亞洲大學
ASIA UNIVERSITY

Final Project Report
Advanced Computer Programming

Game Prices Comparison With Flask and Scrapy

Group : Group 6

Instructor : DINH-TRUNG VU

2025-06

Chapter 1 Introduction

1.1 Group Information

1. **Group Project Repository:**
   https://github.com/HanifiSetiawan/ACP-Group-06
2. **Group members**:
   1. Hanifi Abrar Setiawan– 112021224 (leader)
   2. Abiansyah Adzani Gymnastiar– 112021201

1.2 Overview

The project for this is in the form of a web application. The app was designed to scrape and compare game prices across multiple online video game stores. The stores include: Steam, Xbox, Playstation Store, and Epic Games Store. It uses advanced programming tools and libraries, including:

- **Scrapy**: For web scraping and data extraction.
- **Scrapy Playwright/Selenium**: For a more advanced content scraping.
- **Flask**: To create a web app with dynamic pages.
- **Bootstrap**: For a more personalized user interface.
- **XML Parsing**: Game data from each store is stored and loaded using XML files.

The application enables the users to:

- Search for a specific game.
- Compare the prices across different platforms.
- Find stores that have a discount on specific games.

Chapter 2 Implementation

2.1 app.py (Flask)

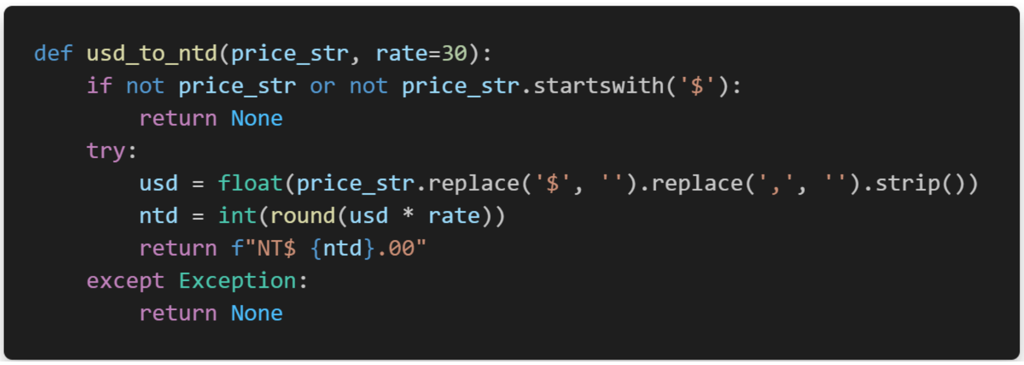
For implementing the website application, Flask was used as the framework. According to their website, Flask is a lightweight WSGI web application framework. It is easy to get started with, and has the ability to scale up to complex applications.

2.1.1 app

Instance of the Flask application, created using Flask(__name__).

2.1.2 usd_to_ntd(price_str, rate=30)

A helper function to easily convert the USD prices to NTD based on a static exchange rate. It must be kept in mind that the conversion rate is not accurate to the realtime conversion rate.



```python
def usd_to_ntd(price_str, rate=30):
    if not price_str or not price_str.startswith('$'):
        return None
    try:
        usd = float(price_str.replace('$', '').replace(',', '').strip())
        ntd = int(round(usd * rate))
        return f"NT$ {ntd}.00"
    except Exception:
        return None
```

Converts a price string in USD (e.g., "$59.99") into NTD using a default exchange rate. The function ensures robustness by handling improperly formatted or missing data gracefully.

2.1.3 load_games_from_xml(xml_path)

Parses an XML file and returns a list of game dictionaries.

```python
def load_games_from_xml(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    games = []
    for item in root.findall('item'):
        game = {child.tag: child.text for child in item}
        games.append(game)
    return games
```

This function parses a given XML file and extracts <item> elements, converting them into Python dictionaries. This allows structured data to be passed into the template for rendering.

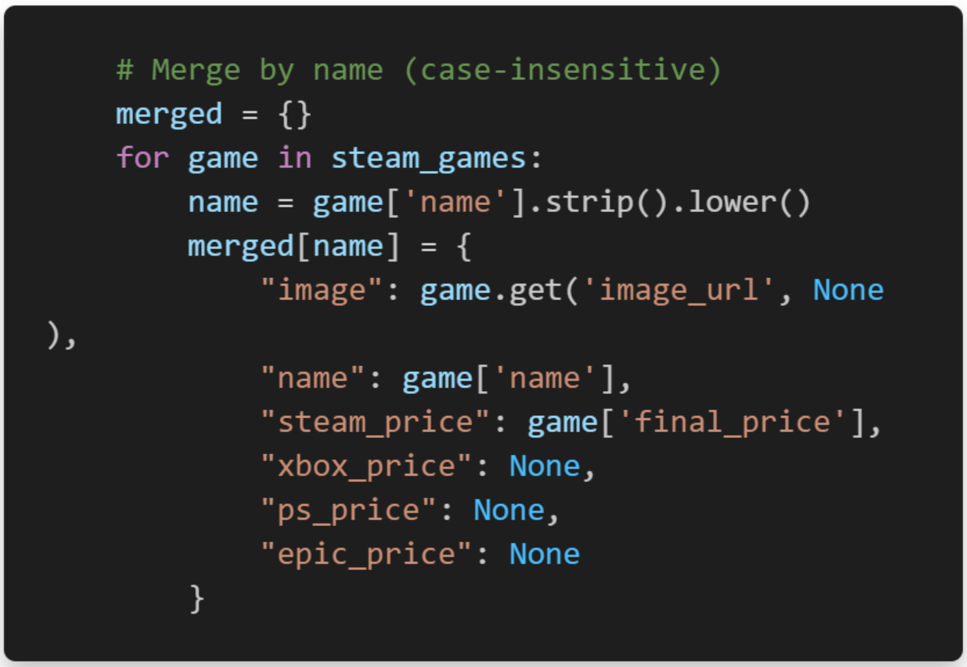
2.1.4 index()

The main route of the web. Features implemented ranges from loading the game data from XML files to implementation of pagination using the 'page' parameter. The full implementations are as follow:

