MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
JUL 27, 2020 3:00 AM PDT

# How to pass parameters to action methods in ASP.NET Core MVC

Explore the various ways in which you can pass parameters to action methods in ASP.NET Core MVC

ASP.NET Core is a cross-platform, open source, lean, fast, and modular framework for building high-performance web applications. There are a number of ways in which you can pass parameters to action methods in ASP.NET Core MVC. You can pass them via a URL, a query string, a request header, a request body, or even a form. This article talks about all of these ways, and illustrates them with code examples.

To work with the code examples provided in this article, you should have Visual Studio 2019 installed in your system. If you don't already have a copy, you can download Visual Studio 2019 here.

[ Also on InfoWorld: 7 best practices for remote agile teams ]

## Create an ASP.NET Core MVC project in Visual Studio 2019

First off, let's create an ASP.NET Core project in Visual Studio 2019. Assuming Visual Studio 2019 is installed in your system, follow the steps outlined below to create a new ASP.NET Core project in Visual Studio.

1. Launch the Visual Studio IDE.

2. Click on “Create new project.”
3. In the “Create new project” window, select “ASP.NET Core Web Application” from the list of templates displayed.
4. Click Next.
5. In the “Configure your new project” window, specify the name and location for the new project.
6. Optionally check the “Place solution and project in the same directory” check box, depending on your preferences.
7. Click Create.
8. In the “Create a New ASP.NET Core Web Application” window shown next, select .NET Core as the runtime and ASP.NET Core 3.1 (or later) from the drop-down list at the top.
9. Select “Web Application (Model-View-Controller)” as the project template to create a new ASP.NET Core MVC application.
10. Ensure that the check boxes “Enable Docker Support” and “Configure for HTTPS” are unchecked as we won’t be using those features here.
11. Ensure that Authentication is set to “No Authentication” as we won’t be using authentication either.
12. Click Create.

Following these steps should create a new ASP.NET Core MVC project in Visual Studio 2019. We’ll use this project in the sections below to illustrate the various methods of passing parameters to action methods in ASP.NET Core 3.1.

## Create an AuthorRepository class in ASP.NET Core MVC

In this example we’ll be using a repository class — the action methods in the controller will interact with the methods of the repository class for CRUD operations. We’ll first create a model class named Author with minimal properties for the sake of simplicity as shown in the code snippet given below.

```
public class Author
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

The AuthorRepository class contains methods for retrieving instances of the Author class from a generic list as well as for adding new instances of the Author class to the generic list. The GetAuthors method returns a page of data, the page number being passed to it as an argument.

```
public class AuthorRepository
{
    List<Author> authors = new List<Author>()
    {
        new Author
        {
            Id = 1,
            FirstName = "Joydip",
            LastName = "Kanjilal"
        },
        new Author
        {
            Id = 2,
            FirstName = "Steve",
            LastName = "Smith"
        }
    };
    public Author GetAuthor(int id)
    {
        return authors.FirstOrDefault(a => a.Id == id);
    }
    public List<Author> GetAuthors(int pageNumber = 1)
    {
        int pageSize = 10;
        int skip = pageSize * (pageNumber - 1);
        if (authors.Count < pageSize)
            pageSize = authors.Count;
        return authors
          .Skip(skip)
          .Take(pageSize).ToList();
    }
    public bool Save(Author author)
    {
        var result = authors.Where(a => a.Id == author.Id);
        if (result != null)
        {
            if (result.Count() == 0)
            {
                authors.Add(author);
                return true;
            }
        }
        return false;
    }
}
```

# Pass parameters via the URL in ASP.NET Core MVC

One of the simplest and easiest ways to pass parameters to an action method is passing it via the URL. The following code snippet illustrates how you can pass parameters in the URL.



```
[HttpGet]
[Route("Default/GetAuthor/{authorId:int}")]
public IActionResult GetAuthor(int authorId)
{
   var data = authorRepository.GetAuthor(authorId);
   return View(data);
}
```

The URL to the endpoint is:

```
GET: http://localhost:8061/Default/GetAuthor/1
```

# Pass parameters via query string in ASP.NET Core MVC

Passing parameters in the query string is another option. It doesn't require changing the routing information and hence is backwards compatible. Consider the following code snippet that illustrates how you can pass parameters via query strings in an action method.

