

## Exercise 13 : Web API

### Objective

In this exercise you add a Web API controller to your project. You will use Postman to test the controller.

This exercise will take around 30 minutes.

### Referenced material

This exercise is based on material from the chapter "Web API".

### Adding an API Controller

	Open the 'Begin' solution in Visual Studio and compile (Shift Ctrl+B). The 'Begin' solution is identical to the 'End' solution from the previous lab.
	<p>In this lab, we are going to create a Web API controller.</p> <p>We will start by creating data transfer objects, or DTOs, which will be used by our controller.</p> <p>Add a folder to the root of the project, called DTOs</p>
	<p>Into the DTOs folder, add the following two classes:</p> <p><b>ThreadDTO:</b></p> <pre>using QAForum.Models; ... public class ThreadDTO {     public int ThreadId { get; set; }     public string Title { get; set; }     public string UserName { get; set; }      public static ThreadDTO FromThread(Models.Thread thread)     {         return new ThreadDTO         {             ThreadId = thread.ThreadId,             Title = thread.Title,             UserName = thread.UserName         };     }      public static IEnumerable&lt;ThreadDTO&gt; FromThreads(IEnumerable&lt;Models.Thread&gt; threads)     {         return threads.Select(t =&gt; FromThread(t));     } }</pre> <p><b>ForumDTO:</b></p> <pre>public class ForumDTO {     public int ForumId { get; set; }      [Required]</pre>

```
[MinLength(4)]
public string Title { get; set; }

public IEnumerable<ThreadDTO>? Threads { get; set; }

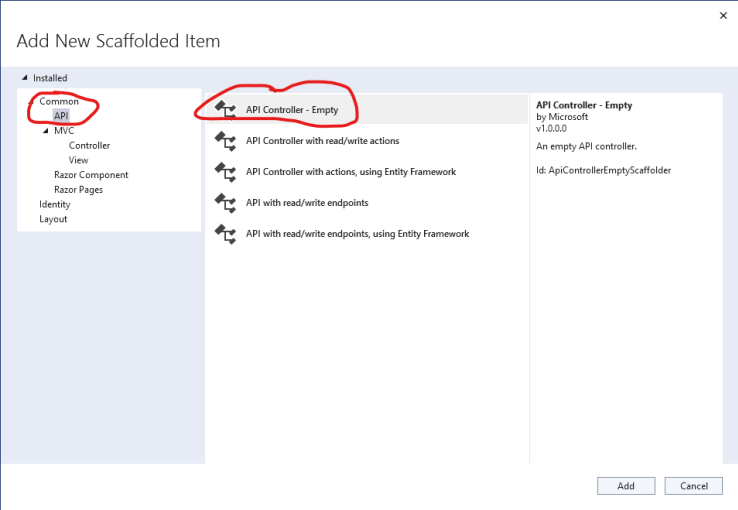
public static ForumDTO FromForum(Forum forum)
{
    return new ForumDTO
    {
        ForumId = forum.ForumId,
        Title = forum.Title,
        Threads = ThreadDTO.FromThreads(forum.Threads)
    };
}

public static IEnumerable<ForumDTO> FromFourms(IEnumerable<Forum> forums)
{
    return forums.Select(f => FromForum(f));
}
```

In the ForumDTO class, note that we have added some data annotations to the Title property, to say that it is required, and it has a minimum length requirement. These requirements are specific to the Web API controller, and do not apply to the MVC web pages that we have been working with up until now. This is a slightly unrealistic difference to introduce into our Web API controller, but it serves to demonstrate that using a DTO allows you to specify very precisely where these attributes apply.

Add another folder at root level, called ApiControllers. It's not necessary to keep your API controllers separate from your MVC controllers, but it makes the project more manageable to arrange it in this way.

