Amazon Automation

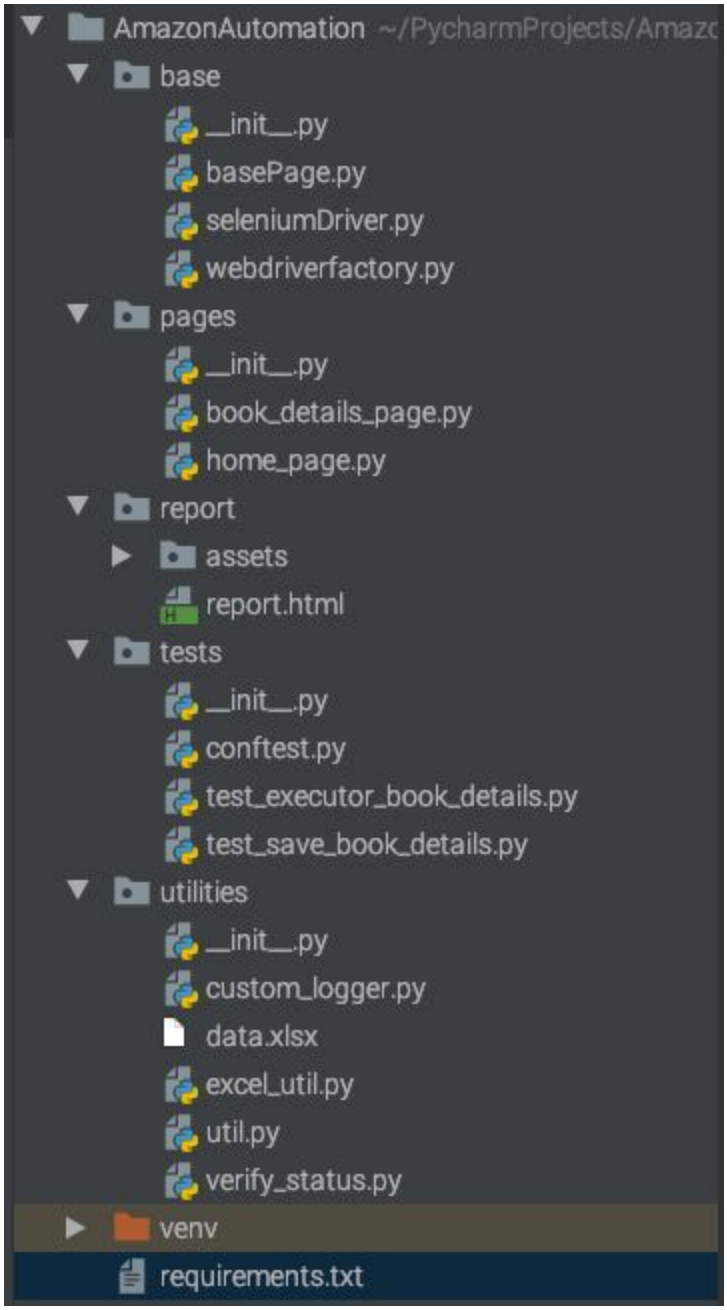
By Selenium Python

Sudendra Priyan U

Project Details

Language	: **Python**
Framework	: **PyTest** used for building test cases **& UnitTest** for execution
Design Pattern	: **Page Object Model**
Browser	: Can run on both **Chrome & Firefox**
Report	: Implemented **Simple HTML** report
Assertion	: **Generalized Custom Assertions** which is highly reusable

Project Structure



Packages

1. **base**
2. **pages**
3. **report**
4. **tests**
5. **utilities**

Overview Of Packages

base: Contains basic files which works as the base for the framework.

pages: Contains all the pages of the project. Each page file has their own xapth & Functions.

report: Will contain the HTML report which is generated after each run

tests: Will contain configuration file and main test classes.

utilities: Will have logger class, Data Sheet, Custom Assertion Class and utility file. This package supplies essential and add on facilities to our project.

Design decisions

- The project was designed with the **code reusability** in mind. We will see in detail about what every files from all the packages does in detail.
- **seleniumDriver.py** from **base** has custom functions which serves as a channel to Selenium Web driver functions.
- **Webdriverfactory.py** from **base** was built to provide flexibility to run our project in **different browsers**. Also the project downloads the driver(Chromedriver/Geckodriver) which it needs **automatically**, This allows the users to not have the drivers locally.
- **basePage.py** from **base** has common functions like verifying page title, Which can be **used across all pages.**

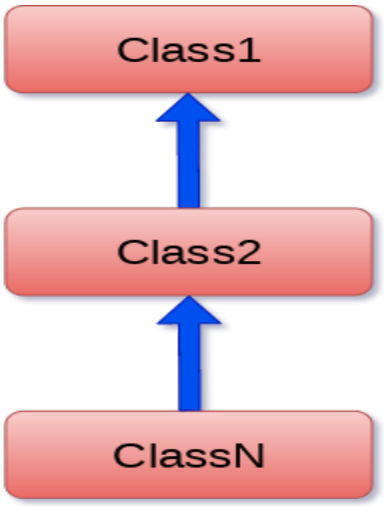
- The **pages** package will have Web pages from Amazon as an **individual** file. **Example**: Say Amazon has home page and book detail page. So we will have 2 files(**home_page.py** and **book_details_page.py**) in the package.
- All the **locators** of a page will be stored in the page file itself. **Reason:** If a locator is changed in a particular page we will have the convenience of accessing it in the page file directly.
- Also I have preferred **xpath** as a primary locator, Since it was more reliable than other locators from amazon page.
- Each page will have **independent functions** which can be achieved only on that page alone.
- The **tests** will have all the test cases (like **test_save_book_details.py**) which we need to execute and also the configuration for our whole project in **conftest.py** file
- To form and verify a test case we will call the **independent functions** of the pages.
- The **fixtures** declared in **conftest.py** will be passed in all our test cases during **initialization** with the help of **autouse** feature in **pytest**
- In **utilities** we will have **custom_logger.py** which will add the logs in our report. Then there is **data.xlsx** which has two sheets **input** and **output,** From which we read data from **input sheet** and write our data to **output sheet.** The **excel_util.py** file is the brain behind reading and writing into the excel sheet.
- The verification part in other words **Assertion.** This is handled in **verify_status.py** from **utilities.** It has two main functions - **singleAssertion()** and **finalAssertion()**
    - **singleAssertion() -** This will be used to gather the assertion results(**PASS/FAIL**) and add to a list called **result_list**.
    - **finalAssertion() -** This will be used at the end of function. This will decide the outcome of a test case. Evaluation is explained below.

      Assertion evaluation:

      If result_list contains only **PASS** then specific test case will be declared as **Passed**,
      If result_list contains at least **one FAIL** then the test case is declared as **Failed**.

Structure

Used Multilevel Inheritance:



Class1 - SeleniumDriver (Base Class)

Class2 - BasePage (Child Class)

ClassN - All Page Specific Classes like **AmazonHome**, **BookDetails**, … etc (**Child Classes**)