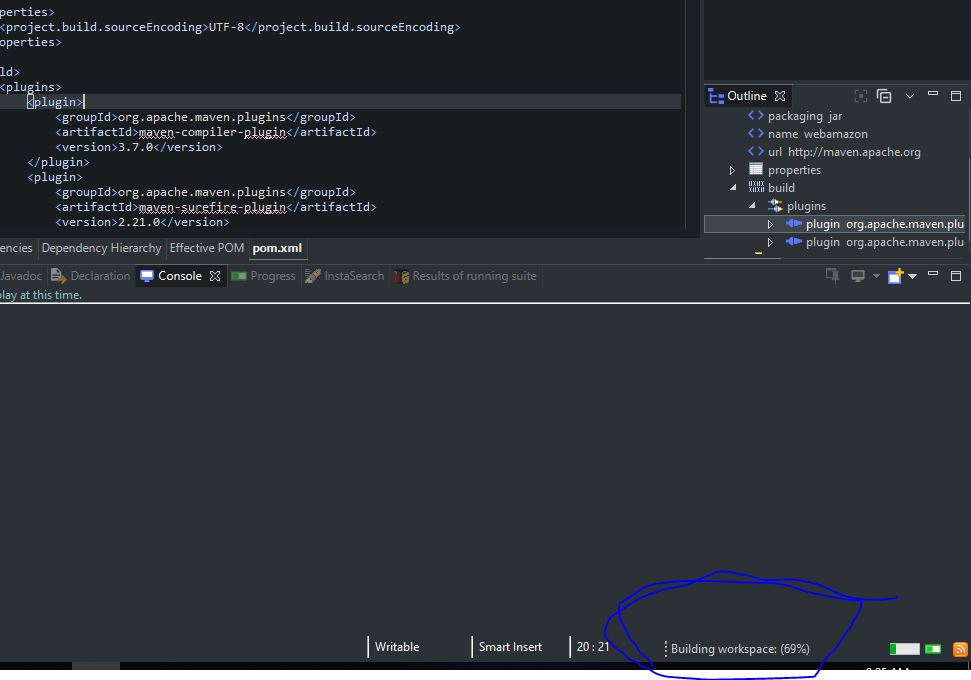
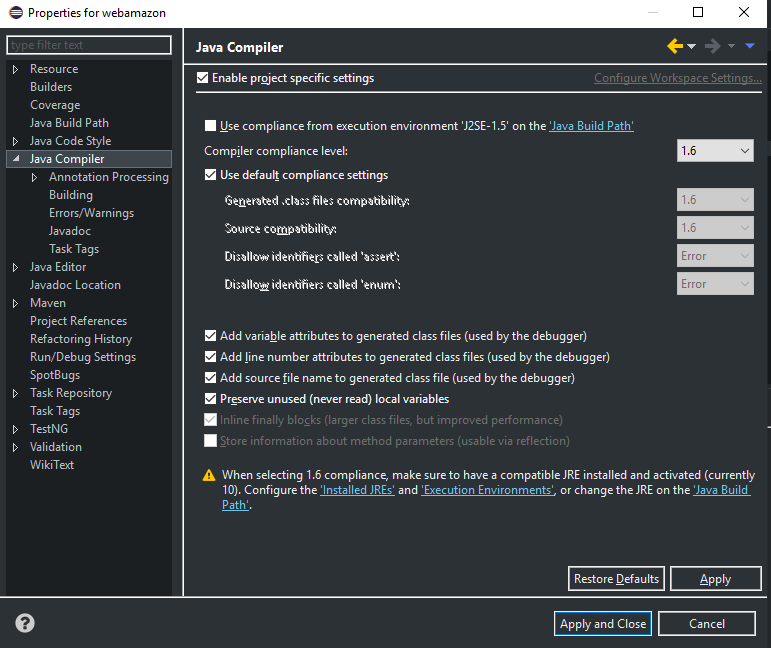
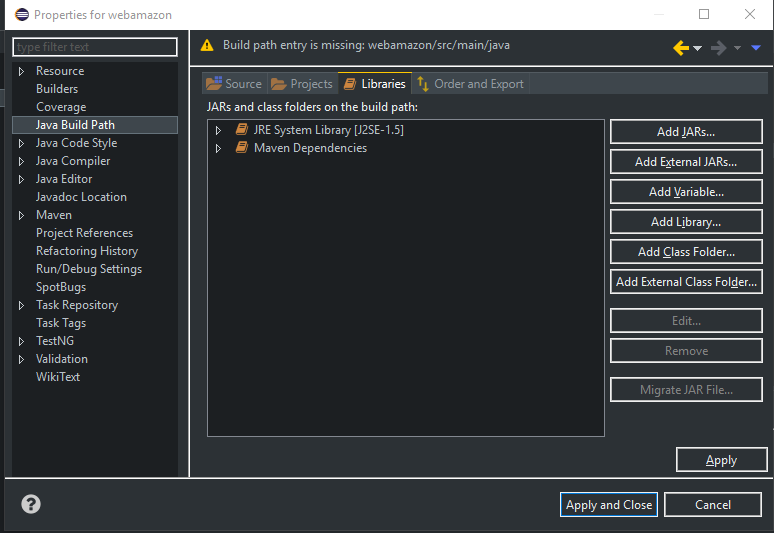
