

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática


Documentación del proyecto PetClinic Performance



Grado en Ingeniería Informática – Ingeniería del Software
Diseño y Pruebas II
Curso 2019 – 2020


Fecha	Versión
03/06/2020	v3r00

Grupo de prácticas	G2-15 B.F.C.N
Autores	Rol
Franco Sánchez, Pablo	Desarrollador
Gutiérrez Prieto, Gabriel	Desarrollador
Lopez, Thibaut	Desarrollador
Rojas Gutiérrez, Rodrigo	Desarrollador
Romero Cáceres, Antonio	Desarrollador
Vidal Pérez, Antonio	Desarrollador

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones


Control de Versiones

Fecha	Versión	Descripción
29/05/2020	v1r00	Elaboración del documento.
30/05/2020	v1r10	Actualización.
01/06/2020	v2r00	Actualización.
03/06/2020	v3r00	Actualización final.

	<p>Diseño y Pruebas II Documentación del Proyecto PetClinic</p>
	<p>Control de Versiones</p>

Índice de contenido

1. Introducción	4
2. Pruebas de rendimiento de Enfermedad	4
3. Test de rendimiento Salas	19
4. Test de rendimiento Booking (Reserva sala).....	30
5. Tests de rendimiento de Opinion	36
6. Tests de rendimiento de Chips	40

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

1. Introducción

En este documento se recogen todas las pruebas de rendimiento realizadas en el proyecto PetClinic del grupo G2-15. Las pruebas de rendimiento del proyecto han sido repartidas a los distintos miembros del equipo, los cuales han ido creando varios escenarios.

2. Pruebas de rendimiento de Enfermedad

Lo primero que he realizado es la grabación generando el código necesario para ejecutar los tests.

Después he procedido a cambiar el código para que sea entendible a la hora de ejecutarlo poniéndole nombres en vez de dejar el que tenían (**request_X**).

He procedido a crear 2 escenarios:

- **CreateDiseaseScn**
- **DeleteDiseaseScn**

Una vez creados estos escenarios, se hacen las llamadas a sus respectivas peticiones. Probando pude ver que el borrar iba más rápido que el crear, y que a su vez solo borraba un elemento de la tabla.

Para hacer el borrar más lento, decidí crear un **HomeDelete** que mantiene en espera al usuario que quiera entrar, y aumente los tiempos de pausa de **listar las enfermedades**.

Hice la prueba y el borrar ya no iba por delante del crear.

El segundo problema lo intenté de varias formas, pero solo encontré 2 soluciones, o bien crear un fichero csv y leer de él, o repetir el bloque del borrado e ir pasando el número de la repetición en la que se encuentra en el get de su url.

Al repetir el bloque noté 2 cosas:

- Que si metíamos más de 1 usuario este repetiría el bloque completo para cada uno de ellos, haciendo que solo un usuario borrara y el otro no hiciese nada.
- Que el tiempo global máximo aumentaría demasiado, aquí no podríamos hacer mucho más.



El código resultante que tendríamos sería el siguiente:

```
class TestPerformanceCreateAndDelete extends Simulation {

    val httpProtocol = http
        .baseUrl("http://www.dp2.com")
        .inferHtmlResources(BlackList("""*.css""", """*.js""", """*.ico""", """*.png"""), WhiteList())
        .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9")
        .acceptEncodingHeader("gzip, deflate")
        .acceptLanguageHeader("es-ES,es;q=0.9")
        .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36")

    val headers_0 = Map(
        "Proxy-Connection" -> "keep-alive",
        "Upgrade-Insecure-Requests" -> "1")

    val headers_2 = Map(
        "Accept" -> "image/webp,image/apng,image/*,*/*;q=0.8",
        "Proxy-Connection" -> "keep-alive")

    val headers_3 = Map(
        "Origin" -> "http://www.dp2.com",
        "Proxy-Connection" -> "keep-alive",
        "Upgrade-Insecure-Requests" -> "1")

    object Home{
        val home= exec(http("Home")
            .get("/")
            .headers(headers_0))
        .pause(9)
    }

    object HomeDelete{
        val home= exec(http("Home")
            .get("/")
            .headers(headers_0))
        .pause(36)
    }
}
```



```
object Login{
    val login= exec(
        http("Login")
            .get("/login")
            .headers(headers_0)
            .resources(http("request_2")
                .get("/login")
                .headers(headers_2))
            .check(css("input[name=_csrf]", "value").saveAs("stoken")))
    ).pause(13)
    .exec(
        http("Logged")
            .post("/login")
            .headers(headers_3)
            .formParam("username", "vet1")
            .formParam("password", "v3t")
            .formParam("_csrf", "${stoken}")
    ).pause(10)
}

object FindOwners{
    val findOwners=exec(http("FindOwners")
        .get("/owners/find")
        .headers(headers_0))
    .pause(10)
}

object ListOwners{
    val listOwners=exec(http("ListOwners")
        .get("/owners?lastName=")
        .headers(headers_0))
    .pause(11)
}

object ShowOwner1{
    val showOwner1=exec(http("ShowOwner1")
        .get("/owners/1")
        .headers(headers_0))
    .pause(10)
}

object CreateDiseaseFormPetOwner1{
    val createDiseaseFormPetOwner1=exec(http("CreateDiseaseFormPetOwner1")
        .get("/diseases/new/1?diseaseId=")
        .headers(headers_0)
        .check(css("input[name=_csrf]", "value").saveAs("stoken"))))
}
```



```
.pause(28)
    .exec(http("DiseaseCreated")
        .post("/diseases/new/1?diseaseId=")
        .headers(headers_3)
        .formParam("pet_id", "1")
        .formParam("symptoms", "Nueva enfermedad")
        .formParam("severity", "LOW")
        .formParam("cure", "Paracetamol")
        .formParam("_csrf", "${token}"))
    .pause(26)
}

object ListDiseases{
    val listDiseases=exec(http("ListDiseases")
        .get("/diseases/diseasesList")
        .headers(headers_0))
    .pause(46)
}

object DeleteNewDisease{

    val deleteNewDisease=repeat(3000,"id"){
        exec(http("DeleteNewDisease")
            .get("/diseases/delete/${id}")
            .headers(headers_0))
    }
}

}

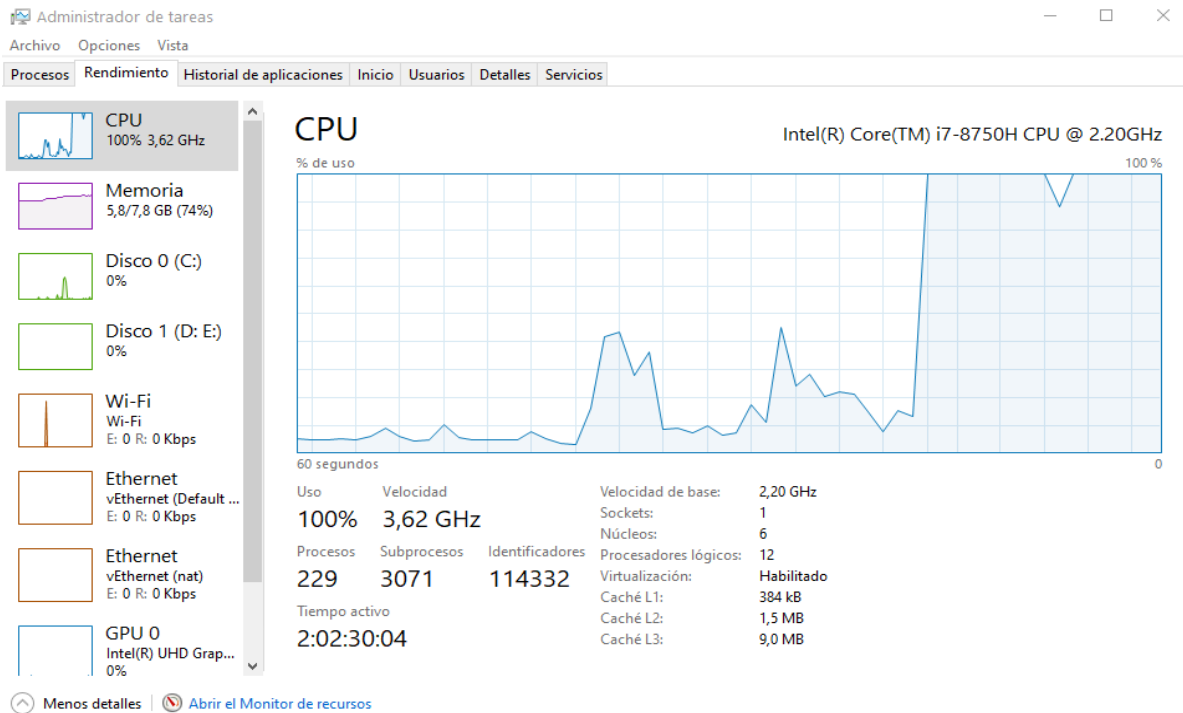
val createDiseaseScn = scenario("TestPerformanceCreateDisease").exec(Home.home,
    Login.login,
    FindOwners.findOwners,
    ListOwners.listOwners,
    ShowOwner1.showOwner1,
    CreateDiseaseFormPetOwner1.createDiseaseFormPetOwner1)

val deleteDiseaseScn = scenario("TestPerformanceDeleteDisease").exec(HomeDelete.home,
    Login.login,
    ListDiseases.listDiseases,
    DeleteNewDisease.deleteNewDisease)

setUp(
    createDiseaseScn.inject(rampUsers(3000) during (100 seconds)),
    deleteDiseaseScn.inject(rampUsers(1) during (100 seconds))
).protocols(httpProtocol)
    .assertions(
        forAll.failedRequests.percent.lte(5),
        global.responseTime.mean.lt(1200),
        global.successfulRequests.percent.gt(97)
    )
}
```

Se puede ver que el tiempo global ha sido sustituido por el máximo de peticiones falladas que no deben superar el 5%.

Ahora mostrare el avance que ha tenido ejecutar este test:



En este momento es cuando nos encontramos ejecutando 1050 usuarios activos.

Ahora veremos cuando ha terminado el crear, pero todavía se siguen borrando usuarios, ya que hay una sola persona realizando esta tarea:

```

=====
020-05-21 19:32:54                                240s elapsed
--- Requests ---
Global (OK=33761 KO=0 )
Home (OK=3001 KO=0 )
Login (OK=3001 KO=0 )
request_2 (OK=3001 KO=0 )
Logged (OK=3001 KO=0 )
Logged Redirect 1 (OK=3001 KO=0 )
FindOwners (OK=3000 KO=0 )
ListOwners (OK=3000 KO=0 )
ShowOwner1 (OK=3000 KO=0 )
ListDiseases (OK=1 KO=0 )
CreateDiseaseFormPetOwner1 (OK=3000 KO=0 )
DiseaseCreated (OK=3000 KO=0 )
DiseaseCreated Redirect 1 (OK=3000 KO=0 )
DeleteNewDisease (OK=378 KO=0 )
DeleteNewDisease Redirect 1 (OK=377 KO=0 )

--- TestPerformanceCreateDisease ---
#####]100%
waiting: 0 / active: 0 / done: 3000
--- TestPerformanceDeleteDisease ---
#####] 0%
waiting: 0 / active: 1 / done: 0
=====

```




Diseño y Pruebas II Documentación del Proyecto PetClinic

Control de Versiones

Vemos que con 3000 usuarios no ha habido ninguna petición fallida, en cambio el encargado de borrar todavía está realizando su tarea. En estos momentos la CPU se ha aliviado un poco.

Pasado un tiempo podemos ver como efectivamente ha cumplido su tarea:


C:\WINDOWS\system32\cmd.exe

```
=====
2020-05-21 19:37:18                               503s elapsed
---- Requests ----
> Global (OK=39006 KO=0 )
> Home (OK=3001 KO=0 )
> Login (OK=3001 KO=0 )
> request_2 (OK=3001 KO=0 )
> Logged (OK=3001 KO=0 )
> Logged Redirect 1 (OK=3001 KO=0 )
> FindOwners (OK=3000 KO=0 )
> ListOwners (OK=3000 KO=0 )
> ShowOwner1 (OK=3000 KO=0 )
> ListDiseases (OK=1 KO=0 )
> CreateDiseaseFormPetOwner1 (OK=3000 KO=0 )
> DiseaseCreated (OK=3000 KO=0 )
> DiseaseCreated Redirect 1 (OK=3000 KO=0 )
> DeleteNewDisease (OK=3000 KO=0 )
> DeleteNewDisease Redirect 1 (OK=3000 KO=0 )

---- TestPerformanceCreateDisease ----
[#####]100%
    waiting: 0 / active: 0 / done: 3000
---- TestPerformanceDeleteDisease ----
[#####]100%
    waiting: 0 / active: 0 / done: 1
=====

Simulation dp2.TestPerformanceCreateAndUpdate completed in 503 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

---- Global Information ----
> request count 39006 (OK=39006 KO=0 )
> min response time 0 (OK=0 KO=- )
> max response time 11696 (OK=11696 KO=- )
> mean response time 1020 (OK=1020 KO=- )
> std deviation 1675 (OK=1675 KO=- )
> response time 50th percentile 67 (OK=67 KO=- )
> response time 75th percentile 1471 (OK=1471 KO=- )
> response time 95th percentile 4959 (OK=4959 KO=- )
> response time 99th percentile 6488 (OK=6488 KO=- )
> mean requests/sec 77.393 (OK=77.393 KO=- )
---- Response Time Distribution ----
> t < 800 ms 26829 ( 69%)
> 800 ms < t < 1200 ms 1484 ( 4%)
> t > 1200 ms 10693 ( 27%)
> failed 0 ( 0%)
=====
```

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

Viendo esto, llegué a pensar si era verdad que había cumplido su tarea y decidí comprobarlo en Docker:

```

docker exec -it 0ddb9bab683e6798b7cdc8c6703912cdd207b029e5022c0ccf9ba8c5862163ef /bin/sh
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 67
Server version: 5.7.8-rc MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.


mysql> select* from diseases;
+-----+-----+-----+-----+-----+
| id | cure | severity | symptoms | pet_id |
+-----+-----+-----+-----+-----+
| 3002 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3004 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3000 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3001 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3003 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3005 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3006 | Paracetamol | LOW | Nueva enfermedad | 1 |
| 3007 | Paracetamol | LOW | Nueva enfermedad | 1 |
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql>

```

Estos “residuos” restantes son debido a que ya había enfermedades en la base de datos antes de que se empezasen a crear, y nosotros empezamos a borrar desde la id 1.

