**Hands-on .NetCore 3.0**

**ASP.Net Core Web API App**

Table of Contents

[Scaffolding Console App Basics 3](#_Toc24920910)

[Creating ASP.Net WebApi Core App using CLI 3](#_Toc24920911)

[Configuring the MS SQL Database 3](#_Toc24920912)

[Connecting to the Database and Creating the New Database Using SSMS 4](#_Toc24920913)

[Creating Model 5](#_Toc24920914)

[Entity Framework Core 3.0: Connecting to the Database 6](#_Toc24920915)

[Installing the Entity Framework Core 3.0 6](#_Toc24920916)

[Create a Context 7](#_Toc24920917)

[Add Connection String 8](#_Toc24920918)

[Adding AutoGenerateBindingRedirect 8](#_Toc24920919)

[Using DI to Read the Connections String 9](#_Toc24920920)

[Add DBContext to Startup 10](#_Toc24920921)

[Alternate to ConfigurationSettings 10](#_Toc24920922)

[Adding Migrations 10](#_Toc24920923)

[Issue with DotNet EF migrations 10](#_Toc24920924)

[Install dotnet-ef 10](#_Toc24920925)

[Add Migrations 11](#_Toc24920926)

[New Migrations Folder Added by the above Action 11](#_Toc24920927)

[Update Database 11](#_Toc24920928)

[Seeding the Data 12](#_Toc24920929)

[Call Seeding Method 13](#_Toc24920930)

[Building Controller 15](#_Toc24920931)

[Adding Controller 15](#_Toc24920932)

[Attributes on Controller and Basic Setup 15](#_Toc24920933)

[Action: GetAllBooks 15](#_Toc24920934)

[Action: Picking a specific book 16](#_Toc24920935)

[Action: Updating a book 16](#_Toc24920936)

[Action: Adding a Book 17](#_Toc24920937)

[Action: Delete a Book 18](#_Toc24920938)

[Testing API 18](#_Toc24920939)

[Getting all books 18](#_Toc24920940)

[Getting book # 2 18](#_Toc24920941)

# Scaffolding Console App Basics

We have already run the commands multiple times, here is the info

* dotnet new console
* dotnet restore [pulls in the dependencies needed by the application]
* dotnet run [compiles and run the application]
* dotnet build [compiles the application]
* dotnet publish [packages up the files for reuse]

Take a look at this document for more details:

<https://itplate.blogspot.com/2019/11/scaffolding-applications-with-net-cli.html>

# Creating ASP.Net WebApi Core App using CLI

|  |  |
| --- | --- |
| Run command **>dotnet new webapi -o WebApi**  It has done the restore for us as well  Open the code with VS code   1. **CD WebApi** 2. and then typing **code .** [code space dot]   The project will look like the image. We’ll put all of our controllers in Controller Folder.  The default template comes with a WeatherForecastController. Press F5 to run the app and then execute the URL as <https://localhost:5001/WeatherForecast>  At this point you should see a JSON response in browser. |  |

# Configuring the MS SQL Database

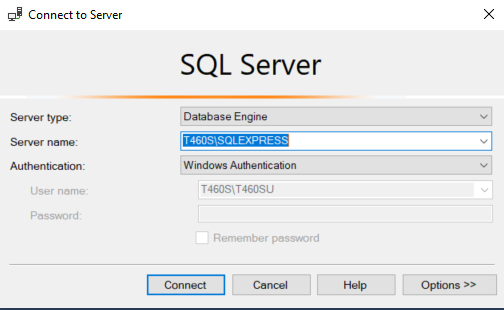
Last time we configured every thing with hand where we first created the model and then applied migrations and then finally updated the database to create the table. Take a look at Notes09 - AspNetCoreMvcApp.docx and then view sections

1. MS SQL Database
2. Creating the Model
3. Entity Framework Core 3.0: Connecting to the Database

In this example, we’ll use the SQL Server Management Studio (SSMS) to create the database.

## Connecting to the Database and Creating the New Database Using SSMS

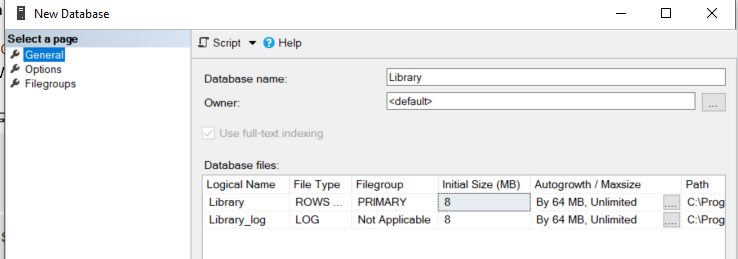
We are using MS SQL Express Database, so open SSMS and connect to it.