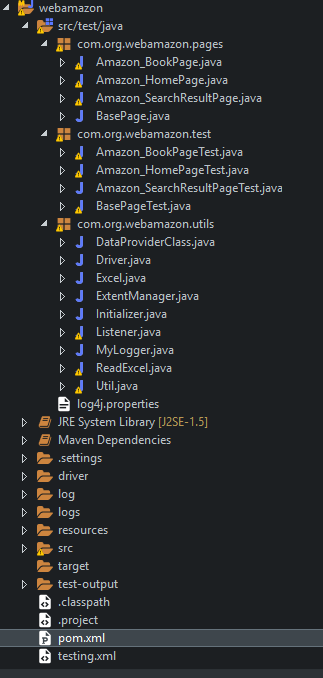
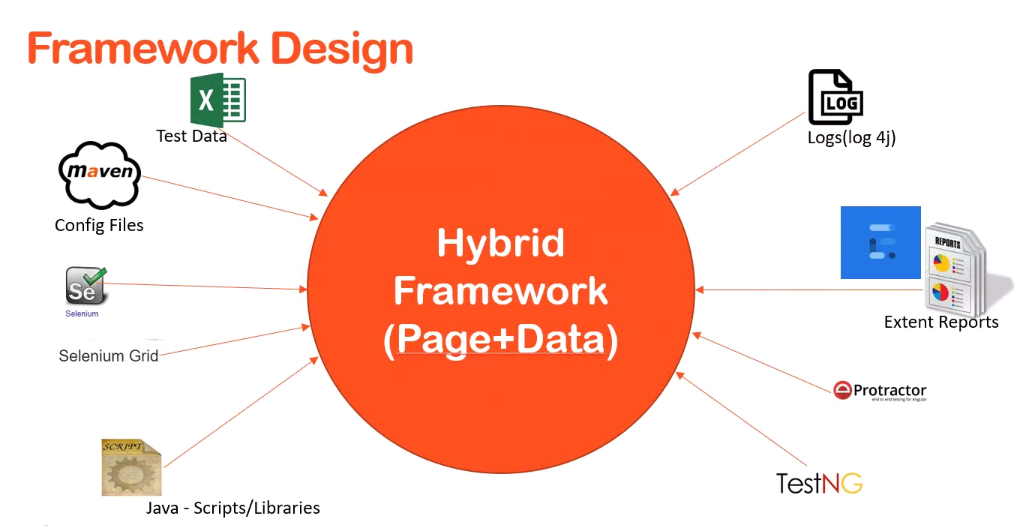
Assignment

