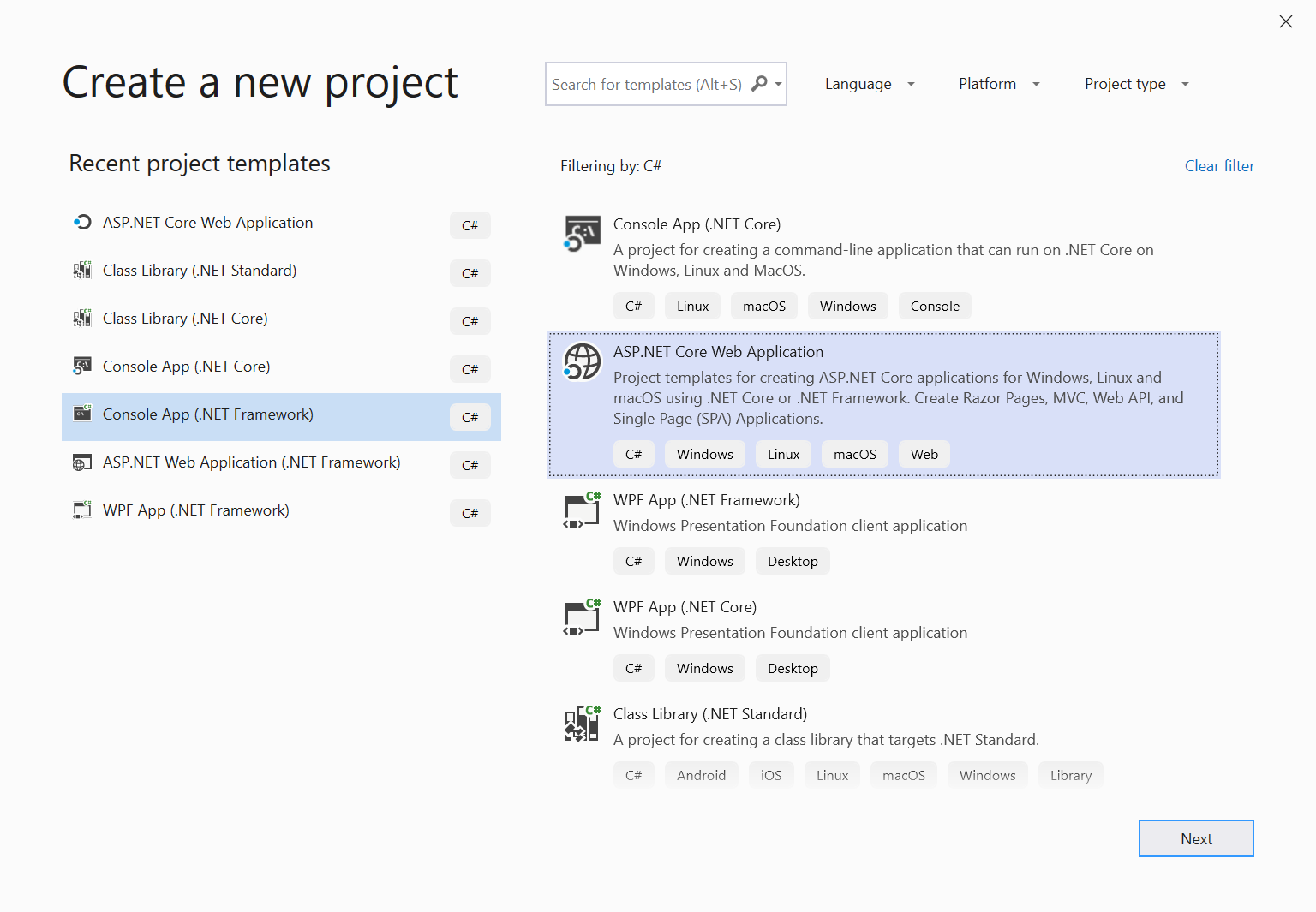
