Lab assignment 3
Due date: Friday, March 29
20 points

Part I (5 points)

How to Consume Third-party Web APIs in ASP.NET Core

Following the direction in the document below to create an ASP.NET MVC application to consume a third party API.

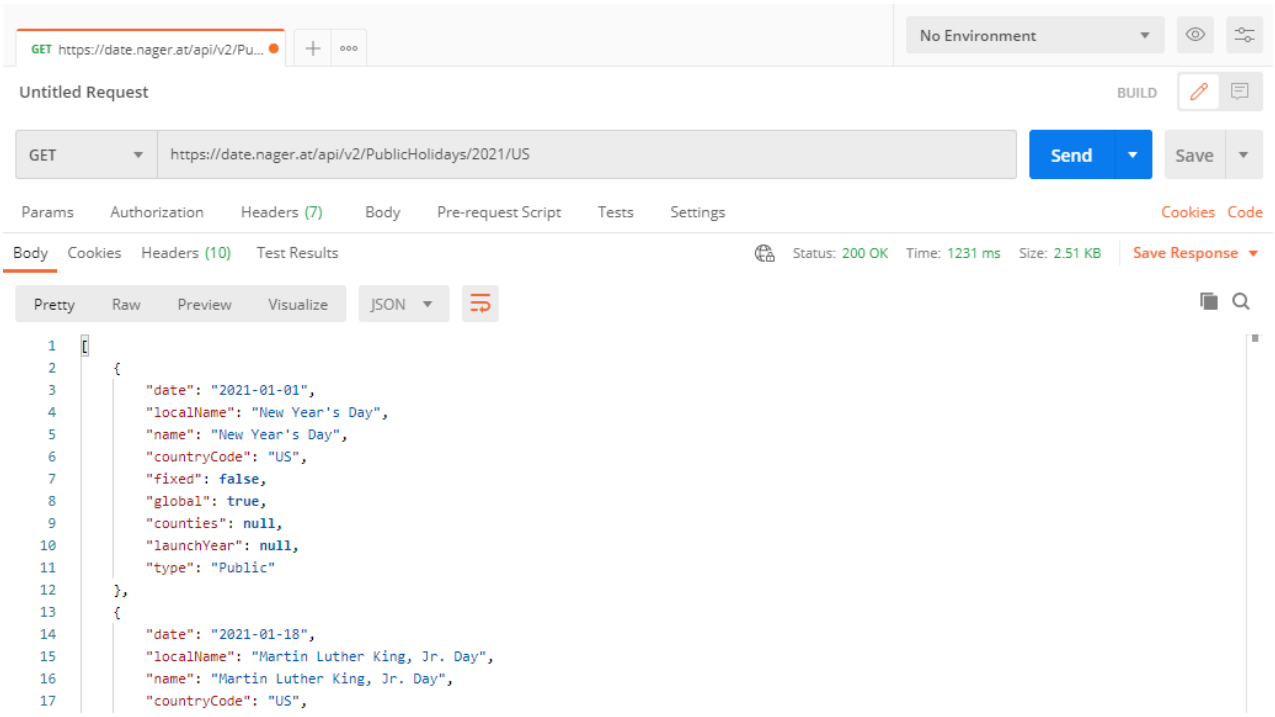
Overview of Third Party API

We will develop an application that will allow the user to input a country code and a year and then we will call a third party API to fetch the list of public holidays of that particular country in that particular year. The third-party API we will consume is called Nager.Date which is a worldwide public holidays API.

It is a very simple API and you can easily test this API in Postman or Swagger by entering the following URL.

https://date.nager.at/api/v2/PublicHolidays/2020/US

The response of this API is the list of public holidays in JSON format as shown below:



Understanding HttpClient Object

The most common and well knows class that allows us to consume third-party APIs in ASP.NET Core application is **HttpClient class. This class gives us the ability to send HTTP requests to third-party APIs and receive HTTP responses returned from those APIs. Every instance of HttpClient maintains its own connection pool that allows it to isolate its requests from requests executed by other instances of HttpClient. This class also acts as a base class for more specific HTTP clients**. For example, you can create FacebookHttpClient or TwitterHttpClient as child classes of base HttpClient and can communicate with Facebook and Twitter APIs using these specific HTTP clients.

