

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“AGILIZACIÓN DE TESTING DE SOFTWARE A TRAVÉS DE AUTOMATIZACIÓN POR PARTE DE LOS USUARIOS”

JORGE CAULLÁN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Marcello Visconti
Profesor Correferente: Pedro Godoy Barrera

Octubre - 2019

DEDICATORIA

Éste trabajo de título es dedicado al conocimiento publico y fue desarrollado sin fines de lucro y con la única finalidad de ayudar a mejorar el proceso de testing de empresas lo requieran. El código desarrollado¹ está completamente abierto para todo tipo de publico que desee utilizarlo o modificarlo, por lo que se pide de todo corazón que no se lucre con éste y las posibles mejoras también sean publicas.

¹<https://gitlab.com/jorge.caullan.12/memoria>

AGRADECIMIENTOS

Quisiera agradecer a todos aquellos que me ayudaron durante el desarrollo de la memoria: A mis compañeros de universidad quienes siempre estaban atentos a cualquier duda y el apoyo mutuo siempre existió; A mis familiares quienes me apoyaron con mucho esfuerzo de forma económica y sanitaria, preocupándose siempre de mi salud física y mental; A mis amigos mas cercanos los cuales me apoyaban constantemente en los días en que requería un despejo de la realidad jugando videojuegos con ellos; Y principalmente a mi pareja quien me apoyo tanto de forma emocional en todo momento que necesitara un poco de paz interior, como de forma intelectual al momento de diseñar visualmente las funcionalidades de la herramienta.

RESUMEN

Resumen— En el desarrollo de esta memoria, se crea una herramienta web (utilizando el framework Django) enfocada en crear automatización de pruebas para ser manejada por usuarios que se dedican a hacer Pruebas de Software (SQA). Para esto, se genera una interfaz pensada en este tipo de usuarios, la cual reduzca lo más posible la programación de código, facilitando así la construcción de las pruebas.

Gracias a esto, los beneficios que se obtienen son tanto de agilidad en la construcción y ejecución de pruebas, como de distribución de los resultados, ya que cualquiera en la red local del servidor puede leer, ejecutar y documentar pruebas.

Palabras Clave— Usuario; Automatización; Prueba; Accesibilidad; Documentación

ABSTRACT

Abstract— In the development of this memory, a web tool is created (using the Django framework) focused on creating test automation to be managed by users dedicated to Software Testing (SQA). For this, an interface designed in this type of users is generated, which reduces code programming as much as possible, thus facilitating the construction of the tests.

Thanks to this, the benefits obtained are both agility in the construction and execution of tests, and distribution of the results, since anyone in the local network of the server can read, execute and document tests.

Keywords— Tester; Automation; Test; Accessibility; Documentation

GLOSARIO

Aplicación Web: Tipo de aplicaciones orientadas a utilizarse a través de un servidor al que los usuarios acceden de manera compartida mediante la navegación web.

BackEnd: Rol que se encarga de desarrollar las funcionalidades de una aplicación o herramienta. Aquel desarrollador que se encarga de diseñar y mantener el servidor.

Docker: Es un programa de código abierto que permite que una aplicación Linux y sus dependencias se empaqueten como un contenedor. Similar a lo que hace una máquina virtual, pero de manera más simplificada.

FrontEnd: Rol que se encarga de desarrollar la interfaz de una aplicación o herramienta. Es aquel que trabaja con todo lo que se ve en una aplicación web.

Lógica del negocio: Requerimientos del funcionamiento de un programa desarrollado o en desarrollo. Todas las características que una aplicación debe cumplir para obtener un correcto funcionamiento.

Open Source: (código abierto) Programa de distribución gratuita y desarrollable por una comunidad. Aplicaciones que son gratuitas para cualquier usuario que desee hacer uso de ellas.

Pruebas de caja gris: Pruebas enfocadas a las funcionalidades sin necesidad de hacer uso de las interfaces usuarias (BackEnd).

Pruebas de caja negra: Pruebas enfocadas al correcto funcionamiento de la interfaz usuaria (FrontEnd).

Pruebas de escritura: En el contexto de esta memoria, es una subsección de las pruebas de caja gris. Enfocando las pruebas al correcto funcionamiento del servidor.

Pruebas de lectura: En el contexto de esta memoria, es una subsección de las pruebas de caja gris. Enfocando las pruebas al análisis de la integridad en la base de datos.

Rutas: Consultas acorde al protocolo de transferencia HTTP, en donde se envía un método y dirección específica a un servidor (enfocado en desarrollo web o móvil).

Testing: Realización de pruebas, ya sean automatizadas o manuales, con la finalidad de obtener una mayor calidad en el producto final.

Testing Funcional: Testing enfocado en comprobar que las funcionalidades del software cumplen con su objetivo.

Usuario: En el contexto de esta memoria, usuario se refiere a personas que poseen conocimientos básicos de software y se dedican a hacer pruebas de esto.

Siglas

API: "Application Programming Interface" Interfaz de programación de aplicaciones.

DI: Departamento de Informática.

QA - SQA: "(Software) Quality Assurance" Aseguramiento de calidad.

SQL: "Structured Query Language" Lenguaje de consulta estructurado (Utilizado para hacer consultas en bases de datos relacionales).

TDD: "Test Driven Development" Desarrollo guiado por pruebas de software.

UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Testing Recursivo	2
1.2 Testing Automatizado Enfocado a Desarrolladores	2
1.3 Objetivos	3
1.3.1 Objetivos Generales	3
1.3.2 Objetivos Específicos	3
CAPÍTULO 2: MARCO CONCEPTUAL	5
2.1 Testing en la actualidad	5
2.1.1 Técnicas de diseño de pruebas	5
2.1.2 Cambios en la metodología de testing	7
2.2 Herramientas de automatización en la actualidad	8
2.2.1 Automatización de testing	8
2.2.2 Herramientas para gestión de pruebas y documentación	13
2.3 Tipos de automatización de testing	17
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	21
3.1 Solución para pruebas de caja negra	21
3.1.1 Actualidad	21
3.1.2 Metodología	22
3.1.3 Requerimientos	22
3.1.4 Beneficios	22
3.2 Solución para pruebas de caja gris	23
3.2.1 Actualidad	23
3.2.2 Metodología	23
3.2.3 Requerimientos	23
3.2.4 Beneficios	24
3.3 Reportes para gestión de pruebas	24
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	26
4.1 Documentación de la Herramienta	26
4.1.1 Infraestructura	26

4.1.2	Uso General de pruebas	35
4.1.3	Pruebas de caja gris	39
4.2	Aplicación en empresas	52
4.2.1	Selyt: pruebas de escritura	52
4.2.2	SONDA: pruebas de lectura	57
CAPÍTULO 5: CONCLUSIONES		61
REFERENCIAS BIBLIOGRÁFICAS		63

ÍNDICE DE FIGURAS

1	Interfaz gráfica utilizada por Selenium Web Driver	9
2	Interfaz gráfica utilizada por Jenkins	10
3	Interfaz gráfica utilizada por PhantomJs	11
4	Interfaz gráfica utilizada por UIAutomator	12
5	Interfaz gráfica utilizada por Autolt	13
6	Interfaz gráfica utilizada por TestLink	14
7	Interfaz gráfica utilizada por Postman	15
8	Interfaz gráfica utilizada por SoapUI	16
9	Interfaz gráfica utilizada por Test Studio	17
10	Orden de ejecución de testings	18
11	Infraestructura de requerimientos para la herramienta	26
12	Resultado esperado al ejecutar el comando “docker-compose up -build”	28
13	Página de descarga de <u>Python</u>	29
14	Página de descarga de <u>Postgres</u>	29
15	Ejemplo de archivo “setings.py” modificado para su utilización en windows . .	30
16	Ejemplo de una respuesta tras utilizar el comando “pip install -user -r requirements.txt”	31
17	Ejemplo de una respuesta tras utilizar el comando “python manage.py migrate”	32
18	Ejemplo de acceso a un servidor local mediante su IP	33
19	Modelo de datos utilizado por la base de datos postgres de la herramienta . . .	34
20	Pasos para agregar una nueva sección	35
21	Pasos para agregar una nueva prueba	36
22	Pasos para agregar variables de entorno	37

23 Diagrama de estados de las pruebas	38
24 Posición del botón que permite cambiar el estado de las pruebas	38
25 Campos base requeridos para el correcto funcionamiento de una prueba de escritura	40
26 Ejemplo de una prueba de escritura finalizada y ejecutada	42
27 Ejemplo de una prueba de escritura ejecutada y con problemas	43
28 Ejemplo de una secuencia de pruebas de escritura	44
29 Pasos necesarios para realizar correctamente una prueba de lectura básica . .	45
30 Ejemplo de una prueba de lectura ejecutada	49
31 Ejemplo del resultado generado por una prueba de lectura	49
32 Ejemplo de una secuencia de pruebas de lectura	50
33 Ejemplo del resultado de una secuencia de pruebas de lectura	50
34 Ejemplo de vista para seleccionar reportes	51
35 Ejemplo de como se debe Guardar correctamente un reporte generado	51
36 Pruebas de escritura comprobando el funcionamiento de rutas de manejo de usuarios en la aplicación	53
37 Pruebas de escritura comprobando el funcionamiento de rutas encargadas de la creación de datos en la aplicación	54
38 Ejecución de secuencias de pruebas de escritura y previsualización de documentación	55
39 Documentación generada automáticamente de las pruebas de escritura definidas	56
40 Prueba de lectura utilizada para validar que se encuentren todas las transacciones en un rango de fechas y resaltar las inconsistencias	57
41 Resultado generado de la ejecución de la prueba expuesta en al figura 40 . . .	58
42 Resultados de una ejecución de secuencias de pruebas de lectura	59
43 Reportes generados a partir de las pruebas de lectura utilizadas	60

INTRODUCCIÓN

En la búsqueda de calidad de software, el “testing” es la principal herramienta que se posee, sin embargo, la realización manual de éste es muy costosa tanto en integridad con el proceso de desarrollo como en el esfuerzo humano requerido.

Para resolver ésto, se utiliza el concepto de “automatización de testing” que consiste en programar las pruebas necesarias para que al momento de utilizarlas únicamente se deban ejecutar las pruebas. Sin embargo, las herramientas actuales que proveen este tipo de automatización poseen 3 principales problemas: Se debe ser desarrollador (estar acostumbrado a los lenguajes de programación); Poseen una interfaz poco intuitiva y mal documentada; Son muy costosas de mantener para una empresa.

Luego, generalizando las herramientas, poseen falencias con la integración de las pruebas con los desarrolladores, llegando a tener que ser los mismos desarrolladores los que creen las pruebas.

En esta memoria, se plantea el desarrollo de una herramienta que posee como objetivo permitir que usuarios que no están acostumbrados a programar, sean capaces de crear pruebas automatizadas de manera ágil y ordenada. Para ello, se diseña una interfaz web que sea intuitiva, fácil de utilizar y esté bien documentada.

Gracias a que la herramienta se desarrolla en un servidor web, se logra integrar en tiempo real las pruebas construidas con los desarrolladores (son accesible por todos los involucrados), Además, la herramienta posee la capacidad de ejecutar una secuencia de pruebas y auto-generar documentación a partir de las pruebas construidas, lo cual facilita guardar un registro del desarrollo.

Esta herramienta se desarrolla en Django y se lanzará como código “open source”, de forma que cualquier empresa pueda utilizarla de manera gratuita como base para la automatización de sus pruebas y de ser necesario, pueda adaptarla a sus metodologías. Por ejemplo, manejar los permisos de las pruebas acorde a los usuarios.

Finalmente, se presenta la herramienta desarrollada, la cual se centra únicamente en la automatización de pruebas de caja gris, sin embargo, se prepara como trabajo futuro la implementación de pruebas de caja negra, utilizando reconocimiento de imágenes como base.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

En la actualidad, existe una gran cantidad de empresas enfocadas en el desarrollo de software, en las cuales, tanto en pequeñas como grandes empresas, la realización de Testing es generalmente realizado por un tipo de usuario específico que hace control de calidad de software (SQA).

Este acto genera varios problemas y entre los problemas más graves, existen 2 que terminan desperdiciando bastante tiempo y dinero, los cuales dependen de si el usuario es desarrollador o no.

1. Cuando el usuario no es un desarrollador, existen pruebas que se deberán volver a realizar cada vez que existan cambios en el código, a lo que se llama: **Testing Recursivo**
2. Cuando el usuario es un desarrollador, **el testing recursivo se puede automatizar**, sin embargo, no existe agilidad en esta práctica ya que en automatizar se utiliza tiempo valioso que podría ser aprovechado en el desarrollo de nuevas funcionalidades.

1.1. Testing Recursivo

La realización de testing funcional, se debe enfocar en 2 tareas principales: Realizar pruebas de las nuevas funcionalidades implementadas y Realizar pruebas de las funcionalidades anteriores que pudieron ser afectadas con los cambios.

A esto último se le llama Testing Recursivo, y es uno de los principales problemas de los QA. Debido a que, a medida que el software crece en tamaño y funcionalidades, cada vez se deben realizar más pruebas de recursividad.

Actualmente, el Testing Recursivo en lo posible se evita², ya que es muy costoso de realizar, ocasionando que puedan surgir problemas en el área de Producción y llegar a provocar un problema grave para el/los clientes finales que utilizan el software.

1.2. Testing Automatizado Enfocado a Desarrolladores

Para resolver el problema del Testing Recursivo, existe la solución de automatizar los test, lo cual reduce los costos sustancialmente ya que solo se deben programar una única vez

²En los proyectos en que los problemas pueden llegar a ser muy graves, es imposible evitar el testing recursivo, por lo que la empresa debe asumir los costos asociados

para luego solo ejecutarlos. Sin embargo, en la actualidad esto posee un gran problema, la automatización de testing está enfocada únicamente para desarrolladores, no existen herramientas profesionales para automatizar pruebas sin tener que programar.

Esto genera una gran barrera para los QA, debido a que su trabajo no es desarrollar, sino, testear funcionalidades. Por otro lado, si el trabajo de la automatización es efectivamente realizada por los desarrolladores, la producción del software será bastante más lenta debido a que para cada funcionalidad nueva se debe desarrollar 2 veces: Una para el funcionamiento y otra para su automatización.

Factores a considerar al momento de implementar la automatización por parte de los desarrolladores:

- En términos de costo monetario, un sueldo de Desarrollador generalmente es mayor al de un QA debido a sus conocimientos, por lo que el tiempo utilizado por estos debiera ser aprovechado lo mejor posible.
- Es imposible eliminar el Rol de “QA” para un software, debido a que siempre es necesario que un humano analice los resultados, por ejemplo, en caso de que los requerimientos cambien y la automatización creada se arruine.
- Si un desarrollador se encarga de automatizar pruebas, reduce el trabajo de los QA y aumenta el trabajo propio, volviendo muy poco ágil la implementación de nuevas funcionalidades.

1.3. Objetivos

1.3.1. Objetivos Generales

- Reducir los costos monetarios y de esfuerzo para la automatización y reporte de testing.
- Permitir que un usuario QA sea capaz de crear automatización de software.

1.3.2. Objetivos Específicos

Para los objetivos específicos de esta memoria se deberá desarrollar una herramienta de automatización de software enfocada a usuarios (QA). Por lo que la herramienta debe cumplir con las siguientes características:

- Fácil de utilizar e intuitiva: Debido a que los usuarios finales de la herramienta no se enfocan en desarrollo, su interfaz debe ser fácil de manejar por cualquier usuario con conocimientos básicos de software.
- Permitir automatizar testing de caja gris: Se permitirá que los usuarios puedan automatizar algoritmos que verifiquen la integridad de los datos y el cumplimiento de las reglas del negocio en sus pruebas.
- Permitir ejecutar secuencias de pruebas: Para facilitar la ejecución de pruebas recursivas, se permitirá a los usuarios que creen secuencias de pruebas de forma que se puedan agrupar distintos tipos de pruebas que estén relacionadas entre si.
- Generar reportes de las pruebas ejecutadas por una automatización: permitir que el usuario defina las pruebas que desea reportar y que éstas posean feedback sobre los resultados de las ejecuciones, de forma que ayuden a los desarrolladores a solucionar los problemas.

CAPÍTULO 2

MARCO CONCEPTUAL

En el siguiente capítulo, se explicará el estado del arte existente en la actualidad sobre el desarrollo de testing, el cual posee 3 puntos de vista importantes.

En el primero, se presentará como ha cambiado el testing de software durante estos años, presentado las ideas principales que se han llevado a cabo en principalmente grandes y medianas empresas.

Luego, se mostrarán las principales y más conocidas herramientas de automatización que se están utilizando hoy en día, de las cuales se realizará una breve descripción sobre su utilidad, mencionando pro's y contra's de cada una. Estas serán utilizadas para el desarrollo de la herramienta propuesta, como una guía en la interfaz de usuario o en la utilidad de cada una.

Finalmente, se realizará un análisis sobre los estudios bibliográficos existentes sobre la automatización en general (sin enfocarse en una herramienta específica), de donde se podrá tener una base para el desarrollo de la memoria, además de una guía para la implementación de buenas prácticas en la automatización de testing de software.

2.1. Testing en la actualidad

2.1.1. Técnicas de diseño de pruebas

Tipos de pruebas

En el área de testing de software, las pruebas pueden centrarse en un objetivo particular, por lo que se pueden definir 4 tipos de pruebas [Müller, 2011]:

1. Pruebas funcionales: Verificar que un nuevo caso de prueba del software funcione correctamente
2. Pruebas no funcionales: Comprobar una característica no funcional del producto, tal como fiabilidad o usabilidad
3. Pruebas de estructura: Verificar que las pruebas a realizar cubren todas las áreas del software. Realización de pruebas de integración de sistemas.
4. Pruebas asociadas a cambios: Confirmar que se han solucionado los defectos (pruebas de confirmación) y localizar cambios no intencionados (pruebas de regresión)

Para efectos de la automatización, el tipo de prueba más relevante para automatizar, son las **pruebas de regresión**, ya que, estas pruebas requerirán de más de una única ejecución.

Para detallar, las pruebas de regresión son producto de los cambios en el código, donde cada vez que existe una nueva funcionalidad o se repara alguna funcionalidad que estaba defectuosa, se debe comprobar que el resto de las funcionalidades del software se mantengan funcionando correctamente, ya que los cambios pudieron dañar indirectamente alguna. Por lo que, cada vez que se agrega una funcionalidad, ésta se debe probar junto a todas las funcionalidades anteriores.

Técnicas de caja negra

Las pruebas de caja negra son aquellas en que el realizador de las pruebas no requiere conocer el camino concreto de lo que hace el software, si no, se requiere únicamente de un input y un output valido, el cual deberá ser ejecutado a través de la prueba y se deberá comprobar que el resultado sea similar al esperado. Estas pruebas es común encontrarlas en los casos de uso del software, los que deben ser abarcados cada uno de ellos.

Para la realización de estas pruebas es necesario el análisis de los valores de frontera, los cuales se refiere a que, para cada caso de prueba, se debe probar utilizando los valores máximos y mínimos establecidos para los cuales debiera funcionar el software correctamente.

Otra característica que se debe tener en cuenta para la realización de pruebas de caja negra, es el planteamiento de tablas de decisión, las cuales definen caminos extras a los casos de prueba, es decir, definir los casos específicos de una funcionalidad de forma que se cubran todos los caminos posibles para cubrir las combinaciones de aceptación de la prueba.

