Create RESTful API Using ASP.NET Core With Entity Framework Core

Introduction

REST

Representational State Transfer is an architectural style where we use a standard to implement it.

The browser is an HTTP client. While it points to any URI, it sends the HTTP request to the server. The server will process it and send back the HTTP Response which contains the representation of the page, probably it can be HTML, JSON or XML and many more media types, based on the response the browser will change its state. The client changes its state depending on the representation of the resource which is accessed is called as representational state transfer or REST.

ASP.NET Core

ASP.NET Core is an open source, cross-platform framework for building modern internet connected applications, click here to learn more about ASP.NET Core. ASP.NET Core MVC is a middleware which provides a framework for building the APIs and web application using MVC pattern

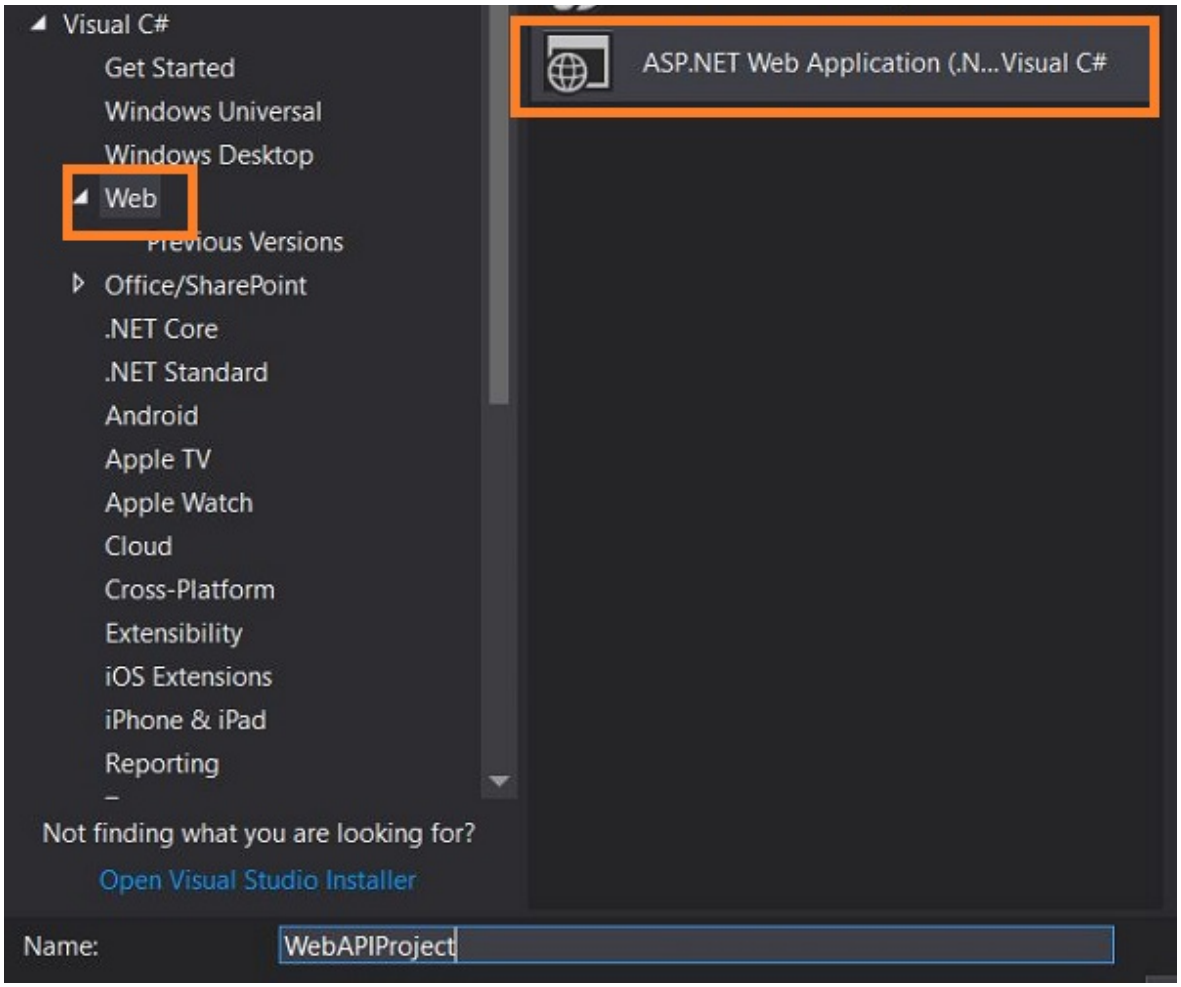
Tooling

- Visual Studio 2017
- Postman

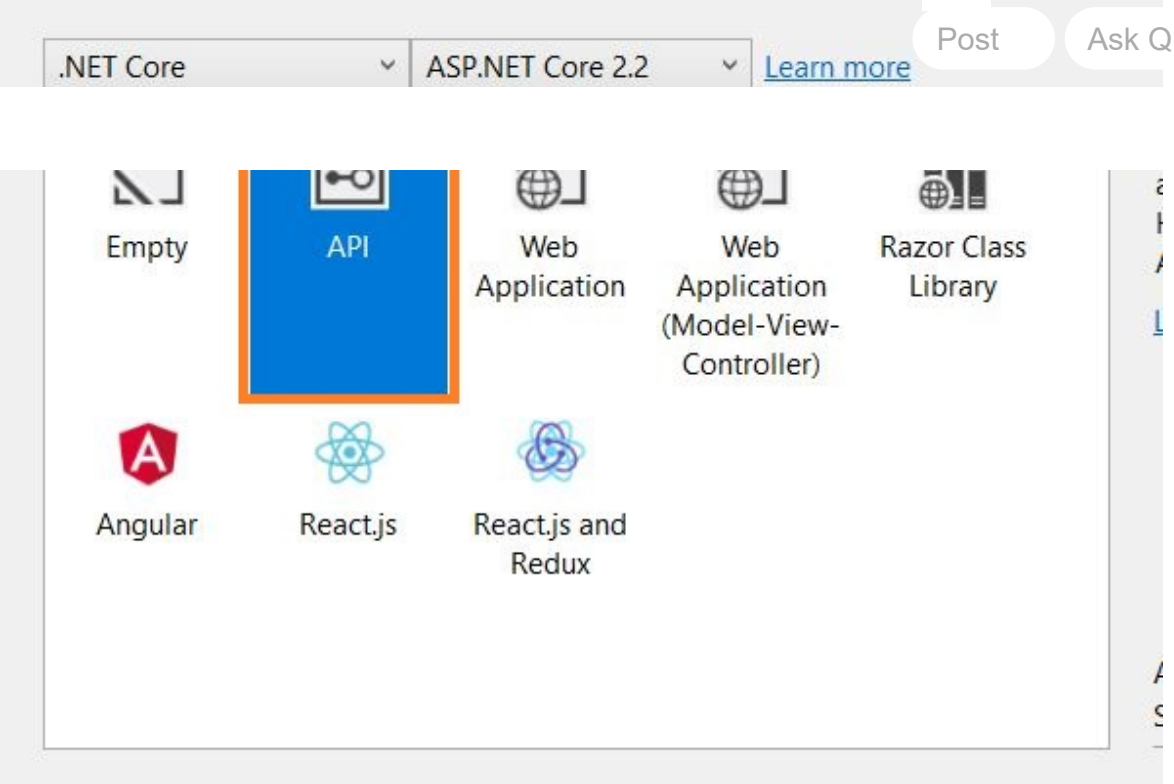
Download the latest Version of .NET CORE. Currently, I used ASP.NET Core 2.2 to develop the application.

