Worth 10% of ASP.NET part 2.

Due January 27, 9AM

Please only upload this word document with your answers in it to the appropriate drop box.

Contents

[Using Asynchronous Tasks in ASP.NET 1](#_Toc441040525)

[When Not to Use Asynchronous Methods 1](#_Toc441040526)

[When to Use Asynchronous Methods 1](#_Toc441040527)

[Using Existing Asynchronous Methods 2](#_Toc441040528)

[Creating Custom Asynchronous Methods 2](#_Toc441040529)

[Returning JSON in an MVC Controller 4](#_Toc441040530)

[Exercise 1: Asynchronous Methods 8](#_Toc441040531)

[Exercise 2: Serving JSON from an MVC Controller (Not Web Api) 9](#_Toc441040532)

# Using Asynchronous Tasks in ASP.NET

In MVC applications, only so many concurrent users can make simultaneous requests before the application starts to lag or even stall because the application has run out of “**request threads**”. You can however use **async** programming methods for time intensive calls for resources over the network to free up the request thread for another user. Asynchronous methods release the request thread while they wait for the resource to arrive. Once the resource is obtained the method takes another request thread from the pool.

In most applications using asynchronous methods will have no noticeable benefits and in some cases using them could even be detrimental. Use tests, profiling and common sense to measure the impact of asynchronous methods in your particular scenario.

## When Not to Use Asynchronous Methods

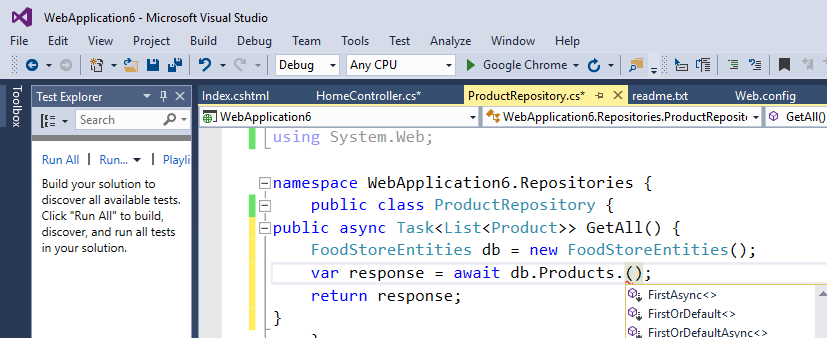
* Short operations
* Simplicity is preferred.

## When to Use Asynchronous Methods

* Delays occur during calls for remote REST services or large amounts of remote data.
* >= .NET 4.5
* Asynchronous methods.

## Using Existing Asynchronous Methods

To view asynchronous methods available to your objects, after the object reference enter a dot and the methods available to it will appear in the drop down. Asynchronous methods will end with **Async**.



## Creating Custom Asynchronous Methods

You can create custom asynchronous methods. One common way to do this is to start the method header with the keywords **async Task**. Then, you can use the key word **await** before each call to the asynchronous method.

Example 1: An Asynchronous Call to the Database

□ This example shows how to create a custom asynchronous method. It also shows how to consume both the custom asynchronous method **GetAll**() and an existing asynchronous method **ToListAsynch**(). In the real world this example really is not practical since the amount of data being retrieved is very small.

To build this example, create an empty web application and add an EDMX that references the *FoodStore* database. Then, create a *Product* repository class and add the following method to it. Note the return type is preceded with **async Task**. Also note how ***await*** precedes the call to each asynchronous method.

|  |
| --- |
| public async Task<List<Product>> GetAll() {  FoodStoreEntities db = new FoodStoreEntities();  var response = await db.Products.ToListAsync();  return response;  } |

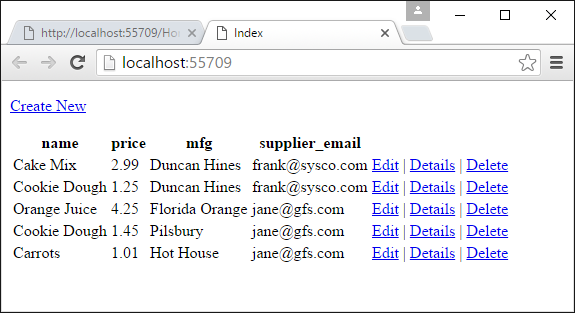
To enable your custom asynchronous method you will also need the following namespace references:

|  |
| --- |
| using System.Data.Entity;  using System.Threading.Tasks; |

Next, add an empty controller and place this ActionResult method in it. Notice once again the return type is an asynchronous task. Also again, *await* precedes the call to the asynchronous method:

|  |
| --- |
| public async Task<ActionResult> Index() {  ProductRepository productRepo = new ProductRepository();  var result = await productRepo.GetAll();  return View(result);  } |

Then, add a view with a List template which references the Product entity. When you run the example you will see the following output:



Example 2: An Asynchronous Call to a Remote Service

□ Here is a more practical example of an asynchronous call for a remote service. To build this example, create an empty Mvc web application. Using NuGet, install Json.NET. Then add an empty Mvc Home controller with these namespace references included:

|  |
| --- |
| using System.Net;  using System.Threading.Tasks;  using Newtonsoft.Json; |