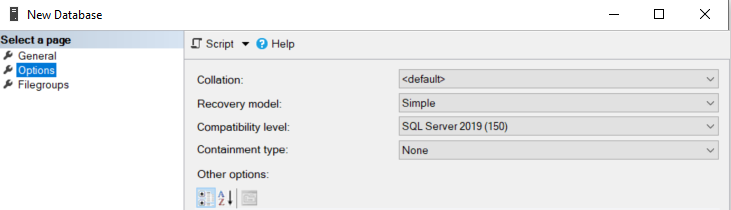
Expand the **databases** node and you’ll see the previous database that we created there “OneDishParty”.

Right click on **Databases** and then click **New Database**.

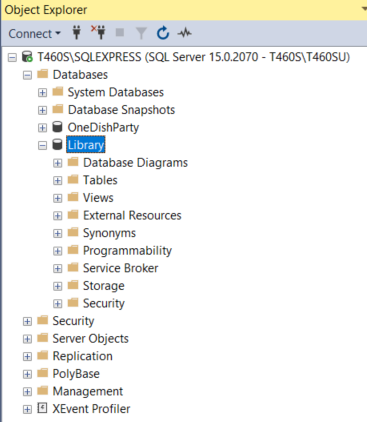
In the database name type “**Library**”



And then on the **options tab**, select “**Simple**” for the **Recovery model**. It will take less space due to this.



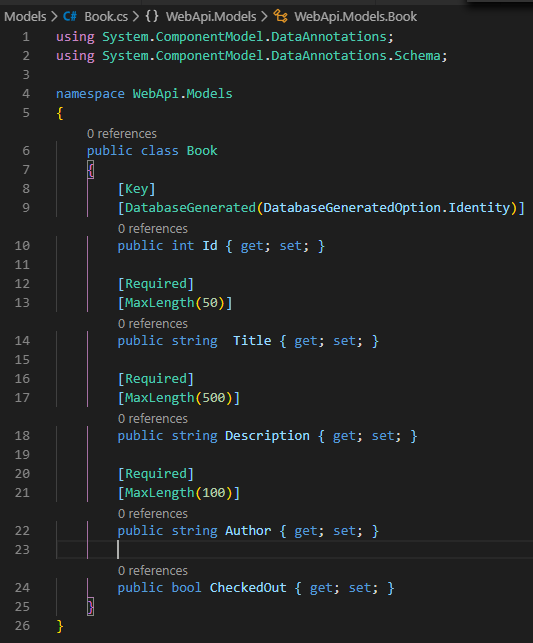
And finally click **Ok** to create the database.



# Creating Model

Since we have created a Library database, it will have books information. Create a **new folder** in the root called **Models** and then add **Book** Model to it.

Right click on Models folder and create a new file with name **Book.cs**. Make sure to out this class in a namespace. Put a cursor at the end of the class name and then press **CTRL+.** and then select create name space with the default name. This will be a POCO class.



Look at the previous example by looking at **Notes09 – AspNetCoreMvcApp.docx** and then section **Creating the Model**.

# Entity Framework Core 3.0: Connecting to the Database

Look at the previous example by looking at **Notes09 – AspNetCoreMvcApp.docx** and then section **Entity Framework Core 3.0: Connecting to the Database**.

## Installing the Entity Framework Core 3.0

On command promt navigate to your working folder and then run the following command

>dotnet add package Microsoft.EntityFrameworkCore.Design

>dotnet add package Microsoft.EntityFrameworkCore.Tools

>dotnet add package Microsoft.EntityFrameworkCore.SqlServer

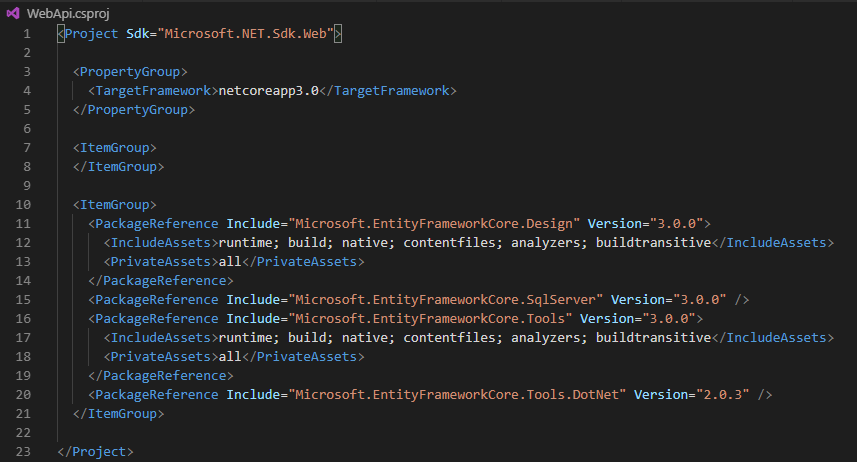
Following will install bunch of CLI tools that we will be using.