Create an ASP.NET Core application

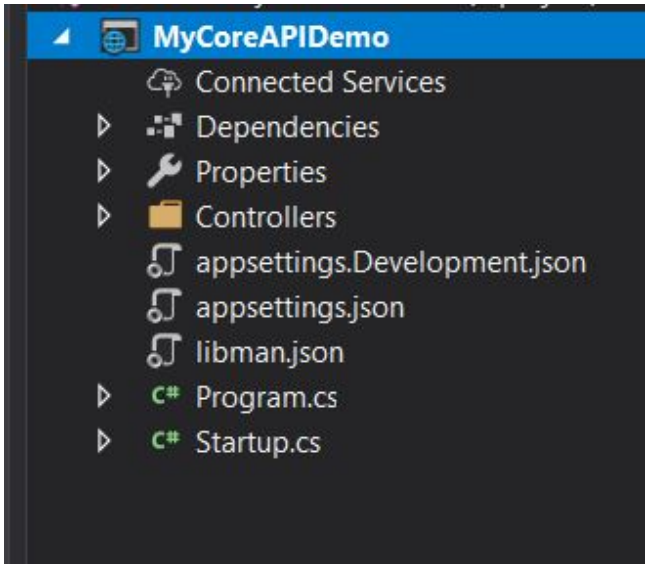
OK.



Choose an API template as shown in the below figure.



By clicking on OK, it will create a new ASP.NET Core project with some pre-defined configuration files and controller.



The program.cs class which contains the main method with a method called Createwebhostbuilder(), is responsible for running and configuring the application. The host for the application is set up with the startup type as startup class.

The startup.cs class contains two important methods,

Configure() - it is used to configure how the ASP.NET Core appli individual HTTP request.

Configure the Entity Framework Core

Create a folder called Entities to organize the entity model classes. Let's create an entity model class.

Author.cs

```
[Table("Author",Schema ="dbo")]
public class Author
{
    [Key]

    public Guid AuthorId { get; set; }

    [Required]
    [MaxLength(50)]
    public string FirstName { get; set; }
    [Required]
    [MaxLength(50)]
    public string LastName { get; set; }

    [Required]
    [MaxLength(50)]
    public string Genre { get; set; }

    public ICollection<Book> Books { get; set; } = new List<Book>();
}
```

Book.cs

```
public class Book
{
    [Key]
    public  Guid BookId { get; set; }
    [Required]
    [MaxLength(150)]
    public string Title { get; set; }
    [MaxLength(200)]
    public string Description { get; set; }

    [ForeignKey("AuthorId")]
    public Author Author { get; set; }

    public Guid AuthorId { get; set; }

}
```

Let's create a context file, add a new class file, and name it as Li

```
  public class LibraryContext:DbContext
        {
        public LibraryContext(DbContextOptions<LibraryContext> options):base
        {
            Database.Migrate();
        }
        public DbSet<Author> Authors { get; set; }
        public DbSet<Book> Books { get; set; }
}
```

Let's define the database connection in the appsettings.json file.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionString": {
    "BookStoreDB": "server=server name;database=BookStore;User ID= server us
  },
  "AllowedHosts": "*"
}
```

Finally, let's register our context in Startup.cs.

```
public void ConfigureServices(IServiceCollection services)
{

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    services.AddDbContext<LibraryContext>
(op => op.UseSqlServer(Configuration["ConnectionString:BookStoreDB"]));
}
```

Generate Database from code-first approach

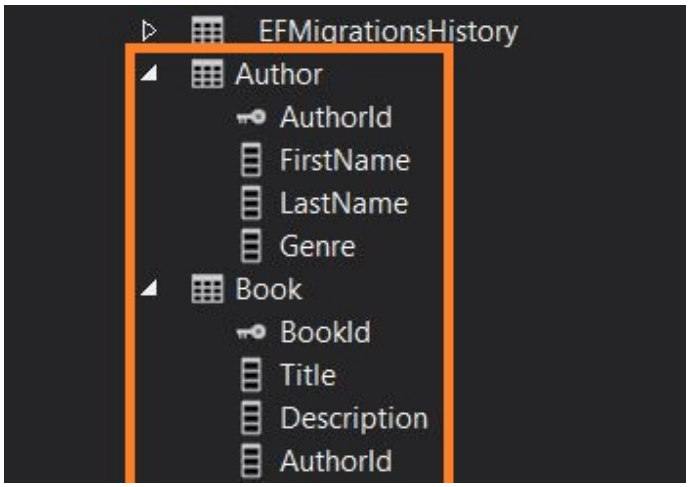
Run the following command in the Package Manager console.

