Scraping FIS World Cup results using Selenium

Elective project

AIS

PHILIPP Quentin

Ljubljana 2021

Table of Contents

Scraping

Web scraping is a technique for extracting content from websites, with a script or program, to transform it for use in another context. It is often used to get a lot of data from a website automatically. However, one of the major drawbacks of this technic is that the script relies on the webpage structure and if a change is made on the website, the web scraper can fail.

To scrape a website, I decided to use Python3 with the framework Selenium. This framework is used for testing web application but can also be used only for web scraping.

The website

The website that I choose for this project is the FIS website. On the home page for the results, there is a list of every competition of the season. By clicking on a competition, this opens a new page for this competition. In every competition, there can be one or many events (race). To access the race detailed results, you need to click on the race.

The organization of the website is described in Figure 1. The first page is a list of competition. Each competition contains one or more events. In each event there is the result.

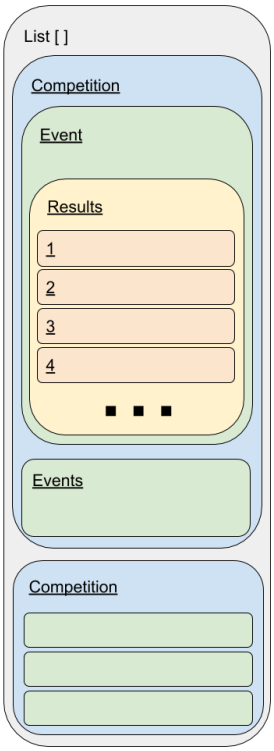

Figure 1: Structure of FIS results website

Link to the website main page:

https://www.fis-ski.com/DB/alpine-skiing/calendar-results.html?&sectorcode=AL&seasoncode=2021&categorycode=WC&seasonmonth=X-2021&saveselection=-1

Code organization

Code structures and files description

.env: Store my Gmail credentials. This file is only a local file and is not published online. The data in this file are loaded with python-dotenv which allows you to put your passwords and sensible data without the risks to publish it online.

.env.example: A file to show how the .env file looks like but without my credentials.

competition.py: Contain the definition of the Competition class.

config.json: Event structures, XPath, and conversions.

dataVisualisation.py: Calculation of data for the GUI.

display.py: Code of the GUI. Made with PyQt5.

emailBase.html: Template for the email.

event.html: Layout of an event for the email.

event.py: Contain the definition of the Event class.

fis.png: Image for the email.

mail.py: Script to send the email notification.

main.py: Script of the web scraper.

results.py: Contain the definition of the Result class.

results.json: Store all the data from the scraping.

Custom objects

The project is coded with an oriented object approach to simplify the management of the data and improve readability of the code. There are 3 classes used to store the data. It is represented with UML diagram on Figure 2.

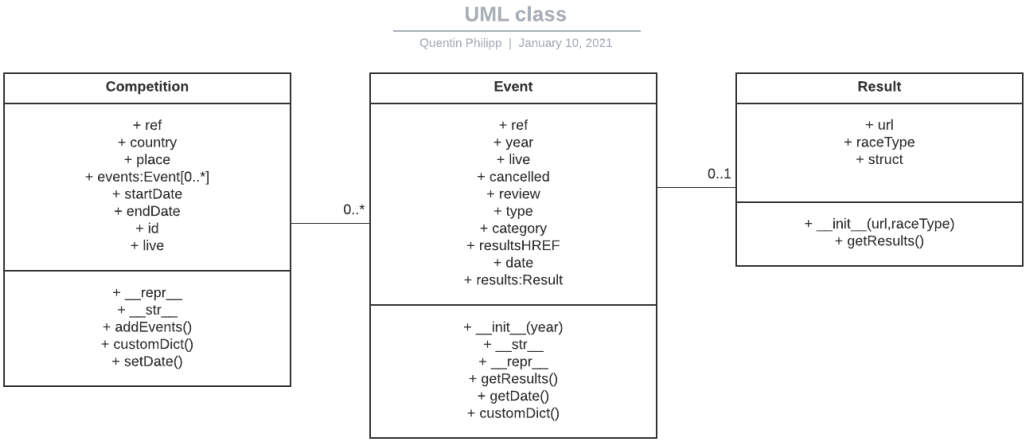

Figure 2: UML classes

The class “Results” has a particular structure because the attributes are added depending on the type of event. The added attributes are found in the “config.json” file which store the data structure for every event type.

Storing data

The data are stored in a JSON file “results.json”. The storing structure is the same as the website structure and use nested dictionary. The data are stored only at the end of the execution of “main.py” script. If the script fails, the results will not be overridden so the email script will still work because the date of events are published in advance, so it will not impact the other script.