>dotnet add package Microsoft.EntityFrameworkCore.Tools.DotNet

And finally issue

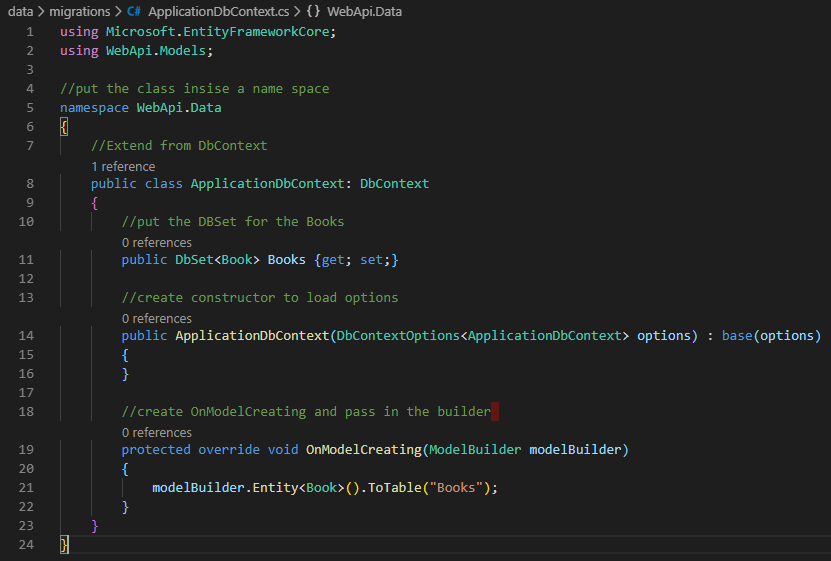
>dotnet restore

The .csproj file should look like



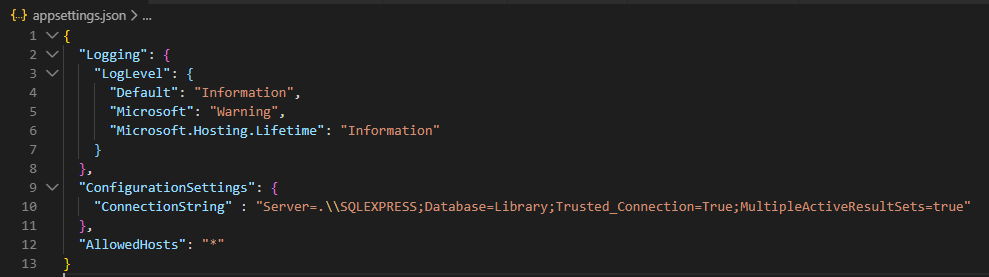
## Create a Context

Create a folder **data** in the root. Then create a folder **migrations** inside it. Then create a file **ApplicationDbContext.cs**.



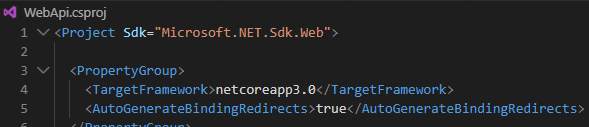
## Add Connection String

Open appsettings.json and add the connection string section to it. Since we are connections to the [SQL Server Express database that we created earlier](#_Connecting_to_the), make sure to put in the proper name here.



## Adding AutoGenerateBindingRedirect

Add the following to the .csproj file

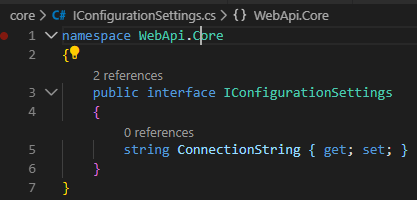


## Using DI to Read the Connections String

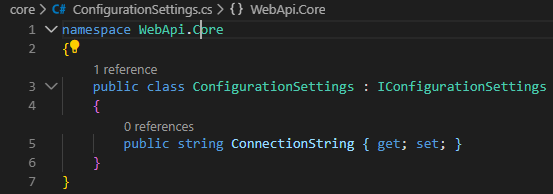
This part will follow what we did in another tutorial. Plz check:

**Notes09 - AspNetCoreMvcApp.docx** 🡺**Using DI to Read the Connection String**

1. Create a folder name “core” in the root
2. Add an interface **IConfigurationSettings** to it



1. Create a class **ConfigurationSettings** and implement the interface **IConfigurationSettings** to it.

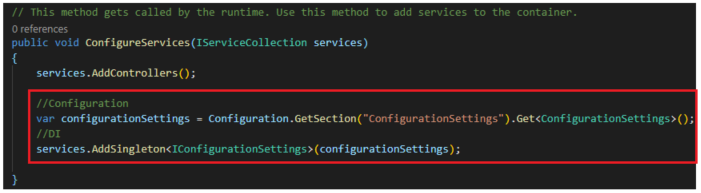


1. Then go to **Startup.cs** file in the root and

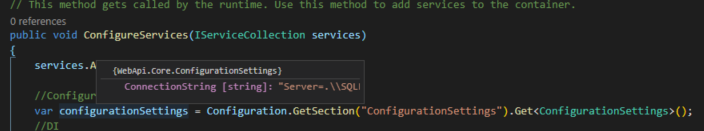
Add the using name space line at the top



And then add the two lines of code to read the section and adding a service for DI.