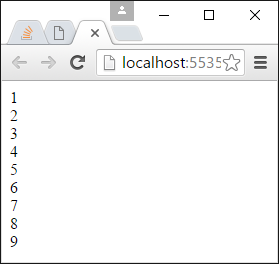
Next, replace your ActionResult method with this asynchronous version:

|  |
| --- |
| public async Task<List<int>> GetNumbers() {  List<int> numbers = new List<int>();  using (WebClient wc = new WebClient()) {  Uri uri = new Uri("http://ssdprogram.ca/testJson.php");  // Call async method.  string json = await wc.DownloadStringTaskAsync(uri);  numbers = JsonConvert.DeserializeObject<List<int>>(json);  }  return numbers;  }  public async Task<ActionResult> Index() {  // Call async method.  ViewBag.Numbers = await GetNumbers();  return View();  } |

After, add an empty view and place this razor syntax in it:

|  |
| --- |
| @foreach(var number in ViewBag.Numbers) {  @Html.Raw(number + "<br/>")  } |

When you finish, the data will appear on the web page:

d

# Returning JSON in an MVC Controller

A nice feature of ASP.NET is you can return JSON from your Mvc controller with the JSON Result type. Allowing JSON results in your application lets you avoid the hassle of using Web API if you are only serving up a few simple services. You can also mix and match serialization and Json.NET methods for building your JSON objects as you need.

Example 3: Using a JsonResult with Default Serialization

□ To start this example, create an empty Mvc project. Then add the following view model:

|  |
| --- |
| public class Movie {  public string Title { get; set; }  public string Genre { get; set;}  public int Year { get; set; }  } |

Then add an empty Mvc home controller Next, replace the default Index() action method with the following ActionResult methods. The Movies() method creates a list of objects and then serializes it to JSON. The default serialization works nicely for simple structures as shown here.

|  |
| --- |
| public ActionResult Index() {  return View();  }  public JsonResult Movies() {  var movies = new List<Movie>();  movies.Add(new Movie{  Title = "Ghostbusters", Genre = "Comedy", Year = 1984 });  movies.Add(new Movie{  Title = "Gone with Wind", Genre = "Drama", Year = 1939 });  movies.Add(new Movie{  Title = "Star Wars", Genre = "Science Fiction", Year = 1977 });  return Json(movies, JsonRequestBehavior.AllowGet);  } |

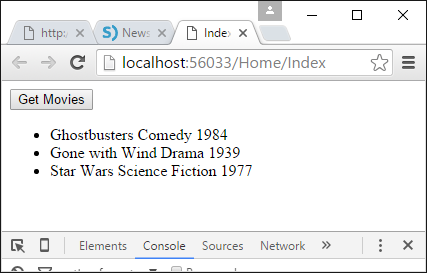
Next, using the NuGet package manager download jQuery. Then create an app.js file to the Scripts directory and add this code to it. Remember to adjust the port number.

|  |
| --- |
| $(document).ready(function () {  $("#btnGetMovies").click(function () {  var actionUrl = "http://localhost:60287/Home/Movies";  $.getJSON(actionUrl, displayData);  });  function displayData(response) {  if (response != null) {  for (var i = 0; i < response.length; i++) {  $("#movieList").append("<li>" + response[i].Title + " "  + response[i].Genre + " " + response[i].Year + "</li>")  }  }  }  }); |

Then create add an empty view for the Index() method and place this code in it. Double check the JavaScript references to ensure they are accurate.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta name="viewport" content="width=device-width" />  <title>Index</title>  <script src="~/Scripts/jquery-2.2.0.js"></script>  <script src="~/Scripts/app.js"></script>  </head>  <body>  <div>  <input name="btnGetMovies" id="btnGetMovies" type="submit" value="Get Movies">  <ul id="movieList"></ul>  </div>  </body>  </html> |

The output from this is:



Example 4: Using Json.NET with Serialization in a JSONResult

□ This example shows another way to perform the same task as Example 3. This time though the example uses Json.NET to perform the serialization. When using Json.NET notice that the ActionResult type must be used which is actually a parent to the JsonResult class. As well, note that the Json is delivered with the ContentResult method.