Right-click on the ApiControllers folder, and choose Add, Controller. Select an API Controller – Empty:



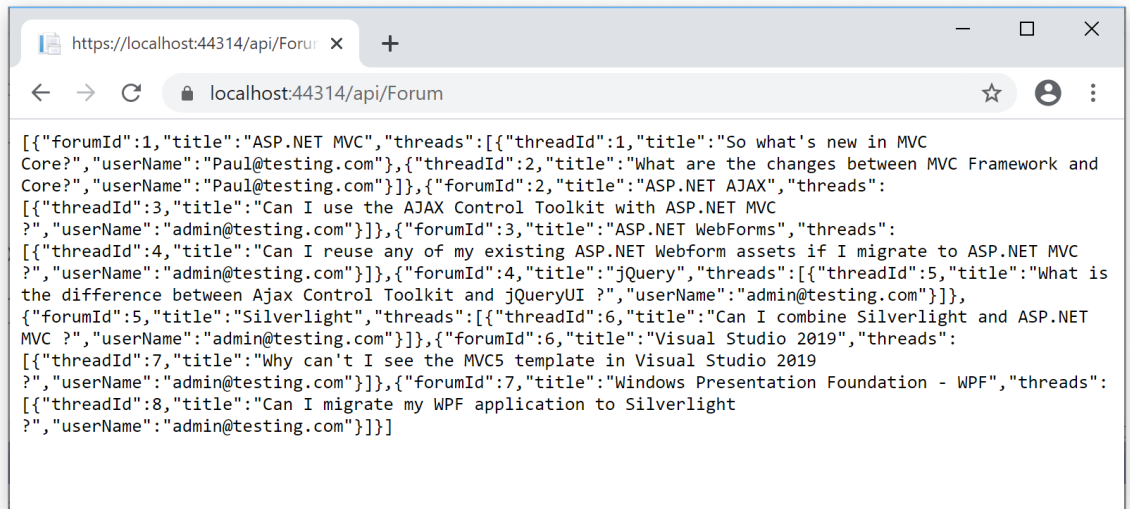
Name the controller ForumController.

	<p>Note that we already have another controller called ForumController, in the Users area. In most cases, the two controllers can coexist happily alongside each other, so long as they are in different areas and also use different routes (URLs).</p> <p>However, in some cases, the HTML helper function <code>Html.ActionLink()</code> can generate routes to the Web API controller instead of the MVC controller. In our application, this happens with the MVC controller's Delete action, because although our Web API controller doesn't have a Delete action, it's an action that's supported as part of Web API controllers in general.</p> <p>To prevent that from happening, we're going to assign a low priority to the [Route] attribute of our new controller. Add the following. Note that higher Order numbers result in lower priority, and that the default is 0:</p> <pre>[Route("api/[controller]", Order = 10)] [ApiController] public class ForumController : ControllerBase {</pre>
	<p>Add code to your new controller to inject a ForumDbContext into it:</p> <pre>private readonly ForumDbContext context;  public ForumController(ForumDbContext context) {     this.context = context; }</pre>
	<p>Add two Get() methods to the controller, one to retrieve a list of forums, and one to retrieve a single forum.</p> <p>Note the use of the [HttpGet] attribute on both methods:</p> <pre>[HttpGet] public IEnumerable&lt;ForumDTO&gt; Get() {     return ForumDTO.FromFourms(context.Forums); }  [HttpGet("{id}")] public ForumDTO Get(int id) {     var forum = context.Forums.Single(f =&gt; f.ForumId == id);     return ForumDTO.FromForum(forum); }</pre>

Run the application. Once it's running, modify the URL by appending the following onto it:

/api/Forum

You should see a JSON representation of all the forums and their threads:



```
[{"forumId":1,"title":"ASP.NET MVC","threads":[{"threadId":1,"title":"So what's new in MVC Core?","userName":"Paul@testing.com"}, {"threadId":2,"title":"What are the changes between MVC Framework and Core?","userName":"Paul@testing.com"}]}, {"forumId":2,"title":"ASP.NET AJAX","threads":[{"threadId":3,"title":"Can I use the AJAX Control Toolkit with ASP.NET MVC ?","userName":"admin@testing.com"}]}, {"forumId":3,"title":"ASP.NET WebForms","threads":[{"threadId":4,"title":"Can I reuse any of my existing ASP.NET Webform assets if I migrate to ASP.NET MVC ?","userName":"admin@testing.com"}]}, {"forumId":4,"title":"jQuery","threads":[{"threadId":5,"title":"What is the difference between Ajax Control Toolkit and jQueryUI ?","userName":"admin@testing.com"}]}, {"forumId":5,"title":"Silverlight","threads":[{"threadId":6,"title":"Can I combine Silverlight and ASP.NET MVC ?","userName":"admin@testing.com"}]}, {"forumId":6,"title":"Visual Studio 2019","threads":[{"threadId":7,"title":"Why can't I see the MVC5 template in Visual Studio 2019 ?","userName":"admin@testing.com"}]}, {"forumId":7,"title":"Windows Presentation Foundation - WPF","threads":[{"threadId":8,"title":"Can I migrate my WPF application to Silverlight ?","userName":"admin@testing.com"}]}]
```