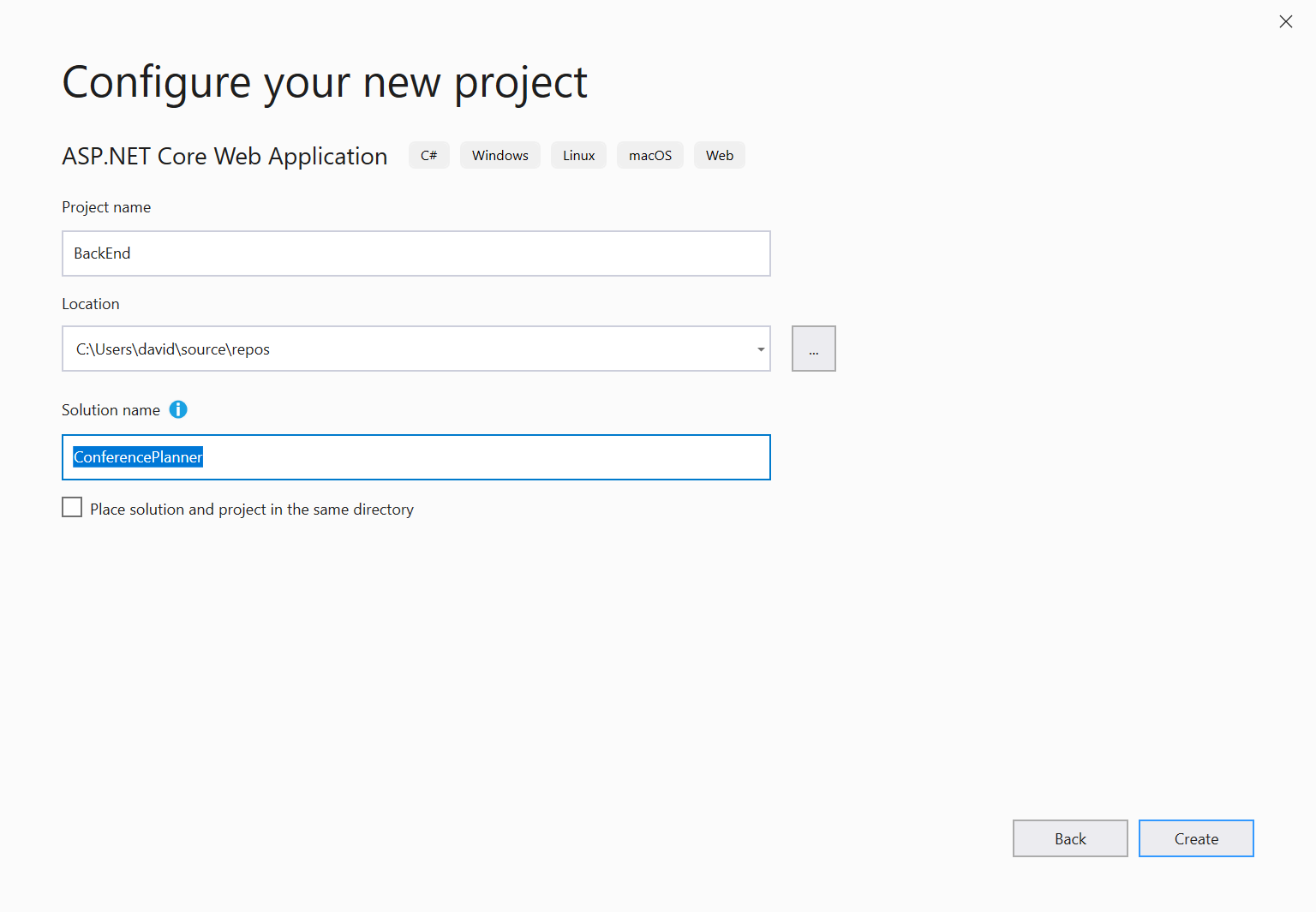
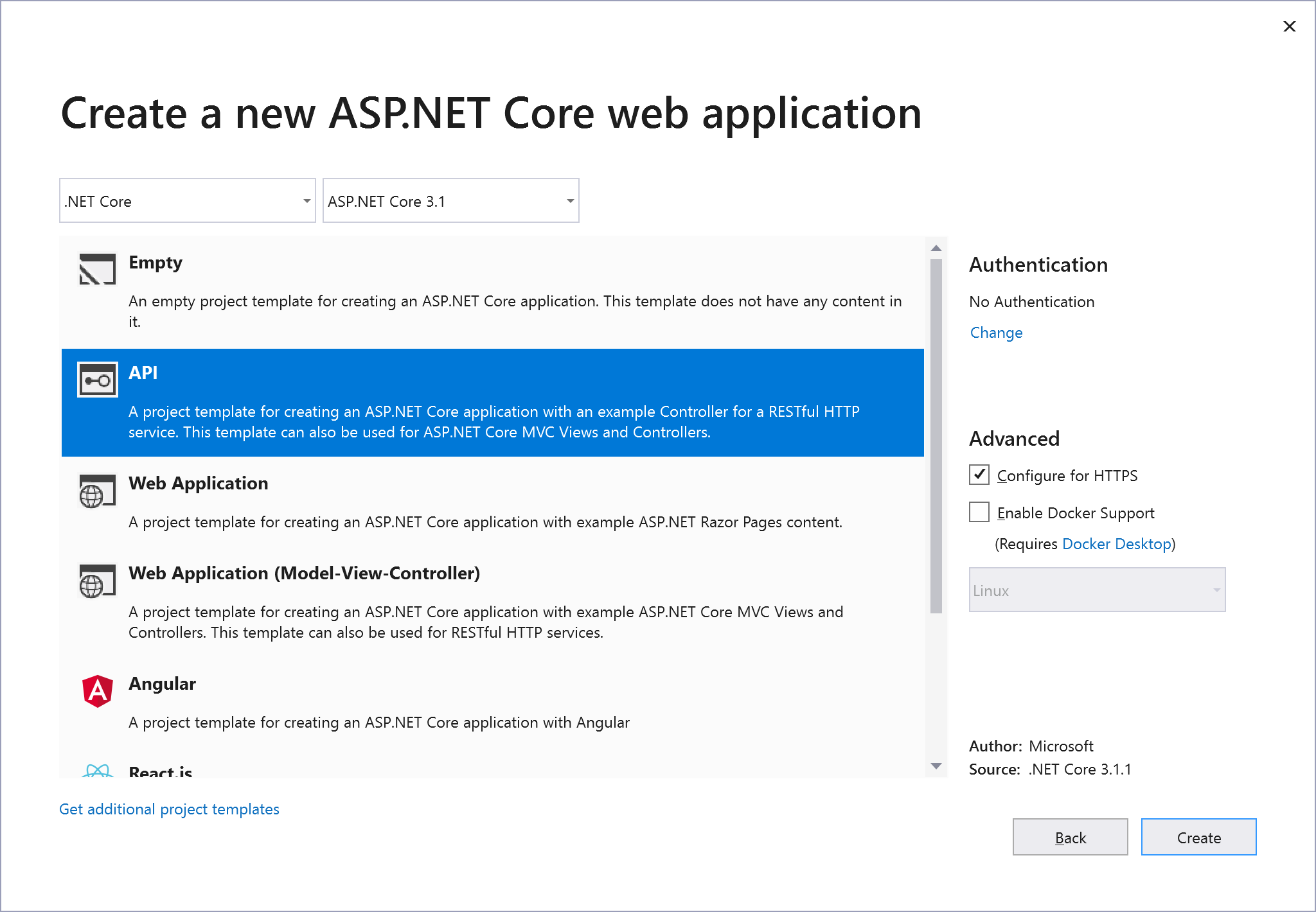
**Web API LAB/Exercise using Swagger**

