¿Cómo podemos capturar los datos de la web?

Alex Stwart Porras Palma

Índice

1. Contexto

En este proyecto se va a proceder a crear un bot de telegram desde el cual realizar el webscraping sobre las ofertas actuales de la plataforma steam, así como la búsqueda de un título de juego y su precio actual dado un precio objetivo.

Enlace al repositorio: https://github.com/aporraspa/TYCVD.git

2. Descripción del data set y Representación Gráfica

El dataset será implementado en la base de datos MongoDB con los siguientes campos:

- id (string) Identificador único del elemento, es un valor autogenerado por MongoDB
- title (string) Nombre del juego
- discount (string) Descuento aplicado sobre el juego
- old_price (string) Precio original antes del descuento
- new_price (string) Precio actual después del descuento
- source (string) Origen del comando usado para extraer la información
- created_at (string) Marca de tiempo que indica cuando fue el objeto añadido a la base de datos

Figura 1: Ejemplo de dato almacenado en MongoDB

3. Descripción del funcionamiento

- Usuario de telegram envía un comando como /search Devil May Cry 5 20 o /sales
- El servidor de Telegram recibe el mensaje y lo redirige al script de python
- Selenium Navega a steam
- Recupera la información deseada de steam
  - Título del juego
  - Precios (viejo/nuevo)
  - Descuento
- Los datos recogidos se devuelven a telegram
- Se pregunta si se desea guardar los datos
  - Sí, se realiza la conexión a MongoDB y se insertan los datos
  - No. Fin

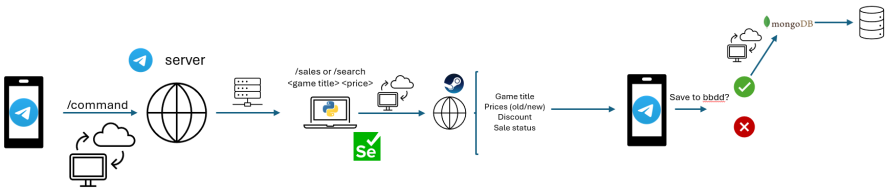

Figura 2: Esquema visual del funcionamiento del bot

4. Propietario

Tanto el código como los esquemas y las imágenes pertenecen al autor de este documento. Los datos han sido extraído de la página de steam conservándose así su propiedad y derecho de uso de la página.

5. Licencia

Released Under CC0: Public Domain License. Todo el proyecto ha sido elaborado con fines educativos no se necesita permiso de distribución es totalmente open source.

6. Inspiración

La idea detrás del proyecto es la posibilidad de crear un histórico de ofertas y de precios, así como evitar la necesidad de entrar en la página web a mirar y o buscar un juego. Un ejemplo del histórico se puede encontrar en la página https://steamdb.info/ la cual ofrece un histórico de los precios dado un intervalo de tiempo.

Figura 3: Ejemplo de steam para histórico de precios

7. Código

A modo de presentación el código se ha estructurado en un jupyter notebook

7.1. Parámetro options

```
options = Options()
options.add_argument("start-maximized")
#options.add_argument("--headless")
```

Listing 1: parámetro options function

Para ejecutar el script se pueden hacer uso de diversos parámetros en options en este caso a modo de muestra se comentará la opción de headless para que se pueda observar como se van realizadno las acciones en el navegador

7.2. Comando search

```
def search_game_by_price(game_name, target_price):
    driver = webdriver.Chrome(service=Service(CHROME_DRIVER_PATH)
        , options=options)

    try:

        driver.get('https://store.steampowered.com/')


        WebDriverWait(driver, 10).until(
```

```
        EC.presence_of_element_located((By.ID, 'store_nav_search_term'))
    )

    search_box = driver.find_element(By.ID, 'store_nav_search_term')
    search_box.clear()
    search_box.send_keys(game_name)
    search_box.send_keys(Keys.RETURN)

    WebDriverWait(driver, 10).until(
        EC.presence_of_all_elements_located((By.CSS_SELECTOR, 'div.responsive_search_name_combined'))
    )


    first_result = driver.find_element(By.CSS_SELECTOR, 'div.responsive_search_name_combined')


    title = first_result.find_element(By.CSS_SELECTOR, 'span.title').text.strip()

    price_block = first_result.find_element(By.CSS_SELECTOR, 'div.search_price_discount_combined')

    try:

        discount_block = price_block.find_element(By.CSS_SELECTOR, 'div.discount_block')

        if 'no_discount' in discount_block.get_attribute('class'):

            final_price_element = price_block.find_element(By.CSS_SELECTOR, 'div.discount_final_price')
            final_price = final_price_element.text.strip()

            final_price = float(final_price.replace('€', '').replace(',', '.').strip())
        else:

            final_price_element = discount_block.find_element(By.CSS_SELECTOR, 'div.discount_final_price')
            final_price = final_price_element.text.strip()

            final_price = float(final_price.replace('€', '').replace(',', '.').strip())

    except Exception as e:
        print(f"Error extracting price for {title}: {e}")
```

```
            return None

        if final_price <= target_price:
            print(f"Game: {title}")
            print(f"Price: {final_price}   ")
            return title, final_price

        print(f"Game: {title} is not under the target price.")
        return None

    except Exception as e:
        print(f"Error during search: {e}")
        return None

    finally:
        driver.quit()
```