```
Add-Migration MyCoreAPIDemo.Entities.LibraryContext
```

This will create a class for migration. Run the following command to update the database.

```
Update-database
```

Let's verify that database and tables from server explorer in Vis



From the above image, you can notice the tables are created based on our model.

Seeding data

Let's add some data to the Author table. For this, we need to override a method OnModelCreating in the LibraryContext class.

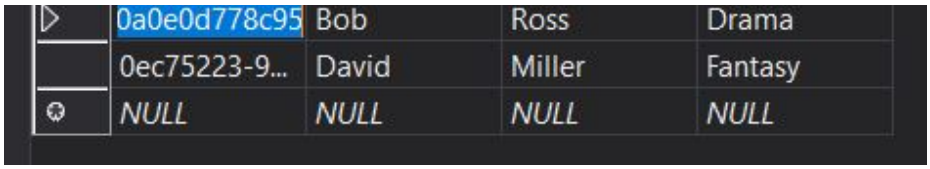
LibraryContext.cs

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Author>().HasData(new Author
    {
        AuthorId= Guid.NewGuid(),
        FirstName = "Bob",
        LastName = "Ross",
        Genre = "Drama"

    }, new Author
    {
        AuthorId=Guid.NewGuid(),
        FirstName = "David",
        LastName = "Miller",
        Genre = "Fantasy"
    });
}
```

Let's run the migration and update command once again.

```
Add-Migration MyCoreAPIDemo.Entities.LibraryContextSeed

```

Let's check the data from server explorer.



▷	0a0e0d778c95	Bob	Ross	Drama
	0ec75223-9...	David	Miller	Fantasy
	NULL	*NULL*	*NULL*	*NULL*

From the above image, you can notice we got the data in the table based on our update from code.

Creating a Repository

Let's add a repository folder to implement the repository pattern to access the context method.

Create two more folders - Contract and Implementation - under the repository folder.

Create an interface ILibraryRepository.cs under Contract folder.

ILibraryRepository.cs

```
01. public interface ILibraryRepository<T>
02. {
03.     IEnumerable<T> GetAllAuthor();
04. }
```

Let's create a class under implementation folder to implement the function.

LibraryRepository.cs

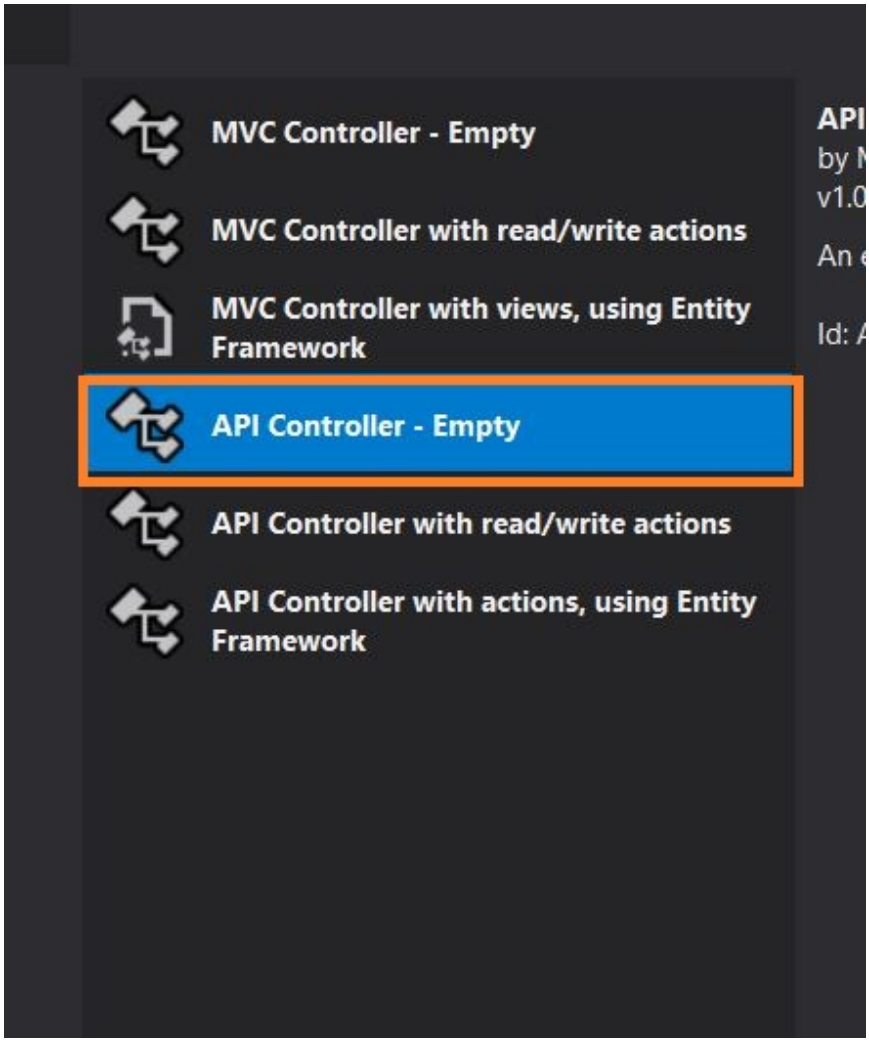
```
01. public class LibraryRepository: ILibraryRepository<Author>
02. {
03.     readonly LibraryContext _libraryContext;
04.
05.     public LibraryRepository(LibraryContext context)
06.     {
07.         _libraryContext = context;
08.     }
09.
10.     public IEnumerable<Author> GetAllAuthor()
11.     {
12.         return _libraryContext.Authors.ToList();
13.     }
14. }
```