1. Add the break point and then run the app. ConfigurationSettings should have the connection string populated



## Add DBContext to Startup

Add the following using statements





Then go to ConfigureServices and add the DbContext to it. Will need to pass in the connection string, so use the ConfigurationSettings used above to get the connection string.



### Alternate to ConfigurationSettings

Rather than getting the connection string into configurations, we can create a **ConnectionStrings** section in appsettings.json and then add **DefaultConnection** to it. And here since the DI is not needed, we can just directly use it as



## Adding Migrations

### Issue with DotNet EF migrations

<https://github.com/aspnet/EntityFrameworkCore/issues/15448> Use the below Install before adding the migrations.

### Install dotnet-ef

> **dotnet tool install -g dotnet-ef --version 3.0.0-\***

Since you just installed the .NET Core SDK, you will need to reopen the Command Prompt window before running the tool you installed.

You can invoke the tool using the following command: dotnet-ef

Tool 'dotnet-ef' (version '3.0.0') was successfully installed..

### Add Migrations

>**dotnet-ef migrations add Initial -s "OneDishParty.csproj"**

Or package manager console

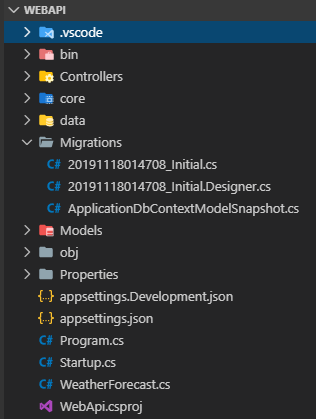
>**Add-Migration Initial**

info: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 3.0.0 initialized 'ApplicationDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None

Done. To undo this action, use 'ef migrations remove'

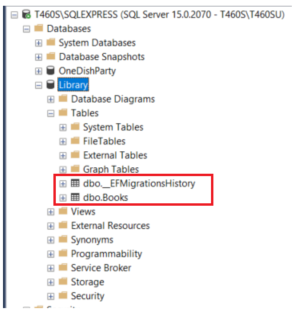
### New Migrations Folder Added by the above Action



## Update Database

Just like create migrations, we need to update the database. Before issuing the command, make sure that the project compiles. Take care of any issues and then run the command.

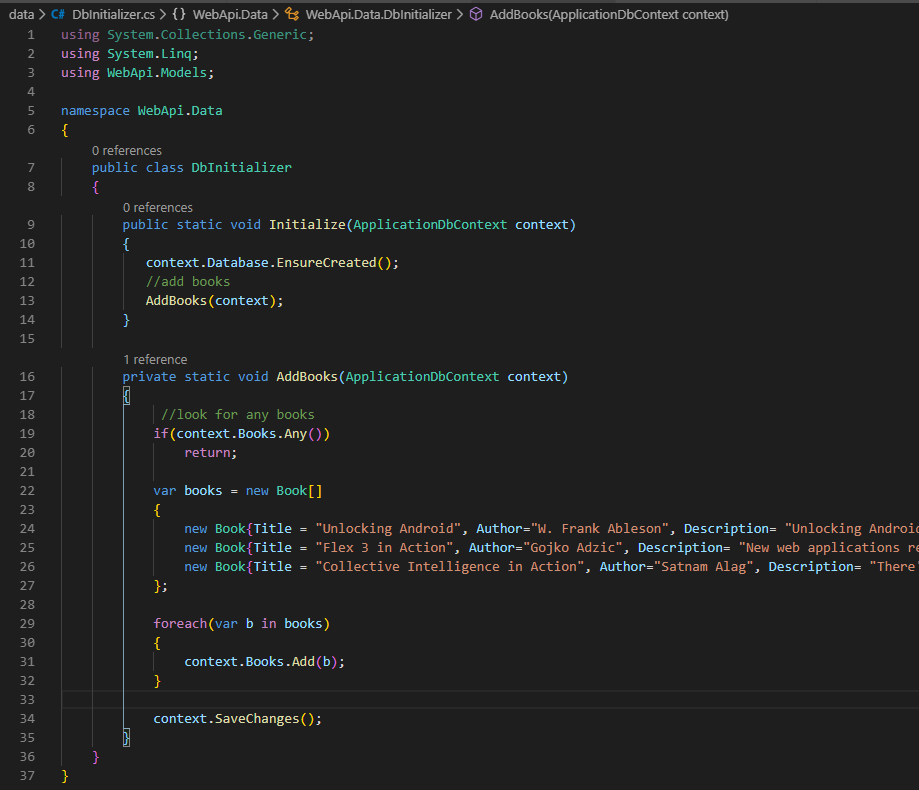
>**dotnet ef database update**



## Seeding the Data

Create a new file inside the folder **data** called **DbInitializer.cs**.