Append onto the new URL:

/3

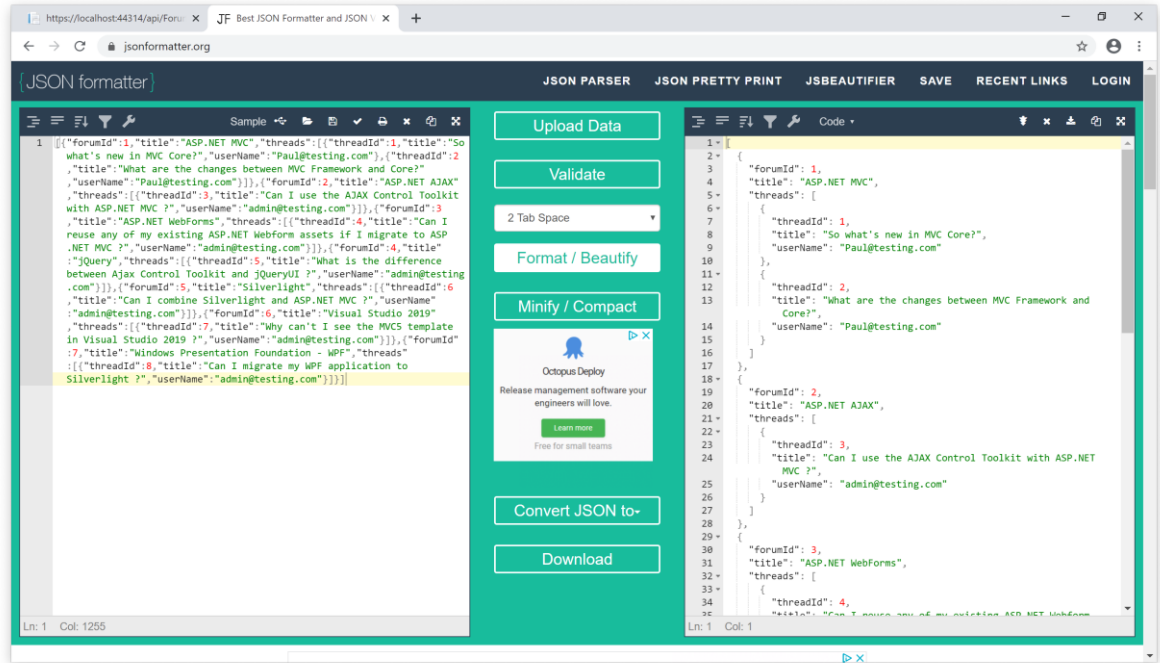
Now you should see a JSON representation of just a single forum:



```
{"forumId":3,"title":"ASP.NET WebForms","threads":[{"threadId":4,"title":"Can I reuse any of my existing ASP.NET Webform assets if I migrate to ASP.NET MVC ?","userName":"admin@testing.com"}]}
```

The data in the web browser is very hard to read! Copy it, then visit this web site:  
<https://jsonformatter.org>

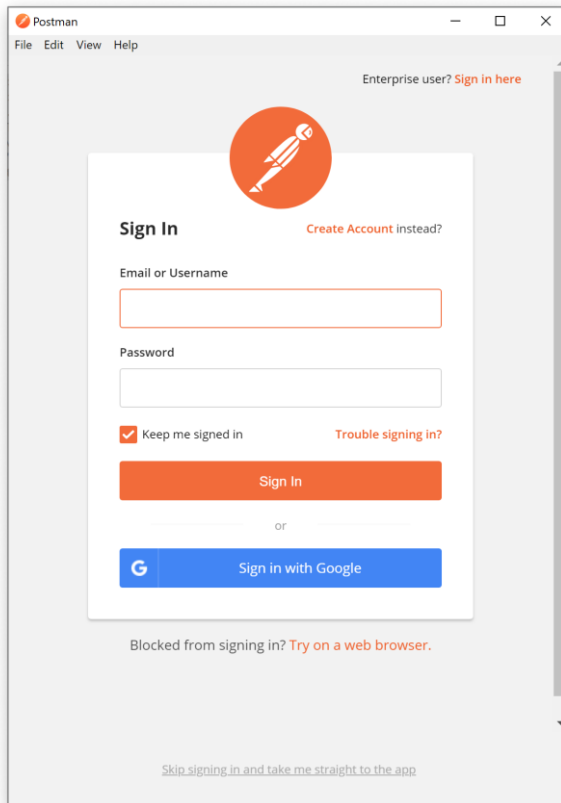
Paste the data into the left hand side of this site, and click the button marked "Format/Beautify". The right hand side of the site now contains the same data, but in a format which is much easier to read.



