Supercharging ASP.NET Core API with OData

Summary

In this article, I'm going to show you how you can supercharge your existing ASP.NET Core APIs with OData to provide better experience for your API consumers with only 4 lines of code.

For the purpose of this tutorial please clone our demo project WashingtonSchools so you can follow up and try the different features we are going to talk about in this article.

You can clone the demo project from here:

https://github.com/hassanhabib/ODataDemo

What is OData?

Let's talk about OData …

OData is an open source, open protocol technology that provides the ability to API developers to develop Queryable APIs, one of the most common things API developers need is pagination for instance.

With OData you can enable pagination, ordering of data, reshaping and restructuring of the data and much more with only 4 lines of code.

One of the most common scenarios today is when developers call an endpoint and pull out data that they are going to filter, reshape and order later on the client side. This seems a bit wasteful. especially if the data is large enough that could cause latency and require further optimizations for a better user experience.

Getting Started

To get this started, this tutorial assumes you already have an API that provides some list of objects

to your end users, for the purpose of this tutorial, we are going to use a sample WashingtonSchools API project that we built to demonstrate this feature.

Our API endpoint api/students returns all the students available in our database.

To enable OData on that endpoint, we are going to need to install a nuget package for all OData binaries that'll enable us to turn our endpoint into a Queryable endpoint.

The nuget package we are targeting here is:

https://www.nuget.org/packages/Microsoft.AspNetCore.OData

Once that's installed, let's go ahead and add OData services to our API.

To do that, go to: Startup.cs file and add in this line of code in your ConfigureServices Function:

```
services.AddOData();
```

Now we need to enable the dependency injection support for ALL HTTP routes.

To do that, in the same file Startup.cs let's go to the Configure function and add these two lines of code:

```
app.UseMvc(routeBuilder => {
    routeBuilder.EnableDependencyInjection();
    routeBuilder.Expand().Select().OrderBy().Filter();
});
```

The first line of code enables the dependency injection to inject OData services into your existing API controller.

The second line of code determines which OData functionality you would like to enable your API consumers to use, we will talk about Expand, Select and OrderBy in details shortly after we complete the setup of OData on your project.

The last line of code we would want to add is an annotation on top of your API controller method, in our example here, we have a StudentsController class that has a GetStudents method to server all the available students in our database.

Now all we need to do is to add this annotation on top of your GetStudents endpoint as follows:

1

```
[EnableQuery]
```

So your controller method code should look like this:

```
1
[HttpGet]
2

3
[EnableQuery]
4

5
public IEnumerable<Student> GetStudents(){
6

7
        return this.context.Students;
8

9
}
```

Now that the setup is done, let’s test OData select, OrderBy and expand functionality.

Select

On your favorite API testing software or simply in the browser since this is a GET endpoint, let’s try to select only the properties that we care about from the students objects.

If the students object consists of an ID, Name and we care only about the Name, we could call our endpoint like this:

```
api/students?$select=Name
```

The results will be a json list of students that only has the Name property displayed. try different combinations such as:

```
api/students?$select=ID,Name
```

The select functionality enables you to control and reshape the data to fit just your need, it makes a big difference when your objects hold large amounts of data, like image data for instance that you don't really need for a specific API call, or rich text that is contained within the same object, using select could be a great optimization technique to expedite API calls and response time.

OrderBy

The next functionality here is the OrderBy, which allows you to order your students based on their names alphabetically or their scores, try to hit your endpoint as follows:

```
api/students?$orderby=Name
```

You can also do:

```
api/students?$orderby=Score desc
```

Expand

One other functionality we want to test here is the expand. The expand functionality comes with a great advantage since it allows navigation across multiple entities.

In our example here, Students and Schools are in a many-to-one relationship, we can pull the school every student belongs to by making an expand call like this:

```
api/students?$expand=School
```

Now we get to see both students and their schools nested within the same list of objects.

Filter

The last functionality here is the Filter. Filtering enables you to query for a specific data with a specific value, for instance, If I'm looking for a student with the name Todd, all I have to do is to make an API call as follows:

```
1
api/students?$filter=Name eq ' Todd'
```

The call will return all students that have the name Todd, you can be more specific with certain properties such as scores. For instance: if I want all students with scores greater than 100, I could make the following API call:

```
1
api/students?filter=Score gt 100
```

There are more features in OData that enables even more powerful API functionality that I urge you to explore such as MaxTop for pagination.

Final Notes

Here are few things to understand about OData:

1. OData has no dependency whatsoever on the entity framework, it can come in handy with EF but it doesn't depend on it, here's a project that runs an OData API without EF: https://github.com/hassanhabib/ODataWithoutEF
2. OData can give you more optimization with EF if you use IQueryable as a data return type that IEnumerable since IQueryable only evaluates after the query is built and ready to be executed.
3. OData is much older than any other similar technology out there, it was released officially in 2007 and it has been around since then being used in large scale applications such as Microsoft Graph which feeds most of Microsoft Office products and XBOX in addition to Microsoft Windows.

The Future of OData

OData team continues to improve the feature and make it even easier and simpler to use and consume on existing and new APIs with both ASP.NET Web API (classic) or ASP.NET Core and we are working on adding even more powerful features in the near future.

Revision #8

Created Wed, Apr 21, 2021 5:28 AM by Nelson

Updated Wed, Apr 21, 2021 6:15 PM by Eric DISDERO