



Integración de un
Sistema Localizador para
el Control de Envíos

CATÁLOGO DE PRUEBAS




Tecnología e Innovación

Historial de Versiones

Fecha	Versión	Descripción	Autor
30/05/2020	1.0	Permite controlar el cumplimiento de un listado de requisitos o recolectar datos ordenadamente y de manera sistemática.	Nilton Pérez


Revisado y Aprobado por:

Nombre	Rol	Firma
Jose Quispe	Jefe del proyecto	
Jhosmell Alfaro	Desarrollador	
Guimer Coaquira	Diseñador	
Jhon Aguilar	Analista	
Nilton Perez	Téster	

	FORMATO
	PRUEBAS UNITARIAS

INTRODUCCION

El presente documento tiene por función proveer un catálogo de pruebas de software con el fin de aislar una parte del código y comprobar que funciona a la perfección. Son pequeños test que validan el comportamiento de un objeto y la lógica.

	FORMATO
	PRUEBAS UNITARIAS

1. PRUEBAS UNITARIAS

Las pruebas unitarias consisten en aislar una parte del código y comprobar que funciona a la perfección. Son pequeños test que validan el comportamiento de un objeto y la lógica.


El unit testing suele realizarse durante la fase de desarrollo de aplicaciones de software o móviles. Normalmente las llevan a cabo los desarrolladores, aunque en la práctica, también pueden realizarlas los responsables de QA.

Hay una especie de mito respecto a las pruebas unitarias. Algunos desarrolladores están convencidos de que son una pérdida de tiempo y las evitan buscando ahorrar tiempo.

Con ellas se detectan antes errores que, sin las pruebas unitarias, no se podrían detectar hasta fases más avanzadas como las pruebas de sistema, de integración e incluso en la beta.

Realizar pruebas unitarias con regularidad supone, al final, un ahorro de tiempo y dinero.



	FORMATO
	PRUEBAS UNITARIAS

PRUEBA UNITARIA EN APLICACIÓN MOVIL

Las aplicaciones de Android se ejecutan en una variedad de dispositivos. Además, el marco de Android y los marcos de código abierto circundantes evolucionan a gran velocidad. Para asegurarse de que su aplicación funcione bien, es importante escribir pruebas de software. Esto le ayuda a mejorar y mantener la aplicación de Android.




- Las pruebas unitarias para Android se pueden clasificar en:
- Pruebas de unidad local: pruebas que pueden ejecutarse en la JVM.
- Pruebas unitarias instrumentadas: pruebas que requieren el sistema Android.

El objetivo de las pruebas unitarias de Android es aislar cada parte del programa y mostrar que las partes individuales son correctas. Una prueba unitaria proporciona un contrato estricto y por escrito que el código debe cumplir. Como resultado, ofrece varios beneficios.

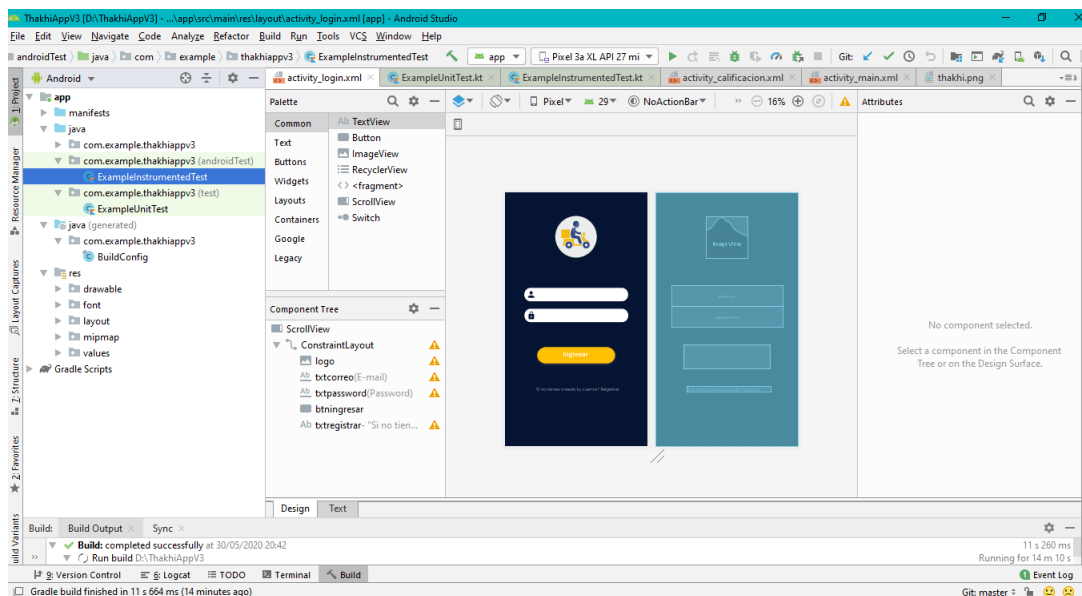
Para probar apps de Android, en general, creas estos tipos de pruebas automatizadas de la IU:

Pruebas locales: Pruebas de unidades que se ejecutan solo en la máquina local. Estas pruebas se compilan para ejecutarse localmente en la máquina virtual Java (JVM) para minimizar el tiempo de ejecución. Si las pruebas dependen de objetos en el marco de trabajo de Android, recomendamos usar Robolectric. Para las pruebas que dependen de tus propias dependencias, usa objetos ficticios para emular el comportamiento de las dependencias.

	FORMATO
	PRUEBAS UNITARIAS

Pruebas instrumentadas: Las pruebas de unidades que se ejecutan en un dispositivo o un emulador de Android. Estas pruebas tienen acceso a información de instrumentación, como el Context para la app que se prueba. Utiliza este enfoque para ejecutar pruebas de unidades que tienen dependencias complejas de Android que requieren un entorno más eficaz, como Robolectric.


Las pruebas unitarias locales de un proyecto de Android deben ubicarse en la app/src/testcarpeta.



AGREGAR UNA PRUEBA

Para crear una prueba instrumentada o de unidad local, puedes crear una prueba nueva para una clase o un método específico siguiendo estos pasos:

- Abre el archivo Java que contenga el código que desees probar.
- Haz clic en la clase o el método que desees probar y, luego, presiona Ctrl + mayúscula + T (⇧⌘T).
- En el menú que aparece, haz clic en Create New Test.
- En el diálogo Create Test, edita cualquier campo, selecciona cualquier método que quieras generar y haz clic en OK.
- En el diálogo Choose Destination Directory, haz clic en el conjunto de fuentes correspondiente al tipo de prueba que desees crear: androidTest para una prueba instrumentada o test para una prueba de unidad local. Luego, haz clic en OK.

	FORMATO
	PRUEBAS UNITARIAS

También asegúrate de especificar las dependencias de bibliotecas de pruebas en el archivo build.gradle del módulo de tu app:

```
dependencies {
    // Required for local unit tests (JUnit 4 framework)
    testImplementation 'junit:junit:4.12'

    // Required for instrumented tests
    androidTestImplementation 'com.android.support:support-annotations:24.0.0'
    androidTestImplementation 'com.android.support.test:runner:0.5'
}
```

EJECUTAR PRUEBA

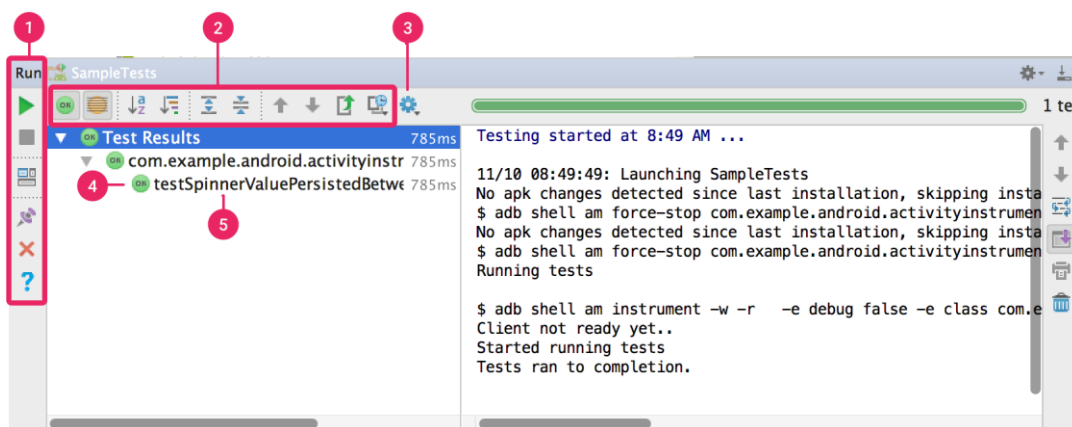
Para ejecutar una prueba, haz lo siguiente:

- Asegúrate de que tu proyecto esté sincronizado con Gradle. Para ello, haz clic en la opción Sync Project de la barra de herramientas.
- Ejecuta tu prueba de una de las siguientes maneras:
 - En la ventana Project, haz clic con el botón derecho en una prueba y selecciona Run.
 - En el editor de código, haz clic con el botón derecho en una clase o un método del archivo de prueba y selecciona Run a fin de ejecutar todos los métodos en la clase.
 - Para ejecutar todas las pruebas, haz clic con el botón derecho en el directorio de prueba y selecciona Run tests.


De forma predeterminada, tu prueba se ejecuta con la configuración de ejecución predeterminada de Android Studio. Si deseas cambiar alguna de las configuraciones de ejecución, como las opciones del ejecutor de instrumentación y de implementación, puedes editar la configuración de ejecución en el diálogo Run/Debug Configurations (haz clic en Run > Edit Configurations).

VER LOS RESULTADOS DE LA PRUEBA

Cuando ejecutas una prueba instrumentada o de JUnit, los resultados de muestran en la ventana Run. Una barra verde significa que se realizaron correctamente todas las pruebas, mientras que una roja indica que falló al menos una. La figura 3 muestra una prueba de ejecución que se ejecutó de manera correcta.



1. Usa la barra de herramientas de ejecución para volver a ejecutar o detener la prueba actual, volver a ejecutar las pruebas con errores (que no se muestran porque solo está disponible para las pruebas de unidades), pausar los resultados y volcar conversaciones.
2. Usa la barra de herramientas de pruebas para filtrar y ordenar resultados de pruebas. También puedes expandir o contraer nodos, mostrar la cobertura de la prueba y exportar o importar resultados de pruebas.
3. Haz clic en el menú contextual para realizar un seguimiento de la prueba de ejecución, mostrar las estadísticas de intercalado, abrir el código fuente en una excepción, desplazarte automáticamente a la fuente y seleccionar la primera prueba con errores una vez que se complete la ejecución.
4. Los íconos de estado de la prueba indican si una prueba tiene un error, se ignoró, falló, está en curso, se completó correctamente, está pausada, se canceló o no se ejecutó.
5. Haz clic con el botón derecho en una línea en la vista de árbol para ver un menú contextual que te permita ejecutar las pruebas en el modo de depuración, abrir el archivo de código fuente de la prueba o saltar a la línea del código fuente que se está probando.

	FORMATO
	PRUEBAS UNITARIAS

PRUEBA UNITARIA EN APLICACIÓN WEB

En nuestro trabajo diario en Autentia realizamos consultoría para el desarrollo de portales empresariales. Todos los desarrollos de estas características implementan los siguientes elementos:

- Funciones de seguridad en el código, para intentar prevenir o minimizar los posibles ataques maliciosos contra nuestras aplicaciones.
- Compilación y despliegue automáticos y desatendidos (compilación nocturna).
- Funciones de traza tanto funcionales como de depuración (log).
- Pruebas unitarias de todos los componentes desarrollados.

Una prueba unitaria es una prueba que comprueba el correcto funcionamiento de una funcionalidad incluida en la aplicación, tanto en las condiciones favorables como en las no favorables (por ejemplo, entradas correctas e incorrectas del usuario para un determinado valor). Las pruebas unitarias normalmente se desarrollan como funciones que utilizan los componentes ya desarrollados, y se lanzan como una batería de pruebas que sólo tienen dos posibles resultados: correcta o incorrecta.

Para el mundo Java hay un estándar open source para el desarrollo de pruebas unitarias para los componentes desarrollados en lenguaje Java: JUnit. El framework JUnit se extiende a menudo con otra serie de frameworks que realizan pruebas unitarias más específicas, insertándose a menudo como plugins en los entornos de desarrollo.