1. **What do you need to do?**
   1. Implemented the scenario
2. **Additional question: Document in detail all possible test cases for the above scenario in a spreadsheet.**
   1. I have derived two test cases. Real data may vary at the time of execution
      1. One: All correct scenario
      2. Two: Multiple failures scenario. Red indicates mandatorily made incorrect values to make the test case fail
      3. Test data file can be found at “source path\resources\ExcelData.xlsx”

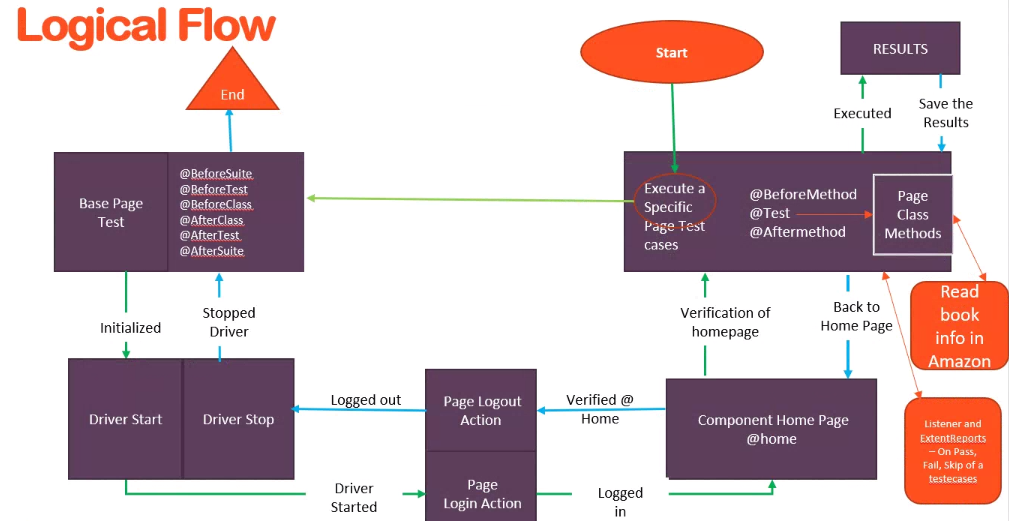
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Search Key | Book Name | Kindle Price | Paper Back | Author 1 | Author 2 | Edition | IBSN |
| data catalog | The Data and Analytics Playbook: Proven Methods for Governed Data and Analytic Quality | $44.49 | $47.35 - $56.50 | Lowell Fryman | Gregory Lampshire | 1st Edition | 978-0128023075 |
| data catalog | The Data and Analytics Playbook: Proven Methods for Governed Data and Analytic Quality | $46.49 | $47.35 - $56.50 | Unknown | Gregory Lampshire | 2st Edition | 978-0128023070 |

1. **Assumptions Made:**
   1. Language of choice – Java
   2. Test Framework – Hybrid (Page Object + Data Driven)
   3. Unit testing Tool – TestNG
   4. Browser – Chrome
2. **Submitting Code:**
   1. Github link - <https://github.com/ersenthilnkumar/amazonweb>
      1. Contains – Source Code, Input Data sheet, drivers
      2. Mail contains – url, read me file, Short Gif video of entire run and Report file view
   2. Instruction to run file:
      1. Sorry I have not used the Mac. So do know how to guide.
      2. So I am guiding for Windows PC
         1. Check out the code from Git
         2. Install Eclipse – Oxygen version
         3. Install JDK – 10
         4. Install Maven in Eclipse using Market place
         5. In eclipse select Import Project menu and open the checked out folder
         6. After importing wait till it downloads the necessary dependencies and Build Workspace process is completed
         7. Make sure you have done some settings in Eclipse as per screen shot as follows.
         8. After importing if nothing happens, open pom.xml file make some edit and undo it and save it.
         9. It will start downloading.
         10. Right click on the testing.xml file run as TestNG file
         11. Execution starts.
         12. For more reference see the attached GIF file in the mail(3rd photo)
         13. See screenshots in next pages
         14. 
3. By When?
   1. Submitted in 4 days
4. What are we looking for?
   1. Good overall documentation/README file.
      1. Attached
   2. Code structure (Page objects etc.).
      1. I have followed the Page object and Data driven modelling
         1. Page Objects – I have created two java files for each pages in two different packages. See below screen shot. Pages package is to hold the functionality in each page(Amazon\_BookPage.java). Ex: I am having function file to verify whether automation is in a test page, Verify a book info for a selected book. Test Package (Amazon\_BookPageTest.java) is to do the unit testing functionality on each methods we created in Page class file. So I can test the isAtSearchedBookPage, verifyBookInfo. I declared all the web elements needs for each pages in class file. Third package is Utils - to hold library and repeatable functions across automation. By combining all the unit test functions, we can form a scenario. Ex: Below is a flow on positive condition
            1. GotoAmazon\_HomePage() – Home Page class methods and tests
            2. isAtAmazon\_HomePage()– Home Page class methods and tests
            3. doSearchBook()– Home Page class methods and tests
            4. isAtAmazonSerchListPage() – Search Page class methods and tests
            5. selectBook()- Search Page class methods and tests
            6. isAtAmazonScearchBookSelectedPage() - Book Info Page class methods and tests
            7. verifyBookInfo() – Book Info Page class methods and tests
            8. 
            9. Testing.xml – is to configure what modules needs to be tested
            10. Pom.xml – to configure essential dependencies to run the framework
            11. Resources folder – to hold input files like Excel Data etc
            12. Logs folder- to have Extent Report for the ran modules, Log4j logs
            13. Drivers folder – hold Chrome and IE drivers
            14. Src- for source files
   3. Design decisions (explain in text).
      1. As Amazon it contains various pages and lot data needs to be verified I have selected hybrid model which comprises of Page Object Model and Data driven model. Below is Frame work design diagram,