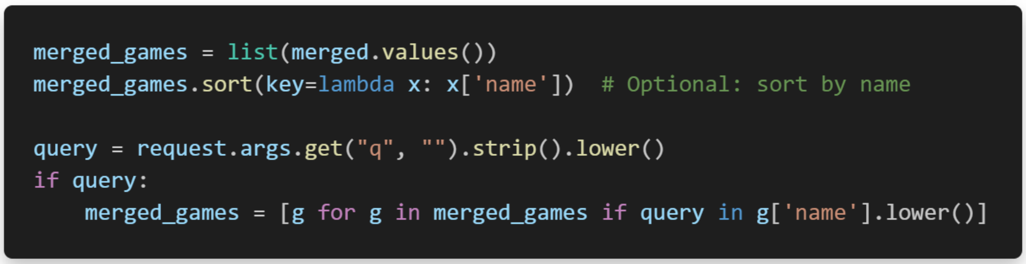
- Loads game data from XML files.

```python
    steam_games = load_games_from_xml('steam_output_playwright_img.xml')
    xbox_games = load_games_from_xml('xbox_output_img.xml')
    ps_games = load_games_from_xml('ps_output_img.xml')
    epic_games = load_games_from_xml('epic_output.xml')
```

- Merges entries from different sources using the lowercase version of the game name as the key.

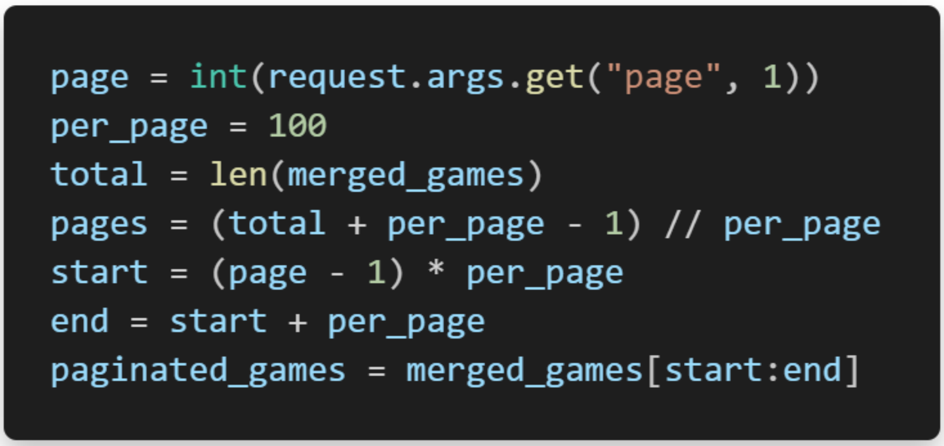


```
    # Merge by name (case-insensitive)
    merged = {}
    for game in steam_games:
        name = game['name'].strip().lower()
        merged[name] = {
            "image": game.get('image_url', None
),
            "name": game['name'],
            "steam_price": game['final_price'],
            "xbox_price": None,
            "ps_price": None,
            "epic_price": None
        }
```



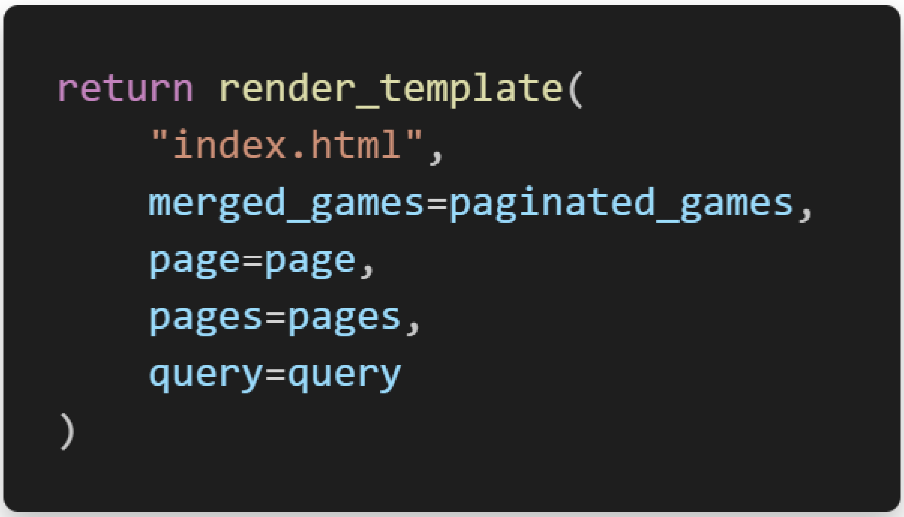
```
merged_games = list(merged.values())
merged_games.sort(key=lambda x: x['name'])  # Optional: sort by name

query = request.args.get("q", "").strip().lower()
if query:
    merged_games = [g for g in merged_games if query in g['name'].lower()]
```

- Converts prices into NTD.
- Supports searching through a q query parameter.
- Implements pagination using the page parameter.



```
page = int(request.args.get("page", 1))
per_page = 100
total = len(merged_games)
pages = (total + per_page - 1) // per_page
start = (page - 1) * per_page
end = start + per_page
paginated_games = merged_games[start:end]
```

- Renders the result in the index.html template.



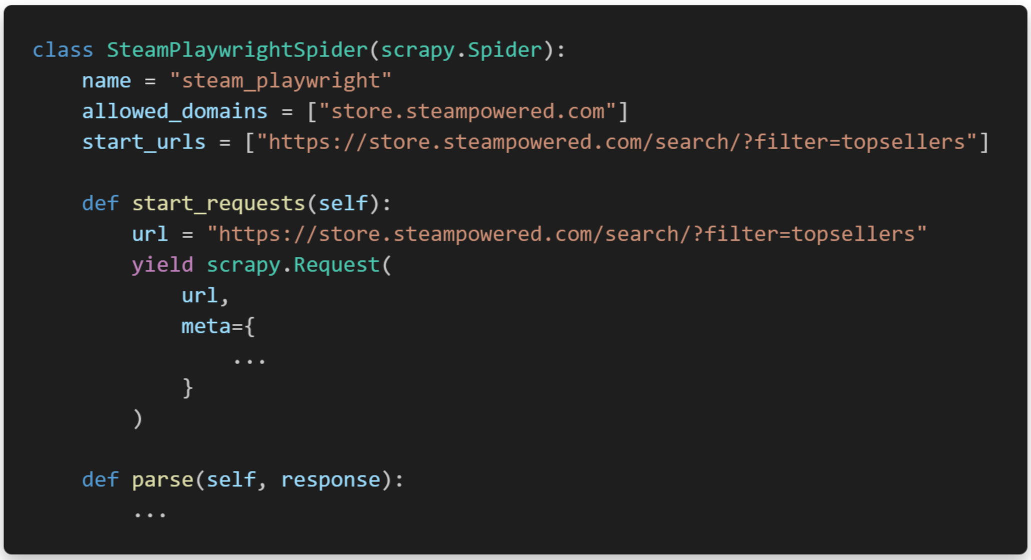
```python
return render_template(
    "index.html",
    merged_games=paginated_games,
    page=page,
    pages=pages,
    query=query
)
```

2.2 steam_store_playwright.py (Scrapy + Playwright)

This Scrapy spider is designed for pages that use dynamic scripting like javascript. The target website is the store.steampowered.com. Playwright itself is a library that enables reliable end-to-end testing for modern web apps. In this case, it can be used to handle pages that require JavaScript.

