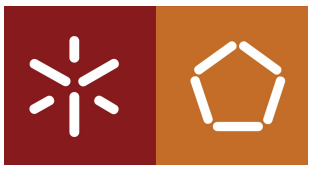

Introdução aos Algoritmos e à Programação

Projeto de Web Scraping com Dados Online

Relatório Prático

Ana Cláudia Andrade de Oliveira (PG55613)

23 de junho de 2025

Índice

Capítulo 1

Introdução

Este trabalho surge no âmbito da cadeira de 1º ano do Mestrado em Matemática e Computação "Introdução aos Algoritmos e Programação". Nesta UC, foi-nos proposta a captura de dados de um *site* à escolha através de um *webscraper* em Python, seguido de uma série de análises e posterior publicação das mesmas.
Foi escolhido como tema de análise os resultados de vários torneios de *Esports* como fuga aos temas mais "comuns"e devido ao meu interesse pessoal pelos mesmos. *ESports* trata-se de uma (ou várias) competições de jogos eletrónicos como Overwatch, Valorant, Dota2, League of Legends, entre outros, onde jogadores por todo o mundo conseguem competir uns contra os outros por prémios monetários.
No mundo digital em que vivemos, a capacidade de transformar dados dispersos em informações úteis é essencial, e este projeto abraçou tal desafio com uma abordagem cuidadosamente planeada. Através do uso de ferramentas como o Python, pacotes como Selenium e BeautifulSoup, e com a construção de uma plataforma acessível através da framework Flask, foi possível criar um produto que não só agrega valor ao conteúdo analisado, mas também propõe um modelo funcional para futuras explorações.
Desde a escolha inicial do site-fonte até às análises gráficas para representar as performances das equipas, cada etapa foi planeada numa tentativa de explorar ao máximo o potencial dos dados recolhidos.

Capítulo 2

Metodologia

Este capítulo descreve a metodologia adotada para a execução do projeto, focando-se na esquematização do trabalho antes do desenvolvimento da *script* de Python, abrangendo a escolha do site-fonte, o processo de extração e tratamento dos dados, e o planeamento para a futura construção do *website* que apresentará as informações extraídas.

2.1 Estudo do *website* de Recolha de Informação

A recolha de dados foi um passo crucial no desenvolvimento do projeto, exigindo uma análise detalhada da estrutura do site e a identificação dos elementos que continham as informações relevantes. Antes de iniciar o processo de extração, foi necessário definir os dados a serem capturados e compreender sua organização no código `HTML` do site.

O site `https://www.gosugamers.net/` foi selecionado como fonte dos dados devido à vasta gama de informações disponíveis sobre resultados de torneios de *esports*. Analisando a estrutura do site, detetou-se que os dados pretendidos estavam armazenados na seção *Matches/Results*, acessível diretamente pelo URL `https://www.gosugamers.net/matches/results`.

Numa primeira visita ao site, foram identificados dois pop-ups que interferem na navegação: um relacionado ao consentimento de informações e outro sobre *cookies*. Para lidar futuramente com esses elementos durante o processo de *web scraping*, será necessário desenvolver um comando no código Python que permita carregar automaticamente nos botões de consentimento.

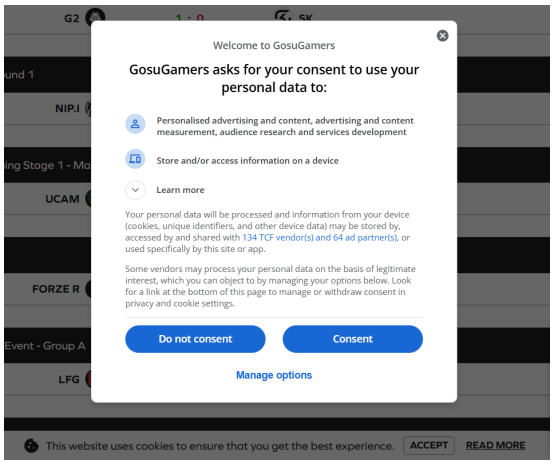

Figura 2.1: Pop-up de consentimento e *cookies* exibidos na entrada do site.

A página de resultados mostrou que as informações sobre todos os jogos estavam distribuídas ao longo de 1.381 páginas[1], contendo 10 caixas por página. Para além de servirem como elementos carregáveis, cada caixa exibia informações resumidas sobre o jogo, incluindo o nome do torneio, a fase, o resultado e os nomes abreviados das equipas. Detalhes adicionais, como nomes de jogadores, personagens utilizados[2], resultados das rondas, países e *rankings* mundiais das equipas estavam apenas disponíveis ao aceder às páginas individuais de cada jogo (carregando em cada caixa).

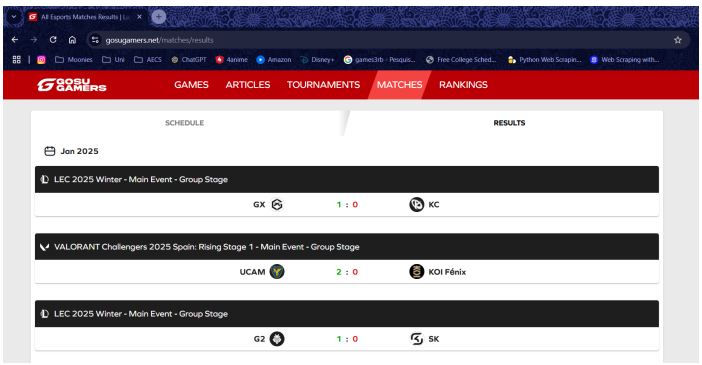

Figura 2.2: Página *Matches/Results*.

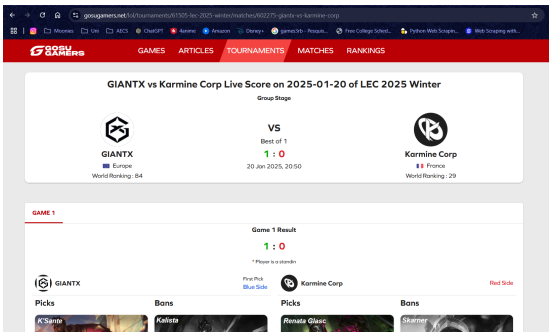

Figura 2.3: Exemplo de informações detalhadas na página de um jogo.

Para futuramente ser possível capturar essas informações, foi realizada uma análise do código `HTML` do site. Identificaram-se características essenciais como classes, XPATHs e *tags* dos elementos que iriam ser necessários, bem como a lógica de interação entre esses elementos. Em especial, destacou-se o uso de links *href* para aceder às páginas detalhadas de cada jogo. Embora os elementos interativos fossem carregáveis, não estavam definidos como botões no `HTML` (à excessão dos botões para mudar de página), pelo que seria necessário capturar os *hrefs* de cada jogo para posteriormente aceder ao conteúdo `HTML` da respetiva página.

