## Automated Web Scraping Report using Selenium

**Task Overview**

This report outlines the development and execution of a Java-based Selenium web scraper targeting the HP Canada online store to extract laptop product information. The solution meets all specified tasks:

1. **Task 1** – Write a Selenium-based Java program to:
   * Open a website in the browser
   * Interact with webpage elements (buttons, modals)
   * Extract data (text, product specs)
   * Save data into a structured file
2. **Task 2** – Scrape multiple pages and combine results
3. **Task 3** – Use advanced Selenium techniques (e.g., waits, handling popups)

**Selected Website Information**

**Website:** HP Canada Laptop Listings  
**URL:** <https://www.hp.com/ca-en/shop/list.aspx?sel=ntb>  
**Purpose:** To extract structured data for available HP laptops sold in Canada.

**Explanation of Solution**

**Open Website with Selenium**

The Java program uses the Microsoft Edge WebDriver to open the website in a maximized browser window:

WebDriver driver = new EdgeDriver(options);

driver.get("https://www.hp.com/ca-en/shop/list.aspx?sel=ntb");

**Interact with Elements**

The program interacts with page elements such as:

* **Cookie Acceptance Button**:

driver.findElement(By.cssSelector("#onetrust-accept-btn-handler")).click();

* **Promotional Modal Close Button** (if present):

WebElement closeButton = driver.findElement(By.cssSelector(".dy-lb-close"));

closeButton.click();

**Extract Data from Products**

Each product listing's details are extracted:

* Product Name
* Price
* Specifications (OS, Processor, Memory, Storage, Display, Graphics)

Collected data is saved in a structured format:

csvWriter.append(String.format("%s,%s,%s,%s\n", brandName,name, price,tempSpecs));

// The tempSpecs variable contains specifications in the following order:

// "Operating System, Processor, Memory, Storage, Display, Graphics"

// So the final CSV row format is: Brand Name, Product Name, Price, OS, Processor, Memory, Storage, Display, Graphics

**Navigate and Scrape Multiple Pages**

Using a loop, the scraper detects and clicks the "Next" button when available, scraping each new page:

WebElement nextButtons = driver.findElement(By.cssSelector(".Pagination-module\_next\_\_T1EtZ"));

nextButtons.click();

wait.until(ExpectedConditions.stalenessOf(products.get(0)));

**Use Advanced Selenium Commands**

* **Explicit Waits**: To wait for elements to load or disappear:

wait.until(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".HorizontalProductTile\_container\_\_3F4IN")));

* **Popup Handling**: Modal dialogs are detected and closed gracefully with a timeout catch.
* **Staleness Wait**: Ensures content updates on page change.

