

Consuming a RESTful API with C#

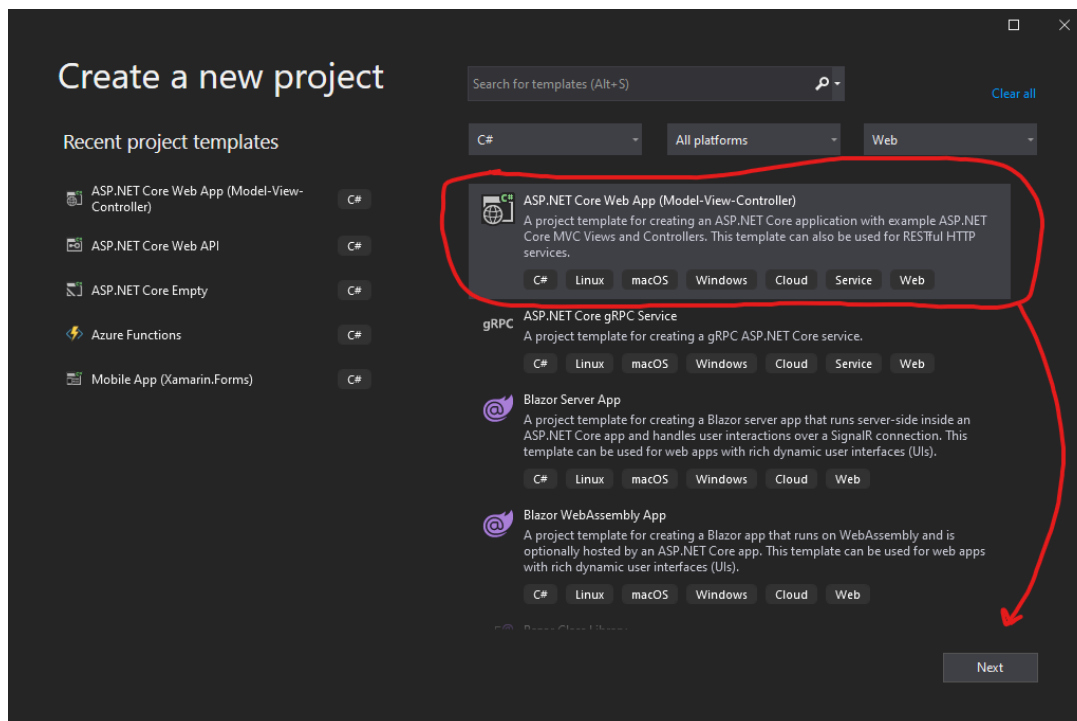
Asp.Net Core

In this simple project I will be accomplishing 3 tasks:

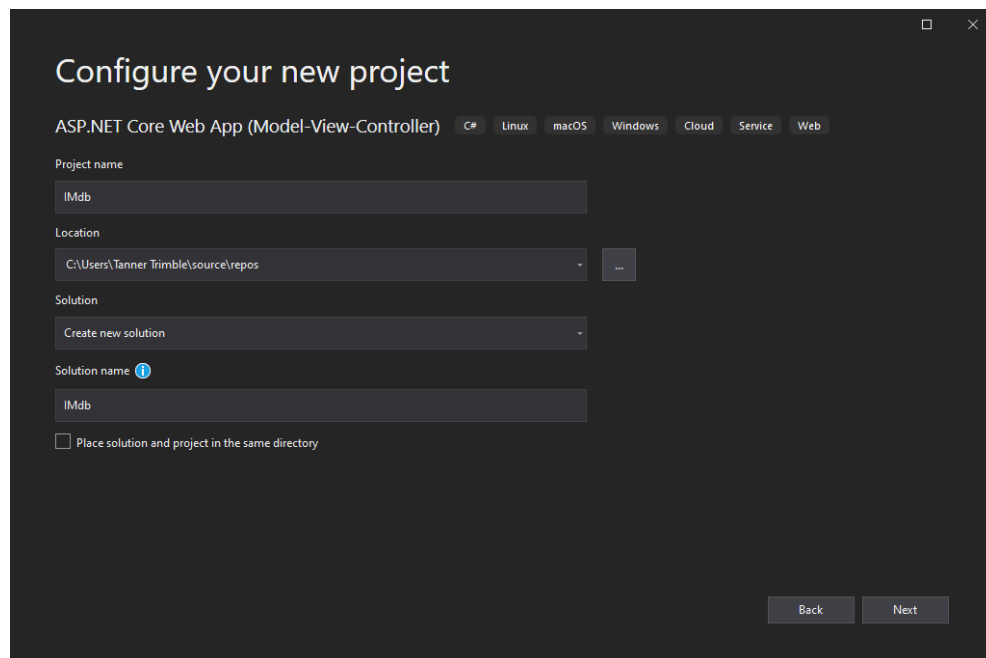
1. Using Http requests to send requests and receive responses from a free public API and displaying the information I receive back.
2. Styling my UI so the information I'm displaying is clean and easy for the user to process using Bootstrap.
3. Implementing Serilog logging and storing those logs to a json file.