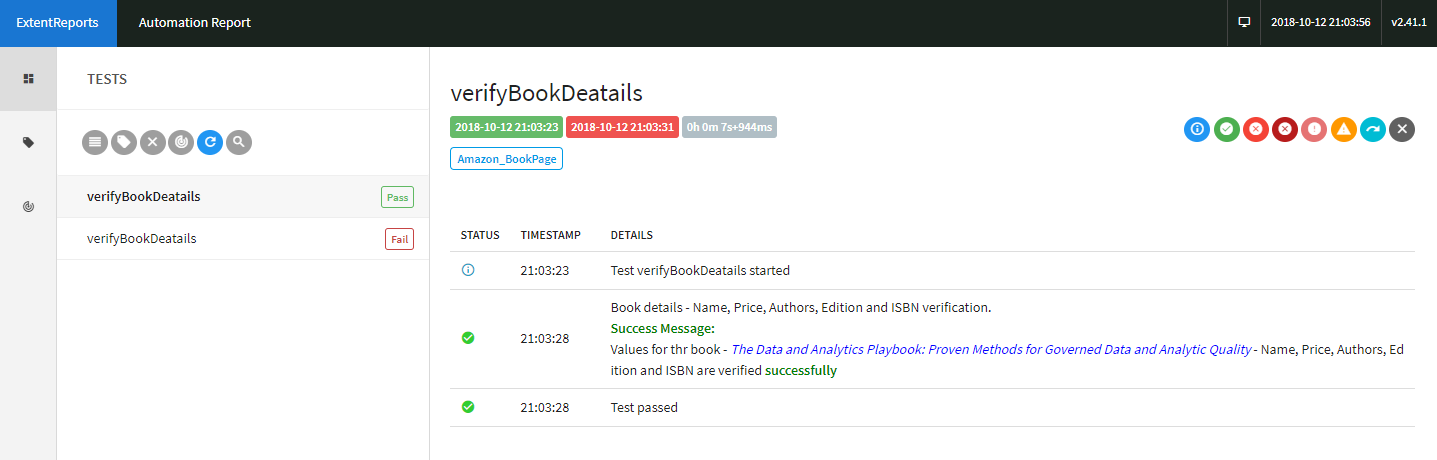
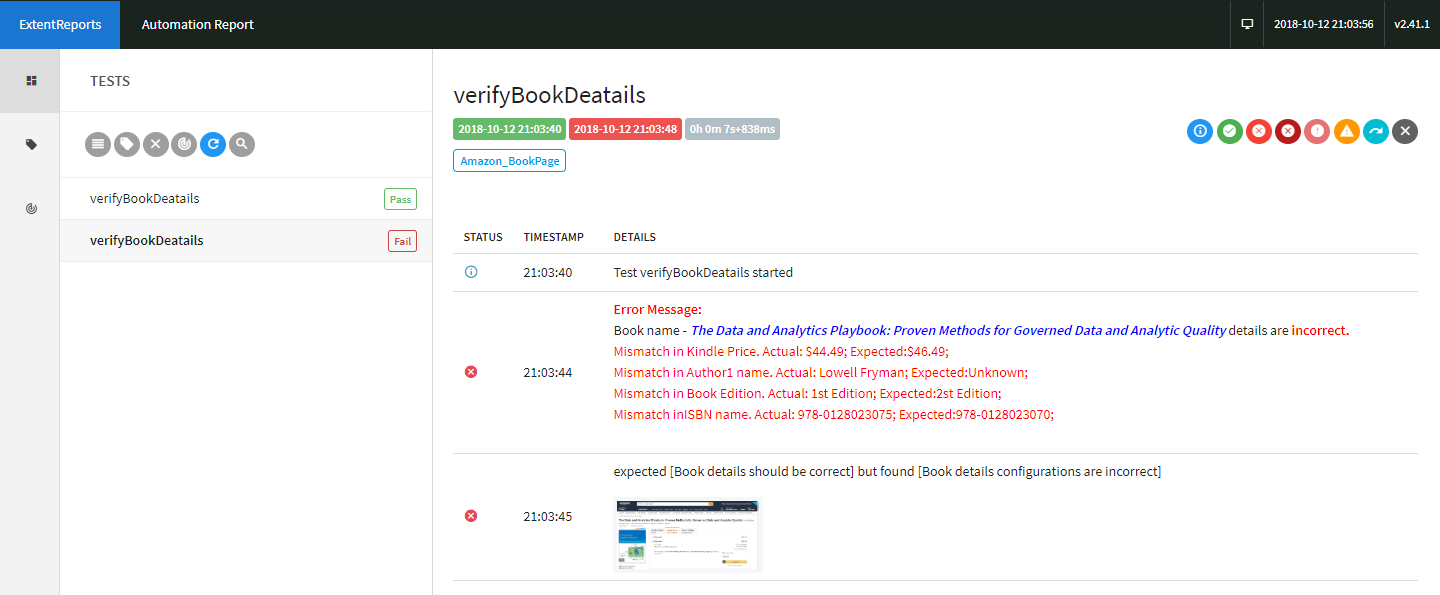