2.2.1 Class SteamPlaywrightSpider(scrapy.Spider)

The class is used for crawling the website. This spider scraps the Steam Store's Top Sellers page using Playwright to handle JavaScript and scrolling.

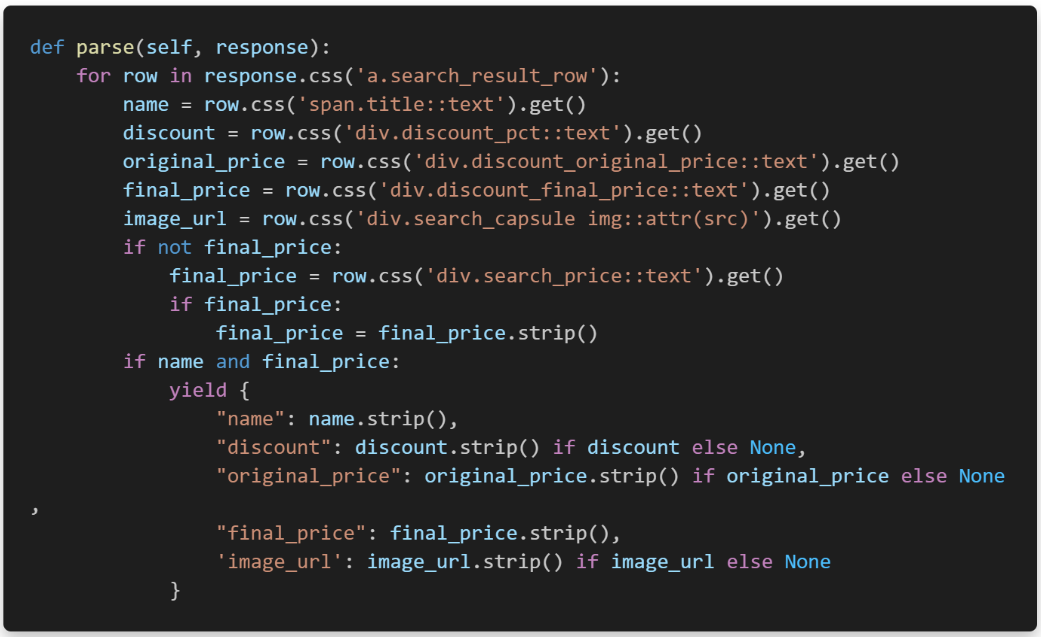


```python
class SteamPlaywrightSpider(scrapy.Spider):
    name = "steam_playwright"
    allowed_domains = ["store.steampowered.com"]
    start_urls = ["https://store.steampowered.com/search/?filter=topsellers"]

    def start_requests(self):
        url = "https://store.steampowered.com/search/?filter=topsellers"
        yield scrapy.Request(
            url,
            meta={
                ...
            }
        )

    def parse(self, response):
        ...
```

This code enables scrolling and loading Steam results dynamically. It is also able to scrape the game’s title, discount info, prices, and image source.

2.2.2 parse(self, response)

The parse function acts as the default method which gets called on every response.



```python
def parse(self, response):
    for row in response.css('a.search_result_row'):
        name = row.css('span.title::text').get()
        discount = row.css('div.discount_pct::text').get()
        original_price = row.css('div.discount_original_price::text').get()
        final_price = row.css('div.discount_final_price::text').get()
        image_url = row.css('div.search_capsule img::attr(src)').get()
        if not final_price:
            final_price = row.css('div.search_price::text').get()
            if final_price:
                final_price = final_price.strip()
        if name and final_price:
            yield {
                "name": name.strip(),
                "discount": discount.strip() if discount else None,
                "original_price": original_price.strip() if original_price else None
,
                "final_price": final_price.strip(),
                'image_url': image_url.strip() if image_url else None
            }
```

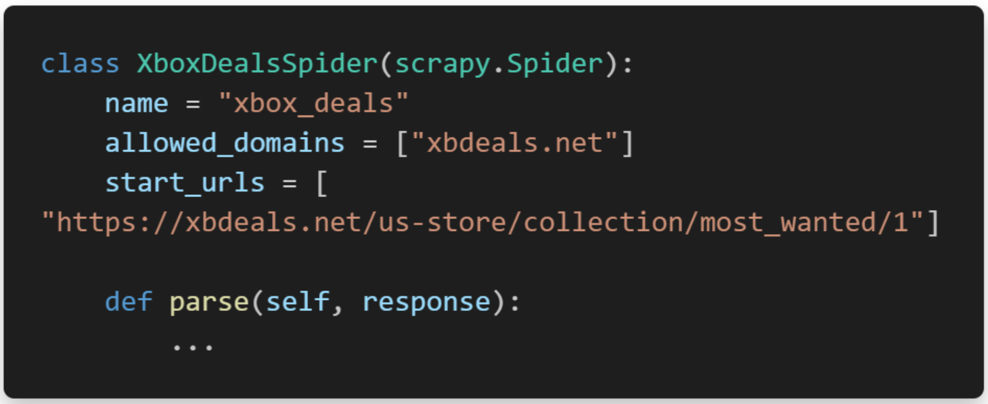
It uses response.css() to select HTML elements like “.table-products > tr > td”. This function is able to extract contents such as title, price, and image url (if available). This code also skips non-game rows (like headers, ads, etc…). Finally, it yields a *dict* that Scrapy writes to XML/JSON/CSV.

2.3 xbox_spider.py (Xbox Deals Scraper)

This file defines a Scrapy spider specifically made to extract game deal information from the xbdeals.net website. It scrapes the Xbox Most Wanted collection and supports pagination to crawl multiple pages of discounted games. Unlike the previous one, this one does not make use of the playwright library.

