## S07 - P07

1. Como es un método con una funcionalidad sobre una base de datos, podemos realizar tests de integración comprobando que la inserción sobre esta base de datos se realiza correctamente. Podemos realizar pruebas de integración porque tiene una dependencia externa que no es una clase java, es una Base de Datos. Si no realizamos dobles, estaremos realizando pruebas de integración, ya que estaremos probando la interacción entre las unidades que conforman el proyecto.
2. DBUnit: La utilizaremos para poder usar el API de DBUnit en los tests.

JUnit: La emplearemos para implementar y ejecutar los tests.

Mysql: Sin esta dependencia no podremos hacer operaciones con la BD.

se situarían en el apartado <dependencies> del POM y tanto DBUnit como JUnit pondré el <scope> test porque esas dos dependencias sólo se usarán en src/test.

## S08 - P08

1. Hemos visto un modelo basado en escenarios, ya que nuestros test se han basado en un conjunto de requerimientos en un mismo escenario. Además describe un funcionamiento normal de la aplicación (crear usuario, comprar mascota...)

Los tests realizados con Katalon se han obtenido siguiendo el método de diseño de casos de prueba de caja negra (functional testing). Esto es porque se central en la "funcionalidad" y no en la implementación.

1. No se pueden integrar los tests de Katalon en el proceso de construcción del proyecto
   1. ¿Por qué? ya que se trata de comandos Selenese, los cuales en nuestro entorno de prácticas los hemos implementado en un plugin de firefox que permite crear scripts de control (action, accessors y assertions) sobre páginas web.
2. Caso de prueba diseñado con el método basado en escenarios:   
    Pepa es una alumna de la Universidad de Alicante matriculada en la asignatura PPSS de Ingeniería Informática. Semanalmente, tras finalizar sus ejercicios asociados a cada sesión de prácticas los sube a la plataforma BitBucket, en el repositorio personal de la asignatura. Dicho repositorio está compartido previamente con la profesora al cargo de la asignatura.  
    Pepa ha terminado los ejercicios de la sesión 8. Se logea en Bitbucket con sus credenciales y procede a subirlos mediante Git a un subdirectorio llamado *p08* del repositorio de la asignatura.   
    Una vez subidos mediante Git, se ha de comprobar si se han subido correctamente los 2 ejercicios que conforman la práctica.
3. Implementarlo:

|  |  |  |
| --- | --- | --- |
| Command | Target | Value |
| open | http://bitbuckect.org/account/signin |  |
| type | name=username | lll1@alu.ua.es |
| type | name=password | 123456 |
| clickAndWait | link=Log in |  |
| verifyText | //h1 | Your Work |
| clickAndWait | link=ppss-G6-lololo-2018 |  |
| verifyText | //h1 | ppss-G6-lololo-2018 |
| clickAndWait | link=P08 |  |
| verifyText | //h1 | P08 |
| verifyText | //table | Test Suite-Ejercicio1.html |
| verifyText | //table | Test Suite-Ejercicio2.html |

## S09 - P09

1. Si, los tests realizados en la práctica 8 podrían haber sido implementados con Selenium WebDriver.
   1. ¿Por qué? ya que WebDriver nos da las herramientas necesarias para crear tests con la misma finalidad que Selenium IDE, además cuenta con la gran ventaja de tratarse de un lenguaje de programación, por lo que dispone de ventajas como la reusabilidad de código o el hecho de poder iterar en un bucle.
2. Si, se pueden integrar los tests realizados con Selenium WebDriver en el proceso de construcción.
   1. ¿Cómo? necesitaremos hacer uso de la librería JUnit al igual que de la librería WebDriver, para poder implementar tests (anotados con la etiqueta @Test). Será necesario que añadamos la dependencia de Selenium-java en nuestro pom.xml para poder usar WebDriver. Esto sería posible añadiendo una goal al proceso de construcción.
3. Test WebDriver - Detective Conan (Realizar!!)