There is a limit to how much more we can do with a web browser when we're trying to see what's going on inside the HTTP messages.

In the Assets folder is the installer for the Postman software. Run the installer, and then run Postman.

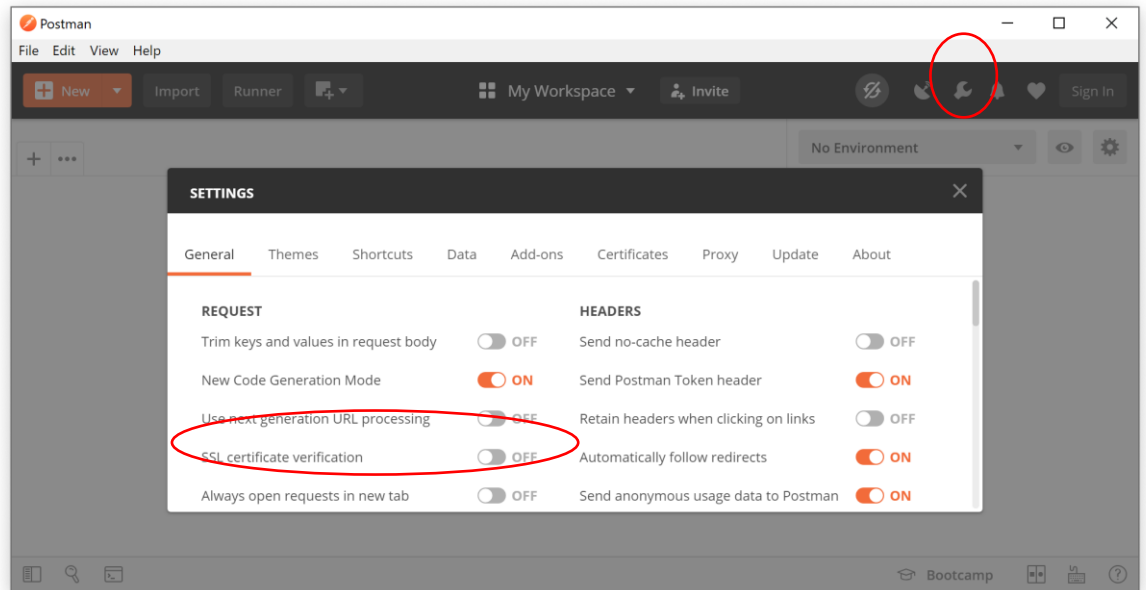
You will be asked to sign in, but you should see an option at the bottom of the screen to continue without an account.



In the top-right corner of the screen are a number of icons. Click the icon that looks like a spanner, the click Settings.

On the settings page, turn off the setting called "SSL Certificate Verification". Turning this option off is something you should not normally need to do if your

SSL certificate is valid, but since we haven't been using a valid certificate for this lab, this step is necessary.



