

DP2-G3-14

A+ Realizados

<https://github.com/fersolesp/DP2-G3-14>

<https://travis-ci.org/github/fersolesp/DP2-G3-14>

<https://sonarcloud.io/dashboard?id=reyblacua> DP2-G3-14

Reyes Blasco Cuadrado

Pablo Cardenal Gamito

José María Cornac Fisas

Vanessa Pradas Fernández

Fernando Luis Sola Espinosa

Contenido

| | |
|---|---|
| 2. Sprint 2, entregable 1: “AssertThat” personalizados | 3 |
| 3. Sprint 3, entregable 2: Tests de IU con Cucumber | 5 |
| 4. Sprint 4, entregable 3: Uso de “feeders” para las pruebas de rendimiento | 9 |

2. Sprint 2, entregable 1: “AssertThat” personalizados

Para este “A+”, nos disponemos a implementar aserciones personalizadas para clases concretas de nuestro modelo, lo cual, nos permitirá crear comprobaciones complejas que puedan ser reutilizadas en lugar de tener repetidas veces una sentencia “Assertions.assert...” que muchas veces será larga, y de esta forma, también facilitamos la lectura y comprensión de la misma. Veámoslo con un ejemplo:

```
1 package org.springframework.samples.petclinic.util;
2
3 import java.util.Objects;
4
5
6
7
8
9 public class InscriptionAssert extends AbstractAssert<InscriptionAssert, Inscription> {
10
11     public InscriptionAssert(final Inscription actual) {
12         super(actual, InscriptionAssert.class);
13     }
14
15     public static InscriptionAssert assertThat(final Inscription actual) {
16         return new InscriptionAssert(actual);
17     }
18
19     public InscriptionAssert hasName(final String name) {
20         this.isNotNull();
21
22         if (!Objects.equals(this.actual.getName(), name)) {
23             this.failWithMessage("Expected inscription's name to be <%s> but was <%s>", name, this.actual.getName());
24         }
25
26         return this;
27     }
28
29     public InscriptionAssert idNotNull() {
30         this.isNotNull();
31
32         if (this.actual.getId()==null) {
33             this.failWithMessage("Inscription id is null");
34         }
35         return this;
36     }
37
38 }
39
40
```

Tenemos, en este caso, una clase “InscriptionAssert”, en la que vamos a definir varias aserciones personalizadas para la clase “Inscription”. En primer lugar, debe extender a “AbstractAssert”, la clase que nos dotará de las herramientas necesarias para realizar esta tarea sencillamente. Creamos también un constructor de la clase, y un método estático “assertThat” que es al que llamaremos para ejecutar la aserción. Ahora, definimos varios métodos correspondientes con cada una de las aserciones que queramos crear. Por ejemplo, tenemos el método “hasName” que, en primer lugar, comprueba que la “Inscription” a la que se aplica no es nula y, a continuación, compara el nombre de esta con el nombre que pasamos por parámetro. También, definimos el mensaje de error que lanzará la aserción al fallar.

Una vez hecho esto, simplemente tenemos que ir a nuestras pruebas, en este caso, por ejemplo, a las de servicio, y hacer uso de la aserción como si se tratara de otra cualquiera:

```
@ParameterizedTest
@CsvSource({
    "1,Inscription1", "2,Inscription2", "3,Inscription3"
})
void shouldFindInscriptionWithCorrectId(final int id, final String nameInscription) {
    Optional<Inscription> inscription = this.inscriptionsService.findInscriptionById(id);
    org.assertj.core.api.Assertions.assertThat(inscription).isPresent();

    //Assert personalizado
    InscriptionAssert.assertThat(inscription.get()).hasName(nameInscription);
}
```

Pensamos que estos “asserts” personalizados son una herramienta muy potente, que, aunque nosotros los hemos usado de forma básica para conocer su existencia, ofrecen la posibilidad de crear aserciones muy complejas como si se trataran de un método corriente y poder reutilizar su llamada ahorrándonos código y posibles fallos de definición de la comprobación, así como aumentar la legibilidad de nuestras pruebas.