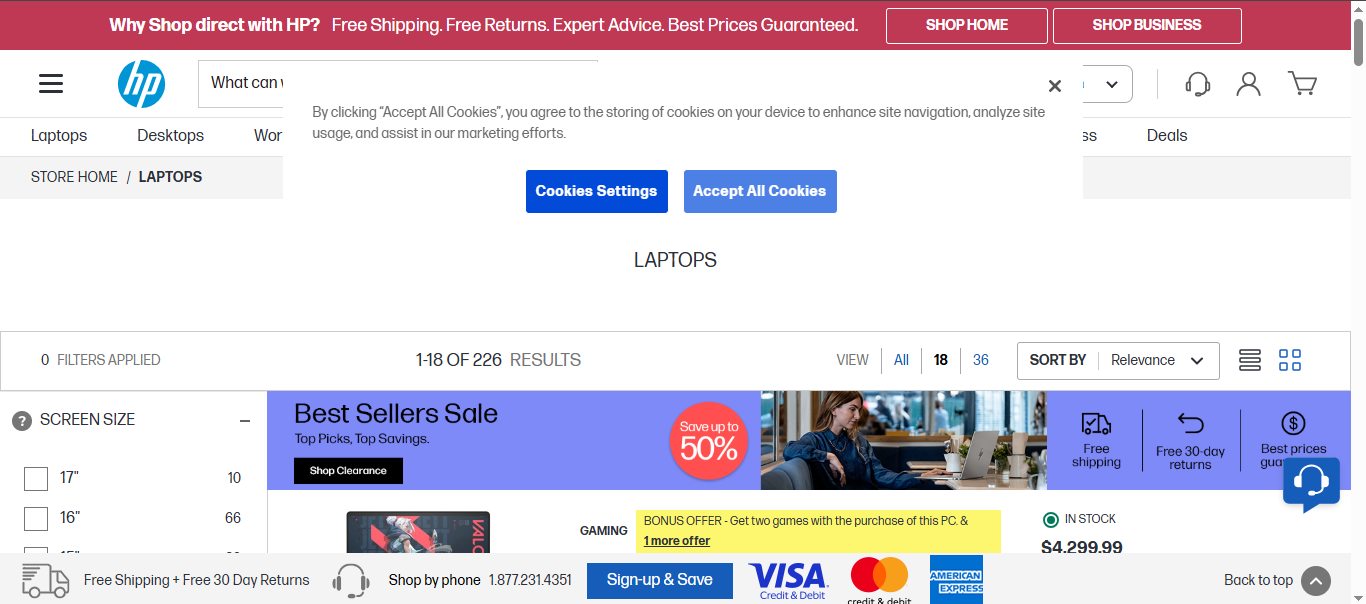
**Output CSV File**

Data is saved to a CSV file using:

FileWriter csvWriter = new FileWriter("hp\_laptops.csv");

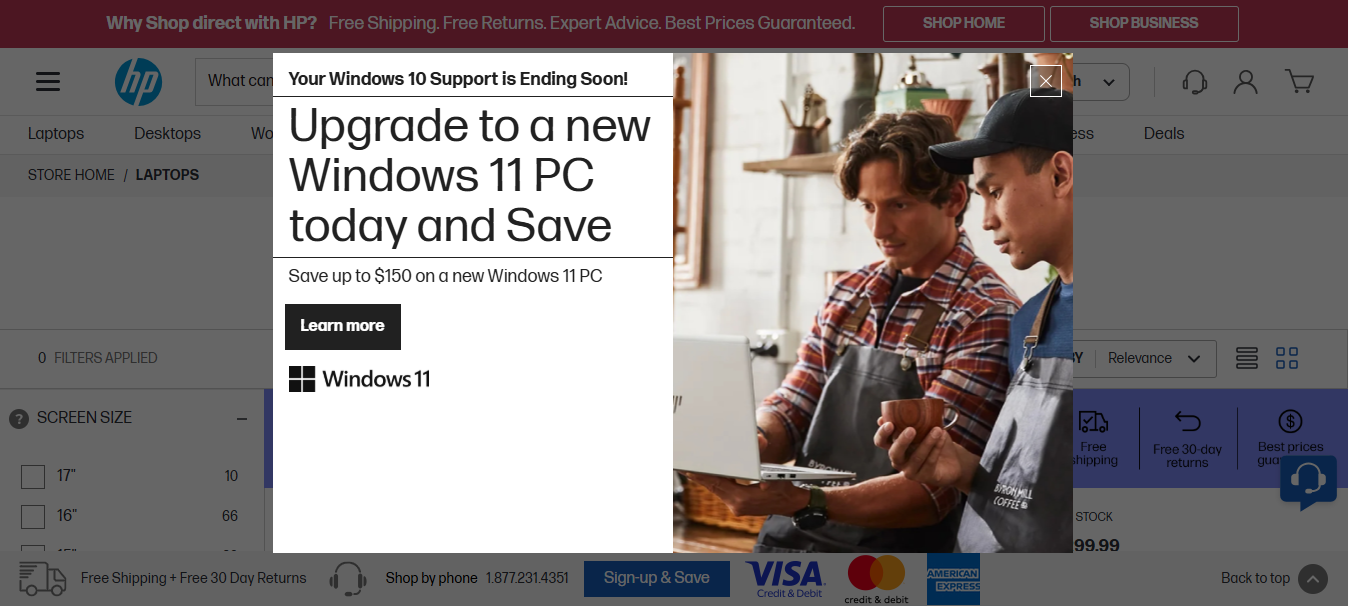
csvWriter.append("Brand Name,Product Name,Price,Operating System,Processor,Memory,Storage,Display,Graphics\n");