I begin by first deciding which API I want to work with. There are many public API's to choose from but the one I decided to use for this project, due to it being easily accessible and it's utility, was the [Movie Database \(IMDB Alternative\)](#) API.

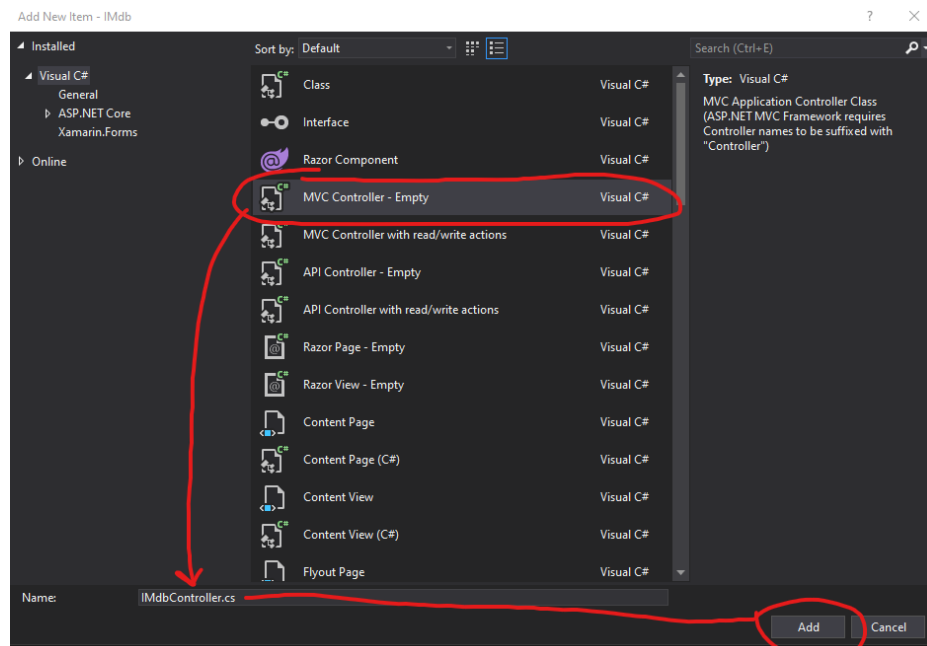
Now that I have the API I will create a template MVC .Net Application:



Give it a name and set its Location Path:

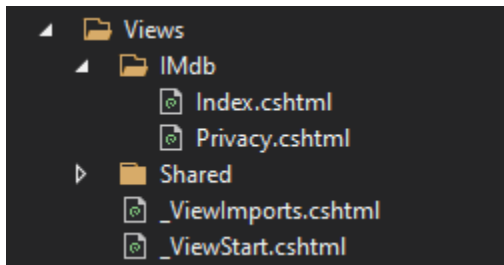


Now that I have my default application created I'll start by adding a new controller to the Controllers folder by right clicking the folder → Add → controller→ and name it IMdbController.cs:



Inside IMdbController.cs I will create a public asynchronous `Task<ActionResult>` method `GetMovies` and have it accept a parameter `searchParameters` as a string. The controller will also need to have a public `ActionResult` `Index` method that returns a `View()`. This will render the `Index.cshtml` page.

Now, I'm going to make some changes within the default scaffolding. First, for namesake, inside the Views folder I want to change the folder name where Index.cshtml is being stored from Home to IMdb.