Listing 2: search game + price function

En esta función Selenium se usa para automatizar la interacción de la página web Steam para recolectar información sobre un juego en específico. Utiliza el webdrive de Chrome, para ello es necesario descargar el chromedriver compatible con la versión de chrome instalada en el dispositivo. Esto permite la ejecución en segundo plano del navegador sin la necesidad abrirlo de manera visible. Navega a la página de steam, espera a que la página cargue busca el elemento de la barra de búsqueda utilizando el ID de HTML, borra por si hubiese texto introducido y escribe el título del juego proporcionado simulando la búsqueda como si fuera un usuario pulsando el botón enter. Tras la búsqueda espera a que los resultados carguen por completo y se recopilan todos los elementos de la búsqueda mediante un selector CSS que escoge el primer resultado de la búsqueda para extraer la información. Se evalúa si el bloque contiene información sobre descuentos, y se recupera. Finalmente se limpia la cadena de caracteres y se sustituyen las comas por puntos. (Esto último se hace por formato propio de decimal)

7.3. Comando sales

```
def sales_from_steam():
    driver = webdriver.Chrome(service=Service(CHROME_DRIVER_PATH)
        , options=options)
    print(f"Navigating to: {URL}")
    driver.get(URL)

    try:
        WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.ID, '
                SaleSection_13268'))
        )

        # Scroll to bottom to load all items
        last_height = driver.execute_script("return document.body
            .scrollHeight")
        while True:
```

```
            driver.execute_script("window.scrollTo(0, document.
                body.scrollHeight);")
            time.sleep(1)
            new_height = driver.execute_script("return document.
                body.scrollHeight")
            if new_height == last_height:
                break
            last_height = new_height

        game_containers = driver.find_elements(By.CSS_SELECTOR, '
            div.v9uRg57bwOaPsvAnkXESO')

        games = []

        for game in game_containers:
            try:
                title = game.find_element(By.CSS_SELECTOR, 'div.
                    StoreSaleWidgetTitle').text
                discount = game.find_element(By.CSS_SELECTOR, '
                    div.cnkoFkzVCby40gJ0jGGS4').text

                try:
                    old_price = game.find_element(By.CSS_SELECTOR
                        , 'div._3fFFsvII7Y2KXNLDk_krOW').text
                    new_price = game.find_element(By.CSS_SELECTOR
                        , 'div._3j4dI1yA7cRfCvK8h406OB').text
                except:
                    old_price = new_price = "N/A"

                games.append({
                    'title': title,
                    'discount': discount,
                    'old_price': old_price,
                    'new_price': new_price
                })

            except Exception as e:
                print(f"Error parsing a game: {e}")
                continue
        return games

    except Exception as e:
        print(f"Error: {e}")
    finally:
        driver.quit()
```

Listing 3: sales function

El comando sales utiliza selenium par acceder y extraer los datos de las ofertas actuales de steam desde su apartado ofertas especiales. De la misma manera que el anterior, comienza iniciando el navegador con el webdriver de chrome. Se dirige directamente a la página de las ofertas especiales y espera a que cargue la sección de las ofertas, éste es localizado mediante HTML ID SalesSection_13268, debido a que no todo el contenido carga a primera vista es necesario desplazarse hacia abajo, el script simula el desplazamiento manual ejecutando JS que se desplaza hasta el final de la página. Una vez hecho esto, se localizan todos los elementos del contenedor (12 en este caso) mediante HTML CSS selector. Se itera sobre cada elemento y se extrae la información.

