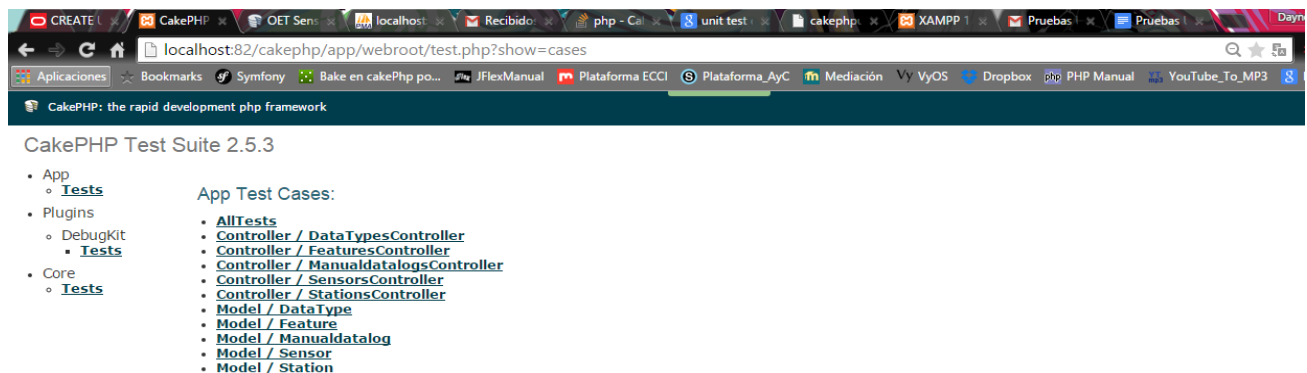


# Pruebas unitarias 2<sup>do</sup> Sprint Grupo Sensores:

Para realizar las pruebas unitarias se realizó la correcta actualización de phpunit ( Versión PHPUnit: 3.7). Además se utilizó la suite de pruebas de CakePHP V.2. Las entidades a probar para este sprint son:

**Sensor, DataType, ManualDatalog, Station y Feature :**



A continuación se mostrará las pruebas que se realizaron para probar el modelo Station. Para los demás modelos las pruebas fueron similares.

Nota: al final de esta sección se muestran las pruebas realizadas a cada modelo con phpunit \*.

## Creación de Fixtures:

Los datos insertados en la Base de Datos, para hacer los tests, son los siguientes, de acuerdo al modelo :

```
<?php
class StationFixture extends CakeTestFixture {
/**
 * Fields
 *
 * @var array
 */
    public $fields = array(
        'id' => array('type' => 'integer', 'null' => false, 'default' => null, 'length' => 10, 'unsigned' => true, 'key' => 'primary'),
        'station' => array('type' => 'integer', 'null' => true, 'default' => null, 'unsigned' => false),
        'description' => array('type' => 'string', 'null' => true, 'default' => null, 'length' => 2550, 'collate' => 'utf8_spanish_ci', 'charset' => 'utf8'),
        'indexes' => array(
            'PRIMARY' => array('column' => 'id', 'unique' => 1)
        ),
        'tableParameters' => array('charset' => 'utf8', 'collate' => 'utf8_spanish_ci', 'engine' => 'InnoDB')
    );

/**
 * Records
 *
 * @var array
 */
    public $records = array(
        array(
            'id' => '1',
            'station' => '1',
            'description' => 'Palo Verde'
        )
    );
}
```

## Pruebas unitarias Station Model:

Para probar el modelo y así comprobar que la comunicación con la base de datos está bien se hicieron dos test en el modelo.

Estos fueron, el primero, sirve para probar que al pedir todos los productos de la base de datos efectivamente se retornen todos los registros que deberían estar en la base de datos.

