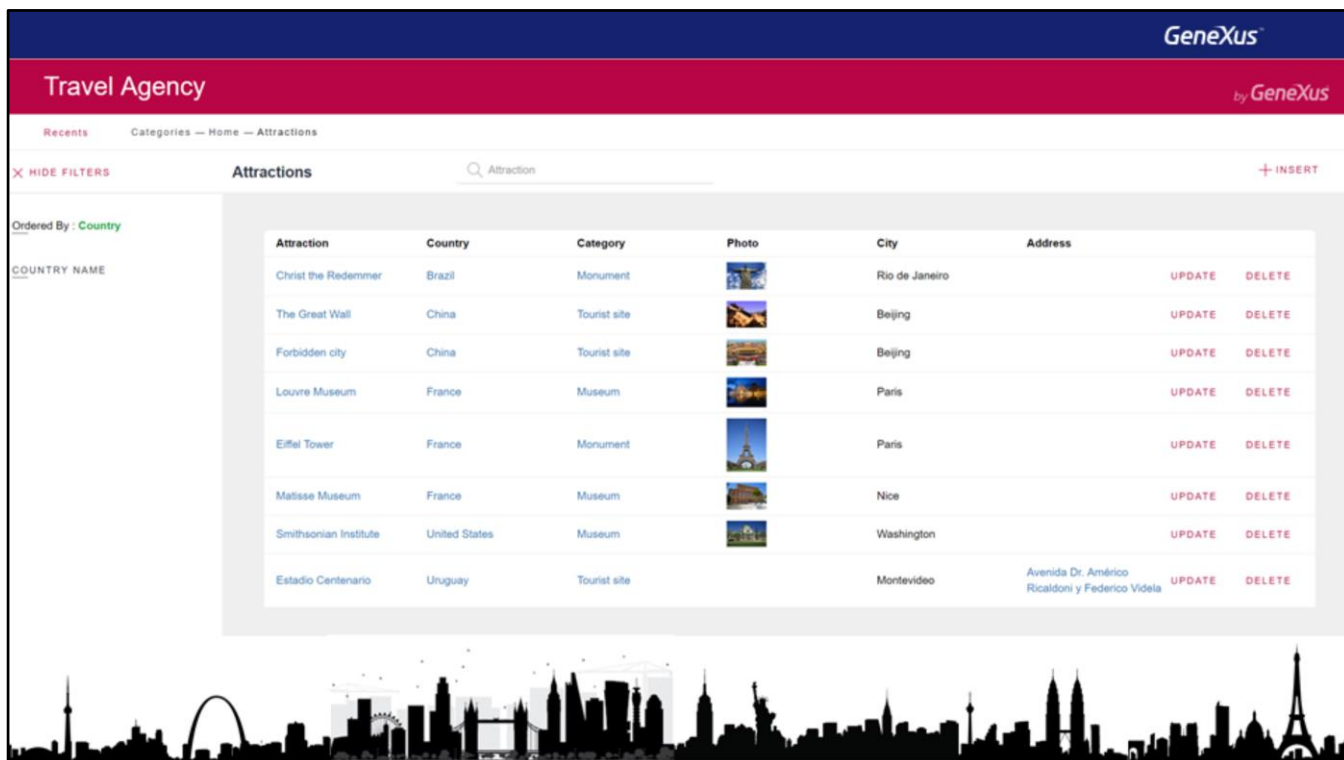


Testing de aplicaciones en GeneXus

Test de Intefaz de Usuario (UI test) e introducción a GXtest

GeneXus™ 16



A medida que hemos ido haciendo crecer nuestra aplicación para la agencia de viajes, hemos ido agregando funcionalidades y haciendo modificaciones a cosas que ya habíamos implementado antes.

Sin embargo algo que hemos omitido es volver a probar la aplicación, luego de hacer un cambio, para asegurarnos que lo que ya teníamos funcionando, se siga comportando correctamente.

Este tipo de tareas puede volverse muy tediosa si la aplicación crece mucho, y sobre todo repetir pruebas de cosas ya probadas – si bien es necesario – es bastante aburrido.