Figura 2.4: Localização dos links *href* que direcionam às páginas detalhadas dos jogos.

Com base nesta análise, foram delineadas as etapas iniciais para o desenvolvimento do *scraper*:

1. Aceder ao site.
2. Acionar os botões de consentimento e *cookies*.
3. Para cada página da secção *Matches/Results*, capturar e armazenar os links *href* que direcionam para as páginas detalhadas de cada jogo.

[1] Número de páginas registrado na data de redação deste relatório
[2] Informações disponíveis apenas em jogos como Valorant e League of Legends (LoL)

2.2 Extração, Armazenamento e Tratamento dos Dados

Após a definição das etapas primárias de acesso ao site e a localização da informação pretendida, foi estabelecido um plano para conseguir extrair toda a informação necessária de forma eficiente, armazenando apenas no computador aquilo que seria verdadeiramente utilizado em passos futuros.
Chegou-se à conclusão que a extração dos dados teria de ser feita através de um *loop* onde, para cada uma das páginas, os *hrefs* teriam de ser extraídos e armazenados num dicionário e, posteriormente, seria necessário percorrer cada um desses links e guardar o conteúdo da página `HTML` de cada jogo.
De forma a extraír os *hrefs*, o conteúdo `HTML` de cada página seria guardado temporariamente apenas enquanto a extração era feita.

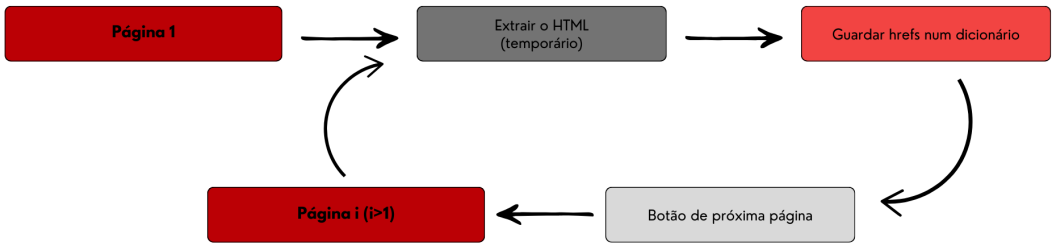

Figura 2.5: Esquema do eventual *loop* de etapas a ser utilizado na extração dos `HTML`'s de cada jogo

Simultaneamente, enquanto que o `HTML` de cada jogo é guardado, toda a informação sobre o mesmo seria armazenada num ficheiro do tipo `JSON`. Esta informação guardada incluiria:

- tipo de jogo (CSGO, Valorant, Dota2, etc.)
- nome da competição, fase, data e resultado
- dados sobre as equipas: nome, país, *ranking* mundial, nomes dos jogadores, personagens escolhidos (caso aplicável)
- mapa onde o jogo foi realizado
- resultados de cada ronda (caso aplicável)

2.2.1 Tratamento dos dados recolhidos

Visto que parte dos objetivos do projeto prático envolve o tratamento dos dados capturados para fornecer resultados que sejam considerados "interessantes", segue uma lista de potenciais estudos que poderiam ser feitos com os dados recolhidos:

- Organização das equipas consoante o seu *ranking* mundial;
- Organização das equipas por país;
- O impacto do *ranking* mundial na performance da equipa (se estas têm tendência a ganhar ou perder, por exemplo);
- Qual o país com mais vitórias/*rankings* mais elevados;
- Gráficos que ilustrem a performance das equipas ao longo do tempo;

- Gráficos que ilustrem os reencontros entre 2 equipas ao longo do tempo;
- etc

Embora nem todas as análises enumeradas acima tenham sido implementadas, todas estas propostas orientaram o planejamento inicial do projeto, garantindo flexibilidade para futuras expansões.

2.3 Construção do *Website*

Por fim, os resultados capturados e os seus respetivos estudos tinham de ser publicados num *website* desenvolvido especificamente para este projeto. A estrutura do site foi planeada previamente e pensada de forma a proporcionar uma navegação intuitiva, com uma apresentação clara das informações.
As principais secções do website planeadas inicialmente incluiram:

- **Página inicial:** refere para que serve o website criado, apresenta o índice de conteúdo com hiperligações para as respetivas páginas;
- **Diferentes organizações das Equipas:** apresenta listas com as diferentes maneiras de visualizar as equipas como, por exemplo, pelo seu *ranking* mundial ou pelo seu país de origem;
- **Página de resultados:** mostra um gráfico para cada equipa, apresentando os resultados das suas partidas ao longo do tempo;
- **Outros estudos dos dados:** inclui possíveis resultados sugeridos na secção anterior.

A hierarquia das informações foi projetada para facilitar o acesso às diferentes camadas de dados, permitindo ao usuário explorar de forma interativa. Gráficos e tabelas serão inseridos posteriormente para melhorar a visualização das estatísticas.

Capítulo 3

Implementação

Este capítulo descreve a implementação prática do projeto, detalhando as etapas desenvolvidas no capítulo anterior em código Python. Serão abordadas as bibliotecas utilizadas, a organização dos scripts, os processos realizados e os principais *outputs* obtidos.
A implementação foi estruturada em três *scripts* principais, que seguem as fases fundamentais do projeto:

- `extrairhtml.py`: Responsável pela extração do conteúdo HTML das páginas dos jogos e pela criação de um arquivo `JSON` contendo os dados organizados.
- `estudodata.py`: Realiza a análise dos dados armazenados no `JSON`, gerando novos arquivos baseados nos estudos planejados na seção de metodologia.
- `app.py`: Implementa o *backend* do website, responsável por publicar os resultados e análises processados.

A divisão dos *scripts* foi definida para garantir uma abordagem clara e funcional, facilitando a futura expansão do projeto. Nos tópicos a seguir, cada etapa da implementação será descrita detalhadamente, evidenciando como as soluções foram desenvolvidas para alcançar os objetivos do projeto.

3.1 Recolha de Dados

Para esta primeira fase foi utilizada, como referido acima, a *script* `extrairhtml.py`. O objetivo principal do seu código era aceder ao site, carregar nos botões de consentimento e *cookies*, extrair o html das páginas de cada jogo e, por fim, armazenar toda a informação importante num único ficheiro `JSON`.
Para tal, utilizámos as bibliotecas `Selenium` e `BeautifulSoup` para navegar, interagir e extrair informação do *site*; `tempfile` e `os` para criar arquivos temporários e manipular o sistema de arquivos e, por fim, `re` e `json` para trabalhar com expressões regulares e armazenar os dados no ficheiro `JSON`.
Abaixo segue-se uma abordagem mais detalhada do que foi realizado em cada uma das etapas deste *script*:

3.1.1 Botão de Consentimento e *Cookies*

A primeira etapa consistiu em aceder ao *site* através do url `https://www.gosugamers.net/matches/results` e carregar em ambos os *pop-ups* de consentimento de informação. Consoante a informação obtida no Capítulo 2, tais *pop-ups* apenas surgem após descer um pouco na página, pelo que realizámos um comando que assim o fizesse.

```python
# Abrir o site no Firefox
driver = webdriver.Firefox()
url = "https://www.gosugamers.net/matches/results"
driver.get(url)
driver.maximize_window()
time.sleep(2)

driver.execute_script("window.scrollTo(0, document.body.scrollHeight / 2);") #scroll até meio da página
time.sleep(2)
```

Figura 3.1: Código de acesso ao *website*

De seguida, com o auxílio do `WebDriverWait` e da análise do `HTML` da página realizada no capítulo anterior, esperou-se que os elementos pretendidos ficassem visíveis (isto é, que o elemento com o `XPATH` característico aparecesse no código `HTML`). Mal estivessem visíveis, o programa foi programado para eperar 2 segundos e, de seguida, carregar nos botões de consentimento
Para ter a certeza que o processo é bem sucedido, o programa devolve "Conkentimento/*Cookies* aceites com sucesso". Caso contrário, surge uma mensagem de erro.

```python
#Botão Consentimento
try:
    consent_button = WebDriverWait(driver, 10).until(
        EC.visibility_of_element_located((By.XPATH, "/html/body/div[7]/div[2]/div[2]/div[3]/div[2]/button[1]"))
    )  # espera até aparecer o botão de consentimento
    time.sleep(2)
    consent_button.click()
    print("Consentimento aceite com sucesso.")
except Exception as e:
    print(f"Erro ao tentar aceitar o consentimento: {e}")
time.sleep(1)

# Botão Cookies
try:
    cookies_button = WebDriverWait(driver, 10).until(
        EC.visibility_of_element_located((By.XPATH, "/html/body/div[3]/div[1]/div[2]/div[2]/button"))
    )  # espera até aparecer o botão de cookies
    cookies_button.click()
    print("Cookies aceites com sucesso.")
except Exception as e:
    print(f"Erro ao tentar aceitar os cookies: {e}")
time.sleep(3)
```

Figura 3.2: Código responsável por carregar automaticamente nos botões de consentimento e *cookies*

3.1.2 Extração e Armazenamento dos *href*

Estamos agora perante um acesso sem restrições a todo o conteúdo do *site*, pelo que, continuando o seguimento das etapas definidas no Capítulo 2, temos como objetivo encontrar e guardar todos os elementos *href* remetentes às páginas individuais de cada jogo realizado.
Devido ao grande número de páginas (1381) e, por consequência, de resultados ($10 \times 1381 = 13810$), os custos computacionais e de tempo necessários para extrair todos os 13810 *links* e, posteriormente, páginas `HTML` seriam bastante superiores àqueles dispostos. De maneira a aliviar este peso, foram utilizadas apenas as primeiras 300 páginas que, apesar de serem apenas uma fração do número original, continuam a ser uma grande base de dados para trabalhar (com $300 \times 10 = 3000$ resultados).
Começámos por definir o dicionário `links_dict{}`, que terá a função de armazenar os 10 links de cada página. Depois disso, a extração dos *links* foi dividida em 2 etapas distintas: a extração dos links da primeira página e das 299 seguintes.

Apesar de ser realizada em 2 passos, a lógica por detrás de cada um é a mesma, à excessão de uma diferença. Para cada página:

- **(exclusivo para as páginas 2-300)** carrega-se no botão que prossegue para a próxima página;
- o seu `HTML` é guardado temporariamente;
- através do comando `find_all`, são identificados todos os elementos do tipo `<a>` e classe específica (estes elementos correspondem a cada caixa vista no Capítulo 2);
- é criada uma secção dentro do dicionário original chamada `page_i_links` onde cada link é adicionado como `link_j`.
- o html temporário é removido;

```
65  # Página 1 ----------------------------------------------------------------
66  html_content = driver.page_source
67  with tempfile.NamedTemporaryFile(delete=False, mode='w', encoding='utf-8') as html_page:
68      html_page.write(html_content)
69      temp_htmlpage = html_page.name  # Nome do html temporário
70
71  # Leitura do conteúdo do html temporário
72  with open(temp_htmlpage, "r", encoding="utf-8") as file:
73      soup = BeautifulSoup(file, "html.parser")
74      a_element = soup.find_all("a", class_="MuiButtonBase-root MuiCardActionArea-root mui-8eq4vn")
75      #a-element permite-nos identificar os hrefs
76      page_1_links = {}
77      for idx, a in enumerate(a_element):
78          href = a.get('href')
79          if href:
80              page_1_links[f'link_{idx + 1}'] = href
81
82  links_dict['page_1'] = page_1_links
83  os.remove(temp_htmlpage)
```

Figura 3.3: Código de extração dos *hrefs* para a primeira página

```
85  # Página 2+ ----------------------------------------------------------------
86  for page in range(299):
87      wait = WebDriverWait(driver, 10)
88      next_page = wait.until(EC.element_to_be_clickable((By.XPATH, "//button[@aria-label='Go to next page']")))
89      print(f"A carregar página {page + 2}...")
90      next_page.click()
91      time.sleep(2)
92
93      html_content = driver.page_source
94      with tempfile.NamedTemporaryFile(delete=False, mode='w', encoding='utf-8') as html_page:
95          html_page.write(html_content)
96          temp_htmlpage = html_page.name
97
98      with open(temp_htmlpage, "r", encoding="utf-8") as file:
99          soup = BeautifulSoup(file, "html.parser")
100         a_element = soup.find_all("a", class_="MuiButtonBase-root MuiCardActionArea-root mui-8eq4vn")
101
102         page_links = {}
103         for idx, a in enumerate(a_element):
104             href = a.get('href')
105             if href:
106                 page_links[f'link_{idx + 1}'] = href
107
108         links_dict[f'page_{page + 2}'] = page_links
109
110     os.remove(temp_htmlpage)
```

Figura 3.4: Código de extração dos *hrefs* das páginas 2-300

No final destas iterações, o dicionário deverá ter a seguinte estrutura:


```
{'page_1': {'link_1': '/counterstrike/tournaments/61576-cct-season-2-european-series-16/mat
ches/604732-fnatic-vs-fire-flux-esports', 'link_2': '/valorant/tournaments/61571-valorant-c
hallengers-2025-north-east-stage-1/matches/605960-unsigned-vs-formulation-gaming', 'link_3'
: '/valorant/tournaments/61584-valorant-challengers-2025-turkiye-birlik-kickoff-split/match
es/604358-fire-flux-esports-vs-fenerbahce-esports', 'link_4': '/valorant/tournaments/61584-
valorant-challengers-2025-turkiye-birlik-kickoff-split/matches/604379-eternal-fire-vs-digit
al-athletics', 'link_5': '/counterstrike/tournaments/61605-united21-season-26/matches/60594
8-copenhagen-wolves-vs-viperio', 'link_6': '/counterstrike/tournaments/60952-esl-challenger
-league-season-49/matches/605004-chinggis-warriors-vs-fengda-gaming', 'link_7': '/counterst
rike/tournaments/60952-esl-challenger-league-season-49/matches/604982-unsettled-resentment-
vs-the', 'link_8': '/counterstrike/tournaments/60952-esl-challenger-league-season-49/matche
s/605030-72c-vs-the-qube-esports', 'link_9': '/counterstrike/tournaments/61591-exort-series
-7/matches/604671-fire-flux-esports-vs-preasy-esport', 'link_10': '/counterstrike/tournamen
ts/61604-cct-season-2-european-series-17/matches/605890-zero-tenacity-vs-adventurers'}, 'pa
```