8. Robot.txt

```
Host: store.steampowered.com
User-Agent: *
Disallow: /share/
Disallow: /news/externalpost/
Disallow: /account/emailoptout/?*token=
Disallow: /login/?*guestpasskey=
Disallow: /join/?*redir=
Disallow: /account/ackgift/
Disallow: /email/
Disallow: /widget/
```

Listing 4: Robots.txt steam

Antes de realizar el scraping es importante revisar el archivo robots.txt de steam, desde aquí se informa a los usuarios a qué partes se pueden acceder mediante el uso de bots. User-agent: * indica que las reglas especificadas a continuación se aplican a todos los agentes de usuario automatizados, sin excepción. Las entradas "Disallow.[en]umeran rutas específicas en el sitio web que los bots no deben visitar. Por ejemplo, los directorios relacionados con cuentas de usuario, operaciones de correo electrónico etc. Cabe destacar que en el caso del uso de selenium la idea es la de simular un comportamiento humano por lo que con un modelo robusto y eficaz sería posible saltarse las alertas de bots, (los captcha). El bot realizado para este proyecto no infringe la política de robots así que se podría continuar con su uso.

9. MongoDB

Como base de datos se ha escogido MongoDB, el motivo es el de aprovechar la tecnologías proporcionadas por la asignatura, así como sacar partido a la estructura de documentos, para almacenar los datos. En este caso al ser un ejemplo sencillo, se tiene en cuenta de que los juegos pueden tener descuento o no, por lo que el esquema flexible de MongoDB puede gestionar estas inconsistencias sin nigún tipo de problema. Además admite operaciones de lectura y escritura de alta velocidad, lo que lo hace ideal para aplicaciones en tiempo real, como un bot de Telegram que recopila y almacena información del juego dinámicamente según las solicitudes de los usuarios o el análisis de ventas. Aunque con la versión gratuita se obtiene una base de datos limitada es más que suficiente para gestionar este proyecto. Para realizar la conexión se hace uso de su sistema en la nube MongoDB atlas lo que es especialmente práctico a la hora de implementar el bot en servidores en la nube.

10. Anaconda

Para hacer un mejor uso tanto de las librerías como del entorno y su espacio, se ha hecho uso de Anaconda, para crear un entorno virtual, el cual se usará como kernel para compilar tanto el jupyter notebook, como el script del bot, de esta manera, una vez finalizado el proyecto, se puede eliminar la carpeta y de esta manera conservar el espacio, manteniendo una práctica limpia y eficaz.

11. Dataset

El dataset ha sido descrito en apartados anteriores junto con su estructura y su almacenamiento, en MongoDB.

12. Vídeos

El enlance a los vídeos se encuentran en el repositorio, concretamente en el README, es importante destacar que para poder acceder a ellos es necesario identificarse con una cuenta perteneciente a la Universitat Oberta de Catalunya.

Hay disponibles dos vídeos:

- En el primero se realiza una demostración del uso del bot, no hay descripciones en él ya que su demostración es implícita, de todas formas se comenta brevemente en el documento. De manera resumida se accede a la página de steam mediante chromedriver, se ha uso del scrolling para obtener los datos almacenados en el objeto contenedor de las ofertas, se extraen los nombre de los títulos del juego, junto con su precio antiguo, nuevo y su descuento si lo hubiese.
- En el segundo vídeo se hace una demostración mediante jupyter notebook, en el cual se ejecutan las dos funciones de script para además de realizar el primer paso, se buscan dos juegos concretamente Devil May Cry 5, y Hogwarts Legacy, el segundo parámetro necesario para esta función es el precio objetivo al que se está dispuesto comprar el juego, es decir, para un valor de 20, el script será devuelto favorablemente, si el precio actual del juego es igual o inferior al descrito.

13. Conclusiones y Líneas futuras

El proyecto demuestra como la automatización del web scraping se puede utilziar para la creación de una herramienta para conseguir y almacenar datos en tiempo real se ha creado un sistema escalable y flexible además los usuarios pueden buscar juegos específicos y recibir información en tiempo real según sus precios deseados, además de ver las ventas en curso directamente desde la tienda de Steam. Además, MongoDB permite almacenar y recuperar posteriormente los juegos consultados, lo que facilita el análisis o el seguimiento histórico.

Como mejoras al proyecto sería interesante implementar un comando en segundo plano que revise periódicamente el precio de un juego o de varios juegos y que envíe una notificación a telegram cuando los juegos bajen del precio establecido por el usuario.