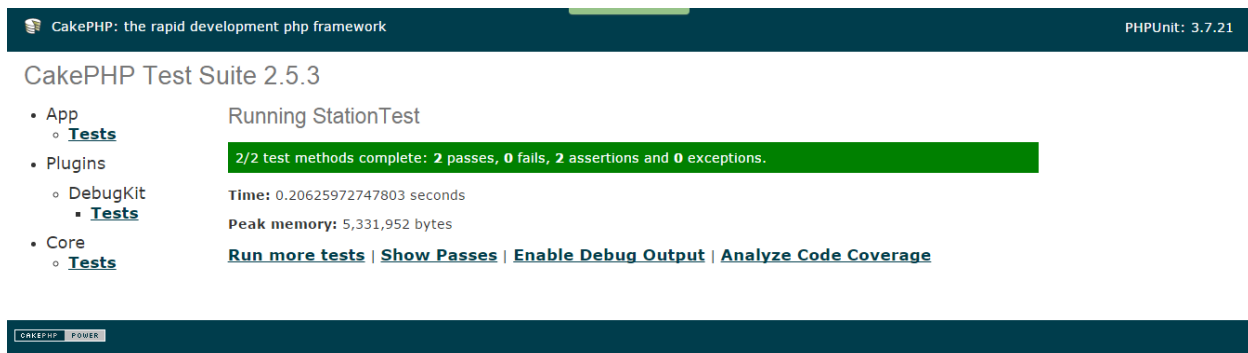
Para lograr probarlo se creó la función **getAllStation()** en el modelo, dicha función es la siguiente:

```
<?php
//Probando:Obtener todo de la base de datos.
public function getAllStations(){
    return $this->find('all',array(
        'fields' => array('id','station','description')));
}

//Probando:Obtener todo de la base de datos según Id.
public function getStationsFiltrados($id = null) {
    return $this->find('first', array(
        'conditions' => array('Station.id' => $id),
        'fields' => array('id','station','description')
    ));
}
```

El segundo sirve para probar si efectivamente las condiciones de consulta se hacen correctamente desde el modelo, para esto se creó la función **getStationsFiltrados(\$id)**, el cual por medio del id, llave primaria, como criterio de consulta, se restringe la devolución de registros.

El resultado de ejecutar las pruebas del modelo Station se pueden observar en la siguiente figura:



The screenshot shows the CakePHP Test Suite 2.5.3 interface. On the left, a sidebar lists the test structure: App (Tests), Plugins (DebugKit, Tests), and Core (Tests). The main area displays the results for 'Running StationTest'. A green banner indicates '2/2 test methods complete: 2 passes, 0 fails, 2 assertions and 0 exceptions.' Below this, the execution time is '0.20625972747803 seconds' and the peak memory is '5,331,952 bytes'. At the bottom, there are links to 'Run more tests', 'Show Passes', 'Enable Debug Output', and 'Analyze Code Coverage'.

En la impresión de las consultas de la base de datos de la siguiente figura se notan los dos select que se hacen el primero que pide todos los registros y el segundo que está restringido con respecto al id = 1. Que corresponden a los llamados de las funciones **getAllStations** y **getStationsFiltrados** :

```
14 SELECT `Station`.`id`, `Station`.`station`, `Station`.`description` FROM `test`.`stations` AS `Station` WHERE 1 = 1
30 SELECT `Station`.`id`, `Station`.`station`, `Station`.`description` FROM `test`.`stations` AS `Station` WHERE `Station`.`id` = 1
```

\* Los resultados de correr todas las pruebas de todos los otros modelos:

CakePHP: the rapid development php framework

PHPUnit: 3.7.21

CakePHP Test Suite 2.5.3

App

Tests

Running DataTypeTest

2/2 test methods complete: 2 passes, 0 fails, 2 assertions and 0 exceptions.

Time: 0.24293684959412 seconds

Peak memory: 5,245,336 bytes

Run more tests | Show Passes | Enable Debug Output | Analyze Code Coverage

Plugins

DebugKit

Tests

Core

Tests

CakePHP: the rapid development php framework

PHPUnit: 3.7.21

CakePHP Test Suite 2.5.3

App

Tests

Running SensorTest

2/2 test methods complete: 2 passes, 0 fails, 2 assertions and 0 exceptions.

Time: 0.10185289382935 seconds

Peak memory: 5,391,000 bytes

Run more tests | Show Passes | Enable Debug Output | Analyze Code Coverage

Plugins

DebugKit

Tests

Core

Tests

CakePHP: the rapid development php framework

PHPUnit: 3.7.21

CakePHP Test Suite 2.5.3

App

Tests

Running FeatureTest

2/2 test methods complete: 2 passes, 0 fails, 2 assertions and 0 exceptions.

