Raport z projektu indywidaulnego

Michał Banaszczak

14 czerwca 2022

Spis treści

1 Cel projektu

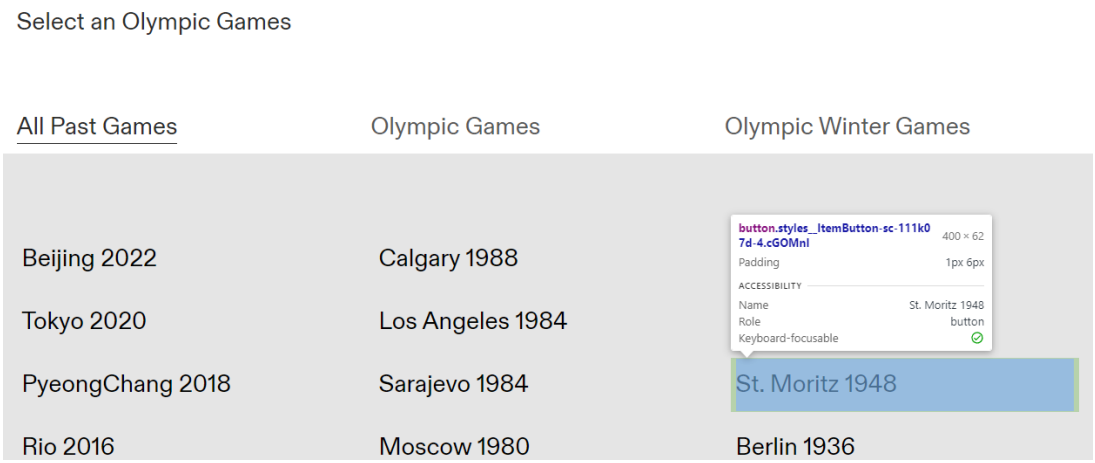
Pierwotnym celem projektu było przeprowadzenie analizy wyników sportowych dotyczących Igrzysk Olimpijskich. Nie mogłem nigdzie jednak znaleźć zadowalających zbiorów daych, dlatego cel został poszerzony o utworzenie takowego na podstawie informacji z oficjalnych stron IO. W trakcie rozwoju projektu jednak spodobały mi się technologie webowe, m.in. Node.js czy ekstensywne zastosowanie Chrome DevTools. Z tego względu zdecydowałem się ponownie poszerzyć zakres projektu o utworzenie dodatkowego zbioru danych na podstawie wyników medalistów z poszczególnych Igrzysk. Wyróżniłem w ten sposób 3 kamienie milowe projektu:

1. utworzenie zbioru danych dot. wyników wszystkich nowożytnych IO,
2. utworzenie zbioru danych medalistów poszczególnych IO,
3. analiza tych zbiorów

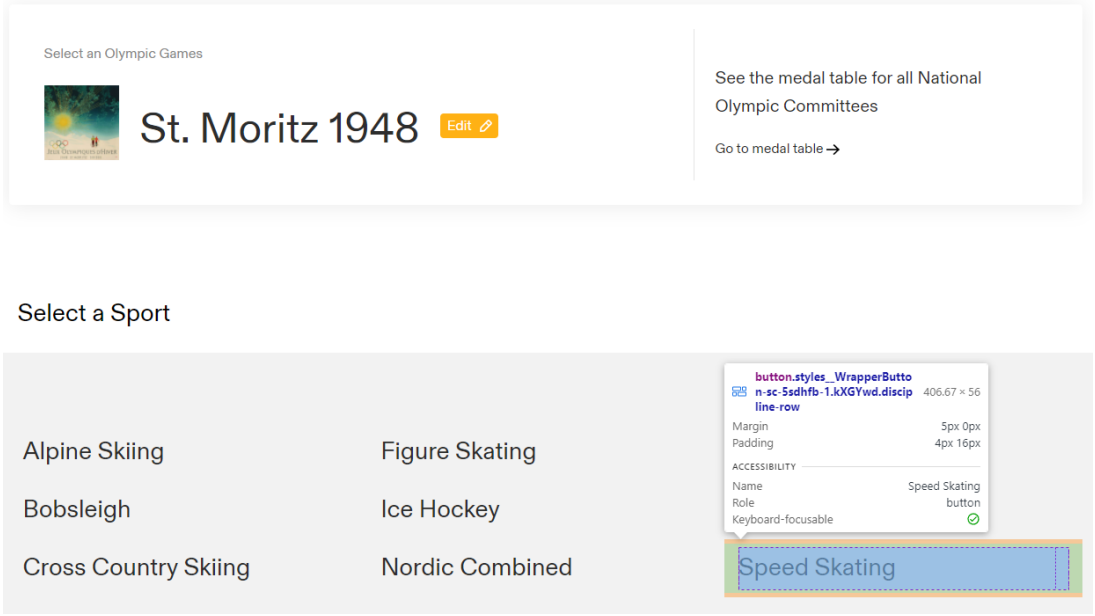
2 Struktura stron źródłowych do scrapowania

2.1 Strona z wynikami wszystkich Igrzysk

W tej aplikacji webowej wszystkie igrzyska, sporty i dyscypliny tworzą strukturę drzewa. Przy każdej interakcji guziki są niszczone, a nie chowane. Koniecznym jest więc tworzenie po każdej interakcji nowej `HTMLCollection`. Podstrony dyscyplin są statyczne.