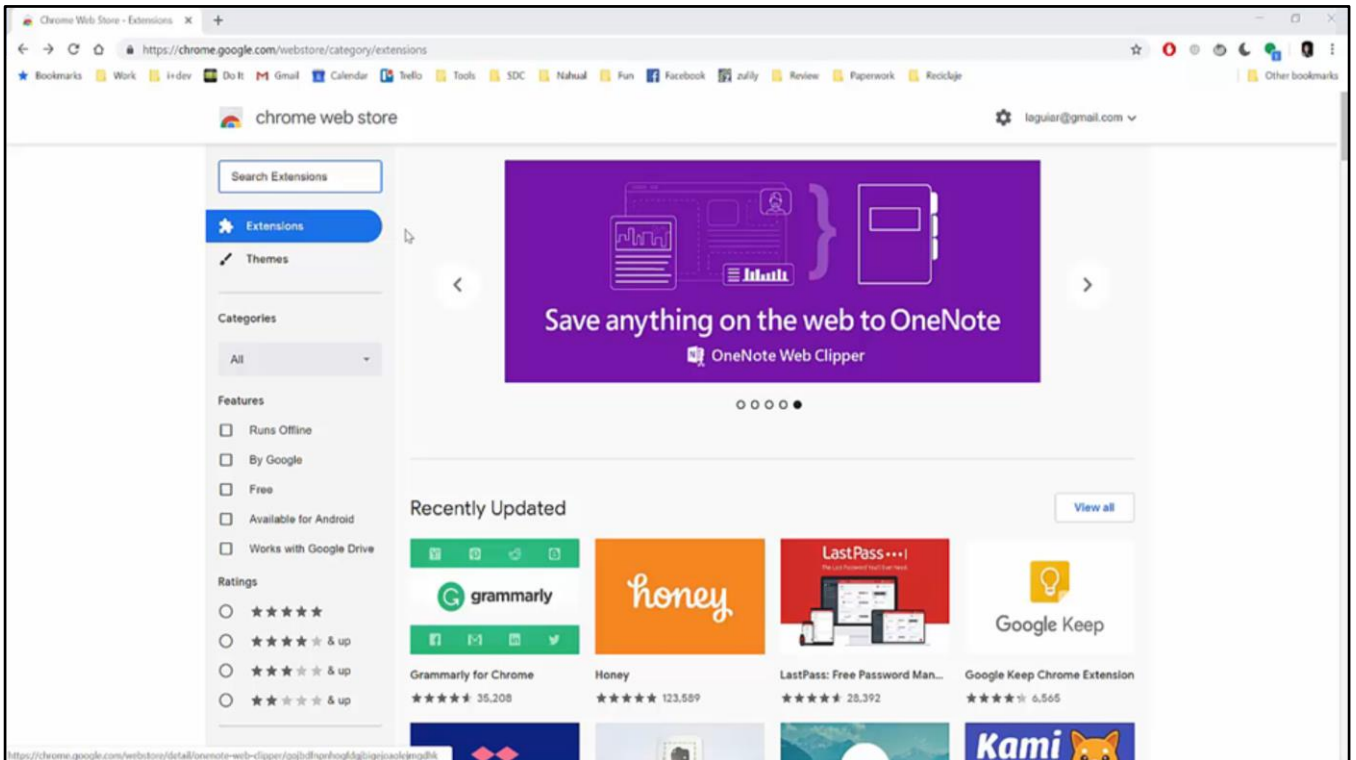


GeneXus nos ayuda en la automatización de estas pruebas, mediante su herramienta GXtest.

GXtest nos permite definir pruebas que luego se reproducen automáticamente, como si fuera una persona que está ingresando los datos y verificando que el sistema funciona correctamente.

Vamos ver esto con un ejemplo

Supongamos que para el negocio de nuestra agencia de viajes, el trabajar con atracciones – es decir, poder crear , modificar y borrar atracciones – es fundamental, y si tuviéramos un error en dicho proceso tendría un gran impacto en nuestro negocio, por ello una vez que este proceso está programado y que ya validamos que funciona bien, lo que queremos es capturar esa validación para que se pueda realizar de forma automática en el futuro como forma de test de regresión.



[DEMO: <https://youtu.be/K2ZeWgYyjdK>]

Para automatizar nuestras pruebas, lo primero que necesitamos es el GXtest Recorder.

Esta herramienta nos va a permitir capturar la ejecución manual y luego vamos a poder llevarla a GeneXus.

El GXtest Recorder es una extensión de google- Chrome, que es gratuita, y se puede bajar del Chrome store.

Una vez que la extensión está instalada, la vamos a tener disponible en Chrome al igual que otras extensiones que podamos tener.

Una vez que instalamos la extensión podemos ejecutarla desde la barra de herramientas del Chrome y al abrir del grabador, podemos apretar en el botón de la cámara para comenzar a grabar.