1. Create and add a new project named BackEnd and name the solution ConferencePlanner using File / New / ASP.NET Core Web Application. Select the Web API template, No Auth, no Docker support. TestApi[](../_blank) [](../_blank) [](../_blank)
2. **Add a new Models folder to the root of the application.**
3. **Add a new Speaker class using the following code:**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace BackEnd.Models

{

public class Speaker

{

public int Id { get; set; }

[Required]

[StringLength(200)]

public string Name { get; set; }

[StringLength(4000)]

public string Bio { get; set; }

[StringLength(1000)]

public virtual string WebSite { get; set; }

}

}

1. **Add a reference to the NuGet package**

 Microsoft.EntityFrameworkCore.SqlServer version 3.1.1.

1. **Add a reference to the NuGet package**

 Microsoft.EntityFrameworkCore.Sqlite version 3.1.1.

1. Next we'll create a new Entity Framework DbContext. Create a new ApplicationDbContext class in the Models folder using the following code:

using Microsoft.EntityFrameworkCore;

namespace BackEnd.Models

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Speaker> Speakers { get; set; }

}

}

1. Create an SQL database in the cloud(Azure) and add the connection string to the appsettings.json file.

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-BackEnd-931E56BD-86CB-4A96-BD99-2C6A6ABB0829;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Logging": {

"LogLevel": {

"Default": "Warning",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

## Register the DB Context Service

1. Add the following code to the top of the ConfigureServices() method in Startup.cs:

services.AddDbContext<ApplicationDbContext>(options =>

{

if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))

{

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));

}

else

{

options.UseSqlite("Data Source=conferences.db");

}

});

This code registers the ApplicationDbContext service so it can be injected into controllers. Additionally, it configures operating system specific database technologies and connection strings

## Configuring EF Migrations

1. **Add a reference to the NuGet package**  Microsoft.EntityFrameworkCore.Tools version 3.1.1.

### Visual Studio: Package Manager Console

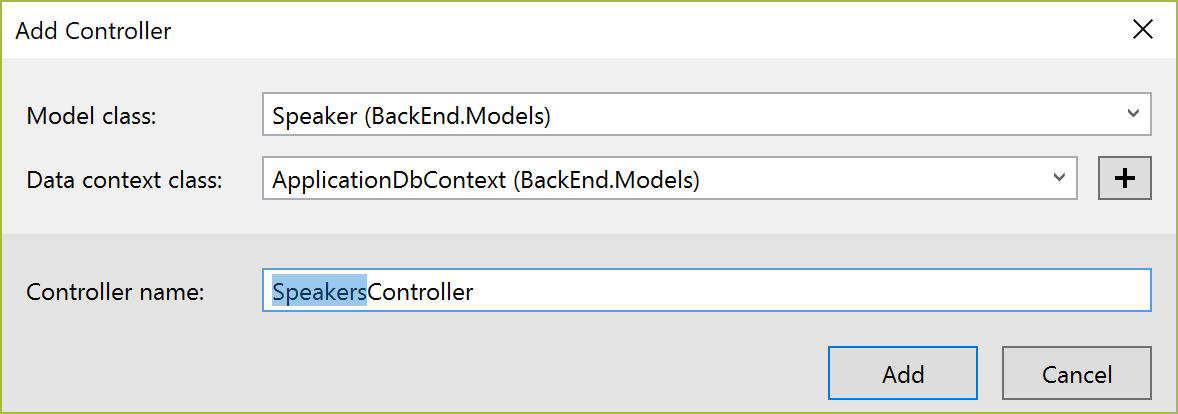
1. In Visual Studio, select the Tools -> NuGet Package Manager -> Package Manager Console
2. Run the following commands in the Package Manager Console
3. Add-Migration Initial

Update-Database

## A quick look at the Weather Forecast Controller

First, open the Controllers folder and take a quick look at the WeatherForecastController. You'll see a simple function that corresponds to the HTTP GET verb. You'll see the output of this controller in a bit, but first we'll build our own API controller for the Speakers model class.

## Scaffolding an API Controller

1. Right-click the Controllers folder and select Add/Controller. Select "API Controller with actions, using Entity Framework".
2. In the dialog, select the Speaker model for the Model Class, ApplicationDbContext for the "Data Context Class" and click the Add button. [](../_blank)

## Testing the API using the Swashbuckle

In this section, we'll be adding documentation to our API using the Swashbuckle NuGet package.

