DevOpsWebScraperGui

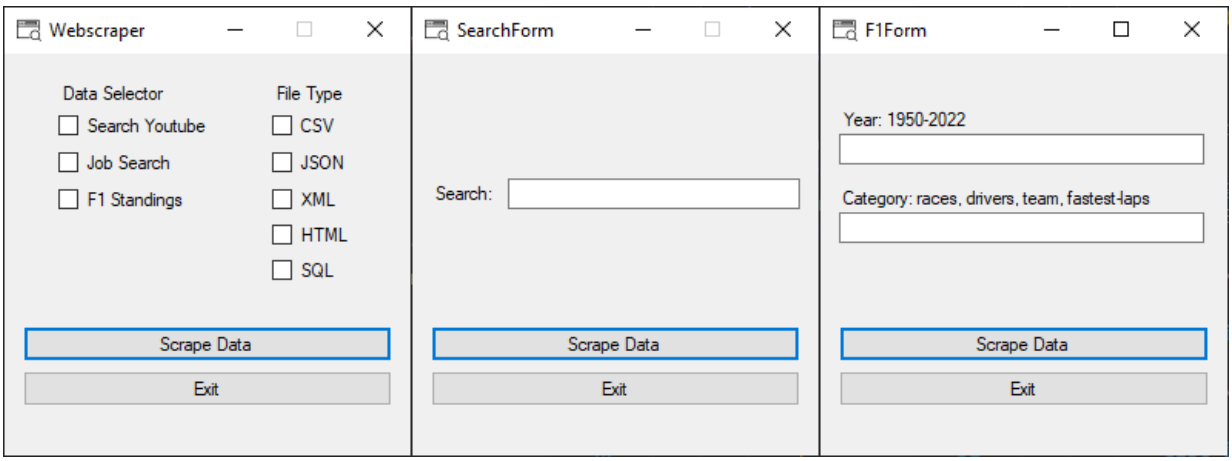
1. Table Of Contents

2. Introduction

This project is made for the DevOps & Security course on thomasmore geel. The goal is to make a console based webscraper application. This must be done with selenium browser automation tool and c#. The project description says that there need to be three scrapable websites: youtube, ictjob and a self chosen one. I chose for F1 standings. When the data is scraped from the web, it needs to be exported of course. It needed to be exportable to csv and json files, but I made the options to export to XML, HTML and sql.

3. Code

The project is build with two main classes: a class for the scraping part and a class for the export part. Besides that, there are 3 windows forms screens. A main screen and two scraper specific screens. You can find screenshots of the screens below. The windows forms screens where optional and a fun addition I added to the project.



3.1. Scrapers

The class is writen in the file: DataScraper.cs. It contains 5 methods, two of which are meant for development and testing. I will be focussing on the three methods used in the program. Communication between the data scrapers and the exporters are handled via a universal type. This is done by a two dimentional string list: List<List<string>>. This way, all scrapers can use all exporters and all exporters can convert data from all scrapers.