2.3.1 Class XboxDealsSpider(scrapy.Spider)

Similar to the Steam Store spider, this class inherits from scrapy.Spider and represents a basic crawler.

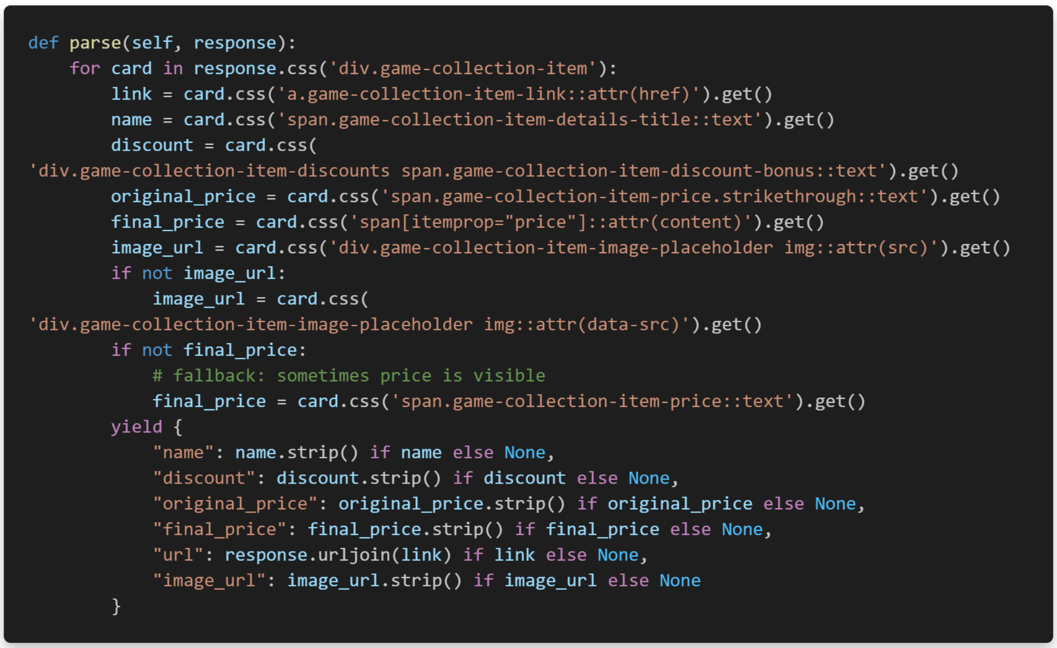


```
class XboxDealsSpider(scrapy.Spider):
    name = "xbox_deals"
    allowed_domains = ["xbdeals.net"]
    start_urls = [
"https://xbdeals.net/us-store/collection/most_wanted/1"]

    def parse(self, response):
        ...
```

The name "xbox_deals" is the unique spider name. To run it simply use the command line with "scrapy crawl xbox_deals" and maybe add "-o output.xml" to save the result into an xml file. Allowed_domains ensures the spider only follows links within this domain for safety and scope control. Finally, start_urls is the initial URL that the spider starts crawling from.

2.3.2 Function parse(response)

Similar to the Steam Spider, this is the main parsing method for the spider. This gets called automatically by Scrapy when receiving a response. The purposes for this is to extract game data from the current page and schedule crawling of the next page when available.

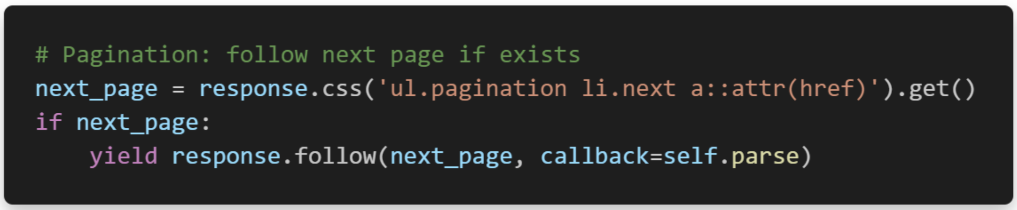


```python
def parse(self, response):
    for card in response.css('div.game-collection-item'):
        link = card.css('a.game-collection-item-link::attr(href)').get()
        name = card.css('span.game-collection-item-details-title::text').get()
        discount = card.css(
'div.game-collection-item-discounts span.game-collection-item-discount-bonus::text').get()
        original_price = card.css('span.game-collection-item-price.strikethrough::text').get()
        final_price = card.css('span[itemprop="price"]::attr(content)').get()
        image_url = card.css('div.game-collection-item-image-placeholder img::attr(src)').get()
        if not image_url:
            image_url = card.css(
'div.game-collection-item-image-placeholder img::attr(data-src)').get()
        if not final_price:
            # fallback: sometimes price is visible
            final_price = card.css('span.game-collection-item-price::text').get()
        yield {
            "name": name.strip() if name else None,
            "discount": discount.strip() if discount else None,
            "original_price": original_price.strip() if original_price else None,
            "final_price": final_price.strip() if final_price else None,
            "url": response.urljoin(link) if link else None,
            "image_url": image_url.strip() if image_url else None
        }
```

For each game card (div.game-collection-item), it extracts contents such as link, name, discount, original_price, final_price, and image_url. Additionally, if image_url or final_price isn't found from it, it uses a fallback.

2.3.3 Pagination Handling

After parsing all games on the page, the spider checks for a next page link using the pagination handling.



```python
# Pagination: follow next page if exists
next_page = response.css('ul.pagination li.next a::attr(href)').get()
if next_page:
    yield response.follow(next_page, callback=self.parse)
```

The code ensures that all pages in the "Most Wanted" list are crawled automatically, enabling full data coverage across multiple pages.

2.4 psstore_spider.py (Playstation Store Deals Scraper)

Same as the previous spiders, this file is a Scrapy spider that scrapes PlayStation game deals from psdeals.net. To be specific, the "Most Wanted" collection from the U.S. store. It extracts game information like names, prices, discounts, and images, and it follows pagination to retrieve deals from all available pages.

2.4.1 Class PSStoreSpider(scrapy.Spider)

This class defines the spider for crawling and scraping PlayStation deals.

```python
class PSStoreSpider(scrapy.Spider):
    name = "psstore"
    allowed_domains = ["psdeals.net"]
    start_urls = [
        "https://psdeals.net/us-store/collection/most_wanted/1"
    ]

    def parse(self, response):
        ...

        #pagination: follow next page if exists
        next_page = response.css("ul.pagination li.next a::attr(href)").get()
        if next_page:
            yield response.follow(next_page, callback=self.parse)
```

The three fields (name, allowed_domains, start_urls) act in a similar way as the Xbox spider. to run the spider, use command line “scrapy crawl psstore” and add “-o output.xml” to save the output as an xml file.