Figura 3.5: *Output* do dicionário `links_dict` após a extração de todos os *hrefs*

3.1.3 Extração dos HTML e Criação do Ficheiro JSON

Finalmente, com todos os *hrefs* armazenados, o programa cria um "url base", neste caso `https://www.gosugamers.net/` e, para cada link do dicionário, vai anexar ao url base o *href* e armazenar a respetiva página `HTML` numa pasta "Página _", dentro de uma outra pasta "Matches".
Ao mesmo tempo que a captura dos `HTML` é feita, o programa retira a informação do torneio da mesma página `HTML` e armazena num novo dicionário, que virá a ser o `JSON` com toda a informação necessária para as secções seguintes. Toda a informação na figura abaixo é possível extraír dessa forma, à excessão do jogo onde foi feita a partida, que devido à sua ausência no `HTML`, tem de ser extraída a partir do link da página.

```
201            tournament_data = {
202                "Game": game_name,
203                "Tournament": tournament_name,
204                "Tournament Phase": tournament_phase,
205                "Date": tournament_date,
206                "Team 1": {
207                    "Name": team1,
208                    "Country": country1,
209                    "World Ranking": ranking1
210                },
211                "Team 2": {
212                    "Name": team2,
213                    "Country": country2,
214                    "World Ranking": ranking2
215                },
216                "Score": score.strip()
217            }
```

Figura 3.6: Estrutura dos dados do ficheiro `JSON`

3.2 Tratamento dos Dados

Após a captura de toda a informação essencial para o ficheiro "`tournaments_data.json`", foi criada uma nova *script* de Python para estudar esses novos dados e tratá-los de uma forma que fosse considerada "interessante"para este estudo das equipas.

Esta nova script, "`estudodata.py`"teve como principal objetivo organizar a informação de várias maneiras distintas, utilizando as bibliotecas `json`, `collections` e `os`.
Iniciamos esta *script* com a criação de uma pasta "jsons"que serviria de local para armazenar os novos ficheiros `JSON` que seriam criados. O primeiro método de organização escolhido foi talvez o mais lógico, tendo em conta a base de dados: organizar as equipas pelo seu *ranking* mundial.
Para isso, foi criado um novo dicionário `teams_by_game`, com a função de armazenar as equipas de cada jogo. De seguida, o programa percorreu todos os torneios do arquivo `JSON` inicial, extraíndo o nome de cada jogo, os *rankings* das equipas e os seus respetivos nomes.
Os nomes repetidos foram depois removidos com o auxílio de um dicionário temporário `unique_teams` e, por fim, os *rankings* organizados foram guardados no arquivo `ranking_list.json`

```python
for torneio in dados:
    game = torneio['Game']

    if game not in teams_by_game:
        teams_by_game[game] = []

    # Adicionar as equipas e seus rankings ---
    team1_ranking = torneio['Team 1'].get('World Ranking', float('inf'))
    team2_ranking = torneio['Team 2'].get('World Ranking', float('inf'))

    teams_by_game[game].append({
        "Name": torneio['Team 1']['Name'],
        "World Ranking": team1_ranking
    })
    teams_by_game[game].append({
        "Name": torneio['Team 2']['Name'],
        "World Ranking": team2_ranking
    })

# Remover equipas repetidas e organizar por ranking ---
for game in teams_by_game:
    unique_teams = {}
    for team in teams_by_game[game]:
        name = team['Name']
        rank = team['World Ranking']
        if name not in unique_teams or rank < unique_teams[name]:
            unique_teams[name] = rank

    teams_by_game[game] = sorted(unique_teams.items(), key=lambda x: x[1])
```

Figura 3.7: Código da organização das equipas pelo seu *ranking* mundial


```
{
    "dota2": [
        [
            "PARIVISION",
            1
        ],
        [
            "BetBoom Team",
            2
        ],
        [
            "Tundra Esports",
            3
        ],
        [
            "Team Liquid",
            4
        ],
        [
            "Xtreme Gaming",
            6
        ],
        [
            "Gaimin Gladiators",
            7
        ],
        [
            "Team Spirit",
            8
        ]
```


Figura 3.8: Estrutura do `JSON` criado

O segundo método de organização da informação escolhido foi organizar as equipas consoante o seu país de origem. Começámos por criar um dicionário `teams_by_coutry` e, para cada partida no `JSON` original, os nomes e países de cada equipa foram extraídos e adicionados à respetiva lista. Finalmente, foram removidas as equipas repetidas e as listas dos países organizadas por ordem alfabética.

```python
for torneio in dados:
    game = torneio['Game']

    if game not in teams_by_game:
        teams_by_game[game] = []

    # Adicionar as equipas e seus rankings ---
    team1_ranking = torneio['Team 1'].get('World Ranking', float('inf'))
    team2_ranking = torneio['Team 2'].get('World Ranking', float('inf'))

    teams_by_game[game].append({
        "Name": torneio['Team 1']['Name'],
        "World Ranking": team1_ranking
    })
    teams_by_game[game].append({
        "Name": torneio['Team 2']['Name'],
        "World Ranking": team2_ranking
    })

# Remover equipas repetidas e organizar por ranking ---
for game in teams_by_game:
    unique_teams = {}
    for team in teams_by_game[game]:
        name = team['Name']
        rank = team['World Ranking']
        if name not in unique_teams or rank < unique_teams[name]:
            unique_teams[name] = rank

    teams_by_game[game] = sorted(unique_teams.items(), key=lambda x: x[1])
```

Figura 3.9: Código da organização das equipas pelo seu país de origem


```
{
    "dota2": [
        [
            "PARIVISION",
            1
        ],
        [
            "BetBoom Team",
            2
        ],
        [
            "Tundra Esports",
            3
        ],
        [
            "Team Liquid",
            4
        ],
        [
            "Xtreme Gaming",
            6
        ],
        [
            "Gaimin Gladiators",
            7
        ],
        [
            "Team Spirit",
            8
```


