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.

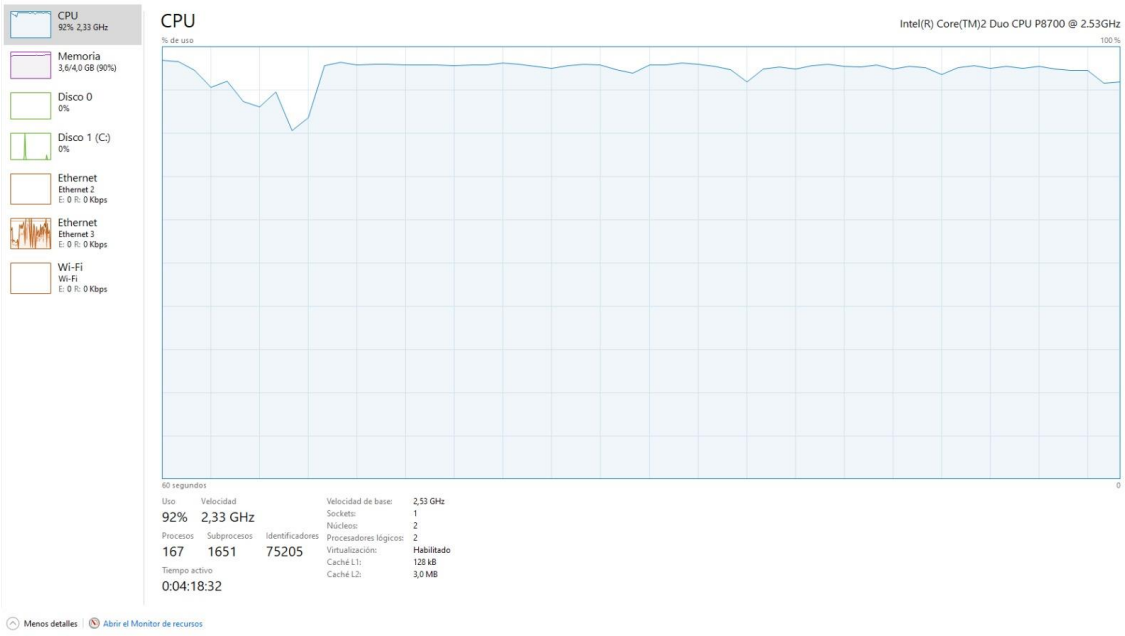
Se ha modificado el código siguiendo las pautas de los vídeos, para su correcta ejecución quedando el siguiente código resultante:

```
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("stoken"))
        ).pause(18)
        .exec(
            http("Logged")
            .post("/login")
            .headers(headers_3)
            .formParam("username", "admin1")
            .formParam("password", "4dm1n")
            .formParam("_csrf", "${stoken}")
        ).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("stoken"))
        ).pause(27)
        .exec(http("RoomUpdated")
            .post("/rooms/1/edit")
            .headers(headers_3)