Toda aplicación Web se ejecuta dentro de un servidor Web o de un servidor de aplicaciones. Estos elementos hacen de contenedor para la aplicación, conteniendo los recursos, datos y objetos que la forman.

PRUEBAS UNITARIAS EN PHP CON PHPUNIT

Las pruebas unitarias ayudan a poder hacer test de nuestros módulos del sistema, ayudando a hacer más rápido el desarrollo sin tener que probar directamente en el navegador. PHPUnit es el framework de Testing más usado en PHP.



PHPUnit es un framework de Testing usado en aplicaciones PHP con el que podemos hacer pruebas de nuestro código, desde pruebas a clases y módulos hasta peticiones HTTP validando estatus y contenido. Nos ayuda a dejar de lado el navegador al momento de hacer código y evitar hacer los típicos echo \$var para probar funcionalidades, también apoya a no recargar la página cada que hacemos algún cambio. Suele usarse el patrón TDD para el diseño de pruebas, aunque no es indispensable. Las pruebas también sirven para en un momento avanzado del proyecto poder refactorizar código y verificar que no se haya roto nada, al correr las pruebas unitarias y verificar que todo funciona bien, podemos asegurar que no hemos quebrado la funcionalidad y que no se ha afectado otros módulos.


Instalación

En la terminal, vamos al directorio del proyecto y lo instalamos con composer. Aunque puedes instalarlo de otras formas como lo dice su página oficial.

```
composer require --dev phpunit/phpunit
```

Aserciones que se utilizaran en las pruebas unitarias

Las aserciones o Asserts son la parte fundamental de cualquier framework de testing, que, en términos generales, son comparaciones parecidas a un IF que ayudan a verificar que un módulo, clase, función, etc, esté enviando la respuesta esperada.

	FORMATO
	PRUEBAS UNITARIAS

1.1.1. AssertArrayHasKey

Comprueba que un objeto exista como llave dentro de un array.

```
public function testArray()
{
    $this->assertArrayHasKey('key', ['key' => 'value']);
}
```

1.1.2. AssertContains

Verifica que un elemento exista dentro de un array.

```
public function testArray()
{
    $this->assertContains('car', ['apple', 'car', 'blue']);
}
```

1.1.3. AssertDirectoryExists

Revisa que exista un directorio.

```
public function testDirectory()
{
    $this->assertDirectoryExists('path/to/directory');
}
```

1.1.4. AssertEmpty

Si un elemento es vacío la prueba será válida. Vacío es que sea null, [], "" o 0. La función inversa es assertNotEmpty.

```
public function testEmpty()
{
    $this->assertEmpty('path/to/directory');
}
```

1.1.5. AssertEquals

Verifica que dos valores sean iguales. El inverso de esta función es assertNotEquals.

```
public function testEquals()
{
    $this->assertEquals('a', 'a');
}
```

AssertFalse

Señala que una condición o valor sea false. El inverso es assertTrue.

```
public function testBoolean()
{
    $this->assertFalse('1' === 1);
}
```

1.1.6. Ejecutando una prueba con PHPUNIT

1. Las pruebas para una clase `Class` van dentro de una clase `ClassTest`.
2. `ClassTest` hereda (en la mayoría de los casos) de `PHPUnit\Framework\TestCase`.
3. Las pruebas son métodos públicos y toman como nombre `test*`.

También, podemos usar la anotación de *docblock* `@test` en el método para marcarlo como un método de prueba.

4. Dentro de los métodos de prueba están los métodos de aserción, como `assertSame()` (ver Aserciones), que se usan para determinar o aseverar si el valor real coincide con el valor esperado.

```
<?php
use PHPUnit\Framework\TestCase;

class StackTest extends TestCase
{
    public function testPushAndPop()
    {
        $stack = [];
        $this->assertSame(0, count($stack));

        array_push($stack, 'foo');
        $this->assertSame('foo', $stack[count($stack)-1]);
        $this->assertSame(1, count($stack));

        $this->assertSame('foo', array_pop($stack));
        $this->assertSame(0, count($stack));
    }
}
```