Figura 3.10: Estrutura do `JSON` criado

O mesmo raciocínio foi aplicado nas 2 próximas abordagens de organização dos dados, pelo que não vale a pena repetir os procedimentos a ser tomados. Falamos da listagem dos reencontros entre duas equipas (caso estas se tenham encontrado mais do que 1 vez) e a listagem de todos os jogos realizados pela mesma equipa. As respetivas linhas de código relativas a estas abordagens e estruturas dos `JSON` criados encontram-se nas 4 figuras abaixo:

```python
reencounters_dict = defaultdict(list)

for match in dados:
    team1 = match['Team 1']['Name'].strip()
    team2 = match['Team 2']['Name'].strip()
    game_key = tuple(sorted([team1, team2]))

    reencounter_data = {
        "Game": match["Game"],
        "Tournament Name": match["Tournament"],
        "Tournament Phase": match["Tournament Phase"],
        "Date": match["Date"],
        "Score": match["Score"]
    }

    reencounters_dict[game_key].append(reencounter_data)

# Filtrar apenas os pares com mais de um jogo
filtered_reencounters_list = []
for teams, matches in reencounters_dict.items():
    if len(matches) > 1:
        filtered_reencounters_list.append({
            "Team 1": teams[0],
            "Team 2": teams[1],
            "Matches": matches
        })
```

Figura 3.11: Código da listagem dos reencontros entre duas equipas


```
    {
        "Team 1": "The Immortal",
        "Team 2": "Yangon Galacticos",
        "Matches": [
            {
                "Game": "dota2",
                "Tournament Name": "EPL World Series: Southeast Asia Season 2",
                "Tournament Phase": "Group Stage",
                "Date": "24 Jan 2025, 07:00",
                "Score": "W:FF"
            },
            {
                "Game": "dota2",
                "Tournament Name": "Dreamleague Season 25",
                "Tournament Phase": "Open Qualifier #2",
                "Date": "05 Jan 2025, 08:15",
                "Score": "1:0"
            }
        ]
    },
    {
        "Team 1": "Groomify",
        "Team 2": "Young Blood",
        "Matches": [
            {
                "Game": "dota2",
                "Tournament Name": "EPL World Series: Southeast Asia Season 2",
                "Tournament Phase": "Group Stage",
                "Date": "21 Jan 2025, 07:00",
```


Figura 3.12: Estrutura do `JSON` criado

```python
# Organizar os jogos por equipa
matches_by_team = defaultdict(lambda: defaultdict(list))

for match in dados:
    team1 = match["Team 1"]["Name"].strip()
    team2 = match["Team 2"]["Name"].strip()

    match_data_team1 = {
        "Tournament Name": match["Tournament"],
        "Tournament Phase": match["Tournament Phase"],
        "Date": match["Date"],
        "Opponent": team2,
        "Score": match["Score"]
    }

    match_data_team2 = {
        "Tournament Name": match["Tournament"],
        "Tournament Phase": match["Tournament Phase"],
        "Date": match["Date"],
        "Opponent": team1,
        "Score": match["Score"]
    }

    # Organizar por equipa e depois por jogo
    matches_by_team[team1][match["Game"]].append(match_data_team1)
    matches_by_team[team2][match["Game"]].append(match_data_team2)
```

Figura 3.13: Código da listagem dos reencontros entre duas equipas


```json
{
    "Yangon Galacticos": {
        "dota2": [
            {
                "Tournament Name": "EPL World Series: Southeast Asia Season 2",
                "Tournament Phase": "Group Stage",
                "Date": "24 Jan 2025, 07:00",
                "Opponent": "The Immortal",
                "Score": "W:FF"
            },
            {
                "Tournament Name": "EPL World Series: Southeast Asia Season 2",
                "Tournament Phase": "Group Stage",
                "Date": "19 Jan 2025, 10:00",
                "Opponent": "Groomify",
                "Score": "0:2"
            },
            {
                "Tournament Name": "PGL Wallachia Season 3",
                "Tournament Phase": "Closed Qualifier",
                "Date": "14 Jan 2025, 12:00",
                "Opponent": "Execration",
                "Score": "0:2"
            },
            {
```


Figura 3.14: Estrutura do `JSON` criado

3.3 Construção da Página *Web*

Por fim, com toda a informação organizada nas suas respetivas pastas, foi a altura de criar o terceiro e último *script* de código Python, que servirá como *backend* do nosso *website*, onde neste irão ser publicados os nossos resultados. Este `app.py` será o responsável pelo processamento dos dados, resposta às solicitações do cliente e realiza operações no servidor. Para a sua criação, foi utilizada uma *framework* `Flask`.
Foi nele que definimos os URL's que o utlizador poderá visitar no navegado, estando estas definidas através da linha de código: `@app.route()`. Nestas rotas, destacam-se:

- `@app.route('/')`: página de acesso ao carregar no link
- `@app.route('/ranking_mundial')`: página relativa à organização das equipas consoante o seu *ranking* mundial
- `@app.route('/equipas_por_pais')`: página onde consta a organização das equipas pelo seu país de origem
- `@app.route('/performance')`: página relativa à prestação de cada equipa
- `@app.route('/reencontros')`: página onde constam os reencontros entre duas equipas

Foram definidas ainda rotas extra, indicadas na Figura 3.15 para carregar os diretórios dos ficheiros `JSON` que iriam ser utilizados e até mesmo os próprios ficheiros.

```python
# Caminho para os arquivos JSON
rankings_file = os.path.join('jsons', 'ranking_list.json')
country_file = os.path.join('jsons', 'country_list.json')

# Carregar dados dos rankings
with open(rankings_file, 'r', encoding='utf-8') as f:
    ranking_list = json.load(f)

# Carregar dados das equipas por país
with open(country_file, 'r', encoding='utf-8') as f:
    country_list = json.load(f)

#carregar dados dos torneios por equipa
@app.route('/jsons/<path:filename>')
def serve_json(filename):
    return send_from_directory('jsons', filename)


```

Figura 3.15: Rotas extra definidas em `app.py`

Acompanhado por este *backend*, o *website necessita também de um frontend*, o interface visual que os usuários vêem e interagem diretamente no navegador. Para tal, foram criados 5 ficheiros *html*, cada um remetendo a uma nova página do *website*.

3.3.1 `index.html`

O este ficheiro `HTML` é responsável por estruturar e estilizar a interface da página inicial, organizando o conteúdo e `CSS` para estilizar os elementos visuais.
Na secção inicial, conhecida como `<head>`, são definidos os "metadados"essenciais para a página. O título da aba do navegador é configurado como "Página de Resultados", e uma "meta tag"é utilizada para tornar a página adaptável para diferentes tamanhos de ecrãs. É também especificado o conjunto de caracteres UTF-8, garantindo que a página suporte corretamente caracteres especiais utilizados em português.

```html
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Página de Resultados</title>
```

Figura 3.16: Configuração da `<head>`

Ainda nesta secção inicial, segue uma secção `<style>` onde são detalhados o estilo da página, título, descrição, barra de índice e título das secções.
O corpo da página (`<body>`) contém o conteúdo visível para o utilizador. No topo, é apresentado um título que dá as boas-vindas à página, seguido por uma breve descrição do seu propósito. Essa descrição destaca que as análises apresentadas são baseadas em dados retirados do site gosugamers, com um link que, ao ser carregado, redireciona o utilizador para o site original num novo separador.