3. Sprint 3, entregable 2: Tests de IU con “Cucumber”

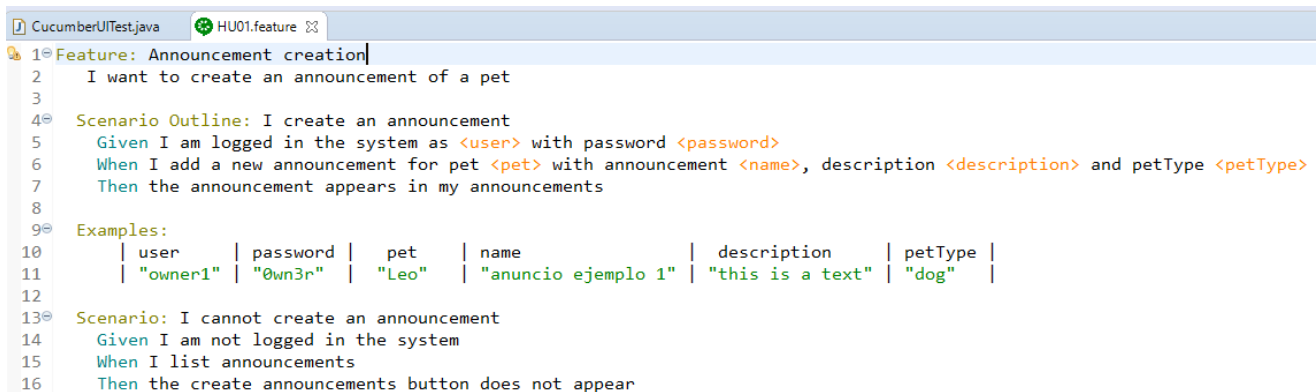
Para este A+ decidimos hacer parte de las pruebas de interfaz de usuario (IU) de nuestro sistema con “Cucumber”. Concretamente, se realizaron las de las áreas de peluquería y de anuncios y adopción.

El uso de “Cucumber” se basa en la definición de pruebas de IU a partir de los escenarios definidos para cada historia de usuario de nuestro sistema, de forma que se vea claramente, para cada escenario, cuál es la interacción que haría el usuario (en este caso la ejecuta el ordenador) con nuestro sistema, conectando de forma directa y muy visual aquello que tenemos sobre papel y que define nuestro sistema (las historias y sus escenarios) con su simulación en forma de prueba.

Para ello, en primer lugar, debemos disponer de una clase “CucumberUITest.java” que, por así decirlo, agrupa todos los “tests” de “Cucumber” que tengamos en nuestro proyecto y permite que definamos parámetros generales para ellos:

```
1 package org.springframework.samples.petclinic.bdd;
2
3 import io.cucumber.junit.Cucumber;
4
5 @RunWith(Cucumber.class)
6 @CucumberOptions(
7     features= {"src/test/java/"},
8     tags = {"not @ignore"},
9     plugin = {"pretty",
10         "json:target/cucumber-reports/cucumber-report.json"},
11     monochrome=true)
12 public class CucumberUITest {
13 }
14 }
```

Una vez hecho esto, nos disponemos a definir, para cada historia de usuario, un archivo “.feature” donde definimos la historia y los escenarios que la componen:



```
1 Feature: Announcement creation
2   I want to create an announcement of a pet
3
4 Scenario Outline: I create an announcement
5   Given I am logged in the system as <user> with password <password>
6   When I add a new announcement for pet <pet> with announcement <name>, description <description> and petType <petType>
7   Then the announcement appears in my announcements
8
9 Examples:
10  | user | password | pet | name | description | petType |
11  | "owner1" | "0wn3r" | "Leo" | "anuncio ejemplo 1" | "this is a text" | "dog" |
12
13 Scenario: I cannot create an announcement
14   Given I am not logged in the system
15   When I list announcements
16   Then the create announcements button does not appear
```

En este caso, por ejemplo, tenemos la primera historia de usuario y dos escenarios para la misma. En el primer escenario, hemos decidido usar “Scenario Outline” de forma que se parametriza y podemos hacer que se ejecute varias veces para distintos valores definidos en la tabla “Examples”. En este caso, tenemos un único ejemplo en la tabla, puesto que, aunque teníamos más, tuvimos que eliminarlos debido a que las pruebas de IU (incluidas las de “Cucumber”) se ejecutan todas una tras otra modificando el estado de la BD y dependen, por tanto, del resultado de la anterior. Por ello, no pudimos, finalmente ejecutar varios ejemplos en el primer escenario.

