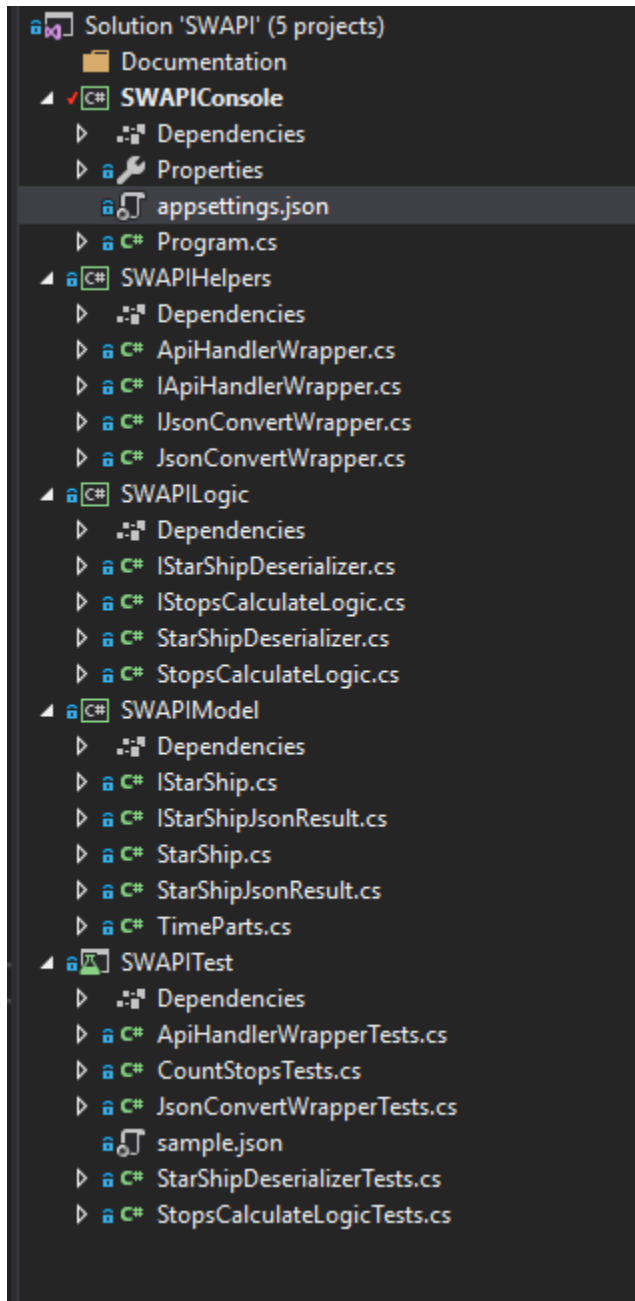


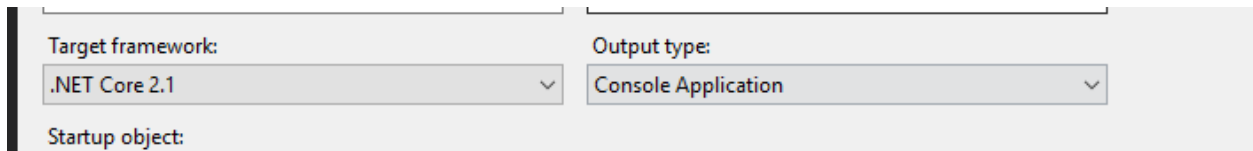
Accompanying documentation

The document below desired to help explaining the implementation and also provides some hints about the solution. The solution created to calculate the star ships stop number for a given distance is SWAPI solution.



The documentation can be found under the Documentation folder.

For all projects the target Framework is .Net Core 2.1.



The screenshot shows a portion of the Visual Studio interface. It features two dropdown menus. The first dropdown is labeled 'Target framework:' and is set to '.NET Core 2.1'. The second dropdown is labeled 'Output type:' and is set to 'Console Application'. Below these, the text 'Startup object:' is visible.

1. SWAPIConsole

The console app was created to gain the distance value for the given run and to provide the result of the run. It is a simple console app that is expecting only one argument at the start. If that was not provided the application will prompt to ask for the distance value. If there were more than one argument provided the application will prompt that and terminate the run. In any case, after one of the scenarios above, the application stops running and asks for any key to exit.

The Exception handling for the solution is also done here, with one exception.

API URL is stored in appsettings.json, so it can be maintained in that file. The code part that is handling the settings is:

```
var builder = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile(configPath, optional: true, reloadOnChange: true);

var configuration = builder.Build();

var url = configuration.GetSection("url").Value;
```

After the URL and the distance value were gained, the program tries to cast the distance to an integer, if the conversion is failing, prompt the issue to the user and terminate the run.

If everything is alright, create the class responsible for calculating the result and calling the corresponding function. After the result was created, the application iterates through all response items and write the result to the console output.

```
var calculate = new StopsCalculateLogic(
    new StarShipDeserializer(new JsonConvertWrapper<StarShipJsonResult>()),
    new ApiHandlerWrapper());
var result = Task.Factory.StartNew(() => calculate.CalculateStops(distance, url));
result.Wait();
foreach (var item in result.Result.Result) Console.WriteLine(item);
```

The console app has this only this Program.cs class with the functionality discussed above.