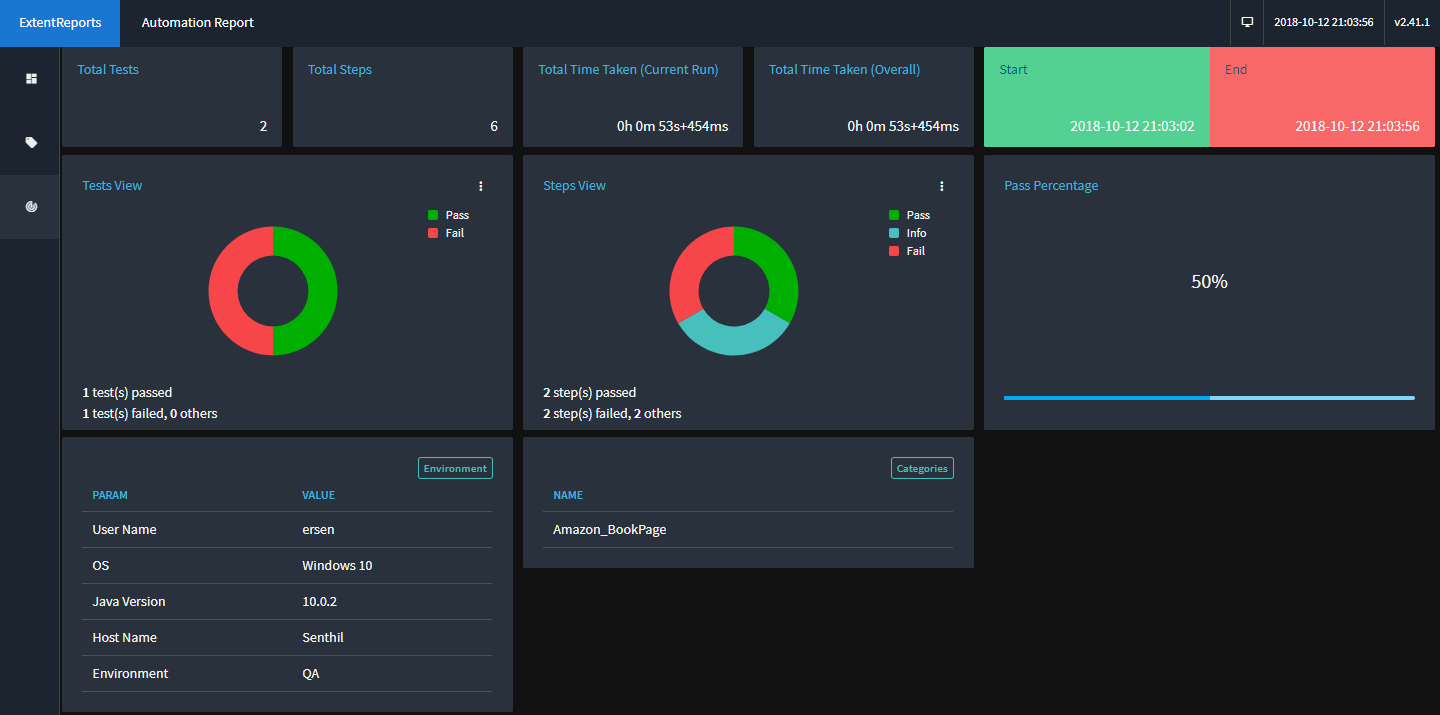