I decided to use a simple JSON file instead of a database like MySQL in this project because the speed of writing and reading the data is not so important, the number of data to store are low and a JSON require less setup process than a database which make the project portable more easily.

Scrapper

The “main.py” file contain the scraping code. It starts with the loading of the page with all the competitions. For each competition, a Competition object is created. During the initialization of this object, the events of the competitions are added and the results.

Once all the data are collected, the script convert all the objects in a storable format, each object is transformed into dictionary without some of their attribute that are not relevant. Those dictionary are then serialized, converted to a JSON structure and written in the “results.json” file

Email

The email is sent using the “smtplib” library. I also used other libraries like “ssl” and “email” to create the emails.

The emails are sent from a personal Gmail address. To store my address and my password I used a .env file and the python-dotenv library. This library allows you to put confidential data in a separate file called “.env” and load those data like environment variable, then it can be used in the script. This way, if you do not share the “.env” file, everyone can see all the code but not the credentials.

One function check which competition are in live in the JSON file and the other function write the email subject, receiver, the main text and attach the picture. Then the email is sent. If there is no event in live, all this process is skipped.

The emails are HTML emails. The positive point of an HTML emails is that it is easy to have a nice-looking email and it is also more personalized. The layout of the email is stored in “emailBase.html” and for every event that are on live, a new event is added to this base email. The event layout is described in “event.html”

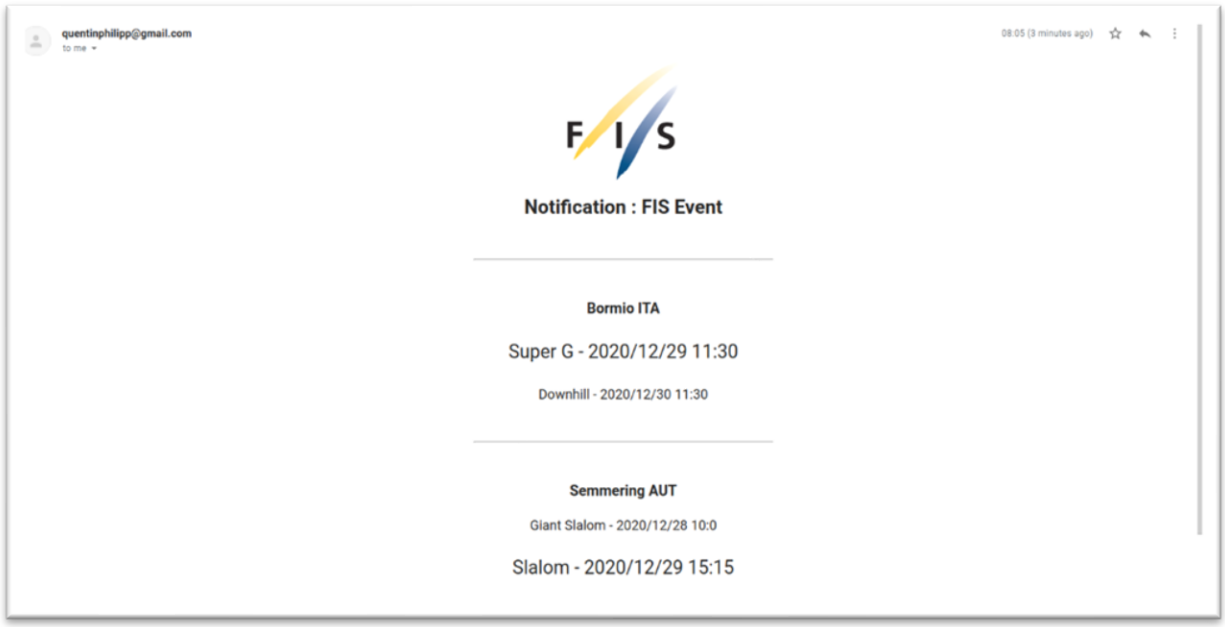

Figure 3:Screenshot of the email from the 29 December

GUI

The Graphical User Interface (GUI) is built with PyQt, a graphical library derived from the popular C++ framework Qt. This graphical interface is composed of two files, “display.py” that create the layout of the GUI, the button, the container, etc... and “dataVisualisation.py” where all the functions to calculate the data to plot, are written.

I added matplotlib library to plot one figure. This library is used to create scientific graphs.