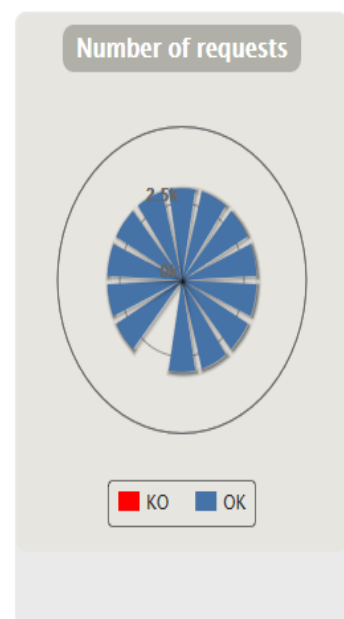
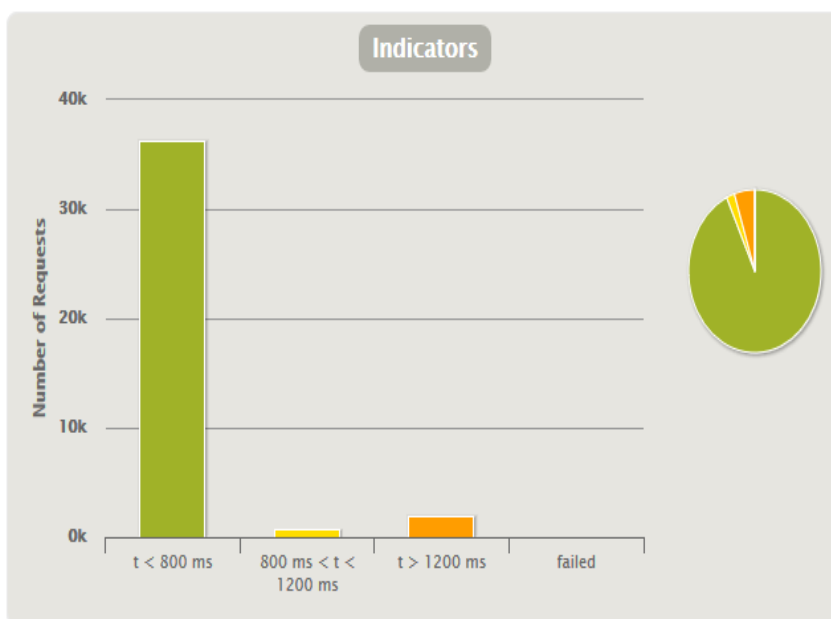
Por lo tanto, ha borrado bien los elementos que se han ido creando.


	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

También podemos apreciar que ha cumplido las restricciones que le puse:

ASSERTIONS	
Assertion	Status
Home: percentage of failed events is less than or equal to 5.0	OK
Login: percentage of failed events is less than or equal to 5.0	OK
request_2: percentage of failed events is less than or equal to 5.0	OK
Logged: percentage of failed events is less than or equal to 5.0	OK
Logged Redirect 1: percentage of failed events is less than or equal to 5.0	OK
FindOwners: percentage of failed events is less than or equal to 5.0	OK
ListOwners: percentage of failed events is less than or equal to 5.0	OK
ShowOwner1: percentage of failed events is less than or equal to 5.0	OK
ListDiseases: percentage of failed events is less than or equal to 5.0	OK
CreateDiseaseFormPetOwner1: percentage of failed events is less than or equal to 5.0	OK
DiseaseCreated: percentage of failed events is less than or equal to 5.0	OK
DiseaseCreated Redirect 1: percentage of failed events is less than or equal to 5.0	OK
DeleteNewDisease: percentage of failed events is less than or equal to 5.0	OK
DeleteNewDisease Redirect 1: percentage of failed events is less than or equal to 5.0	OK
Global: mean of response time is less than 1200.0	OK
Global: percentage of successful events is greater than 97.0	OK

Ahora miraremos los tiempos obtenidos:



	<p style="text-align: center;">Diseño y Pruebas II Documentación del Proyecto PetClinic</p>
	<p style="text-align: center;">Control de Versiones</p>

Vemos que la aplicación se ha ralentizado, pero estos tiempos de aumento se han debido a este tipo de peticiones:

- **Iniciar sesión** cosa que es normal al haber mucho tráfico.
- **Listar y mostrar** los propietarios.
- **Crear** las enfermedades.

Viendo que no ha devuelto ningún fallo la aplicación y el tiempo de retraso no ha sido generado por una única petición si no al revés, por varias peticiones, y estos retrasos no son muy elevadas en cada una de ellas, veo que el rendimiento tanto del crear y borrar es bueno.

A continuación, se ha realizado un máximo de usuarios para este equipo, dando como resultado los fallos que se ven a continuación:

```


> request_2 (OK=3501 KO=0 )
> Logged (OK=3375 KO=126 )
> Logged Redirect 1 (OK=3375 KO=0 )
> FindOwners (OK=3442 KO=0 )
> ListOwners (OK=1842 KO=103 )
> ShowOwner1 (OK=1473 KO=107 )
> FindOwners Redirect 1 (OK=233 KO=0 )
> ListDiseases (OK=1 KO=0 )
> CreateDiseaseFormPetOwner1 (OK=1263 KO=0 )
> ListOwners Redirect 1 (OK=198 KO=0 )
> DiseaseCreated (OK=864 KO=36 )
> DiseaseCreated Redirect 1 (OK=661 KO=12 )
> DeleteNewDisease (OK=2 KO=0 )
> ShowOwner1 Redirect 1 (OK=83 KO=0 )
> DeleteNewDisease Redirect 1 (OK=1 KO=0 )
----- Errors -----
> i.g.h.c.i.RequestTimeoutException: Request timeout to www.dp2. 384 (100,0%)
com/127.0.0.1:80 after 60000 ms

----- TestPerformanceCreateDisease -----] 14%
[#####
waiting: 0 / active: 2977 / done: 523
----- TestPerformanceDeleteDisease -----] 0%
[-----
waiting: 0 / active: 1 / done: 0
=====

2020-05-28 12:40:20 300s elapsed
----- Requests -----
> Global (OK=27567 KO=386 )
> Home (OK=3501 KO=0 )
> Login (OK=3501 KO=0 )
> request_2 (OK=3501 KO=0 )
> Logged (OK=3375 KO=126 )
> Logged Redirect 1 (OK=3375 KO=0 )
> FindOwners (OK=3461 KO=0 )
> ListOwners (OK=1897 KO=104 )
> ShowOwner1 (OK=1531 KO=107 )
> FindOwners Redirect 1 (OK=240 KO=0 )
> ListDiseases (OK=1 KO=0 )
> CreateDiseaseFormPetOwner1 (OK=1286 KO=0 )
> ListOwners Redirect 1 (OK=199 KO=0 )
> DiseaseCreated (OK=879 KO=36 )
> DiseaseCreated Redirect 1 (OK=679 KO=13 )
> DeleteNewDisease (OK=2 KO=0 )
> ShowOwner1 Redirect 1 (OK=138 KO=0 )
> DeleteNewDisease Redirect 1 (OK=1 KO=0 )
----- Errors -----
> i.g.h.c.i.RequestTimeoutException: Request timeout to www.dp2. 386 (100,0%)
com/127.0.0.1:80 after 60000 ms

----- TestPerformanceCreateDisease -----] 16%
[#####
waiting: 0 / active: 2923 / done: 577
----- TestPerformanceDeleteDisease -----] 0%
[-----
waiting: 0 / active: 1 / done: 0
=====

```

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

Como podemos ver ya empiezan a fallar las distintas peticiones al haber aumentado a 3500 usuarios totales.

En esta parte, se realizarán pruebas en dos escenarios distintos, haciendo uso de la herramienta Gatling.

En primer lugar, he usado el grabador de Gatling para poder generar el código de los dos escenarios: **updateDiseaseScn** y **showDiseaseScn**. Posteriormente, realicé una serie de cambios para mejorar la legibilidad del código, se sustituyeron los request de las llamadas por un nombre más intuitivo, como es el caso de Home, Login o Logged entre otros, dando como resultado el siguiente código:

```
class TestPerformanceUpdateAndShowDisease extends Simulation {

  val httpProtocol = http
    .baseUrl("http://www.dp2.com")
    .inferHtmlResources(BlackList("""*.css""", """*.js""", """*.ico""", """*.png"""), WhiteList())
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3;q=0.9")
    .acceptEncodingHeader("gzip, deflate")
    .acceptLanguageHeader("es-419,es;q=0.9")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/81.0.4044.138 Safari/537.36")

  val headers_0 = Map(
    "Proxy-Connection" -> "keep-alive",
    "Upgrade-Insecure-Requests" -> "1")

  val headers_2 = Map(
    "Accept" -> "image/webp,image/apng,image/*,*/*;q=0.8",
    "Proxy-Connection" -> "keep-alive")

  val headers_3 = Map(
    "Origin" -> "http://www.dp2.com",
    "Proxy-Connection" -> "keep-alive",
    "Upgrade-Insecure-Requests" -> "1")

  object Home{
    val home = exec(
      http("Home")
        .get("/")
        .headers(headers_0))
    .pause(9)
  }

  object Login{
    val login = exec(
      http("Login")
        .get("/login")
        .headers(headers_0)
        .check(css("input[name=_csrf]", "value").saveAs("token"))))
    .pause(25)
    .exec(http("Logged")
      .post("/login")
      .headers(headers_3)
      .formParam("username", "vet1")
      .formParam("password", "v3t")
      .formParam("_csrf", "${token}")
    ).pause(50)
  }
}
```



```
object DiseasesList{
    val diseasesList = exec(
        http("DiseasesList")
        .get("/diseases/diseasesList")
        .headers(headers_0))
    .pause(82)
}

object UpdateDisease{
    val updateDisease = exec(
        http("UpdateForm")
        .get("/diseases/1/edit")
        .headers(headers_0)
        .check(css("input[name=_csrf]", "value").saveAs("stoken")))
    ).pause(27)
    .exec(http("DiseaseUpdated")
        .post("/diseases/1/edit")
        .headers(headers_3)
        .formParam("pet_id", "")
        .formParam("symptoms", "updated")
        .formParam("severity", "LOW")
        .formParam("cure", "updated1")
        .formParam("_csrf", "${stoken}")
    ).pause(16)
}

object ShowDiseaseUpdated{
    val showDiseaseUpdated = exec(http("ShowDiseaseUpdated")
        .get("/diseases/1")
        .headers(headers_0))
    .pause(11)
}

val updateDiseaseScn = scenario("DiseaseUpdate").exec(
    Home.home,
    Login.login,
    DiseasesList.diseasesList,
    UpdateDisease.updateDisease
)

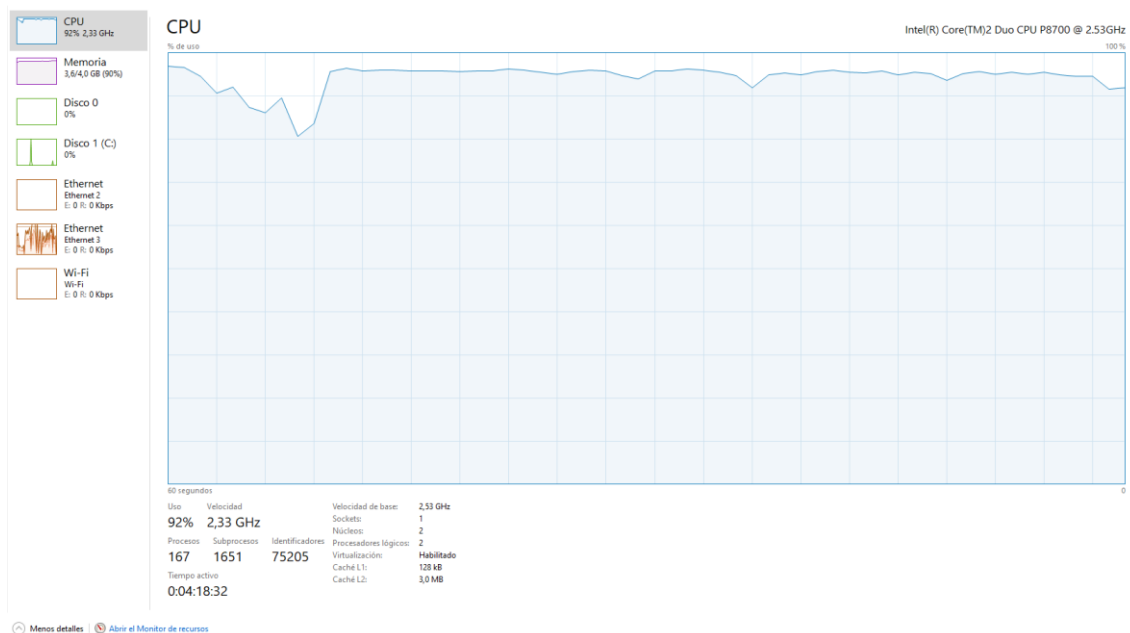
val showDiseaseScn = scenario("DiseaseShow").exec(Home.home,
    Login.login,
    DiseasesList.diseasesList,
    ShowDiseaseUpdated.showDiseaseUpdated)

setUp(updateDiseaseScn.inject(rampUsers(1000) during (100 seconds))
    showDiseaseScn.inject(rampUsers(1000) during (100 seconds))
    ).protocols(httpProtocol)
    .assertions{
        forAll.failedRequests.percent.lte(5),
        global.responseTime.max.lt(7000),
        global.responseTime.mean.lt(1000),
        global.successfulRequests.percent.gt(95)}
}
```