2. SWAPIHelpers

The SWAPIHelpers class library was created for wrappers and helpers used in the solution. The purpose of wrappers is to avoid the integration of third party or any other link into the solution so deeply to make testing hard or not possible. With this approach, the code is separated with wrappers, and the wrappers are injected into the class.

The ApiHandlerWrapper class and his interface IApiHandlerWrapper are responsible for handling any API call to the URL passed as parameter and reading the response and returning it as a string (json) to the caller. The method implemented is creating a WebRequest and is calling the GetResponseAsync to get the API call response.

```
var request = (HttpWebRequest)WebRequest.Create(url);
var response = await request.GetResponseAsync();
using (Stream responseStream = response.GetResponseStream())
{
    if (responseStream == null) return string.Empty;
    var reader = new StreamReader(responseStream, System.Text.Encoding.UTF8);
    return await reader.ReadToEndAsync();
}
```

There is also an error handling here, the only one expect the Program.cs, to get the error message from the response stream.

The class JsonConvertWrapper and his interface IJsonConvertWrapper are responsible to deserialize the json object to the model StarShipJsonResult using the Newtonsoft.Json NuGet package. This class can be extended with other methods like Serialize, but for the solution the only method needed was the Deserialize.

```
public T Deserialize(string json)
{
    return JsonConvert.DeserializeObject<T>(json);
}
```

3. SWAPILogic

This project contains two classes to separate the logic of this given project from the rest of the solution. The class `StarShipDeserializer` has an object of `IJsonConvertWrapper<StarShipJsonResult>` injected into the constructor, to provide the wrapper for deserializing the json object to `StarShipJsonResult`. The only function of the class is very basic, just calling the wrapper deserialize method.

```
public IStarShipJsonResult DeserializeStarShips(string json)
{
    return _jsonConvertWrapper.Deserialize(json);
}
```

`StopsCalculateLogic` class has the main responsibility to calculate the stops count for all ships returned by API, it has two objects injected, `IStarShipDeserializer` presented above, and `IApiHandlerWrapper` presented in helpers section. At the start the API handler is calling the URL gained as a parameter and is returning the json result as a string. If the result is empty the method returns an empty list, otherwise the deserialize is called and the result is checked. If the result is null or has no ships it is returning an empty list. Just for notice the returned result from the API call has wrong count, for example, it shows count 37 and has 10 ships data in result set. This is not affecting the result, that is created by iterating through the star ship list and adding the ship name and the calculated stops by distance, this calculation done by each ship model itself and will be present in the next section.

The returned string list contains items formatted like this
 {ship name}:{ship count stops}

```
var result = new List<string>();
var json = await _apiHandlerWrapper.GetApiCallResultAsync(url);
if(string.IsNullOrEmpty(json)) return result;
var ships = _starShipDeserializer.DeserializeStarShips(json);
if(ships?.Results == null || !ships.Results.Any()) return result;
foreach (var ship in ships.Results) result.Add($"{ship.Name} :{ship.CountStops(distance)}");
return result;
```

4. SWAPIModel

This project has model classes for deserialization of the json response of the API. The StarShipJsonResult is the main class which contains the star ships data in the result object which is a list of StarShip objects.

The StarShip objects have a method called CountStops, which based on the given distance provided as parameter, and the Consumables data of star ship is calculating how many stops needed to travel through given distance.

```
var consumables = ConsumablesInHoursAsync(Consumables);  
if (distance == 0 || MGLT == 0 || consumables == 0) return 0;  
return distance / (consumables * MGLT);
```

The consumables are provided in different formats, so it needs to be transformed, this is done in this part of code

```
var data = consumables.Split(' ');  
  
if (!int.TryParse(data[0], out var value)) return 0;  
  
switch (data[1].Trim().ToLower())  
{  
    case TimeParts.Year :  
    case TimeParts.Years:  
    {  
        value = value * yearDayCount * dayHourCount;  
        break;  
    }  
    case TimeParts.Month:  
    case TimeParts.Months:  
    {  
        value = value * monthDayCount * dayHourCount;  
        break;  
    }  
    case TimeParts.Week:  
    case TimeParts.Weeks:  
    {  
        value = value * weekDayCount * dayHourCount;  
        break;  
    }  
    case TimeParts.Day:  
    case TimeParts.Days:  
    {  
        value = value * dayHourCount;  
        break;  
    }  
}
```

If the value can't be transformed to a valid value, the result returned is 0.

So, if the distance is 0 or the MGLT capacity of the ship is 0 or the consumable transformation returns 0, the program will return -1 for given ship stop count, so if one of the ships result shows -1 it is because one of these issues. Returning 0 in this case can mislead because may exists some ships that can travel the distance without stops and the result of the run for them also will be 0.

5. SWAPITest

The test project is hosting all the test classes created to test the solution. For test use NSubstitute for object mock and FluentAssertions for easy and clears assertions.

Most of the test cases are unit tests, but there is also some integration test in IntegrationTests class and ApiHandlerWrapperTests class.

The solution is available on GitHub too <https://github.com/portikcs/SWAPI>.