It is recommended to create one instance of HttpClient and reuse it throughout the application lifetime. This is because instantiating a new instance of HttpClient for every request can easily exhaust the number of sockets available under heavy loads. This is mainly because when the HttpClient object is disposed of the underlying socket is not immediately released.

Using HttpClient in ASP.NET Core

We will create an application that will allow the user to view the list of public holidays in any country. Let's create an **ASP.NET Core MVC Web Application** and

create the following interface. This interface has just one method GetHolidays which has two parameters countryCode and year which we will receive from the user shortly.

```
public interface IHolidaysApiService
{
    Task<List<HolidayModel>> GetHolidays(string countryCode, int year);
}
```

The GetHolidays method above is returning a list of HolidayModel which is a model class that has the properties mapped with the response of Nager.Date API.

```
public class HolidayModel
{
    public string Name { get; set; }
    public string LocalName { get; set; }
    public DateTime? Date { get; set; }
    public string CountryCode { get; set; }
    public bool Global { get; set; }
}
```

Next, we need to implement a HolidaysApiService class that will implement the IHolidaysApiService declared above. I have declared a private and static HttpClient variable in the class and how it is defined in the static constructor of the class. It is the recommended way of creating HttpClient instances as mentioned on Microsoft official docs.

```
public class HolidaysApiService : IHolidaysApiService
{
    private static readonly HttpClient client;

    static HolidaysApiService()
    {
        client = new HttpClient()
        {
            BaseAddress = new Uri("https://date.nager.at")
        };
    }
}
```

Next we need to define GetHolidays method as shown below:

```
public async Task<List<HolidayModel>> GetHolidays(string countryCode, int
year)
{
    var url = string.Format("/api/v2/PublicHolidays/{0}/{1}", year,
countryCode);
    var result = new List<HolidayModel>();
    var response = await client.GetAsync(url);
    if (response.IsSuccessStatusCode)
    {
```

```
        var stringResponse = await response.Content.ReadAsStringAsync();

        result =
JsonSerializer.Deserialize<List<HolidayModel>>(stringResponse,
            new JsonSerializerOptions() { PropertyNamingPolicy =
JsonNamingPolicy.CamelCase });
    }
else
    {
        throw new HttpRequestException(response.ReasonPhrase);
    }

    return result;
}
```

Coding explanation:

- The first line is building the **Url** of Nager.Date API and using the **year** and **countryCode** parameters

```
var url = string.Format("/api/v2/PublicHolidays/{0}/{1}", year, countryCode);
```

- Next, we are making an API call using the **GetAsync** method that sends a GET request to the specified Uri as an asynchronous operation. The method returns **System.Net.Http.HttpResponseMessage** object that represents an HTTP response message including the status code and data.

```
var response = await client.GetAsync(url);
```

- Next, we are calling **ReadAsStringAsync** method that serializes the HTTP content to a string

```
var stringResponse = await response.Content.ReadAsStringAsync();
```

- Finally, we are using **JsonSerializer** to Deserialize the JSON response string into a List of **HolidayModel** objects.

```
result = JsonSerializer.Deserialize<List<HolidayModel>>(stringResponse,
    new JsonSerializerOptions() { PropertyNamingPolicy = JsonNamingPolicy.CamelCase })
```

That's all we need to consume a third part Public Holidays API. To use our **HolidaysApiService**, we need to first register our service in **Startup.cs** class (or in **Program.cs**)

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton<IHolidaysApiService, HolidaysApiService>();
var app = builder.Build();
```

Note: The **AddSingleton method** creates the service for the first request only. It is then reused for every subsequent request. Note that unlike AddScoped method here the services will be shared across every other client.

Next, we can inject our **HolidaysApiService** in our **HomeController** and call the **GetHolidays** method by passing the countryCode and year parameter we will receive inside the Index action method.