```
[HttpGet]
[Route("Default/GetAuthors/{pageNumber:int}")]
public IActionResult GetAuthors([FromQuery
(Name = "pageNumber")] int pageNumber = 1)
{
   var data = authorRepository.GetAuthors(pageNumber);
   return Ok(data);
}
```

Here is the URL to access this endpoint:

```
GET: http://localhost:8061/Default/GetAuthors?pageNumber=1
```

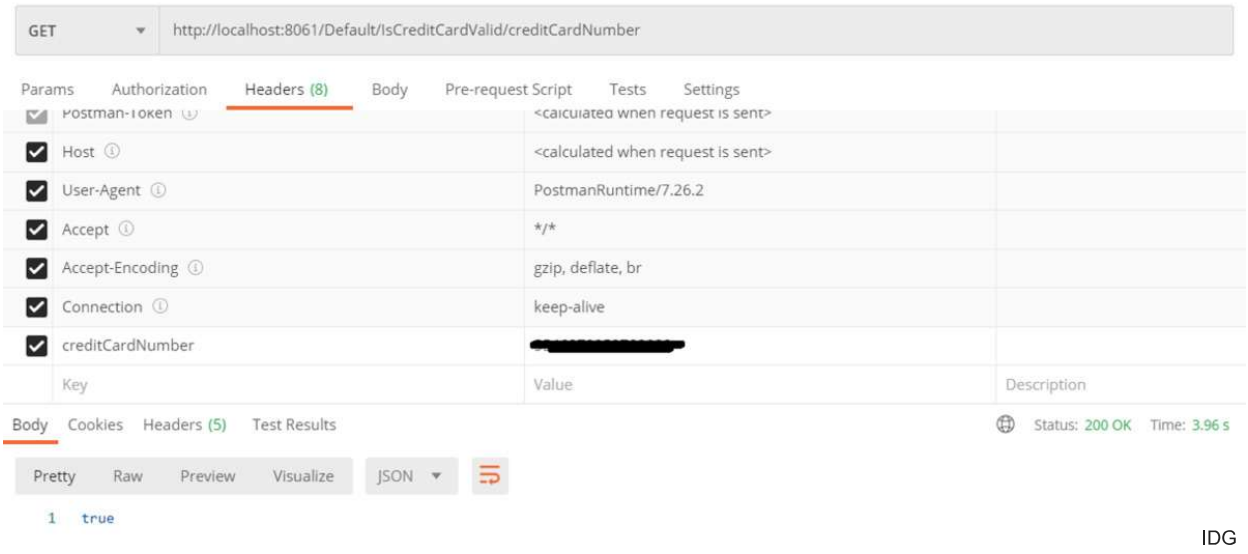
The GetAuthors method accepts the page number as an argument sent to it via query string. Note that pageNumber is an optional parameter — if no parameter is passed to this method, then the page number would be interpreted as 1. The method returns the author records for the specified page. In our example, if there are 100 author records in the data store and the page number is 3, this method would return records 31 to 40. (Note that the number of authors per page is hard coded; it is specified as 10 in the AuthorRepository class.)

## Pass parameters via request header in ASP.NET Core MVC

The request header is yet another option for passing parameters to your action methods. A common use case for this is passing credentials or any other secret data over the wire. The following code snippet illustrates an action method that accepts a credit card number as a parameter and returns true if the credit card number is valid.

```
[HttpGet]
[Route("Default/IsCreditCardValid/{creditCardNumber}")]
public IActionResult IsCreditCardValid([FromHeader] string creditCardNumber)
{
   string regexExpression =
   "^(?:(?<visa>4[0-9]{12}(?:[0-9]{3})?)|" +
   "(?<mastercard>5[1-5][0-9]{14})|" +
   "(?<amex>3[47][0-9]{13})|)$";
   Regex regex = new Regex(regexExpression);
   var match = regex.Match(creditCardNumber);
   return Ok(match.Success);
}
```

For the sake of simplicity, the IsCreditCardValid action method validates Visa, MasterCard, and Amex credit cards only. You can extend the IsCreditCardValid method to validate other card types. Since the credit card number should be passed securely, using the request header is a good choice here. Figure 1 shows how you can specify your credit card number as a parameter via request header.




IDG

*Figure 1.*

## Pass parameters via request body in ASP.NET Core MVC

You will often need to pass parameters via the request body when you're performing insert or update operations. The following code snippet illustrates how you can pass an instance of the Author class via the body of the request.

```
[HttpPost]
[Route("Default/Insert")]
public IActionResult Insert([FromBody] Author author)
{
   return Ok(authorRepository.Save(author));
}
```

Figure 2 shows how you can specify the data to be inserted in the request body.

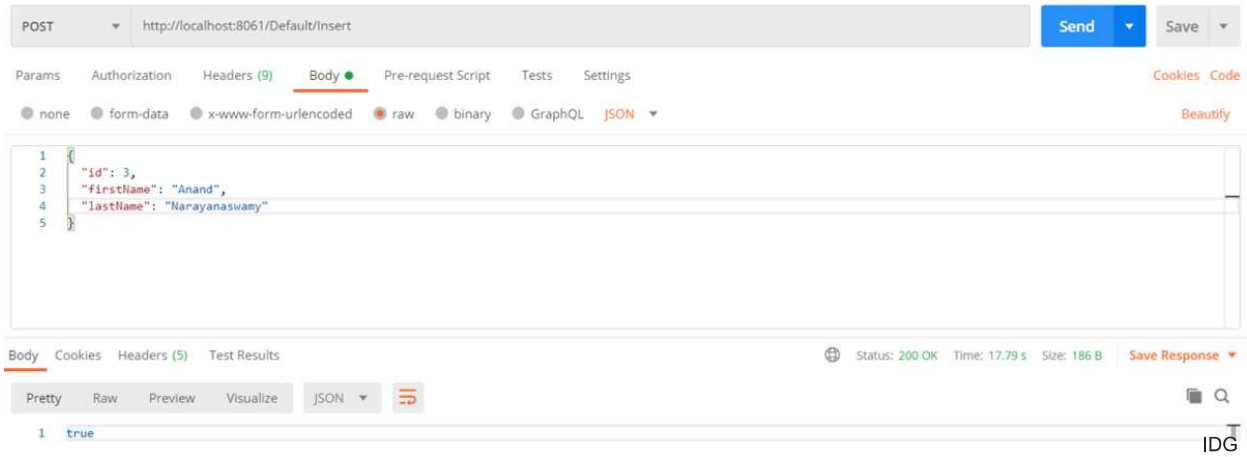



*Figure 2.*

# Complete source code of our DefaultController class

The complete code of the DefaultController class is provided below for your reference.

