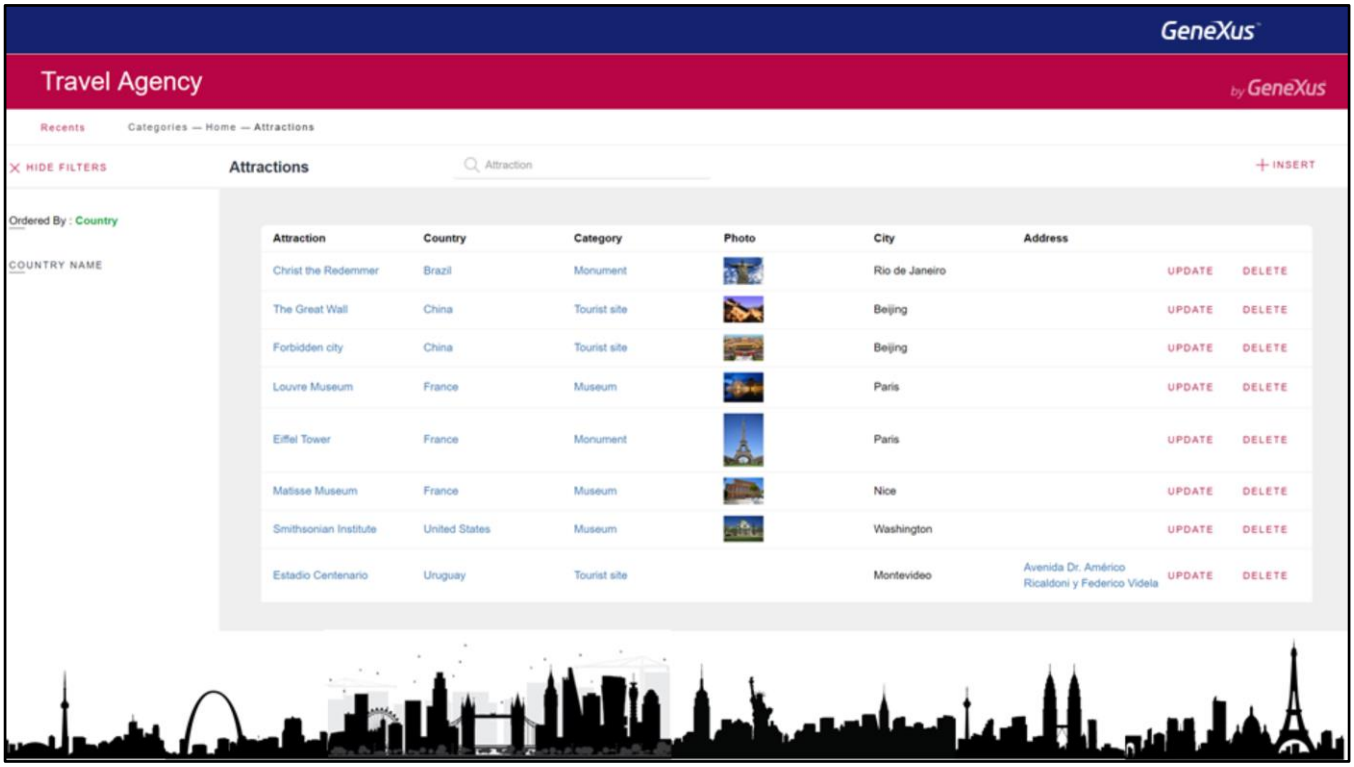
Application testing in GeneXus

User Interface test (UI test) and introduction to GXtest

GeneXus 16



As we make our application for the travel agency grow, we add functionalities and make changes to things that we had implemented before.

However, something we missed was testing the application again after making changes to make sure that what was already functioning continues to work correctly.