Luego, debido a que un software en algún estado específico puede funcionar distinto que en otro estado, es necesario mencionar que deben realizarse todos los tipos de pruebas que impliquen el traspaso de un estado a otro, comprobando que el software no se daña con la realización de algún cambio de estado y que los datos obtenidos sean los esperados.

Técnicas de caja gris

Las pruebas de caja gris son aquellas en que el realizador de las pruebas debe revisar la funcionalidad de los componentes del software ya sea directamente observando la base de datos o la API del servidor, verificando que los cambios realizados sean confiables en la consistencia de los datos resultantes.

Para la realización de este tipo de pruebas, es necesario determinar una serie de pasos con valores específicos para cubrir todo el espacio de pruebas posibles, incluyendo pruebas de decisiones, de los cuales se debe mantener un registro a medida que éstos se van evaluando.

En la actualidad, la mayoría de las pruebas automatizadas, son pruebas de caja gris, ya que

éstas son realizadas por los mismos desarrolladores verificando que los datos utilizados sean consistentes.

2.1.2. Cambios en la metodología de testing

No cabe duda en que el control de pruebas es y siempre ha sido necesario para todo tipo de software, por lo que es muy común que a lo largo de los años, este “evolucione” de manera que cada vez sea más eficiente de ejecutar para los desarrolladores.

Acorde a un estudio realizado el año 2010 [Garousi y Varma., 2010], se ha logrado detectar que a lo largo de los años, la realización de testing ha tenido principalmente los siguientes cambios a como era realizado años atrás:

- La automatización de testings unitarios e integración de sistemas ha tenido un aumento considerable entre 2004 y 2009
- Cada año, más organizaciones están utilizando observaciones y opiniones de expertos para la realización de sus pruebas, mejorando así sus buenas prácticas
- Los mecanismos de generación de casos de pruebas no ha cambiado mucho hasta esa fecha en que se realiza el estudio (2010), por lo que se puede notar una necesidad en esa área
- El costo que está gastando cada empresa en testing tiende a aumentar cada año

Ahora, si se observa la realización de testing de una gran empresa, tal como es **Google** [Whittaker y Carollo., 2012], se puede notar que éstos poseen un gran avance en la automatización de sus pruebas. Proporcionando a los lectores varios consejos sobre las prácticas utilizadas por ellos para tener un testing eficiente.

- Calidad != Pruebas: “La calidad se logra poniendo, desarrollo y pruebas en una licuadora y mezclarlas hasta que no se pueda distinguir una de la otra”
- Roles: Se establecen roles de, ingeniero de software (desarrollador de nuevas funcionalidades), ingeniero de software en pruebas (desarrollador de testing automatizado) y ingeniero de pruebas (realizador de pruebas enfocadas en el usuario en vez del software)
- Estructura Organizacional: El testing se hace derivando cada una de las funcionalidades y generando un rango de completitud de las tareas en lugar de solo señalar que las funcionalidades están buenas o malas

- Gatear, Caminar, Correr: En Google se busca tener el “mínimo producto viable” y se lanza, luego, por cada una de las funcionalidades, por muy pequeñas que sean, si el producto sigue correcto, lo vuelven a lanzar. Es decir, no se hacen grandes lapsos entre un lanzamiento y otro.
- Tipos de Pruebas: Se utilizan 3 tipos de pruebas (automatizadas o no), pequeñas, medianas y grandes. Los cuales pueden ser ejecutados por ingenieros con cualquiera de los roles.

2.2. Herramientas de automatización en la actualidad

Comúnmente, cada empresa utiliza al menos una o dos herramientas para la organización y/o automatización de sus proyectos, sin embargo, cada herramienta posee características positivas y negativas que normalmente se repiten entre ellas, por lo que se categorizarán de la siguiente forma:

2.2.1. Automatización de testing

El siguiente conjunto de herramientas es bastante útil para automatizar testing en general. Sin embargo, cada una de ellas requiere de un aprendizaje de código para realizar la automatización, es decir, están enfocadas para que sean utilizada únicamente por usuarios que tengan conocimientos de desarrolladores.

Selenium

Conjunto de herramientas para automatización web que permite una amplia selección de posibilidades para probar interfaces usuarias. Selenium se divide en las herramientas Selenium WebDriver (Browser nativo para el testing), Selenium Grid (permite realizar tests en muchos servers a la vez), Selenium IDE (extensión de Chrome y Firefox que permite grabar y ejecutar tests) y Selenium Remote Control (permite el control web de otros computadores para realizar pruebas de manera remota).

Fuente: <https://docs.seleniumhq.org/projects/>

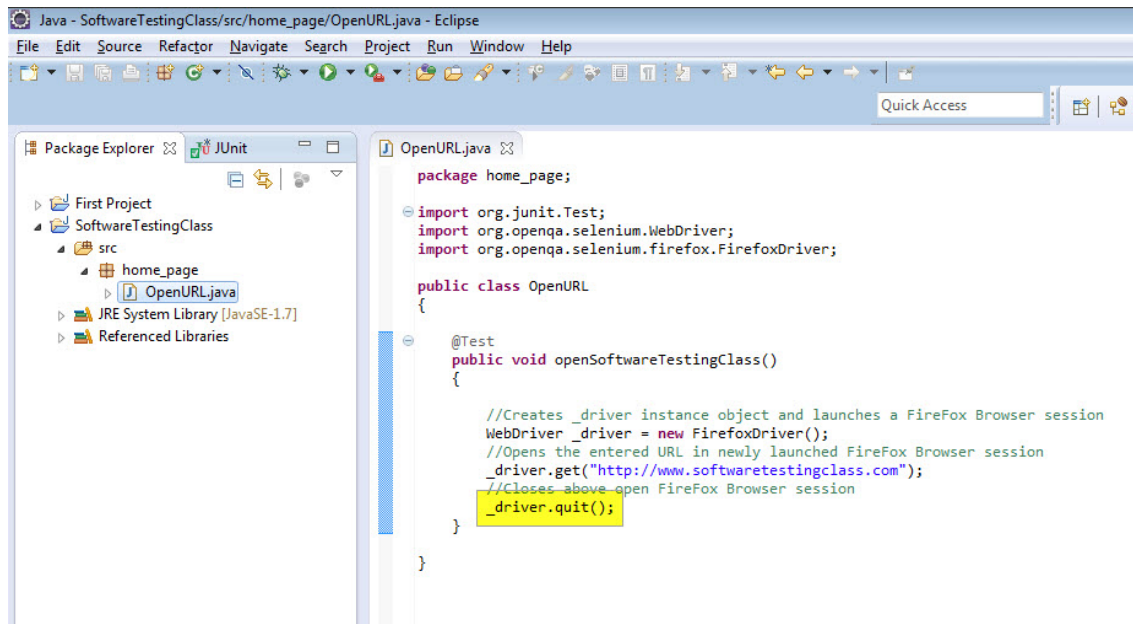


Figura 1: Interfaz gráfica utilizada por Selenium Web Driver

Pro's	Contra's
Open Source (gratis)	Requiere de aprendizaje para cada una de las herramientas
Gran variedad de herramientas para ejecución de testing web	Especializado únicamente para web
Muy buena documentación	Enfocado para desarrolladores

Jenkins

Herramienta para iniciar pruebas continuas y construir integración continua a través de la automatización. Administra los cambios en el código, las pruebas y el ciclo de vida del despliegue

Fuente: <https://jenkins.io>

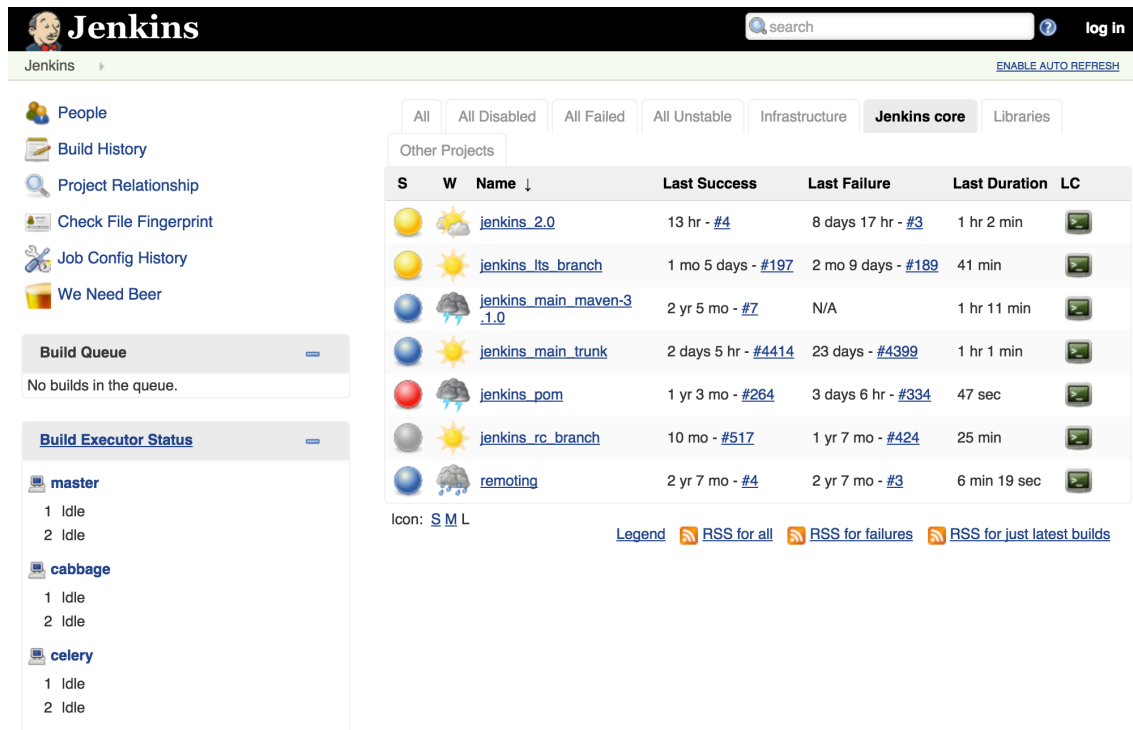


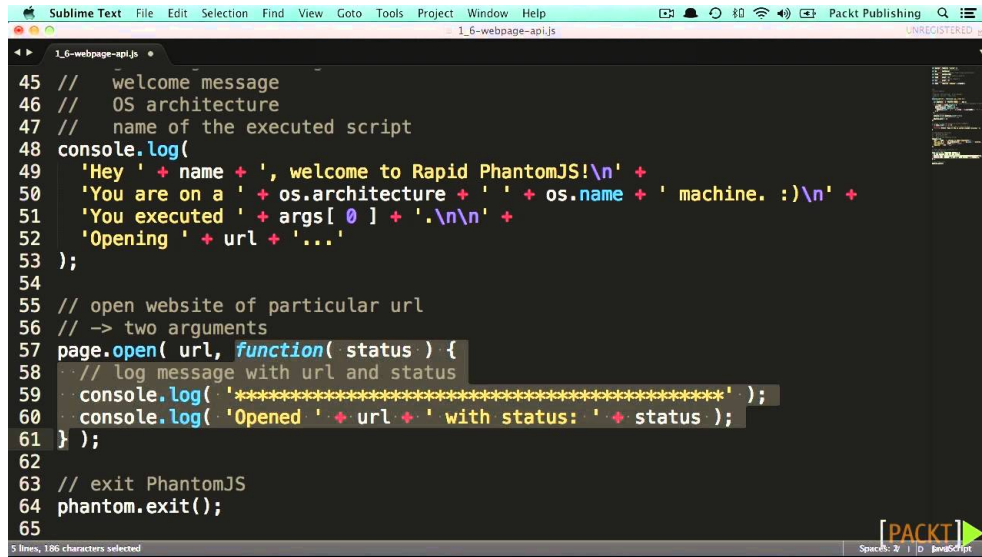
Figura 2: Interfaz gráfica utilizada por Jenkins

Pro's	Contra's
Open Source (gratis)	Requiere de aprendizaje hecho por parte de informáticos (código)
Administración de cambios en código	Enfocado para desarrolladores
Gran comunidad de usuarios	
Muy buena interfaz	

PhantomJS

Se utiliza para automatizar las interacciones de una página web con fines de prueba. Imita y manipula una página web para llevar a cabo la automatización de pruebas.

Fuente: <http://phantomjs.org>



```
45 // welcome message
46 // OS architecture
47 // name of the executed script
48 console.log(
49   'Hey ' + name + ', welcome to Rapid PhantomJS!\n' +
50   'You are on a ' + os.architecture + ' ' + os.name + ' machine. :)\n' +
51   'You executed ' + args[ 0 ] + '\n\n' +
52   'Opening ' + url + '...'
53 );
54
55 // open website of particular url
56 // -> two arguments
57 page.open( url, function( status ) {
58   // log message with url and status
59   console.log( '*****' );
60   console.log( 'Opened ' + url + ' with status: ' + status );
61 } );
62
63 // exit PhantomJS
64 phantom.exit();
65
```

Figura 3: Interfaz gráfica utilizada por PhantomJs

Pro's	Contra's
Open Source (gratis)	Poca documentación
Fácil de utilizar (por desarrolladores web)	Comunidad pequeña
	Requiere de saber JavaScript (enfocado para desarrolladores)

UIAutomator

Marco para pruebas de interfaz de usuario funcional para aplicaciones de Android, permitiendo crear casos de pruebas ejecutables en dispositivos de distintas resoluciones.

Fuente: <https://developer.android.com/training/testing/ui-automator>

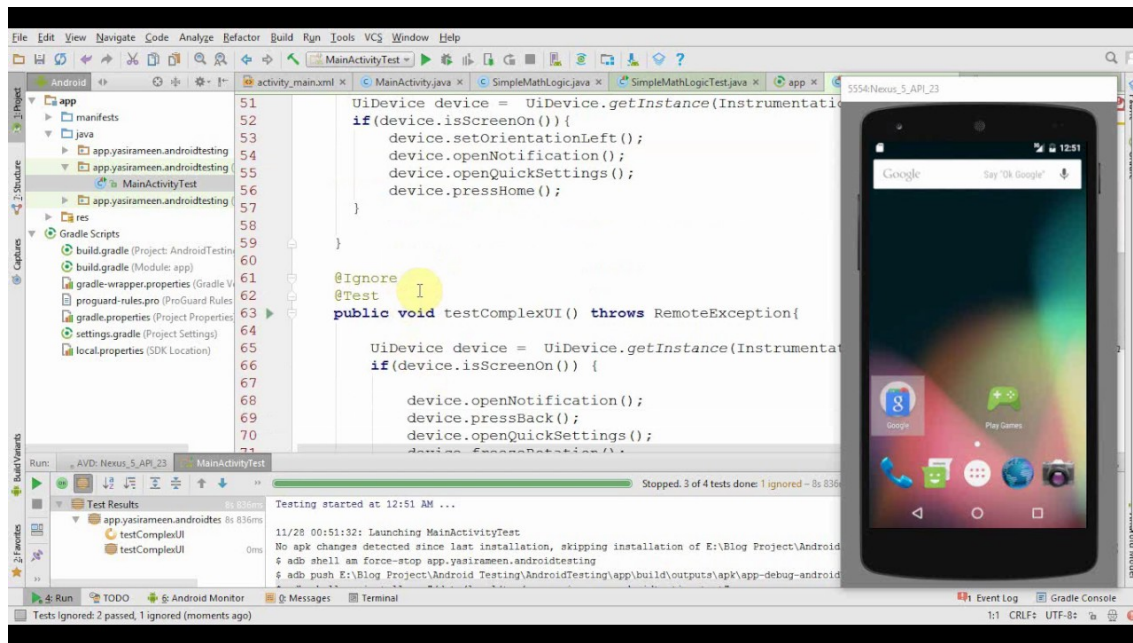


Figura 4: Interfaz gráfica utilizada por UIAutomator

Pro's	Contra's
Open Source (gratis)	Requiere tiempo de aprendizaje
Muy útil para desarrolladores android	Enfocada a desarrolladores
Documentación muy completa	

AutoIt

Aplicación que permite la automatización de secuencia de teclas y manipulación del mouse mediante una secuencia de comandos.

Fuente: <https://www.autoitscript.com/site/>

```

79  GUICtrlSetState(-1, $GUI_DISABLE)
80  GUICtrlCreateGroup("", -99, -99, 1, 1) ;close group
81
82  GUICtrlCreateGroup("Timeout", 230, 450, 200, 50)
83  $idTimeout = GUICtrlCreateInput("", 240, 470, 100, 20, $ES_NUMBER)
84  GUICtrlSetTip(-1, "Optional Timeout in seconds. After the timeout has elapsed the message box will be automatically closed.")
85  GUICtrlCreateGroup("", -99, -99, 1, 1) ;close group
86
87  $idBTNPREVIEW = GUICtrlCreateButton("&Preview", 10, 510, 100)
88  GUICtrlSetTip(-1, "Show the MessageBox")
89  $idBTNCOPY = GUICtrlCreateButton("&Copy", 120, 510, 100)
90  GUICtrlSetTip(-1, "Copy the generated AutoIt code to the Clipboard")
91  $idBTNEXTIT = GUICtrlCreateButton("&Exit", 230, 510, 100)
92  GUICtrlSetTip(-1, "Quit the program")
93
94  $idButton = $idOptOK
95
96  GUISetState() ; will display an empty dialog box
97
98  ; Run the GUI until the dialog is closed
99  While 1
100     $iMSG = GUIGetMsg()
101     Select
102     Case $iMSG = $GUI_EVENT_CLOSE Or $iMSG = $idBTNEXTIT
103         Exit
104     Case $iMSG = $idOptOK
105         $idButton = $idOptOK
106         GUICtrlSetState($idOptFirst, $GUI_CHECKED)
107         GUICtrlSetState($idOptFirst, $GUI_ENABLE)
108         GUICtrlSetState($g_idOptSecond, $GUI_DISABLE)
109         GUICtrlSetState($g_idOptThird, $GUI_DISABLE)
110     Case $iMSG = $g_idOptOkCancel
111         $idButton = $g_idOptOkCancel
112         GUICtrlSetState($idOptFirst, $GUI_CHECKED)
113         GUICtrlSetState($idOptFirst, $GUI_ENABLE)
114         GUICtrlSetState($g_idOptSecond, $GUI_ENABLE)
115     Case $iMSG = $g_idOptCancel
116         $idButton = $g_idOptCancel
117         GUICtrlSetState($idOptFirst, $GUI_CHECKED)
118         GUICtrlSetState($idOptFirst, $GUI_ENABLE)
119         GUICtrlSetState($g_idOptSecond, $GUI_ENABLE)
120     EndSelect
121 EndWhile
    
```

Figura 5: Interfaz gráfica utilizada por Autolt

Pro's	Contra's
Open Source (gratis)	La automatización solo será útil en un computador con una resolución específica
Permite crear secuencias de comandos para manipular aplicaciones externas	Para aplicaciones que no usen pantalla completa no siempre funcionará
	Utilizable solo en windows
	Enfocado para desarrolladores

2.2.2. Herramientas para gestión de pruebas y documentación

Las siguientes herramientas están diseñadas para gestionar y documentar proyectos, a partir de usuarios sin un completo conocimiento de programación. Sin embargo, estas herramientas en general poseen 2 grandes problemas: Poseen muy poca documentación y/o su versión pagada es demasiado costosa.