```
public class DefaultController : Controller
   {
       private readonly AuthorRepository authorRepository =
       new AuthorRepository();
       [HttpGet]
       [Route("Default/GetAuthor/{authorId:int}")]
       public IActionResult GetAuthor(int authorId)
       {
           var data = authorRepository.GetAuthor(authorId);
           return Ok(data);
       }
       [HttpGet]
       [Route("Default/GetAuthors/{pageNumber:int}")]
       public IActionResult GetAuthors([FromQuery
       (Name = "pageNumber")] int pageNumber = 1)
       {
           var data = authorRepository.GetAuthors(pageNumber);
           return Ok(data);
       }
       [HttpGet]
       [Route("Default/IsCreditCardValid/{creditCardNumber}")]
       public IActionResult IsCreditCardValid
       ([FromHeader] string creditCardNumber)
       {
           string regexExpression =
           "^(?:(?<visa>4[0-9]{12}(?:[0-9]{3})?)|" +
           "(?<mastercard>5[1-5][0-9]{14})|" +
           "(?<amex>3[47][0-9]{13})|)$";
           Regex regex = new Regex(regexExpression);
           var match = regex.Match(creditCardNumber);
           return Ok(match.Success);
       }
       [HttpPost]
       [Route("Default/Insert")]
       public IActionResult Insert([FromBody] Author author)
       {
           return Ok(authorRepository.Save(author));
       }
   }
```

Finally, you can also pass parameters via a form. A form is often used when you would like to upload a file. You would need to take advantage of the IFormFile interface in this case.

## How to do more in ASP.NET Core:

- How to use API Analyzers in ASP.NET Core
- How to use route data tokens in ASP.NET Core
- How to use API versioning in ASP.NET Core
- How to use Data Transfer Objects in ASP.NET Core 3.1
- How to handle 404 errors in ASP.NET Core MVC
- How to use dependency injection in action filters in ASP.NET Core 3.1
- How to use the options pattern in ASP.NET Core
- How to use endpoint routing in ASP.NET Core 3.0 MVC
- How to export data to Excel in ASP.NET Core 3.0
- How to use LoggerMessage in ASP.NET Core 3.0
- How to send emails in ASP.NET Core
- How to log data to SQL Server in ASP.NET Core
- How to schedule jobs using Quartz.NET in ASP.NET Core
- How to return data from ASP.NET Core Web API
- How to format response data in ASP.NET Core
- How to consume an ASP.NET Core Web API using RestSharp
- How to perform async operations using Dapper
- How to use feature flags in ASP.NET Core
- How to use the FromServices attribute in ASP.NET Core
- How to work with cookies in ASP.NET Core
- How to work with static files in ASP.NET Core
- How to use URL Rewriting Middleware in ASP.NET Core
- How to implement rate limiting in ASP.NET Core
- How to use Azure Application Insights in ASP.NET Core
- Using advanced NLog features in ASP.NET Core

- How to handle errors in ASP.NET Web API
- How to implement global exception handling in ASP.NET Core MVC
- How to handle null values in ASP.NET Core MVC
- Advanced versioning in ASP.NET Core Web API
- How to work with worker services in ASP.NET Core
- How to use the Data Protection API in ASP.NET Core
- How to use conditional middleware in ASP.NET Core
- How to work with session state in ASP.NET Core
- How to write efficient controllers in ASP.NET Core

---

*Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.*

*Follow*

Tim O'Reilly: the golden age of the programmer is over

How to pass multiple parameters to Web API controller methods

The shifting market for PostgreSQL

Amazon Braket: Get started with quantum computing

Why MongoDB is ‘fundamentally better’ for developers

7 best practices for remote development teams

3 enterprise AI success stories

What is a computational storage drive? Much-

Red Hat OpenShift ramps up security and

When to use Task.WaitAll vs. Task.WhenAll in .NET

The COVID pandemic's lasting impact on cloud usage