Time: 0.095125913619995 seconds

Peak memory: 5,199,592 bytes

Run more tests | Show Passes | Enable Debug Output | Analyze Code Coverage

Plugins

DebugKit

Tests

Core

Tests

CakePHP: the rapid development php framework

PHPUnit: 3.7.21

CakePHP Test Suite 2.5.3

App

Tests

Running ManualdatalogTest

2/2 test methods complete: 2 passes, 0 fails, 2 assertions and 0 exceptions.

Time: 0.079515933990479 seconds

Peak memory: 5,345,368 bytes

Run more tests | Show Passes | Enable Debug Output | Analyze Code Coverage

Plugins

DebugKit

Tests

Core

Tests

## Pruebas unitarias Sensors Controller:

Con el fin de mostrar el correcto funcionamiento de cada uno de los métodos que integran el controlador sensors, se crearon una serie de pruebas o test.

Como primer método de demostración se creó el método **testIndex()**, este método da como resultado todos los sensores que en el momento actual estan en la base de datos.

El siguiente fragmento de código demuestra cómo se implementó el método **testIndex()**:

```
/**
 * testIndex method
 *
 * @return void
 */
public function testIndex() {

    $this->testAction('/sensors/index');

    $this->assertInternalType('array', $this->vars['sensors']);

}
```

El código selecciona cada uno de los sensores actuales de la base de datos, posteriormente verifica que lo que la línea anterior seleccionó sea efectivamente lo que hay en la base, utilizando el método **assertInternalType**. Este método no recibe ningún parámetro, lo que hace es que lo verifica contra la variable 'sensors'.

Posteriormente al método **testIndex()**, se crea el método **testView()**, el cual verifica la correctitud del método **view()** de la clase **SensorsController**, este método permite ver los detalles de algun sensor. Igualmente, se puede ver en la siguiente figura la implementación del método **testView()**.

```
/**
 * testView method
 *
 * @return void
 */
public function testView() {

    $this->testAction('/sensors/view/1');

    $this->assertInternalType('array', $this->vars['sensor']);

}
```

Como se puede ver, selecciona, para este caso de prueba se selecciona el sensor con id 1, una vez que se solicita la tupla con este id, verifica este valor con el método **assertInternalType()**.

El método **testAdd()**, verifica el correcto funcionamiento del método add de la clase SensorsController, este método como su nombre lo indica realiza la inserción de una nueva tupla a la base de datos. Para este metodo fue necesario crear un vector de prueba que funciona como la tupa nueva a insertar, que es lo que muestra la siguiente figura.

```
* @return void
*/
public function testAdd() {

    $expected = array(
        'Sensor' => array(
            'id' => '1',
            'serial' => 'rj45',
            'type_' => 'Temperatura',
            'model_' => 'M-SRT',
            'station_id' => null
        ),
        'Feature' => array(
            (int) 0 => array(
                'id' => '1',
                'name' => 'Temperatura',
                'sensor_id' => '1',
                'ID_FEATURE' => '1'
            )
        ),
        'Manualdatalog' => array(
            (int) 0 => array(
                'id' => '1',
                'data_type_id' => '1',
                'recolection_date' => '2015-05-25',
                'data_' => '3',
                'sensor_id' => '1',
                'datalog' => '45',
                'station_id' => '1',
                'ID_MANUALDATALOGS' => '12'
            )
        )
    );
};
```

Una vez creado este vector a insertar, se invoca la función que testAction que es la que invoca el método a verificar, es este caso el método add, de la clase SensorsController. Igualmente como en los casos anteriores utilizamos la función assertEquals que recibe el resultado de la consulta comparándolo con lo que debería este metodo recibir.

```

        $insercion = array(
            'Sensor' => array(
                'id' => '1',
                'serial' => 'rj45',
                'price' => '8978',
                'type_' => 'Temperatura',
                'model_' => 'H-SRT',
                'installation_date' => '2015-05-12',
                'removal_date' => '2015-05-31',
                'calibration_date' => '2015-05-29',
                'brand' => 'campbell',
                'description' => 'campbell',
                'provider' => 'campbell case',
                'coordinate_x' => null,
                'coordinate_y' => null,
                'station_id' => null,
                'ID_SENSOR' => '23'
            )
        );

        $this->testAction('/sensors/add', array('data' => $insercion, 'method' => 'post'));
        $result = $this->Sensor->getSensorsFiltrados(1);
        $this->assertEquals($expected, $result);
    }
}

```

Ademas, como parte de la prueba de los métodos, se crea la clase StationsControllerTest, que como la clase que corresponde a Sensor, esta clase contiene los métodos que verifican el correcto funcionamiento de los métodos que utiliza la aplicacion para crear Stations. Así mismo para probar cada uno de las clases y sus respectivos métodos también se crean los métodos test, que prueban el correcto funcionamientos de cada implementación, esto para las clases DataTypes, FeaturesController y ManualDatalogs.