TestLink

Herramienta de administración de pruebas que responde preguntas como: ¿Para qué requisitos todavía necesitamos escribir o actualizar casos de prueba? ¿Qué pruebas quieres que ejecute para esta versión? ¿Qué casos de prueba están fallando actualmente y cuáles son los errores?

Fuente: <http://testlink.org/>

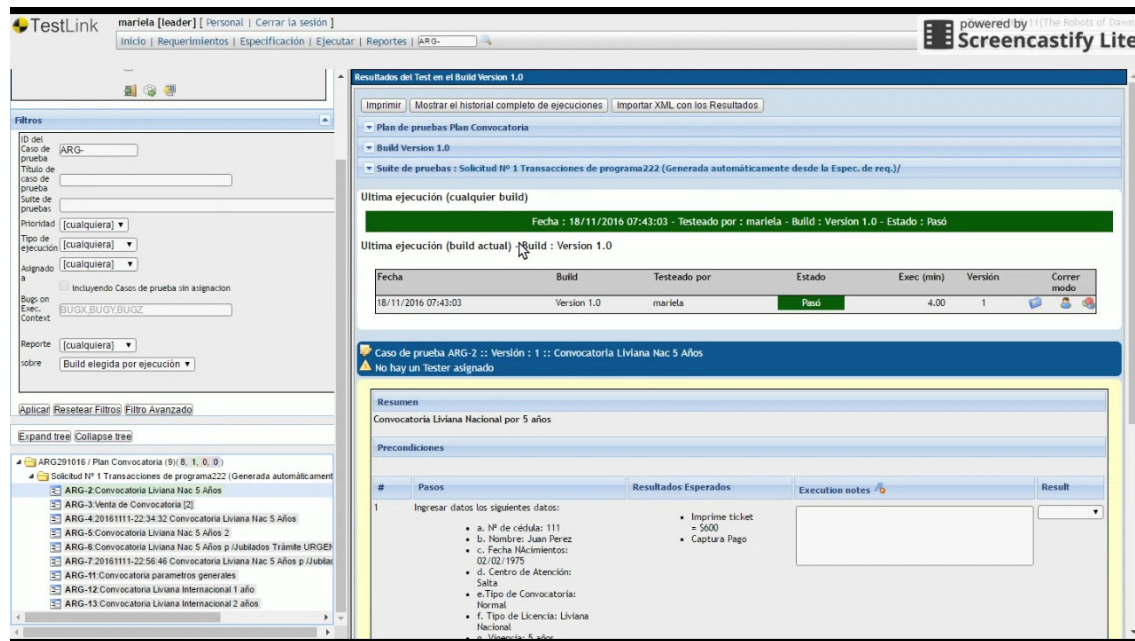


Figura 6: Interfaz gráfica utilizada por TestLink

Pro's	Contra's
Open Source (gratis)	Poca documentación
Permite la organización y gestión de tests y usuarios	Comunidad pequeña
Interfaz intuitiva y fácil de usar	

Postman

Herramienta para probar APIs, permite comprobar el historial de solicitudes HTTP enviadas, previsualizar imágenes, realizar pruebas de producción o organizar configuraciones.

Fuente: www.getpostman.com

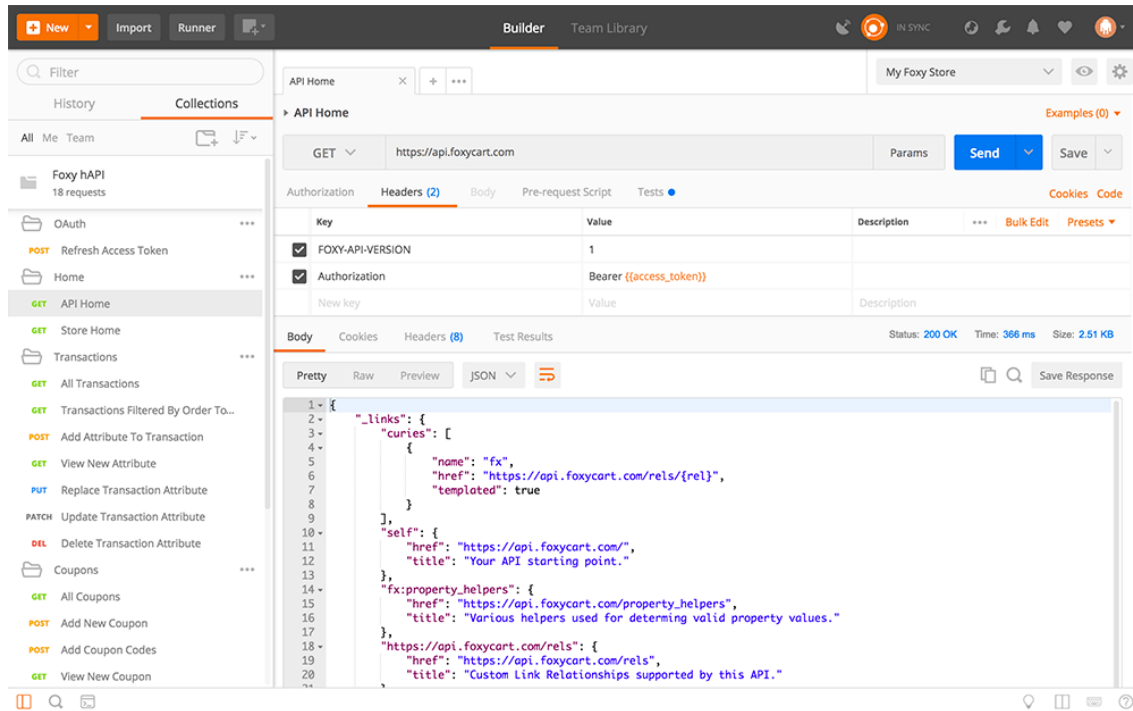


Figura 7: Interfaz gráfica utilizada por Postman

Pro's	Contra's
Posee versión Open Source (gratis)	Versión premium (pagada) para uso compartido y documentación
Permite probar APIs	Limita testear APIs manualmente ruta por ruta
Interfaz intuitiva y fácil de usar	

SoapUI

Herramienta de automatización de APIs que provee de estadísticas, seguridad, simulador de servicios y reportes.

Fuente: <https://www.soapui.org/>

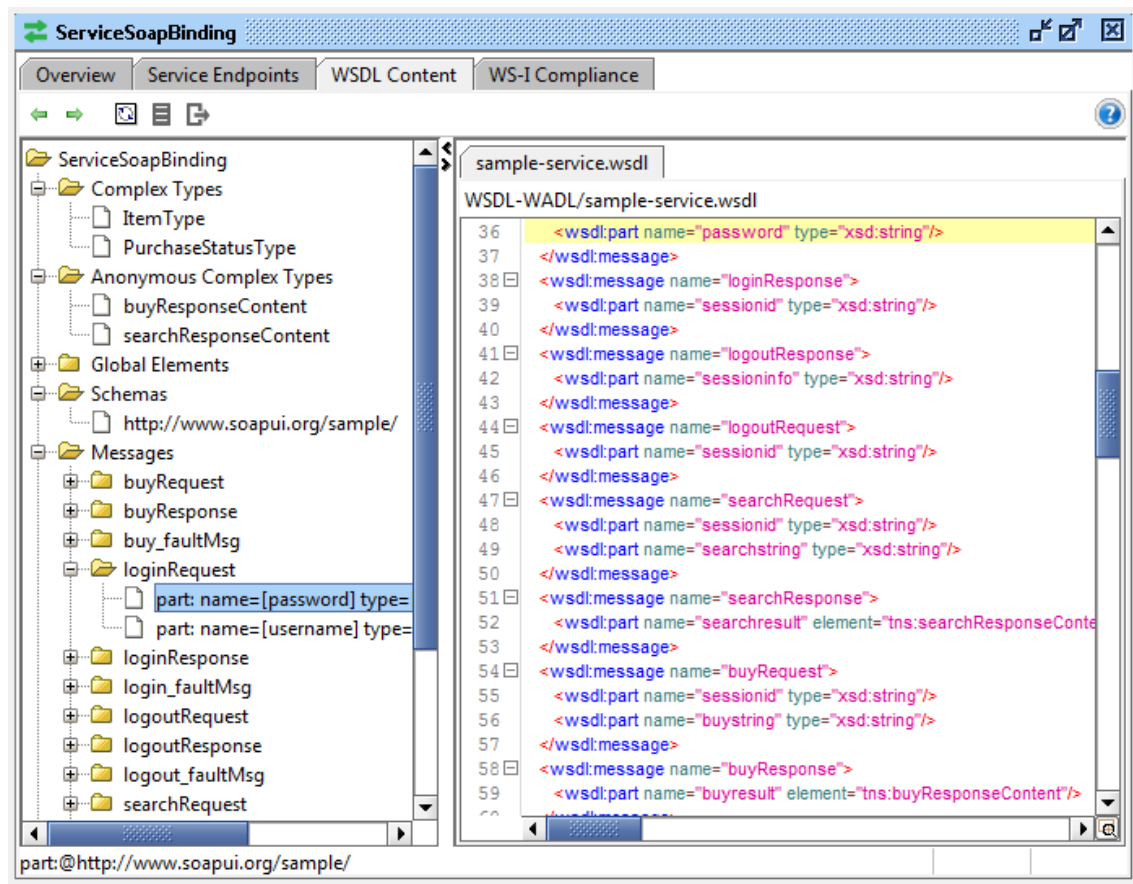


Figura 8: Interfaz gráfica utilizada por SoapUI

Pro's	Contra's
Versión Open Source (gratis)	Versión premium (pagada) con la mayor parte de recursos
Gran comunidad de usuarios	No permite sincronización con más usuarios
Provee reportes al usuario	Enfocado para desarrolladores

Test Studio

Herramienta que permite hacer gestión de pruebas, automatizar pruebas funcionales y realizar pruebas de carga y rendimiento.

Fuente: <https://www.telerik.com/teststudio>

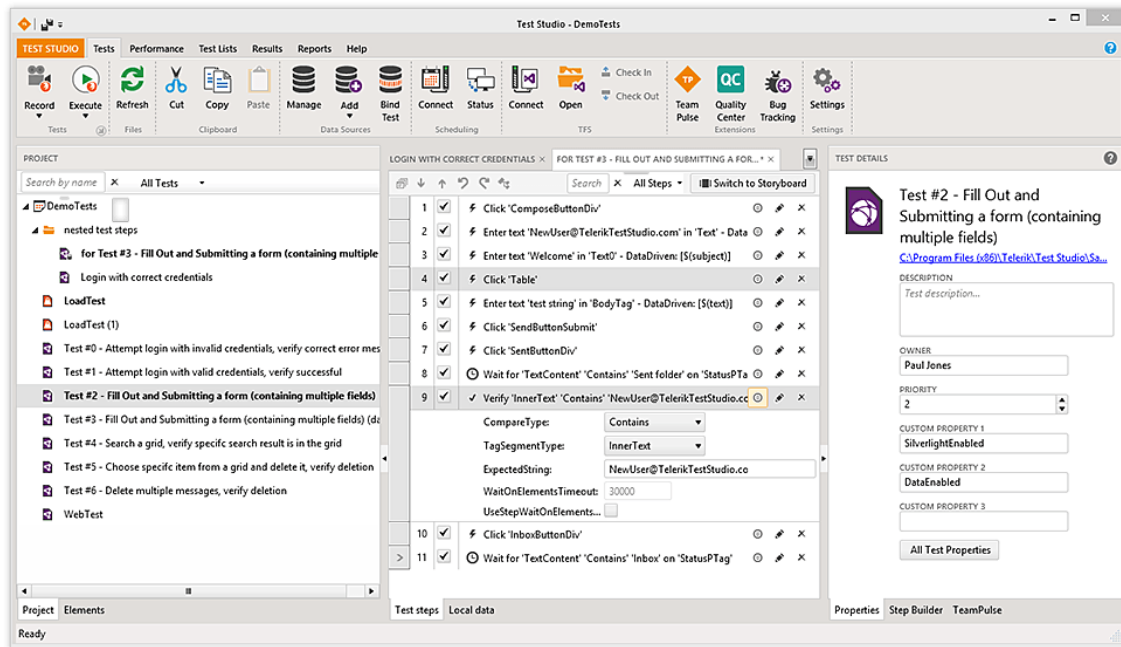


Figura 9: Interfaz gráfica utilizada por Test Studio

Pro's	Contra's
Contiene todo lo necesario para automatizar testing	Solo existe una versión premium (pagada) que posee un coste excesivo.
Permite trabajar con servicios web, de escritorio y móvil.	Requiere de un tiempo de aprendizaje por usuario
Muy bien documentado	

2.3. Tipos de automatización de testing

Gestión para la automatización de testing

Para asegurar la calidad de los productos, acorde a los estudios realizados a varias empresas estadounidenses [Kasurinen y Smolander., 2010], actualmente la automatización se puede dividir en 5 actividades que serán necesarias para una buena gestión de testing:

1. Automatización de aplicación: Planeación de áreas de un software donde podrá ser aplicada exitosamente la automatización, es decir, aquellas áreas en que no sea estrictamente necesario mantener constantemente supervisión humana.
2. Roles en el proceso de desarrollo: Asignación de roles a los desarrolladores de la aplicación, tal como “developer”, “debugger”, “tester”, etc.

3. Estrategia de automatización: Definición de compromiso con la automatización de testing, ya sea desde una automatización para una única tarea, hasta la automatización de cada uno de los casos de prueba de la aplicación.
4. Desarrollo de automatización: Realización de la automatización, utilizando los 3 pasos definidos anteriormente
5. Problemas con automatizaciones: Realización de testing a las automatizaciones de pruebas desarrolladas, de manera que se pueda estar seguros del correcto funcionamiento de estas.

Automatización de pruebas orientadas a software

Para aplicar un testing completo a un software, existen varias necesidades que se deben satisfacer, las cuales se pueden derivar en las siguientes[Vahid y Elberzhager, 2017]:

1. Test-case design: La realización del diseño de los casos de prueba.
2. Test scripting: Documentación de los casos de prueba.
3. Test execution: Ejecución de los test documentados anteriormente.
4. Test evaluation: Evaluación sobre los test ejecutados si se completan correctamente o fallan en su ejecución.
5. Test-result reporting: Reportes de ayuda sobre el posible problema cuando los test fallan.

Los cuales se pueden visualizar más fácilmente a través de la figura 10 [Vahid y Elberzhager, 2017]:

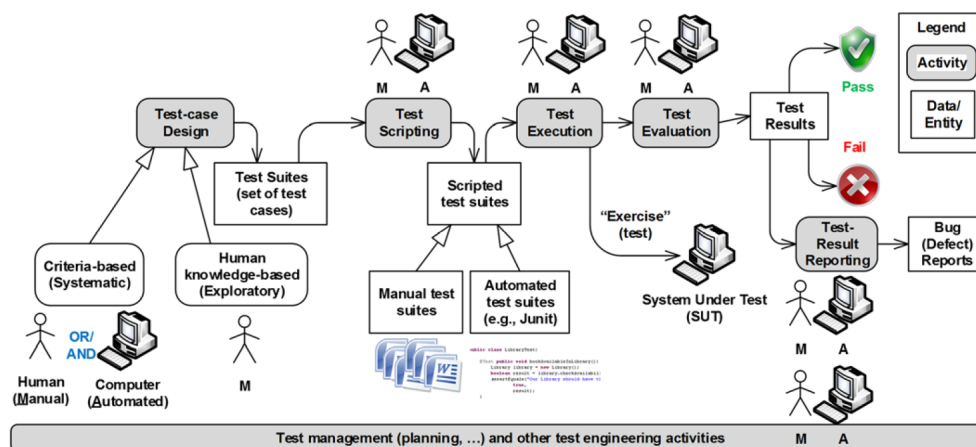


Figura 10: Orden de ejecución de testings

Automatización en Test-case Design

Para el diseño de casos de pruebas, se debe designar una lista de requerimientos necesarios para cumplir con cada una de las funcionalidades que posee el software. Para ésto, se requiere de análisis basados en una de las siguientes opciones:

1. Designación de criterios estandarizados sobre el tipo de software
2. Realización de un análisis exploratorio sobre las funcionalidades del software

De los cuales, el único que puede ser automatizado es el número 1, debido a que el número 2 es dependiente de la experiencia de los testers para el análisis del software y no requiere del uso de ninguna herramienta externa.

Debido a esto, se puede implementar una automatización parcial de un testing utilizando la primera de las opciones y así reduciendo el análisis exploratorio a únicamente un análisis de casos de borde.

Automatización en Test Scripting

Este tipo de test puede ser parcialmente automatizado proporcionando una herramienta que permita a un tester documentar y guardar automáticamente los test que se desean ejecutar a futuro. Luego, estos test deben poder ser ejecutados automáticamente por una herramienta las veces que sean necesarias, sin la intervención de un tester. Actualmente existen gran variedad de herramientas que permiten realizar éstas automatizaciones, sin embargo, cada una está orientada a un ambiente específico, ya sea para desarrollo de aplicaciones móviles, web u otras.

Automatización en Test Execution

Tal como se mencionó anteriormente, éste tipo de automatización es aquel que posee una mayor cantidad de estudio, por lo que también se puede notar una gran cantidad de herramientas que permiten automatizar testing, sin embargo, no todas están directamente ligadas al ítem anterior (test scripting). Éstas consisten en la ejecución de cada uno de los test documentados, las cuales pueden ser realizadas de forma completamente manual, parcialmente automatizada o completamente automatizada. Lo cual está directamente ligada al uso de herramientas que ejecuten scripts para la realización de una prueba.

Automatización en Test Evaluation

Luego de la ejecución de un test, se debe poder evaluar el resultado de éste (si pasa la prueba o falla), lo cual se puede realizar de dos maneras: Observando el comportamiento de la ejecución del testing (manualmente), o Incorporando la evaluación en la ejecución de la prueba, es decir, cada posible error se debe considerar en la ejecución, de manera que se pueda entregar un resultado automático al finalizar la prueba.

Automatización en Test Result Reporting

Finalmente, una fase no menos importante que las anteriores es el reportar a los desarrolladores los defectos encontrados en los test que han fallado, ya que sin esto, solo se entregarán fallos a los desarrolladores, sin definirles las razones de estos y dificultando enormemente la velocidad en la solución de los problemas.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

Como se describe en capítulos anteriores, existe una gran variedad de problemas al momento de automatizar software. Haciendo énfasis en:

- Se debe ser desarrollador para poder automatizar cualquier tipo de software.
- Gran parte de las herramientas actuales poseen una muy mala documentación, a pesar de no ser intuitivas y requieren de una gran curva de aprendizaje para su correcto uso.
- Los costos para utilizar herramientas de gestión pueden ser excesivos por cada usuario que los utilice (no son “open source”).
- Gran parte de las herramientas se enfocan en un único tipo de aplicaciones, ya sea Web, Escritorio o Móvil.

Para solucionar estos problemas, en esta memoria se propone diseñar una herramienta que permita a los QA automatizar pruebas sin la necesidad de utilizar lenguajes de programación específicos para cada proyecto.

