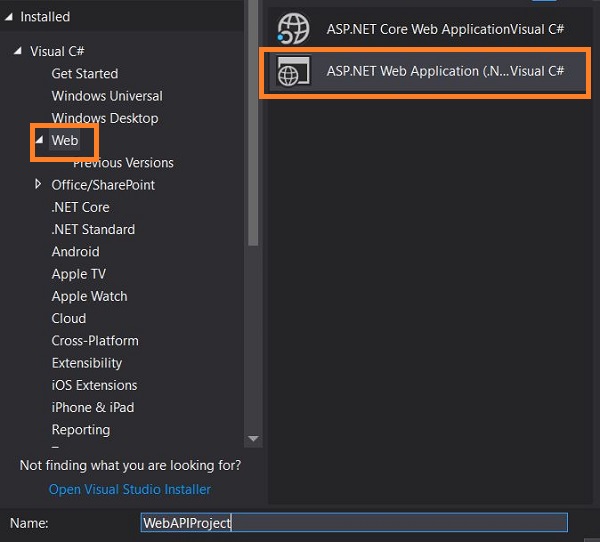
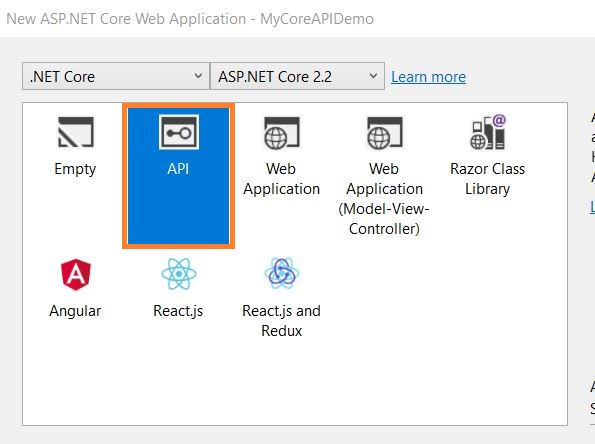
Create RESTful API Using ASP.NET Core With Entity Framework Core

Create an ASP.NET Core application

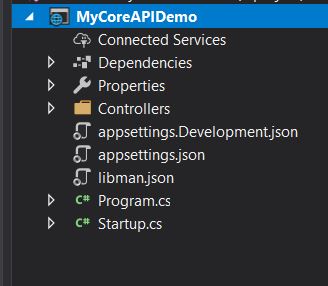
Open Visual Studio, File -> New ->Project, select ASP.NET Core web application template and click OK.



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,

ConfigureServices() - it is used to add service to build dependency injection containers and configure those services,

Configure() - it is used to configure how the ASP.NET Core application will response to an 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**

1. [Table("Author",Schema ="dbo")]
2. **public** **class** Author
3. {

    [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>();

1. }

**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**; }

}

Creating a context file

Let’s create a context file, add a new class file, and name it as LibraryContext.cs.

**LibraryContext.cs**

**public** **class** LibraryContext:DbContext

        {

**public** LibraryContext(DbContextOptions<LibraryContext> options):**base**(options)

        {

            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.

1. {
2. "Logging": {
3. "LogLevel": {
4. "Default": "Warning"
5. }
6. },
7. "ConnectionString": {
8. "BookStoreDB": "server=server name;database=BookStore;User ID= server user id;password= your server password;"
9. },
10. "AllowedHosts": "\*"
11. }

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

1. **public** **void** ConfigureServices(IServiceCollection services)
2. {
3. services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);
4. services.AddDbContext<LibraryContext>(op => op.UseSqlServer(Configuration["ConnectionString:BookStoreDB"]));
5. }

**Generate Database from code-first approach**

Run the following command in the Package Manager console.

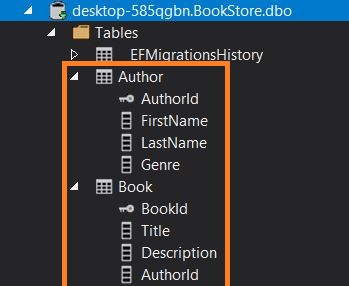
1. Add-Migration MyCoreAPIDemo.Entities.LibraryContext

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

1. Update-database

This will update the database based on our models,

Let’s verify that database and tables from server explorer in Visual Studio.



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"

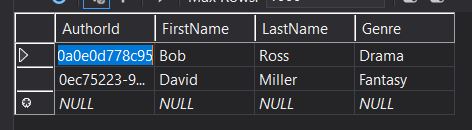
    });

}

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

1. Add-Migration MyCoreAPIDemo.Entities.LibraryContextSeed
3. Update-database

Let’s check the data from server explorer.