Las pruebas se realizaron en un equipo antiguo por lo que no se ha podido realizar las pruebas con una gran cantidad de usuarios, decidí introducir 1000 usuarios, ya que, al introducir unos 3000 usuarios, empezaba a fallar el equipo y salían errores en los análisis, y en el informe generado se obtenían tiempos de respuesta demasiado altos.

Se ha introducido un assertion, hace que el máximo de fallos de las peticiones no supere el 5%.

A continuación, se muestra una captura de pantalla del rendimiento del equipo utilizado para realizar las pruebas, se observa que al ejecutar dichas pruebas llega casi al límite de sus capacidades.



A continuación, se muestra el tiempo que ha tardado en realizar al 100% los escenarios, se observa que han terminado casi a la vez, sin ningún tipo de fallo.

```
=====
2020-05-30 12:18:26                                     145s elapsed
---- Requests ----
> Global (OK=14000 KO=0 )
> Home (OK=2000 KO=0 )
> Login (OK=2000 KO=0 )
> Logged (OK=2000 KO=0 )
> Logged Redirect 1 (OK=2000 KO=0 )
> DiseasesList (OK=2000 KO=0 )
> ShowDiseaseUpdated (OK=1000 KO=0 )
> UpdateForm (OK=1000 KO=0 )
> DiseaseUpdated (OK=1000 KO=0 )
> DiseaseUpdated Redirect 1 (OK=1000 KO=0 )

---- DiseaseShow ----
[#####] 97%
waiting: 0 / active: 22 / done: 978
---- DiseaseUpdate ----
[#####] 95%
waiting: 0 / active: 42 / done: 958
=====
```



Diseño y Pruebas II Documentación del Proyecto PetClinic

Control de Versiones

```
=====
2020-05-30 12:18:30                                149s elapsed
---- Requests -----
> Global (OK=14000 KO=0 )
> Home (OK=2000 KO=0 )
> Login (OK=2000 KO=0 )
> Logged (OK=2000 KO=0 )
> Logged Redirect 1 (OK=2000 KO=0 )
> DiseasesList (OK=2000 KO=0 )
> ShowDiseaseUpdated (OK=1000 KO=0 )
> UpdateForm (OK=1000 KO=0 )
> DiseaseUpdated (OK=1000 KO=0 )
> DiseaseUpdated Redirect 1 (OK=1000 KO=0 )


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

Simulation petclinicDiseases.TestPerformanceUpdateAndShowDisease completed in 149 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

=====
---- Global Information -----
> request count 14000 (OK=14000 KO=0 )
> min response time 0 (OK=0 KO=- )
> max response time 6681 (OK=6681 KO=- )
> mean response time 116 (OK=116 KO=- )
> std deviation 391 (OK=391 KO=- )
> response time 50th percentile 6 (OK=6 KO=- )
> response time 75th percentile 34 (OK=34 KO=- )
> response time 95th percentile 688 (OK=688 KO=- )
> response time 99th percentile 2058 (OK=2058 KO=- )
> mean requests/sec 93.96 (OK=93.96 KO=- )
---- Response Time Distribution -----
> t < 800 ms 13386 ( 96%)
> 800 ms < t < 1200 ms 197 ( 1%)
> t > 1200 ms 417 ( 3%)
> failed 0 ( 0%)
=====
```

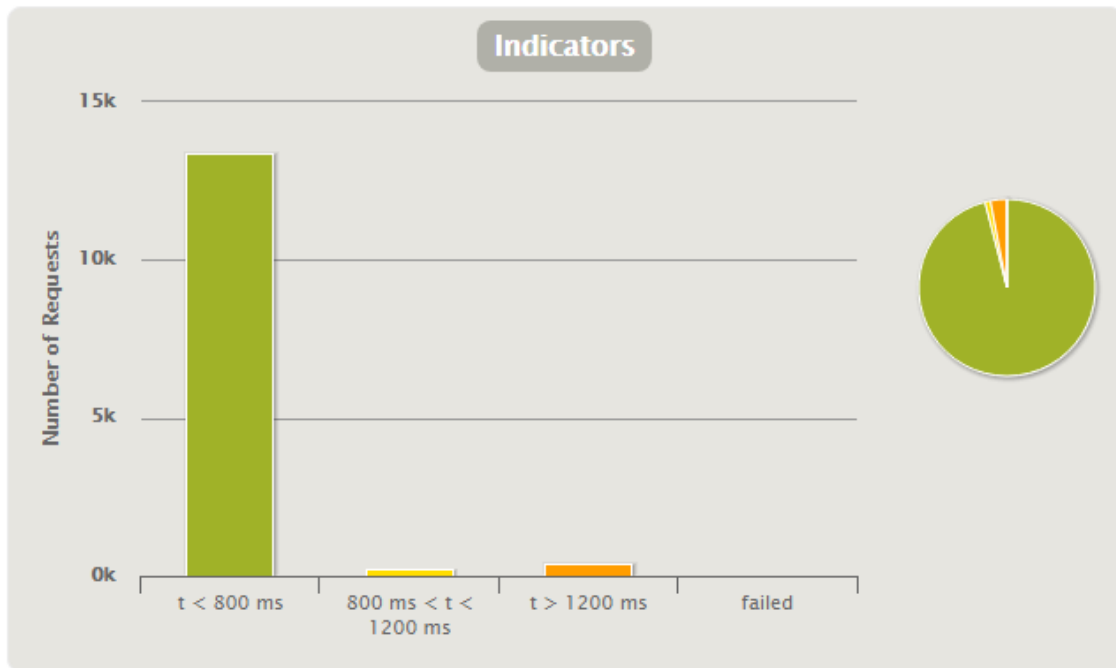
Finalmente, segundos después, terminan las pruebas de ambos escenarios y se puede observar que se ha generado un informe con las distintas métricas, que se pueden observar a continuación:

ASSERTIONS	
Assertion ↕	Status ↕
Home: percentage of failed events is less than or equal to 5.0	OK
LoginForm: percentage of failed events is less than or equal to 5.0	OK
Logged: percentage of failed events is less than or equal to 5.0	OK
Logged Redirect 1: percentage of failed events is less than or equal to 5.0	OK
DiseasesList: percentage of failed events is less than or equal to 5.0	OK
UpdateForm: percentage of failed events is less than or equal to 5.0	OK
ShowDiseaseUpdated: percentage of failed events is less than or equal to 5.0	OK
DiseaseUpdated: percentage of failed events is less than or equal to 5.0	OK
DiseaseUpdated Redirect 1: percentage of failed events is less than or equal to 5.0	OK
Global: max of response time is less than 7000.0	OK
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

	<p>Diseño y Pruebas II Documentación del Proyecto PetClinic</p>
<p>Control de Versiones</p>	

Se puede observar que se cumplen todas las restricciones que se han puesto en el código.

Al ser un número de usuarios reducido y al realizar este tipo de pruebas, como son el visionado y la actualización de una enfermedad, se puede observar que los tiempos obtenidos, son aceptables en la mayoría de las peticiones, aunque hay un pequeño número que sobrepasan los 1200 ms.




Finalmente procedo a comprobar si dichas pruebas han tenido algún efecto sobre la base de datos, se accede al terminal de Docker y se comprueba que efectivamente, se ha realizado la tarea correctamente, como se observa en la primera fila de la tabla Diseases.

```
mysql> select* from diseases;
```

id	cure	severity	symptoms	pet_id
1	updated1	LOW	updated	1
2	tengo el hambre	MEDIUM	la vas a espichar	2
3	necrosis	MEDIUM	no tenemos na que hacer	3
4	me duele el costao	MEDIUM	me duele mucho	4
5	me pica la cabeza	HIGH	y no se que hacer	5
6	dame de comer	HIGH	humano... me puedes rascar?	6
7	como una bola me voy a poner	HIGH	de tanto comer	7

7 rows in set (0.00 sec)

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

A continuación, se observa que el equipo empieza a fallar con 3000 usuarios.

```
=====
2020-05-30 12:36:58                                180s elapsed
---- Requests -----
> Global (OK=26753 KO=21 )
> Home (OK=6000 KO=0 )
> Login (OK=6000 KO=0 )
> Logged (OK=5979 KO=21 )
> Logged Redirect 1 (OK=5979 KO=0 )
> DiseasesList (OK=2680 KO=0 )
> DiseasesList Redirect 1 (OK=115 KO=0 )
---- Errors -----
> i.g.h.c.i.RequestTimeoutException: Request timeout to www.dp2. 21 (100,0%)
com/127.0.0.1:80 after 60000 ms

---- DiseaseShow -----
[-----] 0%
      waiting: 0 / active: 3000 / done: 0
---- DiseaseUpdate -----
[-----] 0%
      waiting: 0 / active: 3000 / done: 0
=====
```

También se puede observar que han transcurrido 180 segundos y aún no han empezado las pruebas, si llegado el caso de que se realizaran sin ningún problema, arrojaría tiempos demasiados elevados.



3. Test de rendimiento Salas

Se ha procedido a crear 2 escenarios:

- **UpdateRoomScn**
- **ShowRoomScn**

Una vez creados estos escenarios, se hacen las llamadas a sus respectivas peticiones. Probando se ha visto que el mostrar va más rápido que el actualizar, pero esto no supone ningún problema.

