors)
* xhttp.open("GET", "http://howdy-api.azurewebsites.net/api/values", true); - initialize a request. In this case, it's the same request your web browser makes when it connects to an URL.
* xhttp.send(); - Send the request.

We need to modify the placeholders for our Web elements so our script can find them.

* Modify the following lines where our page elements are placed in Index.cshtml like so:

|  |
| --- |
| <h1 id="salute" class="text-center"></h1> |

|  |
| --- |
| <h2 id="locale" style="padding:50px;"></h2> |

|  |
| --- |
| <div id="banner" class="col-md-8"></div> |

The id tag lets us reference these elements by name. We removed the inner content of the tag since we're going to replace them with the output from our API.

Now let's attach our script to our button so when we click it the script will run.

* Modify the following line in Index.cshtml like so (all on one line):

|  |
| --- |
| <div class="col-md-4"><button type="button" class="btn btn-primary" onclick="loadGreeting()">Howdy!</button></div> |

* Save your changes, build and run your app and make sure it still works. (Our function currently doesn't do anything so the only change will be that our page elements are missing.)

Now let's get serious and pull the data from our API request and use it to populate our UI.

* In the loadGreeting() function, after the JSON.parse() method, add the line:

|  |
| --- |
| document.getElementById("salute").innerHTML = obj.salute; |

* Run your app, click on the Howdy! button and you should see a greeting in your jumbotron.
* Back in loadGreeting(), add another line below the previous one (all on one line):

|  |
| --- |
| document.getElementById("locale").innerHTML = "Locale: " + obj.locale; |

* Finally, let's add another line to insert our flag image (all on one line):

|  |
| --- |
| document.getElementById("banner").innerHTML = '<img id="banner" style="padding:50px;" src="http://howdy-api.azurewebsites.net/images/' + obj.banner + '" alt=""></img>'; |

So what's going on here?

* document - this is the object that represents the Web page.
* getElementByID() - a method that locates a page element by looking at the id tag.
* .innerHTML - The space between the tags for the labelled element.

The string after this method represents what to replace in the identified element. So each of these lines says, basically, "go to the element labelled 'x' and replace the contents of the tag with the string in this expression".

* Save your changes and make sure your app now works as required.
* Commit your changes with an appropriate message.

# Summary

The ASP.NET framework lets us easily create a Web API and a Web app that consumes that API. With Visual Studio Code and git, we can easily deploy our software to the Internet via Microsoft Azure.