Then create a static method **Initilize**, in this we will initilize the database. We wil pass to it the ***ApplicationDbContext***.

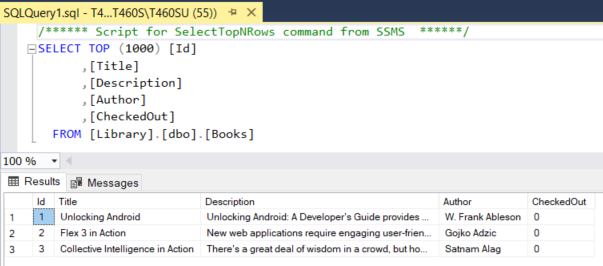


### Call Seeding Method

We have added the books but we need to call it for seeding to happen. Go to Program.cs and we’ll make few changes.

|  |  |
| --- | --- |
| Current View of of the Program.cs | Changes made to the Program.cs |
|  |  |

Run the App after Seeding Methods in place and then look into the DB.



# Building Controller

## Adding Controller

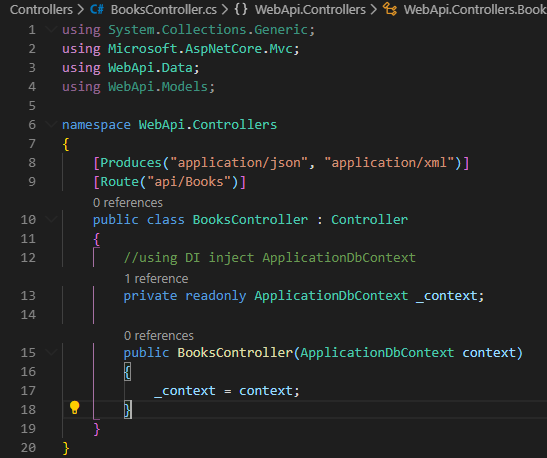
Create a new **Controller** in the **ControllerFolder**. Name it **BooksController.cs**.

It will extend Controller class.

## Attributes on Controller and Basic Setup

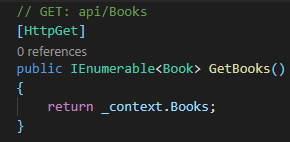
We’ll add Produces and Route attributes to the controller.

Also will linject context via DI.



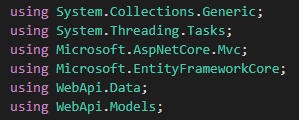
## Action: GetAllBooks

Take note of the attributes on top of the action methods.

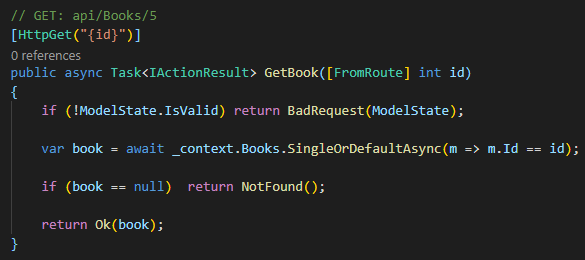


## Action: Picking a specific book

Update your using at the top



And then the action will look like following. Take a note of the attribute and the route being passed.