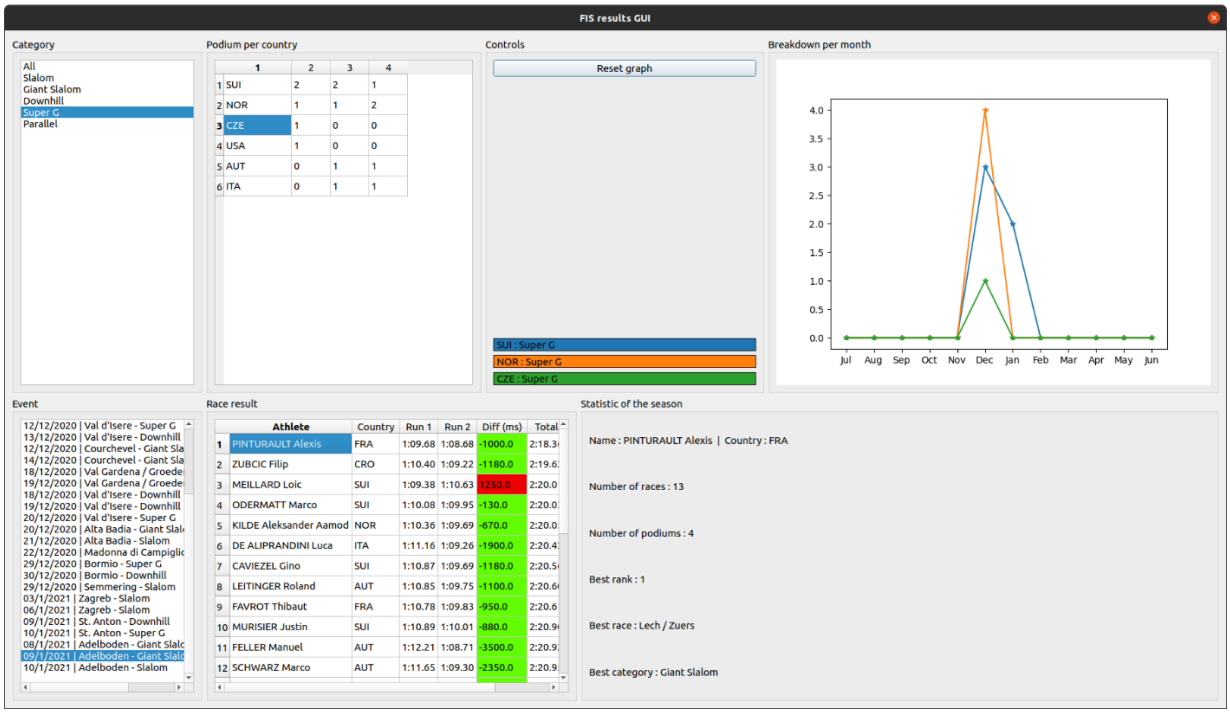

Figure 4: Graphical interface of the project

The first line of widgets is to compare the performance of each country. On the left you can select a category or “All” for all categories. Then by clicking on a country, you can see on the right the breakdown of the podiums per months. On the Figure 4, the breakdown graph shows a comparison between Switzerland, Norway, and Czech Republic only on the “Super G” events.

The second line is a summary of all the past events and allows the user to see the results. By clicking on an event on the left, you can see the result in the middle. If you click on an athlete, you will see more detailed info on this athlete.

The races results are different from one category to another, for example, in “Parallel” the timings are not displayed but only the rank. On the example it is Giant Slalom, this event has two run, so in the table you can see the two times for the runs, the difference of time between the first and the second run and the total time. This is also based on the events structure in the config file.

CRON job

The scripts “main.py” and “mail.py” are executed every morning at 8h00 and 8h05 respectively. To do this I used CRON. CRON is a utility to schedule task on Linux. I put this project on my server and setup tasks to run every day at the same time. This allows me to have a working project even if my computer is not switched on. The scraper run at 8h00 every morning and the email script at 8h05

```
# Example of the cron tasks

#    /min
#   / /hour
#  / / /day(month)
# / / / /month
#/ / / / /day(week)
0 8 * * * /usr/bin/python3.9 /home/quentin/WebScraperIA/main.py

5 8 * * * /usr/bin/python3.9 /home/quentin/WebScraperIA/mail.py
```

Figure 5: Example of 2 CRON tasks

Results and conclusion

The project is stable for now and after 2 weeks of uses I received 10 emails to warn me about live competitions. The email sending feature is the most relevant part of my project because it is useful for me. The GUI is more a proof of concept, it allows me to check the data more quickly, because opening every page of the website takes a bit a time, but it misses some other statistics or more data to be interesting.

To improve the project, I plan to modify the GUI code to automatically download the results file from the server to have updated data automatically and search for other data sources.