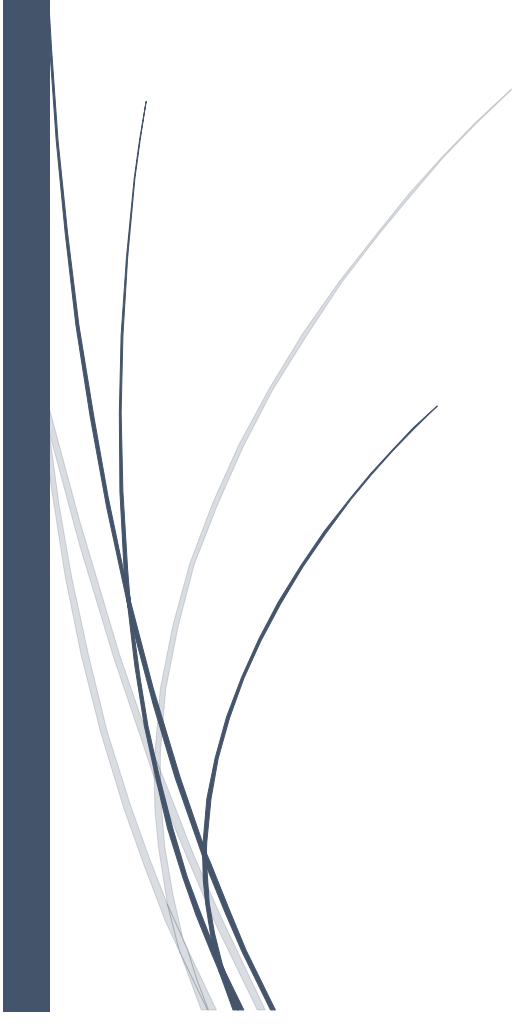
11/17/2023

API_Wrapper - SWAPI

Documentation and functionality



Llayton Titus

Table of Contents

Overview

This SWAPI Wrapper API simplifies access to information related to Star Wars that is found on the Star Wars API (SWAPI). This guide provides information on the API is set up, built and ran and the guide will have a basic explanation of the code structure and how the API makes it's requests.

A simple interface is provided to be able to find the names of the Star Wars films, the starships of the particular selected film and/or the characters of a selected film and all of this information comes form the Star Wars API references above.

Features

The features of the API are listed below:

- Retrieve a list of the Star Wars films
- Get details about starships featured in a specific film.
- Get details about characters featured in a specific film.
- Caching mechanism to improve API response time.

Getting started

The API was created in Visual Studio 2022 using an ASP .NET CORE Web API and .NET 6.0 SDK. The packages installed from the NuGet package manager is listed below:

- AspNetCoreRateLimit
- Microsoft.VisualStudio.Azure.Containers.Tools.Targets
- Swashbuckle.AspNetCore
- Swashbuckle.AspNetCore.Swagger
- Swashbuckle.AspNetCore.SwaggerGen
- System.Net.Http

To run the API, you can open the solution in Visual Studio and then build the solution.

Project structure

The project follows a standard ASP.NET Core Web API structure. Key components include:

- Controllers: Handle incoming HTTP requests and return appropriate responses.
- Services: Implement the SWAPI integration and caching mechanisms.
- Models: Define data structures used throughout the application.

API Endpoints

Listed below is the different endpoints to get the films, starships of the films and characters of the films:

Endpoint to get Films - GET /api/swapi/films

```
// Endpoint to get films.
[HttpGet("films")]
public async Task<IActionResult> GetFilms()
{
    try
    {
        var films = await _swapiWrapper.GetFilmsAsync();
        return Ok(films);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
```

Endpoint to get Starships of the films - GET /api/swapi/starships/{filmId}

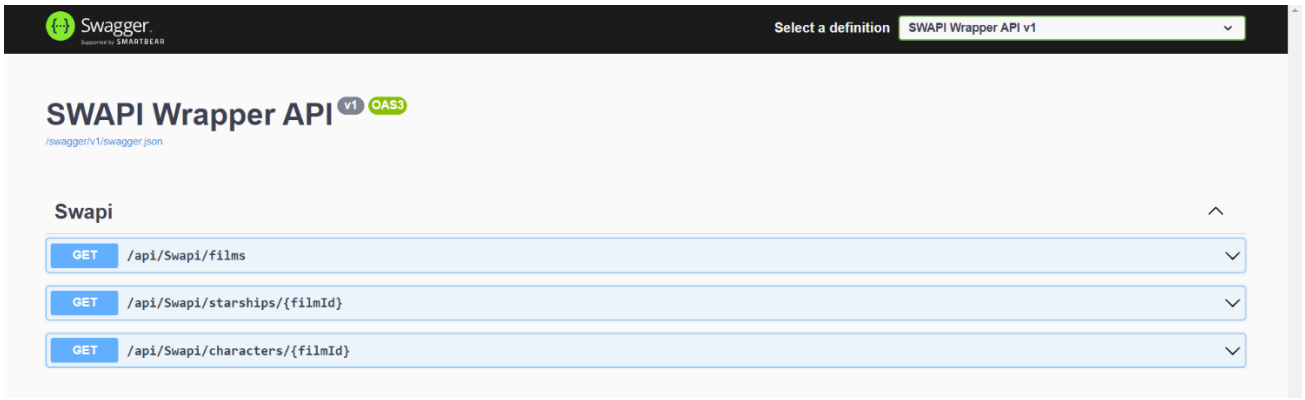
```
// Endpoint to get starships for a film.
[HttpGet("starships/{filmId}")]
public async Task<IActionResult> GetStarshipsForFilm(int filmId)
{
    try
    {
        var starships = await _swapiWrapper.GetStarshipsForFilmAsync(filmId);
        return Ok(starships);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
```