Los métodos también implementados son lo que corresponde al index, y al view. La siguiente imagen muestra parte de la implementación de cada uno de estos métodos que son de la clase DataTypeControllerTest.

```

/**
 * testIndex method
 *
 * @return void
 */
public function testIndex() {
    $this->testAction('/dataTypes/index');
    $this->assertInternalType('array', $this->vars['dataTypes']);
}

/**
 * testView method
 *
 * @return void
 */
public function testView() {
    $this->testAction('/dataTypes/view/1');
    $this->assertInternalType('array', $this->vars['dataType']);
}
}

```

La siguiente figura muestra la salida correspondiente al seleccionar los test de la clase Controller / SensorController.

- App
  - [Tests](#)
- Plugins
  - DebugKit
    - [Tests](#)
- Core
  - [Tests](#)

### App Test Cases:

- [AllTests](#)
- [Controller / DataTypesController](#)
- [Controller / FeaturesController](#)
- [Controller / ManualdatalogsController](#)
- [Controller / SensorsController](#)
- [Controller / StationsController](#)
- [Model / DataType](#)
- [Model / Feature](#)
- [Model / Manualdatalog](#)
- [Model / Sensor](#)
- [Model / Station](#)

El resultado verifica que todos los métodos que se probaron efectivamente hacen su trabajo de manera inequívoca. La siguiente figura constata que esto esté ocurriendo.

### IP Test Suite 2.5.3

<a href="#">ts</a>	Running SensorsControllerTest
;	<b>3/3 test methods complete: 3 passes, 0 fails, 3 assertions and 0 exceptions.</b>
ugKit	Time: 0.146138191223 seconds
<a href="#">ests</a>	Peak memory: 9,085,536 bytes
<a href="#">ts</a>	<a href="#">Run more tests</a>   <a href="#">Show Passes</a>   <a href="#">Enable Debug Output</a>   <a href="#">Analyze Code Coverage</a>

---

## =>Resultado de correr todas las pruebas que se realizaron en este Sprint:

The screenshot shows a web browser window with the URL `localhost:82/cakephp/app/webroot/test.php?case=AllTests`. The browser's address bar and tabs are visible at the top. Below the browser window, the CakePHP Test Suite 2.5.3 results are displayed. On the left, a tree view shows the test categories: App (Tests), Plugins (DebugKit, Tests), and Core (Tests). The main area lists the tests being run: Running All tests, Running DataTypesControllerTest, Running FeaturesControllerTest, Running ManualdatalogsControllerTest, Running SensorsControllerTest, Running StationsControllerTest, Running DataTypeTest, Running FeatureTest, Running ManualdatalogTest, Running SensorTest, and Running StationTest. A green bar indicates that 20/20 test methods were complete, with 20 passes, 0 fails, 20 assertions, and 0 exceptions. Below this, the execution time is 8.8340861797333 seconds and the peak memory is 9,820,368 bytes. At the bottom, there are links to run more tests, show passes, enable debug output, and analyze code coverage. A footer bar at the very bottom shows the status 'CHECKED' and 'POWER' on the left, and '(test) 336 queries took 6488 ms' on the right.

CakePHP Test Suite 2.5.3

- App
  - [Tests](#)
- Plugins
  - DebugKit
  - [Tests](#)
- Core
  - [Tests](#)

Running All tests  
Running DataTypesControllerTest  
Running FeaturesControllerTest  
Running ManualdatalogsControllerTest  
Running SensorsControllerTest  
Running StationsControllerTest  
Running DataTypeTest  
Running FeatureTest  
Running ManualdatalogTest  
Running SensorTest  
Running StationTest

20/20 test methods complete: 20 passes, 0 fails, 20 assertions and 0 exceptions.

Time: 8.8340861797333 seconds  
Peak memory: 9,820,368 bytes

[Run more tests](#) | [Show Passes](#) | [Enable Debug Output](#) | [Analyze Code Coverage](#)

CHECKED POWER

(test) 336 queries took 6488 ms