[Swashbuckle.AspNetCore](https://github.com/domaindrivendev/swashbuckle.aspnetcore) is an open source project for generating Swagger documents for Web APIs that are built with ASP.NET Core.

[Swagger](https://swagger.io/) is a machine readable representation of a RESTful API that enables support for interactive documentation, client SDK generation and discoverability.

Additional information on using Swashbuckle in ASP.NET Core is available in this tutorial: [ASP.NET Web API Help Pages using Swagger](https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger)

1. **Add a reference to the NuGet package** Swashbuckle.AspNetCore version 5.0.0.

This can be done from the command line using dotnet add package Swashbuckle.AspNetCore --version 5.0.0

1. **Add the Swashbuckle services in your ConfigureServices method**:

services.AddControllers();

services.AddSwaggerGen(options =>

options.SwaggerDoc("v1", new OpenApiInfo { Title = "Conference Planner API", Version = "v1" })

);

1. **Configure Swashbuckle by adding the following lines just before UseRouting in the Configure method in Startup.cs:**

app.UseSwagger();

app.UseSwaggerUI(options =>

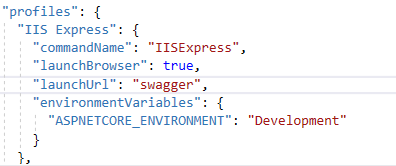
options.SwaggerEndpoint("/swagger/v1/swagger.json", "Conference Planner API v1")