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**

**public** **interface** ILibraryRepository<T>

{

    IEnumerable<T> GetAllAuthor();

}

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

**LibraryRepository.cs**

**public** **class** LibraryRepository: ILibraryRepository<Author>

{

    readonly LibraryContext \_libraryContext;

**public** LibraryRepository(LibraryContext context)

    {

        \_libraryContext = context;

    }

**public** IEnumerable<Author> GetAllAuthor()

    {

**return** \_libraryContext.Authors.ToList();

    }

}

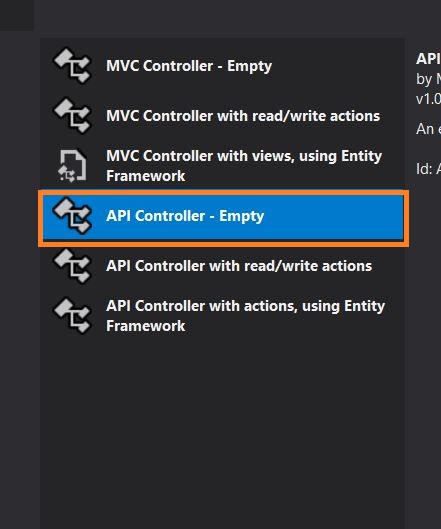
The above method GetAllAuthor() will return the complete list of records from Author table.

Let’s configure the repository using dependency injection. Open Startup.cs file, add the below code in ConfigurationServices method.

1. services.AddScoped<ILibraryRepository<Author>, LibraryRepository>();

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")]

**public** IActionResult GetAllAuthor()

     {

         IEnumerable<Author> authors = \_libraryRepository.GetAllAuthor();

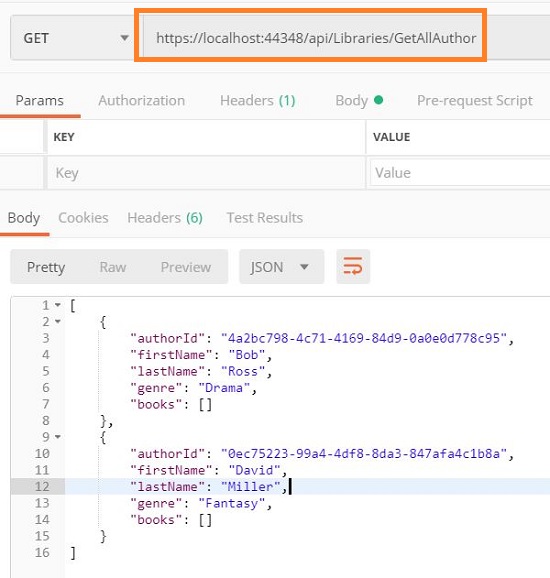
**return** Ok(authors);

     }

 }

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.

## What is CORS?

Cross-Origin Resource Sharing (CORS) manages the cross-origin requests. Unlike same-origin policy, CORS allows making a request from one origin to another. CORS allows the servers to specify who can access the resource on the server from outside.

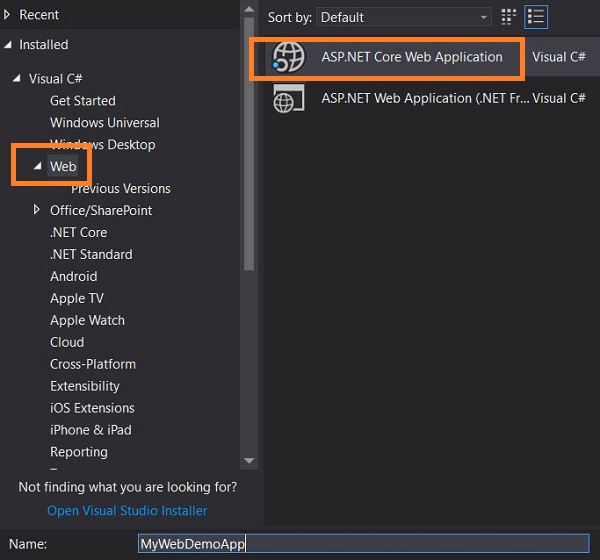
The origin is made up of three parts - the protocol, host, and the port number.

Cross Domain call

Before enabling the CORS, let’s see how the cross-domain call is restricted. Let’s create an ASP.NET Core web application.

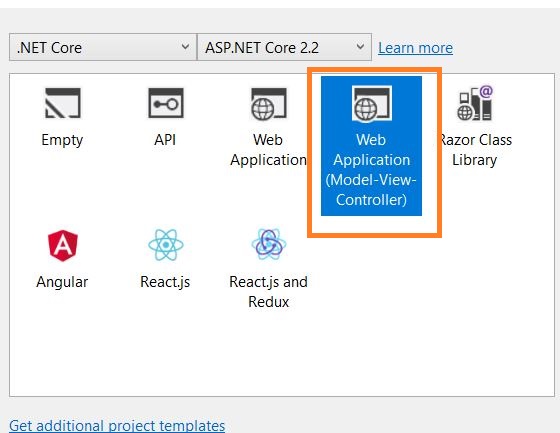
**Step1**

Open Visual Studio, click on NEW ->Project. Select ASP.NET Web Application template under Web, as shown in the below figure.



**Step 2**

Select web application (Model-View-Controller) template, as shown in the below figure,



**Step 3**

Click OK. This will create a web application with a default template.