```
public class HomeController : Controller
{
    private readonly IHolidaysApiService _holidaysApiService;

    public HomeController(IHolidaysApiService holidaysApiService)
    {
        _holidaysApiService = holidaysApiService;
    }

    public async Task<IActionResult> Index(string countryCode, int year)
    {
        List<HolidayModel> holidays = new List<HolidayModel>();
        holidays = await _holidaysApiService.GetHolidays(countryCode,
year);

        return View(holidays);
    }
}
```

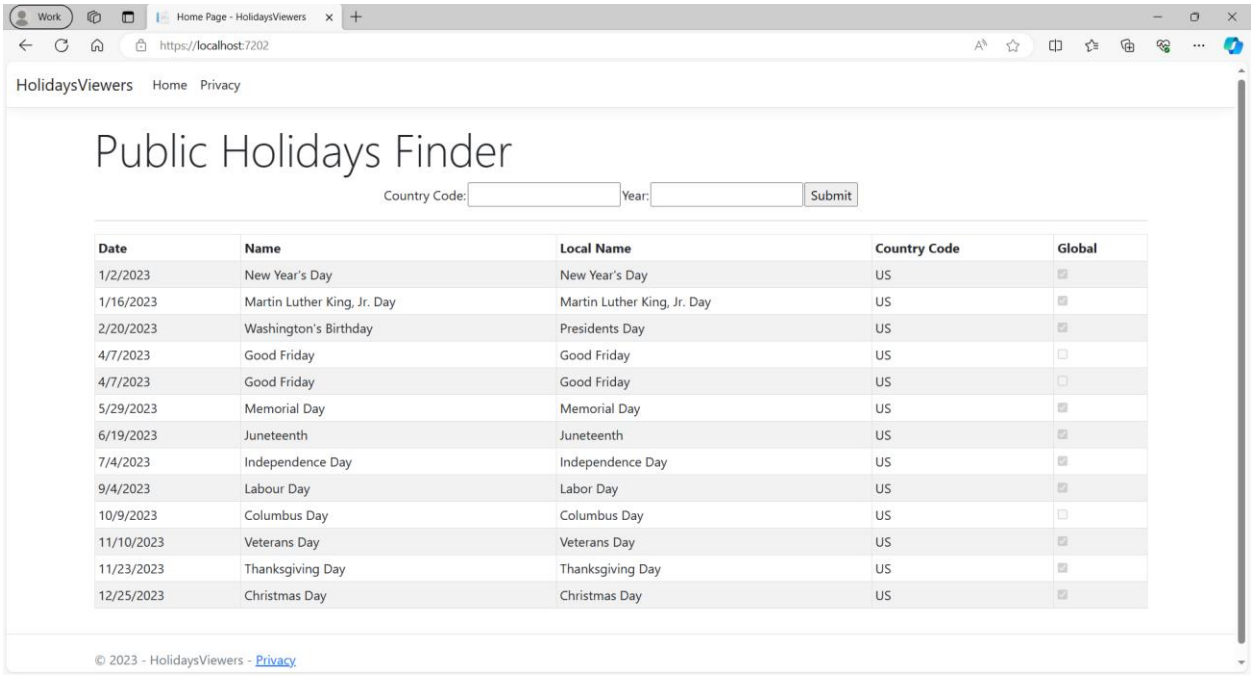
Finally, we need a Razor View to create a form where the user will input country code and year. The form will be submitted to the above Index action which will then call the **GetHolidays** method. Here is the code of **Index.cshtml** Razor view showing an HTML form and a table to display the public holidays. **You need to rewrite the form using HTML tags.**

```
@model List<HolidayModel>
@{
    ViewData["Title"] = "Home Page";
}

<div>
    <h3 class="display-4">Public Holidays Finder</h3>
    <center>
        <form asp-controller="Home" asp-action="Index">
            <table>
                <tr>
                    <td>Country Code: </td>
                    <td><input type="text" id="txtCountryCode"
name="CountryCode" /></td>
                    <td>Year: </td>
                    <td><input type="text" id="txtYear" name="Year"
/></td>
                    <td><input type="submit" value="Submit" /></td>
```

```
                </tr>
            </table>
            <hr />
</form>
    </center>
    @if (Model != null && Model.Count > 0)
    {
        <table class="table table-bordered table-striped table-sm">
            <thead>
            <tr>
                <th>Date</th>
                <th>Name</th>
                <th>Local Name</th>
                <th>Country Code</th>
                <th>Global</th>
            </tr>
            </thead>
            <tbody>
@foreach (var item in Model)
            {
                <tr>
                    <td>@item.Date.Value.ToShortDateString()</td>
                    <td>@Html.DisplayFor(modelItem => item.Name)</td>
                    <td>@Html.DisplayFor(modelItem => item.LocalName)</td>
                    <td>@Html.DisplayFor(modelItem =>
item.CountryCode)</td>
                    <td>@Html.DisplayFor(modelItem => item.Global)</td>
                </tr>
            }
            </tbody>
        </table>
    }