Now, because of this change I will have an Index.cshtml not found error when I try to start up the application. This is because of this code here in the startup.cs file:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

This is setting up the default controller and the action to be called by that controller at startup. This is telling the application to go to the HomeController and call the method Index when it does this Index method will return a View(). This tells the application to go to our Views and render the default .cshtml file. This will be the .cshtml file that shares the name of the method that is returning View(), in this case, Index.cshtml that relates to the HomeController. This relation is distinguished by the folder in which the Index.cshtml file is found. HomeController will search for a folder called Home and if Index.cshtml isn't found, it will search for Index.cshtml within the Views folder but it will not try to access another folder. So I can solve this by either moving Index.cshtml into the Views folder, or I can take the approach of changing this default pattern, which is what I will do.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=IMdb}/{action=Index}/{id?}");
});
```

This now tells the application at startup to go to the IMdb controller and call the Index method. The Index method returns View(). This will send the application to the IMdb folder inside the Views folder, where it will find Index.cshtml and will render it to the browser.

Now that I have my application set up to this point I'm ready to start working with the API. The [Movie Database \(IMDB Alternative\)](#) public API provides very useful documentation for getting started and it is quite simple. Since it takes in movie searches and returns the results related to that search it has only two endpoints:

GET By Search
GET By ID or Title

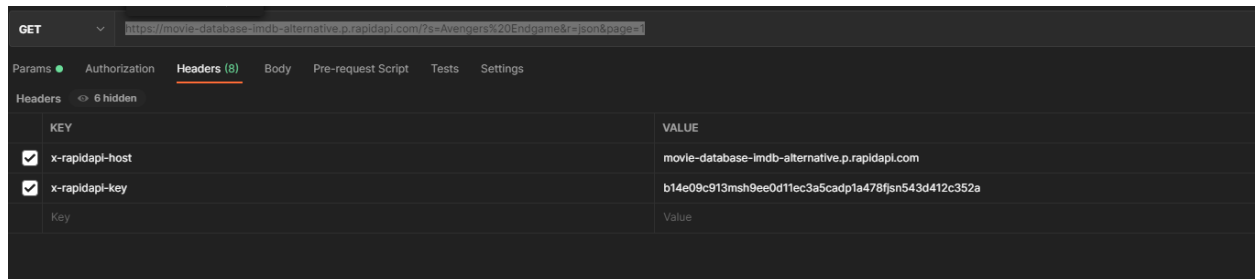
Since I don't know any IMdb movie IDs within the scope of this application I will be working only with the By Search endpoint. Within the documentation provided, I find some code for properly creating a HttpClient object sending a request to the API and then receiving the response:

```
(C#) HttpClient ▾ Copy Code
var client = new HttpClient();
var request = new HttpRequestMessage
{
    Method = HttpMethod.Get,
    RequestUri = new Uri("https://movie-database-imdb-alternative.p.rapidapi.com/?s=Avengers%20Endgame&r=json&page=1"),
    Headers =
    {
        { "x-rapidapi-host", "movie-database-imdb-alternative.p.rapidapi.com" },
        { "x-rapidapi-key", "b14e09c913msh9ee0d11ec3a5cadp1a478fjsn543d412c352a" },
    },
};
using (var response = await client.SendAsync(request))
{
    response.EnsureSuccessStatusCode();
    var body = await response.Content.ReadAsStringAsync();
    Console.WriteLine(body);
}
```

This is great for getting started but not everything I need for the needs of this application. I need to be able to store this information in variables that I can then pass to and render on my Index.cshtml page. To do this I'm going to need to do 3 things:

1. Create class objects that resemble the json objects I'm going to be receiving from the API
2. Deserialize the json response so it can be stored in my class object.
3. Pass those class objects to the View's Model.
4. Access and display the data I passed to the View's Model

First, I need to know how to design my class objects so that the json data can be properly deserialized. To do this my objects have to have the same attributes as the json objects. To figure this out I need to see how a response from a request looks. A good way to do this is by using an API tool like [Postman](#). Simply put the given URI into the request bar, make sure it is set to GET, and add the two header keys and values in the header section.



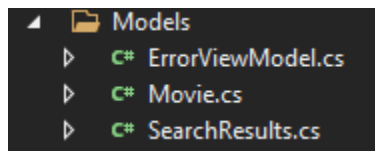
Once I send the request it will return this:

```
"Search": [
  {
    "Title": "Avengers: Endgame",
    "Year": "2019",
    "imdbID": "tt4154796",
    "Type": "movie",
    "Poster": "https://m.media-amazon.com/images/M/MV5BMTc5MDE2ODcwNV5BMl5BanBnXkFtZTgwMzI2NzQ2NzM0._V1_SX300.jpg"
  },
  {
    "Title": "Avengers: Endgame and the Latest Captain Marvel Outrage!!",
    "Year": "2019",
    "imdbID": "tt10025738",
    "Type": "movie",
    "Poster": "https://m.media-amazon.com/images/M/MV5BZjg2ZTM3OTgtY2ExMS00OGM4LTg3NDktNjQ0MjJiZDZlMGFkXkE5XkFqcGdeQXVyMDY3OTcyOQ@@._V1_SX300.jpg"
  },
  {
    "Title": "Marvel Studios' Avengers: Endgame LIVE Red Carpet World Premiere",
    "Year": "2019",
    "imdbID": "tt10240638",
    "Type": "movie",
    "Poster": "https://m.media-amazon.com/images/M/MV5BNTBjZDgwZTYtMjdmYjY0ZmUyLTk4NTUtMzZjZmExODQ3ZmY4XkE5XkFqcGdeQXVyMjkzMDgyNTg0._V1_SX300.jpg"
  },
  {
    "Title": "Avengers Endgame: the Butt Plan",
    "Year": "2019",
    "imdbID": "tt10399328",
    "Type": "movie",
    "Poster": "https://m.media-amazon.com/images/M/MV5BNTQ1OWQzODktMTY3Zi00OTQxLWExOTYtZTNjZjY5ZTY4M2UyXkE5XkFqcGdeQXVyMTAzMzk0NjAy._V1_SX300.jpg"
  }
],
"totalResults": "4",
"Response": "True"
```

I now know that my class needs to look like this:

- Search
 - Title
 - Year
 - imdbID
 - Type
 - Poster
- totalResults
- Response

I will create these classes in my Models folder and it will look like this:



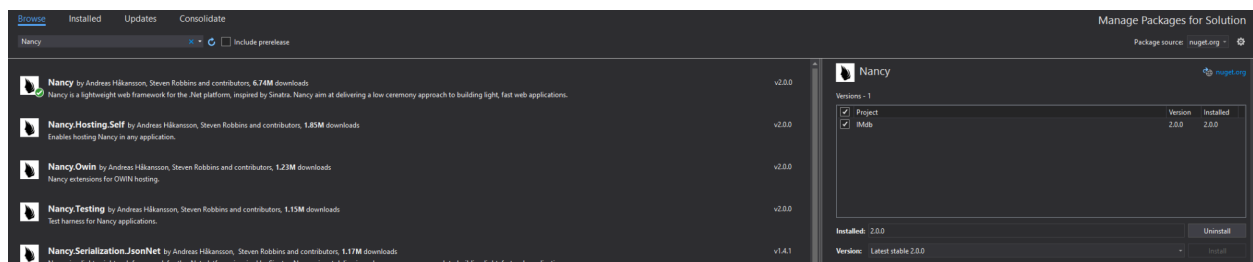
```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace IMdb.Models
{
    4 references
    public class SearchResults
    {
        1 reference
        public IEnumerable<Movie> Search { get; set; }
        1 reference
        public int totalResults { get; set; }
        1 reference
        public bool Response { get; set; }
    }
}
```

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace IMdb.Models
{
    1 reference
    public class Movie
    {
        1 reference
        public string Title { get; set; }
        1 reference
        public string Year { get; set; }
        0 references
        public string imdbID { get; set; }
        0 references
        public string Type { get; set; }
        1 reference
        public string Poster { get; set; }
    }
}
```

I have to make sure that the attribute names match up exactly with the json data in order to avoid any issues with the deserializer. Now that I have my classes set up I'm ready to move on to step 2. To deserialize the response I will need to add a nuget package called Nancy. I'll right click on the solution → Manage NuGet Packages for Solution → Search "Nancy" → install first result:



I'll add the using Nancy.json, and using IMdb.Models to the top of my IMdbController.cs file. Then inside the GetMovie method I made in the beginning I will add the following code:

```
References
public async Task<IActionResult> GetMovie(string searchParameter)
{
    var client = new HttpClient();
    var request = new HttpRequestMessage
    {
        Method = HttpMethod.Get,
        RequestUri = new Uri($"https://movie-database-imdb-alternative.p.rapidapi.com/?s={searchParameter}&r=json&page=1"),
        Headers =
        {
            { "x-rapidapi-host", "movie-database-imdb-alternative.p.rapidapi.com" },
            { "x-rapidapi-key", "b14e09c913msh9ee0d11ec3a5cadpla478fjsn543d412c352a" },
        },
    };

    try
    {
        var response = await client.SendAsync(request);
        response.EnsureSuccessStatusCode();

        var body = await response.Content.ReadAsStringAsync();

        JavaScriptSerializer serializer = new();

        var results = serializer.Deserialize<SearchResults>(body);

        return View("Index", results);
    }
    catch (Exception e)
    {
    }

    return View("Index");
}
```

This closely resembles the default code I was given but now once I have the response body I will create a JavaScriptSerializer object provided by the Nancy.json library I added and deserialize the response body into an object of the SearchResults class I created. This brings us to step 3. I will return the View("Index", results). What this does is tell the application to render the .cshtml file with the name I provided which will be Index.cshtml and pass along data in results to the view's model.

Up to this point I have created a method that when called will send a request to and receive a response from the API. This response will be deserialized and stored in an object of my SearchResults class and that object will be passed to the view so that we can access its data and render it to the browser. This brings me to step 4.

I'll need to go into my Index.cshtml file and start by deleting most of the default html. Then I'm going to be changing the title and adding a form to take input from the user and submit it to the IMdb controller GetMovie method. I will also be adding some simple bootstrap css classes to keep it looking nice. Once I'm done my code looks like this:

```

@model
ViewData["Title"] = "Home Page";

<div class="container">
  <h1 class="display-4 text-center">Welcome to your Movie Database</h1>
  <form method="post" asp-controller="IMdb" asp-action="GetMovie">
    <div class="form-group">
      <label class="">Search for a Movie:</label>
      <input type="text" name="searchParameter" class="form-control" placeholder="ex: The Last Samurai" required/>
    </div>
    <button type="submit" class="btn btn-primary" style="margin-bottom: 1rem;" value="Submit">Search</button>
  </form>

```

This gives me a text input that I can input my search parameter into and once I press the submit button it will store the input into the searchParameters parameter in my GetMovie method and insert it into our request URI.

Next I'm going to take that data and display it. I can do this building on this fundamental line of code:

```
@model IMdb.Models.SearchResults
```

This model will expect to store a SearchResults class object that it will receive from the View("Index", results) from the GetMovie method. I will create a foreach loop that will go through each item stored in the model.Search list and for each of those items display its contents. I will stylize this list once again using some bootstrap so it appears clean and easy to digest. When I'm done my code looks like this:

```

@model IMdb.Models.SearchResults
ViewData["Title"] = "Home Page";

<div class="container">
  <h1 class="display-4 text-center">Welcome to your Movie Database</h1>
  <form method="post" asp-controller="IMdb" asp-action="GetMovie">
    <div class="form-group">
      <label class="">Search for a Movie:</label>
      <input type="text" name="searchParameter" class="form-control" placeholder="ex: The Last Samurai" required/>
    </div>
    <button type="submit" class="btn btn-primary" style="margin-bottom: 1rem;" value="Submit">Search</button>
  </form>

  <ul class="list-group text-left">
    <@if (Model != null)>
    {
      <foreach (var movie in Model.Search)>
      {
        <li class="list-group-item">
          <div class="row">
            <div class="col">
              
              <h3><b>@movie.Title</b></h3>
              <h6>@movie.Year</h6>
            </div>
          </div>
        </li>
      }
    }
  </ul>
</div>

```


When I run my application this is what you will see:

Welcome to your Movie Database

Search for a Movie:

ex: The Last Samurai

Search


If I search for the movie Avatar this will be displayed:

Welcome to your Movie Database

Search for a Movie:

ex: The Last Samurai

Search



Avatar

2009

... other items follow but for brevity I'm only showing the first result.