Ésta herramienta estará diseñada para la automatización de 2 tipos de pruebas de software, de forma que se pueda cubrir tanto automatizaciones de BackEnd como de FrontEnd, enfocando el desarrollo especialmente para las pruebas de BackEnd. Además, permitirá generar reportes de la herramienta que se automatice, lo que se podrá utilizar tanto para generar documentación de los casos de prueba, como para reportar errores en la aplicación a los desarrolladores.

3.1. Solución para pruebas de caja negra

3.1.1. Actualidad

Hoy en día, existen muy pocas herramientas orientadas a automatizar directamente el FrontEnd de una aplicación, y las herramientas existentes poseen 2 típicos problemas: están orientadas a un tipo de aplicación específica (Móvil, Web o de Escritorio) y/o la documentación que poseen es precaria, a pesar de utilizar una interfaz compleja y que requiere ser estudiada para poder automatizar con ella. Sin embargo, en otros casos la automatización puede funcionar correctamente grabando el movimiento del mouse durante un test y reproduciéndolo cuando se desea hacer la prueba, sin embargo, esto tiene el problema de que

debe cumplir con estrictas condiciones, ya sea, utilizar una pantalla de resolución específica o la necesidad de utilizar secuencia de comandos de teclado, además de la necesidad de realizar la prueba a un tiempo acorde a la velocidad del computador y posibilidad de fallos.

3.1.2. Metodología

En el caso de la herramienta que será diseñada en esta memoria, se utilizará Python como lenguaje principal de desarrollo, debido a que éste posee librerías que pueden ser muy útiles. Para automatizar pruebas de caja negra, se utilizará reconocimiento de imagen de manera que en todo momento el computador pueda encontrar la posición de cada Botón o Campo que necesite, de esta forma, no se requerirá una pantalla de tamaño específico y la prueba podrá auto gestionar su velocidad según las respuestas del computador

3.1.3. Requerimientos

Sin embargo, el uso de reconocimiento de imagen posee ciertas desventajas, pues obliga a que las imágenes de la aplicación sean estáticas y únicas. Es decir, los botones no deben poseer transparencia ni efectos visuales ya que, al ser muy cambiantes, dificulta demasiado el reconocimiento. Para complementar esto, la herramienta desarrollada permitirá que se automaticen secuencias de teclas, de manera que si existe un botón que puede estar preseleccionado, solo bastará con presionar una tecla (por ejemplo "Enter"). Esto a su vez, será muy necesario dado que hará posible llenar formularios para las aplicaciones que se automaticen.

3.1.4. Beneficios

Uno de los principales beneficios de esta herramienta será su facilidad de uso, ya que no requerirá de aprendizaje de un lenguaje de programación para diseñar cada algoritmo, sino, se utilizará una interfaz de usuario intuitiva para cualquier tipo de usuario con conocimientos básicos de informática. Gracias a esto, no se requerirá que los desarrolladores automaticen, pues podrán hacerlo los QA.

Luego, entre los beneficios de usar esta metodología en las pruebas de caja negra, lo primero que se puede notar es que las automatizaciones realizadas por esta herramienta no se limitarán a un único tipo de plataforma, sino que podrá ser utilizada para automatizar aplicaciones de tipo Web, Móviles (con la ayuda emuladores de móviles para pc) y de Escritorio.

Generalmente, en las empresas de desarrollo de software, antes de desarrollar una interfaz, esta debe ser diseñada, por lo que antes de esto es muy usual que se tenga un diseño (imágenes) con todos los componentes que la aplicación utilizará. Gracias a esto, un segundo

beneficio será que, los QA podrán diseñar los algoritmos de testing en paralelo con el desarrollo, definiendo a la vez las reglas del negocio (casos de uso) que podrán utilizarse como documentación, obteniendo así una agilidad mucho mayor a la tradicional (TDD).

3.2. Solución para pruebas de caja gris

Para complementar las pruebas de caja negra, para usuarios con mayores conocimientos de informática (comprensión de uso de Bases de Datos y funcionamiento de servidores API REST), se permitirá que automaticen pruebas de caja gris, es decir, que permitan leer y ejecutar el BackEnd de una aplicación con el objetivo de permitir evaluar pruebas de consistencia y/o cumplimiento de las reglas de negocio.

3.2.1. Actualidad

En la actualidad, existe una enorme variedad de herramientas que permiten automatizar testing, sin embargo, todas están condicionadas a plataformas específicas o a lenguajes de programación específicos, por lo que podría decirse que para cada aplicación podría utilizarse una herramienta distinta y que sea compatible para la tecnología que use la aplicación.

3.2.2. Metodología

Para manejar las pruebas de caja gris en esta herramienta, al igual que para la automatización de caja negra, se pretende utilizar Python debido a las librerías que posee para poder conectarse a las bases de datos externas y ejecutar “requests” a servidores externos. Gracias a esto, se podrá permitir que el usuario que desee automatizar una aplicación, se pueda conectar a la base de datos (de testing) de ésta. En el caso de la primera versión de la herramienta, se considerará únicamente bases de datos relacionales (SQL), debido a que son las más utilizadas de la actualidad.

3.2.3. Requerimientos

Ya que esta herramienta estará enfocada para la automatización de software, será necesario que los QA que ejecuten las pruebas de caja gris, tengan conocimientos básicos de informática, principalmente sobre la base de datos que los desarrolladores están utilizando, y sobre el funcionamiento de los servidores API REST. De manera que puedan realizar consultas para la verificación de los datos y a la vez puedan definir (documentar) las rutas necesarias para manejar los datos.

3.2.4. Beneficios

A pesar de que las automatizaciones requerirán de conocimientos de informática, solo será necesario comprender los comandos más básicos del lenguaje Python, conocer SQL y la lógica del negocio. Para esto, se espera que la curva de aprendizaje sea muy baja debido a que la herramienta tendrá una interfaz usuaria que apoyará al usuario a definir las pruebas que desea automatizar. De esta forma, se logra generalizar el alcance de la herramienta para poder ser utilizada por cualquier aplicación, sin importar el lenguaje de programación en el que esté desarrollado.

Las pruebas de caja blanca constarán de 2 posibles automatizaciones:

- **Lectura** (enfocado a que las automatizaciones comprueben que la lógica del negocio funciona correctamente): La realización de automatizaciones de lectura, permitirá a los QA establecer las reglas del negocio para cada aplicación, de modo que ésta podrá ser utilizada para comprobar que los datos son consistentes y validos luego de una ejecución manual (o automatizada). A su vez, los algoritmos automatizados se podrán utilizar para documentar los casos de uso de la aplicación.
- **Escritura** (para permitir automatizar pruebas de funcionalidades específicas de la aplicación, sin pasar por FrontEnd): La realización de automatizaciones de escritura, permitirá a los QA ejecutar directamente rutas para comprobar los funcionamientos de cada funcionalidad (completa o parcial) de la aplicación. Siendo esto de gran utilidad para los desarrolladores BackEnd, dado que serán capaces de notar casi inmediatamente cuando una ruta deja de funcionar.

Además, para ambos tipos de pruebas la herramienta permitirá a los usuarios ejecutar una secuencia de pruebas en lugar de solo una, de modo que si se desea probar una funcionalidad completa, se pueda ejecutar sin mayores dificultades. Por ejemplo, en caso de las pruebas de escritura, si la aplicación se tratase sobre un blog, se debiera poder leer, crear, editar y eliminar post's, lo cual podría hacerse en una única secuencia de pruebas automatizada.

3.3. Reportes para gestión de pruebas

Como se ha mencionado anteriormente, para dar un valor extra a la herramienta, esta podrá permitir la generación de reportes para dar apoyo tanto a los desarrolladores como a los QA evitando que deban documentar por si mismos las cosas que ya han echo en la herramienta.

La generación de reportes, se dividirá en 2 tipos:

- Generación de documentación sobre casos de pruebas y funcionalidades.

- Generación de resultados de las pruebas automatizadas, entregando pruebas de los resultados obtenidos de manera que no sea necesario que un QA le informe a los desarrolladores.

De las cuales se permitirá a los usuarios elegir las opciones que desean documentar. Además, debido a que algunas aplicaciones pueden ser muy grandes, la cantidad de casos de uso (automatizaciones) puede nublar los resultados, por lo que se permitirá a los QA que asignen secciones y los casos de prueba que desean documentar.

La documentación de las pruebas de escritura, en los resultados se incluirá la respuesta obtenida por la consulta además de una tabla con cada uno de los resultados esperados y el estado sobre si la respuesta obtenida satisface a la esperada.

Luego, para los resultados las pruebas de lectura de la herramienta, cada prueba documentada poseerá links a reportes generados en formato “xlsx” cuyos nombres especificarán la fecha y hora en que se ejecutó la prueba, y en el contenido de los archivos se mostrarán los datos extraídos directamente desde la base de datos de la prueba, los cuales serán analizados acorde a las reglas del negocio definidas en las validaciones, y marcando los datos con colores Verde, Amarillo o Rojo acorde a si los datos cumplieron con lo esperado o no, además, cuando una celda se marque con amarillo o rojo, se generará un comentario (dentro de la casilla) especificando el problema.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

4.1. Documentación de la Herramienta

4.1.1. Infraestructura

La infraestructura de la herramienta, se subdivide en 3 ítems:

- Instalación y Uso
- Estado de las pruebas
- Arquitectura de la base de datos

Los cuales poseen gran relevancia debido a que la herramienta aspira a ser un código “Open Source”. Es decir, en cualquier organización se debe poder utilizar sin requerir ayuda externa.

Instalación y Uso

Los requerimientos de la aplicación se pueden notar a través de la Figura [?]

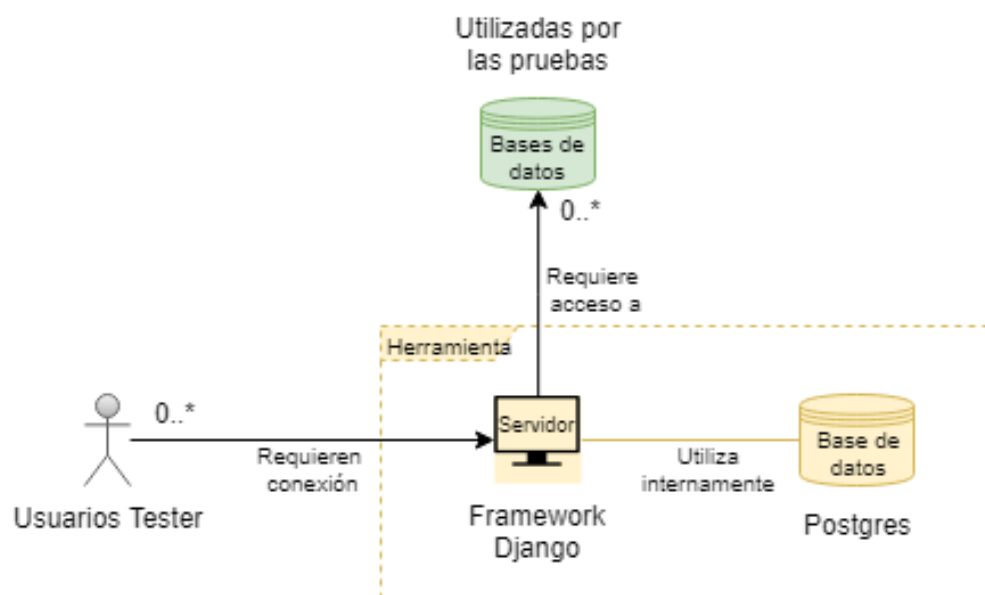


Figura 11: Infraestructura de requerimientos para la herramienta

Donde se puede notar que la herramienta utiliza el Framework “Django” (basado en el lenguaje de programación “Python”), el cual a su vez, utiliza una base de datos “Postgres” para almacenar cada una de las pruebas creadas en la herramienta.

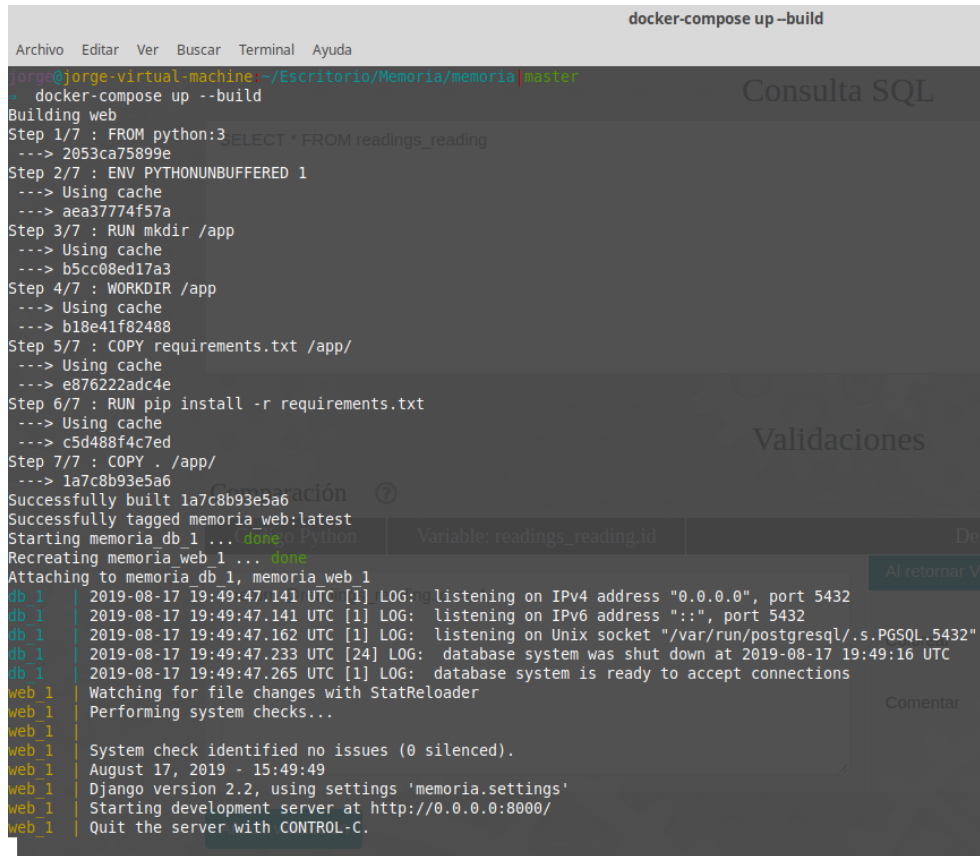
Además, es necesario mencionar que para que las pruebas de lectura creadas funcionen correctamente, el computador en que se instale la herramienta requerirá tener acceso a cualquier base de datos (para pruebas de lectura) o servidor (para pruebas de escritura) que quiera utilizarse en una prueba.

Para la instalación de la herramienta, en primer lugar se debe descargar el repositorio que la contiene³, luego de esto, existen 2 formas de utilizarla acorde al sistema operativo del computador:

- Linux (utilizando Docker): Esta forma de instalación es la manera más simple de instalar la herramienta, y es definida específicamente para Linux debido a que Docker es capaz de funcionar en Windows, pero no correctamente. Para hacer uso de este método, se debe poseer instalado Docker en el computador.
 1. Para instalar todos los requerimientos en un contenedor Docker y ejecutarla una primera vez, lo único que habrá que hacer será utilizar el comando. **“docker-compose up -build”** desde la consola del sistema (estando en la carpeta principal del repositorio clonado)⁴.

³Link al repositorio: <https://gitlab.com/jorge.caullan.12/memoria>

⁴Comandos básicos para utilizar una terminal en linux: <https://hipertextual.com/archivo/2014/04/comandos-basicos-terminal/>



```

docker-compose up --build

jorge@jorge-virtual-machine:~/Escritorio/Memoria/memoria$ docker-compose up --build
Building web
Step 1/7 : FROM python:3.7
--> 2053ca75899e
Step 2/7 : ENV PYTHONUNBUFFERED 1
--> Using cache
--> aea37774f57a
Step 3/7 : RUN mkdir /app
--> Using cache
--> b5cc08ed17a3
Step 4/7 : WORKDIR /app
--> Using cache
--> b18e41f82488
Step 5/7 : COPY requirements.txt /app/
--> Using cache
--> e876222adc4e
Step 6/7 : RUN pip install -r requirements.txt
--> Using cache
--> c5d488f4c7ed
Step 7/7 : COPY . /app/
--> 1a7c8b93e5a6
Successfully built 1a7c8b93e5a6
Successfully tagged memoria_web:latest
Starting memoria_db_1 ... done
Recreating memoria_web_1 ... done
Attaching to memoria_db_1, memoria_web_1
db_1 | 2019-08-17 19:49:47.141 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db_1 | 2019-08-17 19:49:47.141 UTC [1] LOG: listening on IPv6 address "::", port 5432
db_1 | 2019-08-17 19:49:47.162 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db_1 | 2019-08-17 19:49:47.233 UTC [24] LOG: database system was shut down at 2019-08-17 19:49:16 UTC
db_1 | 2019-08-17 19:49:47.265 UTC [1] LOG: database system is ready to accept connections
web_1 | Watching for file changes with StatReloader
web_1 | Performing system checks...
web_1 | System check identified no issues (0 silenced).
web_1 | August 17, 2019 - 15:49:49
web_1 | Django version 2.2, using settings 'memoria.settings'
web_1 | Starting development server at http://0.0.0.0:8000/
web_1 | Quit the server with CONTROL-C.

```

Figura 12: Resultado esperado al ejecutar el comando “docker-compose up --build”

2. Luego, sin detener el servidor ejecutado en el paso 1, en una consola paralela a la actual se debe utilizar el comando “docker-compose exec web python manage.py migrate”, lo cual configurará automáticamente la base de datos que el servidor requiere.
 - En adelante, para ejecutar el servidor de la herramienta, se deberá utilizar el comando “docker-compose up” (teniendo la consola posicionada en el directorio principal de la herramienta).
- Windows: Para la instalación de la herramienta en un computador con el sistema operativo Windows y sin la utilización de Docker, se debe seguir la siguiente serie de pasos.
 1. Asegurar que el computador posea Python y Postgres instalados, de lo contrario, se deben instalar desde la página oficial de cada uno.

