Using MongoDB with Web API and ASP.NET Core

Posted by: Mahesh Sabnis (../../Author.aspx?AuthorName=Mahesh Sabnis) , on 9/29/2016, in **Category** ASP.NET Core (../../BrowseArticles.aspx?CatID=88)

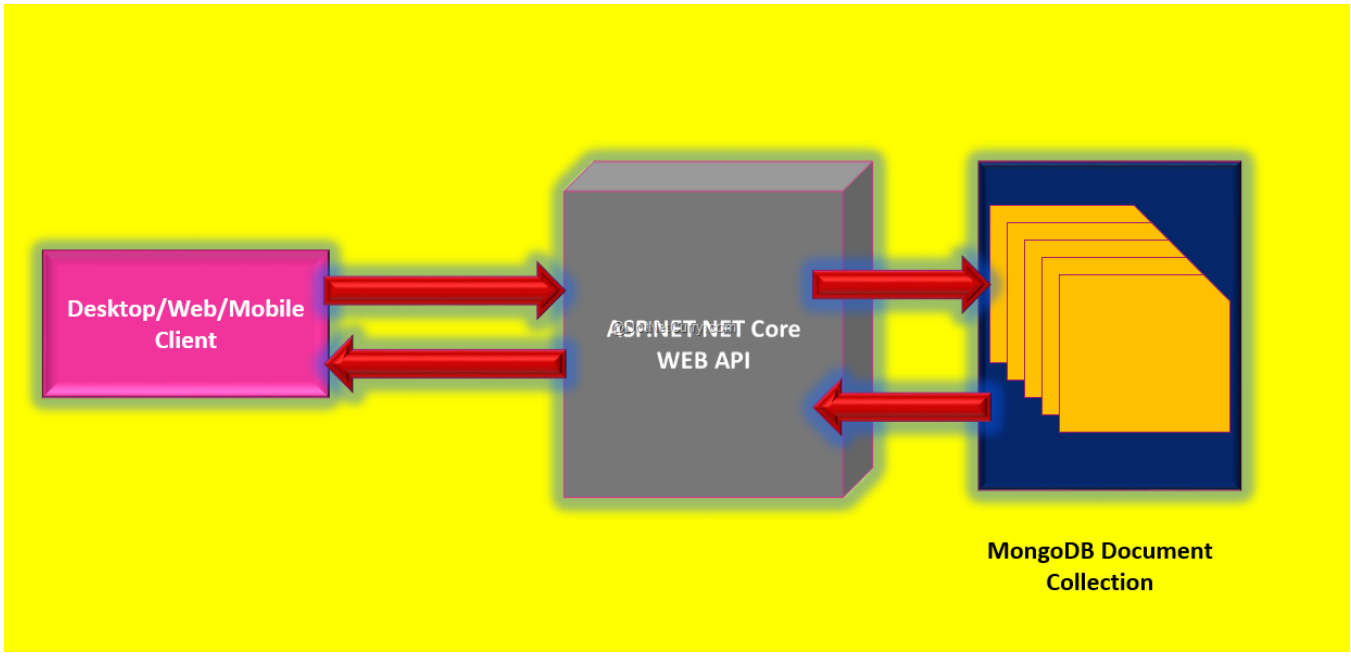
Abstract: Using MongoDB with ASP.NET Web API and ASP.NET Core to perform CRUD operations

---

This article was originally published on 3/27/2016 and updated on 9/29/2016.

MongoDB is a NoSQL document-oriented database that allows you to define JSON based documents which are schema independent. The schema can be mapped with Tables in a Relational Database. A schema in MongoDB is called as collection, and a record in this schema is called as document

In open source modern web applications, the use of a NoSQL database is gaining popularity due to its non-relational behavior. In this demo, we will create a Web API using ASP.NET Core which will perform CRUD Operations on a collection. The following diagram explains the implementation of the application.



The advantage of a Web API is that it can be used as HTTP services and can be subscribed by any client application ranging from Desktop to Mobiles. Using Web API as a medium, these client apps can easily interact with a NoSQL database like MongoDB.

Note: A newer version of ASP.NET Core (v2) is now available. Read about it here www.dotnetcurry.com/aspnet/1402/aspnet-core-2-new-features (https://www.dotnetcurry.com/aspnet/1402/aspnet-core-2-new-features)

Web API and MongoDB - The Implementation

We need MongoDB for creating collections and storing documents in it. MongoDB can be downloaded from this link (https://www.mongodb.org/downloads?_ga=1.198684911.1330621169.1458992732#production). Install the database. Once the installation is over, the system drive will create a MongoDB folder in the following path

```
C:\Program Files\MongoDB
```

We also need to create a data folder where the data will be stored. On my machine I have created E:\MongoDbData\data

Open the Command prompt and navigate to the following folder

```
C:\Program Files\MongoDB\Server\3.2\bin
```

and run the command as shown in the following image

```
C:\Program Files\MongoDB\Server\3.2\bin>mongod --dbpath e:\MongodbData\data
```

This will connect to MongoDB on port 27017.