Zrzut ekranu 1: Widok aplikacji przed wybraniem konkretnych igrzysk



Zrzut ekranu 2: Lista sportów po wybraniu konkretnych igrzysk

```
> document.querySelectorAll('[class^=styles__ItemButton]')
< NodeList(106) [button.styles__ItemButton-sc-111k07d-4.cGOMnI,
  button.styles__ItemButton-sc-111k07d-4.cGOMnI,
  button.styles__ItemButton-sc-111k07d-4.cGOMnI,
```

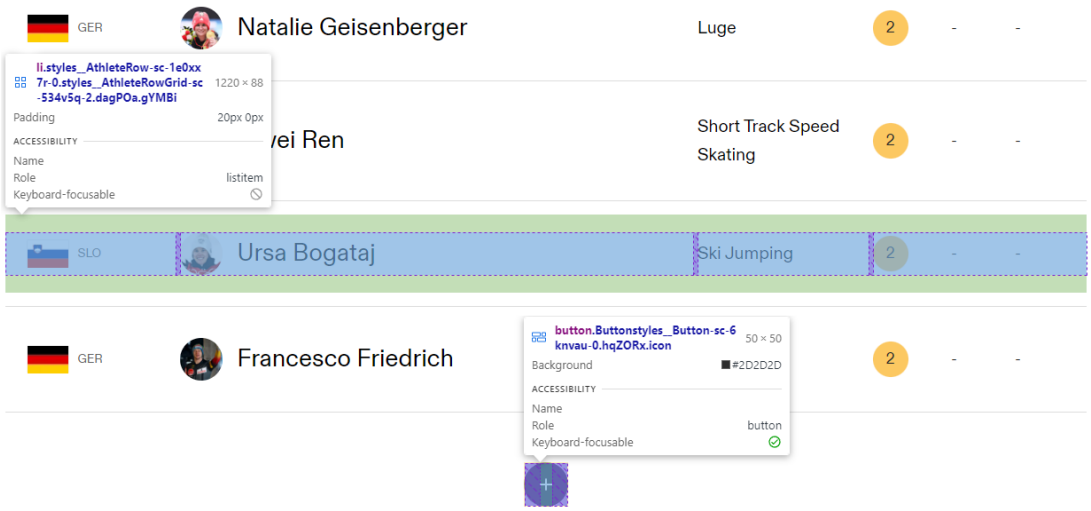
Zrzut ekranu 3: Lista elemntów poszczególnych IO przed interakcją

```
> document.querySelectorAll('[class^=styles__ItemButton]')
< ▼ NodeList [] i
    length: 0
```

Zrzut ekranu 4: Lista elemntów poszczególnych IO po interakcji

2.2 Strona z wynikami medalistów

Na stronie z wynikami medalistów znajduje się aplikacja z listą zawodników, która przy każdym doładowaniu powiększa się o kolejnych dziesięciu atletów. Guzik do ładowania dalszej części listy (oczywiście) jest niszczony i później tworzony nowy. Aplikacja ma buga, który powoduje ponowne załadowanie ostatnio pobranych atletów przy zmianie karty, jednak nie wpłynie on na działanie scrapera. Podstrony z danymi sportowców są statyczne, natomiast ilość wierszy tabeli z danymi nie zawsze jest równa i nie mają one charakterystycznych klas, więc trzeba przeiterować przez wszystkie w poszukiwaniu interesujących nas danych.



Zrzut ekranu 5: Fragment listy zawodników

```
> document.querySelector('[class^="Buttonstyles__Button"]')
<   ▶ <button data-cy="more-button" type="button" class="Buttonstyles_
> document.querySelectorAll('[class^="styles__AthleteRow"]').length
< 20
> document.querySelector('[class^="Buttonstyles__Button"]').click()
< undefined
> document.querySelector('[class^="Buttonstyles__Button"]')
< null
> document.querySelectorAll('[class^="styles__AthleteRow"]').length
< 30
```

Zrzut ekranu 6: Zachowanie strony przy doładowywaniu kolejnych zawodników

3 Działanie scraperów

3.1 Wykorzystane technologie

Wszystkie scrapery oparte są na tych samych technologiach - środowisku Node.js v17.9.0 oraz bibliotece Puppeteer v13.5.2. Ten z kolei korzysta z silnika przeglądarkowego Chromium 100.0.4889.0.

