Documento de evidencia de pruebas.

|  |  |
| --- | --- |
| Asignatura: TCY-0101 | Sección: 001V |

|  |  |
| --- | --- |
| Integrantes:  Karen Leyton.  Christian Villarroel. | Docente:  Patricio Soto. |

|  |  |
| --- | --- |
| Fecha: 06/06/2020 | Versión: 1.0 |

# Resumen.

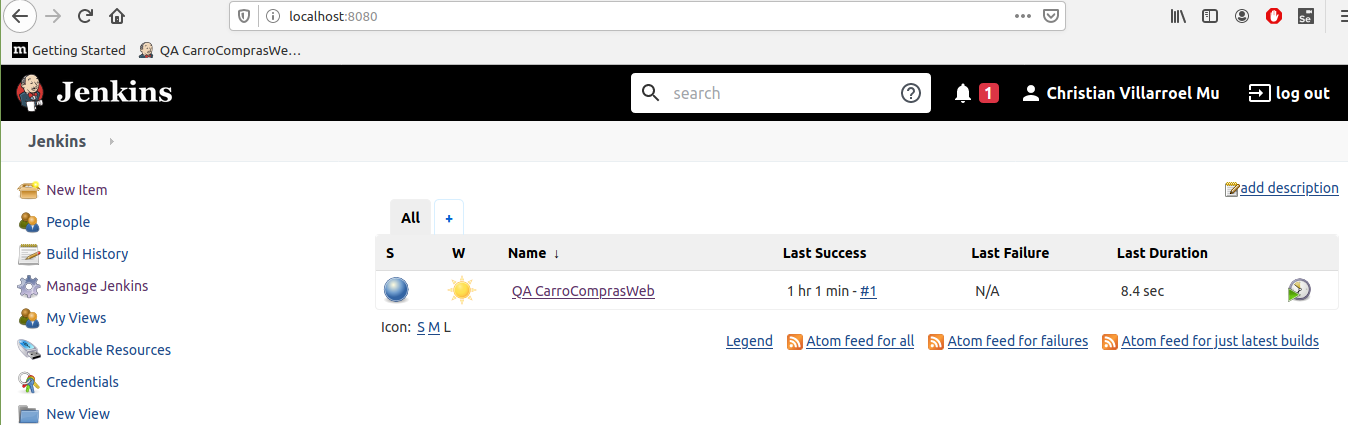
El presente informe tiene por objetivo el otorgar un analisis de caja gris para la aplicación “CarritoCompraWeb”, así como recomendaciones para la mejora continua del proyecto.

* Jenkins.
* Selenium.
* Test Unitarios: Junit & Mockito.
* SonarQube.

# Jenkins.

Mediante el uso de Jenkins, el equipo de Quality Assurance construyó una plataforma de integración continua (C.I.) y automatización mediante Jobs para el proyecto.

La utilidad de Jenkins radica en que pasa a ser el elemento orquestador de las distintas herramientas empleadas en el proyecto, este fue configurado para deployar directamente desde el repositorio de GitHub, automatizando la construcción y ejecución de las pruebas unitarias que se detallarán en los siguientes elementos.



|  |
| --- |
| Repositorio de Github:  <https://github.com/Chrisvimu/CalidadDeSW> |
|  |

Para poder manejar las distintas dependencias del proyecto, se realizó una migración de este a **Maven**, que es un orquestador de dependencias y que permite automatizar los procesos de building, testing y deployment para la aplicación.

La decisión de migrar el proyecto a Maven, se debe a la gran cantidad de conflictos que existían para la ambientación del proyecto en una plataforma de Linux, y para evitar ocupar dependencias del Ide en el cual fue compilado el proyecto, **Netbeans 8.2.2.**

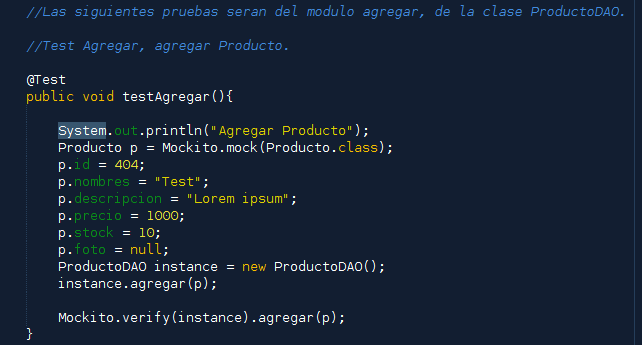
# Test Unitarios Junit & Mockito.

