**Initial report for the xe.gr QA Challenge**

**Author: Panagiotis Kalogeropoulos**

Contents

[Current test setup 3](#_Toc106330888)

[General guidelines 4](#_Toc106330889)

[Part 1 test challenge 5](#_Toc106330890)

[Part 2 test challenge 6](#_Toc106330891)

[Part 3 test challenge 7](#_Toc106330892)

[Contact 9](#_Toc106330893)

# Current test setup

The whole test suite has been created with the following:

**OS**: Windows 10

**Java version**: 1.8

**IDE**: Netbeans 13

**Browser**: Chrome 102

**Chrome driver version**: 102.0.5005.61

# General guidelines

The whole test suite has been created using the Page Object Model (POM) design pattern. All POMs are located in the “gr.xe.selenium.pom” package. Our main POM is the BasePOM object class and all other POMs are its subclasses. POMs have been created for:

* The xe.gr main page (MainPage.class)
* The Login page (LoginPage.class)
* The search results page (ResultsPage.class)

The functionality created for these POMs is the minimum necessary for all our current tests to run, not everything has been modeled as this would take much more time to finish and is not the purpose of this test.

Appropriate comments and Javadoc have been created describing all the decisions and the logic behind my code. Some liberties have been taken due to the fact that I am not totally familiar with the business aspect of things and I do not know what happens in the background of the xe.gr site. In every case all my code decisions have been documented properly.

# Part 1 test challenge

The part 1 test challenge solution can be found in the XESimpleTestSuitejava test file (gr.xe.selenium.qaChallenge package). The test method that runs this solution is the “propertySearchTabTest” method.

Our test needs to check that we get back at least one result. The best way to do that is to check what happens in the app when we have no results and test against that scenario. In our case, when no results appear we get the “No Results found” container. So, if we have at least one result, then this container will not appear.

# Part 2 test challenge

The part 2 test challenge solution can be found in the “XESimpleTestSuite” java test file (gr.xe.selenium.qaChallenge. package). The test method that runs this solution is the “checkPriceFiltersTest” method.

# Part 3 test challenge

The part 3 test challenge solution can be found in the “SelectLandForSaleApplyPlotFilterSaveSearch” java test file (gr.xe.selenium.qaChallenge. package). This scenario was the most intriguing of all as it required maintenance and rewriting but no clear instructions were given on how I should “maintain” the test code.

I kept the same java test file as before and for each existing test method, I created another one with the same name but added the “refactored” tag in each name. For instance, for the method “visitXeSelectLandForSaleFromCategoriesPerformSearch” we created the “visitXeSelectLandForSaleFromCategoriesPerformSearchRefactored” method. I tried to keep for each method the same functionality as before but changed the way it is implemented, using POMs, different test login and assertions. The reason I kept both methods (the old one and the refactored) if for you to have a quicker reference as to what changed in each one of them. Of course, the old methods do not run in our test suite as we have set them the “enabled = false” tag.

In the “visitXeSelectLandForSaleFromCategoriesPerformSearchRefactored” method is where we test the main scenario of this test challenge which is (copying from your pdf):

*So the test is to select the Land for Sale category, perform a search, when in results select the land plot filter, try to save this search, log in and then try again.*

Since we need to test login functionality, we have to start our tests in a clean state – so we need to check if we are logged in or not. There is a variety of ways to check that but for this particular test I am using just a quick url matching. You can see more comments and details in the implementation in the BasePOM.isUserLoggedIn and the BasePOM.logout methods.

Another important aspect of this test is how we handle our test user’s data. It is a major security issue to have sensitive user data (such as usernames and passwords) hardcoded in our tests, so we need to store that data elsewhere. There are a few great ways we can eliminate this problem (user encryption, a separate database or even Jenkins pipelines) but for the time being, the solution that I incorporated was to create a csv file (located in the src\main\config\ credentials.csv).

Also, a utility class object named TestUser has been created that handles all the business logic behind the retrieval of those data, decoupling all that functionality – clients of the TestUser do not know where and how these data were retrieved.

Finally, one big issue we need to deal with is Localization and asserting text and messages. It is very easy and tempting to insert hardcoded text but in the long run this is a nightmare. And that is because texts and messages are some of the most frequently changeable parts in an application. We need a way to handle texts without having to care if they change all the time or what Locale we test against. So that is why I created the LocalizationReader class (located in the “gr.xe.selenium.utilities” package) that offers (for now at least) a very simple way to get the texts from a centralized place. This place is for now a json file in the src\main\localization\ dummyResources.json path, where we read it and extract (using LocalizationEnum) the appropriate message.

# Contact

Is there something I may have missed? Do you have any additional questions or feedback? For any reason whatsoever you can contact me:

By email: [p.kalogeropouloss@gmail.com](mailto:p.kalogeropouloss@gmail.com)

Or

By phone: +306994663141