            .formParam("name", "updated")
            .formParam("floor", "1")
            .formParam("medicalTeam", "nadena")
            .formParam("_csrf", "${stoken}")
        ).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 este test:



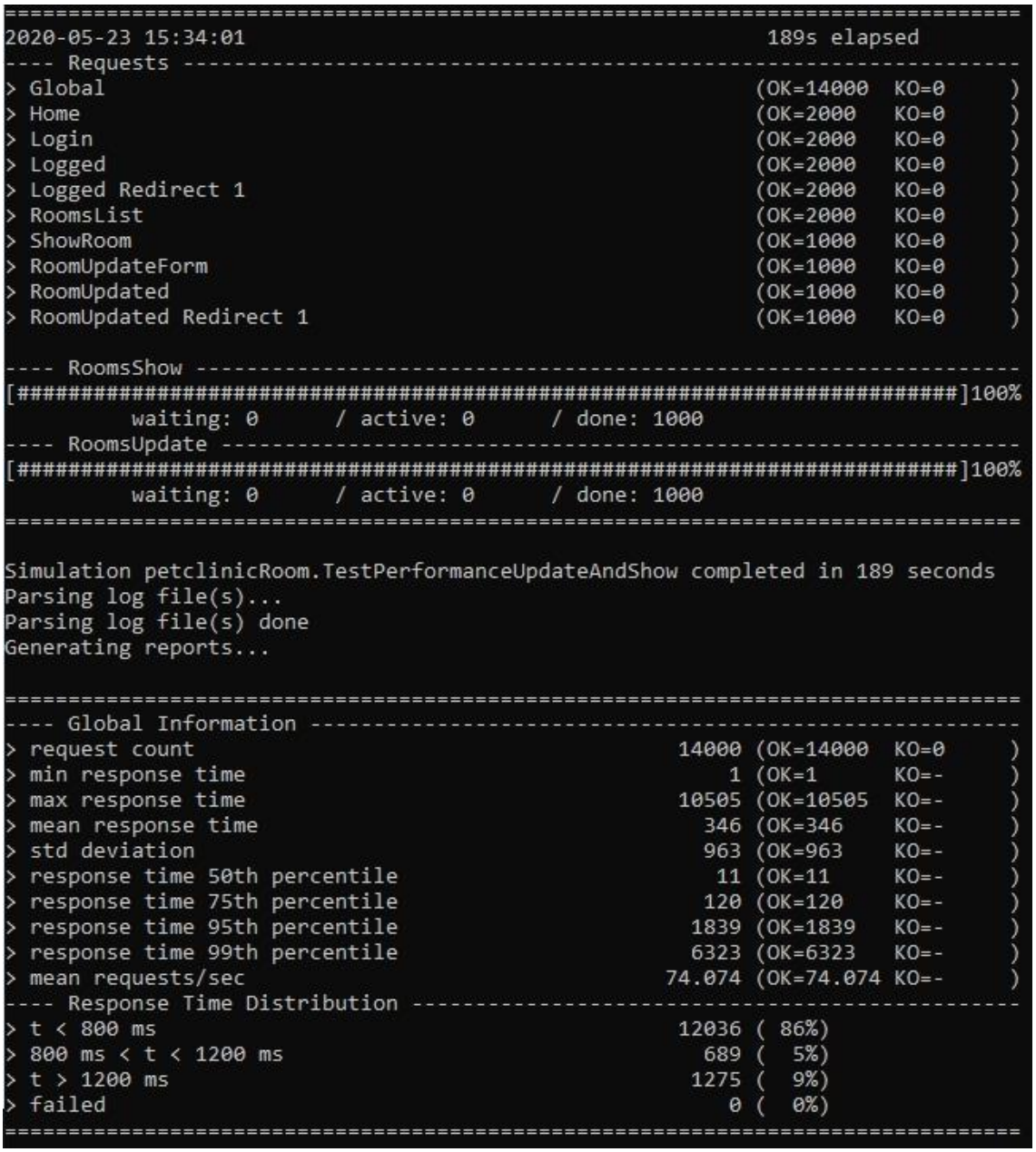
En este momento es cuando nos encontramos ejecutando 1000 usuarios activos.

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

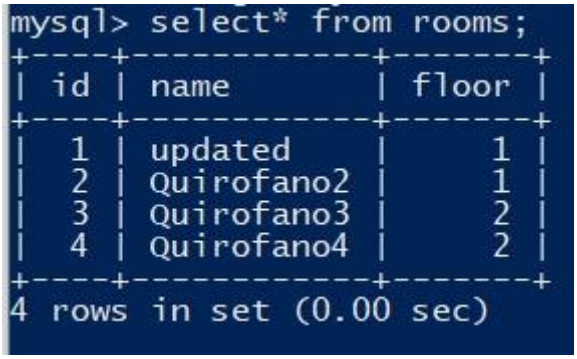
```
================================================================================
2020-05-23 15:33:32                                           160s elapsed
---- Requests ------------------------------------------------------------------
> Global                                                   (OK=13686  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