The above method GetAllAuthor() will return the complete list of records from Author table.

code in ConfigurationServices method.

Create API Controller

Right-click on controller and go to Add->Controller. Choose an empty API template and name the controller. I named the controller as LibrariesController.



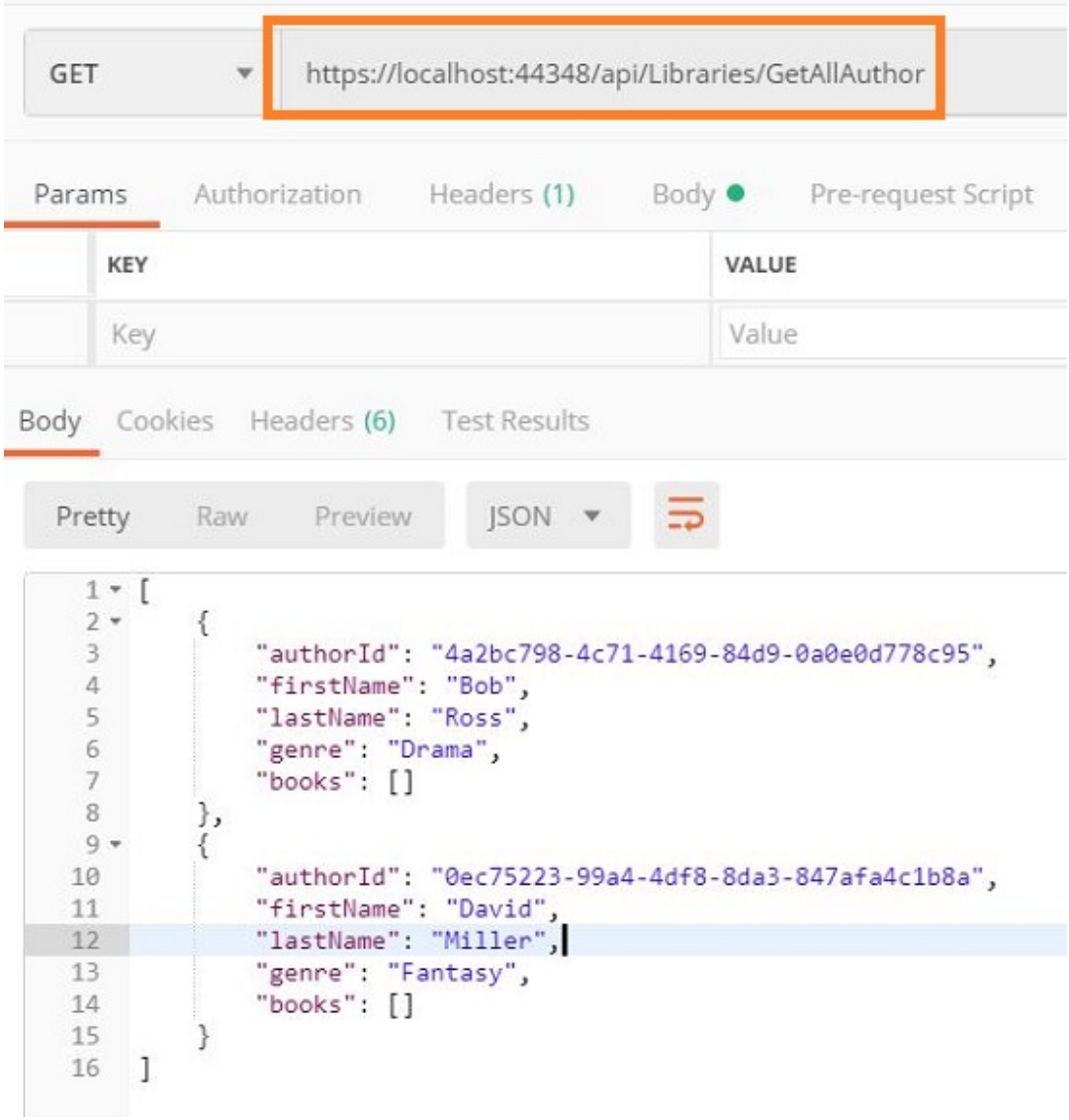
LibrariesController.cs

```
[Route("api/Libraries")]
 [ApiController]
 public class LibrariesController : ControllerBase
 {
     private readonly ILibraryRepository<Author> _libraryRepository;

     public LibrariesController(ILibraryRepository<Author> libraryRepository
     {
         _libraryRepository = libraryRepository;
```

```
        // GET: api/Libraries/GetAllAuthor
        [HttpGet]
        [Route("GetAllAuthor")]


            IEnumerable<Author> authors = _libraryRepository.GetAllAuthor();
            return Ok(authors);
        }

    }
```

Yes, we have created a WEB API with an endpoint *api/Libraries/GetAllAuthor* to get an author list from the database.

Let's test the API using Postman tool.



Yes, we got an author list as a response.

Conclusion

We have seen how to create a REST API using the ASP.NET Core with Entity Framework core, will see how to create a CRUD APIs using ASP.NET Core in my future article.

