**Documentation:**

Here’s an example of automating a search and add-to-cart scenario on Amazon.com using Python Selenium with the Page Object Model (POM).

**Project Structure:**

POM

│

├── PageObjectClasses/

│ ├── base\_page.py

│ ├── home\_page.py

│ └── search\_results\_page.py

│

└── test\_shopping.py

**Explanation:**

* **Base Page (base\_page.py)**: Contains common methods for interacting with web elements (find element, click element, enter text) and navigating to the base URL (open).
* **Home Page (home\_page.py)**: Represents the Amazon home page and includes methods specific to performing a search (search\_for\_product). Uses try-except blocks to catch and handle exceptions gracefully.
* **Search Results Page (search\_results\_page.py)**: Represents the search results page where we define actions like clicking on the first product (click\_first\_product). Uses try-except blocks to handle exceptions during element interactions.
* **Test Script (test\_shopping.py)**: Implements test cases using unittest. Sets up WebDriver (setUp), performs test actions (test\_search\_and\_add\_to\_cart), and handles cleanup (tearDown). Uses assertions (self.fail) to mark test failures when expected actions (like searching or clicking) fail.

**Note:**

* try-except blocks, which is used for **Exception Handling**, in each method ensure that if any Selenium action fails (e.g., element not found), the error is caught and logged, preventing the test from crashing abruptly.
* Implementing the Page Object Model in Python Selenium testing enhances test automation practices by promoting clean, scalable, and maintainable code.