Los test unitarios de código fueron realizados mediante **Junit** un framework diseñado para realizar casos de prueba, en cuanto al uso dado por el equipo de trabajo este fue principalmente para comparar resultados esperados y comportamientos esperados del software. Además de esto se utilizó **Mockito**, que es un framework para pruebas utilizado para “simular” dependencias dentro del software, con esto se puede aislar la prueba de una función determinando si se presentan problemas en ella, y no en relación a otros objetos, clases o métodos de los cuales dependa.

Testing por funcion del modulo ProductoDAO:

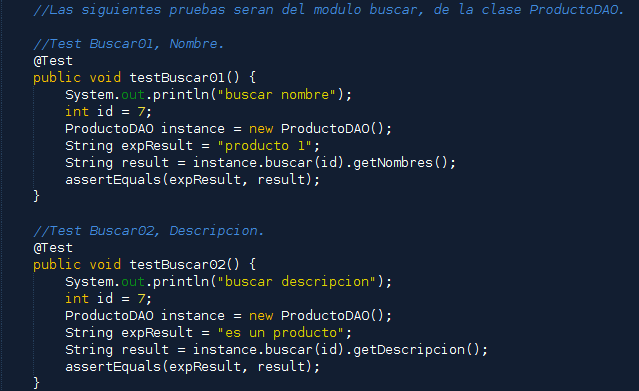
## Función Agregar.

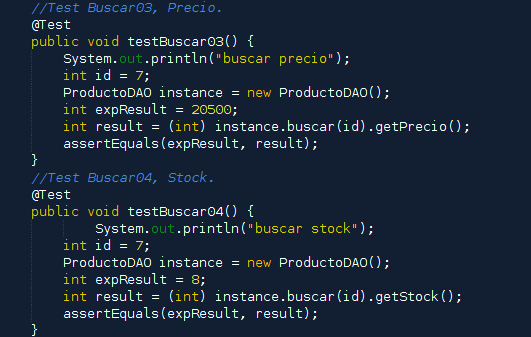
En el método se busca simular la clase de producto, luego asignar los valores de testing a esta, para posteriormente verificar que el método agregar producto funcione correctamente e inserte un objeto con los mismos atributos.



## Función Listar.

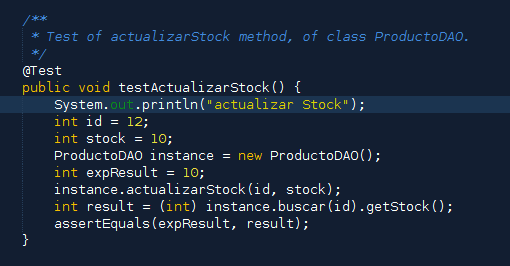
El test se separó en 4, para poder identificar errores por cada campo, las siguientes pruebas unitarias buscan identificar el resultado esperado “expResult” con el obtenido en la consulta mediante el método buscar.





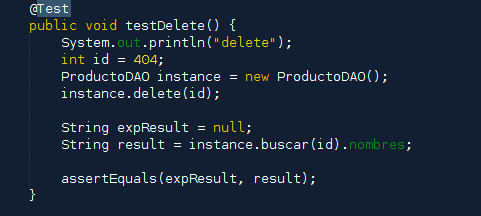
## Función Actualizar.

En este test, primero se asignan los parámetros a utilizar, “id” y “stock”, luego estos son utilizados para invocar la función de “actualizarStock”, con lo que se cambia el stock disponible para el producto con id 12. Luego se compara mediante un assert de Junit.