3.2 Proces scrapowania wszystkich wyników

Ze względu na specyfikę aplikacji webowej z wynikami Igrzysk Olimpijskich i rozmieszczenie danych na tysiącach podstron, scrapowanie zostało podzielone na dwa etapy.

1. Zadaniem skryptu `linkRetriever.js` było przejście przez *drzewo* aplikacji głównej strony z wynikami wykorzystując algorytm DFS, zbierając macierz danych z wyszczególnionymi kolumnami `[Host, Rok, Typ, Sport, Dyscyplina, Link]`. Przez konieczność wykonania całego skrapowania wewnątrz skryptu ewaluacyjnego, przerwanie działania skryptu przed ukończeniem skutkuje utratą wszystkich danych. Skutkiem działania jest plik `eventlinks.csv` z 6575 wierszami danych.

2. Skrypt `scoreRetriever.js` iteruje przez linki zebrane przez poprzedni skrypt i rozrzesza zbiór danych o kolumny `[Miejsce, Państwo, Sportowiec, Wynik, Przypisy]`. Poszczególne strony zawierają jedynie zawartość statyczną, dzięki czemu zebrane dane można zapisywać w trakcie scrapowania, a w razie błędów po drodze nie traci się dotychczasowych postępów scrapowania. Pomocna funkcjonalność biorąc pod uwagę, że plik wynikowy, `scoresFromAllGamesNotParsed.csv`, ma ostatecznie 162473 wierszy. Skrypt ten nie zajmuje się formatem danych.

3.3 Formatowanie wyników

Formatem plików zajmuje się moduł `scoreFormatter.js`. Dla łatwiejszej analizy danych zastosowano poniższe przekształcenia:

- zamiana kodów państw na pełne nazwy
- zamiana nazw medali na numer miejsca
- usunięcie = z numerów miejsc zajętych ex aequo
- kapitalizacja pierwszych lilter nazwisk sportowców
- wyniki w formacie *min:sek.setnesek* na *sek.setnesek*
- jeśli istniał przypis tłumaczący brak wyniku, to zostawał wpisany w miejsce takowego

Oczywiście setki różnych dyscyplin miały setki różnych sposobów na zapis wyniku i setki sposobów na ich interpretację. Analiza i przekształcanie wszystkich rodzajów wyników wykroczyłaby poza ramy czasowe tego projektu. Wykonano tylko format jednego z częściej używanych zapisów jako *proof of concept*.

3.4 Proces scrapowania zawodników

Tak samo jak dla wyników wszystkich dyscyplin, strona z medalistami poszczególnych IO wykorzystuje mechanizm dynamiczengo doładowywania zawartości w trakcie przeglądania. Analogicznie więc scrapowanie zostało podzielone na dwa etapy, jednak przez mniejszą objętość danych zajmuje się nimi jeden skrypt - `athleteRetriever.js`. Jest on zdolny do scrapowania danych ze wszystkich nowożytnych IO, ale trzeba podmienić link do właściwej strony.

1. Pierwszy etap polega na rozwinięciu całej tabeli wyników, która przy każdym wciśnięciu guzika powiększa się o 10 wierszy. Cała tabela zostaje zwrócona ze skryptu ewaluacyjnego w postaci macierzy o kolumnach `[Kraj, Sportowiec, Sport, Złoto, Srebro, Brąz, Link]`.
2. Teraz iterując po wierszach macierzy przechodzimy po podstronach każdego ze sportowców, zbierając stamtąd kolejne dane. Macierz zostaje powiększona o kolumny `[Wzięto udział, Rok urodzenia]` i zapisano do pliku `medalistsYEARNotParsed.csv`.

Format plików jest dostosowany do kolumn nowej tabeli natomiast proces przebiega tak samo, jak wyżej, więc nie będę go ponownie opisywał. Sformatowany plik zostaje zapisany jako `medalistsYEAR.csv`.