3.1.1. Youtube Search

The youtube search method takes one parameter, the search term. The user can input this in on of the screens. It then puts the searchterm in the url that the selenium driver is connecting to. This way, the program doesn't have to first load youtube and then load youtube again with the searchterm in the search bar. This minimizes the amount of requests that happen to youtube servers. The first thing that the scraper will get from the driver or from the webpage is the 'accept all' button for the cookies. I tried to send cookies through the driver, but Youtube changes its cookies slightly constantly so that wasn't an option. The button is saved as a IWebElement and later clicked on. Next, all videos on the webpage are stored in a array of IWebElements. This way, a foreach loop can be used to loop over all the videos and add their data to the list of string lists. I designed the lists in a way that the first item in the list is always a list of the names of the columns of the data. Those column names are hardcoded for the youtube searcher since those are not on the page itself. After that, the 5 latest released videos are extracted and added to the list. To finish the method, the driver is stopped and the final version of the list is returned.

```
public static List<List<string>> YoutubeSearch(string searchTerm)
{
    String urlSearch = "https://www.youtube.com/results?search_query=
{0}&sp=CAISAhAB";

    // Setting up the driver manager that automatically handles all
drivers (drivers don't need to be installed)
    new DriverManager().SetUpDriver(new ChromeConfig());
    // Starting the google chrome driver
    IWebDriver driver = new ChromeDriver();

    // Navigating the chrome page to the desired link
    driver.Navigate().GoToUrl(String.Format(urlSearch,
searchTerm.Replace(' ', '+')));

    System.Threading.Thread.Sleep(4000);

    //https://www.youtube.com/results?search_query=Rick+Astley&sp=CAISAhAB
    // Accept cookies
    try
    {
        IWebElement cookieButton =
driver.FindElement(By.CssSelector("#content > div.body.style-scope.ytd-
consent-bump-v2-lightbox > div.eom-buttons.style-scope.ytd-consent-bump-
v2-lightbox > div:nth-child(1) > ytd-button-renderer:nth-child(2) > yt-
button-shape > button"));
        cookieButton.Click();
        System.Threading.Thread.Sleep(4000);
    }
```

```
    finally { }

    // Get all videos from the page in a collection
    ReadOnlyCollection<IWebElement> videos =
driver.FindElements(By.CssSelector("#contents > ytd-video-renderer"));

    // Create empty dicrionary list where all the videos will be added to.
    List<List<string>> videoData = new List<List<string>>();

    // Create an empty list where the starting data is going to
    List<string> headerList = new List<string>
    {
        "Title", "Url", "Uploader", "Views", "UploadDate"
    };
    videoData.Add(headerList);

    int max = 5;
    // Every video gets its own dictionary in the list
    for (int i = 0; i < 5; i++)
    {
        try
        {
            // Create an empty list where all data is going to
            List<string> list = new List<string>();

            // Defining the variable where the data will be stored in
            string str_videoTitle, str_videoUrl, str_uploader, str_views,
str_uploadDate;

            // Getting the web elements from the from the page
            IWebElement elem_videoTitle =
videos[i].FindElement(By.CssSelector("#video-title > yt-formatted-
string"));
            IWebElement elem_videoUrl =
videos[i].FindElement(By.CssSelector("#video-title"));
            IWebElement elem_uploader =
videos[i].FindElement(By.XPath("div[1]/div/div[2]/ytd-channel-
name/div/div/yt-formatted-string/a"));
            IWebElement elem_views =
videos[i].FindElement(By.CssSelector("#metadata-line > span:nth-
child(3)"));
            IWebElement elem_uploadDate =
videos[i].FindElement(By.CssSelector("#metadata-line > span:nth-
child(4)"));

            // Extracting the data from the web elements
            str_videoTitle = elem_videoTitle.Text;
            str_videoUrl = elem_videoUrl.GetAttribute("href");
            str_uploader = elem_uploader.Text;
            str_views = elem_views.Text;
            str_uploadDate = elem_uploadDate.Text;

            // Adding the extracted data to the dictionary.
            list.Add(str_videoTitle);
```

```
            list.Add(str_videoUrl);
            list.Add(str_uploader);
            list.Add(str_views);
            list.Add(str_uploadDate);

            videoData.Add(list);
        }
        catch { max++; }

    }

    driver.Quit();

    return videoData;
}
```

3.1.2. Search Job

This method is almost exactly the same as the youtube search method. It also has one parameter, it also inserts the search term in the url, it saves all jobs in a list, it adds the headers of the columns, ... The only big difference is that it has to sort the list of videos by date before extracting the video's. It also changes the search method to searching for all given words. Then finally it extracts all data from the jobs with one exception: some of the items in the jobs list aren't actual jobs but are advertisements. These are skipped in the loop by using a try statement. If the item is an ad, the driver will return an error because it cant find the title of the job and the loop will just skip that one and go one list item further to maintain the top five items.

```
public static List<List<string>> SearchJob(string searchTerm)
{
    String urlSearch = "https://www.ictjob.be/en/search-it-jobs?keywords=
{0}";

    // Setting up the driver manager that automatically handles all
drivers(drivers don't need to be installed)
    new DriverManager().SetUpDriver(new ChromeConfig());
    // Starting the google chrome driver
    IWebDriver driver = new ChromeDriver();

    driver.Navigate().GoToUrl(String.Format(urlSearch,
searchTerm.Replace(' ', '+')));
    System.Threading.Thread.Sleep(5000);

    //click the button to sort on date
    IWebElement dateButton = driver.FindElement(By.CssSelector("#sort-by-
date"));
    dateButton.Click();
    System.Threading.Thread.Sleep(2500);

    // Click the button to change the search method
    IWebElement allWordsButton = driver.FindElement(By.XPath("//*
[@id=\"column-keywords-options\"]/label[2]/span"));
```

```
    allWordsButton.Click();
    System.Threading.Thread.Sleep(10000);

    // Getting all the jobs in a list
    ReadOnlyCollection<IWebElement> jobs =
driver.FindElements(By.CssSelector("#search-result-body > div > ul >
li"));

    // Defining empty return list
    List<List<string>> jobData = new List<List<string>>();

    // Starting to fill the list with data.
    List<string> headerList = new List<string>
    {
        "JobTitle", "Company", "Location", "Keywords", "Url"
    };
    jobData.Add(headerList);

    int max = 5;

    for (int i = 0; i < max & i < jobs.Count; i++)
    {
        try
        {
            // Create an empty list where all data is going to
            List<string> list = new List<string>();

            // Defining the variable where the data will be stored in
            string str_jobTitle, str_company, str_location, str_keywords,
str_url;

            // Getting the web elements from the from the page
            IWebElement elem_jobTitle =
jobs[i].FindElement(By.CssSelector("span.job-info > a > h2"));
            IWebElement elem_company =
jobs[i].FindElement(By.CssSelector("span.job-info > span.job-company"));
            IWebElement elem_location =
jobs[i].FindElement(By.CssSelector("span.job-info > span.job-details >
span.job-location > span > span"));
            IWebElement elem_keywords =
jobs[i].FindElement(By.CssSelector("span.job-info > span.job-keywords"));
            IWebElement elem_url =
jobs[i].FindElement(By.CssSelector("span.job-info > a"));

            // Extracting the data from the web elements
            str_jobTitle = elem_jobTitle.Text;
            str_company = elem_company.Text;
            str_location = elem_location.Text;
            str_keywords = elem_keywords.Text;
            str_url = elem_url.GetAttribute("href");

            // Adding the extracted data to the dictionary.
            list.Add(str_jobTitle);
            list.Add(str_company);
```

```
            list.Add(str_location);
            list.Add(str_keywords);
            list.Add(str_url);

            jobData.Add(list);
        }
        catch
        {
            max++;
        }

    }

    // Finishing up the method with quitting the driver and returning the
data.
    driver.Quit();
    return jobData;
}
```