Abaixo da descrição, encontra-se uma barra de índice que organiza os links para as diferentes seções do site. Esses links permitem o acesso a páginas específicas que detalham análises, como o **ranking** mundial por jogo, a organização de equipas por país, a análise das performances, e os reencontros entre equipas. A barra de índice está estruturada como uma lista e foi projetada para ser visualmente atrativa e de fácil navegação, com links destacados em azul e sublinhados ao passar o cursor.

```
<body>
    <h1>Bem-vindo à página WEB relativa ao TP3</h1>
    <p class="description">Aqui estarão disponíveis todas as análises feitas aos dados recolhidos da página <a href="https://www.gosugamers.net" target="_blank'

    <!-- Barra de índice -->
    <div class="index-bar">
        <h2>Índice</h2>
        <ul>
            <li><a href="/ranking_mundial">Ranking Mundial de cada jogo</a></li>
            <li><a href="/equipas_por_pais">Equipas Organizadas por País</a></li>
            <li><a href="/performance">Análise das Performances</a></li>
            <li><a href="/reencontros">Reencontros entre Equipas</a></li>
        </ul>
    </div>
</body>
```

Figura 3.17: Configuração do `<body>`

3.3.2 `ranking_mundial.html` e `equipas_por_pais.html`

O conteúdo `<head>` destes `HTML`s seguem a mesma estrutura do anterior. Relativamente ao `<body>`, em `ranking_mundial.html` em cima aparece um título principal marcado por um `<h1>`, seguida pela secção principal da página: caixa tipo `<div>` e classe "*table-container*", onde estão organizadas várias tabelas com os *rankings* mundiais das equipas para cada jogo. Já em `equipas_por_pais.html`, algo semelhante é criado para criar uma classe "*country-table*", onde estão organizadas as várias tabelas com as dependendo da sua nacionalidade.

É importante salientar acerca destas tabelas, que foi adicionado um comando `"overflow-y: auto;"` nas definições de customização da tabela, de maneira a inserir uma barra de *scroll*. Este elemento torna a informação muito mais fácil de encontrar, bem como faz o *site* ter um aspeto bastante mais apelativo.

No final da página, tem uma última secção `<div>` que marca um elemento carregável para voltar ao fim da página. Neste caso, esse elemento é o próximo texto "Voltar ao Índice".

```
<body>
    <h1>Ranking Mundial de cada Jogo</h1>

    <!-- Tabelas de rankings -->
    <div class="tables-container">
        {% for jogo, rankings in ranking_list.items() %}
            <div class="table-wrapper">
                <div class="game-title">{{ jogo }}</div>
                <table id="table-{{ loop.index }}">
                    <thead>
                        <tr>
                            <th>Equipa</th>
                            <th>Ranking</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for equipa, ranking in rankings %}
                            <tr>
                                <td>{{ equipa }}</td>
                                <td>{{ ranking }}</td>
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        {% endfor %}
    </div>

    <!-- Link de Voltar -->
    <div class="back-link">
        <a href="/">Voltar ao Índice</a>
    </div>
</body>
```

Figura 3.18: Configuração do `<body>` de `ranking_mundial.html`

```
<body>
    <h1>Equipas Organizadas por País</h1>

    <!-- Tabelas de equipas por país -->
    <div class="country-table">
        {% for pais, equipas in country_list.items() %}
            <div class="table-wrapper">
                <h4>{{ pais }}</h4>
                <table id="table-{{ loop.index }}">
                    <tbody>
                        {% for equipa in equipas %}
                            <tr>
                                <td>{{ equipa }}</td>
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        {% endfor %}
    </div>
```

Figura 3.19: Configuração do `<body>` de `equipas_por_pais.html`

3.3.3 `performance.html` e `reencontros.html`

Estas duas páginas, ao contrário das outras faladas anteriormente. têm a particularidade de utilizar a biblioteca `Chart.js` para a criação de gráficos dinâmicos e interativos.
O principal objetivo destas páginas era fornecer uma visualização gráfica dos resultados dos jogos ao longo do tempo, quer fosse de uma equipa ou do reencontro entre 2. Devido à grande quantidade de equipas existentes na nossa base de dados, foi necessário construir uma estrutura de página funcional e de clara compreensão.
A página foi então dividida numa barra lateral, onde num lado estavam dispostas todas as equipas (respetivamente, encontros entre 2 equipas), e no outro seriam exibidos os gráficos e uma tabela detalhada com os resultados dos jogos. Para além disso, foi incluida uma barra de pesquisa através do comando `searchBar.addEventListener()`
A funcionalidade interativa da página é implementada com JavaScript. Em primeiro lugar, os dados de cada partida são carregados de um arquivo JSON chamado `tmatches_list.json`. Ao carregar no nome de qualquer equipa, o conteúdo principal é atualizado dinamicamente.
Na área dos gráficos, é gerado um gráfico de linha consolidado com os resultados das partidas, exibindo as datas no eixo X e a diferença de pontos no eixo Y. Esse gráfico é construído utilizando a biblioteca `Chart.js`. Simultaneamente, a tabela abaixo do gráfico é preenchida com os detalhes das partidas da equipa selecionada.

Capítulo 4

Resultados

Dada por completo a abordagem passo a passo do que foi feito no código, falta apenas mostrar os resultados obtidos. À excessão dos ficheiros `JSON`, o elemento de maior relevo a detalhar aqui trata-se do `website` criado ao correr a *script* `app.py`.
A página inicial apresenta um estilo minimalista, que permite ao utilizador vizualizar toda a informação com facilidade. O título e a breve descrição revelam o motivo de criação e quais os conteúdos que poderão ser encontrados, enquanto que o índice surge como elemento de ligação a cada uma das páginas seguintes.