Una vez que tenemos el recorder activo, lo que vamos a hacer es ejecutar la aplicación de forma normal – haciendo la tarea que haríamos normalmente si quisiéramos validar esto a mano – por ejemplo vamos a comenzar haciendo clic en el botón Insertar para agregar una nueva atracción – vieron que en la esquina inferior derecha nos apareció un pop-up, un mensaje, que nos indica que la acción que acabamos de ejecutar fue capturada por el recorder; esto nos va a aparecer en cada acción que ejecutemos.

Vamos ahora a crear una nueva atracción, con un nombre que ya existe, en un país que ya existe y esperamos que nos salte una validación que nos diga que eso es un error. Vamos a indicar entonces que queremos que esta validación sea capturada, decimos que esperamos que este texto aparezca aquí, por lo cual creamos un AssertText.

Al crear este Assert lo que hacemos es decir que esperamos que este mensaje de error aparezca, es

decir, si creamos una atracción con el nombre 'estadio centenario' y ponemos país 'Uruguay' y NO nos aparece el mensaje de error entonces la prueba fallaría.

Ahora vamos a continuar nuestra prueba, indicando que si ponemos un nombre nuevo – diferente – ahora si esperamos que la Atracción se grabe. Ingresamos los valores de esta nueva atracción que estamos creando y confirmamos.

Si miramos el recorder vamos a ver cada una de las acciones que fuimos ejecutando, los lugares donde hicimos clic, donde escribimos un valor, etc. y esto es el script de test que luego se va a ejecutar de forma automática.

Vamos a continuar nuestra prueba y vamos a validar que se haya creado la atracción para lo cual la vamos a buscar, y vamos a nuevamente a hacer un Assert indicando que esperamos que este valor aparezca aquí.

Ahora vamos a modificarlo, porque nuestra prueba es un ciclo completo, queremos dar de alta, queremos modificar y queremos eliminar, vamos a cambiarle el nombre a Estadio-Centenario-3, confirmamos... ahora vamos a buscar por "Estadio Centenario 3" y nuevamente vamos a hacer un Assert indicando que esperamos que este valor aparezca porque si todo fue bien y se pudo hacer la edición, entonces el nombre 'Estadio Centenario 3' tendría que estar apareciendo aquí. Y finalmente lo vamos a borrar, vamos a hacer también un Assert asegurándonos que aparezca el mensaje de confirmación.

Y luego confirmamos completando el ciclo completo de nuestra prueba.

--

Una vez que completamos la prueba, apagamos la grabadora de forma de finalizar la captura y de esta manera tenemos escrito aquí nuestro script de test.

Este script podemos guardarlo y ejecutarlo desde aquí tantas veces como queramos.

Lo que vamos a hacer ahora es hacer clic en el botón "Play" para que la prueba se reproduzca como esperamos.

Vamos viendo a medida que el script ejecuta, que los pasos se van marcando en verde en caso de que su ejecución haya sido exitosa (es decir, se haya podido hacer clic en el control esperado, que se haya podido ingresar el valor esperado y que los Asserts se cumplan).

Desde el Recorder, no solo podemos grabar y ejecutar pruebas, sino que también podemos editar el script - eliminando pasos que sean innecesarios – quizá al ejecutar la prueba hice clics que no eran necesarios y podemos eliminarlos – o podemos también agregar validaciones adicionales. Una vez que estamos satisfechos con que nuestra prueba valida aquello que esperamos, entonces estamos prontos para llevar nuestra prueba a GeneXus.

RECORDER → GENEXUS

Si bien no esta hecha aun la integración automática, es muy sencillo llevar es test a nuestra KB, lo que tenemos que hacer es exportar la prueba haciendo clic en el botón Download, y esto nos va a generar un script – un archivo de texto- que es código GeneXus.

Desde acá vamos a copiar el código, y luego lo vamos a pegar en un objeto especial en GeneXus.

Vamos entonces a la KB y vamos a crear un nuevo objeto de tipo UITest, al cual vamos a nombrar "TestAttractions", y allí vamos a pegar el código que capturó el recorder.

UI-TEST

El objeto UI-Test es un nuevo Objeto en esta versión de GeneXus y requiere licencia de GXtest para utilizarse.