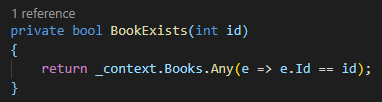


## Action: Updating a book

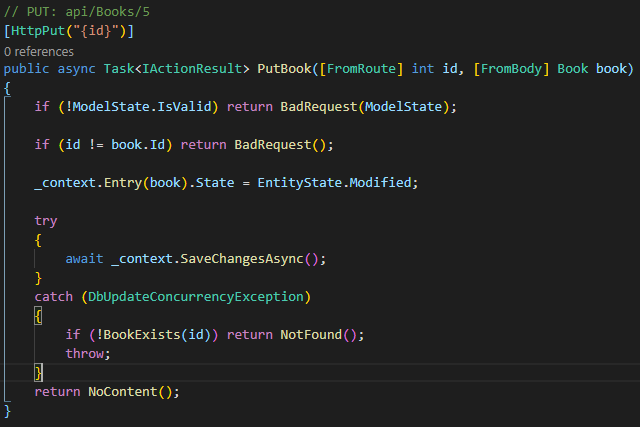
Adding using



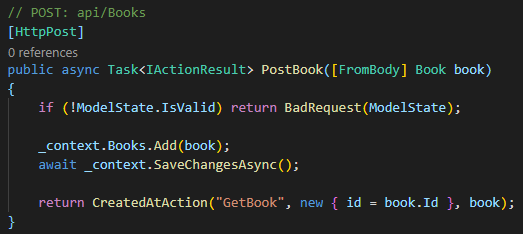
Add helper function to check the book



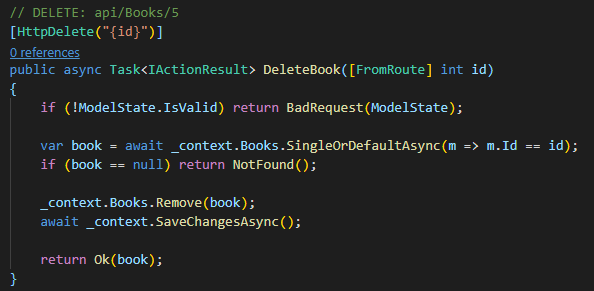
And the action to update the book. Take a look at the attribute and the arguments being passed in.



## Action: Adding a Book



## Action: Delete a Book



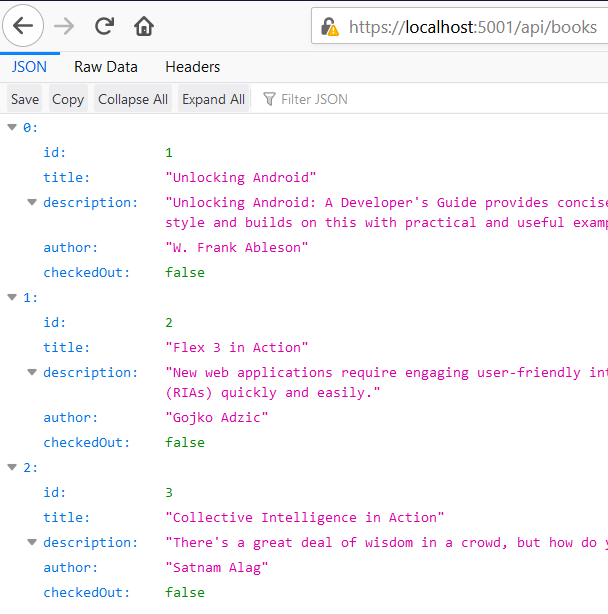
# Testing API

Run the application

## Getting all books

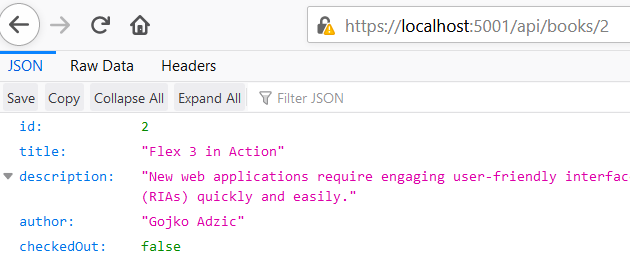
Change the URL to following. If all goes good then you should see 3 books that we have seeded before.

<https://localhost:5001/api/books>



## Getting book # 2

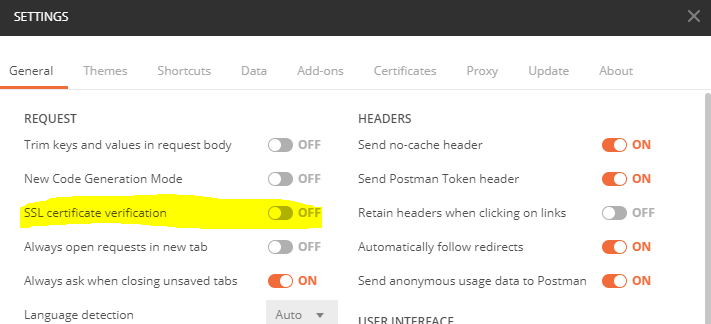
Change the url to <https://localhost:5001/api/books/2>