El segundo escenario, sin embargo, no está parametrizado, si no que se ejecuta con los valores que tenga definidos en los “steps” o pasos para su ejecución. Estos pasos, son la definición en código de la interacción que se realiza simulando a un usuario real y se corresponden con las frases definidas en los escenarios del archivo “.feature”. Por ejemplo, para la sentencia “When” del primer escenario en el que añadimos un “announcement”, tenemos definido el siguiente método en el archivo “HU01StepsDefinitions”:

```
@Log
public class HU01StepDefinitions extends AbstractStep {

    @LocalServerPort
    private int port;

    @When("I add a new announcement for pet {string} with announcement {string}, description {string} and petType {string}")
    public void iAddANewAnnouncement(final String pet, final String name, final String description, final String type) {
        this.getDriver().get("http://localhost:" + this.port);
        this.getDriver().findElement(By.cssSelector("li:nth-child(3) span:nth-child(2)")).click();
        this.getDriver().findElement(By.linkText("Add New Announcement")).click();
        this.getDriver().findElement(By.id("name")).click();
        this.getDriver().findElement(By.id("name")).sendKeys(name);
        this.getDriver().findElement(By.id("petName")).click();
        this.getDriver().findElement(By.id("petName")).sendKeys(pet);
        this.getDriver().findElement(By.id("description")).click();
        this.getDriver().findElement(By.id("description")).sendKeys(description);
        {
            WebElement dropdown = this.getDriver().findElement(By.id("type"));
            dropdown.findElement(By.xpath("//option[. = '" + type + "']")).click();
        }
        this.getDriver().findElement(By.cssSelector("option:nth-child(3)")).click();
        this.getDriver().findElement(By.cssSelector(".btn-default")).click();
    }
}
```

Vemos claramente, cómo definimos la interacción del “web driver” con nuestra aplicación para ese paso del escenario y cómo, en este caso, que está parametrizado, recogemos los valores de los parámetros indicados en el “.feature” y los usamos aquí como valores que enviamos en el formulario de creación del “announcement”. Debemos remarcar también el uso parametrizado del puerto en que funciona nuestra aplicación, ya que lo hemos diseñado de forma que cada vez que se lancen las pruebas de IU, lo haga en un puerto aleatorio y este, lo recogemos en la variable “port” para poderse lo indicar al “web driver”.

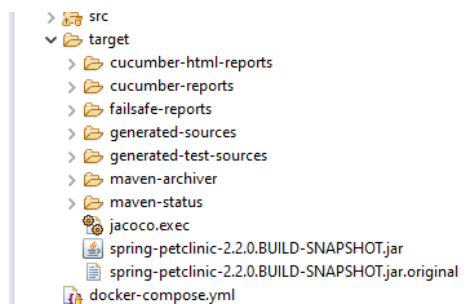
Por otro lado, de forma general, tenemos las aserciones en los pasos “Then”, de forma que comprobamos que la interacción que hemos llevado a cabo con el sistema ha sido exitosa. Por ejemplo, aquí comprobamos que existen cuatro “announcements” al crear uno nuevo, ya que al principio del escenario había únicamente tres:

```
@Then("the announcement appears in my announcements")
public void theNewAnnouncementAppears() {
    this.getDriver().get("http://localhost:" + this.port);
    this.getDriver().findElement(By.cssSelector("li:nth-child(3) span:nth-child(2)")).click();
    int announcementsNumber = this.getDriver().findElements(By.xpath("//table[@id='announcementsTable']/tbody/tr")).size();
    Assert.assertEquals(announcementsNumber, 4);
    this.stopDriver();
}
}
```