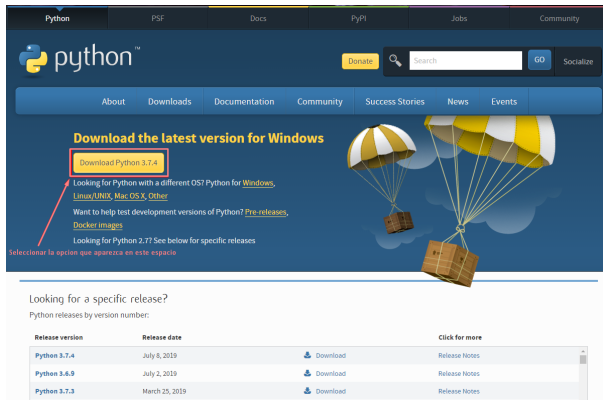


Figura 13: Página de descarga de Python

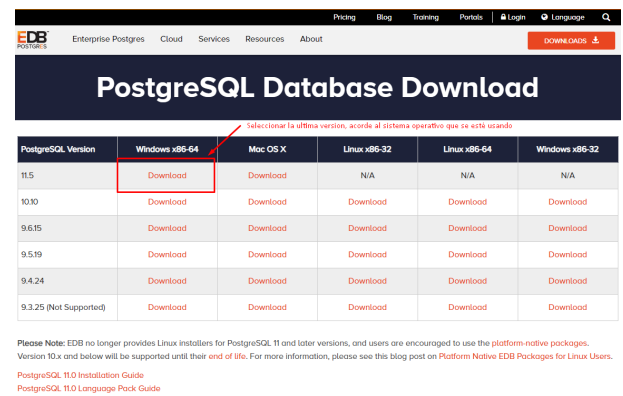
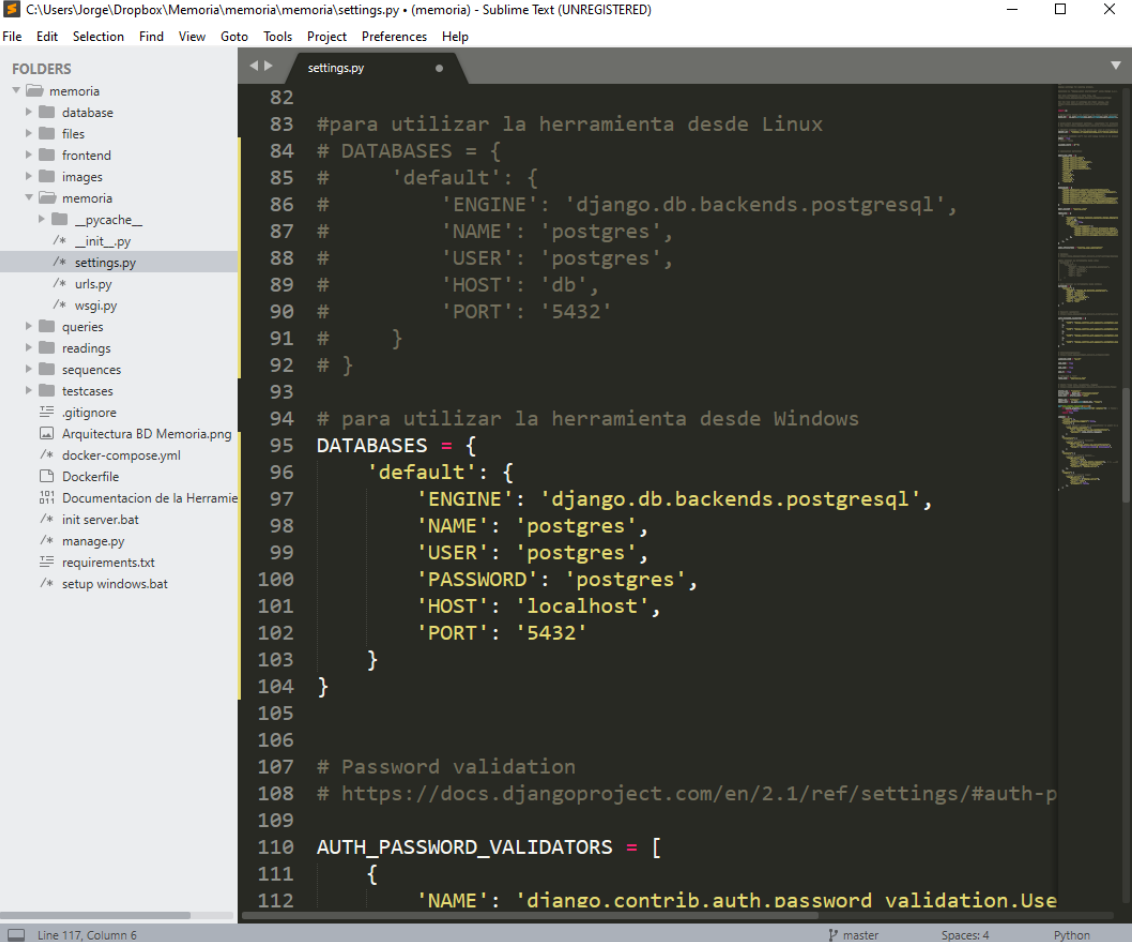


Figura 14: Página de descarga de Postgres

- Utilizar un editor de texto para modificar el archivo “/memoria/settings.py”, modificando la variable “DATABASES”⁵, donde lo más probable será que se deba modificar el valor de ‘HOST’:‘db’ por ‘HOST’:‘localhost’ y agregar la variable ‘PASSWORD’ utilizando la contraseña asignada a postgres en el paso anterior.

⁵Configuración Database: <https://docs.djangoproject.com/en/1.9/ref/settings/#databases>



```
82
83 #para utilizar la herramienta desde Linux
84 # DATABASES = {
85 #     'default': {
86 #         'ENGINE': 'django.db.backends.postgresql',
87 #         'NAME': 'postgres',
88 #         'USER': 'postgres',
89 #         'HOST': 'db',
90 #         'PORT': '5432'
91 #     }
92 # }
93
94 # para utilizar la herramienta desde Windows
95 DATABASES = {
96     'default': {
97         'ENGINE': 'django.db.backends.postgresql',
98         'NAME': 'postgres',
99         'USER': 'postgres',
100         'PASSWORD': 'postgres',
101         'HOST': 'localhost',
102         'PORT': '5432'
103     }
104 }
105
106
107 # Password validation
108 # https://docs.djangoproject.com/en/2.1/ref/settings/#auth-p
109
110 AUTH_PASSWORD_VALIDATORS = [
111     {
112         'NAME': 'django.contrib.auth.password validation.Use
```

Figura 15: Ejemplo de archivo “settings.py” modificado para su utilización en windows

Nota: los pasos 3 y 4 pueden ser evitados ejecutando el archivo “setup windows.bat” en el directorio principal de la herramienta.

3. Instalar las librerías necesarias para la herramienta a través de Python, utilizando el comando “pip install -user -r requirements.txt”, utilizando la consola de Windows⁶ y ubicándola en el directorio principal del repositorio descargado.

⁶Tutorial de uso básico de la consola de Windows: <https://computerhoy.com/paso-a-paso/software/comandos-esenciales-consola-windows-cmd-que-debes-conocer-77943>

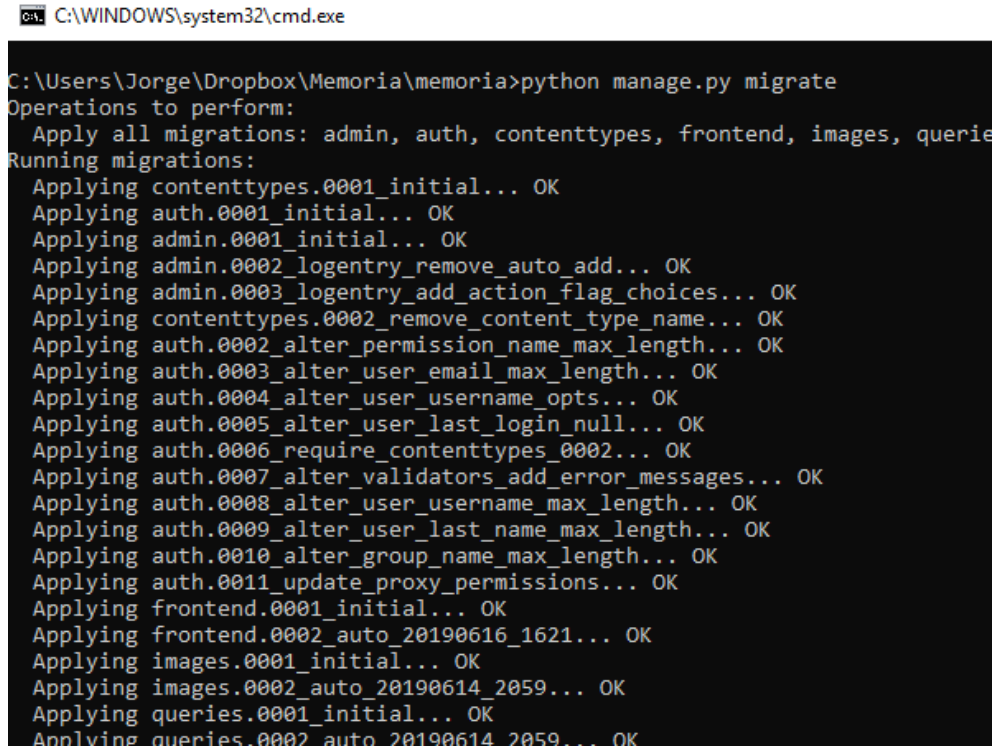
CA: Símbolo del sistema

```
C:\Users\Jorge\Dropbox\Memoria\memoria>pip install --user -r requirements.txt
Collecting certifi==2019.6.16 (from -r requirements.txt (line 1))
  Using cached https://files.pythonhosted.org/packages/69/1b/b853c7a9d4f6a6d00749e94eb
Collecting chardet==3.0.4 (from -r requirements.txt (line 2))
  Using cached https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb
Collecting cx-Oracle==7.1.3 (from -r requirements.txt (line 3))
  Using cached https://files.pythonhosted.org/packages/0b/3f/6b106ba91ead909be1b4cd8fc
Collecting Django==2.2 (from -r requirements.txt (line 4))
  Using cached https://files.pythonhosted.org/packages/54/85/0bef63668fb170888c1a2970e
Collecting et-xmlfile==1.0.1 (from -r requirements.txt (line 5))
  Using cached https://files.pythonhosted.org/packages/22/28/a99c42aea746e18382ad9fb36
Collecting idna==2.8 (from -r requirements.txt (line 6))
  Using cached https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd
Collecting jdcalf==1.4.1 (from -r requirements.txt (line 7))
  Using cached https://files.pythonhosted.org/packages/f0/da/572cbc0bc582390480bbd7c4e
Collecting libsass==0.18.0 (from -r requirements.txt (line 8))
  Using cached https://files.pythonhosted.org/packages/92/f3/a0c4a5db3531ccb79621de62b
Collecting openpyxl==2.6.2 (from -r requirements.txt (line 9))
  Using cached https://files.pythonhosted.org/packages/ba/06/b899c8867518df19e242d8cbc
Collecting pycopg2==2.8.2 (from -r requirements.txt (line 10))
  Using cached https://files.pythonhosted.org/packages/00/f3/afd8570a32712953c83e6ba2d
Collecting pytz==2019.1 (from -r requirements.txt (line 11))
  Using cached https://files.pythonhosted.org/packages/3d/73/fe30c2daaaa0713420d0382b1
Collecting requests==2.22.0 (from -r requirements.txt (line 12))
  Using cached https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00a
Requirement already satisfied: six==1.12.0 in c:\users\jorge\appdata\roaming\python\py
Collecting sqlparse==0.3.0 (from -r requirements.txt (line 14))
  Using cached https://files.pythonhosted.org/packages/ef/53/900f7d2a54557c6a37886585a
Collecting urllib3==1.25.3 (from -r requirements.txt (line 15))
  Using cached https://files.pythonhosted.org/packages/a6/69/347f523e7431ae633d63841be7
```

Figura 16: Ejemplo de una respuesta tras utilizar el comando “pip install --user -r requirements.txt”

4. Para la primera vez que se ejecuta la herramienta, desde la consola de windows se deberá ejecutar el comando “python manage.py migrate” para que se creen las bases de datos que utiliza la herramienta ⁷.

⁷si el paso 4 falla, significa que el archivo “settings.py” no quedó bien configurado y la herramienta no posee acceso a la base de datos



```
C:\WINDOWS\system32\cmd.exe

C:\Users\Jorge\Dropbox\Memoria\memoria>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, frontend, images, queries
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying frontend.0001_initial... OK
  Applying frontend.0002_auto_20190616_1621... OK
  Applying images.0001_initial... OK
  Applying images.0002_auto_20190614_2059... OK
  Applying queries.0001_initial... OK
  Applying queries.0002_auto_20190614_2059... OK
```

Figura 17: Ejemplo de una respuesta tras utilizar el comando “python manage.py migrate”

- En adelante, para ejecutar el servidor de la herramienta, ejecutando el archivo “init server.bat”.

Finalmente, para utilizar la herramienta. Cualquier computador que esté conectado en la misma red local del computador en que fue ejecutado el servidor podrá utilizar la herramienta ingresando desde el navegador a la dirección: “[*ipdel servidor*]:8000”

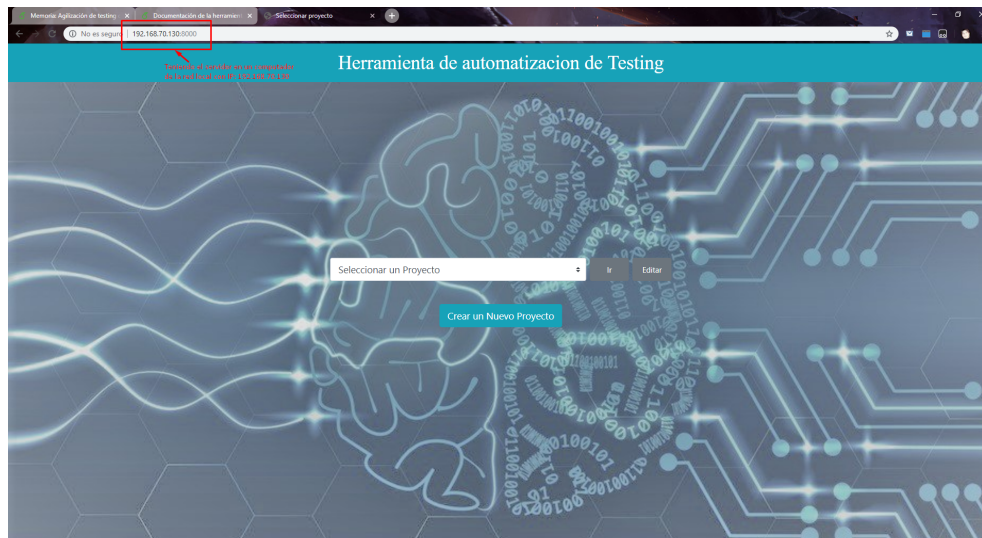


Figura 18: Ejemplo de acceso a un servidor local mediante su IP

Arquitectura en la Base de datos

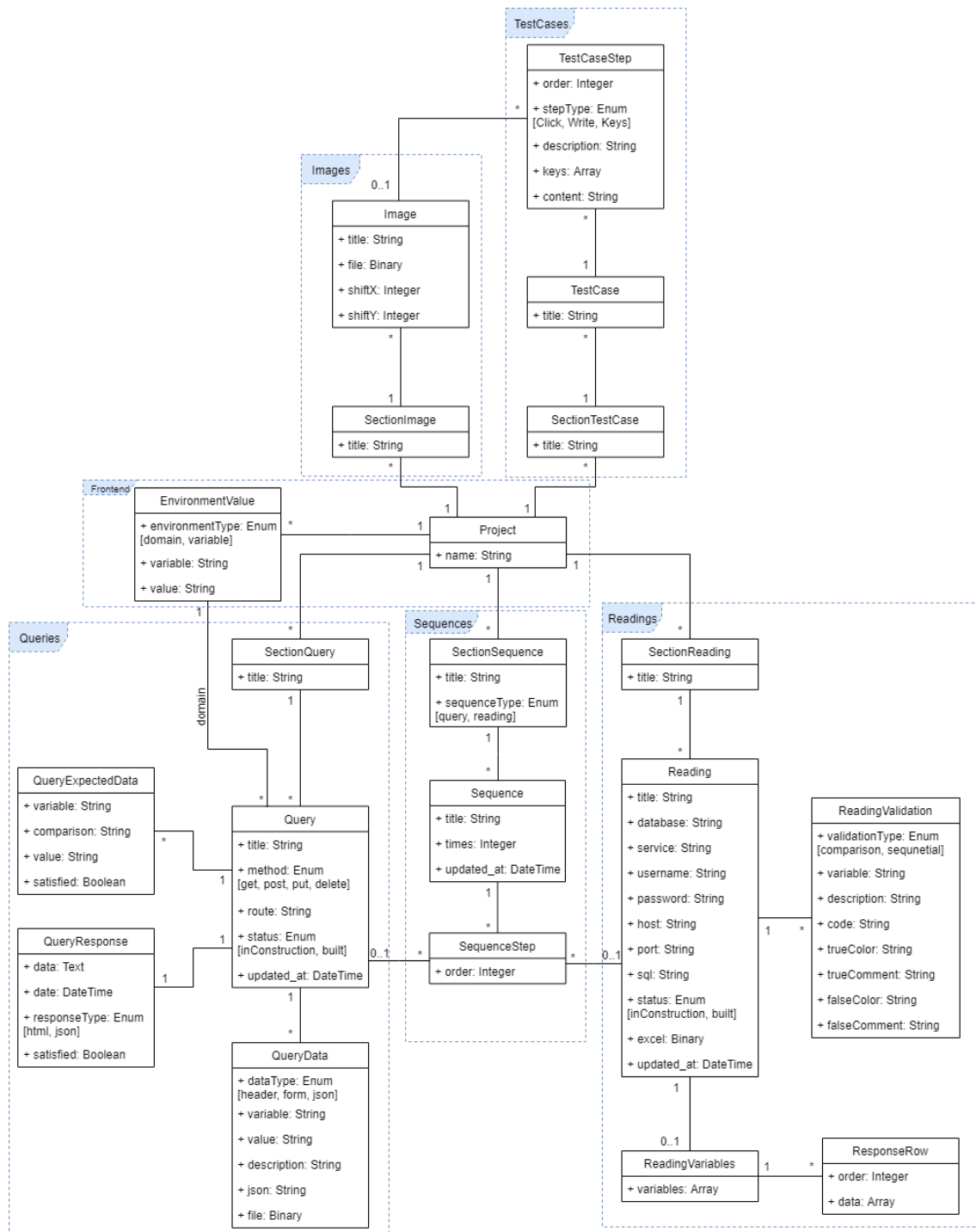


Figura 19: Modelo de datos utilizado por la base de datos postgres de la herramienta

4.1.2. Uso General de pruebas

Tanto las pruebas de lectura como de escritura, poseen características en común:

- La creación de las pruebas poseen una interfaz idéntica.
- Pueden utilizar variables de entorno en todos sus campos de texto.
- Poseen un “Estado actual de la prueba”.

Agregar Sección

Las secciones en cada uno de los tipos de pruebas son utilizados únicamente para la organización de estas pruebas, estas se utilizan para “encapsular” tipos de pruebas, diferentes productos de una aplicación o lo que el usuario desee. Para crear una nueva sección, se debe hacer clic en el símbolo “+” en centro de la barra lateral izquierda e ingresar el título de la sección. Al crearla, ésta estará disponible para seleccionar cuando se desee agregar una prueba.

Si una sección fue creada pero no posee ninguna prueba, ésta no aparecerá en la lista de pruebas hasta que se le asigne alguna.

Las secciones son únicas para cada tipo de prueba, por lo que al crear una, ésta estará disponible solo en el tipo de prueba en el cual fue creada.