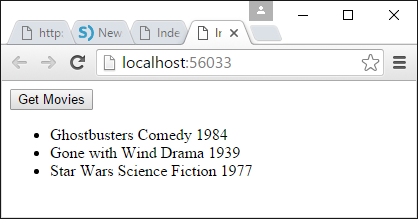
To build this example, start with the code from Example 3. Using NuGet add a reference to Json.NET. Then replace the methods in the controller with these revisions. This code uses Json.NET with serialization to generate the JSON though. To build it, replace your action methods in the controller class with the following code.

|  |
| --- |
| public ActionResult Index() {  return View();  }  public ActionResult Movies() {  var movies = new List<Movie>();  movies.Add(new Movie{  Title = "Ghostbusters", Genre = "Comedy", Year = 1984 });  movies.Add(new Movie{  Title = "Gone with Wind", Genre = "Drama", Year = 1939 });  movies.Add(new Movie{  Title = "Star Wars", Genre = "Science Fiction", Year = 1977 });  var json = JsonConvert.SerializeObject(movies);  return new ContentResult { Content = json.ToString(),  ContentType = "application/json" };  } |

A reference to this namespace is also needed:

|  |
| --- |
| using Newtonsoft.Json; |

When you run the project the data appears once again with the help of Json.NET:



Example 5: Using Json.NET without Serialization in a JSONResult

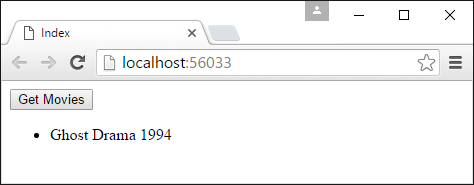
□ Just in case serialization becomes difficult to achieve directly you can build your JSON object step-by-step. This example shows how build up your JSON object incrementally with Json.NET and then return it in an action result. To build this example start with Example 4 and replace the Movies() action method with this revision:

|  |
| --- |
| public ActionResult Movies() {  dynamic movieArray = new JArray();  dynamic movieObject = new JObject();  movieObject.Title = "Ghost";  movieObject.Genre = "Drama";  movieObject.Year = 1994;  movieArray.Add(movieObject);  return new ContentResult { Content = movieArray.ToString(),  ContentType = "application/json" };  } |

The following namespace reference is required:

|  |
| --- |
| using Newtonsoft.Json.Linq; |

When you run the project the following output will appear:



# Exercise 1: Asynchronous Methods

Please answer the following question true or false. (0.5 marks each)

1. \_\_\_T\_\_\_\_\_\_ Calls to asynchronous methods in ASP.NET are preceded with the key word *await*.
2. \_\_\_\_T\_\_\_\_\_ A call for a large data set from a remote web service is a good candidate for an asynchronous method.
3. \_\_\_\_T\_\_\_\_\_ Individually, asynchronous methods are usually quicker to return data than synchronous methods.
4. \_\_\_\_T\_\_\_\_\_ Asynchronous methods free up the request thread for another user. When the asynchronous method finishes it takes an available request thread from the pool.
5. \_\_\_\_T\_\_\_\_\_ Asynchronous methods are always suitable when retrieving small amounts of local data.
6. \_\_\_\_T\_\_\_\_\_ Asynchronous methods help to enable higher numbers of concurrent users during times of high traffic.

# Exercise 2: Serving JSON from an MVC Controller (Not Web Api)

Create an application that uses Json.NET to return a list of all manufacturers and their discounts from the FoodStore database in JSON format. Invoke this routine and display the results using JavaScript.

Write a custom asynchronous method to retrieve the manufacturer list from the FoodStore database in a repository class. Show your entire repository class with this custom asynchronous method here: (2 marks)

|  |
| --- |
| namespace WebApplication18.Repository  {  public class ManufacturerRepository  {  public async Task<List<ManufacturerVM>> GetAll()  {  JB\_FoodStoreEntities db = new JB\_FoodStoreEntities();  List<ManufacturerVM> response = new List<ManufacturerVM>();  response = await db.Manufacturers.Select(m => new ManufacturerVM { mfg = m.mfg, mfgDiscount = (decimal)m.mfgDiscount }).ToListAsync();  return response;  }  }  } |

Show your entire MVC controller (not web api) which uses an asynchronous ActionResult method to call your custom asynchronous method to get all manufacturers from the database. Serialize the data using Json.NET and return this content to the view. Show your entire controller here: (3 marks)

|  |
| --- |
| public class HomeController : Controller  {  // GET: Home    public async Task<ActionResult> Manufacturer()  {  ManufacturerRepository manufacturerRepo = new ManufacturerRepository();  var manufacturers = await manufacturerRepo.GetAll();  //var query = manufacturers.Select(m => new ManufacturerVM { mfg = m.mfg, mfgDiscount = (decimal)m.mfgDiscount });  var json = JsonConvert.SerializeObject(manufacturers);    return new ContentResult  {  Content = json.ToString(),  ContentType = "application/json"  };  }    public ActionResult Index()  {    return View();  }  }  } |

Show the JavaScript that you use to call your REST service created above here: (3 marks)

|  |
| --- |
| $(document).ready(function () {  $("#btnGetManufacturers").click(function () {  var actionUrl = "<http://localhost:58992/Home/Manufacturer>";  $.getJSON(actionUrl, displayData);  });    function displayData(response) {  if (response != null) {  for (var i = 0; i < response.length; i++) {  $("#manufacturerList").append("<li>" + response[i].mfg + " "  + response[i].mfgDiscount + "</li>")  }  }  }  }); |

Host your site on a subdomain at Arvixe and keep it online until I can check it. Show the URL of your application for this question here: (2 marks)

|  |
| --- |
| http://jsonmvc.jamped.com/home/index |