Open another instance of the command prompt and navigate to the **bin** folder and run the following command

```
C:\Program Files\MongoDB\Server\3.2\bin>mongo
```


This will connect to the default **test** database. Run the following command on **>** (command prompt)

```
> use EmployeeDB
```

This will create a database of name EmployeeDB if it does not exist already, else it will be opened for transactions if the database already exists. In this database we can create transaction using the following command

```
db.createCollection('Products')
```

The Schema for the Products collection can be defined using following command from the command prompt

```
db.Products.insert({'ProductId':1,'ProductName':'Desktop All in
One','Price':43000,'Category':'Electronics'})
```

Run the following command

```
>db.Products.find({})
```

The following result will be displayed

```
{ "_id" : ObjectId("56f6c1b5762db5ae2ecc4d95"), "ProductId" : 1, "ProductName" : "Desktop All in One", "Price" :
"Category" : "Electronics" }
```

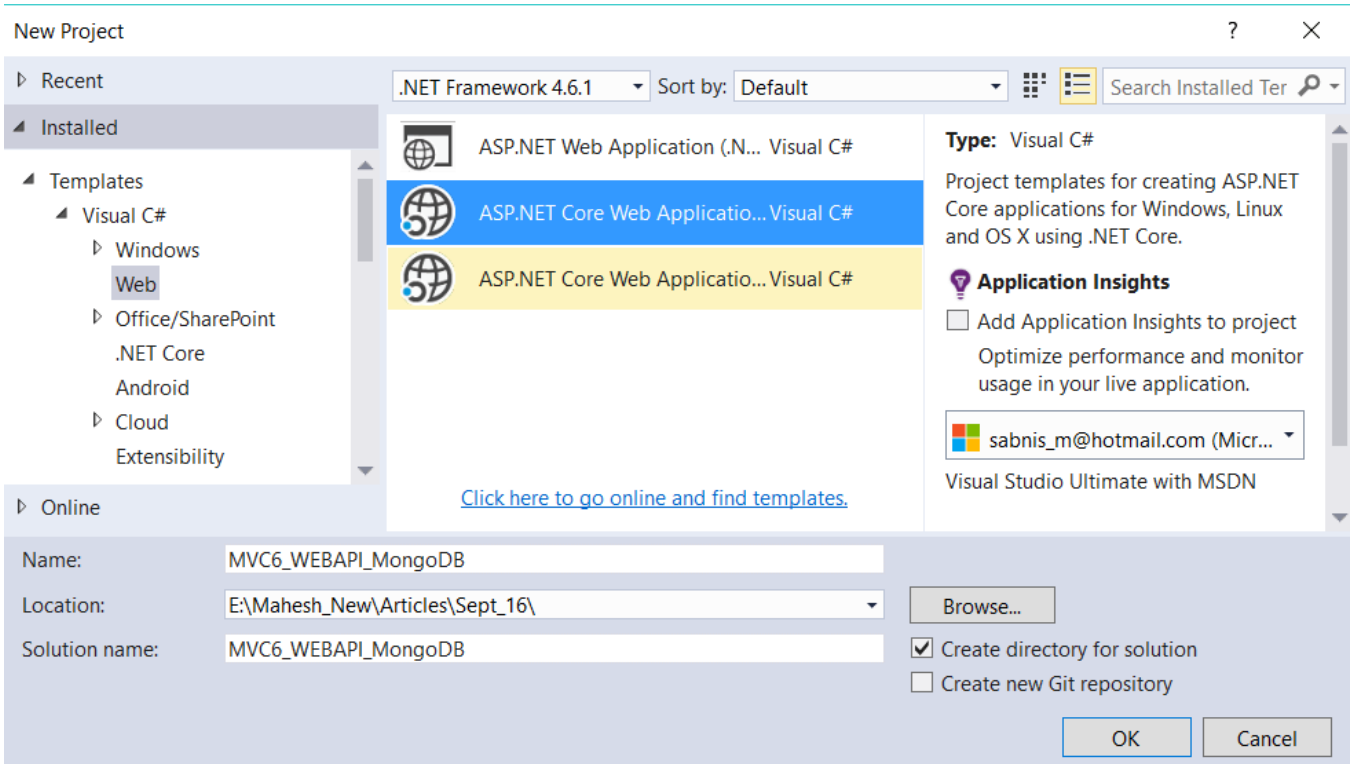

The schema will add **_id** property. This property will be an **ObjectId** which will be generated automatically.

Now since the database and collection is ready, we can create a Web API application. Please visit this link (https://www.dotnetcurry.com/aspnet/1219/create-aspnet-webapi-using-mvc-6) to read about creating Web API using MVC 6.