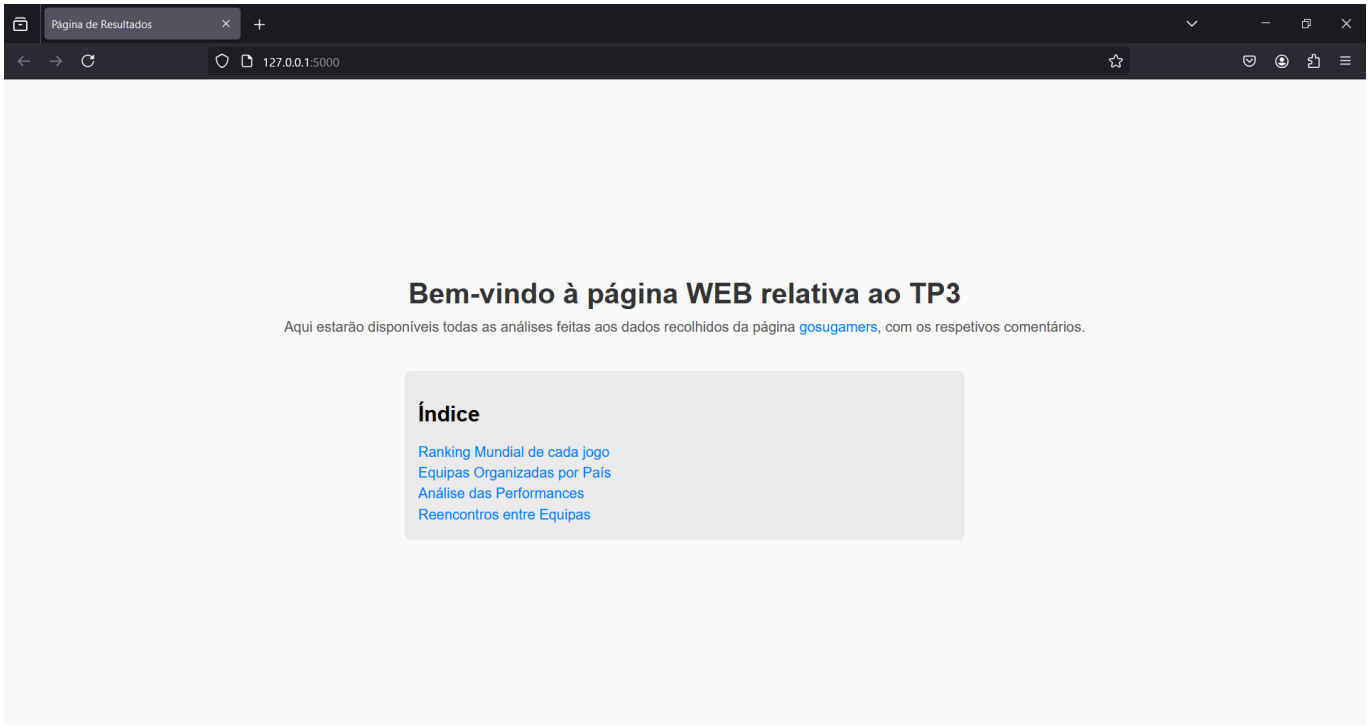

Figura 4.1: Página inicial do *site*

Carregando nas duas primeiras páginas, obtemos os resultados obtidos nas Figuras 4.2 e 4.3 respetivamente: a organização das equipas consoante o seu *ranking* mundial e o seu país de origem. A limitação de conteúdo visível acompanhada de uma barra de *scroll* tornam a página bastante mais acessível, com mais informação disposta em frente aos olhos do utilizador.

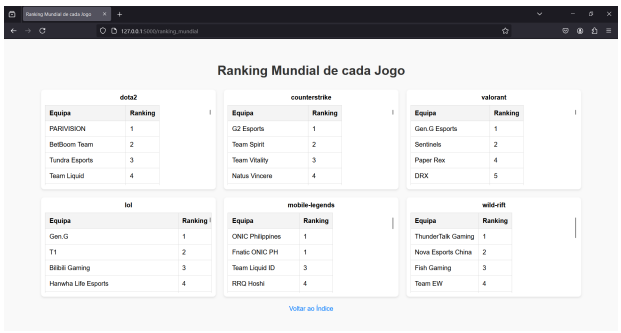

Figura 4.2: Página relativa ao *ranking* mundial de cada jogo

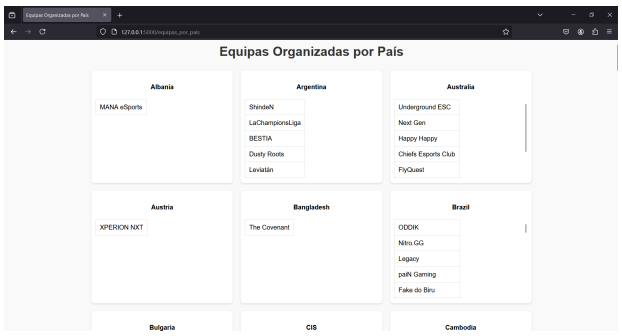

Figura 4.3: Página relativa à organização das equipas por país

De forma a avaliar a performance de uma equipa nas suas partidas, em 1 ou mais jogos, o utilizador pode dirigir-se à terceira página do índice, onde encontrará, à esquerda, uma lista carregável das equipas ou uma barra de pesquisa. Qualquer um destes métodos irá resultar num gráfico interativo, onde cada cor representa um jogo diferente em que a equipa competiu, o eixo das abcissas representa o dia de cada torneio e as ordenadas representam a diferença de pontos entre a equipa pretendida e a adversária. Por fim, em baixo do gráfico foi colocada uma tabela com mais informação acerca de cada jogo.