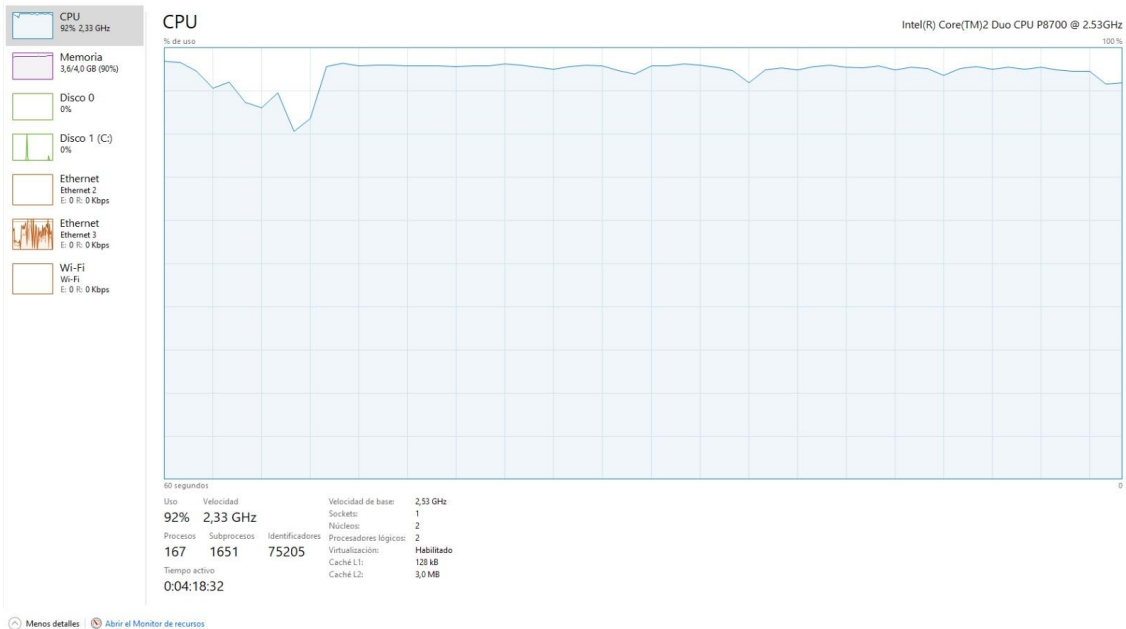
Se ha modificado el código siguiendo las pautas de los vídeos, para su correcta ejecución quedando el siguiente código resultante:

```
class TestPerformanceUpdateAndShow extends Simulation {  
  
  val httpProtocol = http  
    .baseUrl("http://www.dp2.com")  
    .inferHtmlResources(BlackList("""*.css""", """*.js""", """*.ico""", """*.png"""), Whitelist())  
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,  
    application/signed-exchange;v=b3;q=0.9")  
    .acceptEncodingHeader("gzip, deflate")  
    .acceptLanguageHeader("es-419,es;q=0.9")  
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
    Chrome/81.0.4044.138 Safari/537.36")  
  
  val headers_0 = Map(  
    "Proxy-Connection" -> "keep-alive",  
    "Upgrade-Insecure-Requests" -> "1")  
  
  val headers_2 = Map(  
    "Accept" -> "image/webp,image/apng,image/*,*/*;q=0.8",  
    "Proxy-Connection" -> "keep-alive")  
  
  val headers_3 = Map(  
    "Origin" -> "http://www.dp2.com",  
    "Proxy-Connection" -> "keep-alive",  
    "Upgrade-Insecure-Requests" -> "1")  
  
  object Home{  
    val home = exec(  
      http("Home")  
        .get("/")  
        .headers(headers_0)  
    ).pause(5)  
  }  
  
  object Login{  
    val login = exec(  
      http("Login")  
        .get("/login")  
        .headers(headers_0)  
        .check(css("input[name=_csrf]", "value").saveAs("token"))  
    ).pause(18)  
    .exec(  
      http("Logged")  
        .post("/login")  
        .headers(headers_3)  
        .formParam("username", "admin1")  
        .formParam("password", "4dm1n")  
        .formParam("_csrf", "${token}")  
    ).pause(12)  
  }  
}
```

```
object RoomsList{
    val roomsList = exec(
        http("RoomsList")
        .get("/rooms/roomsList")
        .headers(headers_0))
    .pause(13)
}
object UpdateRoom{
    val updateRoom = exec(
        http("RoomUpdateForm")
        .get("/rooms/1/edit")
        .headers(headers_0)
        .check(css("input[name=_csrf]", "value").saveAs("token")))
    ).pause(27)
    .exec(http("RoomUpdated")
        .post("/rooms/1/edit")
        .headers(headers_3)
        .formParam("name", "updated")
        .formParam("floor", "1")
        .formParam("medicalTeam", "nadena")
        .formParam("_csrf", "${token}")
    ).pause(14)
}
object ShowRoom{
    val showRoom = exec(http("ShowRoom")
        .get("/rooms/1")
        .headers(headers_0))
    .pause(11)
}
val updateRoomScn = scenario("RoomsUpdate").exec(Home.home,
    Login.login,
    RoomsList.roomsList,
    UpdateRoom.updateRoom)
val showRoomScn = scenario("RoomsShow").exec(Home.home,
    Login.login,
    RoomsList.roomsList,
    ShowRoom.showRoom)
setUp(
    updateRoomScn.inject(rampUsers(1000) during (100 seconds)),
    showRoomScn.inject(rampUsers(1000) during (100 seconds))
).protocols(httpProtocol)
.assertions(
    forAll.failedRequests.percent.lte(5),
    global.responseTime.max.lt(50000),
    global.responseTime.mean.lt(1200),
    global.successfulRequests.percent.gt(95)
)
}
```

Se ha añadido las restricciones para que se compruebe que el total de peticiones de cada objeto no supere el 5%.

Ahora mostraremos el avance que ha tenido ejecutar esta prueba:




En este momento es cuando nos encontramos ejecutando 1000 usuarios activos.

Ahora veremos cuando ha terminado de mostrar los elementos, pero todavía se sigue actualizando:

```
=====
2020-05-23 15:33:32                                160s elapsed
---- Requests ----
> Global (OK=13686 KO=0 )
> Home (OK=2000 KO=0 )
> Login (OK=2000 KO=0 )
> Logged (OK=2000 KO=0 )
> Logged Redirect 1 (OK=2000 KO=0 )
> RoomsList (OK=2000 KO=0 )
> ShowRoom (OK=1000 KO=0 )
> RoomUpdateForm (OK=1000 KO=0 )
> RoomUpdated (OK=843 KO=0 )
> RoomUpdated Redirect 1 (OK=843 KO=0 )

---- RoomsShow ----
[#####]100%
  waiting: 0 / active: 0 / done: 1000
---- RoomsUpdate ----
[#####] 68%
  waiting: 0 / active: 314 / done: 686
=====
```


	<p>Diseño y Pruebas II Documentación del Proyecto PetClinic</p>
<p>Control de Versiones</p>	

Vemos que con 1000 usuarios no ha habido ninguna petición fallida, en cambio los encargados de actualizar todavía están realizando su tarea. Esto es debido a que lógicamente se dedica más tiempo a actualizar que al mostrar un elemento.

Pasado un tiempo podemos ver como efectivamente ha cumplido su tarea:

```
=====
2020-05-23 15:34:01                                189s elapsed
---- Requests -----
> Global (OK=14000 KO=0 )
> Home (OK=2000 KO=0 )
> Login (OK=2000 KO=0 )
> Logged (OK=2000 KO=0 )
> Logged Redirect 1 (OK=2000 KO=0 )
> RoomsList (OK=2000 KO=0 )
> ShowRoom (OK=1000 KO=0 )
> RoomUpdateForm (OK=1000 KO=0 )
> RoomUpdated (OK=1000 KO=0 )
> RoomUpdated Redirect 1 (OK=1000 KO=0 )

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

Simulation petclinicRoom.TestPerformanceUpdateAndShow completed in 189 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

----- Global Information -----
> request count 14000 (OK=14000 KO=0 )
> min response time 1 (OK=1 KO=- )
> max response time 10505 (OK=10505 KO=- )
> mean response time 346 (OK=346 KO=- )
> std deviation 963 (OK=963 KO=- )
> response time 50th percentile 11 (OK=11 KO=- )
> response time 75th percentile 120 (OK=120 KO=- )
> response time 95th percentile 1839 (OK=1839 KO=- )
> response time 99th percentile 6323 (OK=6323 KO=- )
> mean requests/sec 74.074 (OK=74.074 KO=- )
---- Response Time Distribution -----
> t < 800 ms 12036 ( 86%)
> 800 ms < t < 1200 ms 689 ( 5%)
> t > 1200 ms 1275 ( 9%)
> failed 0 ( 0%)
=====
```




Una vez finalizado el test decidimos comprobar si realmente se había actualizado en **docker**:

```
mysql> select* from rooms;
+----+-----+-----+
| id | name      | floor |
+----+-----+-----+
| 1  | updated   | 1     |
| 2  | Quirofano2 | 1     |
| 3  | Quirofano3 | 2     |
| 4  | Quirofano4 | 2     |
+----+-----+-----+
4 rows in set (0.00 sec)
```

Podemos ver que el elemento con id 1 ha sido actualizado.

```
mysql> select* from room_medical_team;
+-----+-----+
| room_id | medical_team |
+-----+-----+
| 1       | nadena       |
| 2       | Tijeras      |
| 2       | Pinzas       |
| 2       | Bandejas     |
| 2       | Bisturi      |
| 2       | Mascarillas  |
| 3       | Tijeras      |
| 3       | Pinzas       |
| 3       | Bandejas     |
| 3       | Bisturi      |
| 3       | Mascarillas  |
| 4       | Tijeras      |
| 4       | Pinzas       |
| 4       | Bandejas     |
| 4       | Bisturi      |
| 4       | Mascarillas  |
+-----+-----+
16 rows in set (0.00 sec)
```

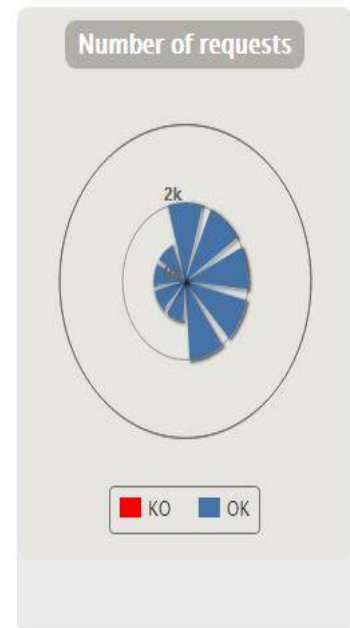
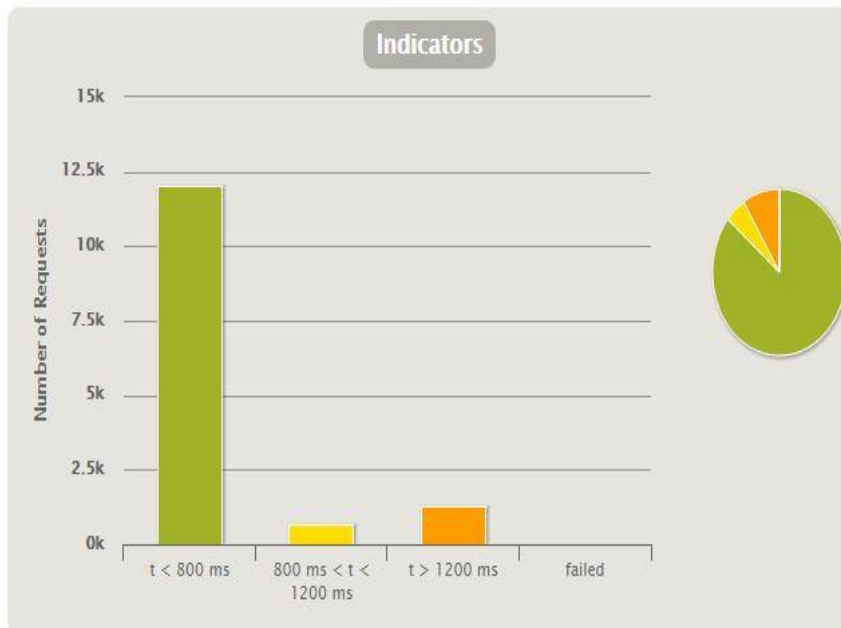
Se ha comprobado que también ha sido actualizado su equipo médico que es el que tiene la id 1.


	<p>Diseño y Pruebas II Documentación del Proyecto PetClinic</p>
<p>Control de Versiones</p>	

Podemos apreciar que ha cumplido las restricciones que se le puso:

ASSERTIONS	
Assertion ↕	Status ↕
Home: percentage of failed events is less than or equal to 5.0	OK
Login: percentage of failed events is less than or equal to 5.0	OK
Logged: percentage of failed events is less than or equal to 5.0	OK
Logged Redirect 1: percentage of failed events is less than or equal to 5.0	OK
RoomsList: percentage of failed events is less than or equal to 5.0	OK
RoomUpdateForm: percentage of failed events is less than or equal to 5.0	OK
ShowRoom: percentage of failed events is less than or equal to 5.0	OK
RoomUpdated: percentage of failed events is less than or equal to 5.0	OK
RoomUpdated Redirect 1: percentage of failed events is less than or equal to 5.0	OK
Global: max of response time is less than 50000.0	OK
Global: mean of response time is less than 1200.0	OK
Global: percentage of successful events is greater than 95.0	OK

Ahora miraremos los tiempos obtenidos:



	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

Vemos que la aplicación se ha ralentizado, pero estos tiempos de aumento se han debido a este tipo de peticiones:

- **Iniciar sesión** cosa que es normal al haber mucho tráfico.
- **Listar y mostrar** las salas.
- **Actualizar** las salas.

Viendo que no ha devuelto ningún fallo la aplicación y el tiempo de retraso no ha sido generado por una única petición si no al revés, por varias peticiones, y que el equipo donde se han realizado estas pruebas no es muy bueno, veo que el rendimiento es aceptable para el equipo en el que se ha ejecutado.