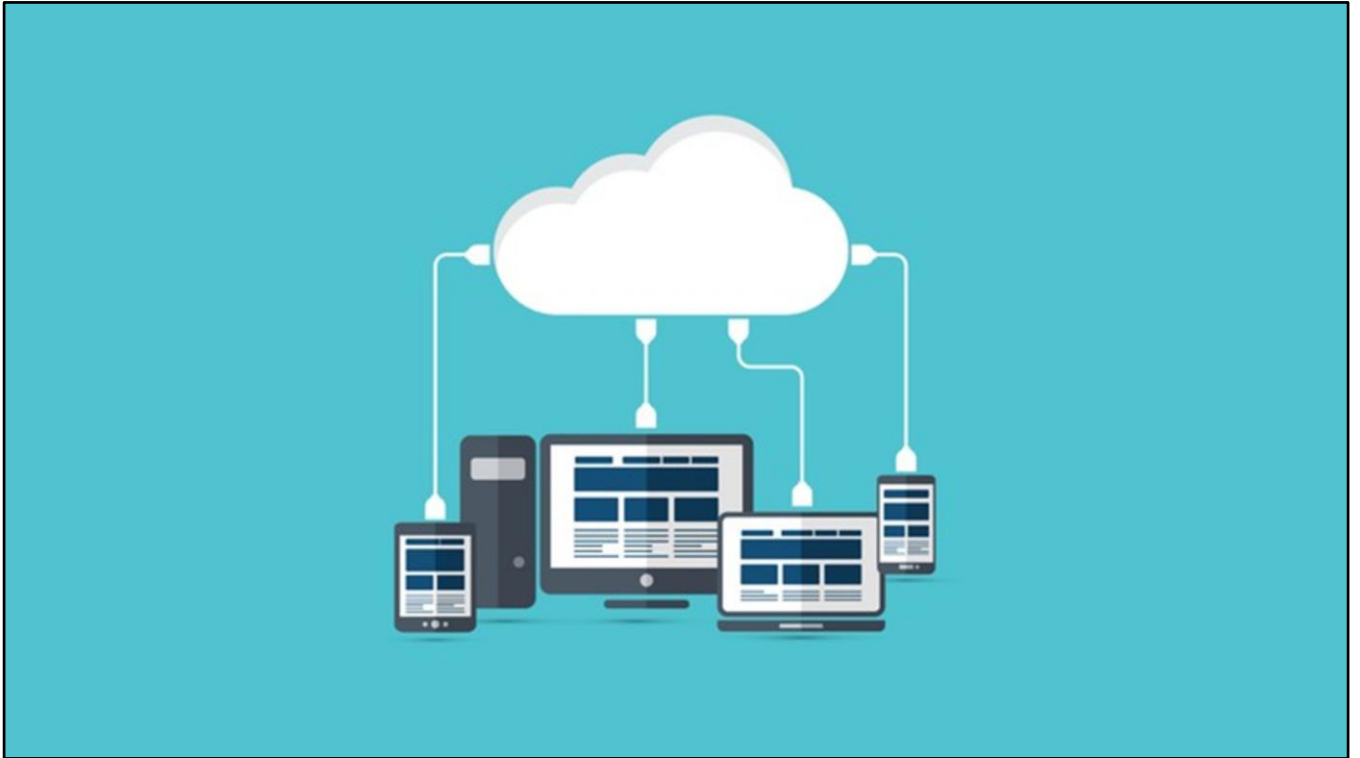
La gracia de tener los tests dentro de la KB es que pasan a formar parte de nuestra base de conocimiento y por tanto los tests van a poder versionarse, compartirse y extenderse.

Este nuevo objeto, utiliza funciones de GXtest que nos permiten controlar el browser utilizando diferentes comandos – por ejemplo abrir el browser en cierta URL, seleccionar/activar diferentes controles, realizar validaciones – es decir Asserts –en pantalla, etc.

El Recorder nos facilita la tarea, escribiendo el código automáticamente, pero una vez en GeneXus podemos extender la prueba, quizá alimentar los datos a probar desde un dataprovider, o agregar validaciones en la base de datos (por ejemplo podríamos comprobar que efectivamente no exista el registro en la BD luego que lo borramos en la pantalla, para lo cual en el test no habíamos hecho ningún Assert, entonces podríamos agregar aquí un Assert validando que el registro no exista en la BD).

Podemos ejecutar nuestro script de la misma manera que ejecutamos las pruebas unitarias, dando botón derecho y seleccionado la opción “Run This Test” o incluso desde el Test-Explorer seleccionando el test de interfaz

Podemos cambiar también el navegador en que ejecutan nuestras pruebas especificando que queremos utilizar Firefox en vez de Chrome – que es el navegador por defecto- y podemos ejecutar las pruebas desde aquí (7:35)



No es la idea ejecutar las pruebas desde el IDE de GeneXus sino que una vez que mi batería de pruebas esta construida y funciona localmente, lo que voy a hacer es ejecutarlo en algún otro lugar para no tener que tener instalados todos los browsers y todos los sistemas operativos donde quiera probar, instalados en mi máquina.