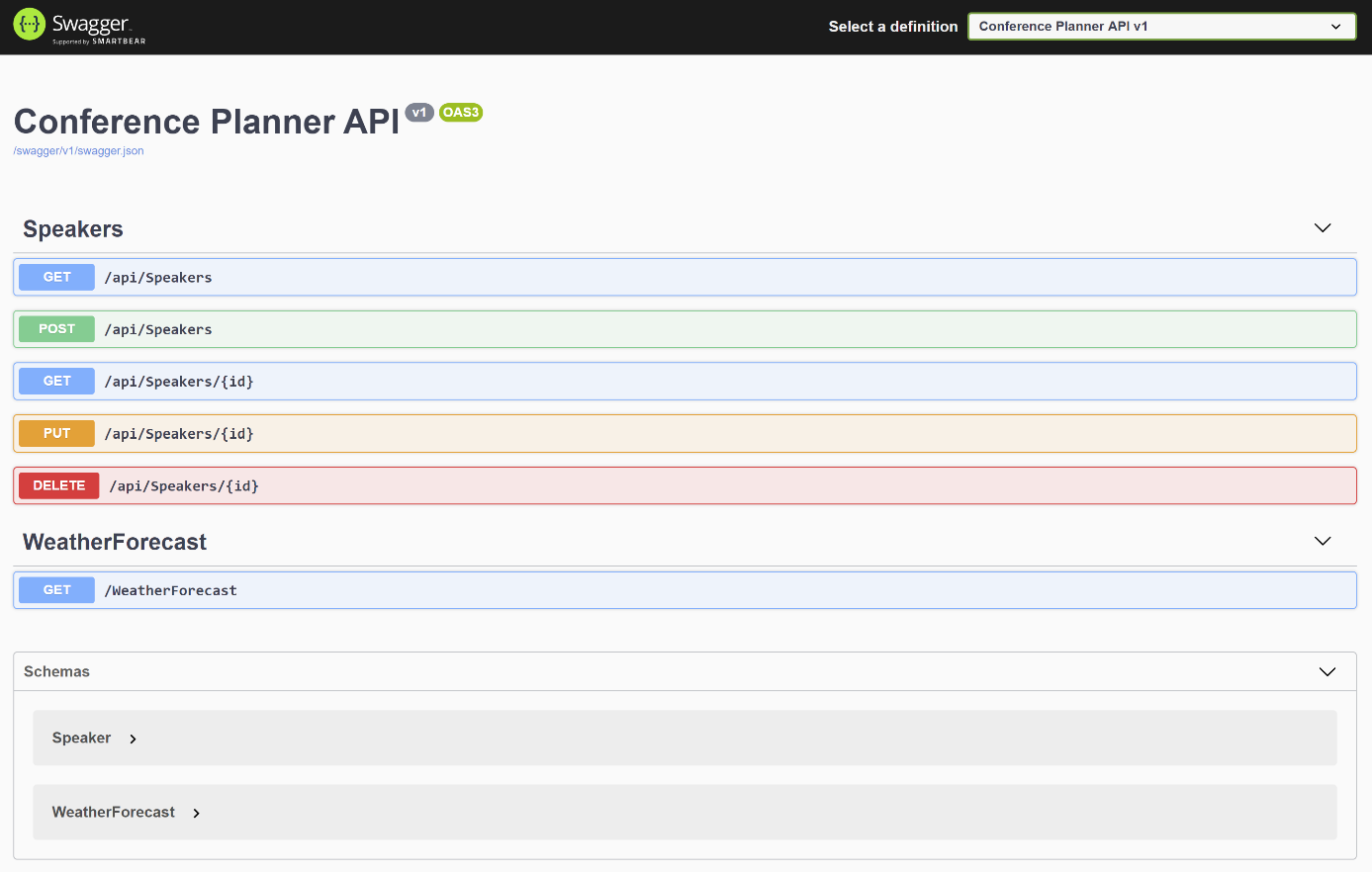
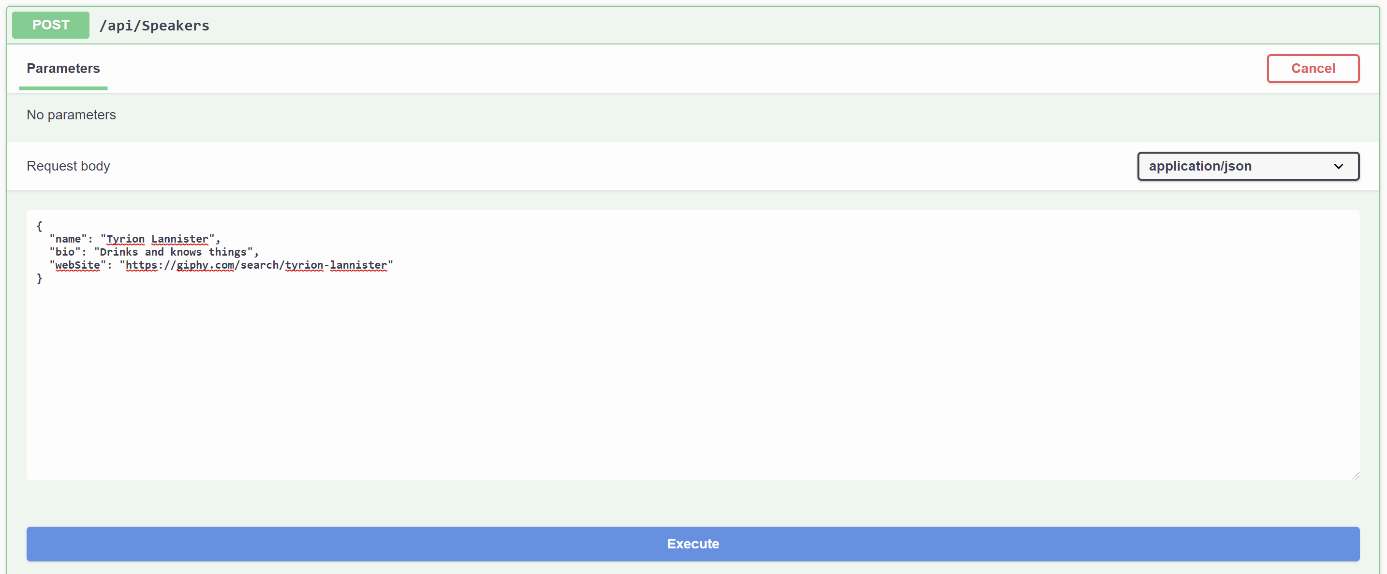
);