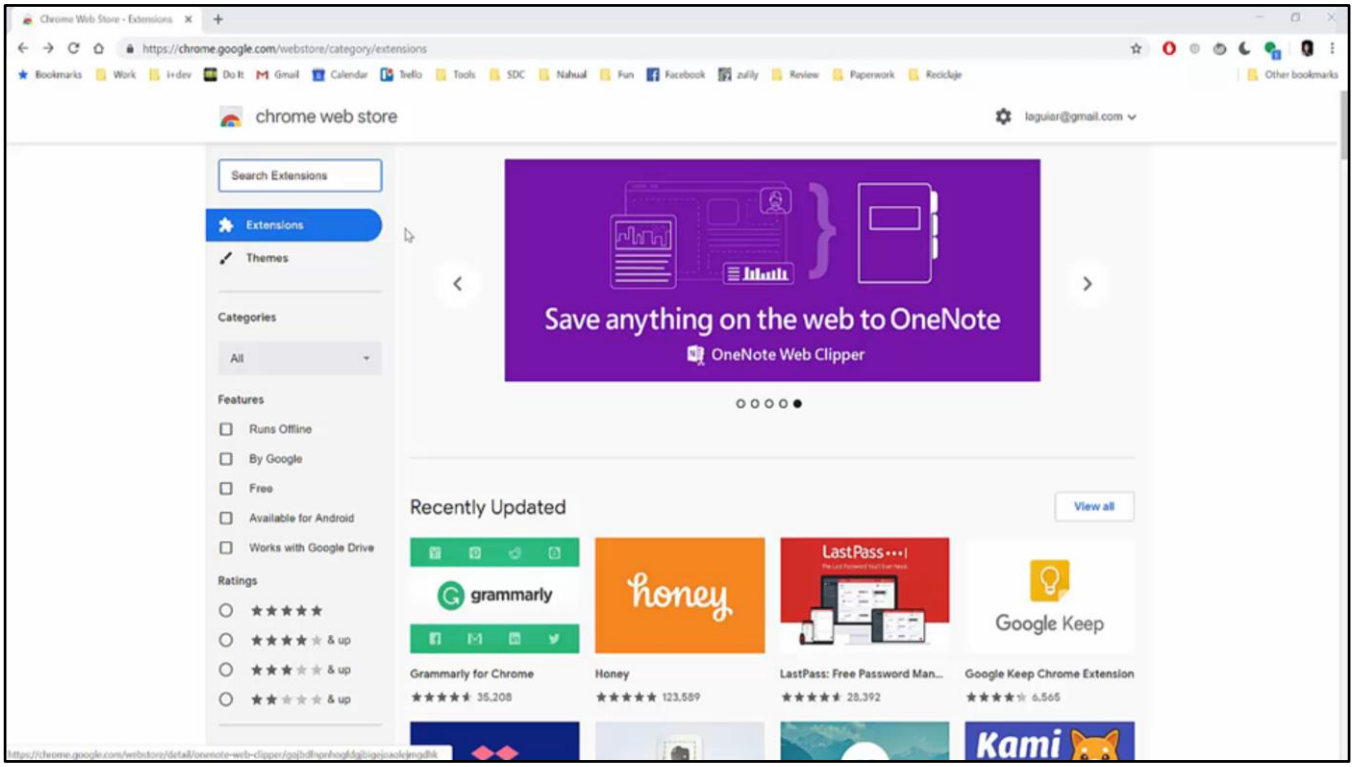
This sort of tasks may become quite tedious with large applications and mostly the fact of repeating tests of things already tested. Though necessary, it's very boring to do.

GeneXus aids us in the automation of such tests with its Gxtest tool.

GXtest enables us to define tests that are later automatically reproduced, as if it were an individual entering data and verifying that the system functions correctly.

We will see this with an example.

Suppose that, for the business of our travel agency, working with attractions –that is, the possibility of creating, modifying and deleting attractions– is a main aspect and if we have an error in that process, that would have a great impact on our business. So, once this process has been programmed and when we have validated that it is actually functioning properly, we will want to capture that validation so that it may be done automatically in the future as a form of regression testing.



[DEMO: https://youtu.be/K2ZeWgYyjdk]

The first thing we need to automate our tests is the GXtest Recorder.
This tool enables us to capture the manual execution to then take it to GeneXus. The GXtest Recorder is an extension of google- Chrome that is free and may be downloaded from Chrome store.
With the extension installed, we will have it available in Chrome, like other extensions we may have.
After installing the extension we may execute it from the Chrome tool bar, and when we open the recorder, we press the camera button to start recording.
With the recorder active, we execute the app as we usually do –in the same way as we would do it to validate this manually. For example, we will start by clicking on the Insert button to add a new attraction – note the pop-up in the lower right corner with a message indicating that the action we have just executed was captured by the recorder; this will happen for every action we execute.