Logical Flow:

1. Below is the logical flow. Green line shows the ramp up process and blue is ramp down process.
2. To start as per our scenario, control enters the Amazon\_BookPageTest file
3. Initialized and assign the global variables using DataProviders
4. Then as per TestNG annotations priority, @BeforeSuite was executed to start the driver, next @BeforeTest was executed to open home page, next @BeforeClass as it was not implemented it was skipped. All these happen in BasePageTest file
5. Then control back to Amazon\_BookPageTest file and executed the @BeforeMethod and does the essential preliminary functions like search and select from a search.
6. Then control goes to @Test, there book info was verified. Here, when success I am calling ExtentReport to log the message and on failure, calling Listener to take screenshot then ExtentReport to log the failure and attach the taken screenshot by Listener with as assertion message.



* 1. Comments in the code.
     1. Added as much as possible
  2. Reporting the result.
     1. Extent Report was used. Find the file inside the logs folder. Refer following diagrams

* 1. Reusable code/componentizing of code.
     1. I have created Utils package to hold repeatable actions like check element is displayed, taking screenshots when tc is failed
  2. Choice of locators, and ways to reduce test script maintenance if locator changes.
     1. I have declared the locator at the start of each Page class file. So that in future if any changes in the property we can just change it there alone

public class Amazon\_BookPage extends BasePage

{

//various web components to select

@FindBy(xpath = "//a[@id = 'breadcrumb-back-link']")

WebElement textBackToSearchResult;

* 1. Assertion types and ease of debugging if test script fails.
     1. I have used assertEquals without using try catch block. Reason I am verifying many info in a single function and assert is to take screenshot by the listener whenever the steps got failed. I have tried to avoid may try catch in a same function as I need to verify many details.

Main Code:

Assert.assertEquals("Search result did not happen for the given book", "Search result should display for the given book");

Extent Report Call for listener Code

test.log(LogStatus.FAIL, "Error Message

Listerner Code:

@Override

public void onTestFailure(ITestResult result)

{

String path = Util.getscreenshot(result.getName() + "\_" + result.getThrowable().toString().split(":")[1]);

test.log(LogStatus.FAIL, result.getThrowable().getMessage() + test.addScreenCapture(path));

MyLogger.log.info("Test "+result.getName()+" failed");

}