**Note: Due to how the middleware and pipeline are structured, you'll want to place this before the app.UseEndpoints() statement.**

1. Go to launchSettings.json and change the launchUrl to: “swagger”





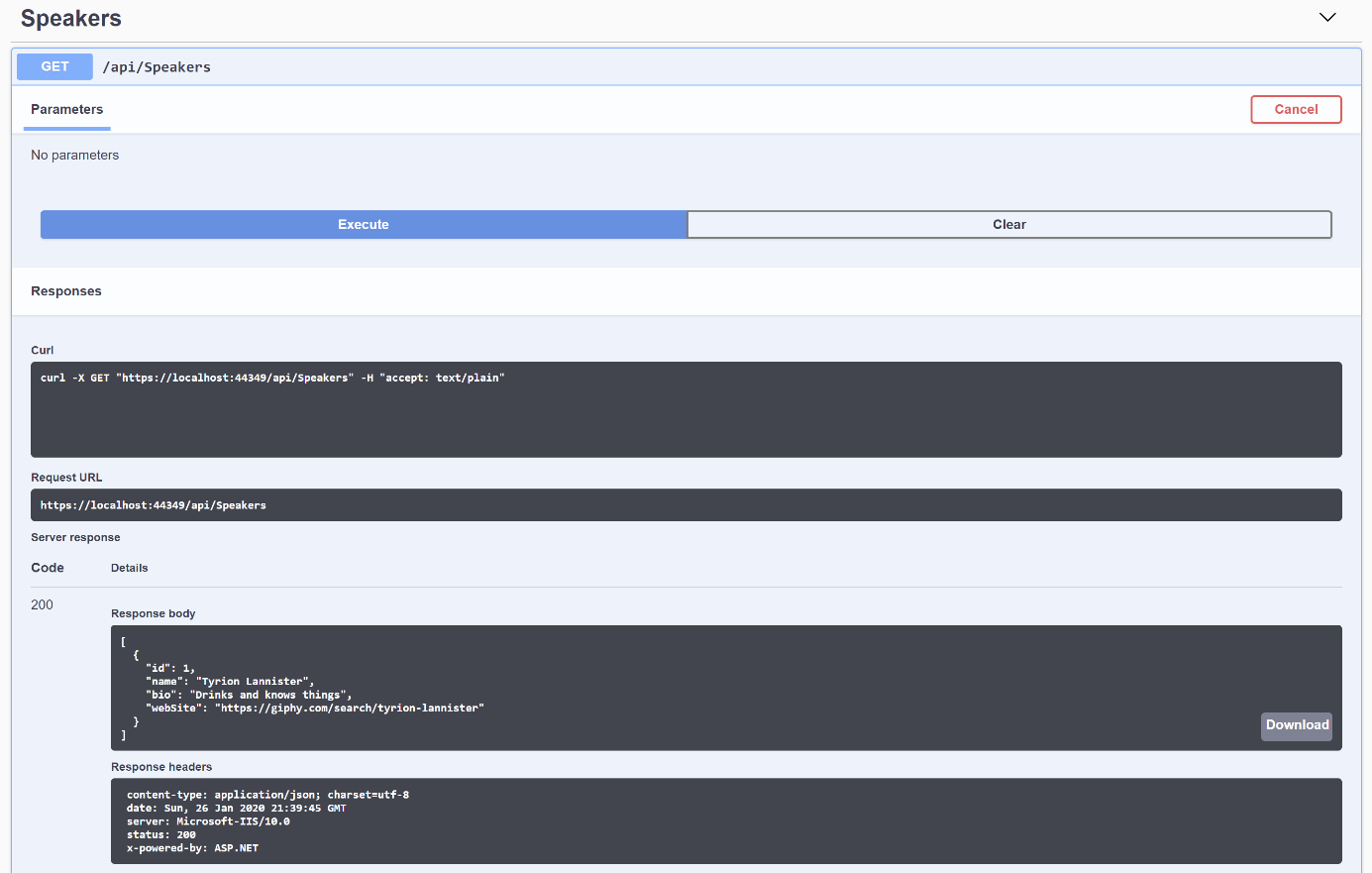
1. Run the application (F5 in Visual Studio).
2. Browse to the Swagger UI at http://localhost:<random\_port>/swagger. [](../_blank)
3. First, click on the GET button in WeatherForecast section. You'll see the values that were listed in the WeatherForecastController earlier.
4. In the Speakers section, click on the GET button. You'll see there are not speakers returned. Let's add one!
5. In the Speakers section, click on the POST button. Referencing the example on the right, fill in a speaker request. Leave the ID blank, that will be filled in by the database. [](../_blank)
6. {

