41509056

Antonet Zwane

Bachelor of Science in Information Technology

24 January 2025

CMPG 323

GitHub Link:

https://github.com/antonetzwane/CMPG-323-Netflix-Assignment-41509056

GitHub Project:

https://github.com/users/antonetzwane/projects/4/views/1

One Drive Link

https://nwuac-my.sharepoint.com/:x:/r/personal/41509056_mynwu_ac_za/Documents/netflix.xlsx?d=wae725968d1004c79ab009844a29b81af&csf=1&web=1&e=UASAS3

Power BI Report

https://app.powerbi.com/links/eD2iw-JcjZ?ctid=b14d86f1-83ba-4b13-a702-b5c0231b9337&pbi_source=linkShare

API Swagger URL

https://localhost:7166/swagger/index.html

How to:

- The API project is created to manipulate, access, write and read the provided excel file namely, Netflix.xlsx, found in data.world.
- Different methodologies such as Scrum and Agile are used to break or divide the project into smaller manageable units of work.
- Source control, methods like, GET, POST, PATCH/PUT, DELETE, FETCH and deduplicate are used for functionality of the Netflix API Solution.

Throughout the project these resource or software:

- Power BI (both desktop and service)
- Visual Studio 2022
- .NET Core Frameworks
- GitHub
- Excel

Tasks:

- As part of Scrum Implementation and Source Control a repository and a project on GitHub were created with specific tasks, milestones, labels, statuses and more to keep track and monitor the project's progress.

Integration: Building the API

- After creating a new project on visual studio, download EPPlus, this allows access to the excel file. You can achieve this by clicking Tools, NuGet Package Manager, Manage NuGet Packages for Solution, then searching for EPPlus on the browse side.

- Create a model to represent the structure of the database:

```
using System;
using System.Collections.Generic;

namespace NetflixAPISolution.Models
{
    16 references
    public class NetflixData
    {
        9 references
        public int Id { get; set; }
        7 references
        public string? title { get; set; } = null;
        5 references
        public string? rating { get; set; } = null;
        5 references
        public string? ratingLevel { get; set; }
        5 references
        public int ratingDescription { get; set; }
        7 references
        public int releaseyear { get; set; }
        5 references
        public int userratingscore { get; set; }
        5 references
        public int userratingsize { get; set; }

        //NetflixExcel ex = new NetflixExcel(@"C:\Users\zwane\Downloads\netflix.xlsx", 1);//netflix.xlsx"
    }
}
```

- Create a datalink class to load data from Excel to Visual Studio using EPPlus which always excel files to be used in Visual Studio:

```
using OfficeOpenXml;
using System.Collections.Generic;
using System.IO;
using Microsoft.EntityFrameworkCore;
using System;
using Microsoft.Office.Interop.Excel;

namespace NetflixAPISolution.Models
{
    2 references
    public class DataLink
    {
        2 references
        public List<NetflixData> LoadDataFromExcel(string filePath)
        {
            ExcelPackage.LicenseContext = LicenseContext.NonCommercial;
            var netflixItems = new List<NetflixData>();

            using (var package = new ExcelPackage(new FileInfo(filePath)))
            {
                var xlWorkbook = new ExcelPackage(new FileInfo(@"C:\Users\zwane\Downloads\netflix.xlsx"));

                ExcelWorksheet workSheet = xlWorkbook.Workbook.Worksheets[1];
                var worksheet = package.Workbook.Worksheets[""]; // First sheet
                int rowCount = worksheet.Dimension.Rows;

                for (int row = 2; row <= rowCount; row++) // Start from row 2 to skip headers
                {
```

```
netflixItems.Add(new NetflixData
{
    /**title = worksheet.Cells[row, 1].Text,
    rating = worksheet.Cells[row, 2].Text,
    ratingLevel = worksheet.Cells[row, 3].Text,
    ratingDescription = int.Parse(worksheet.Cells[row, 4].Text),
    releaseyear = int.Parse(worksheet.Cells[row, 5].Text),
    userratingscore = int.Parse(worksheet.Cells[row, 6].Text),
    userratingsize = int.Parse(worksheet.Cells[row, 7].Text)**/

    title = worksheet.Cells[row, 1].Text,
    rating = worksheet.Cells[row, 2].Text,
    ratingLevel = worksheet.Cells[row, 3].Text,
    ratingDescription = int.TryParse(worksheet.Cells[row, 4].Text, out var ratingDescription) ? ratingDescription : 0,
    releaseyear = int.TryParse(worksheet.Cells[row, 5].Text, out var releaseYear) ? releaseYear : 0,
    userratingscore = int.TryParse(worksheet.Cells[row, 6].Text, out var userRatingScore) ? userRatingScore : 0,
    userratingsize = int.TryParse(worksheet.Cells[row, 7].Text, out var userRatingSize) ? userRatingSize : 0
});


netflixItems;
```

- Create a controller class to load and save the dataset into a staging database, to avoid making changes to the raw data.