We will now create a new attraction, with an existing name, in a country that already exists and we expect a validation stating an error for that. We will then indicate that we want that validation to be captured.
And we say that we expect this text to appear here, so we create an AssertText.
When we create that Assert we are stating that we expect this error message to appear, that is, if we create an attraction with the name 'Centenario Stadium' for country 'Uruguay' and we DO NOT get the error message then the test would be failing.

Now we go on with out test indicating that if we state a new –different– name, then we expect the Attraction to be recorded. We enter the values for this new attraction created and we confirm.

In the recorder we will see each action we executed, as well as the points where we clicked, and the spots where we wrote a value, and so on. This is the test script that will be automatically executed later.

Moving on with our test we will now validate that the attraction for which we will search has been created

and we do a new Assert, indicating that we expect this value to appear here.

Now we will modify it, because our test is a full cycle. We want to make the entry, then modify and also delete. We will change the name to Estadio-Centenario-3, and we confirm... Now we will search for "Estadio Centenario 3' and again we do an Assert indicating that we expect this value to appear because, if everything went ok and the edition was possible, then the name 'Estadio Centenario 3' would have to appear here. .
In the end, we will delete it, and we will also do an Assert to make sure that the confirmation message appears. Then we confirm to complete the full cycle of our test.

--
Once we have completed the test, we turn the recorder off so as to finalize the capture and now we will have our test script written here.
We can save this script and execute it from here as many times as we want.

What we will do now is click on the "Play" button so that the test is reproduced as we expect. As the script is executed we can see that steps are shown in green when the execution has been successful (that is, when we have been able to click on the control expected, and when we were able to enter the value expected and the Asserts are actually taking place).

From the Recorder, in addition to recording and executing tests, we may also edit the script - deleting unnecessary steps. In executing the test we could have made unnecessary clicks and we can delete those, and we may also add further validations. Once we are satisfied with our test validating what we expect, we are then ready to take our test to GeneXus.

RECORDER →GENEXUS

Even when the automatic integration is not done yet, it is very simple to take the test to our KB, exporting it by means of a click on the Download button to generate a script –text file- that is GeneXus code.
From here we will copy the code and then we paste it on a special object in GeneXus.

We then go to the KB and create a new object of the UITest type, with the name "TestAttractions", and there we paste the code captured in the recorder.