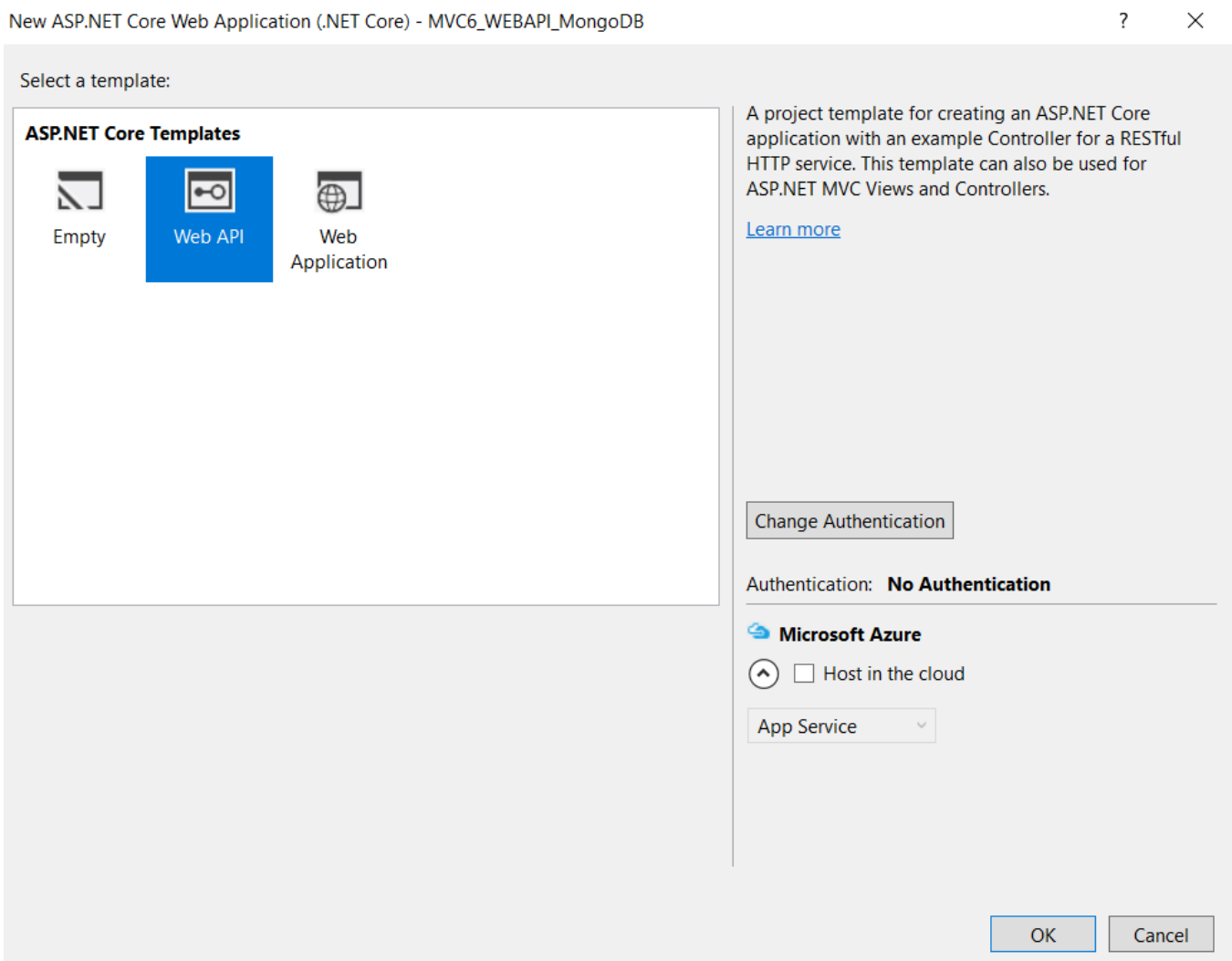
Creating the MongoDB Application

We will be using Visual Studio 2015 for creating this application. We need to install ASP.NET Core which can be downloaded from this link (https://get.asp.net/).

Step 1: Open Visual studio and create a new ASP.NET Web Application as shown in the following image



Name this application as MVC6_WEBAPI_MongoDB. Click on the **OK** button which will open the following window which shows ASP.NET Templates. Select Web API as shown in the following image



This will create a Web API project.

Step 2: Open the project.json file and in the **dependencies** section, add the following package dependency:

```
"mongocsharpdriver": "2.3.0"
```

Save the project and the Mongo CSharp driver will be installed for the project.

Step 3: In the project add the **Models** folder, in this add a new class file of name Product.cs. Add the following code in this file

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace MVC6_WEBAPI_MongoDB.Models
{
    public class Product
    {
        public ObjectId Id { get; set; }
        [BsonElement("ProductId")]
        public int ProductId { get; set; }
        [BsonElement("ProductName")]
        public string ProductName { get; set; }
        [BsonElement("Price")]
        public int Price { get; set; }
        [BsonElement("Category")]
        public string Category { get; set; }
    }
}
```

The class contains **Id** property of the type **ObjectId.** This property is mandatory so that the CLR object can be mapped with Collection in MongoDB. The class contains properties having the **BsonElement** attribute applied on it. This represent the mapped property with the MongoDB collection.

Step 3: Add the DataAccess.cs class file in the Models folder with the following code in it