Click the New button, and create a Request

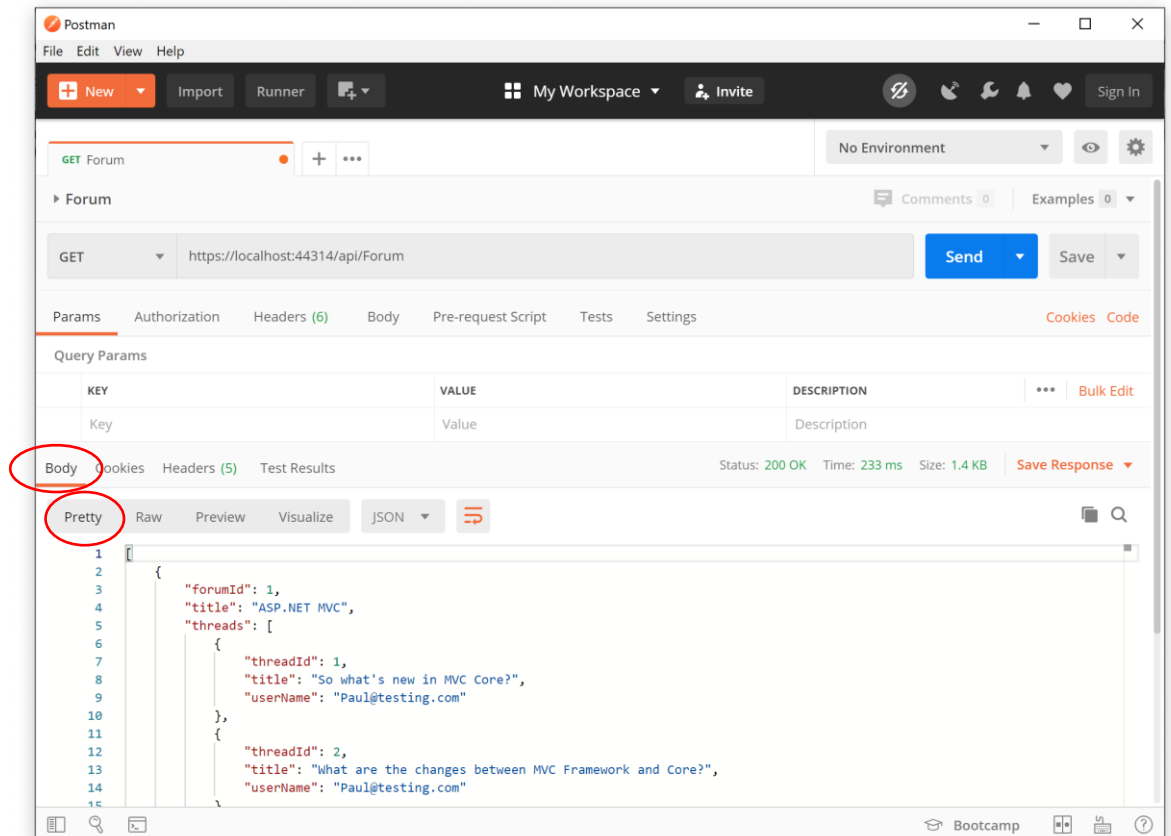
On the next screen, you need to enter a name for the request, and also a name for a collection in which the request will be saved. It doesn't matter what names you use here, but choose something sensible. In the screenshot below, the request has been called "Forum", and the collection "QAForum".

Click the button to create the collection, then click the Save button.

With the application running, copy the URL from the web browser, and paste it into the box at the top of Postman. Modify the URL, if necessary, to end: /api/Forum

Click the Send button.

The request will be sent to the web server, and in the bottom of the screen you will see the response. Ensure the “Body” section of the response is showing, and then choose the “Pretty” view:



This shows the same data as you saw in the web browser, but it's automatically beautified. You can also see, in green above the response body, the status code that was returned: 200 OK.

Modify the URL – add onto it:

/3

Send the request, and again look at the response. Again, it shows the same as the web browser but in a more detailed format.



	<p>Next, change the URL – replace the 3 with a 0.</p> <p>This time, an exception occurs in the program. If you started the program with F5, it will go into Debug mode – press F5 to continue. (If you started the program with Ctrl-F5 it won't go into Debug mode).</p> <p>Now, Postman shows that the body of the response contains the exception type and message (System.InvalidOperationException: Sequence contains no elements), and that the status code is 500 Internal Server Error.</p> <p>The reason this has happened is that there is no forum with an ID of 0. But the response that we've got is not correct. The URL ends with /api/Forum/0, and the problem is that there is no resource which corresponds to this URL.</p>
	<p>Stop the program from running (there's no need to stop Postman), and change the second Get() method as follows:</p> <pre data-bbox="335 730 1493 1070"> [HttpGet("{id}")] public ActionResult&lt;ForumDTO&gt; Get(int id) {     var forum = context.Forums.SingleOrDefault(f =&gt; f.ForumId == id);     if (forum == null)     {         return NotFound();     }     return ForumDTO.FromForum(forum); } </pre> <p>Start the program again, then re-send the request from Postman. Now, a request with an invalid ID returns a status of 404 Not Found, which makes much more sense.</p>
	<p>Stop the program again, and add a Post method to the controller. The Post method is used to create new entries in the database:</p> <pre data-bbox="335 1254 1493 1729"> [HttpPost] public ActionResult&lt;ForumDTO&gt; Post(ForumDTO dto) {     var forum = new Forum     {         Title = dto.Title     };      context.Forums.Add(forum);     context.SaveChanges();      // Create a new DTO, to reflect the ID that was assigned on save     return CreatedAtAction         ("Get", new { id = forum.ForumId }, ForumDTO.FromForum(forum)); } </pre>