En un portátil diferente se han realizado las pruebas para los escenarios de creación y borrado de las entidades y para ello se han creado los escenarios:

- **createRoomScn**
- **deleteRoomScn**

Para ello se han seguido los pasos en los videos de teoría. Se ha usado recorder.bat para la grabación de los pasos a seguir y luego usaremos gatling.bat para el análisis del rendimiento de estos escenarios. El código final de la prueba resulta así:

```
RoomCreateAndDelete.scala X
DP2-1920-G2-15 > src > performance > Room > RoomCreateAndDelete.scala
1 package dp2
2
3 import scala.concurrent.duration._
4
5 import io.gatling.core.Predef._
6 import io.gatling.http.Predef._
7 import io.gatling.jdbc.Predef._
8
9 class RoomCreateAndDelete extends Simulation {
10
11     val httpProtocol = http
12         .baseUrl("http://www.dp2.com")
13         .inferHtmlResources(Blacklist(""".*.css""", "".*.js""", "".*.ico""", "".*.png""), WhiteList())
14         .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9")
15         .acceptEncodingHeader("gzip, deflate")
16         .acceptLanguageHeader("es-ES;q=0.9,en;q=0.8,fr;q=0.7")
17         .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36")
18
19     val headers_0 = Map(
20         "Proxy-Connection" -> "keep-alive",
21         "Upgrade-Insecure-Requests" -> "1")
22
23     val headers_2 = Map(
24         "Accept" -> "image/webp,image/apng,image/*,*/*;q=0.8",
25         "Proxy-Connection" -> "keep-alive")
26
27     val headers_3 = Map(
28         "Origin" -> "http://www.dp2.com",
29         "Proxy-Connection" -> "keep-alive",
30         "Upgrade-Insecure-Requests" -> "1")
31
32     object Home{
33         val home = exec(
34             http("Home")
35                 .get("/")
36                 .headers(headers_0)
37         ).pause(5)
38     }
39 }
```



Diseño y Pruebas II Documentación del Proyecto PetClinic

Control de Versiones

```
RoomCreateAndDelete.scala X
DP2-1920-G2-15 > src > performance > Room > RoomCreateAndDelete.scala
39 object Login{
40     val login = exec(
41         http("Login")
42         .get("/login")
43         .headers(headers_0)
44         .check(css("input[name=_csrf]", "value").saveAs("stoken")))
45     ).pause(18)
46     .exec(
47         http("logged")
48         .post("/login")
49         .headers(headers_3)
50         .formParam("username", "admin1")
51         .formParam("password", "4dmin")
52         .formParam("_csrf", "${stoken}")
53     ).pause(12)
54 }
55
56 object RoomList{
57     val roomList = exec(http("RoomList")
58         .get("/rooms/roomsList")
59         .headers(headers_0))
60     .pause(13)
61 }
62
63 object DeleteRoom{
64     val deleteRoom = exec(http("DeleteRoom")
65         .get("/rooms/delete/6")
66         .headers(headers_0))
67     .pause(12)
68 }
```

```
RoomCreateAndDelete.scala X
DP2-1920-G2-15 > src > performance > Room > RoomCreateAndDelete.scala
70 object CreateRoom{
71     val createRoom = exec(http("CreateRoomForm")
72         .get("/rooms/new")
73         .headers(headers_0)
74         .check(css("input[name=_csrf]", "value").saveAs("stoken"))))
75     .pause(27)
76     .exec(http("CreatedRoom")
77         .post("/rooms/new")
78         .headers(headers_3)
79         .formParam("name", "room")
80         .formParam("floor", "1")
81         .formParam("medicalTeam", "bisturi")
82         .formParam("_csrf", "${stoken}"))
83     .pause(17)
84 }
85
86 val createRoomScn = scenario("CreateRoom").exec(Home.home,
87     Login.login,
88     RoomList.roomList,
89     CreateRoom.createRoom)
90
91 val deleteRoomScn = scenario("DeleteRoom").exec(Home.home,
92     Login.login,
93     RoomList.roomList,
94     DeleteRoom.deleteRoom)
95
96
97 setup(
98     createRoomScn.inject(rampUsers(500) during (100 seconds)),
99     deleteRoomScn.inject(rampUsers(500) during (100 seconds))
100 ).protocols(httpProtocol)
101 .assertions(
102     global.responseTime.max.lt(5000),
103     global.responseTime.mean.lt(5000),
104     global.successfulRequests.percent.gt(95)
105 )
106 }
```

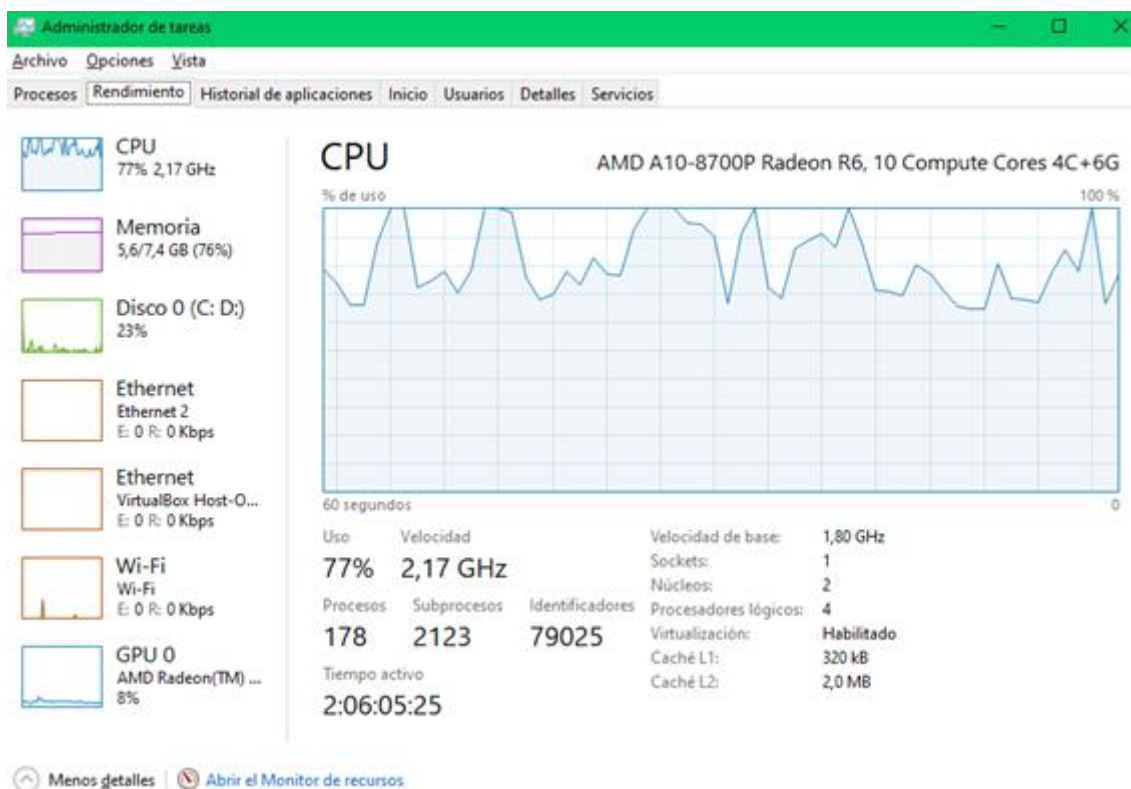
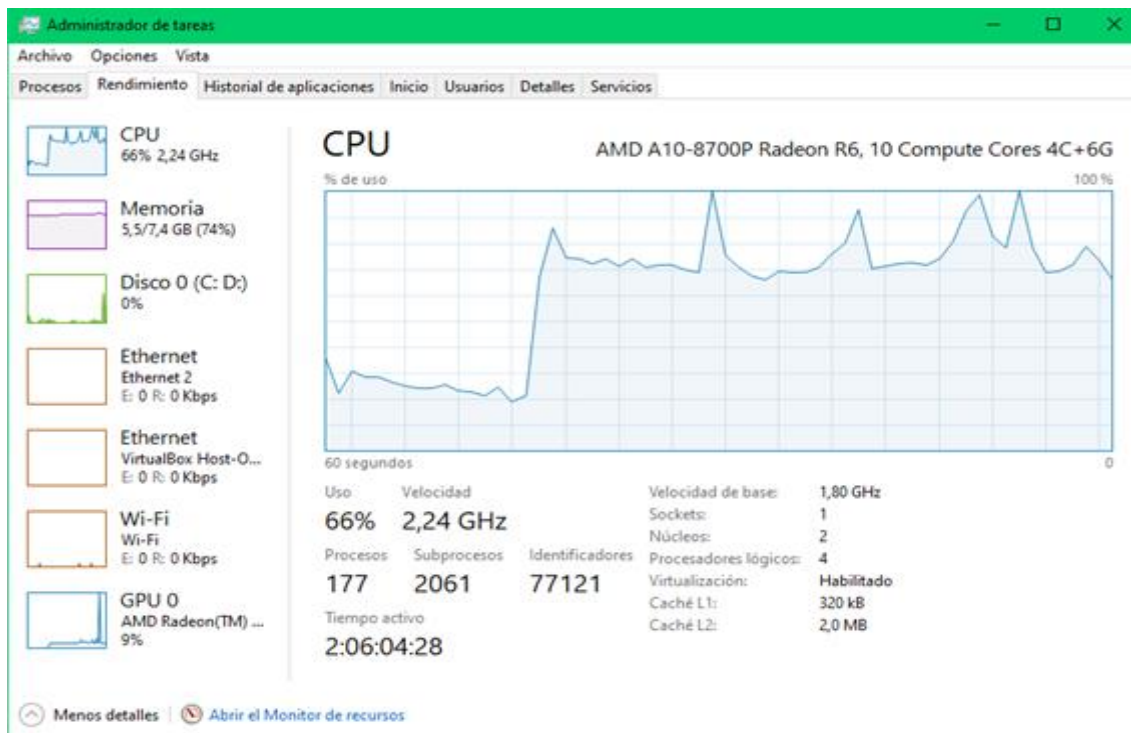


Diseño y Pruebas II

Documentación del Proyecto PetClinic

Control de Versiones

Debido a que el ordenador usado tiene algunos problemas de rendimiento, con pocos usuarios empieza a realizar respuestas más lentas. Una prueba con 300 usuarios (total de 600) podemos ver el rendimiento en el administrador de tareas:




Y el resultado de las pruebas se consideran buenos, aunque se hayan obtenido algunas respuestas más tardías:



Pero una vez que se suben el número de usuarios a 500 por escenario (total de 1000) empezamos a obtener excepciones:




	<p>Diseño y Pruebas II Documentación del Proyecto PetClinic</p>
<p>Control de Versiones</p>	

Con la prueba inicial de los 300 usuarios por escenario pasamos todos los Assertions menos uno de ellos por lo tiempos de respuesta.

▶ ASSERTIONS	
Assertion +	Status +
Home: percentage of failed events is less than or equal to 5.0	OK
Login: percentage of failed events is less than or equal to 5.0	OK
Logged: percentage of failed events is less than or equal to 5.0	OK
Logged Redirect 1: percentage of failed events is less than or equal to 5.0	OK
RoomList: percentage of failed events is less than or equal to 5.0	OK
CreateRoomForm: percentage of failed events is less than or equal to 5.0	OK
DeleteRoom: percentage of failed events is less than or equal to 5.0	OK
CreatedRoom: percentage of failed events is less than or equal to 5.0	OK
CreatedRoom Redirect 1: percentage of failed events is less than or equal to 5.0	OK
DeleteRoom Redirect 1: percentage of failed events is less than or equal to 5.0	OK
RoomList Redirect 1: percentage of failed events is less than or equal to 5.0	OK
CreateRoomForm Redirect 1: percentage of failed events is less than or equal to 5.0	OK
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 5000.0	OK
Global: percentage of successful events is greater than 95.0	OK

Como conclusión podemos decir que estas pruebas muestran un rendimiento adecuado para estos escenarios, con un equipo mejor se podría mostrar un mejor rendimiento de estos.