</div>
```

It is now time to test our application and see if we will be able to consume the third Party API. Press F5 in Visual Studio and you will see a page similar to the following. You can input a country code e.g. US, DE, etc., and a year e.g. 2021, and click the “Submit” button and if everything goes well you will see our code calling a third-party API, fetching a list of Public Holidays from the API and displaying it on the page.

Output sample



Submission:

1. Submit the project in zip file (extension zip)
2. A pdf file with all the code you created.
3. A zoom link or YouTube that demonstrates all the features of the program (less than 4 minutes).
   Write the link on the comment section in the Dropbox.
4. Completion of each team member

Note:

The program is running correctly and met all the requirements is 5 points, but
Incomplete item 1(-5 points)
Incomplete item 2 (-5 points)
Incomplete item 3 (-5 points)

Part II (20 points)

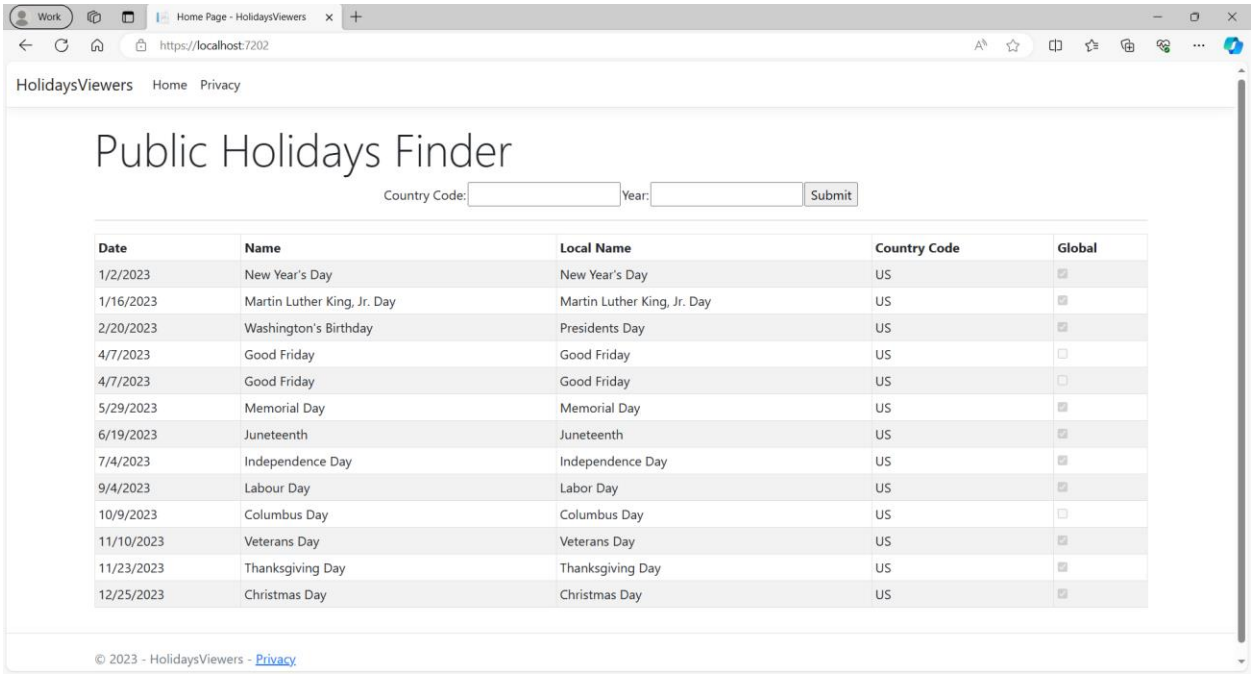
Design your client using HTML and other dynamic tools such as Ajax.

Submission:

1. Submit the project in zip file (extension zip)

2. A pdf file with all the code you created in the program
3. Write a procedure how to run the program part II on the comment box. If I cannot run the program, you will get zero credit for the lab assignment.
4. A zoom link or YouTube that demonstrates all the features of the program (less than 4 minutes).
   Write the link in the comment section in the Dropbox.
5. Completion of each team member

Output sample



Note:

The program is running correctly and met all the requirements is 20 points, but
Incomplete item 1(-20 points)
Incomplete item 2 (-15 points)
Incomplete item 3 (-10 points)
Incomplete item 3 (-10 points)