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
//using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using NetflixAPISolution.Models;
using static Microsoft.EntityFrameworkCore.DbLoggerCategory;
using NetflixAPISolution.Repository;

namespace NetflixAPISolution.Controllers
{

    [ApiController]
    [Route("api/[controller]")]

    1 reference
    public class DatasetController : ControllerBase
    {
        private readonly NetflixInterface _repository;

        0 references
        public DatasetController(NetflixInterface repository)
        {
            _repository = repository;
        }

        private readonly List<NetflixData> _stagingDataset = new();

        [HttpPost("load-from-excel")]
```

- With CRUD operations such as GET, to retrieve data from the dataset.

```
[HttpGet("staging-data")]
0 references
public IActionResult GetStagingData()
{
    return Ok(_stagingDataset);
}
```

- GET by ID, to retrieve data by ID.

```
//Code to GET or retrieve items from the dataset by ID.
[HttpGet("{id}")]
1 reference
public IActionResult GetById(int id)
{
    var item = _stagingDataset.FirstOrDefault(x => x.Id == id);
    if (item == null)
    {
        return NotFound($"Item with ID {id} not found.");
    }
    return Ok(item);
}
```

- POST, which creates a new entry in the data set.

```
//Code to create new entry in the Netflix dataset.
[HttpPost]
0 references
public IActionResult CreateItem([FromBody] NetflixData newItem)
{
    if (_stagingDataset.Any(x => x.Id == newItem.Id))
    {
        return BadRequest("Item with the same ID already exists.");
    }

    _stagingDataset.Add(newItem);
    return CreatedAtAction(nameof(GetById), new { id = newItem.Id }, newItem);
}

//Code to update an existing row in the Netflix dataset.
[HttpPut("{id}")]
0 references
public IActionResult UpdateItem(int id, [FromBody] NetflixData updatedItem)
{
    var item = _stagingDataset.FirstOrDefault(x => x.Id == id);
    if (item == null)
    {
        return NotFound($"Item with ID {id} not found.");
    }

    // Update fields
    item.title = updatedItem.title;
    item.releaseyear = updatedItem.releaseyear;

    return NoContent();
}
```

- FETCH retrieves data from the dataset and inserts it into the API as staging data.
- DELETE, removes existing rows.
- DEDUPLICATE removes duplicates from the dataset.

```
[HttpPost("fetch-dataset")]
0 references
public IActionResult FetchDataset([FromQuery] string filePath)
{
    if (string.IsNullOrEmpty(filePath) || !System.IO.File.Exists(filePath))
    {
        return BadRequest("Invalid file path provided.");
    }

    var loader = new DataLink();
    var data = loader.LoadDataFromExcel(filePath);
    _stagingDataset.Clear(); // Clear existing data
    _stagingDataset.AddRange(data);
    return Ok("Dataset fetched and loaded successfully.");
}

//Code to remove all duplicate records in the dataset.
[HttpPost("deduplicate")]
0 references
public IActionResult Deduplicate()
{
    var distinctItems = _stagingDataset
        .GroupBy(x => new { x.title, x.rating, x.ratingLevel, x.ratingDescription, x.releaseyear, x.userratingscore, x.userratingsize })
        .Select(g => g.First())
        .ToList();

    _stagingDataset.Clear();
    _stagingDataset.AddRange(distinctItems);

    return Ok("Duplicates removed successfully.");
}
```

- Create a Repository for an Interface class and a Repository class.
- Test the API by running it, using SWAGGER UI in Program.cs class.

```
using Microsoft.EntityFrameworkCore;
using NetflixAPISolution.Models;
using NetflixAPISolution.Repository;

var builder = WebApplication.CreateBuilder(args);


//Code to load the repository class into controller class.
builder.Services.AddSingleton<NetflixRepository, NetflixRepository>();

var filePath = @"C:\Users\zwane\Downloads\netflix.xlsx";
// Add services to the container.
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

- Clone your GitHub repository, click Git, Clone Repository and paste the GitHub repository link to sign.

- The challenging part is to push to the repository after changes or updates on Git changes.

Visualization

- Create a Power BI report.
- In the report, create a High-Level Summary and Detailed Summary page.
- The difference between these pages is that the High-Level Summary Page is not for detailed visuals, it gives executives an insight into the data being analyzed, pointing out important information. The Detailed Summary page is for details making use of the whole data.
- Publish the report on the Power BI Service by clicking publish and signing in with your credentials.