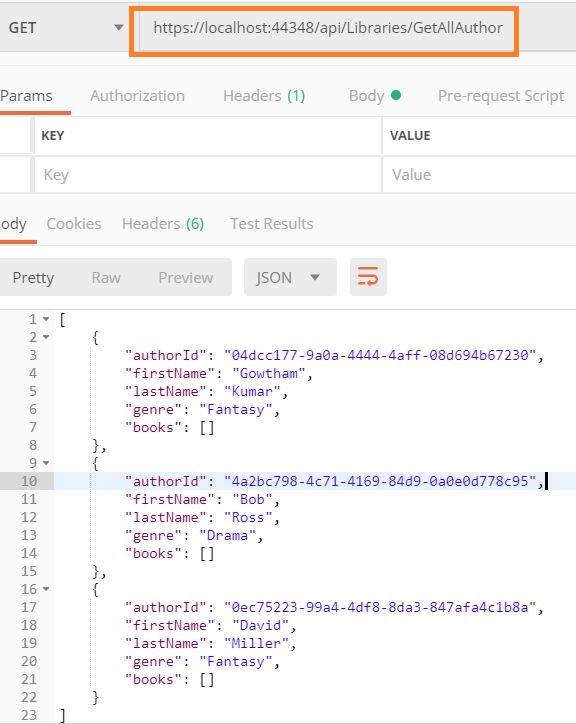
**Step 4**

Go to the Index.cshtml page and add the below code and run the application.

1. <script>
2. $.ajax({
3. url: "https://localhost:44348/api/Libraries/GetAllAuthor",
4. success: function (result) {
5. console.log(result);
6. }
7. })
8. </script>

From the above code, you can notice the AJAX call I made to access the API which is not from the same origin. This is from the ASP.NET Core API application which is created in my last [article](https://www.c-sharpcorner.com/article/create-restful-api-using-asp-net-core-with-entity-framework-corecreate-restful-a/).

Testing the API in the Postman tool.



In the browser console, you will get an error message as shown in the below figure.

Enabling CORS In ASP.NET Core API Application

Now it’s time to Enable CORS in our API application so that we can access it from a different origin.

**Enable CORS in ASP.NET Core API Application**

**Enabling CORS Globally**

Open the ASP.NET Core API application which we created in my last article.

Go to Startup.cs file and add the below code in Configure method, which will inject CORS into a container.

1. app.UseCors(options => options.AllowAnyOrigin());

Add the below code in ConfigureServices method,

services.AddCors(c =>

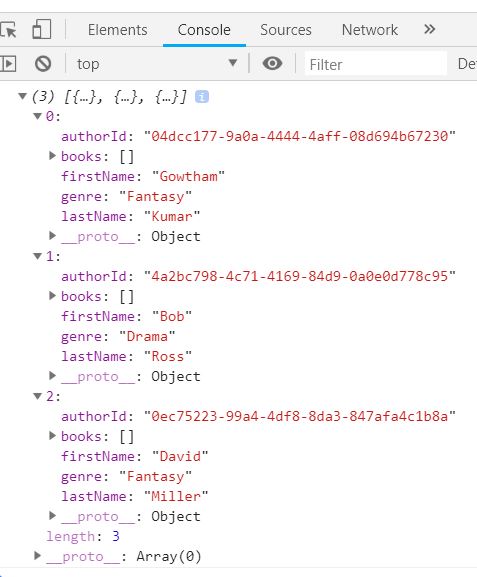
{

    c.AddPolicy("AllowOrigin", options => options.AllowAnyOrigin());

});

The above code tells that the API’s can be accessed from any origin globally.

Run the application,



From the above figure you can notice we got a response from the API successfully and the response it printed in browser console was as expected.

**Enabling for origin**

Go to Startup.cs file and add the below code in Configure method,

1. app.UseCors(options=>options.WithOrigins("https://localhost:44342"));

Add the below code in ConfigureServices method

services.AddCors(c =>

            {

                c.AddPolicy("AllowOrigin", options => options.WithOrigins("https://localhost:44342"));

            });

Go to controller and decorate the action with Enable CORS attribute, as given below,

1. // GET: api/Libraries/GetAllAuthor
2. [HttpGet]
3. [Route("GetAllAuthor")]
4. [EnableCors("AllowOrigin")]
5. **public** IActionResult GetAllAuthor()
6. {
7. IEnumerable<Author> authors = \_libraryRepository.GetAllAuthor();
8. **return** Ok(authors);
9. }

Now this API can be accessed only from the origin https://localhost:44342.

We can also define EnableCors at the controller level so that all the actions under this controller can be accessed from the origin https://localhost:44342

1. [Route("api/Libraries")]
2. [ApiController]
3. [EnableCors("AllowOrigin")]
4. **public** **class** LibrariesController : ControllerBase
5. {
6. **private** **readonly** ILibraryRepository<Author> \_libraryRepository;


10. **public** LibrariesController(ILibraryRepository<Author> libraryRepository)
11. {
12. \_libraryRepository = libraryRepository;
13. }
15. // GET: api/Libraries/GetAllAuthor
16. [HttpGet]
17. [Route("GetAllAuthor")]
19. **public** IActionResult GetAllAuthor()
20. {
21. IEnumerable<Author> authors = \_libraryRepository.GetAllAuthor();
22. **return** Ok(authors);
23. }
24. }
25. }