Endpoint to get Characters of the films - GET /api/swapi/characters/{filmId}

```
// Endpoint to get characters for a film.
[HttpGet("characters/{filmId}")]
public async Task<IActionResult> GetCharactersForFilm(int filmId)
{
    try
    {
        var characters = await _swapiWrapper.GetCharactersForFilmAsync(filmId);
        return Ok(characters);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal server error: {ex.Message}");
    }
}
```

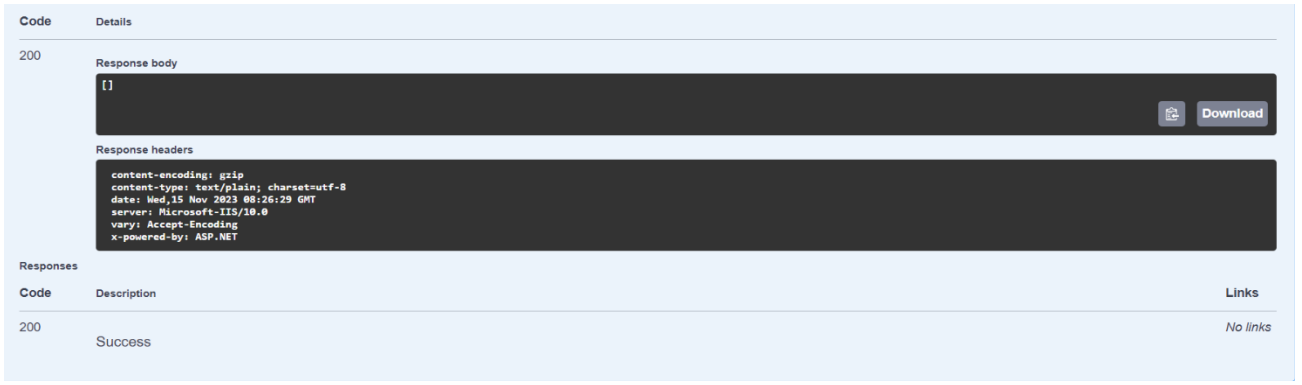
Output

When running the API from Visual Studio this is the expected output:



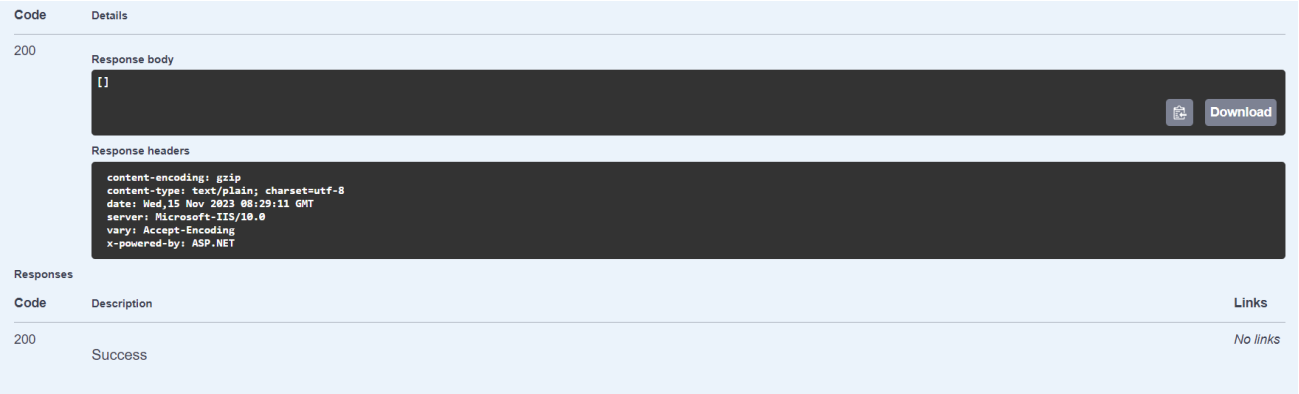
When executing the "GET /api/swapi/films" command, this is the output:

When executing the "GET /api/swapi/starships/{filmId}" command this is the output:



The "[]" that is received, shows that the array where the starships should be is empty. Although everything runs as a success as indicated with code 200, the array is still empty when it should show the names of the starships for the selected film.