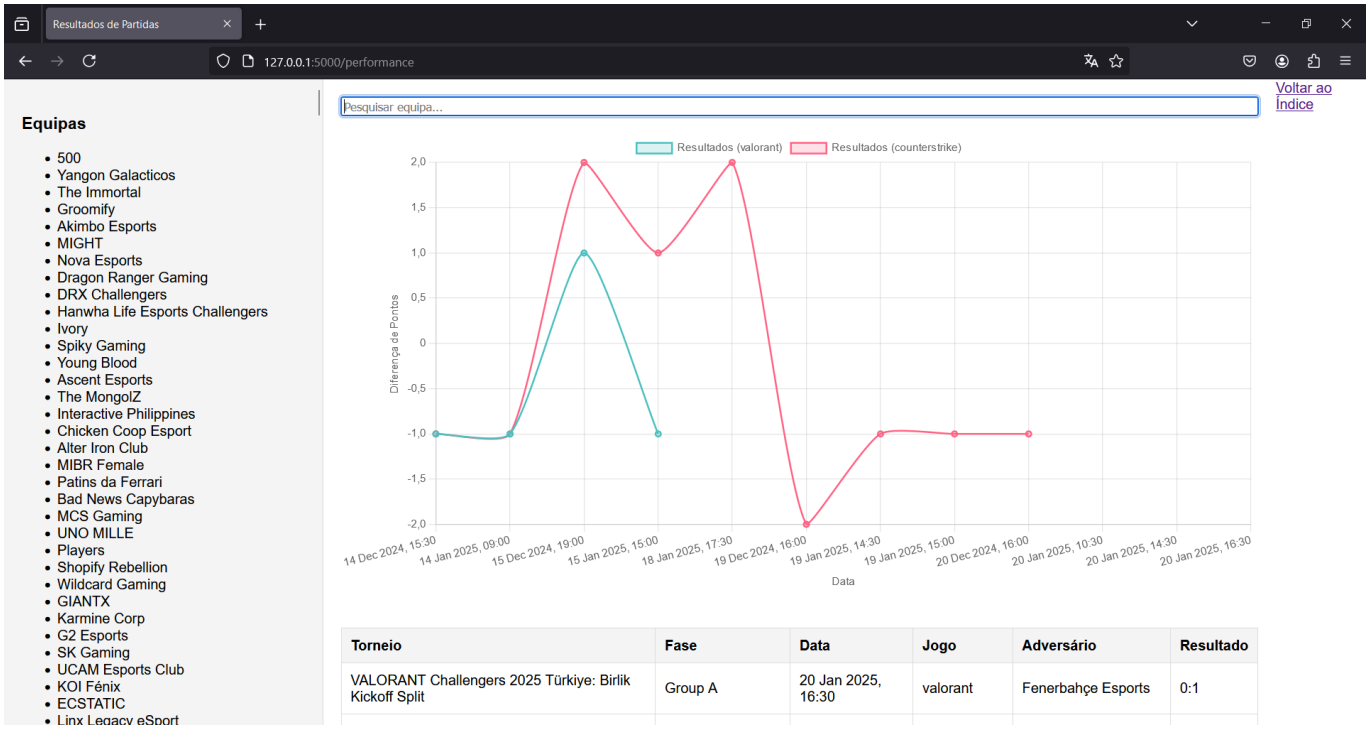

Figura 4.4: Página relativa aos resultados da equipa "Fire Flux Esports"ao longo do tempo

Finalmente, algo semelhante à página anterior foi criado, agora para os reencontros entre 2 equipas. A estrutura da página é a mesma, mas agora a informação disponibilizada na coluna da esquerda trata-se das 2 equipas cujos reencontros queremos analisar.

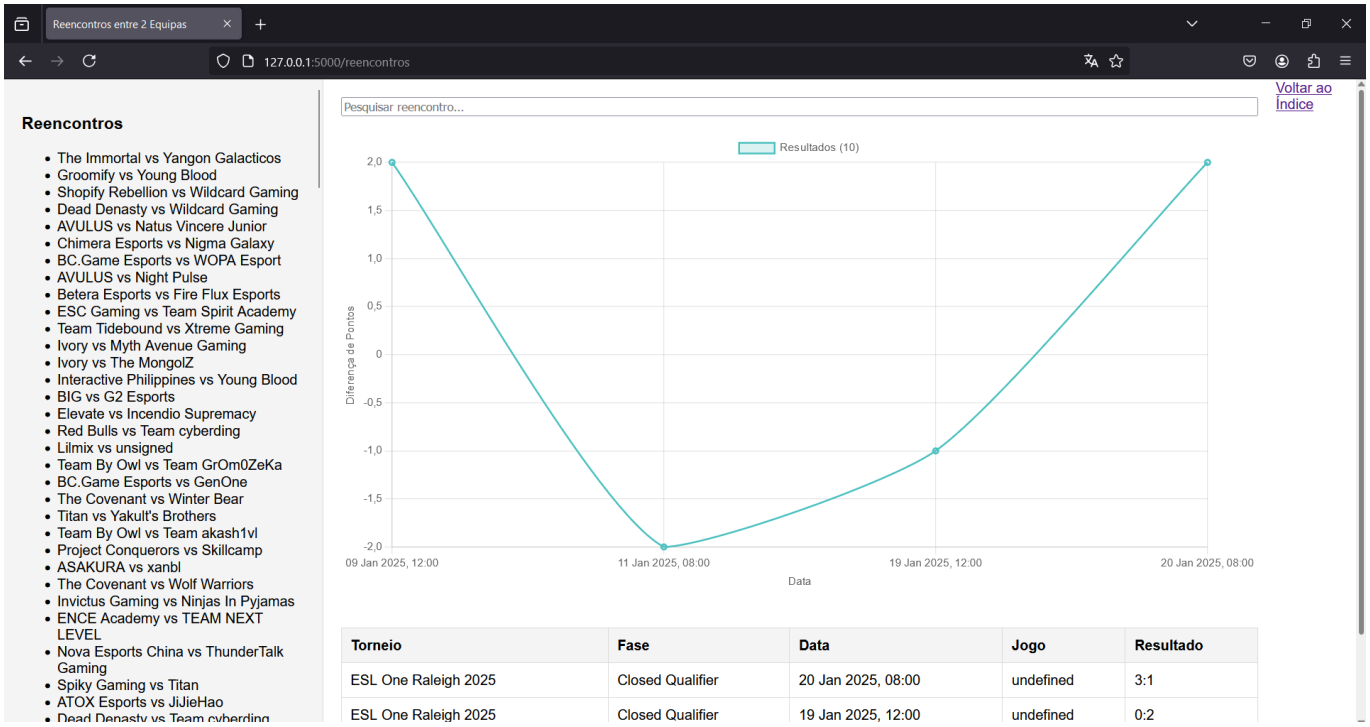

Figura 4.5: Página relativa aos resultados dos reencontros "Team Tidebound vs Xtreme Gaming"

Apesar destes serem os resultados pretendidos, existem certas melhorias a fazer em projetos futuros, maioritariamente em termos de optimização. Para além disso, o website tem dificuldades em reconhecer e marcar o torneio no gráfico quando o resultado é "W:FF".

Capítulo 5

Conclusão

Desde a escolha do *site* original, que exigiu uma análise detalhada da estrutura e da disponibilidade dos dados, até a implementação de soluções para lidar com restrições como pop-ups e várias páginas, este projeto serviu como um grande momento de aprendizagem em várias áreas da programação.
A decisão por limitar a análise às 300 primeiras páginas foi estratégica para viabilizar o trabalho, mas destacou o potencial de expansão para investigações futuras.
Embora várias abordagens tenham sido implementadas, como análises de rankings e desempenho das equipas, outras possibilidades, como a relação entre rankings e geografias ou o impacto de determinados personagens nos jogos permanecem como ideias para trabalhos futuros. Ideias como estas mostram o potencial de evolução do projeto.
No final, o trabalho cumpriu seu objetivo principal: criar um sistema funcional que captura, organiza e apresenta dados de forma intuitiva e visualmente atrativa. Mais do que uma aplicação prática de programação e algoritmos, este projeto testou os meus limites de raciocínio, criatividade, disciplina e paciência, face aos vários desafios enfrentados e soluções encontradas.

Capítulo 6

Webgrafia

1. GosuGamers. Matches/Results - Esports Tournament Data. Disponível em: https://www.gosugamers.net/matches/results.
2. Documentação Selenium. The Selenium Browser Automation Project. Disponível em: https://www.selenium.dev/documentation/.
3. Selenium Web Scraping: How to Scrape Dynamic Sites Step-by-Step. Disponível em: https://www.scraperapi.com/web-scraping/selenium/.
4. Documentação BeautifulSoup. Disponível em: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.
5. Flask Framework. Documentação Oficial. Disponível em: https://flask.palletsprojects.com/.
6. Flask- Creating first simple application. Disponível em: https://www.geeksforgeeks.org/flask-creating-f
7. Chart.js. Gráficos Dinâmicos e Interativos. Disponível em: https://www.chartjs.org/docs/.
8. Python Web Scraping: Full Tutorial with Examples. Disponível em : https://www.scrapingbee.com/blog/web-scraping-101-with-python/.