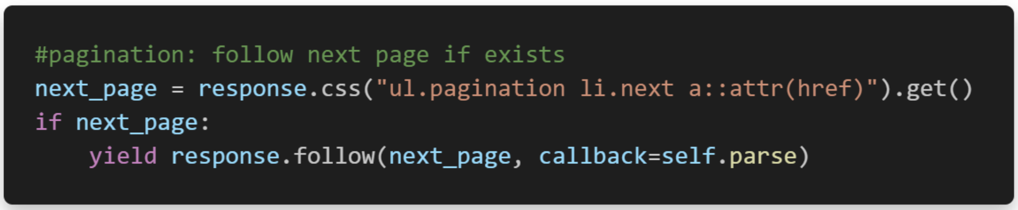
2.4.2 Function parse(response)

This is the spider's main parsing method and gets triggered automatically when Scrapy loads a page. The purpose is to extract relevant game data from the current page and move on to the next page if pagination is available.

```python
def parse(self, response):
    # Extract game information here
    for game in response.css("div.game-collection-item.col-md-2.col-sm-4.col-xs-6"):
        name = game.css("span.game-collection-item-details-title::text").get()
        link = game.css("a.game-collection-item-link::attr(href)").get()
        discount = game.css("span.game-collection-item-discount::text").get()
        original_price = game.css("span.game-collection-item-price strikethrough::text").get()
        final_price = game.css("span.game-collection-item-price-discount::text").get()
        image_url = game.css('div.game-collection-item-image-placeholder img::attr(src)').get()
        if not image_url:
            image_url = game.css('div.game-collection-item-image-placeholder img::attr(data-src)').get()
        if not final_price:
            final_price = game.css("span.game-collection-item-price ::text").get()
        yield {
            "name": name.strip(),
            "link": link.strip(),
            "discount": discount.strip() if discount else None,
            "original_price": original_price.strip() if original_price else None,
            "final_price": final_price.strip(),
            "image_url": image_url.strip() if image_url else None,
        }
```

This code extracts data such as name, link, discount, original_price, final_price, and image_url. This code loops over each game element (div.game-collection-item.col-md-2.col-sm-4.col-xs-6) to get those contents.

2.4.3 Pagination Handling



```python
#pagination: follow next page if exists
next_page = response.css("ul.pagination li.next a::attr(href)").get()
if next_page:
    yield response.follow(next_page, callback=self.parse)
```

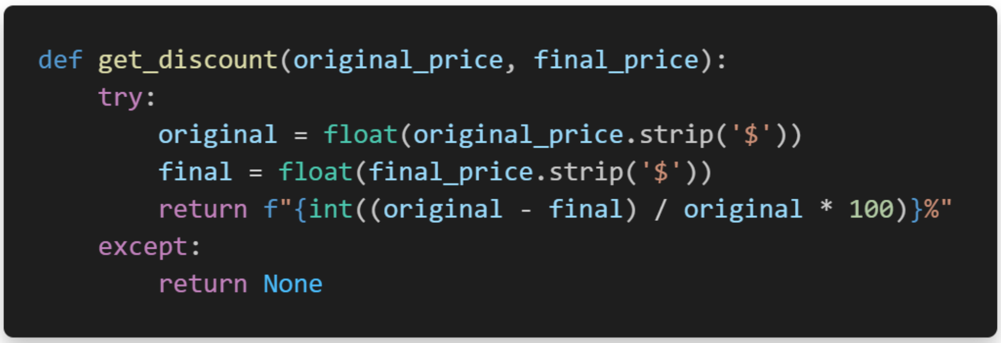
At the end of the parse method, a pagination logic is implemented so that if a “next page” link exists, it uses response.follow() to call parse() again recursively on the next page. This allows the spider to continue scraping across all pages in the collection, enabling it to cover all the most-wanted deals.

2.5 epic_spider.py (Epic Games Store Deals Scrapper)

This Scrapy spider scrapes the Top Sellers collection from the Epic Games Store. It extracts information about popular games, such as name, prices, discounts, and image, and includes debug tools to assist in parsing dynamic or inconsistent responses.

2.5.1 Function get_discount(original_price, final_price)

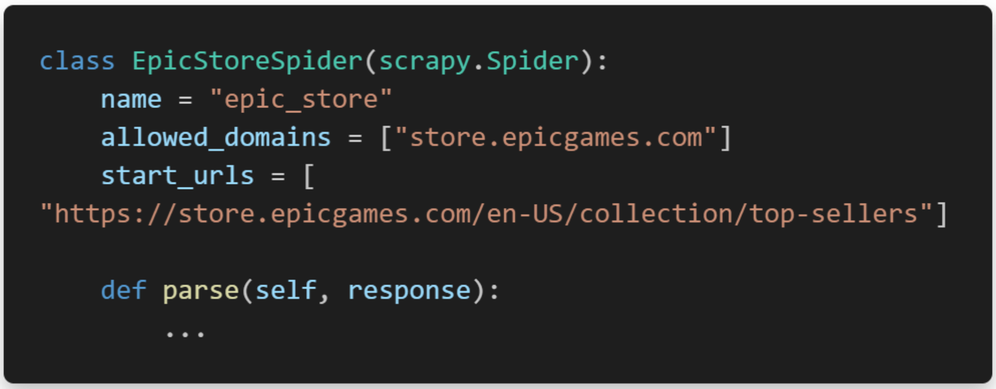
This is a function used to calculate the percentage discount, as it isn't really provided on the website.



```python
def get_discount(original_price, final_price):
    try:
        original = float(original_price.strip('$'))
        final = float(final_price.strip('$'))
        return f"{int((original - final) / original * 100)}%"
    except:
        return None
```

This code first strips the '$’ sign from the price text. Then, it converts both original_price and final_price to floats. The calculation for the discount number itself is a simple division to and multiply by 100. If there’s an issue, it will return None.