```
using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Driver.Builders;
using System.Collections.Generic;

namespace MVC6_WEBAPI_MongoDB.Models
{
    public class DataAccess
    {
        MongoClient _client;
        MongoServer _server;
        MongoDatabase _db;

        public DataAccess()
        {
            _client = new MongoClient("mongodb://localhost:27017");
            _server = _client.GetServer();
            _db = _server.GetDatabase("EmployeeDB");
        }

        public IEnumerable<Product> GetProducts()
        {
            return _db.GetCollection<Product>("Products").FindAll();
        }


        public Product GetProduct(ObjectId id)
        {
            var res = Query<Product>.EQ(p=>p.Id,id);
            return _db.GetCollection<Product>("Products").FindOne(res);
        }

        public Product Create(Product p)
        {
            _db.GetCollection<Product>("Products").Save(p);
            return p;
        }

        public void Update(ObjectId id,Product p)
        {
            p.Id = id;
            var res = Query<Product>.EQ(pd => pd.Id,id);
            var operation = Update<Product>.Replace(p);
            _db.GetCollection<Product>("Products").Update(res,operation);
        }
        public void Remove(ObjectId id)
        {
            var res = Query<Product>.EQ(e => e.Id, id);
            var operation = _db.GetCollection<Product>("Products").Remove(res);
        }
    }
}
```

The above code uses the following classes:

MongoServer - This represents an instance of the MongoDB Server.

MongoClient - This class is used to read the server instance for performing operations on the database. The constructor of this class is passed with the MongoDB Connection string as shown in the following box

```
"mongodb://localhost:27017"
```

MongoDatabase - This represents Mongo Database for performing operations. This class provides following methods

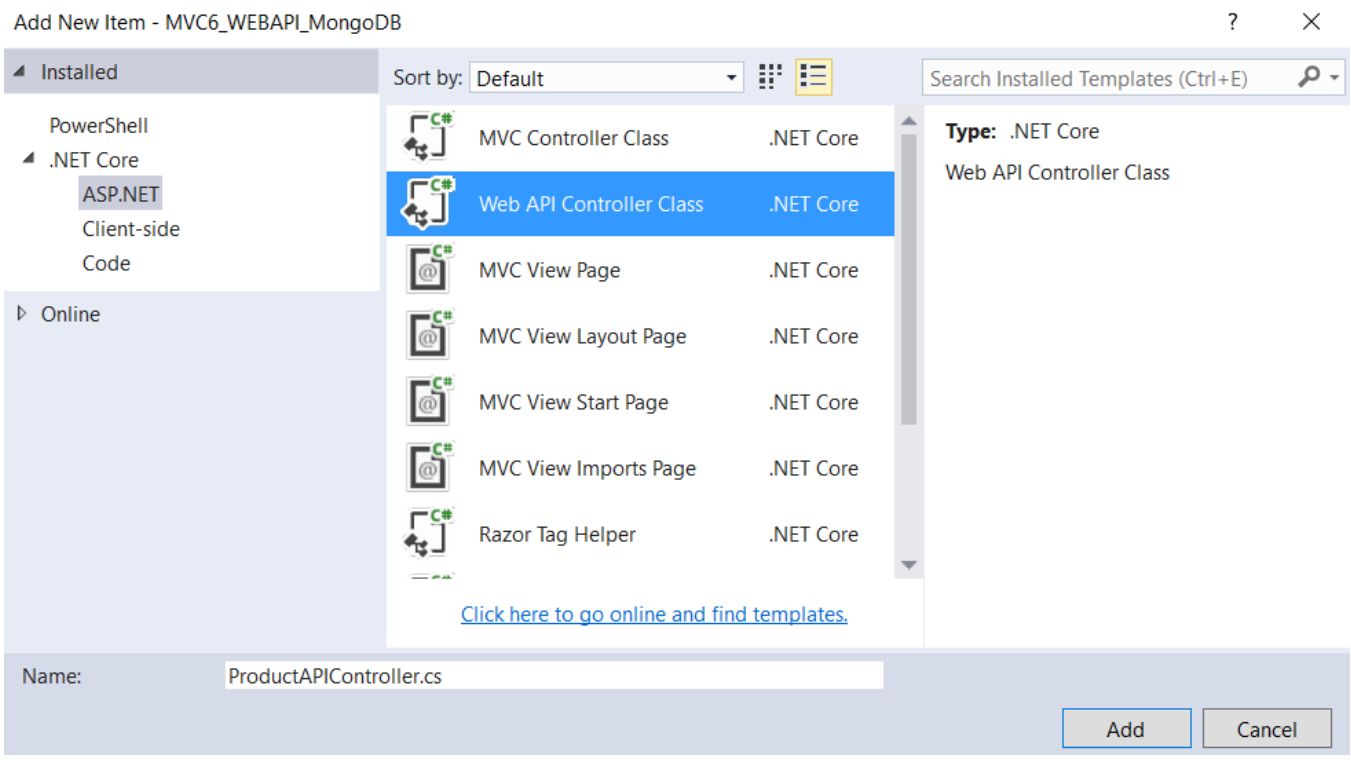
- GetCollection<T>(***collection***)
- · T is the CLR object to be ***collection.***
- · This returns **MongoCollection.**
- Methods
- FindAll() - Returns all documents in collection()
- FindOne() - Returns a single document based on Mongo Query object generated based on **_id.**
- Save() - Save a new document in collection.
- Update() - Update a document.
- Remove() - Remove a document.

The above code uses all these methods for performing CRUD operations.

Step 4: We will register the DataAccess class in the Dependency Injection feature provided by the ASP.NET Core. To do this open the Start.cs file and add the following line in ConfigureServices() method

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<DataAccess>();
    services.AddMvc();
}
```

