

HTTP methods and status codes and using HttpClient

Estimated time for completion: 60 minutes

Overview:

In this lab you will flesh out the movies service by providing support for all the pertinent HTTP methods and by issuing different status codes.

You will then implement the client code to use the movie service, check HTTP response status codes and consume the JSON results.

Goals:

* Provide support for POST, PUT and DELETE
* Issue appropriate HTTP status codes
* Build the client application using HttpClient

Lab Notes:

The lab uses a movie review website as its base. Registered users can view movies and reviews. Reviewers can write reviews. Administrators can do most anything. The accounts (username and passwords are the same value) for the application are:

* **Registered Users:** user1, user2 and user3
* **Reviewers:** reviewer1, reviewer2 and reviewer3
* **Administrator:** admin

More HTTP methods

In this part of the lab you will implement the remainder of the HTTP methods (POST, PUT and DELETE) for the movie service.

Criteria:

* Provide support for POST, PUT and DELETE

Steps:

1. In this part you will implement the missing HTTP methods from the movie service.
   1. Open the solution from ~/Methods\_StatusCodes/before/service.
   2. Open ~/api/MoviesController.cs.
   3. Add support for the POST, PUT and DELETE methods. The MovieService member variable provides the APIs needed.
   4. Also in this lab, the WebAPI route is not registered. Add the route in ~/App\_Start/RouteConfig.cs. Make sure to use MapHttpRoute and not any other APIs.
   5. You can test these new APIs using Fiddler and the composer tool.

For the POST and PUT you will need to set the Content-Type HTTP request header to application/json and set the body to a JSON representation. The header in Fiddler will look like:

Content-Type: application/json

And the body will look like:

{"ID":1, "Title":"A Nous la Liberte", "Rating":"NR", "YearReleased":1932, "Description":"One of the all-time great comedy classics....", "Country":"France"}

Consuming the movie service with HttpClient

In this part of the lab you will build the client application to consume the movie service. To do so you will use HttpClient. You will check for success and failure responses and you will consume the returned JSON for use in the client application.

Criteria:

* Build the client application using HttpClient
* Check for success and failure responses
* Parse JSON results

Steps:

1. In this part you will implement the client-side code to make HTTP requests using HttpClient.
   1. Open the solution from ~/Methods\_StatusCodes/before/client.
   2. Open ~/MovieService.cs.
   3. The MovieService class encapsulates invoking the movie service.

Notice that there are five methods that need to be filled in. These methods correspond to the five methods on the movie service.

The user interface is already coded to invoke these methods, so your job is to implement these. Notice they all return Task<T>; the client is written to assume that the invocation will be asynchronous.

Also, notice that many of the methods return a MovieResponse. This class represents the outcome of invoking the movie service and contains a Movie object or an Error string. Populate these fields appropriately for the method being invoked.

If you have problems then you can debug the HTTP calls with Fiddler, but don’t forget to change the URL from http://localhost to http://localhost.fiddler.

**Helpful links:**

* [Basics of Task Parallelism and Task<T>](http://msdn.microsoft.com/en-us/library/dd537609.aspx)
* [Calling a WebAPI from .NET](http://www.asp.net/web-api/overview/web-api-clients/calling-a-web-api-from-a-net-client)
* [Using HttpClient](http://blogs.msdn.com/b/henrikn/archive/2012/08/07/httpclient-httpclienthandler-and-httpwebrequesthandler.aspx)

HTTP status codes

In this part of the lab you will issue Created (201), NotFound (404) and NoContent (204) HTTP status codes where appropriate. You will then update the client application to accommodate the changes in the service.

Criteria:

* Issue appropriate HTTP status codes

Steps:

1. In this part you will change the movie service to return the appropriate HTTP status codes depending on the situation.
   1. Open the service solution again (if you don’t still have it open).
   2. Open ~/api/MovieService.cs.
   3. Modify the Get, Put and Delete to return NotFound (404) if the id parameter doesn’t match a movie in the database.
   4. Modify Put and Delete to return NoContent (204) if the operations are successful.
   5. Modify Post to return Created (201) if the operation is successful. Return the newly created Movie object as the response body. Also, set the Location HTTP response header to the URL for the new movie (which will need to include the database-generated movie ID).

It will be helpful to use the Url.Link helper method to generate the URL.

* 1. Use Fiddler to observe the results on the network and test these new status codes.

Solutions:

The final solution for this lab is available in the ~/after directory.