Es posible indicarle a GXtest que queremos ejecutar las pruebas en la nube, por ejemplo en una herramienta como puede ser SauceLab que es una posible nube para correr pruebas funcionales en diferentes sistemas operativos y navegadores.

Puede obtener mas información sobre esto en nuestro wiki en la URL q se ve en pantalla.

[https://wiki.genexus.com/commwiki/servlet/wiki?41131,Saucelabs+GXtest,](https://wiki.genexus.com/commwiki/servlet/wiki?41131,Saucelabs+GXtest)



INTEGRACION CONTINUA

Este tipo de pruebas de interfaz que hemos visto, se usa mucho para validar ciclos funcionales enteros de aquellas cosas en nuestro negocio que NO pueden fallar.

En definitiva, queremos asegurarnos con estas pruebas, que si un usuario sigue el camino feliz – o va por la línea amarilla de la ruta, es decir, no hace nada raro, no se aparta del camino y está utilizando la funcionalidad crítica de nuestro negocio – saber que ese flujo va a ser exitoso y no se a encontrar con ningún error.

Estos son los flujos que vamos a automatizar primero, ya que son los más importantes – y es lo que queremos asegurarnos que nunca se rompe cuando realizamos cambios en la aplicación.

Hasta ahora hemos visto cómo automatizar una prueba, pero la estamos ejecutando manualmente.

La gracia de estas pruebas es que su ejecución suceda en forma automática.



Es decir, que la ejecución de las pruebas, sea parte de nuestro proceso automatizado de integración continua con Jenkins.

No vamos a entrar en detalles en este curso, pero Jenkins es un motor de Integración Continua que nos permite automatizar la ejecución de los diferentes pasos que forman parte del armado de un build de nuestra aplicación.

Build Pipeline: KB Update, Build and Test.

This is a pipeline view to show jobs dependencies

Run History Configure Add Step Delete Manage

Update KB (on demand) Build KB Run Unit Tests Run UI Automation

Pipeline #7

#7 Update KB (on demand)
Oct 19, 2018 12:25:18 AM
13 sec
admin

#9 Build KB
Oct 19, 2018 12:25:34 AM
1 min 30 sec

#14 Run Unit Tests
Oct 19, 2018 12:27:19 AM
1 min 7 sec

#3 Run UI Automation
Oct 19, 2018 12:38:34 AM
2 min 28 sec

https://wiki.genexus.com/commwiki/servlet/wiki?40737,Running+UI+tests+under+CI,

El ejemplo mas típico de un Pipeline en Jenkins - es decir, de una secuencia de pasos que es necesario ejecutar cuando se arma un nuevo build de la aplicación es el siguiente:

1. En el primer paso se hace un Update en la KB donde armamos la aplicación. Es decir, se traen del GeneXus Server todos los objetos que hayan sido modificados desde el último update
2. Se hace un build en GeneXus – es decir, se reorganiza, especifica y compilan los objetos con cambios que se acaban de bajar del GeneXus Server.
3. En el tercer paso se ejecutan las pruebas unitarias.
4. Y si todo va bien se ejecutan finalmente las pruebas de UI.

Luego de todo esto - dependiendo del nivel de cobertura de las pruebas – el build de la aplicación estaría listo para ser liberado a producción, o para que un tester haga manualmente las pruebas de la nueva funcionalidad.

La gracia es que cuando entra el tester, al menos ya sabemos que todas las pruebas de regresión pasan, y que todas las pruebas unitarias de la nueva funcionalidad también, por tanto le resta hacer el test de integración o el ciclo completo de usuario de la nueva funcionalidad a liberar.

Para saber más de Integración continua y la ejecución automática de pruebas ir al wiki.--
https://wiki.genexus.com/commwiki/servlet/wiki?40737,Running+UI+tests+under+CI,



<http://www.genexus.com/gxtest>

GXtest es un gran aliado que nos ayuda a hacer que nuestra aplicación sea confiable, reduciendo los tiempos de prueba gracias a su facilidad para generar y ejecutar pruebas automáticas.

Además es una herramienta fundamental para las automatizaciones necesarias en un proceso de integración continua, que nos permiten construir software de manera ágil y con calidad. Si quiere saber más sobre esta herramienta, acceda a la URL que está en pantalla.



Videos

training.genexus.com

Documentation

wiki.genexus.com

Certifications

training.genexus.com/certifications