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

4. Test de rendimiento Booking (Reserva sala).

Se han procedido a crear 4 escenarios:

- List Bookings
- Create Booking
- Update Booking
- Delete Booking

Para ello configuramos gatling y el ordenador para poder grabar las consultas que haremos a la aplicación. Una vez grabando, entramos en la página de inicio y nos autorizamos con una cuenta “veterinarian”. Abrimos la lista de bookings, creamos uno, actualizamos otro y borramos otro. Con esto gatling generará un archivo scala que después de retocarlo para la creación de los 4 escenarios quedaría así:

```
class Booking extends Simulation {

  val httpProtocol = http
    .baseUrl("http://www.dp2.com")
    .inferHtmlResources(BlackList("*.css", "*.js", "*.ico", "*.png", ".*\\.js",
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng
    .acceptEncodingHeader("gzip, deflate")
    .acceptLanguageHeader("es,en-US;q=0.9,en;q=0.8")
    .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, lik

  val headers_0 = Map(
    "Proxy-Connection" -> "keep-alive",
    "Upgrade-Insecure-Requests" -> "1")

  val headers_2 = Map(
    "Accept" -> "image/webp,image/apng,image/*,*/*;q=0.8",
    "Proxy-Connection" -> "keep-alive")

  val headers_3 = Map(
    "Origin" -> "http://www.dp2.com",
    "Proxy-Connection" -> "keep-alive",
    "Upgrade-Insecure-Requests" -> "1")

  object Home {
    val home = exec(http("Home")
      .get("/")
      .headers(headers_0))
      .pause(6)
  }

  object Login {
    val login = exec(http("Login")
      .get("/login")
      .headers(headers_0)
      .check(css("input[name=_csrf]", "value").saveAs("stoken")))
      .pause(17)
    exec(http("Logged")
      .post("/login")
      .headers(headers_3)
      .formParam("username", "vet1")
      .formParam("password", "v3t")
      .formParam("_csrf", "${stoken}"))
      .pause(17)
  }
}
```




```
object CreateBooking {
  val createBooking = exec(http("Create Form")
    .get("/bookings/new")
    .headers(headers_0)
    .check(css("input[name=_csrf]", "value").saveAs("stoken"))))
  .pause(23)
  .exec(http("Create Post")
    .post("/bookings/new")
    .headers(headers_3)
    .formParam("fecha", "2020-05-24")
    .formParam("petId", "1")
    .formParam("vetId", "1")
    .formParam("roomId", "1")
    .formParam("_csrf", "${stoken}"))
  .pause(19)
}

object UpdateBooking {
  val updateBooking = exec(http("Update Form")
    .get("/bookings/edit/1")
    .headers(headers_0)
    .check(css("input[name=_csrf]", "value").saveAs("stoken"))))
  .pause(5)
  .exec(http("Update Post")
    .post("/bookings/edit/1")
    .headers(headers_3)
    .formParam("fecha", "2020-01-05")
    .formParam("petId", "6")
    .formParam("vetId", "3")
    .formParam("roomId", "2")
    .formParam("_csrf", "${stoken}"))
  .pause(5)
}


object DeleteBooking {
  val deleteBooking = exec(http("delete")
    .get("/bookings/1/delete")
    .headers(headers_0))
  .pause(6)
}
```



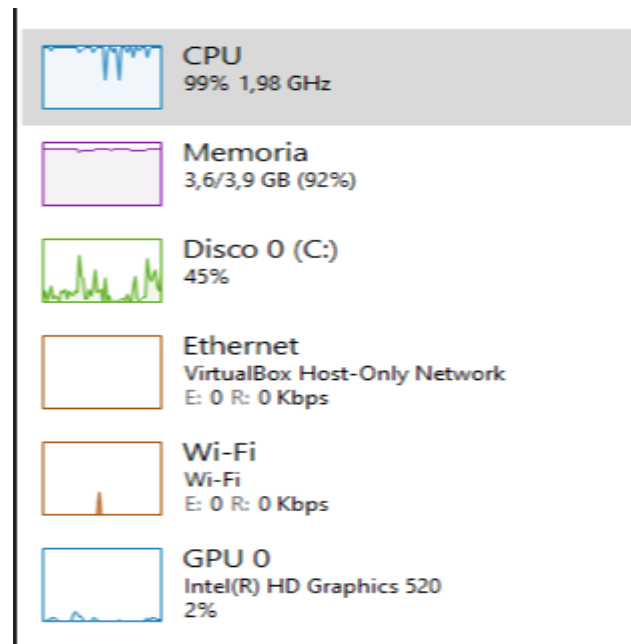
```
val listScn = scenario("List").exec(Home.home,  
    Login.login,  
    ListBookings.listbookings)  
  
val createScn = scenario("Create").exec(Home.home,  
    Login.login,  
    ListBookings.listbookings,  
    CreateBooking.createBooking)  
  
val updateScn = scenario("Update").exec(Home.home,  
    Login.login,  
    ListBookings.listbookings,  
    UpdateBooking.updateBooking)  
  
val deleteScn = scenario("Delete").exec(Home.home,  
    Login.login,  
    ListBookings.listbookings,  
    DeleteBooking.deleteBooking)  
  
setUp(  
    listScn.inject(rampUsers(500) during (10 seconds)),  
    createScn.inject(rampUsers(500) during (10 seconds)),  
    updateScn.inject(rampUsers(500) during (10 seconds)),  
    deleteScn.inject(rampUsers(500) during (10 seconds)),  
    ).protocols(httpProtocol)
```

Con este archivo terminado podemos lanzar las pruebas de rendimiento. Primero haremos una prueba de estrés, que consiste en ver el número máximo de usuarios que permite la aplicación teniendo un buen comportamiento.

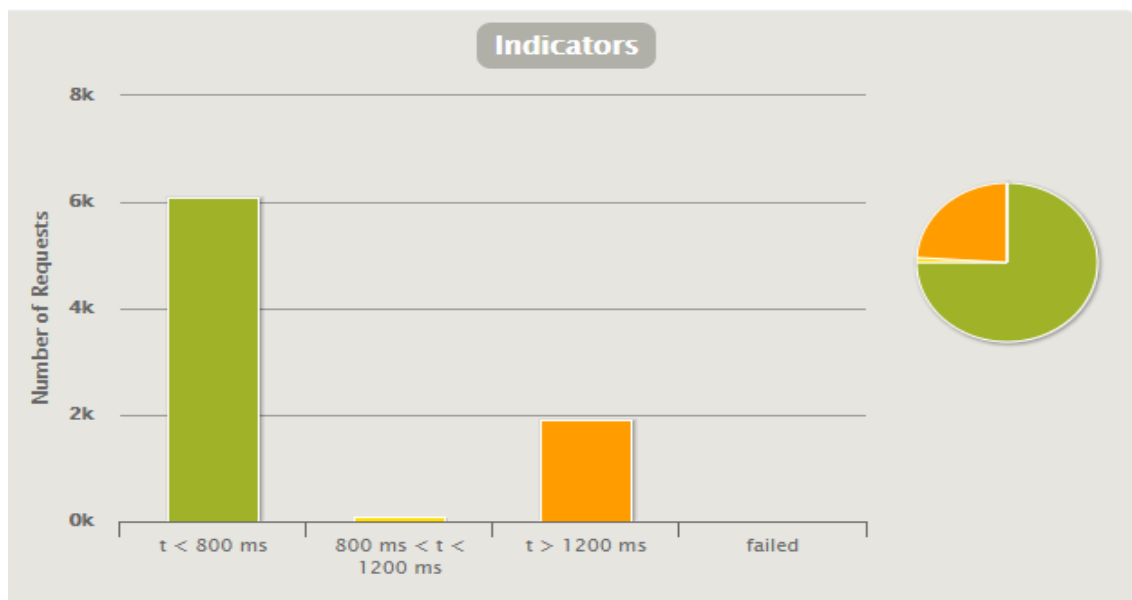
Esta prueba la haremos con total de 1200 usuarios.

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

Durante el proceso este fue el uso de la CPU y la RAM del ordenador:




Podemos observar como el ordenador está trabajando al máximo rendimiento, esto puede ser debido a que este pc no tiene muchos recursos y tiene varios años de uso.



Al responderse la amplia mayoría de las peticiones (aunque algunas pocas tarden demasiado) contaremos con que es un buen resultado.

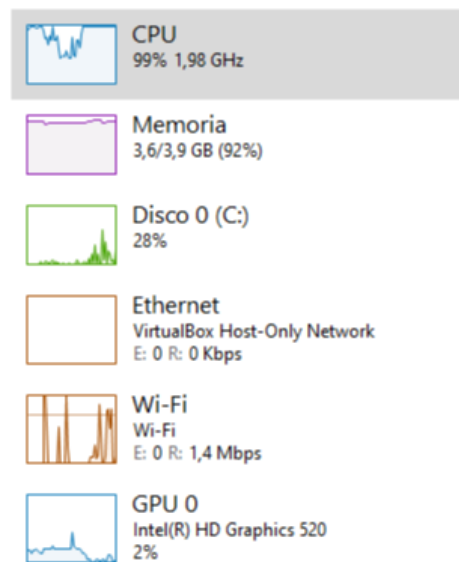
Si analizamos las tablas vemos que los métodos de booking tardan demasiado, así que sería recomendable revisarlos. Sobre todo, tarda demasiado el list, esto puede ser porque tenga muchas dependencias y con las queries traigan muchos objetos desde la base de datos.

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

STATISTICS Expand all groups Collapse all groups													
Requests ^	Executions					Response Time (ms)							
	Total ↕	OK ↕	KO ↕	% KO ↕	Cnt/s ↕	Min ↕	50th pct ↕	75th pct ↕	95th pct ↕	99th pct ↕	Max ↕	Mean ↕	Std Dev ↕
Global Information	8100	8100	0	0%	56.25	1	34	784	13190	35642	41017	2688	6239
Home	1200	1200	0	0%	8.333	3	21	211	641	880	1236	150	226
Login	1200	1200	0	0%	8.333	1	5	55	216	468	1114	48	104
Logged	1200	1200	0	0%	8.333	1	7	117	443	763	1311	96	171
Logged Redirect 1	1200	1200	0	0%	8.333	1	5	27	418	707	1182	63	145
List booking	1200	1200	0	0%	8.333	17	11383	14080	35960	38828	41017	13099	9817
delete	300	300	0	0%	2.083	2	4900	7095	14376	23820	29912	5788	4795
Create Form	300	300	0	0%	2.083	6	5459	7546	14675	25589	29902	6280	4830
Update Form	300	300	0	0%	2.083	8	5416	7466	12566	20245	29176	6095	4116
Create Post	300	300	0	0%	2.083	2	5	14	1209	1896	2820	162	424
Create P...direct 1	300	300	0	0%	2.083	10	32	54	671	896	1256	98	198
Update Post	300	300	0	0%	2.083	2	5	38	1025	1894	2733	176	417
Update P...direct 1	300	300	0	0%	2.083	14	39	100	784	1274	1914	162	284

Ahora pasaremos a pruebas de cuello de botella. Usaremos 20000 usuarios, aunque ya empieza a fallar antes, pero con 20000 la aplicación falla completamente, salvándose un porcentaje muy pequeño de las peticiones.

El uso de la CPU y RAM es el siguiente:



Es prácticamente el mismo para la otra prueba, porque en los dos casos la CPU y la RAM están al límite.

Podemos observar la gran cantidad de KOs que nos da la prueba y las pocas que se salvan tardan mucho en ser respondidas.



Diseño y Pruebas II

Documentación del Proyecto PetClinic

Control de Versiones





5. Tests de rendimiento de Opinion

Las historias de usuario que han sido comprobadas son todas las relacionadas con la entidad Opinion que son: create, update, delete, list y list mine. Todas ellas han sido comprobadas en los tests pero para ellos solo se han usado tres escenarios:

- CreateOpScn: escenario de creación de una opinión.
- UpdateOpScn: escenario de actualización de una opinión.
- DeleteOpScn: escenario de borrado de una opinión.

El motivo de que se usen solo tres escenarios para las cinco historias es el siguiente, para actualizar o borrar una opinión hay que acceder primero al listado de opiniones y luego al listado propio, por lo tanto, se comprueban dichas historias porque son pasos necesarios para la actualización y el borrado.