**Output Screenshots (Descriptions)**

****

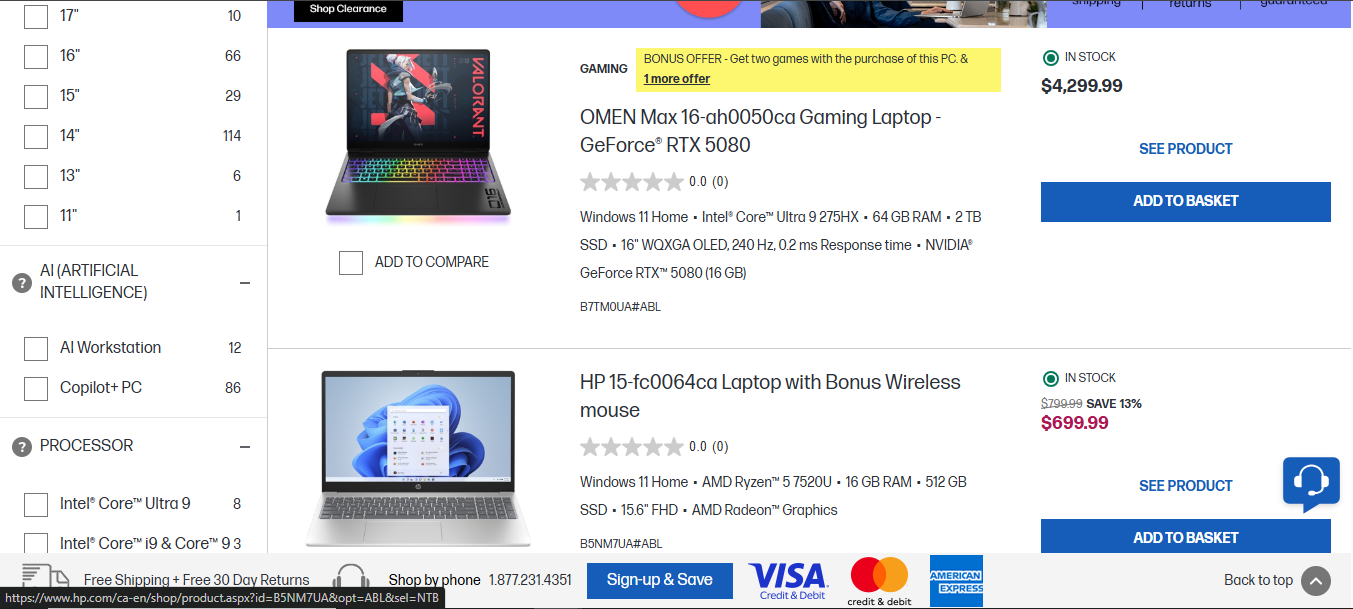
**Fig. 1** – **Homepage with Cookie Consent**

**Explanation:**  
This screenshot shows the homepage of the HP Canada laptops listing page opened in the Edge browser using Selenium. At the screen, a **cookie consent banner** is displayed, prompting the user to accept cookies before interacting with the page content.  
To proceed with scraping, the Selenium script programmatically clicks the "Accept All Cookies" button by locating it using the #onetrust-accept-btn-handler CSS selector. This step is essential to ensure that the rest of the webpage elements are interactable and to avoid any interruptions during data extraction.