Step 5: In the Controllers folder, add a new Web API Controller class of name ProductAPIController as shown in the following image



In this class add the following code

```
using System.Collections.Generic;
using Microsoft.AspNet.Mvc;

using MVC6_WEBAPI_MongoDB.Models;
using MongoDB.Bson;

namespace MVC6_WEBAPI_MongoDB.Controllers
{
    [Route("api/Product")]
    public class ProductAPIController : Controller
    {
        DataAccess objds;

        public ProductAPIController()
        {
            objds = d;
        }

        [HttpGet]
        public IEnumerable<Product> Get()
        {
            return objds.GetProducts();
        }
        [HttpGet("{id:length(24)}")]
        public IActionResult Get(string id)
        {
            var product = objds.GetProduct(new ObjectId(id));
            if (product == null)
            {
                return NotFound();
            }
            return new ObjectResult(product);
        }

        [HttpPost]
        public IActionResult Post([FromBody]Product p)
        {
            objds.Create(p);
            return new HttpOkObjectResult(p);
        }
        [HttpPut("{id:length(24)}")]
        public IActionResult Put(string id, [FromBody]Product p)
        {
            var recId = new ObjectId(id);
            var product = objds.GetProduct(recId);
            if (product == null)
            {
                return HttpNotFound();
            }

            objds.Update(recId, p);
            return new OkResult();
        }

        [HttpDelete("{id:length(24)}")]
        public IActionResult Delete(string id)
        {
            var product = objds.GetProduct(new ObjectId(id));
            if (product == null)
            {
                return NotFound();
            }

            objds.Remove(product.Id);
            return new OkResult();
        }
    }
}
```

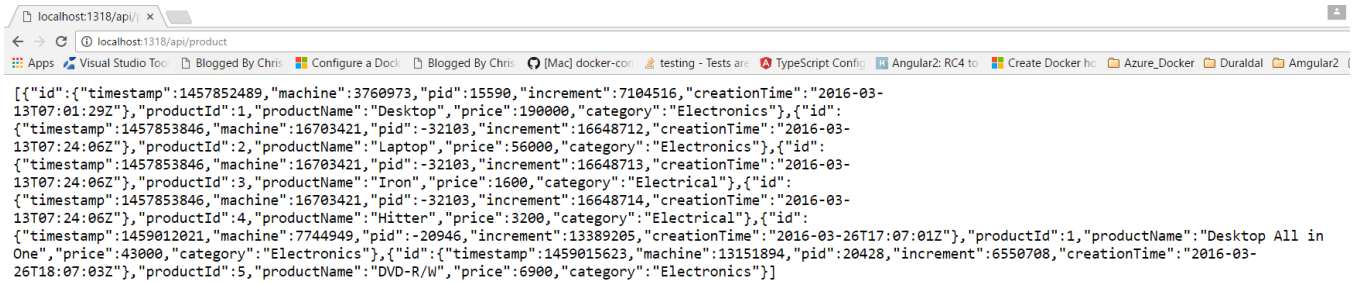
The above Web API class uses DataAccess class for performing CRUD operations. The Web API class contains GET, POST, PUT and DELETE methods for Http operations.

Step 5: Open the **launchSettings.json** in the **Properties** folder and add the following settings in it:

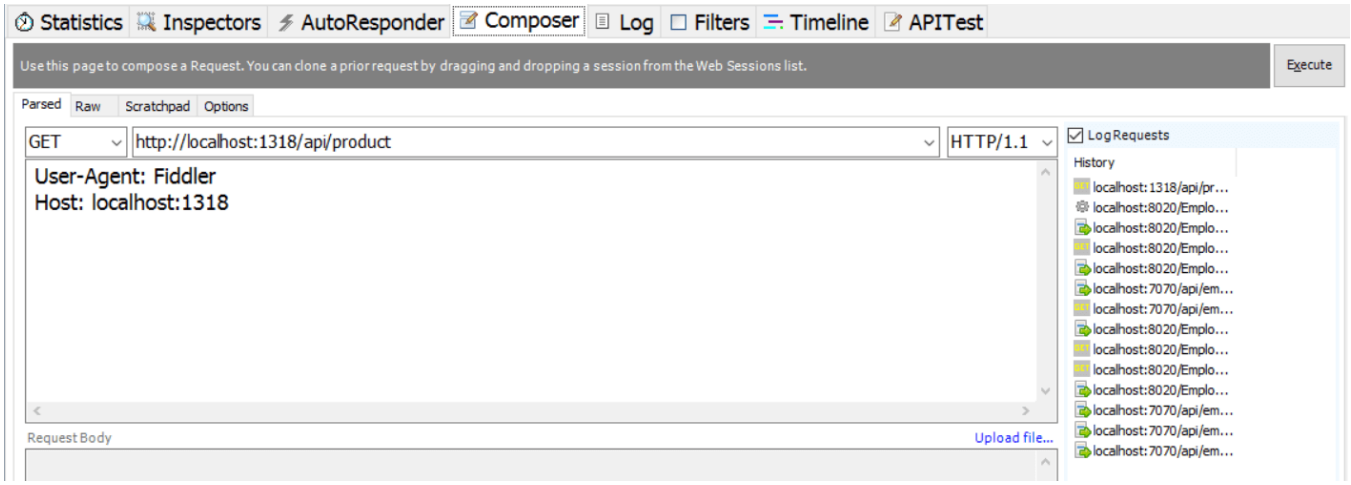
```
"profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "launchUrl": "api/Product",
      "environmentVariables": {
        "Hosting:Environment": "Development"
      }
    }
```

This provides launchUrl to run the application in IIS Express.

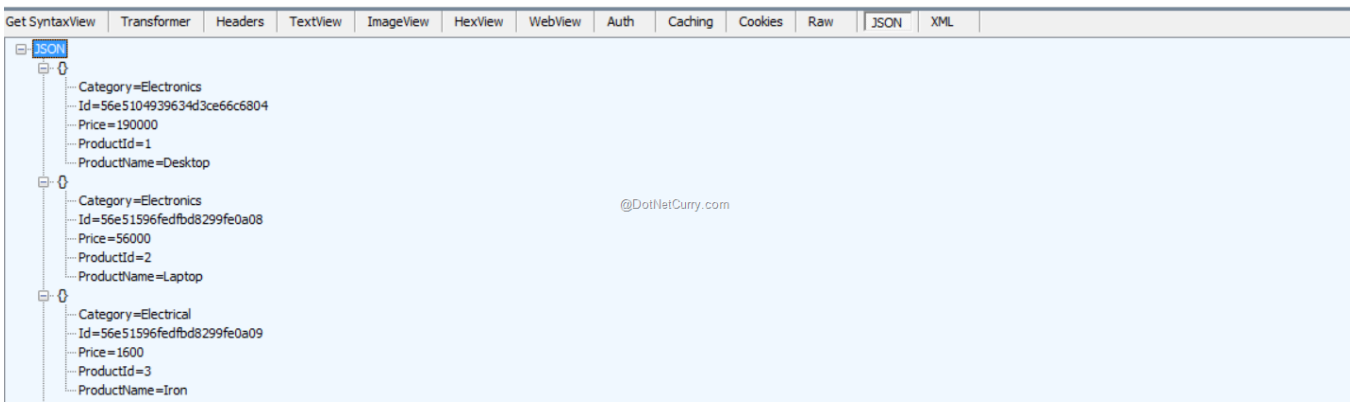
Run the application in browser and the following result will be displayed:



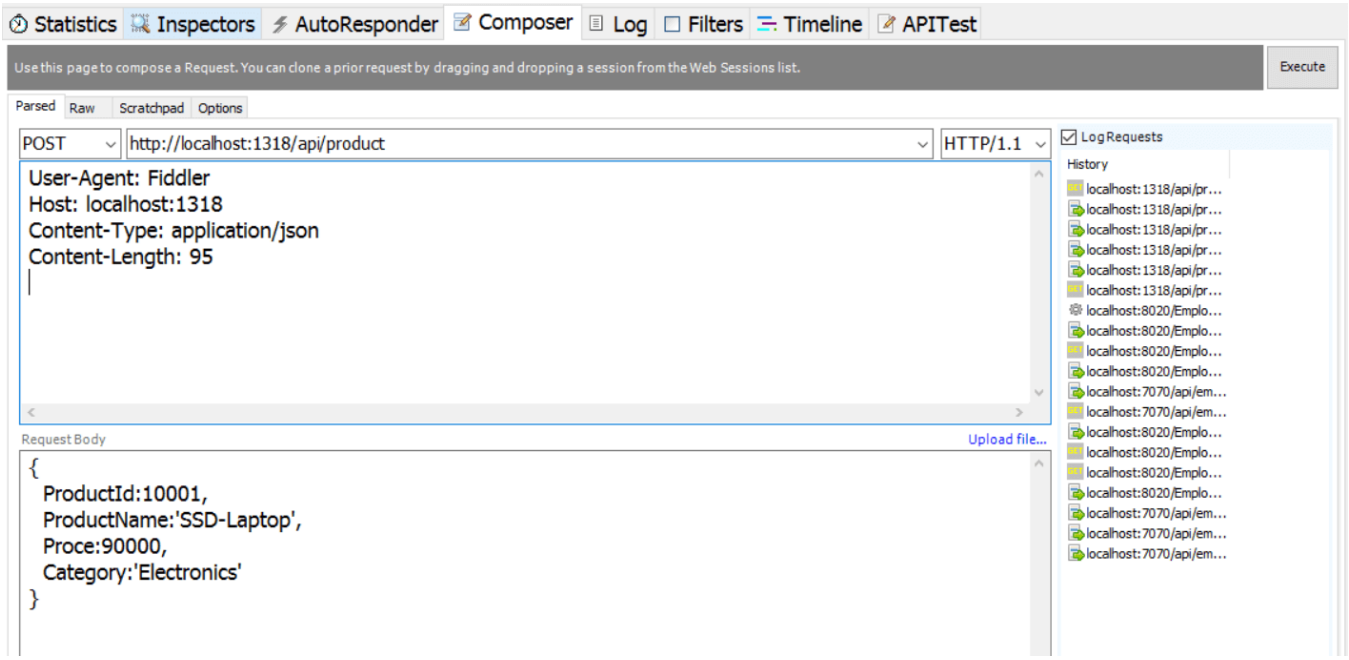
To Test this we will make use of Fiddler tool. Open the fiddler tool and enter the following URL in it