2.5.2 Class EpicStoreSpider(scrapy.Spider)



```python
class EpicStoreSpider(scrapy.Spider):
    name = "epic_store"
    allowed_domains = ["store.epicgames.com"]
    start_urls = [
"https://store.epicgames.com/en-US/collection/top-sellers"]

    def parse(self, response):
        ...
```

Similar to previous ones, the name is the unique name of the spider, run using "scrapy crawl epic_store -o output.xml". Other attributes are allowed_domains, which restricts the crawler to only scrape pages under this domain, and start_urls, used to begin the from the Top Sellers collection page.

2.5.3 Function parse(response)

```python
def parse(self, response):
    # Always save the HTML for debugging
    with open('epic_debug.html', 'w', encoding='utf-8') as f:
        f.write(response.text)

    self.logger.info(f"Response status: {response.status}")
    self.logger.info(f"Response length: {len(response.text)}")
    self.logger.info(f"First 500 chars: {response.text[:500]}")

    if not response.text or len(response.text) < 1000:
        yield {"error": "Empty or very short response", "status": response.status}
        return

    found = False
    for game in response.css('div[data-component="DiscoverOfferCard"]'):
        found = True
        # Get the name
        name = game.css('div.css-rgqwpc::text').get()
        # Get the image
        image_url = game.css('img[data-testid="picture-image"]::attr(src)').get()
        # Get the link
        link = game.css('a::attr(href)').get()
        # Get the prices
        original_price = game.css('span.css-4jky3p::text').get()
        final_price = game.css('span.css-12s1vua::text').get()
        # Get the discount
        discount = game.css('span.eds_1xxntt819::text').get()
        yield {
            "name": name.strip() if name else None,
            "image_url": image_url.strip() if image_url else None,
            "link": response.urljoin(link) if link else None,
            "original_price": original_price.strip() if original_price else None,
            "final_price": final_price.strip() if final_price else None,
            "discount": discount.strip() if discount else None
        }
    if not found:
        yield {"error": "No game cards found with selector", "status": response.status}
```

The parsing method for this spider can be divided into three parts.

1. Response validity Check:

```python
    if not response.text or len(response.text) < 1000:
        yield {"error": "Empty or very short response", "status": response.status}
        return
```

If the response is empty or unusually short (less than 1000 characters), it yields a special dictionary.

2. Game Data Extraction Loop:

```
for game in response.css('div[data-component="DiscoverOfferCard"]'):
```

Each iteration extracts name, image_url, link, original_price, final_price, and discount. Since the discount isn't available, it instead will call the get_discount function.

3. Fallback for Missing Results:

```
if not found:
            yield {"error": "No game cards found with selector", "status": response.status}
```

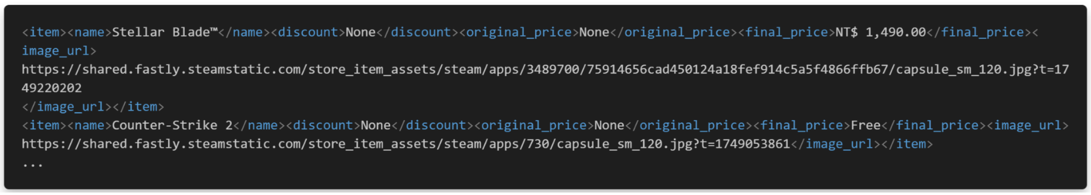
If no matching game cards are found (e.g., layout change or site update), it yields an error.

Chapter 3 Results

3.1 Scraping Outputs

This project successfully scrapes game deals from multiple digital storefronts using Scrapy spiders. Each spider collects information such as game title, price, discount, image, and link to the game.

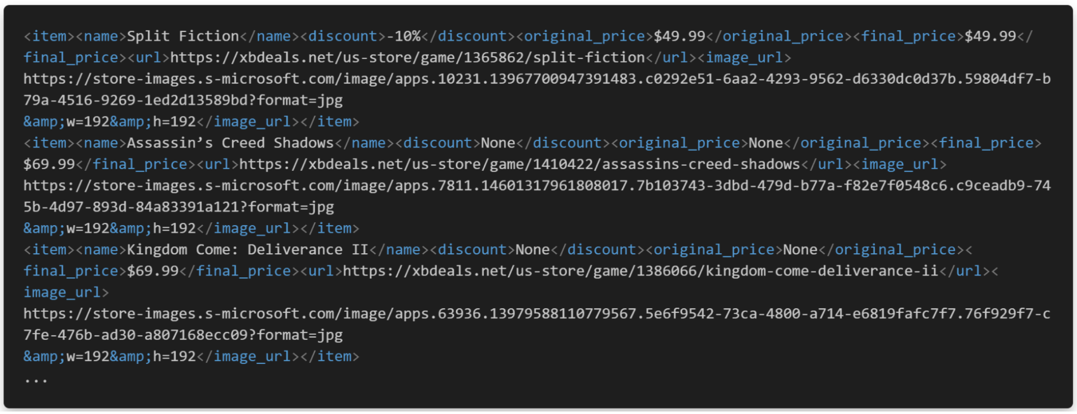
3.1.1 Steam Store



```
<item><name>Stellar Blade™</name><discount>None</discount><original_price>None</original_price><final_price>NT$ 1,490.00</final_price><
image_url>
https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/3489700/75914656cad450124a18fef914c5a5f4866ffb67/capsule_sm_120.jpg?t=17
49220202
</image_url></item>
<item><name>Counter-Strike 2</name><discount>None</discount><original_price>None</original_price><final_price>Free</final_price><image_url>
https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/730/capsule_sm_120.jpg?t=1749053861</image_url></item>
...
```

The Steam spider scraped discounted games from the Steam Specials page. It accurately extracted game titles, original and discounted prices, and discount percentages, along with the game's thumbnail and store link. The spider ensured it filtered only items that were actively discounted and gracefully handled cases where pricing or discounts might be missing.