****

**Fig. 2** – **Modal Popup Dialog**

**Explanation:**  
This screenshot captures the **modal popup dialog** that appears shortly after the page loads. It is an overlay window (often for promotional or newsletter signup purposes) that blocks interaction with the main content behind it.  
The Selenium script handles this interruption by waiting for the modal to become visible using an explicit wait and then locating the **close button** using the selector .dy-modal-wrapper .dy-modal-contents .dy-lb-close. Once found, the script clicks this button to close the modal. This ensures uninterrupted access to the product listings for accurate data scraping.

****

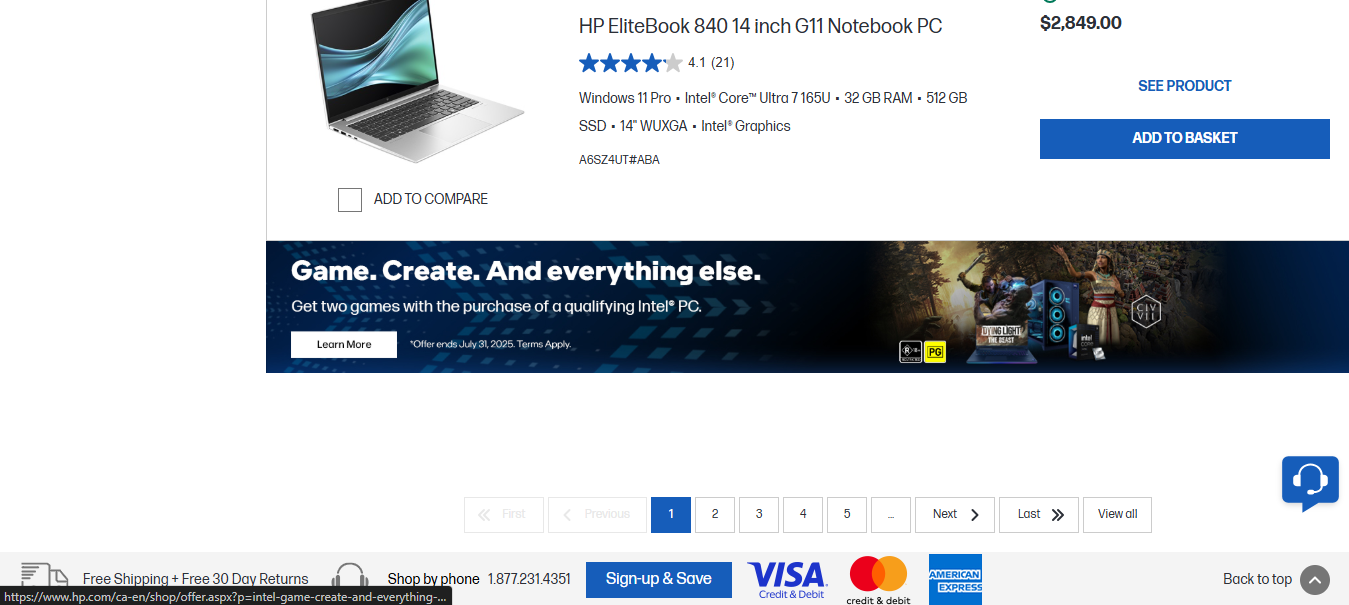
**Fig. 3** – **Extracted Product Listings**

**Explanation:**  
This screenshot displays the **main product listing section** of the HP Canada laptops page after dismissing the cookie consent banner and the modal popup. Each laptop is represented as a product tile containing essential information such as product name, price, and a brief set of specifications.