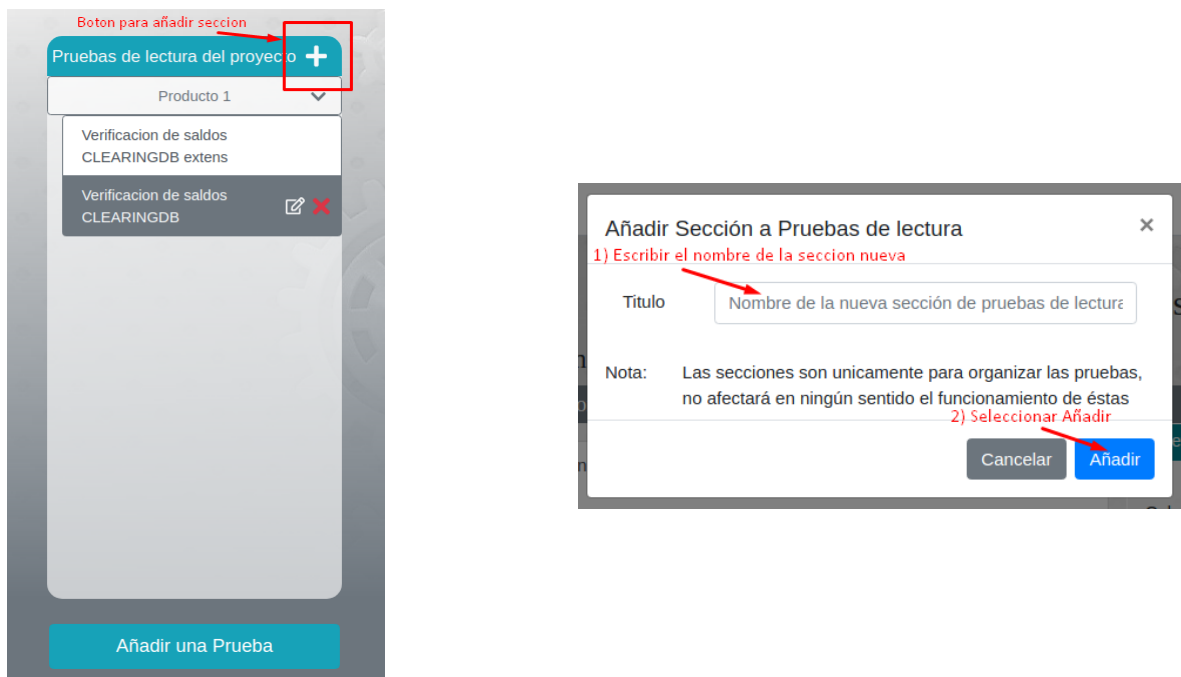


Figura 20: Pasos para agregar una nueva sección

Agregar Prueba

Para la creación de una prueba, se debe hacer clic en “Añadir una Prueba” ubicada en la zona inferior de la barra lateral izquierda. A la cual se le deberá asignar un Título y seleccionar una sección del tipo de prueba actual. Para ello, se requiere que exista al menos una sección creada en el tipo de prueba actual.

Luego de que la prueba esté creada, ésta tendrá todos sus datos vacíos, por lo que será necesario llenarlos y guardarlos para poder ejecutarla.

Cada prueba creada puede editar su título o ser borrada desde la barra lateral izquierda, además de poder ser arrastrada a una sección ya existente.



Figura 21: Pasos para agregar una nueva prueba

Tipos de variables de entorno del proyecto

Las variables de entorno, son variables que pueden ser utilizadas en todo el proyecto, y estas se subdividen en: Dominios y Variables.

Los Dominios son variables que se deben utilizar únicamente en las “Pruebas de Escritura de Caja Gris”, y se utilizan para definir las rutas principales de un proyecto. Para direccionar a la API de un servidor de: desarrollo, testing, beta, localhost, etc. Por ejemplo, para asignar un dominio se deberá asignar un nombre al dominio (como “localhost”) y en “Valor”, se deberá definir la ruta base como “localhost:8000” (se puede incluir “http://” al principio de la ruta

base si se desea por comodidad).

Las Variables son utilizadas para facilitar el manejo de texto que puede ser muy cambiante, ya sean tokens, identificadores u otro tipo de dato. Estas pueden ser utilizadas en cualquier input de texto de la aplicación, haciendo referencia a la variable encerrándola entre llaves y corchetes a la vez.

Por ejemplo, si en las variables de entorno existe una variable “token” con valor “123456789”, en cualquier input de texto de la aplicación se podrá referenciar de la siguiente forma: “token_actual = {[token]}”, lo que al momento de ejecutarse, se leerá como “token_actual = 123456789”.



Figura 22: Pasos para agregar variables de entorno

Estados de las pruebas

En la herramienta, cada prueba creada puede poseer varios estados en términos de uso de la prueba, los cuales se definen en la siguiente figura:

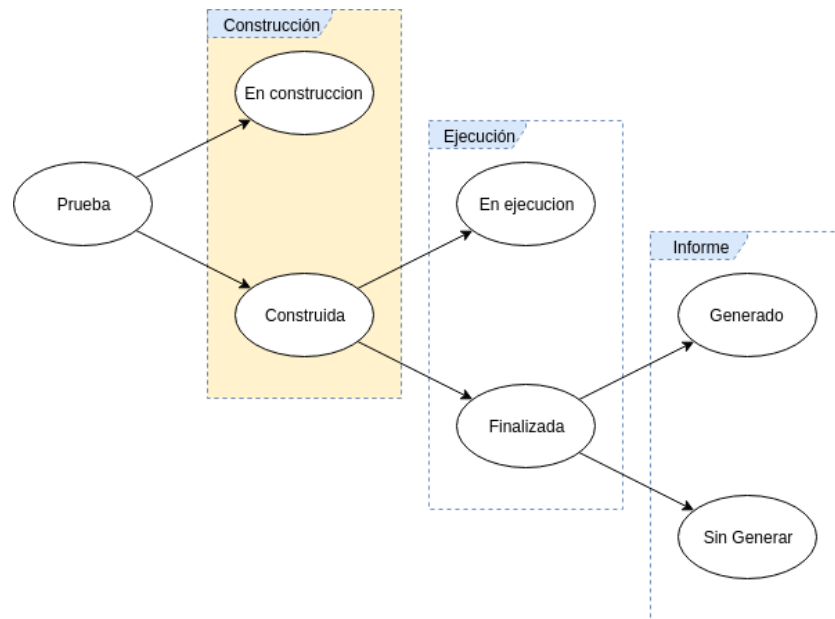


Figura 23: Diagrama de estados de las pruebas

Donde se puede notar que existen 3 tipos de estados que atraviesa una prueba (Construcción, Ejecución e Informe). De los cuales se debe destacar Construcción, debido a que es el área en que mayores beneficios ofrece la herramienta, permitiendo construir ágilmente las pruebas y marcar cada una como “En construcción” o “Construida” para la comprensión de los usuarios (testers o desarrolladores) externos a la construcción de la prueba.



Figura 24: Posición del botón que permite cambiar el estado de las pruebas

4.1.3. Pruebas de caja gris

Las pruebas de caja gris están enfocadas en automatizar pruebas de software sin utilizar una interfaz usuaria. Por ejemplo, acceder a la base de datos para comprobar la lógica del negocio o acceder directamente al “Back End” de la aplicación y se subdividen en pruebas de escritura y pruebas de lectura.

Para hacer uso de estas se requiere de un conocimiento básico en el desarrollo de software, tal como reconocer una base de datos junto a sus credenciales y comprender funcionamiento de una API.

Cada tipo de prueba posee además la capacidad de “Ejecutar Secuencias”, lo cual ayuda a ejecutar en orden una serie de pruebas de manera que se facilite obtener un resultado que requiera de varias bases de datos o varios servidores.

Pruebas de escritura

Las pruebas de escritura se enfocan en hacer consultas directamente al “Back End” de un servidor, de forma que se prueben y documenten las rutas de una API, con la ventaja de además ser capaz de ejecutar secuencias de consultas con la finalidad de probar ágilmente un conjunto de consultas ya construidas. Para esto, las consultas utilizan un protocolo HTTP⁸ para el envío de datos.

Cada prueba de escritura debe como mínimo poseer 3 datos para ser ejecutada: método, dominio y ruta.

- El método (HTTP verbs) es aquel que indica el tipo de acción que se utilizará, en el que actualmente puede elegirse entre GET, POST, PUT y DELETE.
- El dominio es aquel que define la “url” principal del servidor que se desea testear. Éstas deben definirse en las variables de entorno del proyecto, asignándole un nombre de dominio y un valor (tal como se especifica en la sección 4.1.2).
- La ruta es el sitio específico al cual se realizará la consulta, el cual deberá estar definido por el servidor del dominio.

⁸Protocolo HTTP para envío de datos: https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto

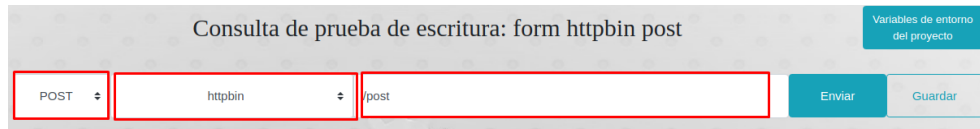


Figura 25: Campos base requeridos para el correcto funcionamiento de una prueba de escritura

El envío de datos en cada consulta se debe hacer mediante el contenedor que esta debajo de la barra con los datos mencionados anteriormente. Estos se componen de 3 tipos: Headers, Body as form-data y Body as json.

- Los Headers pueden ser requeridos por los servidores generalmente para identificar las credenciales de los usuarios u otras razones, por lo que son considerados como data extra en la consulta.
- Los atributos “Body as” se utilizan para enviar data internamente, sin embargo, son exclusivos el uno del otro. Cuando existe data en “Body as form data” al momento de enviar la Query, la posible data que exista en “Body as json” será ignorada:
 - A través de “Body as json”, se puede enviar la data en formato json. Esto es especialmente útil cuando los datos enviados son simples o el servidor requiere que la data sea json.
 - Utilizando “Body as form data”, se extiende la posibilidad de datos, permitiendo incluso enviar archivos. Además, se permite a cada variable agregar una descripción para la comprensión de cada una.

Luego de tener los datos principales de una consulta existe la posibilidad de ingresar “Respuestas Esperadas” para ella. Esto está pensado para respuestas de tipo API con formato JSON⁹, de donde se analizará la respuesta obtenida al enviar la consulta, con cada una de las respuestas esperadas.

Para la realización de una respuesta esperada, al ingresar una variable se puede utilizar “->” para referenciar una variable de un nivel más profundo en el JSON entregado. Por ejemplo, si en la respuesta JSON se entregarse:

```
{
  "some_dict": {
    "a": 5
  }
}
```

⁹Definición del formato JSON: <https://www.json.org/json-es.html>

En las respuestas esperadas se podría ingresar (sin incluir las comillas):

- Variable: "some_dict ->a"
- Tipo de comparación: "Mayor a"
- Valor: "0"

Lo que al ejecutar, comprobaría internamente que "5 > 0", por lo que la prueba fue satisfactoria.

Luego de guardar y ejecutar una prueba de escritura, las respuestas esperadas aparecerán coloreadas de verde o rojo, lo que significa que esa prueba cumplió o no con la comparación. Además, si todas las pruebas se cumplen, junto al título "Respuesta Esperada" aparecerá entre paréntesis "Correcta". En caso contrario, dirá "Con problemas".

Finalmente, luego de una ejecución correcta, aparecerá la Respuesta Obtenida al final de la página, la cual idealmente debe ser una respuesta JSON para que puedan comprobarse las respuestas esperadas, sin embargo, también se pueden obtener respuestas del tipo HTML, lo cual hará aparecer un botón para previsualizar la interfaz de la respuesta obtenida.

Consulta de prueba de escritura: httpbin post

Variables de entorno del proyecto

POST

httpbin

/post

Enviar

Guardar

Headers

Body as form-data

Body as json

Variable	Valor	Descripción
Agregar una variable	Valor	Descripción (opcional)

Respuesta Esperada (Correcta)

Variable	Tipo de comparación	Valor
args	Existe	
headers -> Accept-Encoding	Igual a	{{encoding}}
args	Es un Diccionario	
Agregar una variable	-----	Valor

Respuesta Obtenida (18 de Julio de 2019 a las 23:11)

```

{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "tarjeta estudiante": "123456789"
  },
  "headers": {
    "Accept": "*/*",

```

Figura 26: Ejemplo de una prueba de escritura finalizada y ejecutada

Consulta de prueba de escritura: httpbin post

POST httpbin /post

Envia

La variable **args** no es distinta a {}

Ejecución correcta
Problemas en la obtención de algunas respuestas esperadas

Headers Body as form-data Body as json

Variable	Valor	Descripción
Agregar una variable	Valor	Descripción (opcional)

Resposta Esperada (Con problemas)

Variable	Tipo de comparación	Valor
args	Existe	
headers -> Accept-Encoding	Igual a	[[encoding]]
args	Distinto a	{}
Agregar una variable	-----	Valor

Resposta Obtenida (18 de Julio de 2019 a las 23:09)

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "tarjeta estudiante": "123456789"
  },
  "headers": {
    "Accept": "*//*",
    "Accept-Encoding": "gzip, deflate",
```

Figura 27: Ejemplo de una prueba de escritura ejecutada y con problemas

Luego, para facilitar la realización de algunas pruebas, se permite que el usuario genere secuencias de pruebas. Las cuales facilitan la ejecución de pruebas de regresión que deben ser ejecutadas cada vez que se modifica el código. Coloreando en verde aquellas pruebas que todas sus “Respuestas Esperadas” fueron satisfactoriamente comprobadas.

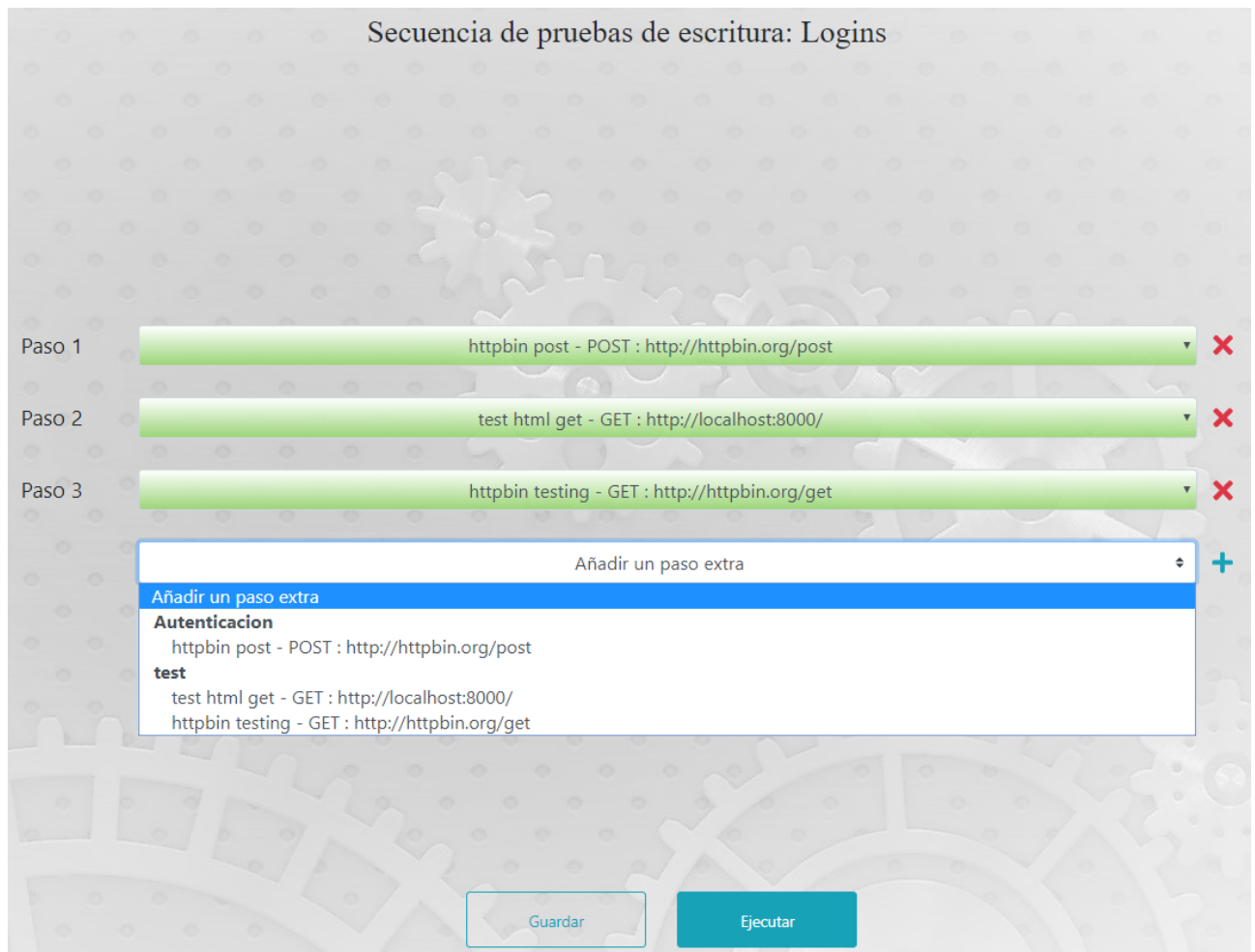


Figura 28: Ejemplo de una secuencia de pruebas de escritura

Pruebas de lectura

Las pruebas de lectura se enfocan principalmente en la lectura de la base de datos, enfocándose en comprobar que la lógica del negocio se cumple correctamente al igual que la consistencia de los datos luego de las transacciones.

Cada prueba de lectura debe poseer 2 datos esenciales para lograr ser ejecutada correctamente: Credenciales de la base de datos y consulta SQL.

1. Las credenciales de la base de datos utilizada, deben incluir como mínimo: el tipo de base de datos utilizada (actualmente Postgres u Oracle), el nombre del servicio y el nombre de usuario (la contraseña es necesaria si la base de datos posee una). Luego, el Host y Puerto no son necesarios, a menos que la base de datos esté definida en un servidor externo.

2. La consulta SQL es necesario que comience con “SELECT” por temas de seguridad de la base de datos, y para ser probada es necesario que se haya probado la conexión a la base de datos anteriormente. Luego de probada la consulta, se tendrá la posibilidad de ver las variables que se podrán definir en las validaciones.

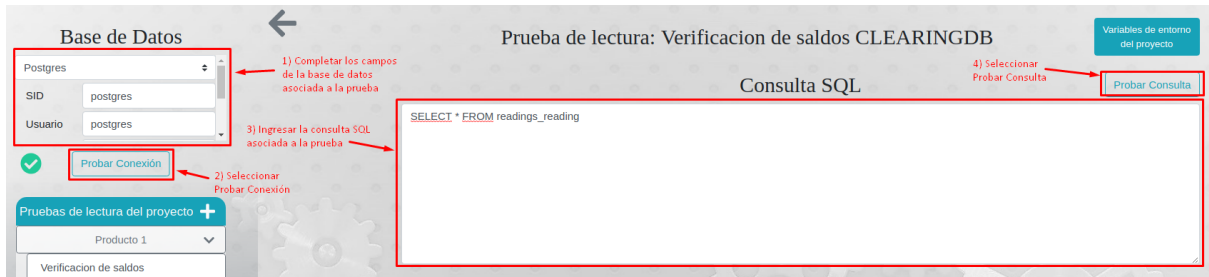


Figura 29: Pasos necesarios para realizar correctamente una prueba de lectura básica

Luego de haber logrado una ejecución básica, la herramienta provee al usuario la capacidad de realizar 2 tipos de validaciones de los datos obtenidos por la consulta SQL. Las validaciones de comparación y las validaciones secuenciales. Las cuales permiten utilizar código Python para la comparación de las variables obtenidas por la consulta SQL (observables en cualquier momento utilizando el botón “Ver Variables” generado luego de haber probado una consulta) y validar una variable objetivo que se verá en el resultado de la ejecución.

Se reitera, que en este punto las “validaciones” de los datos, se refiere a validaciones en la respuesta obtenida por la consulta SQL y en el orden que se haya especificado para esta. Por lo que cuando se habla de una “fila”, se refiere a un dato de la consulta, y la “fila siguiente” se refiere al dato siguiente al actual acorde al orden de la consulta SQL.

Las validaciones de comparación son sentencias lógicas que se utilizan para comparar variables de una misma fila en la consulta. Por ejemplo, si se desea que comprobar que una variable “saldo” sea mayor a 0, en el código Python se debiera anotar algo como:

```
if @tabla.saldo > 0:
    return True
```

Lo cual hará que en el resultado (un archivo .xlsx) se coloree y comente la Variable Objetivo acorde a lo seleccionado en la sección derecha de la validación. Los ítem “Al retornar Verdadero” y “Al retornar Falso”.

Las validaciones secuenciales, son sentencias lógicas que actúan de la misma manera que las validaciones de comparación, sin embargo, estas están pensadas para la comparación de variables que se incluyen en la fila anterior a la actual. Lógicamente, para que esto tenga sentido, el resultado de la consulta SQL debe estar ordenado según las transacciones que se estén evaluando.

En ambos tipos de validaciones, los campos “Código Python” y “Comentar” poseen la capacidad de referenciar las variables obtenidas, utilizando “@” y escribiendo parte del nombre de la variable, lo que generará una lista con las opciones de variables (se debe seleccionar alguna o terminar de escribir el nombre de la variable incluyendo el nombre de la tabla que la contiene). En el caso de las validaciones secuenciales, para referenciar la variable de la fila anterior, se debe escribir el nombre de la variable y agregar “_anterior” (al final de la variable). Por ejemplo, si se desea validar que el “saldo actual” sea igual al “saldo anterior” menos el “costo actual”, en el código Python se debiera anotar algo como:

```
if @tabla.saldo == @tabla.saldo_anterior - @tabla.costo:  
    return True
```

Para crear una validación, se debe seguir una serie de pasos que permitirán al usuario validar cualquier variable obtenida en la consulta SQL:

1. Seleccionar “Añadir Validación” en el tipo de validación que se quiera agregar (Comparación o Secuencial).
2. Seleccionar la variable objetivo de la validación que se realizará, por ejemplo, en una base de datos que posea la tabla “finanzas” y esta posee los atributos “saldo, fecha, costo”, si se desea validar que los saldos estén correctos, se debiera seleccionar la variable “*finanzas.saldo*” como variable objetivo ya que ésta será marcada como verdadera o falsa en el resultado de una ejecución. Además, para el ordenamiento de las validaciones, se dará la opción de escribir una breve descripción de lo que se revisará en la validación.
3. Se debe escribir un código en Python que realice la validación descrita anteriormente. Este paso debe ser realizado asumiendo que lo que se está escribiendo es el contenido de una “Función”¹⁰ de Python. Dentro de esta función se aceptará cualquier tipo de código mientras posea una sintaxis de Python, por lo que el alcance de posibles validaciones puede ser muy grande, sin embargo, se recomienda al usuario que cada validación posea una sintaxis simple (por ejemplo, limitándose a la utilización de “strings” y “comparaciones”) para una fácil comprensión de otros usuarios.
4. En la sección derecha de la validación, se refiere a 2 posibilidades del resultado de una validación: “Al retornar verdadero” y “Al retornar falso”. Cada una de estas poseen un campo “Colorear” y “Comentar”, las cuales hacen referencia a cuando el código Python escrito se cumple o no en cada “fila” del archivo *xlsx* resultante. Por ejemplo, si en el código Python se compara que la variable “saldos” debe ser mayor a 0 se retorne Verdadero, la herramienta realizará tal comparación en cada dato obtenido de la consulta SQL, si se cumple la condición, la “variable objetivo” mencionada en el paso 2 se coloreará y comentará según lo especificado en el campo “Al retornar verdadero”. En

¹⁰Funciones en Python: https://www.w3schools.com/python/python_functions.asp

caso de que la condición no se cumpla, la “variable objetivo” se coloreará y comentará según lo especificado en el campo “Al retornar falso”.

Para los usuarios que no poseen experiencia previa trabajando Python, al pie de la página se incluyen vínculos con cursos básicos de Python para definir lo que es un “string”¹¹ y para mostrar el modo correcto de uso de “comparaciones”¹². Un resumen simple sobre la definición de cada concepto y su sintaxis en Python es la siguiente:

- **Strings:** Los “strings” en Python se refieren básicamente a texto estático, el cual puede ser utilizado para realizar comparaciones o ser asignado a variables. La sintaxis que posee en python es simplemente utilizar comillas simples o comillas dobles alrededor de un texto. Ej:

```
print("Esto es texto")
print('esto también es texto')
print('123') #es texto y no un número debido a las comillas
```

Observación 1: “print” es un comando únicamente utilizado para imprimir por consola de python por lo que su uso no es importante en términos de la herramienta. Observación 2: “” es utilizado para comentar en una línea todo lo que esté después de ser mencionado, utilizado solo para el ordenamiento del programador.

- **Funciones:** Las funciones son utilizadas para crear marcos de trabajo en un código, permitiendo que se definan variables locales que no interfieran con otras funciones. Una función normalmente tiene como input una serie de variables, y esta puede terminar en cualquier momento utilizando el comando “return” para devolver un valor (en este caso, el valor retornado debe ser “True” o “False”). Ej:

```
def funcion_ejemplo(input1, input2, input3):
    print("Ejemplo de una función en python")
    return True
    #Estas líneas no se considerarán debido
    #a que están después de un return
```

En el caso de la herramienta, la primera línea en que se define el nombre de la función y los input, éstos son los valores obtenidos de cada fila, por lo que no debe ser definida.

- **Variables:** Las variables son asignaciones de valores específicos a nombres definidos por el usuario utilizando la sintaxis “variable = valor”. Luego, estas variables pueden ser tratadas y modificadas como se desee. Ej:

¹¹Definición de “strings” en Python: https://www.w3schools.com/python/python_strings.asp

¹²Comparaciones en Python: https://www.w3schools.com/python/python_conditions.asp

```
variable1 = 1
variable2 = 2
var_texto = "también se puede asignar texto en las variables"
variable2 = variable1 + variable2
#donde el valor de la variable "variable2" será "3"
```

- Comparaciones: Las comparaciones son una de las utilidades más importantes al momento de querer realizar validaciones de datos, donde se procede a comparar 2 variables o una variable junto a un texto o número, mediante la sintaxis "*variable1 comando variable2*". Donde "*comando*" puede definirse acorde a la comparación que se desea hacer:

- == : Para comparar si las variables son iguales
- != : Para comparar si las variables son distintas Nota: Para las siguientes comparaciones ambas variables deben ser numéricas y no strings.
- < : Para comparar si la primera variable es menor a la segunda
- <= : Para comparar si la primera variable es menor o igual a la segunda
- > : Para comparar si la primera variable es mayor a la segunda
- >= : Para comparar si la primera variable es mayor o igual a la segunda

Como resultado de cada comparación se obtendrá un resultado "True" si es verdadero o "False" si es falso. Luego, esto puede ser complementado utilizando "sentencias condicionales", donde se poseen 3 tipos de sentencias, "if", "elif" y "else". Las cuales poseen una sintaxis en que se debe definir "*if condicion:*" luego de un salto de línea y una indentación de todas las líneas que se deseen ejecutar en el caso de que la *condicion* sea verdadera. "elif" y "else" son complementarios a "if" en el caso de que la primera *condicion* sea False, se puede utilizar "*elif condicion2:*" para asignar una segunda condición, y "else:" en caso de que no se cumpla ninguna condición anterior. Ej:

```
var1 = 1
var2 = "1"
var3 = 1
if var1 == var2:
    print("esta linea se ignorará")
    #debido a que var1 es un numero y var2 un string
elif var1 >= var3:
    print("aqui si se entrará")
    #debido a que ambas son numericas y son iguales
else:
    print("no se llegará a este punto")
    #debido a que ya se aceptó una sentencia condicional anterior
```

Prueba de lectura: {[tarjeta estudiante]} Verificacion de saldos CLEARINGDB

Variables de entorno del proyecto
Ver Variables

Consulta SQL

SELECT * FROM readings_reading
INNER JOIN {[reading section]}
ON readings_sectionreading.id = readings_reading.section_id
ORDER BY readings_reading.id ASC;

Validaciones

Comparación

Codigo Python
Variable: readings_sectionreading.title
Descripción: Verificacion del titulo de la seccion

return @readings_sectionreading.title == "Producto 1"

Al retornar Verdadero
Al retornar Falso

Colorear
No colorear

Comentar
La validación pertenece a la sección @readings_sectionreading.title

Añadir validación

Secuencial

Codigo Python
Variable: readings_reading.id
Descripción: Atencion de los valores

return @readings_reading.id == (@readings_reading.id_anterior + 1)

Al retornar Verdadero
Al retornar Falso

Colorear
Verde

Comentar
No comentar la celda

Añadir validación

Figura 30: Ejemplo de una prueba de lectura ejecutada

	A	B	C	D	E	F	G
1	readings_reading.id	readings_reading.title	readings_reading.database	readings_reading.username	readings_reading.sql	readings_sectionreading.title	
2		1 Verificacion de saldos				Producto 1	
3		2 {[tarjeta estudiante]} Verificacion de postgresql	postgres	postgres	SELECT * FROM readings_reading	Producto 1	La validación pertenece a la sección Producto 1
4		3 Test 2	postgres			Producto 2	
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							

Figura 31: Ejemplo del resultado generado por una prueba de lectura

Finalmente, las pruebas de lectura poseen la opción de ejecutarse en secuencia de la misma manera que las pruebas de escritura. Generando un único archivo .xlsx que contendrá en páginas separadas cada una de las pruebas seleccionadas en la ejecución.

Página 49 de 63

Secuencia de pruebas de lectura: Secuencia de tarjetas

Paso 1

Paso 2

Añadir un paso extra

Producto 1
{{tarjeta estudiante}} Verificacion de saldos CLEARINGDB - postgres, postgres

Producto 2
Test 2 - postgres, postgres

Figura 32: Ejemplo de una secuencia de pruebas de lectura

	A	B	C
1	readings_sectionreading.id	readings_sectionreading.title	readings_sectionreading.project_id
2		1 Producto 1	1
3		2 Producto 2	1
4			
5			
6			
7			

123456789 Verificacion de saldo Test 2

Figura 33: Ejemplo del resultado de una secuencia de pruebas de lectura

Generación de Reportes

La Generación de reportes permite al usuario mostrar de forma ordenada los resultados de las pruebas que se posean. Para esto, se debe seleccionar “Generar Documentación” en la página que aparece luego de seleccionar el proyecto.

A partir de esto, se generará una lista ordenada con todas las pruebas que estén marcadas como “Construidas”. Ésta vista está pensada para poder filtrar las pruebas que el usuario requiera en cada ocasión, permitiendo a la vez seleccionar si el usuario requiere de solo el detalle de la prueba o también se requieren los resultados.

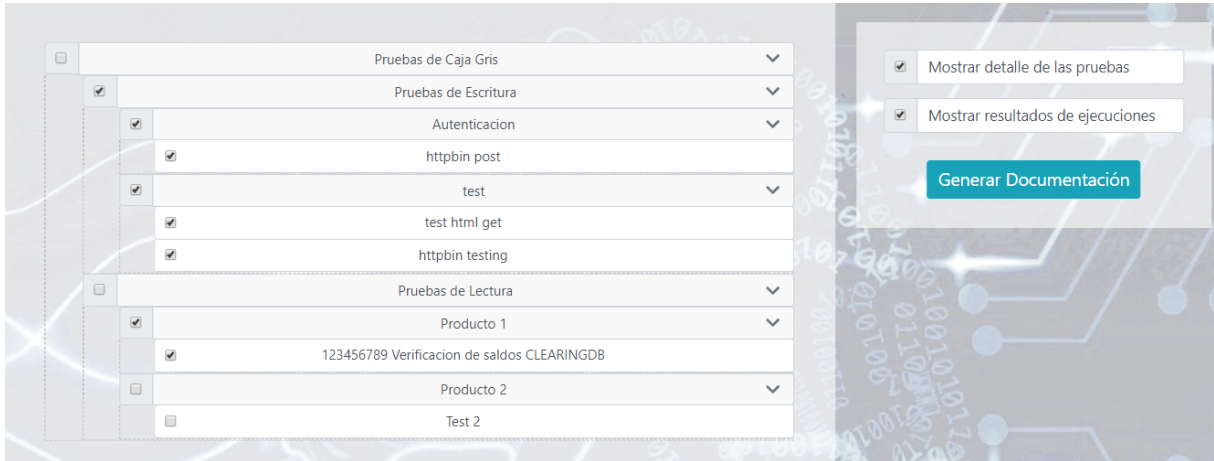


Figura 34: Ejemplo de vista para seleccionar reportes

Luego de esto, al seleccionar “Generar Documentación” se redireccionará a una vista que contendrá de forma ordenada cada una de las pruebas seleccionadas. En la cual existirá un botón en la esquina superior derecha llamado “Guardar como PDF”, el cual hará que se abra la ventana de imprimir. Una vez en la ventana, en el Destino debe seleccionarse la opción “Guardar como PDF” y para obtener una mejoría visual, también se debe seleccionar el checkbox “Gráficos de fondo”, para finalmente seleccionar “Guardar”.

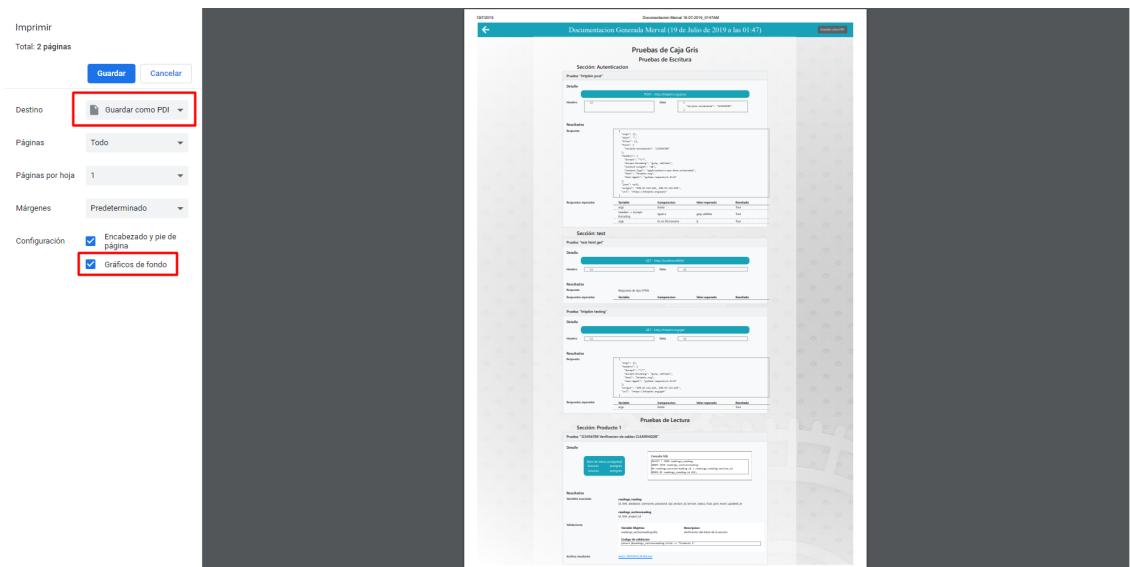


Figura 35: Ejemplo de como se debe Guardar correctamente un reporte generado

4.2. Aplicación en empresas

Para validar el funcionamiento correcto de la herramienta, se utilizan datos de proyectos reales en 2 empresas: Selyt y SONDA, cada uno enfocado en ejecutar pruebas de escritura y pruebas de lectura respectivamente.

4.2.1. Selyt: pruebas de escritura

Selyt, es una empresa en emprendimiento que se enfoca en el desarrollo de una aplicación móvil enfocada en ventas de productos. Además, esta aplicación utiliza un servidor API Rest para controlar el BackEnd del servicio, por lo que es una plataforma ideal para la utilización de pruebas de escritura.

En este proyecto, se hace uso de los mismos desarrolladores para la creación y utilización de las pruebas, para así probar y documentar 2 de los servicios más utilizados de la misma.

Consultas

Seuencias

Lista de rutas del proyecto +

Users

Login

Me

Register

Admin Login

Products

Brands

Añadir una Consulta

Consulta de prueba de escritura: Me

GETlocalhost/me

Enviar

Guardar

Headers

Body as form-data

Body as json

Variable	Valor	Descripción
Authorization	Bearer ([access_token])	Descripción (opcional)
Agregar una variable	Valor	Descripción (opcional)

Respuesta Esperada (Correcta)

Respuesta Obtenida (4 de Septiembre de 2019 a las 19:30)

```

{
  "role": {
    "id": 0,
    "name": "SELLER"
  },
  "email": "jorge.caullan.12@sansano.usm.cl",
  "groupId": "5d67e3d2f69d980025286619",
  "createdAt": "2019-09-02T18:23:30.844Z",
  "updatedAt": "2019-09-02T18:23:30.844Z",
  "profile": {
    "isTrained": false,
    "campaignId": [],
    "rut": "188394183",
    "name": "JORGE",
    "lastName": "CAULLAN",
  }
}

```

Consultas

Seuencias

Lista de rutas del proyecto +

Users

Login

Me

Register

Admin Login

Products

Brands

Añadir una Consulta

Consulta de prueba de escritura: Register

POSTlocalhost/users

Enviar

Guardar

Headers

Body as form-data

Body as json

```

{
  "email": "example@gmail.com",
  "rut": "18487483-0",
  "password": "asdasd123",
  "name": "Prueba",
  "lastName": "test",
  "phone": "123456789",
  "referralCode": "JORGE00789"
}

```

Respuesta Esperada (Con problemas)

Respuesta Obtenida (4 de Septiembre de 2019 a las 19:46)

```

{
  "role": {
    "id": 0,
    "name": "SELLER"
  },
  "email": "example@gmail.com",
  "groupId": "5d67e3d2f69d980025286619",
  "createdAt": "2019-09-04T23:46:49.035Z",
  "updatedAt": "2019-09-04T23:46:49.035Z",
  "profile": {
    "isTrained": false,
    "campaignId": [],
    "rut": "184874830",
    "name": "PRUEBA",
    "lastName": "TEST",
  }
}

```

Figura 36: Pruebas de escritura comprobando el funcionamiento de rutas de manejo de usuarios en la aplicación

Consultas

Seuencias

Lista de rutas del proyecto +

Users >

Products >

Brands v

All Brands

Create Brand  

Añadir una Consulta

Consulta de prueba de escritura: Create Brand

Variables de entorno del proyecto

Construida

POST localhost /brands

Enviar Guardar

Headers Body as form-data Body as json

```
{
  "name": "MarcaPrueba",
  "logo": "string",
  "banner": "Banner",
  "categoryId": "5d57e3d0f69d980025296574",
  "categories": [
    {
      "name": "string"
    }
  ],
  "steps": [

```

Respuesta Esperada (Correcta)

Respuesta Obtenida (4 de Septiembre de 2019 a las 20:06)