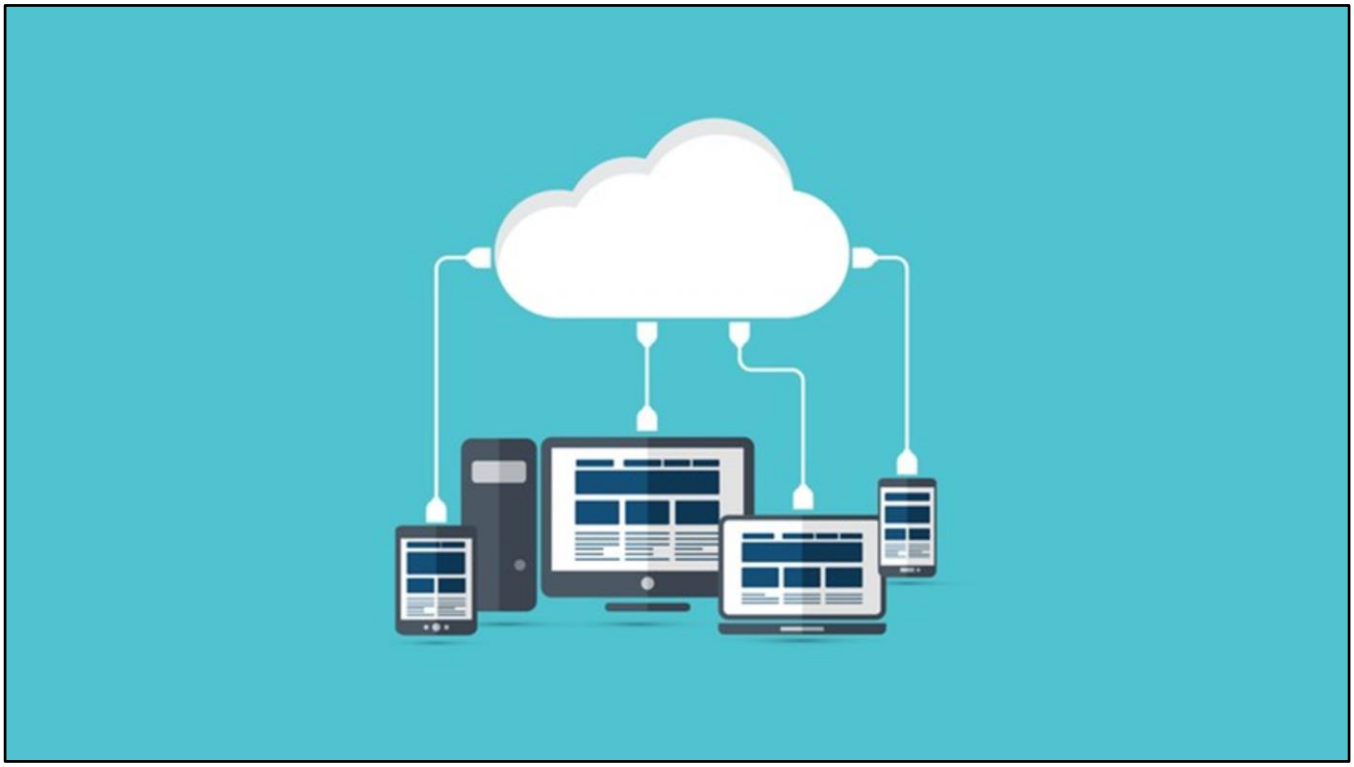
UI-TEST
The UI-Test object is a new Object in this version of GeneXus that requires the Gxtest license to be used. What's interesting about having the tests within the KB is that they are then part of our knowledge base and therefore we will be able to do test versioning, in addition to sharing and extending the tests.

This new object uses functions of GXtest that enable us to control the browser using different commands –for example opening the browser in a given URL, selecting/activating different controls, making validations, that is to say Asserts, on screen, and so on.

The Recorder makes the task easier for us by writing the code automatically. But, once in GeneXus, we may extend the test, and perhaps feed the data to be tested from a dataprovider, or add validations to the database (for example, we could verify that the record actually does not exist in the DB after we delete it on screen, for which we did not include any Assert in the test, so we could add an Assert here, validating that the record does not exist in the DB).

We may execute our script in the same way as we executed the unit test, with right button and selecting the "Run This Test" option, or also from the Test-Explorer, selecting the interface test.

We may also change the browser where our tests are executed, specifying that we want to use Firefox instead of Chrome –which is the default browser- and we may execute the tests from here (7:35)



The idea is not to execute the tests from the GeneXus IDE but rather execute it once my series of tests is built and it is functioning locally, in some other place in order to avoid having all browsers and all operating systems installed in my computer where I want the testing.

We may instruct GXtest that we want to execute the tests in the cloud, for example in a tool like SauceLab, which is a possible cloud for running functional tests on different operating systems and browsers.

You will find further information on this in our wiki, in the URL shown on screen.

https://wiki.genexus.com/commwiki/servlet/wiki?41131,Saucelabs+GXtest,

CONTINUED INTEGRATION

The kind of interface tests we have seen is widely used for validating whole functional cycles of those things in our business that SHOULD NOT fail.

In sum, we want to make sure that, with these tests, when a user goes through the happy path –or rides on the route's yellow line, meaning that nothing strange is done and he stays on his way using the critical functionality of our business– we will know that the flow will be successful and no errors will be encountered.