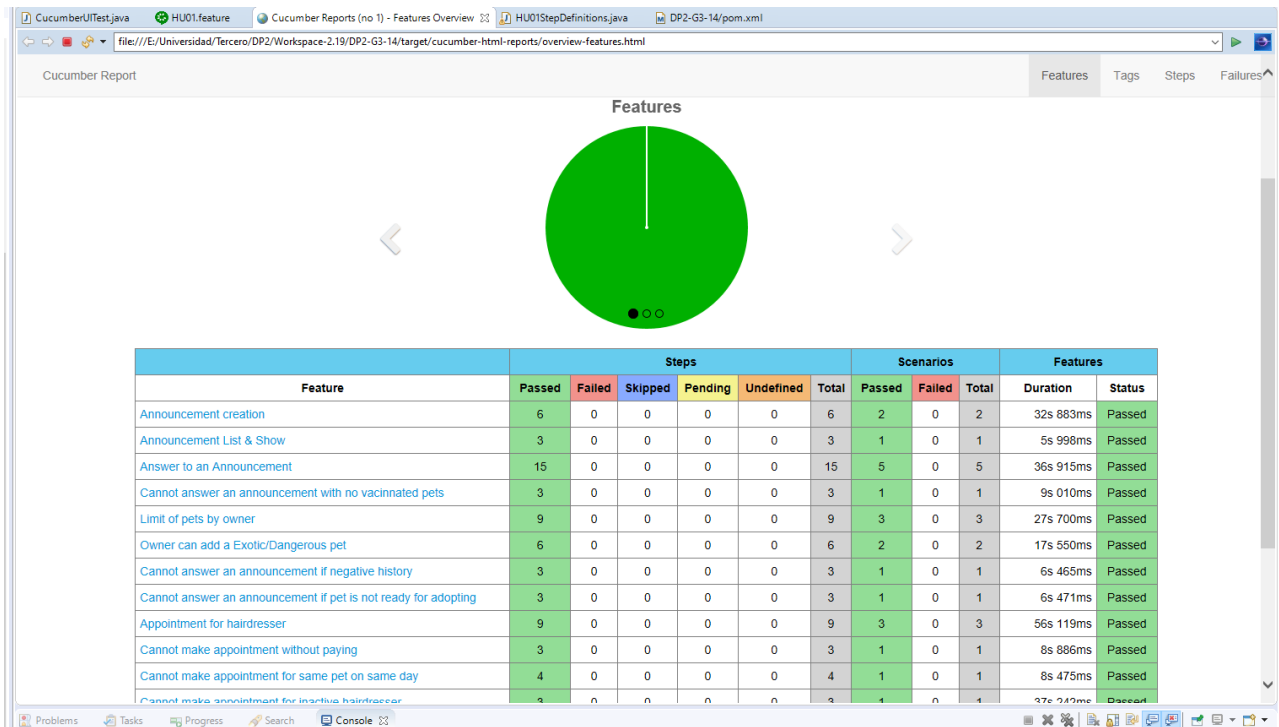
“Cucumber”, además, nos ofrece la posibilidad de generar “reports” donde analiza los resultados obtenidos en la ejecución de sus pruebas. Para ello hemos definido un “profile” de “Maven” donde le indiquemos que queremos que genere estos “reports”. Aquí tenemos un fragmento:

```
<configuration>
  <includes>
    <include>**/CucumberUITest.java</include>
  </includes>
</configuration>
</plugin>
<plugin>
  <groupId>net.masterthought</groupId>
  <artifactId>maven-cucumber-reporting</artifactId>
  <version>3.8.0</version>
  <executions>
    <execution>
      <id>post-integration-test</id>
      <goals>
        <goal>generate</goal>
      </goals>
      <configuration>
        <projectName>${name}</projectName>
        <outputDirectory>target</outputDirectory>
        <cucumberOutput>target/cucumber-reports/cucumber-report.json</cucumberOutput>
        <buildNumber>1</buildNumber>
        <parallelTesting>false</parallelTesting>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Este análisis se genera en la carpeta “target” de nuestro proyecto:



Y ofrece archivos “html” con gráficas interactivas y multitud de datos interesantes:



Pensamos que el sobreesfuerzo que supone implementar las pruebas de IU en “Cucumber” respecto a hacerlas sin él, es ínfimo comparado con la ventaja que supone ver nombrada cada prueba, con el nombre de su escenario, cómo se ejecuta según cada historia de usuario y, en definitiva, cómo materializa aquello tan abstracto del papel en una interacción simulada con nuestro sistema.

4. Sprint 4, entregable 3: Uso de “feeders” para las pruebas de rendimiento

Se ha procedido a estudiar el uso de “feeders” en “Gatling”, generando datos con “csv-generator” y usando estos datos para las peticiones “post” que realiza “Gatling” en la “HU01” al crear un anuncio. De esta forma, logramos simular una interacción aún más realista en las pruebas de rendimiento con nuestro sistema, de forma que se creen “announcements” con los valores definidos en el archivo “csv”.

Por otro lado, con la finalidad de comprobar con un que todas las peticiones hechas al sistema en la prueba de rendimiento son correctas, hacemos uso, también, del siguiente “assert”:

```
setUp(scen).assertions(forAll.failedRequests.percent.lte(0))
```

A continuación, se presentan los últimos valores del “csv”, mostrando que vamos a utilizar 1000 anuncios (el “csv” llega hasta 1001 porque la primera línea corresponde a los nombres de cada columna de valores).

| | A | B | C | D | E | F | G | H |
|------|---|---|---|---|---|---|---|---|
| 984 | 987, Da Nang, Ashlee, "Korea, Democratic People's Republic of", True, snake | | | | | | | |
| 985 | 988, Kuching, Molli, United Kingdom, False, cat | | | | | | | |
| 986 | 989, Indianapolis, Jinny, Palau, False, dog | | | | | | | |
| 987 | 990, San Fernando, Fawne, Saint Lucia, False, bird | | | | | | | |
| 988 | 991, City of San Marino, Pollyanna, Cuba, True, bird | | | | | | | |
| 989 | 992, Makassar, Claresta, Costa Rica, True, cat | | | | | | | |
| 990 | 993, Port Hedland, Kimmy, Namibia, False, hamster | | | | | | | |
| 991 | 994, Gaborone, Amii, Macao, False, lizard | | | | | | | |
| 992 | 995, Rockhampton, Robinia, French Guiana, False, hamster | | | | | | | |
| 993 | 996, Faisalabad, Tersina, Bermuda, True, snake | | | | | | | |
| 994 | 997, Jakarta, Verla, Colombia, True, snake | | | | | | | |
| 995 | 998, Oslo, Dorothy, Myanmar, True, lizard | | | | | | | |
| 996 | 999, La Serena, Bibby, Niue, True, hamster | | | | | | | |
| 997 | 1000, Brasília, Chickie, Yemen, True, cat | | | | | | | |
| 998 | 1001, Konya, Coral, Malta, False, lizard | | | | | | | |
| 999 | 1002, Labasa, Kirbee, "Micronesia, Federated States of", False, lizard | | | | | | | |
| 1000 | 1003, Managua, Teriann, Svalbard and Jan Mayen, True, cat | | | | | | | |
| 1001 | 1004, Yakutsk, Ermengarde, Norway, False, lizard | | | | | | | |
| 1002 | | | | | | | | |
| 1003 | | | | | | | | |
| 1004 | | | | | | | | |

Inicializamos el atributo “csvFeeder”, que poseerá todos los valores anteriores:

```
val csvFeeder = csv("announcements.csv").circular
```

Hemos usado la estrategia “.circular” que consiste en usar los valores del archivo “csv” desde el primero hasta el último y cuando se llegue a este, se comienza de nuevo por el principio.

A la hora de ejecutar el “post” de creación de los “announcements”, usamos la orden “.feed(csvFeeder)”, que nos permite usar los valores almacenados en nuestro archivo “csv” y que previamente habíamos cargado en “csvFeeder”, y en cada “.formParam”, para el valor del campo usamos el nombre de la columna de nuestro archivo “csv” que le corresponda de la forma:

\${columna_del_archivo_csv}