3.1.3. F1 Standings

For the last method in in this chapter, the f1 standings website is scraped. On this website, you can find all driver champions, constructor champions, races from each year and fastest laps from each year from the start of F1 until now. This method accepts two parameters. One for the category to display (race, driver, team and fastest lap) and one for the year from which the data must be pulled. This data will be inserted in the url. A cookies button is scraped and clicked as well. The data in this webpage is put in a html table. This makes it very easy to read. First, the names of the columns are extracted form the table header. Then, the data is read from the table row per row. To finish the method, the driver quits again and the list with data is returned.

```
public static List<List<string>> ScrapeF1Data(string categoryChoice,
string yearChoice)
{
    string url = "https://www.formula1.com/en/results.html/{0}/{1}.html";

    // Setting up the driver manager that automatically handles all
drivers(drivers don't need to be installed)
    new DriverManager().SetUpDriver(new ChromeConfig());
    // Starting the google chrome driver
    IWebDriver driver = new ChromeDriver();

    driver.Navigate().GoToUrl(String.Format(url, yearChoice,
categoryChoice));

    IWebElement acceptCookies =
driver.FindElement(By.XPath("/html/body/div[5]/div/div/div[2]/div[3]/div[2
]/button[2]"));
    acceptCookies.Click();

    IWebElement dataTable = driver.FindElement(By.CssSelector("body >
```

```
div.site-wrapper > main > article > div > div.ResultArchiveContainer >
div.resultsarchive-wrapper > div.resultsarchive-content > div > table"));

    List<List<string>> jobData = new List<List<string>>();

    ReadOnlyCollection<IWebElement> columnNames =
dataTable.FindElements(By.CssSelector("thead > tr > th"));
    ReadOnlyCollection<IWebElement> columnData =
dataTable.FindElements(By.CssSelector("tbody > tr"));

    List<string> headerList = new List<string>();
    foreach (IWebElement columnName in columnNames)
    {
        if (!columnName.GetAttribute("class").Equals("limiter"))
        {
            if (columnName.GetAttribute("class").Equals("header"))
            {

headerList.Add(columnName.FindElement(By.TagName("abbr")).GetAttribute("ti
tle"));
            }
            else
            {
                headerList.Add(columnName.Text);
            }
        }
    }
    jobData.Add(headerList);

    foreach (IWebElement row in columnData)
    {
        List<string> list = new List<string>();
        ReadOnlyCollection<IWebElement> rowData =
row.FindElements(By.CssSelector("td"));
        foreach (IWebElement data in rowData)
        {
            if (!data.GetAttribute("class").Equals("limiter"))
            {
                if (data.GetAttribute("class").Equals(""))
                {
                    if (data.FindElement(By.TagName("a")).Text.Equals(""))
                    {
                        string name = "";
                        ReadOnlyCollection<IWebElement> span =
data.FindElement(By.TagName("a")).FindElements(By.TagName("span"));
                        for (int i = 0; i < 2; i++)
                        {
                            name += (span[i].Text);
                        }
                        list.Add(name);
                    }
                    else
                    {
                        list.Add(data.FindElement(By.TagName("a")).Text);
```

```
                    }
                }
                else
                {
                    list.Add(data.Text);
                }
            }
        }
        jobData.Add(list);
    }
    driver.Quit();
    return jobData;
}
```

All of the code can be found in the github repo mentioned in the conclusion.

3.2. Exporters

4. Conclusion

The project was very fun to make. I enjoyed getting back into c#. I learned alot about how big companies design their site and I will take this knowledge and try to build better, more effecient websites.

The source code for the project can be found on my github repository. The installer to the project is located in the Releases tab in the latest release.

5. Sources

- Selinum Webscraper
- Microsoft .net Documentation