## S10 - P10

1. Un servidor proxy sirve para interceptar las peticiones HTTP generadas y grabarlas en un controlador tipo recording.
2. Por ejemplo, en la práctica usamos un servidor proxy junto con un controlador de grabación para interceptar las peticiones realizadas para loguearnos y hacer sign out, obteniendo así sus correspondientes samplers.
3. Pasos para agregarlo:
   1. Sobre el Banco de Trabajo: Añadir > ElementosDeNoPrueba > Servidor HTTP proxy
   2. Sobre el Plan de Pruebas: Añadir > Hilos > Grupo de Hilos
   3. Sobre el Grupo de Hilos: Añadir > Controlador Lógico > Controlador Grabación
   4. Navegador Firefox: Preferencias > Servidor proxy > Configuración manual del proxy
   5. Desde JMeter, en el Servidor proxy: clicamos sobre Arrancar
4. Secuencia de peticiones HTTP ejecutadas:
   1. News Page, Log Page, FAQ Page, Log Page, Gump Page, Log Page, News Page, Log Page

## S11 - P11

1. “una cobertura de condiciones (nivel 3) del 100% no implica una cobertura de líneas (nivel 1) del 100%"
   1. justificación en código

if( a && b) { … } // Casos de prueba: 1 → a = true + b = false | 2 → a = false + b = true

Un 100% de cobertura en condiciones implica que *a* se pruebe en sus vertientes V/F y *b* se pruebe también en sus vertientes V/F. No implica que la decisión *a && b* se tenga que cumplir en sus vertientes V/F (nivel 2), por lo que en decisiones con condiciones múltiples es posible que se den esos casos donde no se llegue a ejecutar el 100% de líneas.

1. informe de Cobertura → métricas
   1. Statement coverage: número de líneas que se ejecutan
   2. Branch coverage: número de condiciones ejecutadas en sus 2 vertientes (V/F)
   3. Complexity (CC): cota superior del número de casos de prueba necesarios para cubrir todos los caminos linealmente dependientes

## S12 - P12

1. V/F?
   1. F. No es un requisito, es una recomendación practicar CI en modelos iterativos. En XP es un principio básico, pero en otro modelo iterativo, como es UP, no, es opcional. (?)
   2. F. El alcance de las pruebas se planifica en función a los recursos disponibles (tiempo y coste). Por tanto, se puede saber con anterioridad a la implementación de los tests.
   3. F. El modelo en V realiza una planificación de las pruebas en un modelo de desarrollo secuencial (¿esto no implica que se emplee planificación predictiva, no?)
2. Construir un plan según un modelo W

|  |  |  |  |
| --- | --- | --- | --- |
| TAREAS | Objetivo concreto de cada tarea | Herramienta/método asociado a la ejecución de la tarea | En prácticas hemos llevado a cabo esta tarea, y concretamente hemos hecho ... |
| Tarea 1 | Preparar análisis del código | Definir mínimos de cobertura de código |  |
| Tarea 2 | Preparar tests de aceptación de propiedades emergentes no funcionales | Pruebas de rendimiento (CN) |  |
| Tarea 3 | Preparar tests de aceptación de propiedades emergentes funcionales | Basado en requerimientos (CN)  Basado en escenarios (CN) |  |
| Tarea 4 | Preparar tests de integración | Guías en función de los distintos tipos de interfaces (CN) |  |
|  | UNITARIOS |  |  |
|  | UNITARIOS |  |  |
| Tarea 5 | Ejecutar tests de integración | Java, Maven, JUnit, DBUnit |  |
| Tarea 6 | Ejecutar tests de aceptación de propiedades emergentes funcionales | Katalon recorder  Java, Maven, JUnit, WebDriver |  |
| Tarea 7 | Ejecutar tests de aceptación de propiedades emergentes no funcionales | JMeter |  |
| Tarea 8 | Ejecutar análisis del código | Java, Maven, Cobertura |  |