```
object FormAnnouncements {
  val formAnnouncements = exec(http("FormAnnouncements")
    .get("/announcements/new")
    .headers(headers_0)
    .check(css("input[name=_csrf]", "value").saveAs("token")))
    .pause(10)
    .feed(csvFeeder)
    .exec(http("AnnouncementCreated")
      .post("/announcements/new")
      .headers(headers_2)
      .formParam("name", "${name}")
      .formParam("petName", "${petName}")
      .formParam("description", "${description}")
      .formParam("type", "${type}")
      .formParam("canBeAdopted", "${canBeAdopted}")
      .formParam("_csrf", "${token}"))
    .pause(15)
}
```

Por último, introducimos la siguiente configuración del escenario de simulación (última línea de código del setUp):

```
setUp(
  scn.inject(rampUsers(1000) during (10 seconds))
).protocols(httpProtocol)
.assertions(
  global.responseTime.max.lt(6000),
  global.responseTime.mean.lt(1000),
  global.successfulRequests.percent.gt(95),
  forAll.failedRequests.percent.lte(0)
)
```

Podemos observar que los resultados que obtenemos en la consola de “Gatling” son exitosos.

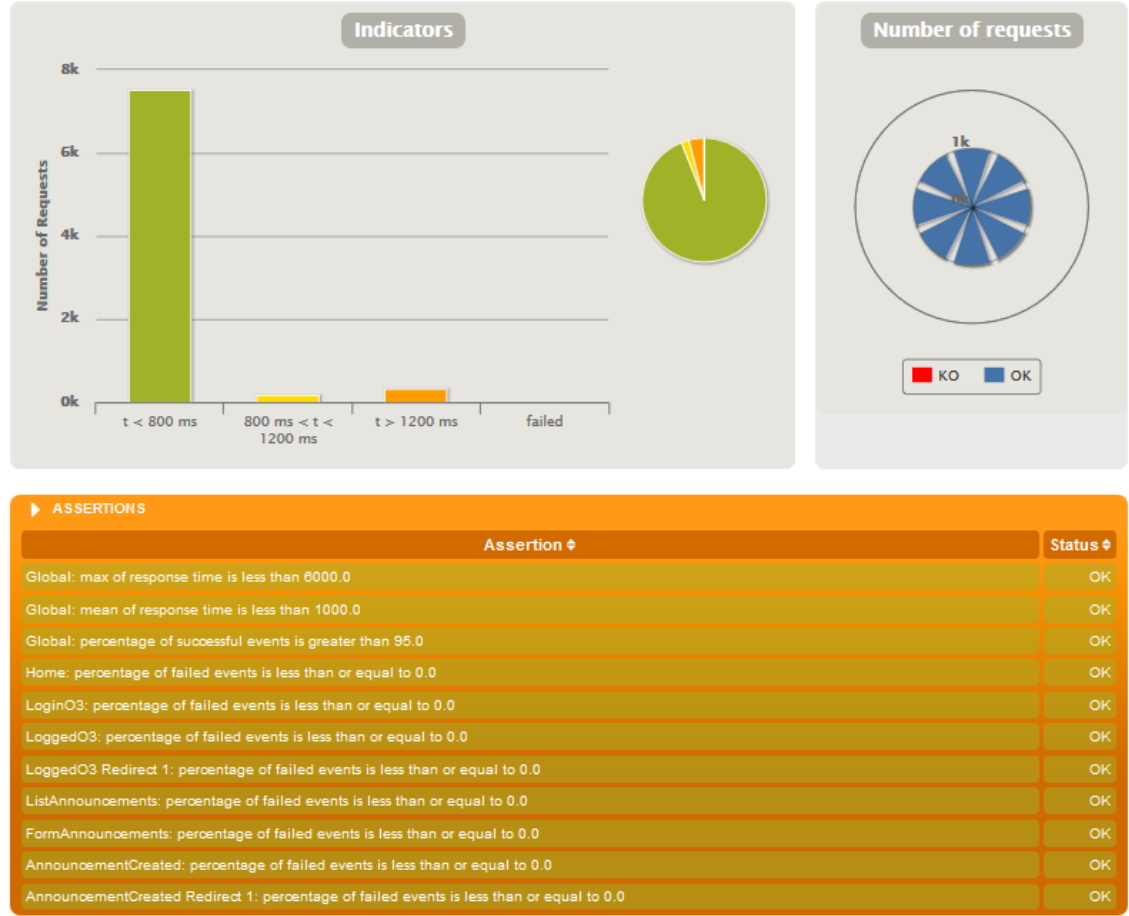
```
=====
2020-05-29 19:21:32                                     70s elapsed
---- Requests -----
> Global (OK=8000 KO=0 )
> Home (OK=1000 KO=0 )
> Login03 (OK=1000 KO=0 )
> Logged03 (OK=1000 KO=0 )
> Logged03 Redirect 1 (OK=1000 KO=0 )
> ListAnnouncements (OK=1000 KO=0 )
> FormAnnouncements (OK=1000 KO=0 )
> AnnouncementCreated (OK=1000 KO=0 )
> AnnouncementCreated Redirect 1 (OK=1000 KO=0 )