Start the program again, then go into Postman:

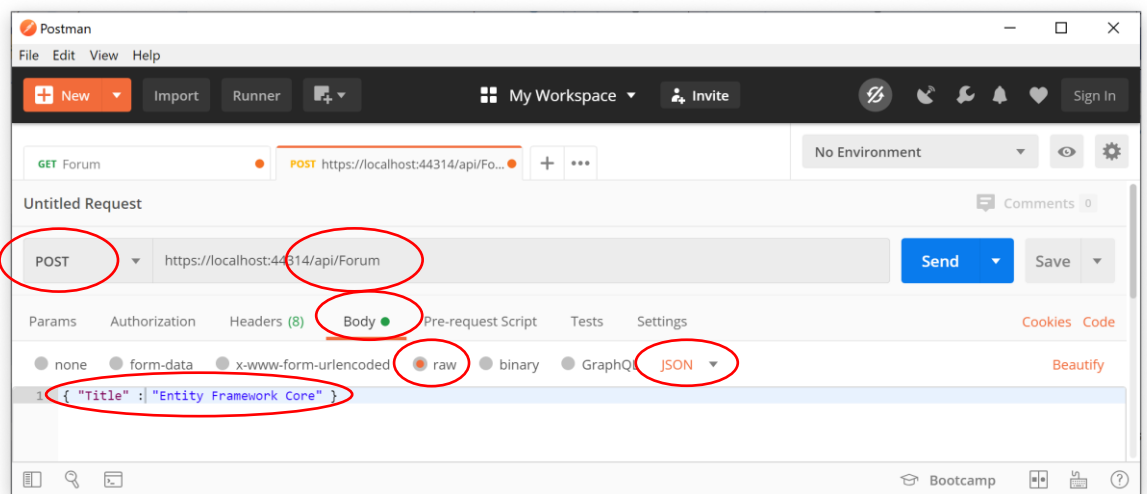
Change the HTTP method to POST, using the drop-down to the left of the URL

Change the URL to <https://localhost:xxxx/api/Forum>

In the top part of the screen, under the Request section (*not* the Response), click on Body. Then, choose the Raw radio button, and on the right hand side of the radio buttons choose the media type of JSON

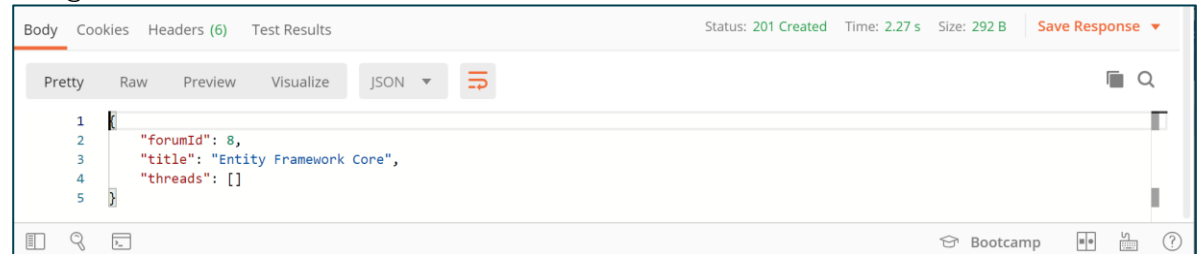
Enter the following into the Body:

```
{ "Title" : "Entity Framework Core" }
```