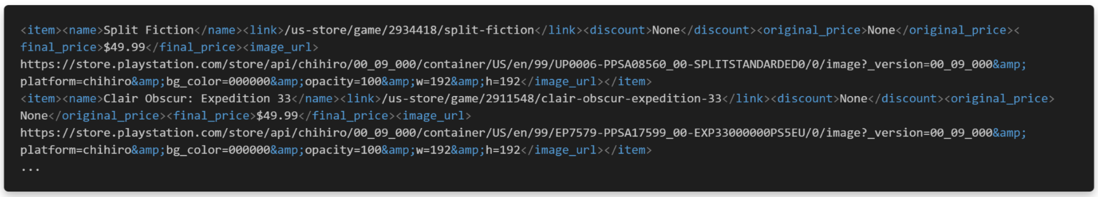
3.1.2 Xbox Store



```
<item><name>Split Fiction</name><discount>-10%</discount><original_price>$49.99</original_price><final_price>$49.99</
final_price><url>https://xbdeals.net/us-store/game/1365862/split-fiction</url><image_url>
https://store-images.s-microsoft.com/image/apps.10231.13967700947391483.c0292e51-6aa2-4293-9562-d6330dc0d37b.59804df7-b
79a-4516-9269-1ed2d13589bd?format=jpg
&amp;w=192&amp;h=192</image_url></item>
<item><name>Assassin’s Creed Shadows</name><discount>None</discount><original_price>None</original_price><final_price>
$69.99</final_price><url>https://xbdeals.net/us-store/game/1410422/assassins-creed-shadows</url><image_url>
https://store-images.s-microsoft.com/image/apps.7811.14601317961808017.7b103743-3dbd-479d-b77a-f82e7f0548c6.c9ceadb9-74
5b-4d97-893d-84a83391a121?format=jpg
&amp;w=192&amp;h=192</image_url></item>
<item><name>Kingdom Come: Deliverance II</name><discount>None</discount><original_price>None</original_price><
final_price>$69.99</final_price><url>https://xbdeals.net/us-store/game/1386066/kingdom-come-deliverance-ii</url><
image_url>
https://store-images.s-microsoft.com/image/apps.63936.13979588110779567.5e6f9542-73ca-4800-a714-e6819fafc7f7.76f929f7-c
7fe-476b-ad30-a807168ecc09?format=jpg
&amp;w=192&amp;h=192</image_url></item>
...
```

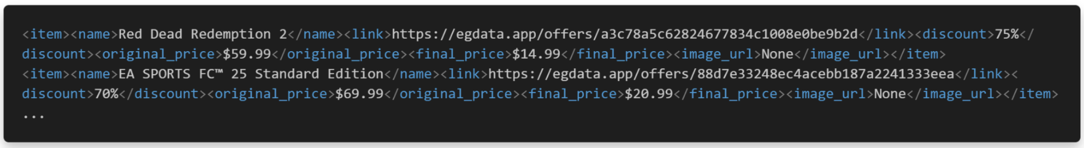
The Xbox spider targeted the most wanted deals section on xbdeals.net. It parsed game cards to collect data such as the name, discount rate, prices, game links, and images. It included fallback logic to handle cases where elements like images or prices were not directly available. Additionally, the spider supported pagination, allowing it to crawl through multiple pages of deals.

3.1.3 PlayStation Store


```
<item><name>Split Fiction</name><link>/us-store/game/2934418/split-fiction</link><discount>None</discount><original_price>None</original_price><
final_price>$49.99</final_price><image_url>
https://store.playstation.com/store/api/chihiro/00_09_000/container/US/en/99/UP0006-PPSA08560_00-SPLITSTANDARDED0/0/image?_version=00_09_000&amp;
platform=chihiro&amp;bg_color=000000&amp;opacity=100&amp;w=192&amp;h=192</image_url></item>
<item><name>Clair Obscur: Expedition 33</name><link>/us-store/game/2911548/clair-obscur-expedition-33</link><discount>None</discount><original_price>
None</original_price><final_price>$49.99</final_price><image_url>
https://store.playstation.com/store/api/chihiro/00_09_000/container/US/en/99/EP7579-PPSA17599_00-EXP33000000PS5EU/0/image?_version=00_09_000&amp;
platform=chihiro&amp;bg_color=000000&amp;opacity=100&amp;w=192&amp;h=192</image_url></item>
...
```

The PlayStation spider extracted data from the most wanted deals collection on psdeals.net. It captured relevant game deal information such as title, discount, pricing, image, and store links. The spider was designed to handle inconsistencies in the HTML structure and continued to the next page when pagination links were available, ensuring broader data coverage.

3.1.4 Epic Games Store

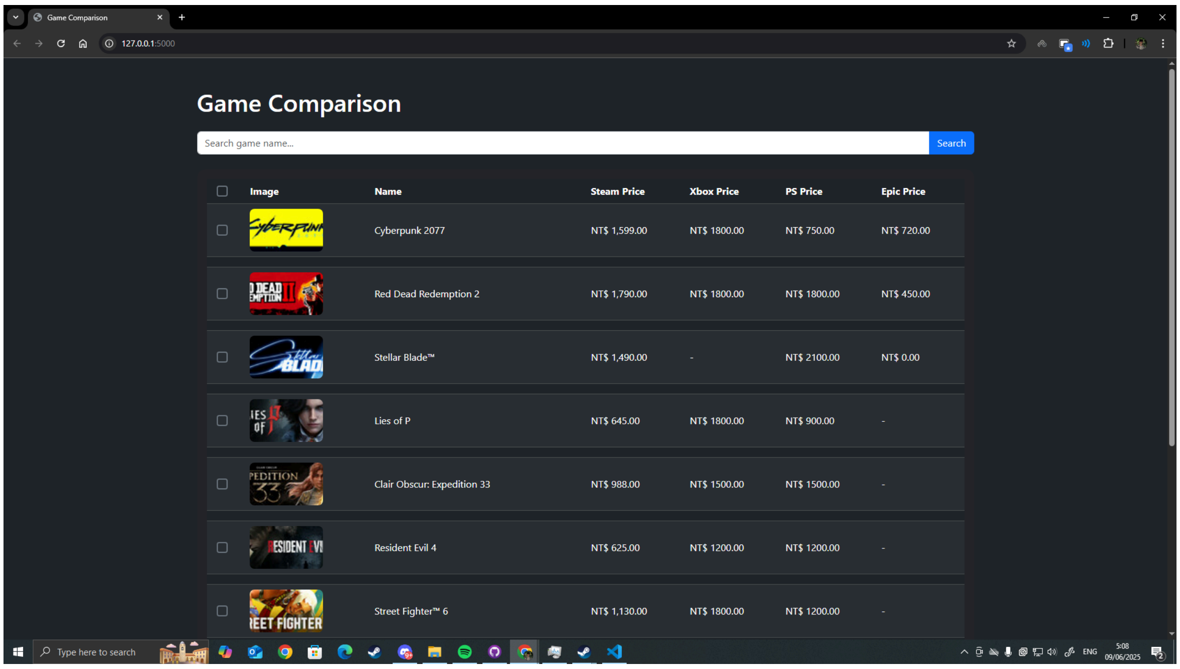

```
<item><name>Red Dead Redemption 2</name><link>https://egdata.app/offers/a3c78a5c62824677834c1008e0be9b2d</link><discount>75%</
discount><original_price>$59.99</original_price><final_price>$14.99</final_price><image_url>None</image_url></item>
<item><name>EA SPORTS FC™ 25 Standard Edition</name><link>https://egdata.app/offers/88d7e33248ec4acebb187a2241333eea</link><
discount>70%</discount><original_price>$69.99</original_price><final_price>$20.99</final_price><image_url>None</image_url></item>
...
```

The Epic Games spider scraped the top sellers section of the Epic Store. It handled a React-rendered layout and included logging and HTML output for debugging. The spider extracted information including the game title, prices, discounts, image, and link. It was equipped with error handling for short or empty responses and included a custom function to compute the discount percentage when not provided directly.