To finish this application off I'm going to add some logging using Serilog. To do this I need to start by installing the Serilog, Serilog.Settings.Configuration and Serilog.Sinks.File NuGet packages. I will follow the same process I did to install the Nancy NuGet package. In my program.cs I include the following code:

```
var configuration = new ConfigurationBuilder()
    .AddJsonFile("appsettings.json")
    .Build();

Log.Logger = new LoggerConfiguration()
    .ReadFrom.Configuration(configuration)
    .CreateLogger();
```

This will build a configuration based on my appsettings.json file and configure the global Log.Logger from Serilog based on that configuration. Inside my appsettings.json I will include the following json:

```
{
  "Serilog": {
    "MinimumLevel": {
      "Default": "Debug",
      "Override": {
        "Microsoft": "Warning",
        "System": "Warning"
      }
    },
    "WriteTo": [
      {
        "Name": "File",
        "Args": {
          "path": "Logs/Log.json",
          "rollingInterval": "Day",
          "formatter": "Serilog.Formatting.Json.JsonFormatter"
        }
      }
    ]
  },
  "AllowedHosts": "*"
}
```

This will set the logger to write to a file in a Logs folder named Log.json and will replace the logs every day. The formatter will format the logged data according to the Serilog.Formatting.Json.JsonFormatter design. I will then added logging to my code and when I'm done my code will look like this:

```
0 references
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        var configuration = new ConfigurationBuilder()
            .AddJsonFile("appsettings.json")
            .Build();

        Log.Logger = new LoggerConfiguration()
            .ReadFrom.Configuration(configuration)
            .CreateLogger();

        try
        {
            Log.Logger.Warning($"Application Starting up....");
            CreateHostBuilder(args).Build().Run();
        }
        catch (Exception e)
        {
            Log.Logger.Fatal($"{e}: {e.Message}");
        }
        finally
        {
            Log.CloseAndFlush();
        }
    }
}
```

```

0 references
public async Task<IActionResult> GetMovie(string searchParameter)
{
    Log.Logger.Debug($"Searching for the movie {searchParameter}");

    var client = new HttpClient();
    var request = new HttpRequestMessage
    {
        Method = HttpMethod.Get,
        RequestUri = new Uri($"https://movie-database-imdb-alternative.p.rapidapi.com/?s={searchParameter}&r=json&page=1"),
        Headers =
        {
            { "x-rapidapi-host", "movie-database-imdb-alternative.p.rapidapi.com" },
            { "x-rapidapi-key", "b14e09c913msh9ee0d11ec3a5cadp1a478fjsn543d412c352a" },
        },
    };

    Log.Logger.Information($"Http request generated: Method: {request.Method} URI: {request.RequestUri}");

    try
    {
        var response = await client.SendAsync(request);
        response.EnsureSuccessStatusCode();

        var body = await response.Content.ReadAsStringAsync();

        JavaScriptSerializer serializer = new();

        var results = serializer.Deserialize<SearchResults>(body);

        Log.Logger.Debug($"Response Received: {results.Response} | Number of Results: {results.totalResults}");

        return View("Index", results);
    }
    catch (Exception e)
    {
        Log.Logger.Error($"Failed to receive a successful response. Exception message: {e.Message}");
    }

    return View("Index");
}

```

After running the application and doing some searches I will have the following logs:

```

{
  "Timestamp": "2021-10-03T17:47:35.6475519-06:00",
  "Level": "Warning",
  "MessageTemplate": "Application Starting up..."
}
{
  "Timestamp": "2021-10-03T17:48:02.5987634-06:00",
  "Level": "Debug",
  "MessageTemplate": "Searching for the movie Star Wars"
}
{
  "Timestamp": "2021-10-03T17:48:02.6068482-06:00",
  "Level": "Information",
  "MessageTemplate": "Http request generated: Method: GET URI: https://movie-database-imdb-alternative.p.rapidapi.com/?s=Star Wars&r=json&page=1"
}
{
  "Timestamp": "2021-10-03T17:48:03.4257075-06:00",
  "Level": "Debug",
  "MessageTemplate": "Response Received: True"
}
{
  "Timestamp": "2021-10-03T17:48:35.1898573-06:00",
  "Level": "Debug",
  "MessageTemplate": "Searching for the movie Rogue One"
}
{
  "Timestamp": "2021-10-03T17:48:35.1904763-06:00",
  "Level": "Information",
  "MessageTemplate": "Http request generated: Method: GET URI: https://movie-database-imdb-alternative.p.rapidapi.com/?s=Rogue One&r=json&page=1"
}
{
  "Timestamp": "2021-10-03T17:48:35.8532870-06:00",
  "Level": "Debug",
  "MessageTemplate": "Response Received: True"
}

```

I've accomplished what I set out to accomplish. I have an application that consumes a public API and displays the data it gets back, I have it stylized using Bootstrap, and I have logging using Serilog that is written to a log.json file.