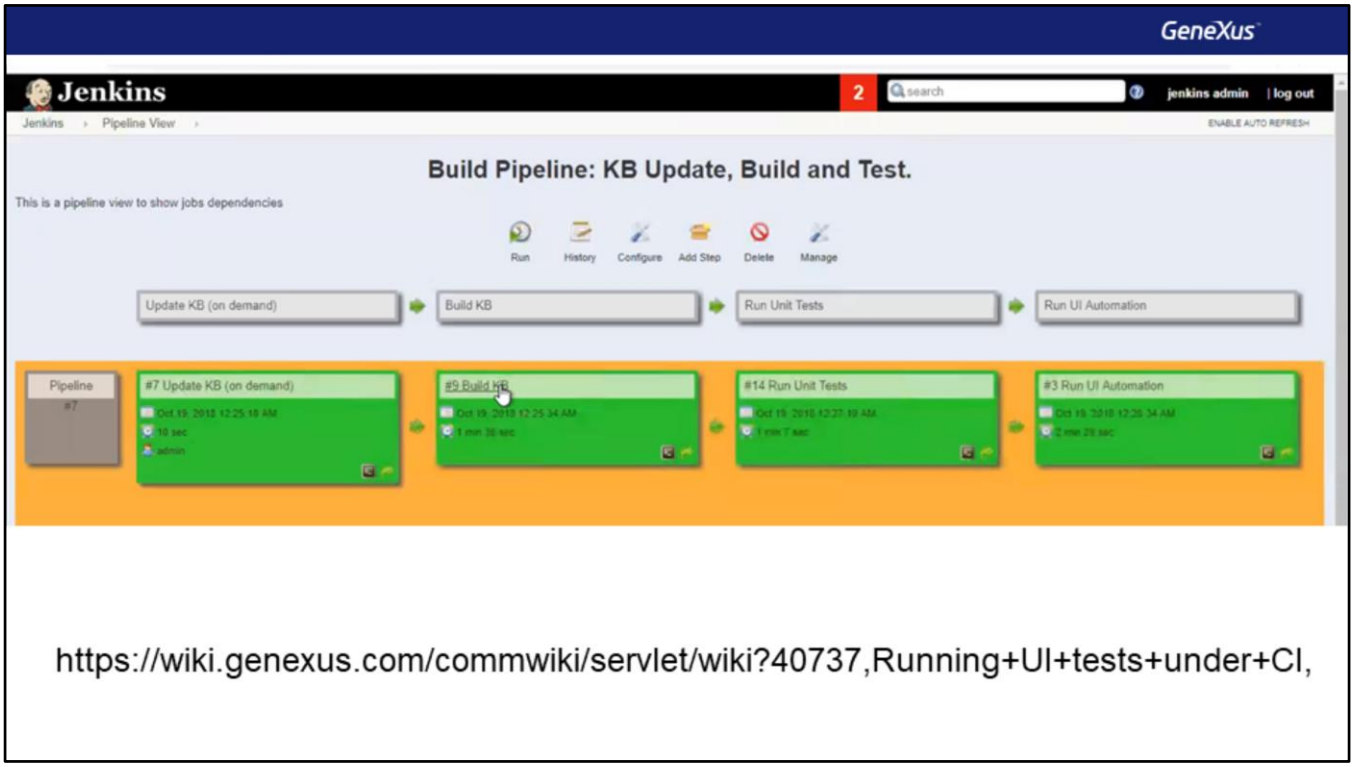
These are the flows that we will automate first because they are the most important flows –and what we want to be sure of is that it will never break when we introduce changes to the application.

So far we have seen how a test is made automatic. However we were executing it manually.

What's interesting in these tests is that their execution should occur in an automatic manner.

That means that the test execution will be part of our automated process for continued integration with Jenkins.

We will not see the details in this course, but just to mention it, Jenkins is a Continued Integration engine that enables us to automate the execution of the various steps that are part of the set up in a build of our application.



The typical example of a Pipeline in Jenkins –that is, a sequence of steps we must execute when setting up a new build of our app– is the following:

1. In the first step, an Update is done in the KB where we set up the application. This means that we bring, from GeneXus Server, all the objects that have been modified since the last update.
2. A Build is done in GeneXus –meaning that the objects with changes that were just downloaded from GeneXus Server are reorganized, specified and compiled.
3. In step three, the unit tests are executed.
4. If everything is ok, the UI test are then executed in the end.