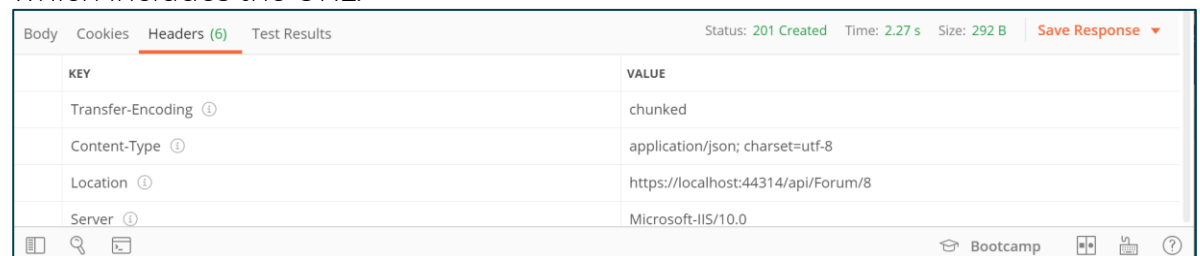


Send the Post request. The bottom part of Postman now shows the response. The response code is 201 Created – we used the `CreateAtActionResult()` method to generate a `CreateAtActionResult` object, and this object sets the response code to 201.

When returning a 201 status code, we also return the JSON representation of the new object, which you can now see in Postman. Note that it includes the newly-assigned ID.



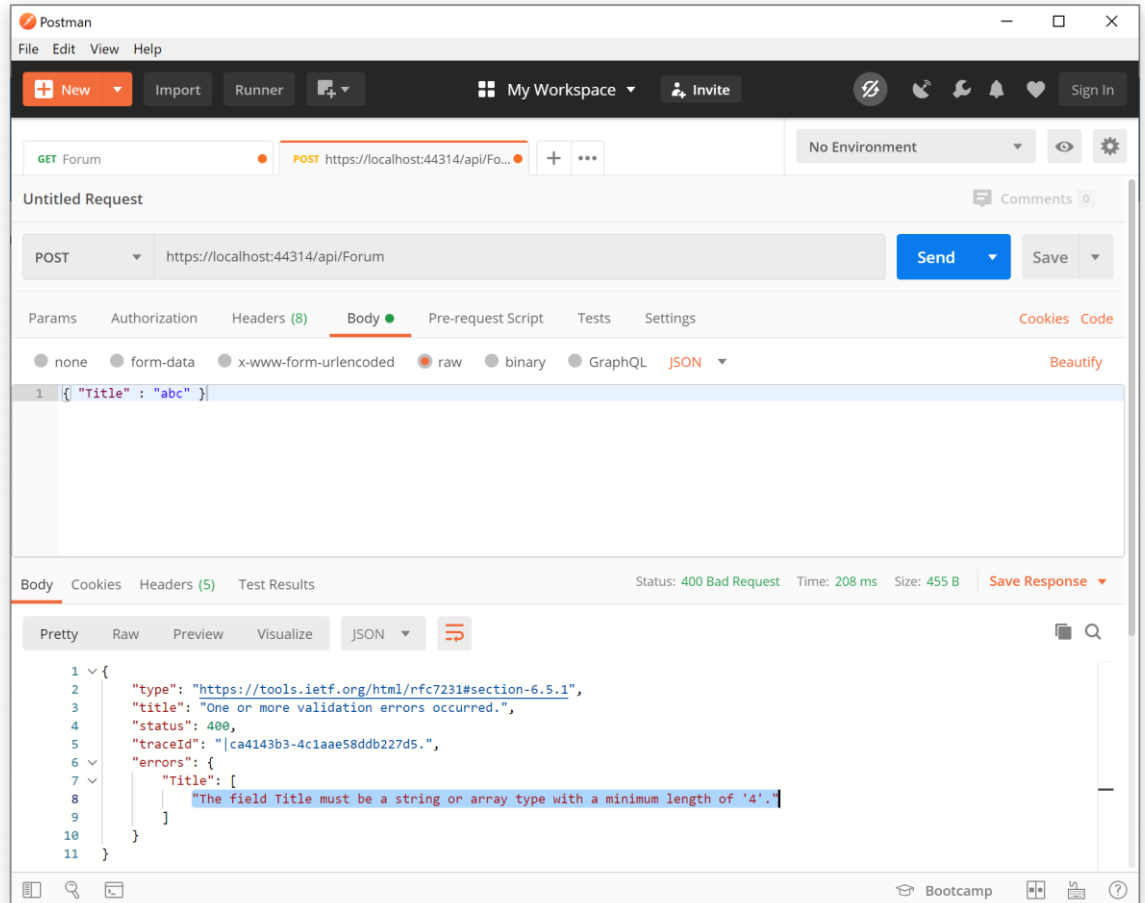
We also indicate the URL where the object can be downloaded. Click the Headers tab in the Response section of Postman, and notice the Location entry, which includes the URL.