3.2 Flask Web App

The Flask application serves as the frontend interface to display the scraped data in a user-friendly format. It aggregates and renders game deals from all supported platforms.

3.2.1 Home Page



Game Comparison

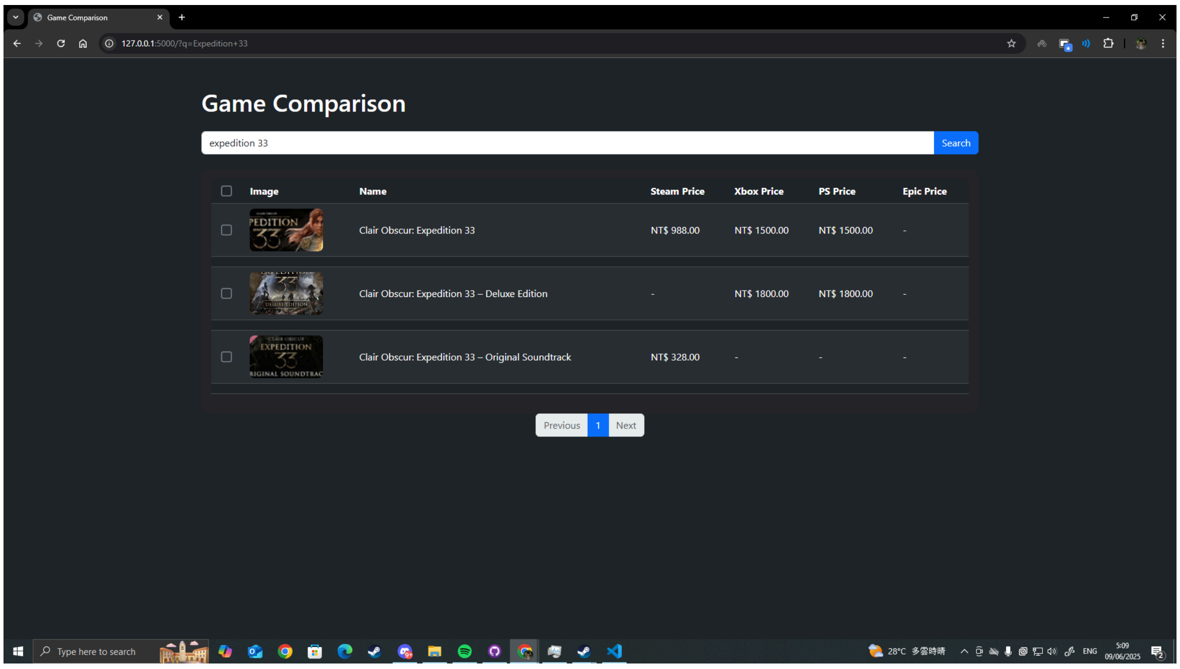
Search game name...

Search

	Image	Name	Steam Price	Xbox Price	PS Price	Epic Price
		Cyberpunk 2077	NT$ 1,599.00	NT$ 1800.00	NT$ 750.00	NT$ 720.00
		Red Dead Redemption 2	NT$ 1,790.00	NT$ 1800.00	NT$ 1800.00	NT$ 450.00
		Stellar Blade™	NT$ 1,490.00	-	NT$ 2100.00	NT$ 0.00
		Lies of P	NT$ 645.00	NT$ 1800.00	NT$ 900.00	-
		Clair Obscur: Expedition 33	NT$ 988.00	NT$ 1500.00	NT$ 1500.00	-
		Resident Evil 4	NT$ 625.00	NT$ 1200.00	NT$ 1200.00	-
		Street Fighter™ 6	NT$ 1,130.00	NT$ 1800.00	NT$ 1200.00	-

The home page displays a curated collection of game deals organized by platform. For each game, it shows the title, price, discount, and a thumbnail that links directly to the game's store page. This allows users to easily browse the latest and most popular discounts from Steam, Xbox, PlayStation, and Epic.

3.2.2 Search Function



Game Comparison

expedition 33

Search

	Image	Name	Steam Price	Xbox Price	PS Price	Epic Price
		Clair Obscur: Expedition 33	NT$ 988.00	NT$ 1500.00	NT$ 1500.00	-
		Clair Obscur: Expedition 33 – Deluxe Edition	-	NT$ 1800.00	NT$ 1800.00	-
		Clair Obscur: Expedition 33 – Original Soundtrack	NT$ 328.00	-	-	-

Previous 1 Next

The search function enables users to find specific games by title. It performs a case-insensitive match across all platforms and displays the relevant results dynamically. This feature enhances usability by allowing quick filtering through a potentially large dataset of deals.

Chapter 4 Conclusion

In Conclusion, this project has successfully implemented different methods of advanced computer programming, especially using framework to build a web and scraping websites to gather contents. By utilizing Scrapy, spiders were created for four major online game stores: Steam, Xbox, Playstation, and Epic Games. Each spider was designed to gather information such as game titles, prices, discounts, store links, and image source. Each spider was also specifically tailored to handle different challenges for each website.

The scraped data was also successfully displayed in a web app using Flask as the framework. This organizes the information in a nice and compact way for the user to use. There's also a working search feature for users to quickly filter based on game titles. Together, these tools work hand-in-hand in creating a price comparison website for video games.

In conclusion, the final product of this project provides insightful technical skills in web scraping and Flask development exercise. Other than that, it also provides new technical skills in problem-solving abilities when handling inconsistent HTML structures, missing values, and dynamic websites that use JavaScript.