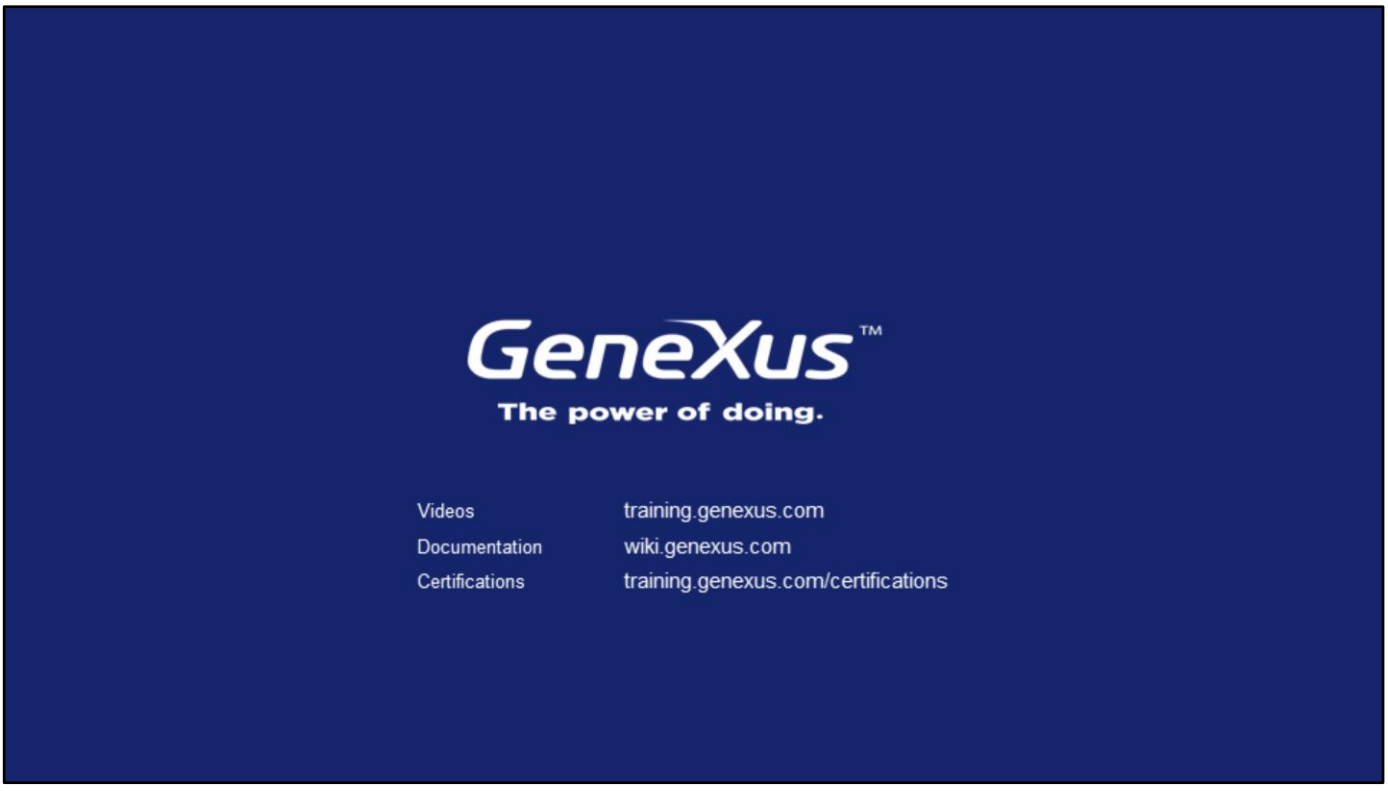
After all this –and depending on the tests' degree of coverage– the application's Build will be ready for release to production, or for a tester to manually perform the tests of the new functionality.
The idea is that, when the tester appears, we already know, at least, that all the regression tests are ok, and so are all the unit tests of the new functionality. So what remains is to do the integration test, or the full user cycle for the new functionality to be released.

You will find further information on Continued Integration and on automatic test execution in our wiki.--
https://wiki.genexus.com/commwiki/servlet/wiki?40737,Running+UI+tests+under+CI,

---

GXtest is a great ally that aids us in making our app reliable while reducing the test time because it makes the generation and execution of automatic tests easy.

In addition to, it is also a fundamental tool in the automations required for a continued integration process, enabling us to build software in an agile manner, preserving quality aspects.
For further information on this took, go to the URL shown on screen.



Documentation = DOCUMENTATION
Certificaciones = CERTIFICATIONS