```
{
  "enabled": false,
  "name": "MarcaPrueba",
  "logo": "string",
  "banner": "Banner",
  "categories": [
    {
      "name": "string",
      "id": "5d70518b749cd605c496c6d9"
    }
  ],
  "steps": [
    {
      "_id": "5d70518b749cd605c496c6da",
      "position": 0,

```

Consultas

Seuencias

Lista de rutas del proyecto +

Users >

Products v

Create Product  

All Products

Brands >

Añadir una Consulta

Consulta de prueba de escritura: Create Product

Variables de entorno del proyecto

Construida

POST localhost /brands/[brandId]/products/

Enviar Guardar

Headers Body as form-data Body as json

```
{
  "name": "Example product",
  "description": "string",
  "price": 10000,
  "priceDescription": "string",
  "pricelessDescription": "string",
  "commission": 0,
  "discountPrice": 0,
  "discountDescription": "string",
  "images": [
    "string"
  ]
}
```

Respuesta Esperada (Correcta)

Variable	Tipo de comparación	Valor	
name	Existe		✗
name	Igual a	Example product	✗
price	Existe		✗
price	Igual a	10000	✗
Agregar una variable	-----	Valor	+

Respuesta Obtenida (4 de Septiembre de 2019 a las 20:04)

Figura 37: Pruebas de escritura comprobando el funcionamiento de rutas encargadas de la creación de datos en la aplicación

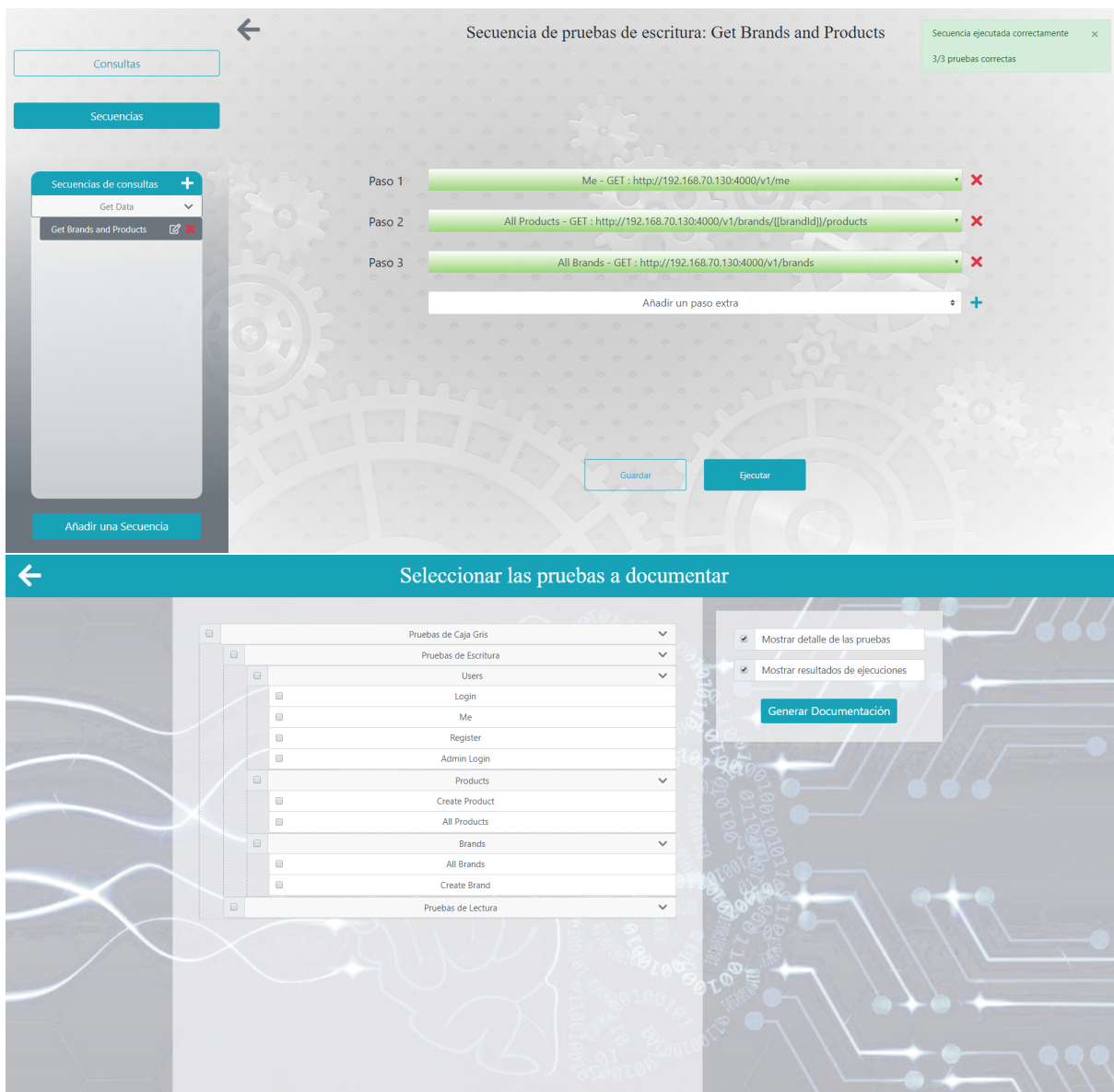


Figura 38: Ejecución de secuencias de pruebas de escritura y previsualización de documentación

Gracias a esto, logra generar automáticamente la siguiente documentación que pudo ser usada en adelante como guía para los nuevos desarrolladores.

4.2.2. SONDA: pruebas de lectura

SONDA, es una empresa de 45 años de antigüedad en el mercado (desde 1974) dedicada a los servicios de Tecnología de la Información (TI) con servicios en todo latinoamérica. Entre sus proyectos desarrollados en Chile, se puede mencionar Transantiago (transporte público de Santiago) y Merval (Metro de Valparaíso).

En SONDA, se hace uso de las pruebas de lectura debido a la integridad y consistencia que requieren las bases de datos que utilizan, debido a que en general manejan transacciones de dinero. Entre sus pruebas, se realizan validaciones acorde a las reglas del negocio del proyecto, lo cual les facilita bastante la realización de pruebas ya que cada proyecto posee reglas del negocio lo bastante específicas y complejas como para memorizarlas (se requiere apoyo visual para lograr validar ciertas reglas).

En las siguientes pruebas, se realizan lecturas y validaciones de varias bases de datos "Oracle" que poseen sus propias lógicas de utilización:

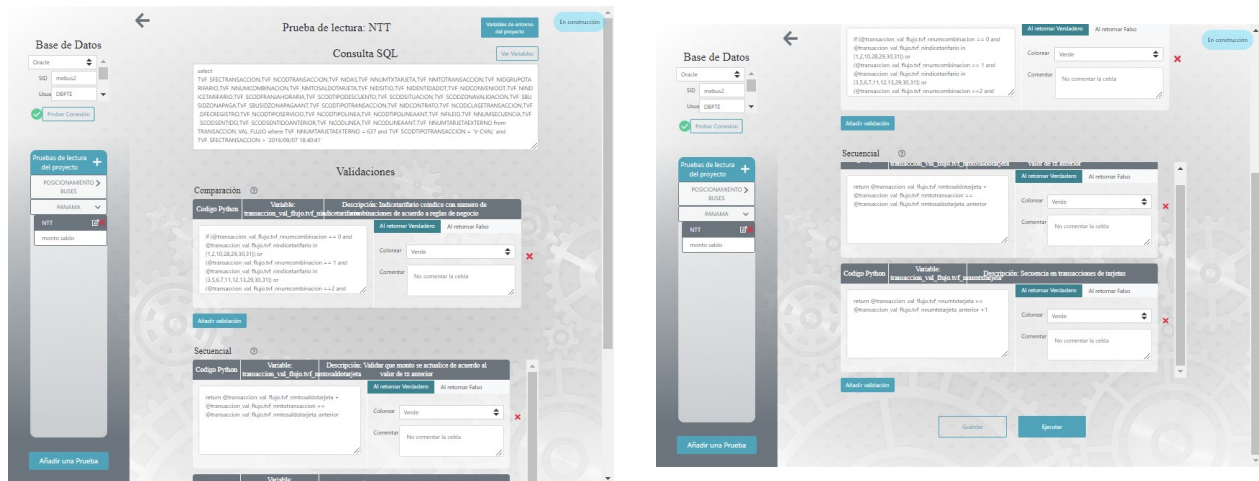
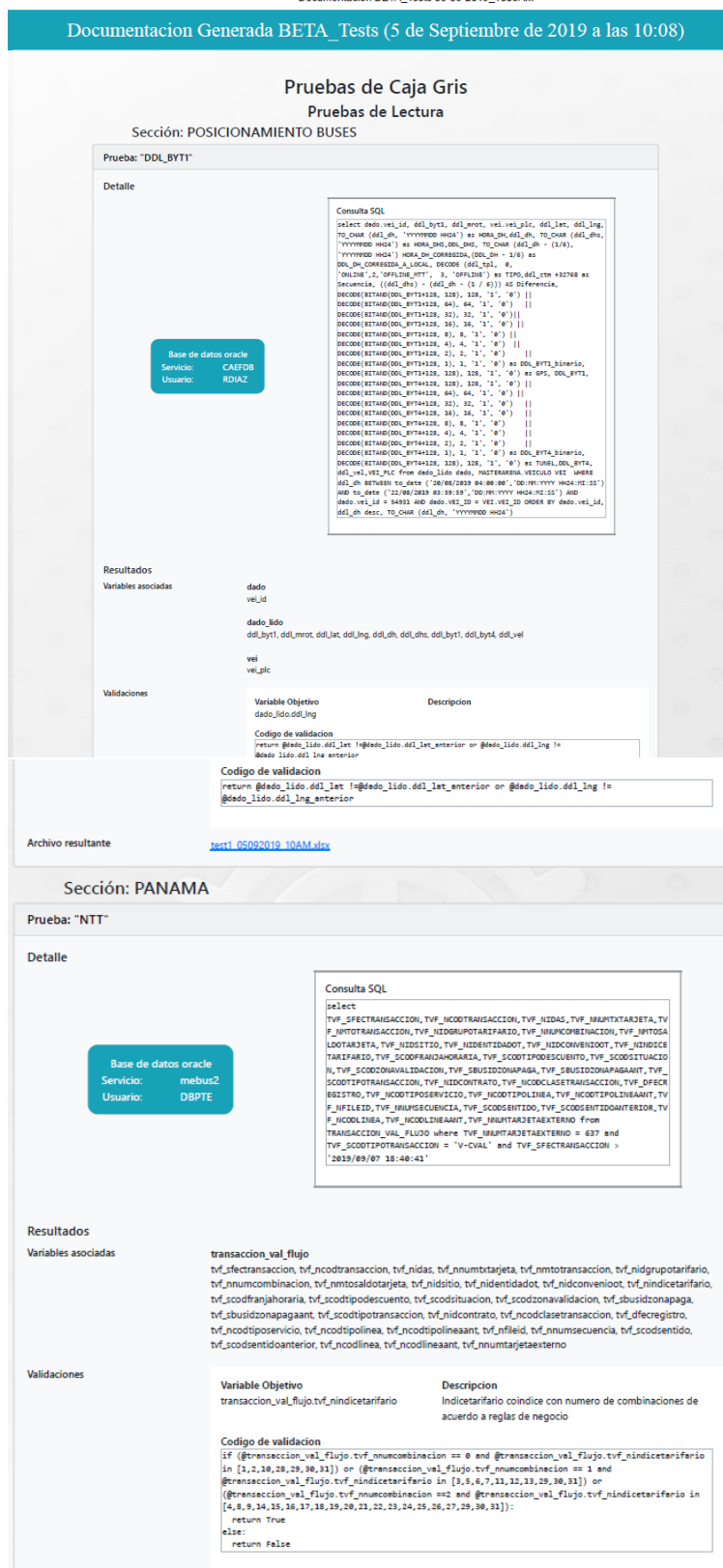


Figura 40: Prueba de lectura utilizada para validar que se encuentren todas las transacciones en un rango de fechas y resaltar las inconsistencias

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	transaccion_wal_bf_bf_electra	transaccion_wal_bf_bf_scofid	transaccion_wal_bf_bf_nides	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat	transaccion_wal_bf_bf_munat
	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda	moneda
1														
2	2013/09/07 18:40:42		19	620009	82	75		4700		1			1	
3	2013/09/07 18:40:43		19	620109	82	75		4700		1				
4	2013/09/07 18:41:02		19	620063	86	75		4665		1				
5	2013/09/07 18:41:13		19	620083	85	50		4613		29				
6	2013/09/07 18:41:24		20	626404	96	25		4390		30				
7	2013/09/07 18:41:47		19	620061	87	75		4513		11				
8	2013/09/07 18:41:58		20	6201209	88	0		4513		1				
9	2013/09/07 18:42:11		19	620138	89	0		4513		18				
10	2013/09/07 18:42:20		19	620136	87	0		4490		2				
11	2013/09/07 18:42:31		20	6201209	91	0		4490		0				
12	2013/09/07 18:42:41		19	620096	92	50		4440		7				
13	2013/09/07 18:42:52		20	6201209	93	0		4440		1				
14	2013/09/07 18:43:16		19	620083	94	50		4390		29				
15	2013/09/07 18:43:28		19	620060	95	30		4353		10				
16	2013/09/07 18:43:38		20	626404	96	0		4353		1				
17	2013/09/07 18:43:55		19	620066	97	75		4260		11				
18	2013/09/07 18:44:25		19	620138	98	0		4205		1				
19	2013/09/07 18:44:37		19	6201209	99	0		4205		1				
20	2013/09/07 18:44:54		19	620063	100	35		4170		13				
21	2013/09/07 18:45:07		20	626404	101	0		4170		1				
22	2013/09/10 13:14:28		19	620096	102	75		4095		2				
23	2013/09/10 13:15:09		19	6201301	104	25		4070		2				
24	2013/09/10 13:15:31		19	620160	105	25		4045		2				
25	2013/09/10 13:15:53		19	6201301	106	25		4020		2				
26	2013/09/10 13:15:21		19	6201301	107	25		3995		2				
27	2013/09/10 13:15:38		19	620160	108	25		3970		2				
28	2013/09/10 13:15:00		19	620160	109	25		3945		2				
29	2013/09/10 13:15:17		19	6201301	110	25		3895		2				
30	2013/09/10 13:15:39		19	620160	111	25		3870		2				
31	2013/09/10 14:02:32		19	620160	112	25		3745		2				
32	2013/09/10 14:06:56		19	6201301	113	25		3795		2				
33	2013/09/10 14:07:19		19	6201301	114	25		3770		2				
34	2013/09/10 14:07:41		19	620160	115	25		3745		2				
35	2013/09/10 14:02:36		19	620024	116	75		3440		1				
36	2013/09/10 16:42:51		19	6201301	128	25		3620		2				
37	2013/09/10 16:43:00		19	6201390	129	25		3585		28				
38	2013/09/10 16:44:29		19	620024	130	75		3440		1				
39														
40														
41														
42														
43														
44														

Figura 41: Resultado generado de la ejecución de la prueba expuesta en al figura 40



CAPÍTULO 5

CONCLUSIONES

Durante el proceso de desarrollo de la herramienta surgieron varios problemas respecto a cómo debiera ser el producto final, en donde uno de los mayores problemas fue la metodología que se utilizaría para realizar las validaciones en las pruebas de lectura. Debido a que estas validaciones deben utilizar una lógica muy específica para cada proyecto, existían 2 opciones en donde no se limitará el alcance de las validaciones:

- Utilizar una interfaz de árboles lógicos que permita definir las reglas del negocio.
- Utilizar extractos de código Python para capturar la lógica de una validación.

La opción por la que se optó en esta versión de la herramienta, fue la utilización de código Python (debido a que la implementación de una interfaz de árboles lógicos, sería demasiado lenta de desarrollar). Sin embargo, esto generó un problema con la premisa de la memoria, que el uso de la herramienta debe poder ser utilizado por usuarios no especializados en desarrollo, para mitigar esto, se hizo que el código que se debiera utilizar fuera lo mas simple posible y facilitando hacer referencias a las variables que se pueden utilizar para comparar. Además, como complemento, en la documentación de la herramienta se incluye una muy breve explicación de la sintaxis de Python y las principales funcionalidades que se necesitarán.

Luego, un segundo problema que se tuvo durante el desarrollo, se encontró una limitación con la posibilidad de realizar de pruebas de estrés. Donde se pudo notar que, la aplicación no tenía problemas para la ejecución de pocas pruebas a la vez, sin embargo, cuando la cantidad subía mucho, el servidor de la herramienta se “estresaba” antes que la aplicación que se deseaba testear. Para controlar esto, se puede hacer de varias maneras: utilizar un computador con mejores capacidades para la ejecución de las pruebas; utilizar varios computadores para que testearan en paralelo un servidor; o contratando un servicio de servidores que permitan ejecutar muchos “threads”.

Sin embargo, las 3 soluciones anteriores implicaban entre mucho tiempo de desarrollo (y complicaciones para la instalación) o la necesidad de pagar extra por el uso de la herramienta, lo cual seria una repercusión muy negativa debido a que la herramienta es “Open Source”, resultando en que no se implementara ninguna de las opciones.

Además de esto, en vista a lo que las empresas requerían, se incluyó en la herramienta la posibilidad de ejecutar secuencias de pruebas, tanto para pruebas de lectura como para pruebas de escritura. Gracias a esto, se facilita la ejecución de pruebas más completas sobre los proyectos.

Finalmente, se determinó que el desarrollo de las pruebas de caja negra serían demasiado lentas de desarrollar para esta memoria, por lo que solo se definió la metodología y la interfaz de esta. De esta forma, se puede decir que el trabajo futuro de esta memoria será:

- Creación de la árboles lógicos para la validación de pruebas de lectura.
- Permitir automatización de pruebas de caja negra haciendo uso de la metodología especificada.

Luego, completando esas tareas se podrá tener una herramienta base que cubra la mayoría de las posibles automatizaciones necesarias para un proyecto de software, la cual es utilizable por cualquier tipo de usuario con conocimientos básicos de software.

Además, al ser una herramienta “Open Source”, cada empresa que la desee utilizar podrá modificarla y adaptarla localmente para que se adapte a su empresa y metodologías.

REFERENCIAS BIBLIOGRÁFICAS

- [Garousi y Varma., 2010] Garousi, V. y Varma., T. (2010). A replicated survey of software testing practices in the canadian province of alberta: What has changed from 2004 to 2009? *Journal of Systems and Software*, 83.11:2251–2262.
- [Kasurinen y Smolander., 2010] Kasurinen, Jussi, O. T. y Smolander., K. (2010). Software test automation in practice: empirical observations. *Advances in Software Engineering*.
- [Müller, 2011] Müller, T. (2011). Libro programa de estudio de nivel básico para probador certificado. istqb.
- [Vahid y Elberzhager, 2017] Vahid, G. y Elberzhager, F. (2017). Test automation: not just for test execution. *IEEE Software*, 34.2:90–96.
- [Whittaker y Carollo., 2012] Whittaker, James A., J. A. y Carollo., J. (2012). How google tests software. Addison-Wesley.