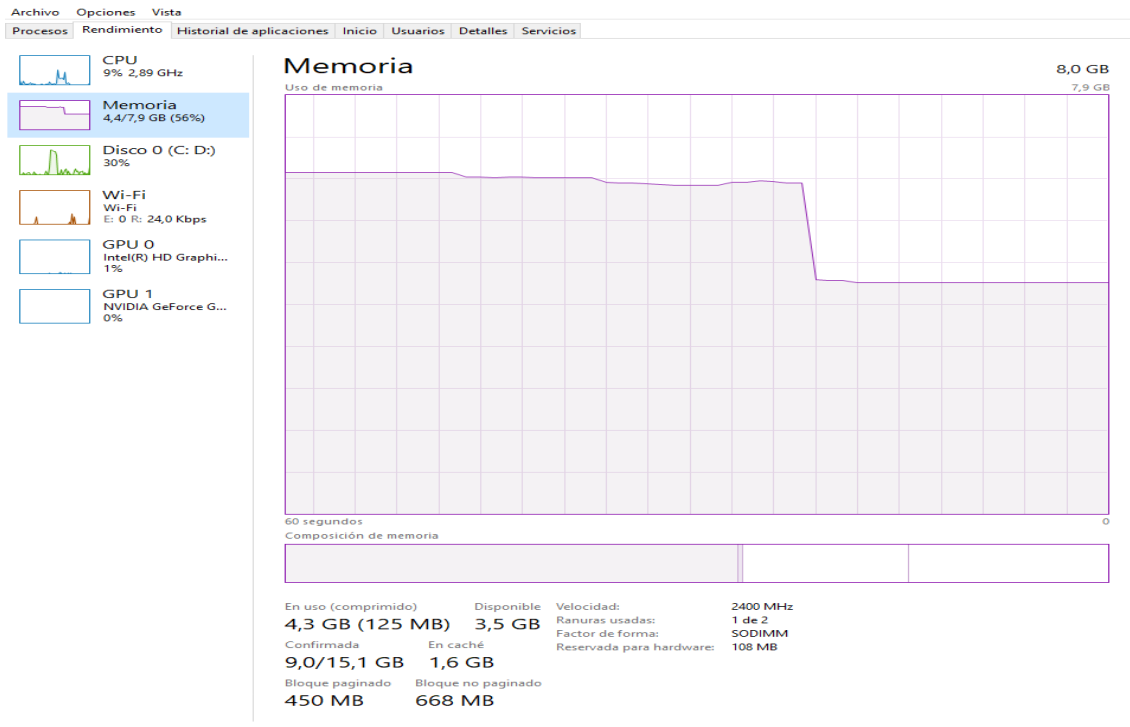
Primero veremos la prueba de rendimiento con la que hemos alcanzado el número de usuarios simultáneos para alcanzar un cuello de botella.

```
C:\WINDOWS\system32\cmd.exe
request count          1076469 (OK=56610 KO=1019859)
min response time      0 (OK=0 KO=0)
max response time      53874 (OK=53874 KO=21373)
mean response time     3692 (OK=4235 KO=3662)
std deviation          3578 (OK=8300 KO=3110)
response time 50th percentile 3043 (OK=1149 KO=3104)
response time 75th percentile 4344 (OK=3009 KO=4372)
response time 95th percentile 12456 (OK=21102 KO=9038)
response time 99th percentile 17450 (OK=51724 KO=16917)
mean requests/sec      7634.532 (OK=401.489 KO=7233.043)
--- Response Time Distribution ---
t < 800 ms             22753 ( 2%)
800 ms < t < 1200 ms  5930 ( 1%)
t > 1200 ms           27927 ( 3%)
failed                 1019859 ( 95%)
--- Errors ---
i.n.c.AbstractChannel$AnnotatedSocketException: No buffer spac 730190 (71,60%)
available (maximum connections reached?): connect: www.dp2.c...
i.n.c.AbstractChannel$AnnotatedSocketException: Address already 241205 (23,65%)
in use: no further information: www.dp2.com/127.0.0.1:80
i.n.c.ConnectTimeoutException: connection timed out: www.dp2.c 48048 ( 4,71%)
m/127.0.0.1:80
i.n.c.AbstractChannel$AnnotatedConnectException: Connection re 416 ( 0,04%)
used: no further information: www.dp2.com/127.0.0.1:80
-----
reports generated in 0s.
please open the following file: C:\Users\ijfnfu\Desktop\Carrera\Tercero\DP\gatling-charts-highcharts-bundle-3.3.1\result
testrendimientoopinion-20200525191222253\index.html
presione una tecla para continuar . . .
val delete = exec(http("Delete")
    .get("/opinions/7/delete")
    .headers(headers_0))
    .pause(5)
}

val createOpScn = scenario("CreateOpinion").exec(Home.home, Login.login, Logged.logged, Vets.vets, Create.cre
val updateOpScn = scenario("UpdateOpinion").exec(Home.home, Login.login, Logged.logged, Vets.vets, List.list,
val deleteOpScn = scenario("DeleteOpinion").exec(Home.home, Login.login, Logged.logged, Vets.vets, List.list,

setUp(
    createOpScn.inject(rampUsers(50000) during (10 seconds)),
    updateOpScn.inject(rampUsers(50000) during (10 seconds)),
    deleteOpScn.inject(rampUsers(50000) during (10 seconds))
).protocols(httpProtocol)
```

Los tres escenarios se lanzan en el mismo archivo scala, con un total de 150.000 usuarios, cada escenario tiene 50.000 usuarios. Podemos ver que el 95% de los usuarios fallan. Ahora veamos el rendimiento del ordenador.



Hay que tener en consideración que la captura de pantalla se tomó justo al finalizar las pruebas, por ello vemos la memoria bajar y el procesador en niveles bajos. Teniendo en cuenta esto, el procesador no bajó del 80% durante la ejecución y la memoria se mantuvo por encima del 70%. Cabe destacar que el ordenador usado es de gama alta y por ello el rendimiento del sistema varía con respecto a los otros usados.



Diseño y Pruebas II Documentación del Proyecto PetClinic

Control de Versiones

Ahora analicemos la prueba de rendimiento óptimo.

```
C:\WINDOWS\system32\cmd.exe
> Create (OK=4666 KO=0 )
> List (OK=9329 KO=0 )
> List Redirect 1 (OK=9329 KO=0 )
> Create Redirect 1 (OK=4666 KO=0 )
> Created (OK=4193 KO=0 )
> Created Redirect 1 (OK=4193 KO=0 )
> ListMine (OK=7791 KO=0 )
> ListMine Redirect 1 (OK=7791 KO=0 )
> Delete (OK=3374 KO=0 )
> Update (OK=3374 KO=0 )
> Update Redirect 1 (OK=3374 KO=0 )
> Delete Redirect 1 (OK=3374 KO=0 )
> Updated (OK=2893 KO=0 )
> Updated Redirect 1 (OK=2893 KO=0 )
-----
> i.n.c.AbstractChannel$AnnotatedConnectException: Connection re 1433 (100,0%)
fused: no further information: www.dp2.com/127.0.0.1:80
-----
DeleteOpinion -----
[#####] 63%
waiting: 0 / active: 1805 / done: 3195
CreateOpinion -----
[#####] 74%
waiting: 0 / active: 1269 / done: 3731
UpdateOpinion -----
[#####] 43%
waiting: 0 / active: 2805 / done: 2195
-----

object Delete {
  val delete = exec(http("Delete")
    .get("/opinions/7/delete")
    .headers(headers_0))
  .pause(5)
}

val createOpScn = scenario("CreateOpinion").exec(Home.home, Login.login, Logged.logged, Vets.vets, Create.cre
val updateOpScn = scenario("UpdateOpinion").exec(Home.home, Login.login, Logged.logged, Vets.vets, List.list,
val deleteOpScn = scenario("DeleteOpinion").exec(Home.home, Login.login, Logged.logged, Vets.vets, List.list,

setUp(
  createOpScn.inject(rampUsers(5000) during (100 seconds)),
  updateOpScn.inject(rampUsers(5000) during (100 seconds)),
  deleteOpScn.inject(rampUsers(5000) during (100 seconds))
).protocols(httpProtocol)
.assertions(
  global.responseTime.max.lt(5000),
  global.responseTime.mean.lt(1000),
  global.successfulRequests.percent.gt(95)
)
)

----- Global Information -----
> request count 194682 (OK=193249 KO=1433 )
> min response time 0 (OK=0 KO=2001 )
> max response time 3178 (OK=3178 KO=2059 )
> mean response time 54 (OK=39 KO=2027 )
> std deviation 294 (OK=241 KO=10 )
> response time 50th percentile 4 (OK=4 KO=2028 )
> response time 75th percentile 6 (OK=6 KO=2032 )
> response time 95th percentile 13 (OK=11 KO=2044 )
> response time 99th percentile 2027 (OK=1509 KO=2047 )
> mean requests/sec 1093.719 (OK=1085.669 KO=8.051 )
----- Response Time Distribution -----
> t < 800 ms 189772 ( 97%)
> 800 ms < t < 1200 ms 1232 ( 1%)
> t > 1200 ms 2245 ( 1%)
> failed 1433 ( 1%)
----- Errors -----
> i.n.c.AbstractChannel$AnnotatedConnectException: Connection re 1433 (100,0%)
fused: no further information: www.dp2.com/127.0.0.1:80
-----

Reports generated in 0s.
Please open the following file: C:\Users\ijnfu\Desktop\Carrera\Tercero\DP\gatling-charts-highcharts-bundle-3
.3.1\results\testrendimientoopinion-20200525194435759\index.html
Global: max of response time is less than 5000.0 : true
Global: mean of response time is less than 1000.0 : true
Global: percentage of successful events is greater than 95.0 : true
Presione una tecla para continuar . . .

object Delete {
```

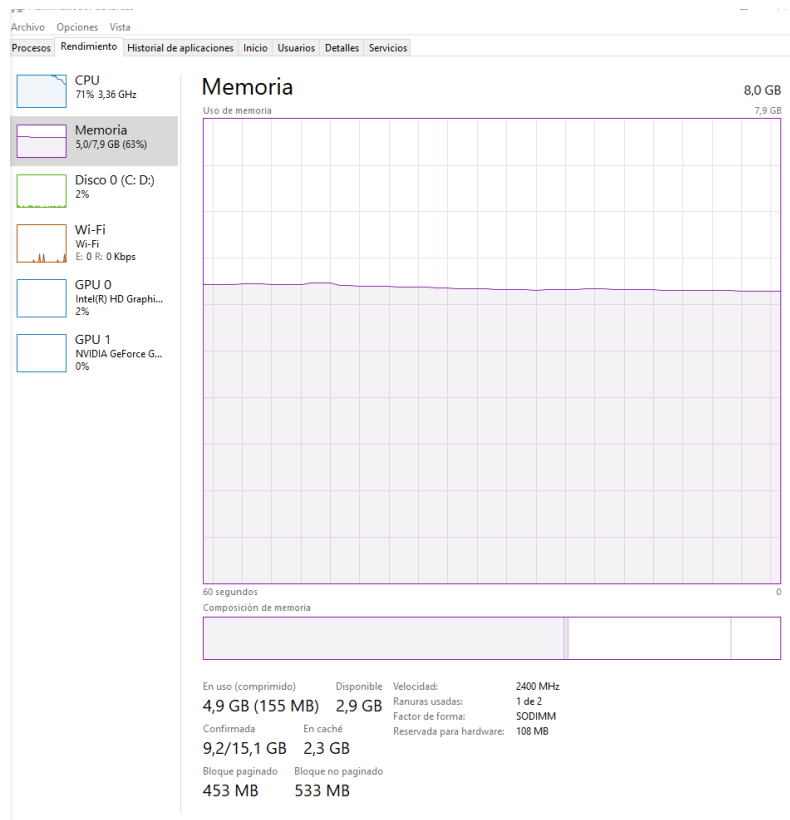
Una vez más los tres escenarios se encuentran en el mismo archivo scala. El total de usuarios es de 15.000 donde cada escenario son 5.000. En esta prueba además tenemos una serie de assertions que se superan correctamente.




Diseño y Pruebas II Documentación del Proyecto PetClinic

Control de Versiones

En cuanto al rendimiento del ordenador.



Observamos que esta vez el procesador no llega al 80% y en general se mantiene sobre el 70%. En cuanto a la memoria, en este caso no llega al 70% y se mantiene sobre el 60%. Este es un rendimiento óptimo de la aplicación con un uso decente del sistema sobre el que está corriendo.

	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

6. Tests de rendimiento de Chips

Las historias de usuario que van a ser probadas en este caso son de la HU-10 a HU-13, es decir, todas las relacionadas con Chip. Para ello, se han creado cuatro escenarios de prueba, para probar cada una de las 4 historias, en el siguiente archivo scala:

```

1 package dp2
2
3 import scala.concurrent.duration._
4
5 import io.gatling.core.Predef._
6 import io.gatling.http.Predef._
7 import io.gatling.jdbc.Predef._
8
9 class ChipPerformance extends Simulation {
10
11     val httpProtocol = http
12         .baseUrl("http://www.dp2.com")
13         .inferHtmlResources(BlackList(""".*.png""", """.*.css""", """.*.ico""", """.*.js"""), WhiteList())
14         .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,"+
15             "image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9")
16         .acceptEncodingHeader("gzip, deflate")
17         .acceptLanguageHeader("es-ES,es;q=0.9,en;q=0.8")
18         .userAgentHeader("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)"+
19             "Chrome/81.0.4044.138 Safari/537.36")
20
21     val headers_0 = Map(
22         "Proxy-Connection" -> "keep-alive",
23         "Upgrade-Insecure-Requests" -> "1")
24
25     val headers_1 = Map(
26         "Origin" -> "http://www.dp2.com",
27         "Proxy-Connection" -> "keep-alive",
28         "Upgrade-Insecure-Requests" -> "1")
29
30     val headers_2 = Map(
31         "Proxy-Connection" -> "keep-alive")
32
33     val headers_3 = Map(
34         "Origin" -> "http://www.dp2.com",
35         "Proxy-Connection" -> "keep-alive",
36         "Upgrade-Insecure-Requests" -> "1")

```





```
35
36     val headers_6 = Map(
37         "Proxy-Connection" -> "Keep-Alive",
38         "User-Agent" -> "Microsoft-WNS/10.0")
39
40     val headers_11 = Map(
41         "A-IM" -> "x-bm,gzip",
42         "Proxy-Connection" -> "keep-alive")
43
44     val uri1 = "http://clientservices.googleapis.com/chrome-variations/seed"
45
46     object Home {
47         val home = exec(http("Home")
48             .get("/")
49             .headers(headers_0))
50         .pause(9)
51     }
52
53     object Login {
54         var login = exec(
55             http("Login")
56                 .get("/login")
57                 .headers(headers_0)
58                 .check(css("input[name=_csrf]", "value").saveAs("stoken")))
59         .pause(18)
60         .exec(
61             http("Logged")
62                 .post("/login")
63                 .headers(headers_3)
64                 .formParam("username", "admin1")
65                 .formParam("password", "4dm1n")
66                 .formParam("_csrf", "${stoken}")
67         ).pause(9)
68     }
69
```