Next Recommended Article

ASP.NET Core 2.1 - Implement Entity Framework Core In A Code First Approach

ASP.NET Core | Entity Framework Core | RESTful API

Gowtham K TOP 100

Gowtham K is awarded as MVP by Microsoft for his exceptional contribution in Microsoft technologies under the category "Developer Technologies" for the year 2016, 2017 and 2018. He has more than 5 years of expe... Read more

https://www.c-sharpcorner.com/members/gowtham-k

74 4.2m 5 3

13 5

Type your comment here and press Enter Key (Minimum 10 characters)

You could get the same thing with no code: InstantWebAPI.com

George P Mar 12, 2020

1916 2 0 0 0 Reply

Thank you so much for this excellent tutorial!!!

John Somerville Sep 05, 2019

1910 8 0 1 0 Reply

I cannot connect to sql server. When i type Add-Migration command i get a network-related error.

Thomas Athanasiou Aug 10, 2019

1907 11 0 0 0 Reply

But url that is being used here not according to REST

Dheeraj Kumar Jul 02, 2019

1888 30 0 0 0 Reply

FEATURED ARTICLES

CRUD Operations In PostgreSQL With EF Core And ASP.NET Core Web API

Views In Snowflake

Covid-19 Tracker Website With React, Material.UI And Chart.js

PnP Date Time Picker Control In SharePoint Framework

Snowflake And Its Features

View All