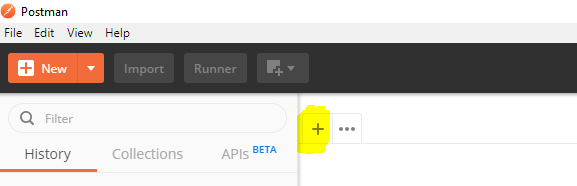
# Testing API With Postman

Download postman and install it: <https://www.getpostman.com/downloads/>

In the setting, turn off SSL verification

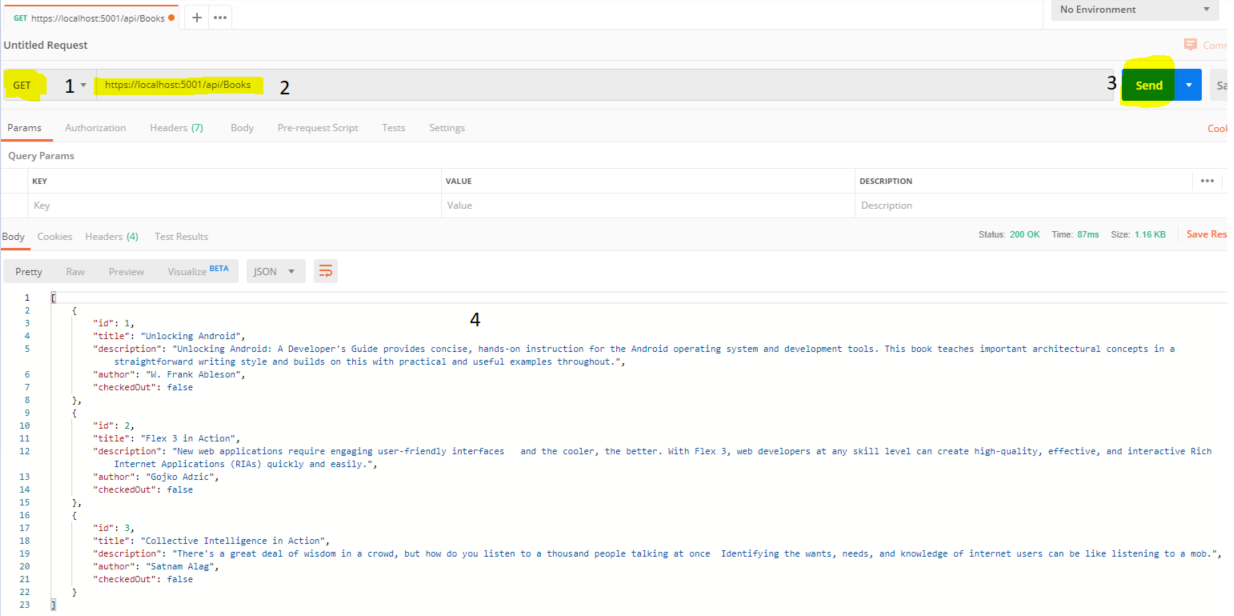


Afer install, you can skip the signin/signup step. Click on the + to open up a new tab



## Getting All books

1. Change the request to Get
2. Put in the URL
3. Click Send
4. View the results



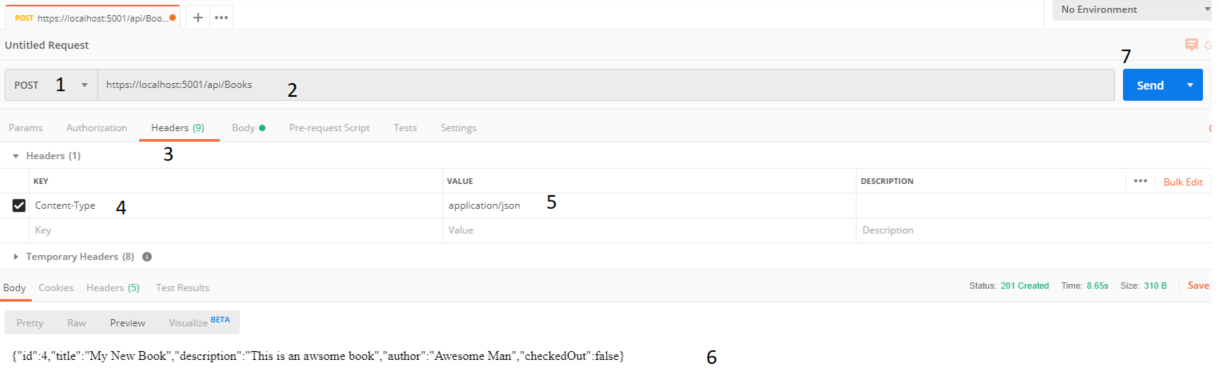
## Viewing Book 3

1. Change the request to Get
2. Put in the URL
3. Click Send
4. View the results

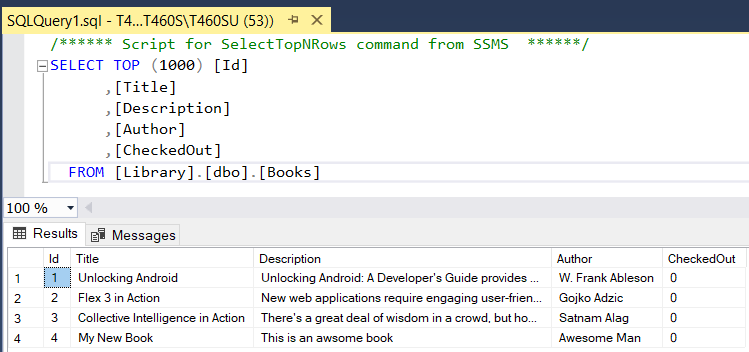


## Inserting A Book

1. Change the request to Post
2. 2. Put in the URL
3. Click Headers
4. Select content type
5. Provide the value for the Content type as application/json
6. Provide the pay load as json. Don’t put the Id element. After the post, a record will be pulled from the DB.
7. Click the Send button, on success you’ll see the new book with ID 5 in the response section and Status=201 Created.