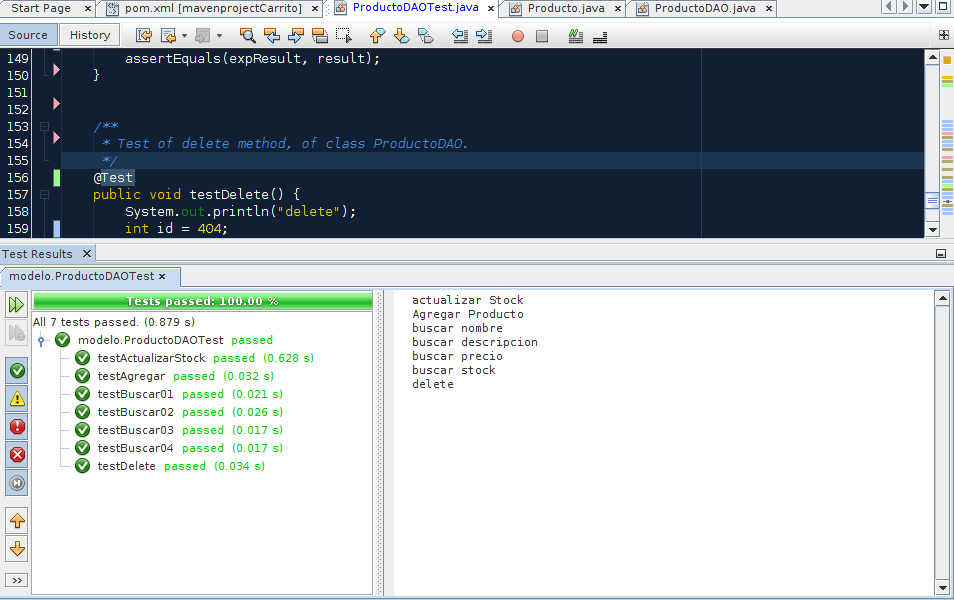
# Función Borrar.

En este test se asigna el id de 404, el cual previamente se utilizó para insertar un producto a la base de datos. Luego este es borrado y se corrobora mediante la búsqueda del nombre del producto.



# Resultados Test Unitarios.

Mediante las pruebas unitarias descritas anteriormente se pudo comprobar que los métodos del módulo Producto estaban correctamente codificados, estas pruebas se realizarán automáticamente en futuras builds del proyecto, mediante **Maven** y **Jenkings**.



# Testing con Selenium.

En cuanto a las pruebas web realizadas mediante Selenium, estas se encuentran detalladas en un documento externo, debido a la cantidad de pruebas involucradas, en el documento de evidencia se detalla la ejecución de las pruebas, con capturas de pantalla de los inputs y de los outputs de cada caso de prueba.

En el documento de “Casos de prueba Eva 2” se detallan las instrucciones para el tester sobre los casos de prueba web a realizar, así como el resultado del test.



# Resultados del análisis.

Consideramos que el software “CarritoComprasWeb” no cumple con los estándares mínimos de calidad de acuerdo al testing realizado y según las especificaciones del proyecto entregadas por el cliente.

En cuanto al testing realizado, el software aprobó únicamente un 13% de las pruebas realizadas, por lo que se recomienda realizar una iteración en el ciclo de desarrollo dentro de la cual se corrijan los test que dieron como resultado un fallo.

Para que el proyecto sea desarrollado de la forma más eficiente y eficaz, el equipo de Q/A presenta las siguientes recomendaciones:

1. Integrar el Software, sus dependencias y ambiente de desarrollo a **Docker**, esta herramienta permitirá al proyecto tener un ciclo de desarrollo continuo más expedito, ya que soluciona la dificultad más grande encontrada por el equipo para el testing del software: Dependencias obsoletas. Mediante la Dockerizacion del proyecto, este podrá ser deployado en cualquier entorno sin necesidad de una extensa configuración para su uso.
2. Integrar **Maven**, para mantener un control de versiones automático entre las dependencias del proyecto, y así poder automatizar el manejo de las dependencias y plugIns asociadas al sistema. Como equipo de Q/A decidimos tomar la iniciativa en la Mavenizacion del proyecto, configurando el archivo POM de este con las dependencias que a la fecha son necesarias para el funcionamiento del proyecto.
3. Aislar dependencias al entorno de desarrollo, para evitar enlazar el sistema al Ide de **Netbeans 8.2.2**, se recomienda compilar nuevamente los Jars y Wars del proyecto, ya que estos han sido compilados con elementos deprecados, que no se encuentran en otras versiones de Netbeans y **JDK.**
4. Desarrollar los otros módulos del producto que se encuentran incompletos, tales como LogIn, sistema interno de usuarios, empleados, administración de productos, y proceso de compra. Ya que al momento del testing únicamente el módulo correspondiente al carrito de compras funcionaba correctamente en el navegador.
5. Aplicar validaciones a los métodos, así como manejo de excepciones. Actualmente son inexistentes, pero se requiere un control de los inputs otorgados por los usuarios previo al parseo de estos dentro de la aplicación, como equipo de Q/A recomendamos implementar estos mediante **expresiones regulares**.