================================================================================
```

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:

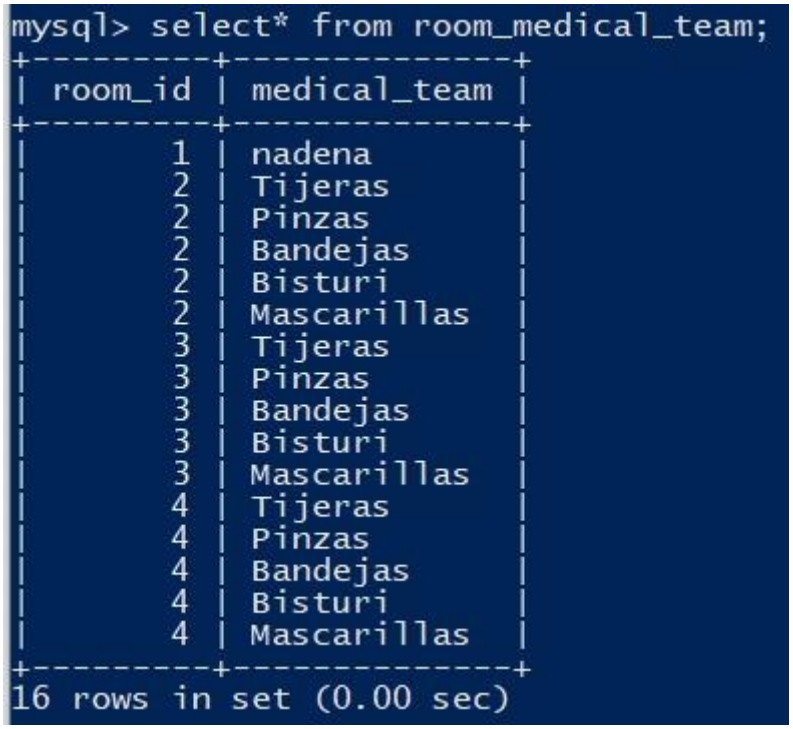


Una vez finalizado el test decidimos comprobar si realmente se había actualizado en **docker**:



```
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)
```

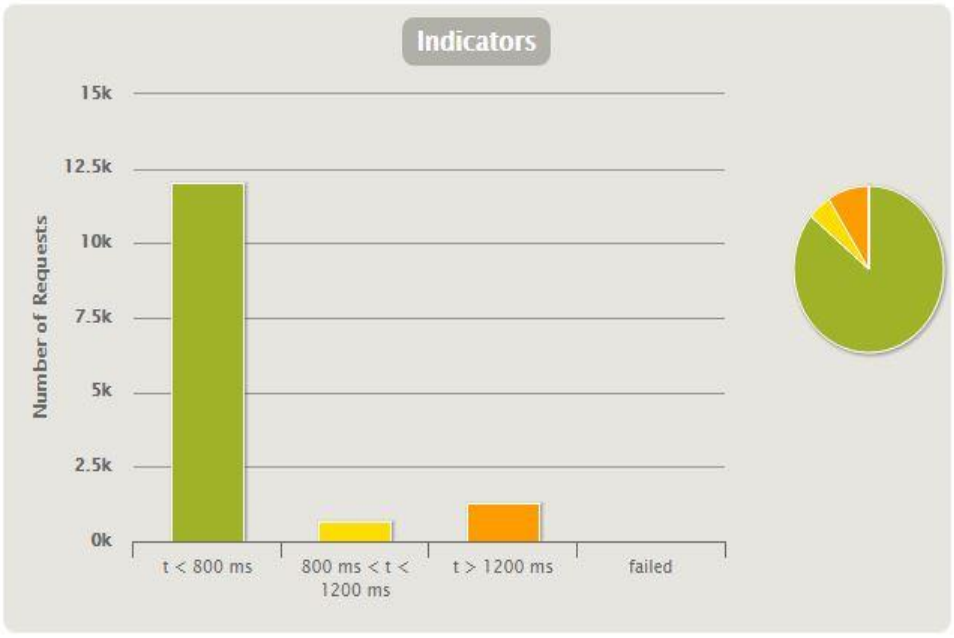
Se ha comprobado que también ha sido actualizado su equipo médico que es el que tiene la id 1.

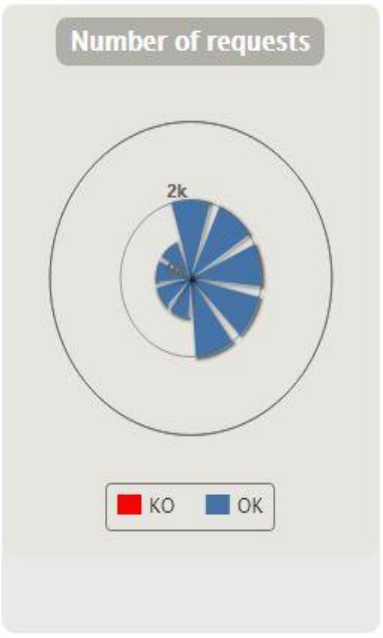
Podemos apreciar que ha cumplido las restricciones que se le puso:

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:





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.