Click on **Execute** button and the following result will be displayed



To Post the data, enter the following details in Fiddler



Click on Execute button and the data will be posted. Run the following command from the Mongo Command prompt

```
>db.Products.find({})
```

The following result will be displayed

```
{ "_id" : ObjectId("57e6ba890a085531204aec9c"), "ProductId" : 10001, "ProductName" : "SSD-Laptop",
```

Conclusion: Using **Mongo C# Driver** we can easily connect to the popular MongoDB database and perform CRUD operations. Using ASP.NET WebAPI, MongoDB data can be easily made available to various client apps for storing and reading data.

Download the source code of this article (https://github.com/dotnetcurry/mvc6-mongodb-webapi) (Github)

This article has been editorially reviewed by Suprotim Agarwal. (https://www.dotnetcurry.com/author/suprotim-agarwal)

WHAT OTHERS ARE READING!

gRPC with ASP.NET Core 3.0 (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1514)

Authentication in ASP.NET Core, SignalR and VueJS applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1511)

Role Based Security in an ASP.NET Core Application (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1505)

Zero Downtime Deployment for ASP.NET applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1503)

Developing Web Applications in .NET (Different Approaches and Current State) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1501)

ASP.NET Core Vue CLI Templates (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1500)

Was this article worth reading? Share it with fellow developers too. Thanks!

Share on Facebook (https://facebook.com/sharer/sharer.php?u=https%3A%2F%2Fwww.dotnetcurry.com%2Faspnet-mvc%2F1267%2Fusing-mongodb-nosql-database-with-aspnet-webapi-core) Share on Twitter (https://twitter.com/intent/tweet/?text=Using%20MongoDB%20with%20Web%20API%20and%20ASP.NET%20Core%20%7C%20DotNetCurry&url=https%3A%2F%2Fwww.dotnetcurry.com%2Faspnet-mvc%2F1267%2Fusing-mongodb-nosql-database-with-aspnet-webapi-core) Share on LinkedIn (https://www.linkedin.com/shareArticle?mini=true&url=https%3A%2F%2Fwww.dotnetcurry.com%2Faspnet-mvc%2F1267%2Fusing-mongodb-nosql-database-with-aspnet-webapi-core&title=Using%20MongoDB%20with%20Web%20API%20and%20ASP.NET%20Core%20%7C%20DotNetCurry) Share on Google+ (https://plus.google.com/share?url=https%3A%2F%2Fwww.dotnetcurry.com%2Faspnet-mvc%2F1267%2Fusing-mongodb-nosql-database-with-aspnet-webapi-core)

AUTHOR

Mahesh Sabnis is a DotNetCurry author and Microsoft MVP having over 17 years of experience in IT education and development. He is a Microsoft Certified Trainer (MCT) since 2005 and has conducted various Corporate Training programs for .NET Technologies (all versions). Follow him on twitter @maheshdotnet (http://twitter.com/maheshdotnet)

FEEDBACK - LEAVE US SOME ADULATION, CRITICISM AND EVERYTHING IN BETWEEN!

