TestNG Framework

This document briefly explains the way the framework has been organised, and also the test cases.

This framework has been built mainly using TestNG in java. All the test cases are stored in the **scenarios** package under **test.** We could further group these into modules as well, but for now, I've just placed all the tests under the same package.

Each test class extends the BaseScenario class. BaseScenario handles the initialisation of web drivers and loading the configuration file. I'm using an open source library called **webdrivermanager,** which takes care of updating the web driver binaries. It also allows you to instantiate different browsers without needing to set the system variables. The BaseScenario uses ThreadLocal web drivers so that the test methods can be run in parallel without any conflicts.

Here, the flow is divided into three layers. Test case -> Page layer -> Fragment layer. Basically, the test case will be calling one of the methods in the required pages, which in turn call the methods in the fragment layer which contain the actual implementation. This helps as it keeps the test cases readable, and all the asserts are done in the middle (page) layer, whereas the implementation details are pushed to the outermost layer.

For now, the design is such that each test case gets its test data from the Excel file, and that must be passed as a Map via Data Providers. This ensures that even though you just have a single set of test data for a particular test case, you can easily just add additional rows in the excel to run those cases, without needing to modify anything in the code.

I'll briefly explain the various packages in this project now.

The dataaccess package mainly contains classes which deal with the physical files present on the system. ExcelReader takes care of reading the testdata.xlsx. FileHelper deals with methods pertaining to directory structure, file access etc.. PropertyReader contains a singleton property instance, which contains all the config details, like the URL. This is maintained in config.properties. The classes under excelhelper are mostly just POJOs to make the code readable while parsing and using Excel data.

The fragments package contains actual implementation details of how we use the web driver to interact with the web UI. Each fragment implements an interface that contains all the xpaths corresponding to that fragment. Most xpaths have a token string in them, which can be replaced to some value, which makes it more generic.

The genericshelper helps with the generics that you see in the test cases file. The given, then, when, and, methods allow you to make the code more readable. These methods basically take an object which implements the GenericModule interface. So, all the pages implement this tagging interface to be able to conform with the above methods for readability.

The modules package contains all the page classes. This is the middle abstraction layer, and contains all the actions that can be performed in a given page in the web UI. Each of these extend an AbstractModule class which contains all the fragments which contain the implementation details. This AbstractModule in turn implements the GenericModule interface.

The utilities module contains all the utility classes. It has the BaseScenario, the data provider class and the WebDriverUtils. The WebDriverUtils has methods which wrap over the web driver to provide better support in case of failures. There is a method which acts as a generic wrapper around all the individual actions, and this has been done with the use of lambda expressions.

The xpaths package contains all the interfaces with the xpaths.

Test cases:

1. A round trip flight booking - Upto the Payments section. (2 test cases with different cities and traveller, via data providers.)
2. Search for a flight, and check if the user is able to filter out the flight he wants. I feel this is an important test case as each user will inadvertently use the filters while booking flights to make it easier for him to choose a flight, and a broken filter would provide a really bad user experience to him.
3. Search for a flight, then go back to the start page, check if the previous search is saved, and proceed to book flight from that saved search. This again is important, as I may search for a flight, then go to some other links to get some other details. At this point, having the user type in all the search details again would lead to a bad experience. So, using a saved search to go right back to previous search helps a great deal. Now, checking if a flight can actually be booking after following this path of actions is important, as it would be bad UX if the user couldn't take any actions after going back to the saved search.

Things that I feel could be improved:

1. The test data in the excel file is managed in a pretty primitive way right now. I feel this can be greatly improved by segregating them into separate modules and perhaps using a property file reference so that we don't get tied to a particular schema for all the test cases.
2. The test cases in the test package can be segregated into different modules. Right now, as all the tests are in the scenarios package, it could get really big as newer tests are added.
3. There is currently no support for logging or reporting via email.
4. Some methods in the fragments are quite specialised right now due to time constraints. I would like them to be more generic and reusable. And, there are some assumptions being made, which I would want to get rid of with future fixes.
5. The framework currently has no support to choose what exactly you want to run. You could still specify the test case name in the maven parameters, but having a UI/excel file where you can pick what test cases or modules to run would makes things very easy.

Executing the tests:

1. You can clone the git project from: https://github.com/AkashSharma93/TestNGFramework.git
2. In the project directory, execute: **mvn test**