The Selenium script locates all these tiles using the CSS selector .HorizontalProductTile\_container\_\_3F4IN. For each tile, it extracts:

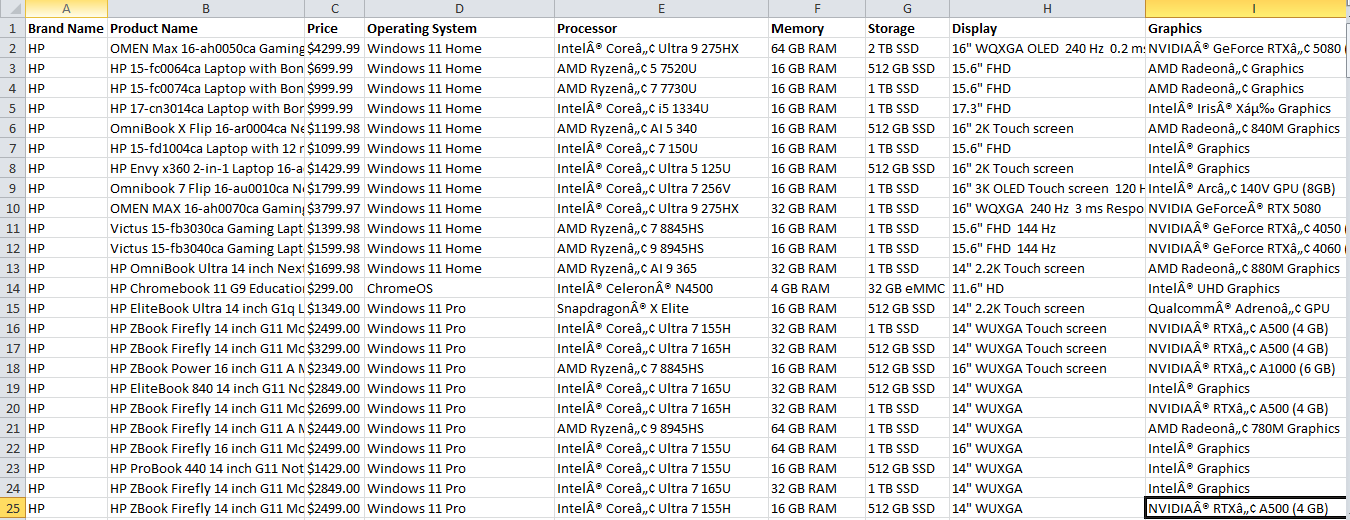
* **Product Name**
* **Price**
* **Specifications** (including OS, Processor, Memory, Storage, Display, and Graphics)

These details are parsed and formatted into rows, which are then written into a CSV file. This marks the core data extraction stage of the automation.

  
**Fig. 4** – **Navigating to Next Page**

**Explanation:**  
This screenshot shows the pagination control at the bottom of the product listing page, specifically highlighting the ‘Next’ button. After scraping data from the current page, the Selenium script checks for the availability and visibility of this button using the CSS selector .Pagination-module\_next\_\_T1EtZ.

If the button is enabled (i.e., not disabled), the script scrolls it into view using JavaScript and then clicks it to load the next set of products. An explicit wait ensures that the script waits for the previous product elements to become stale before continuing to scrape the new page content. This process is repeated until there are no more pages to navigate, ensuring that all laptops across multiple pages are extracted and saved.

  
**Fig. 5** – **Final CSV File Opened in Excel**

**Explanation:**  
This screenshot shows the **contents of the generated CSV file (**hp\_laptops.csv**)**, which contains the structured output of the scraped data. Each row in the file represents a laptop product with the following columns:

* **Brand Name**
* **Product Name**
* **Price**
* **Operating System**
* **Processor**
* **Memory**
* **Storage**
* **Display**
* **Graphics**

The data was written using a FileWriter in the Java program, and fields were cleaned to remove commas to maintain CSV integrity. This final output verifies that the program successfully scraped, processed, and exported information from multiple pages into a readable and reusable format.

**Conclusion**

The Selenium Java scraper successfully completed all required tasks, including:

* Website interaction
* Data extraction
* Multipage navigation
* CSV export
* Advanced waits and popup handling

This project demonstrates proficiency in real-world web scraping using Selenium.

**Technologies Used**

* Java
* Selenium WebDriver (Edge)
* CSV FileWriter
* Eclipse IDE
* HP Canada eCommerce Website