When executing the "GET /api/swapi/characters/{filmId}" command this is the output:



The same happens to characters that happens to starships. "[]" indicates that there is an empty array, meaning it could not find the information that needs to be displayed, but everything still runs as a success as shown by the error code 200.

Caching Mechanism

The application implements a caching mechanism using MemoryCache to store API responses for 5 minutes. This improves performance by reducing the need to make repeated requests to the SWAPI.

A code snippet below shows where the cached method is created:

```
// Helper method to get cached data from the specified endpoint.
private async Task<string> GetCachedDataAsync(string endpoint)
{
    if (_cache.TryGetValue(endpoint, out string cachedData))
    {
        return cachedData;
    }

    string apiUrl = $"{_baseUrl}{endpoint}";
    HttpResponseMessage response = await _httpClient.GetAsync(apiUrl);

    if (response.IsSuccessStatusCode)
    {
        string responseData = await response.Content.ReadAsStringAsync();
        _cache.Set(endpoint, responseData, TimeSpan.FromMinutes(5));
        return responseData;
    }
    else
    {
        throw new Exception($"Failed to fetch data from {apiUrl}. Status code: {response.StatusCode}");
    }
}
```

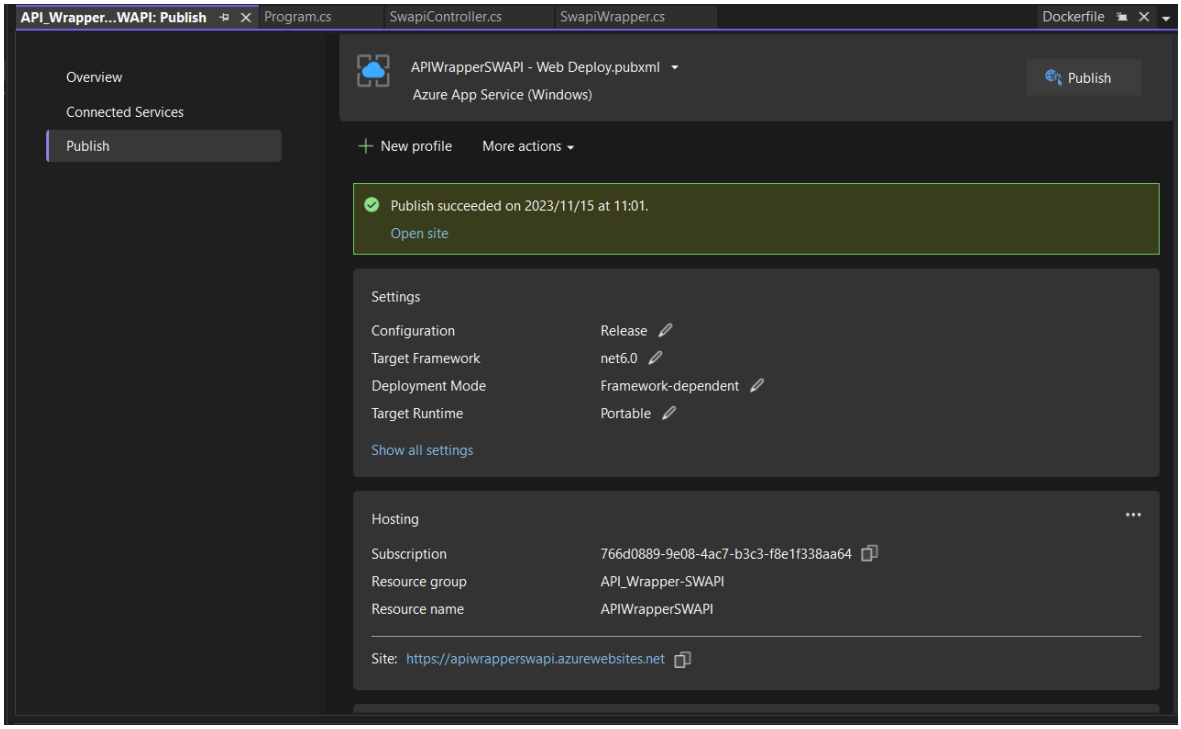
Error handling

Error handling was applied throughout the code to make sure that when an error is encountered, we know what the cause of the error is. This is shown in the code snippets that has been provided above and also in the Visual studio solution.

Deploy and Publish

The API is deployed to an Azure App service and can be accessed through the link below:

Link to website: https://apiwrapperswapi.azurewebsites.net



End