Send a GET request to that URL to see that the new forum has been entered into the database correctly.

Change the forum title in the Request to “abc”, and change the request type back to POST, then re-submit the request.

This time, you will see a 400 Bad Request status message. The body of the response explains why this is a bad request: “The field Title must be a string or array type with a minimum length of '4'.” We set that restriction on the DTO at the start of the lab, and we didn't need to write any code to get this error because the [ApiController] attribute has taken care of that for us.



### If You Have Time – Adding More Actions

Below, you will find the code for two more actions: Put (which is used for editing or updating an item) and Delete.

Copy them into the controller, and take some time to understand how they work. Then, use Postman to test them. We will leave you to work out for yourself the steps required to do this testing!

```
[HttpPut("{id}")]
public ActionResult Put(int id, ForumDTO dto)
{
    if (id != dto.ForumId)
    {
        return BadRequest
            ("The forum id in the body must match the id in the URL");
    }

    var forum = context.Forums.SingleOrDefault(f => f.ForumId == id);
```

	<pre>         if (forum == null)         {             return NotFound();         }          forum.Title = dto.Title;         context.SaveChanges();          return NoContent();     }      [HttpDelete("{id}")]     public ActionResult&lt;ForumDTO&gt; Delete(int id)     {         var forum = context.Forums.SingleOrDefault(f =&gt; f.ForumId == id);         if (forum == null)         {             return NotFound();         }          context.Forums.Remove(forum);         context.SaveChanges();          return ForumDTO.FromForum(forum);     } </pre>	
--	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

### If You Have Even More Time – Investigating Web API Clients

	<p>We create Web API controllers in order for other programs to be able to connect to them.</p> <p>In this section, we will look at how to create clients, in both C# and in JavaScript/JQuery.</p>	
	<p>Run the program.</p> <p>While the program is running, locate the file WebApiClientCSharp.sln, which can be found in the Assets/WebApiClientCSharp folder. Open it in a new instance of Visual Studio. (Don't close down the instance of Visual Studio in which QAForum is loaded – this should currently be running your application, including your Web API controller.)</p> <p>Have a look through the code, which shows how to access a Web API controller from a C# program – a simple console application, in this case.</p> <p>Run the client program, and check it works as expected. Then, stop both programs from running.</p>	
	<p>Next, we're going to look at a JavaScript example which uses JQuery.</p> <p>In the folder Assets/WebApiClientJavascript is a file called client_demo.html. Drag this file onto the wwwroot folder in the solution explorer. Open the file, and study the Javascript code at the bottom of the file.</p> <p>Run the QAForum application, then, when it starts, change the web browser's URL to end with /client_demo.html. Try out both the buttons on the page, and check they work as expected.</p>	

	<p>Take a moment to understand exactly what's happening in the Javascript demo.</p> <p>When you click a button, it does not reload the web page – if we wanted that behaviour, we would have simply used MVC.</p> <p>It does not fetch HTML from the server at all – it's possible to create an MVC action that returns a partial view, and then to write some Javascript that adds that HTML to the existing web page, but that is not what we've done here.</p> <p>Instead, the web page fetches JSON data from the server. The server has no knowledge of what's going to be done with that data. It's purely down to the web page to decide what action to take. What this particular web page does is create some HTML based on that JSON data, but there's nothing to stop another web site from using the same data in a different way. The code that generates this HTML is as follows:</p> <pre>var output = \$('&lt;div&gt;'); output.append(\$('&lt;h3&gt;').text('Here are the forums:')); \$.each(data, function (i, forum) {     output.append(\$('&lt;p&gt;').text(forum.title)); }); \$('#output').html(output);</pre> <p>This is an extremely common way to build modern websites, although the client code is often written using a Javascript framework such as React or Angular, rather than plain Javascript/JQuery.</p>
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

You have now added a Web API controller to your project. You have understood how the most common HTTP verbs can be implemented in Web API, and how to test your controller using Postman.

You have also looked at two different techniques for consuming Web API data in client applications.