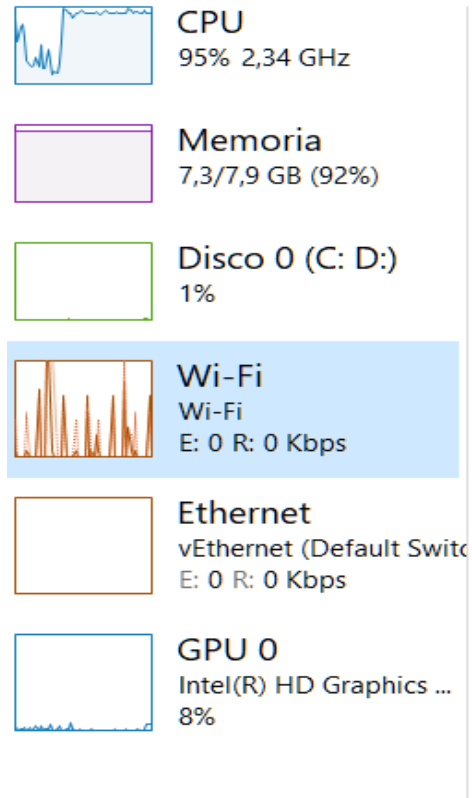
```
70     object FindOwner {
71         val findOwner = exec(http("FindOwner")
72             .get("/owners/find")
73             .headers(headers_0))
74         .pause(16)
75     }
76
77     object ListOwners {
78         val listOwners = exec(http("ListOwners")
79             .get("/owners?lastName=")
80             .headers(headers_0))
81         .pause(8)
82     }
83
84     object Owner {
85         val owner = exec(http("Owner")
86             .get("/owners/5")
87             .headers(headers_0))
88         .pause(12)
89     }
90
91     object AddChip {
92         val addChip = exec(
93             http("AddChip")
94                 .get("/owners/5/pets/6/chips/new")
95                 .headers(headers_0)
96                 .check(css("input[name=_csrf]", "value").saveAs("stoken")))
97         .pause(19)
98         .exec(
99             http("ChipAdded")
100                 .post("/owners/5/pets/6/chips/new")
101                 .headers(headers_3)
102                 .formParam("pet_id", "6")
103                 .formParam("serialNumber", "123")
104                 .formParam("model", "model123")
105                 .formParam("geolocatable", "true")
106                 .formParam("_csrf", "${stoken}")
107             ).pause(24)
```

Primero, se realiza una prueba de estrés, con los siguientes valores:

```
175     setUp(
176         createScn.inject(rampUsers(200) during (10 seconds)),
177         updateScn.inject(rampUsers(200) during (10 seconds)),
178         showScn.inject(rampUsers(200) during (10 seconds)),
179         deleteScn.inject(rampUsers(200) during (10 seconds))
180     ).protocols(httpProtocol)
181     .assertions(
182         global.responseTime.max.lt(5000),
183         global.responseTime.mean.lt(5000),
184         global.successfulRequests.percent.gt(95)
185     )
186 }
```

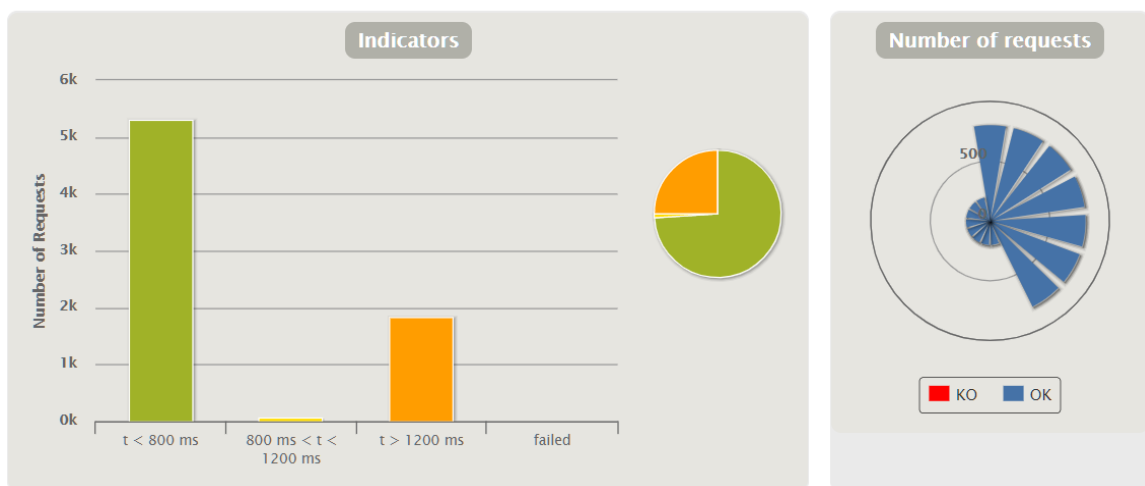
	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

Para esta situación, el portátil que se está usando para realizar estas pruebas se encuentra en el siguiente estado:



Los resultados obtenidos son los siguientes:

> **Global Information**



ASSERTIONS	
Assertion	Status
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 5000.0	OK
Global: percentage of successful events is greater than 95.0	OK




Diseño y Pruebas II Documentación del Proyecto PetClinic

Control de Versiones

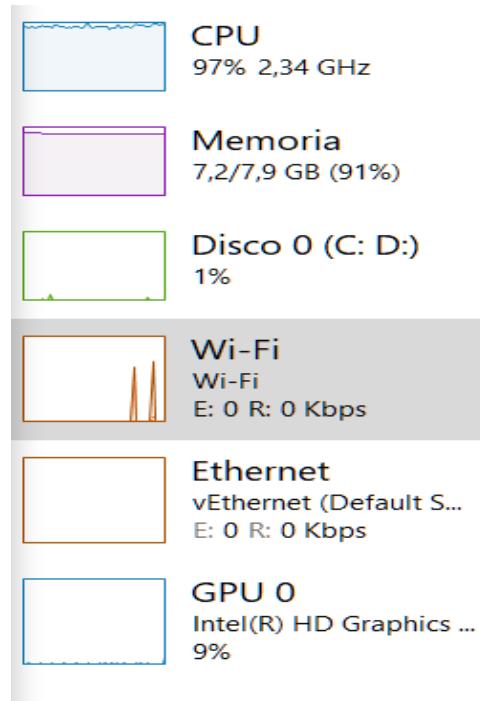
STATISTICS Expand all groups Collapse all groups													
Requests ^	Executions					Response Time (ms)							
	Total ↕	OK ↕	KO ↕	% KO ↕	Cnt/s ↕	Min ↕	50th pct ↕	75th pct ↕	95th pct ↕	99th pct ↕	Max ↕	Mean ↕	Std Dev ↕
Global Information	7200	7200	0	0%	44.172	0	11	1411	19332	27965	40658	3085	6589
Home	800	800	0	0%	4.908	2	4	5	27	312	421	13	45
Login	800	800	0	0%	4.908	0	1	2	3	5	19	2	1
Logged	800	800	0	0%	4.908	4	13	73	482	668	1083	85	159
Logged Redirect 1	800	800	0	0%	4.908	1	3	4	7	14	29	3	3
FindOwner	800	800	0	0%	4.908	1	2	3	5	8	15	2	1
ListOwners	800	800	0	0%	4.908	464	11987	19311	28012	34373	40658	13049	8338
Owner	800	800	0	0%	4.908	37	13456	17552	26365	33920	39920	13488	7128
AddChip	200	200	0	0%	1.227	1	3	9	2038	3377	4835	293	754
UpdateChip	200	200	0	0%	1.227	12	524	2006	4659	6831	11892	1252	1704
ChipDeleted	200	200	0	0%	1.227	15	482	1763	4876	6233	10563	1229	1681
ShowChip	200	200	0	0%	1.227	13	589	2250	5182	6603	10310	1454	1840
ChipAdded	200	200	0	0%	1.227	27	42	59	96	165	241	51	28
ChipAdde...direct 1	200	200	0	0%	1.227	26	45	59	110	185	217	53	31
ChipUpdated	200	200	0	0%	1.227	29	55	77	143	174	217	66	36
ChipUpda...direct 1	200	200	0	0%	1.227	27	54	100	203	260	287	79	58

Una vez visto el rendimiento óptimo, se buscan los cuellos de botella. Para ello, se realizan las pruebas con los siguientes valores:

```
setUp(  
    createScn.inject(rampUsers(1000) during (10 seconds)),  
    updateScn.inject(rampUsers(1000) during (10 seconds)),  
    showScn.inject(rampUsers(1000) during (10 seconds)),  
    deleteScn.inject(rampUsers(1000) during (10 seconds))  
).protocols(httpProtocol)  
.assertions(  
    global.responseTime.max.lt(5000),  
    global.responseTime.mean.lt(5000),  
    global.successfulRequests.percent.gt(95)  
)
```

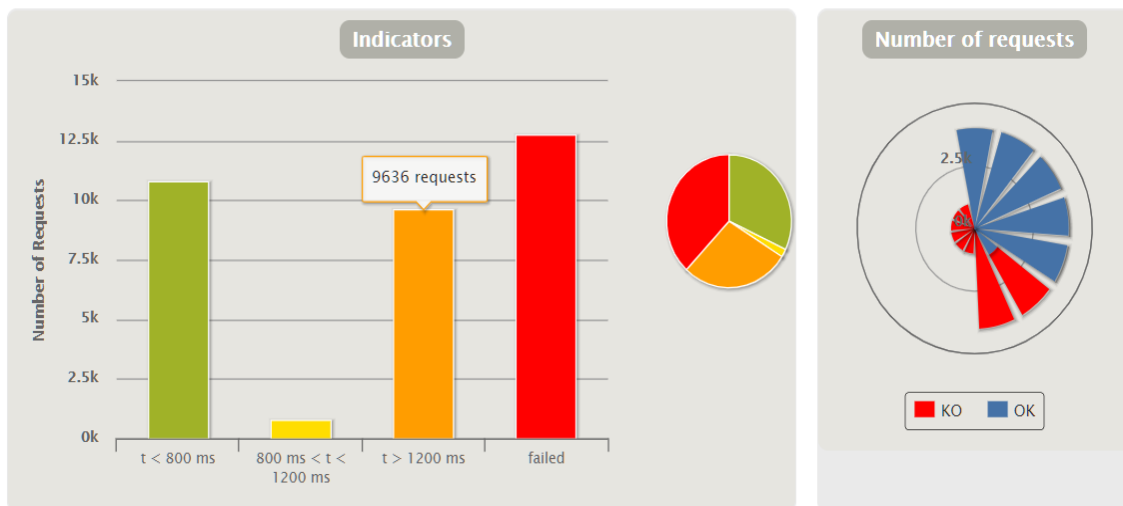
	Diseño y Pruebas II Documentación del Proyecto PetClinic
	Control de Versiones

Para esta situación, el portátil que se está usando para realizar estas pruebas se encuentra en el siguiente estado:



Los resultados obtenidos son los siguientes:

> Global Information



▶ ASSERTIONS	
Assertion ↕	Status ↕
Global: max of response time is less than 5000.0	KO
Global: mean of response time is less than 5000.0	KO
Global: percentage of successful events is greater than 95.0	KO



Diseño y Pruebas II
Documentación del Proyecto PetClinic

Control de Versiones

STATISTICS														Expand all groups Collapse all groups	
Requests ^	Executions					Response Time (ms)									
	Total ↕	OK ↕	KO ↕	% KO ↕	Cnt/s ↕	Min ↕	50th pct ↕	75th pct ↕	95th pct ↕	99th pct ↕	Max ↕	Mean ↕	Std Dev ↕		
Global Information	34000	21213	12787	38%	90.186	1	5157	60000	60002	60003	60058	22269	27168		
Home	4000	4000	0	0%	10.61	3	174	767	3274	3900	4249	593	928		
Login	4000	4000	0	0%	10.61	1	43	99	215	940	1550	78	138		
Logged	4000	4000	0	0%	10.61	6	5422	7123	9471	12063	16000	5348	2651		
Logged Redirect 1	4000	4000	0	0%	10.61	1	5230	6868	7550	7621	7645	4772	2269		
FindOwner	4000	4000	0	0%	10.61	1	10	163	6921	7580	7640	926	2039		
ListOwners	4000	1210	2790	70%	10.61	113	60001	60001	60002	60004	60017	51223	16412		
Owner	4000	3	3997	100%	10.61	53807	60001	60001	60003	60011	60058	59998	124		
ShowChip	1000	0	1000	100%	2.653	2118	60000	60001	60002	60004	60036	45184	25193		
UpdateChip	1000	0	1000	100%	2.653	2111	60000	60001	60002	60004	60017	44318	25658		
AddChip	1000	0	1000	100%	2.653	2109	60000	60001	60002	60004	60013	44146	25744		
ChipDeleted	1000	0	1000	100%	2.653	2114	60000	60001	60002	60004	60011	44433	25599		
ChipAdded	1000	0	1000	100%	2.653	2011	60001	60001	60002	60006	60010	54624	16791		
ChipUpdated	1000	0	1000	100%	2.653	2002	60000	60001	60002	60005	60013	32703	28815		

ERRORS		
Error ↕	Count ↕	Percentage ↕
i.g.h.c.i.RequestTimeoutException: Request timeout to www.dp2.com/127.0.0.1:80 after 60000 ms	11147	87.174 %
i.n.c.AbstractChannel\$AnnotatedConnectException: Connection refused: no further information: www.dp2.com/127.0.0.1:80	1640	12.826 %