---- Registered -----
[#####]100%
      waiting: 0 / active: 0 / done: 1000
=====

Simulation dp2.HU01Test completed in 70 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...
```

Por otro lado, accedemos a los gráficos de los resultados y se observa que son bastante satisfactorios.

> **Global Information**



Y, por último, observamos cómo se han añadido correctamente los 1000 anuncios en la base de datos.

```
1 • SELECT * FROM petclinic.announcement;
```

| | id | name | can_be_adopted | description | pet_name | owner_id | type_id |
|---|------|--------------|----------------|---------------|-----------|----------|---------|
| | 978 | Naypyidaw | 0 | Kuwait | Bobinette | 3 | 4 |
| | 979 | Geneva | 0 | Sudan | Marnia | 3 | 3 |
| | 980 | Ankara | 1 | Colombia | Daryl | 3 | 2 |
| | 981 | Shenyang | 0 | Niger | Dulce | 3 | 5 |
| | 982 | Gothenb... | 0 | Haiti | Gerrie | 3 | 2 |
| | 983 | Bulawayo | 0 | Christmas... | Deloria | 3 | 1 |
| | 984 | Lubumb... | 1 | Turks and... | Cissiee | 3 | 5 |
| | 985 | Da Nang | 1 | Korea, De... | Ashlee | 3 | 4 |
| | 986 | Indiana... | 0 | Palau | Jinny | 3 | 2 |
| | 987 | San Fer... | 0 | Saint Lucia | Fawne | 3 | 5 |
| | 988 | Kuching | 0 | United Kin... | Molli | 3 | 1 |
| | 989 | Port He... | 0 | Namibia | Kimmy | 3 | 6 |
| | 990 | Makassar | 1 | Costa Rica | Claresta | 3 | 1 |
| | 991 | City of S... | 1 | Cuba | Pollyanna | 3 | 5 |
| | 992 | Gaborone | 0 | Macao | Amii | 3 | 3 |
| | 993 | Faisalabad | 1 | Bermuda | Tersina | 3 | 4 |
| | 994 | Rockha... | 0 | French Gu... | Robinia | 3 | 6 |
| | 995 | Jakarta | 1 | Colombia | Verla | 3 | 4 |
| | 996 | Oslo | 1 | Myanmar | Dorothy | 3 | 3 |
| | 997 | La Serena | 1 | Niue | Bibby | 3 | 6 |
| | 998 | Brasilia | 1 | Yemen | Chickie | 3 | 1 |
| | 999 | Konya | 0 | Malta | Coral | 3 | 3 |
| | 1000 | Labasa | 0 | Micronesi... | Kirbee | 3 | 3 |
| * | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

Por último, debemos decir que elegimos la HU01 de creación de “announcements” porque era la que nos permitía hacer uso de “feeders” con mayor facilidad, ya que un único “owner” puede crear todos los anuncios que quiera, pero si fuera, por ejemplo, coger cita para su mascota para el peluquero, sólo puede coger una, sin pagar las anteriores, por lo que tendríamos que haber generado también 1000 “owners”, hacer que se registraran, generar una mascota para cada uno y, finalmente, coger cita para la peluquería para ellas.

Nuestra intención era la de ilustrar el uso y la utilidad de los “feeders” para darles realismo a las pruebas de rendimiento de forma sencilla.