"name": "Mel Gibson",

"bio": "Like to watch Netflix",

"webSite": "http://giphy.com/search/mel-gibson"

}

1. When you click the Execute button, you should see a success response from the server. Now, trying out the GET endpoint above should show your newly added speaker. [](../_blank)
2. Open up a second Visual studio and create a console project and paste the following code:

NOTE: Do not close Visual studio or the browser, leave it opened. Otherwise we cannot GET/POST our data from our console program to our API.

using System;

using System.IO;

using System.Net;

namespace Examples.System.Net

{

public class WebRequestGetExample

{

public static void Main()

{

// Create a request for the URL.

WebRequest request = WebRequest.Create(

"https://localhost:44315/api/Speakers/");

// If required by the server, set the credentials.

request.Credentials = CredentialCache.DefaultCredentials;

// Get the response.

WebResponse response = request.GetResponse();

// Display the status.

Console.WriteLine(((HttpWebResponse)response).StatusDescription);

// Get the stream containing content returned by the server.

// The using block ensures the stream is automatically closed.

using (Stream dataStream = response.GetResponseStream())

{

// Open the stream using a StreamReader for easy access.

StreamReader reader = new StreamReader(dataStream);

// Read the content.

string responseFromServer = reader.ReadToEnd();

// Display the content.

Console.WriteLine(responseFromServer);

}

// Close the response.

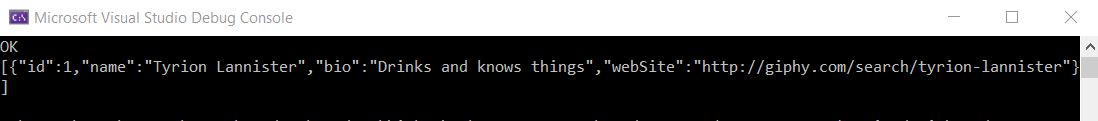
response.Close();

}

}

}

1. You will now see the following response in the console.



1. Try to do a POST

using System;

using System.IO;

using System.Linq;

using System.Net;

using System.Text;

using Newtonsoft.Json;

namespace postdata

{

public class Program

{

static void Main(string[] args)

{

PostJson("https://localhost:44315/api/Speakers", new Speaker

{

name = "Will Smith",

Bio = "likes to drive cars",

webSite = "www.smith.com"

});

}

private static void PostJson(string uri, Speaker postParameters)

{

string postData = JsonConvert.SerializeObject(postParameters);

byte[] bytes = Encoding.UTF8.GetBytes(postData);

var httpWebRequest = (HttpWebRequest)WebRequest.Create(uri);

httpWebRequest.Method = "POST";

httpWebRequest.ContentLength = bytes.Length;

httpWebRequest.ContentType = "application/json";

using (Stream requestStream = httpWebRequest.GetRequestStream())

{

requestStream.Write(bytes, 0, bytes.Count());

}

var httpWebResponse = (HttpWebResponse)httpWebRequest.GetResponse();

Console.WriteLine(httpWebResponse);

}

}

public class Speaker

{

public string name { get; set; }

public string Bio { get; set; }

public string webSite { get; set; }

}

}

1. Congratulations! You have now made your own REST Api and you have made a program to GET data and to POST data.
2. That program doesn’t look so good. Try to challenge yourself and represent it in MVC app instead of the console. =)