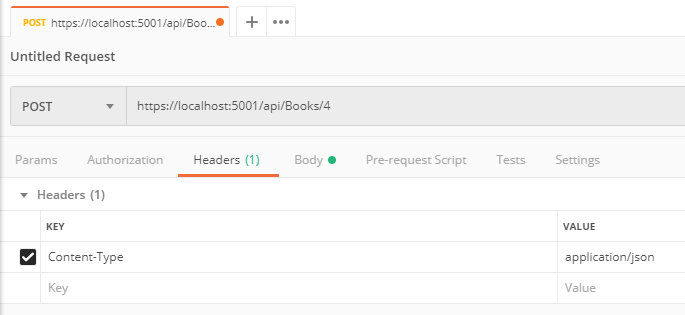
Go to the database and double check the item



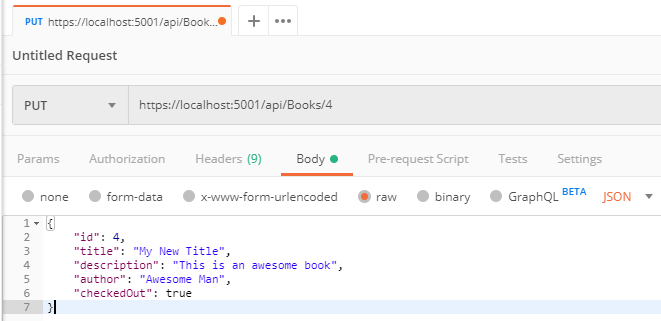
## Updating a Book

Lets update book 4.

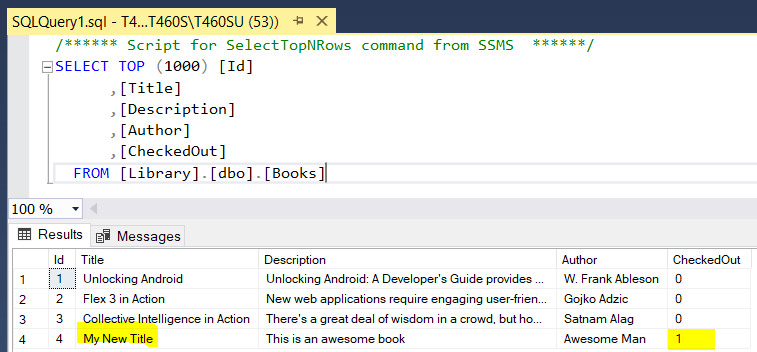
1. Change the action to POST.
2. Go to Headers tab and put the content type as application/json.



1. Go to the Body tab

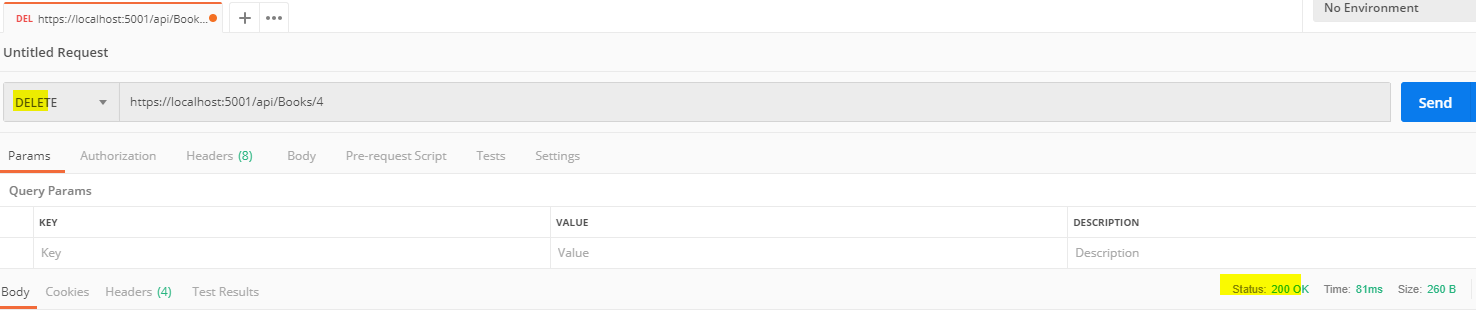


1. Click the send and then either look at the Database directly or do a new Get to view the book # 4. You’ll see your changes.



## Deleting a Book

Lets delete book#4 that we created earlier. Do a delete request and add 4 to the URL for the ID and then click Send.



You should see Status: 200 Ok

Also, query the database and there should be no book with id = 4.