3.5 Trudności i ograniczenia

Podstawowym problemem jaki stanowiła oficjalna strona z wynikami Igrzysk Olimpijskich jest ilość dynamicznego contentu i mutacji strony jakie zachodzą przy każdym kliknięciu. Ten problem rozwiązano przez zastosowanie MutationObservera, który nasłuchiwał zmian na stronie - w pierwszym scraperze jakichkolwiek, a w drugim konkretnie pojawienia się guzika do dalszego ładowania tabeli.
Kolejnym ograniczeniem była funkcjonalność metody `Page.evaluate()`, czy raczej właśnie brak funkcjonalności. Funkcja zwrotna podawana jako pierwszy argument metody jest wstrzykiwana do strony jako skrypt ewaluacyjny. Jeśli ta funkcja przyjmuje parametry pochodzące z Node'owego środowiska, to należy je podać jako kolejne argumenty metody - również zostaną wstrzyknięte do przeglądarki i skrypt ewaluacyjny będzie

miał do nich dostęp. Nie dotyczy to niestety referencji do funkcji - te nie zostają przeniesione. Z tego względu funkcje wykorzystywane w skrypcie ewaluacyjnym musiały zostać zdefiniowane wewnątrz callbacka, tworząc tzw. *pyramid of doom*.
Oczywistym jest również konieczność korzystania w takim zastosowaniu z asynchroniczności JavaScriptu. Wykorzystywałem do tego głównie Promise'y oraz składnię async/await wprowadzone kolejno w ECMAScript 6 i 2017 (w międzyczasie zmieniono sposób nazywania nowych wydań). Mimo tego nie udało mi się uciec od tzw. *callback hell* - w rekordowym miejscu występuje 6 zagnieżdżeń funkcji zwrotnych. Trzeba jednak przyznać, że zdecydowanie jest to nietypowe zastosowanie składni JSa.

4 Analiza zbioru

Technologie wykorzystywane do analizy to Jupyter Notebook z iPythonem 3.10.0. W głównej mierze wykorzystywano biblioteki NumPy, Pandas, Matplotlib. Przeprowadzono analizę i wizualizację na danych z utworzonego zbioru, ale w niektórych punktach odnoszono się także do dodatkowych źródeł (wszystkie załączone w bibliografii). Po dokładny opis analizowanych zagadnień odsyłam do pliku `allGamesAnalysis.ipynb` lub `analizaWynikowIO.html`. Wykorzystywane środowisko pozwala na integrację kodu z opisami i wizualizacją, w związku z czym nie chcę bez sensu powtarzać tu treści.

5 Refleksje

Dane zawarte na oficjalnych stronach IO wbrew oczekiwaniom są bardzo niespójne, konwencje nazewnicze sportów i dyscyplin różnią się w latach, tak samo formaty zapisu wyników. Tych w starszych Igrzyskach często zwyczajnie brakuje. W różnych miejscach aplikacji podawane są pełne nazwy państw biorących udział, a w innych skrótowce, w dodatku niezgodne ze standardem ISO 3166. Wszystko to znacząco utrudniało przygotowanie zbioru danych, jak również jego analizę.
Dzięki temu uważam, że projekt dał mi solidne zrozumienie podstaw akwizycji danych i problemów jakie można napotkać przy web-scrapingu - zwłaszcza, że większość udało mi się rozwiązać i ujednolicić zbiory danych. Z projektu wynoszę dobre zrozumienie działania asynchroniczności w JavaScripcie, a także funkcjonalności biblioteki Puppeteer.
Do analizy danych odnoszę wrażenie równie dobrze nadałby się tu SQL - tak naprawdę to w większości przypadków naśladowałem atrybutem `pd.DataFrame.loc` zachowanie `WHERE`. Jupyter Notebook ma jednak tą przewagę, że analiza, wizualizacja i opis słowny danych może znajdować się w tym samym dokumencie, co znacznie ułatwia tworzenie takich analiz i późniejsze ich czytanie.
Nie zdążyłem jednak wszystkich swoich pomysłów z analizy danych wdrożyć w życie. Przykładowe zależności, na których zbadanie brakło mi czasu:

- histogramy czasów dla jakiejś wybranej dyscypliny, zmiana wyników wraz z upływem lat
- ilość medalistów w zależności od kontynentu
- wpływ dopingu na wyniki rosyjskich sportowców na przestrzeni lat (zrobiłem tylko dla stałego punktu w czasie)
- zależność między wiekiem sportowców, a ilością zdobytych medali

Wierzę jednak, że gdybym przeanalizował te punkty, wymyśliłbym kolejne i tak mógłbym w kółko, a analiza, którą przeprowadziłem rozpatruje ciekawe aspekty zebranych danych. Konkludując, uważam że cele projektu zostały spełnione w zadowalającym stopniu, a ja sam wynoszę z niego znajomość nowych technologii.

6 Bibliografia i źródła wykorzystywane w projekcie

1. Oficjalna strona IO z wynikami
2. Oficjalna strona IO z medalistami
3. Dokumentacja Puppeteera
4. Dokumentacja Pandas
5. Dokumentacja Pyplot
6. Populacja państw członkowskich ZSRR, stan na 1989
7. Państwa członkowskie ZSRR i daty wystąpień