Click here to post your Comments

CATEGORIES

.NET Web

.NET Framework, Visual Studio and C#

Patterns & Practices

Cloud and Mobile

JavaScript

.NET Desktop

Interview Questions & Product Reviews

JOIN OUR COMMUNITY

51,659
fans
(https://www.facebook.com/dotnetcurry)

7,923
followers
(https://www.twitter.com/dotnetcurry)

106,812
subscribers
(https://www.dotnetcurry.com/magazine/)

POPULAR ARTICLES

Developing Web Applications in .NET (Different Approaches and Current State) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1501)

ASP.NET Core Vue CLI Templates (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1500)

Role Based Security in an ASP.NET Core Application (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1505)

Reactive Azure Service Bus Messaging with Azure Event Grid (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1498)

Authentication in ASP.NET Core, SignalR and VueJS applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1511)

State in Multi-threaded C# Applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1502)

Shipping Pseudocode to Production (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1497)

Azure DevOps for Angular Applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1504)

Developing Desktop applications in .NET (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1506)

Asynchronous Producer Consumer Pattern in .NET (C#) (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1509)

Using Secrets in Azure Pipelines (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1507)

Microsoft Azure Cloud Roadmap (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1512)

What’s New in .NET Core 3.0? (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1513)

Azure DevOps for TypeScript React.JS App (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1499)

Zero Downtime Deployment for ASP.NET applications (http://www.dotnetcurry.com/ShowArticle.aspx?ID=1503)

C# .NET BOOK

(http://www.dotnetcurry.org/r/dnc-csharpbk-web-300x600x2-green)

Tags

ASP.NET MVC (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ASPNET-MVC)

ASP.NET CORE (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ASPNET-CORE)

ASP.NET (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ASPNET)

SHAREPOINT (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/SHAREPOINT)

DESIGN PATTERNS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/PATTERNS-PRACTICES)

C# (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/CSHARP)

LINQ (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/LINQ)

WPF (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/WPF)

WCF (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/WCF)

VISUAL STUDIO (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/VISUALSTUDIO)

VSTS & TFS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/VSTS-TFS)

AZURE (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/WINDOWS-AZURE)

ENTITY FRAMEWORK (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ENTITYFRAMEWORK)

ANGULAR.JS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/ANGULARJS)

REACT.JS (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/REACTJS)

JQUERY (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/JQUERY-ASPNET)

JAVASCRIPT (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/HTML5-JAVASCRIPT)

HTML5 (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/HTML5-JAVASCRIPT)

.NET CORE (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/DOTNET-STANDARD-CORE)

.NET FRAMEWORK (HTTPS://WWW.DOTNETCURRY.COM/TUTORIALS/DOTNETFRAMEWORK)

(https://www.dotnetcurry.com)

SERVER-SIDE

ASP.NET (https://www.dotnetcurry.com/tutorials/aspnet)

ASP.NET Core (https://www.dotnetcurry.com/tutorials/aspnet-core)

ASP.NET MVC (https://www.dotnetcurry.com/tutorials/aspnet-mvc)

WCF (https://www.dotnetcurry.com/tutorials/wcf)

SharePoint (https://www.dotnetcurry.com/tutorials/sharepoint)

CLIENT-SIDE

Angular.js (https://www.dotnetcurry.com/tutorials/angularjs)

React.js (https://www.dotnetcurry.com/tutorials/reactjs)

jQuery (https://www.dotnetcurry.com/tutorials/jquery-aspnet)

Backbone.js (https://www.dotnetcurry.com/tutorials/backbonejs)

HTML5 (https://www.dotnetcurry.com/tutorials/html5-javascript)

CSS (https://www.dotnetcurry.com/tutorials/bootstrap-css)

.NET

C# (https://www.dotnetcurry.com/tutorials/csharp)

Visual Studio (https://www.dotnetcurry.com/tutorials/visualstudio)

VSTS & TFS (https://www.dotnetcurry.com/tutorials/vsts-tfs)

LINQ (https://www.dotnetcurry.com/tutorials/linq)

Entity Framework (https://www.dotnetcurry.com/tutorials/entityframework)

.NET Framework (https://www.dotnetcurry.com/tutorials/dotnetframework)

.NET Standard & .NET Core (https://www.dotnetcurry.com/tutorials/dotnet-standard-core)

WPF (https://www.dotnetcurry.com/tutorials/wpf)

WinForms (https://www.dotnetcurry.com/tutorials/winforms)

CLOUD AND MOBILE

Microsoft Azure (https://www.dotnetcurry.com/tutorials/windows-azure)

DevOps (https://www.dotnetcurry.com/tutorials/devops)

Xamarin (https://www.dotnetcurry.com/tutorials/xamarin)

Powershell (https://www.dotnetcurry.com/tutorials/powershell)

Machine Learning & AI (https://www.dotnetcurry.com/tutorials/machine-learning-ai)

UWP & Windows Store (https://www.dotnetcurry.com/tutorials/windows-store)

Windows Phone (https://www.dotnetcurry.com/tutorials/windowsphone)

SKILL UP

Design Patterns (https://www.dotnetcurry.com/tutorials/patterns-practices)

Software Gardening (https://www.dotnetcurry.com/tutorials/software-gardening)

.NET Interview Q&A (https://www.dotnetcurry.com/tutorials/dotnetinterview)

Magazines (https://www.dotnetcurry.com/magazine/)

Books (http://www.jquerycookbook.com/)

Product Reviews (https://www.dotnetcurry.com/tutorials/product-articles-review)

FOLLOW US

- Facebook (https://www.facebook.com/dotnetcurry)
- Twitter (https://www.twitter.com/dotnetcurry)
- Github (https://github.com/dotnetcurry)
