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 borrase 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",
```

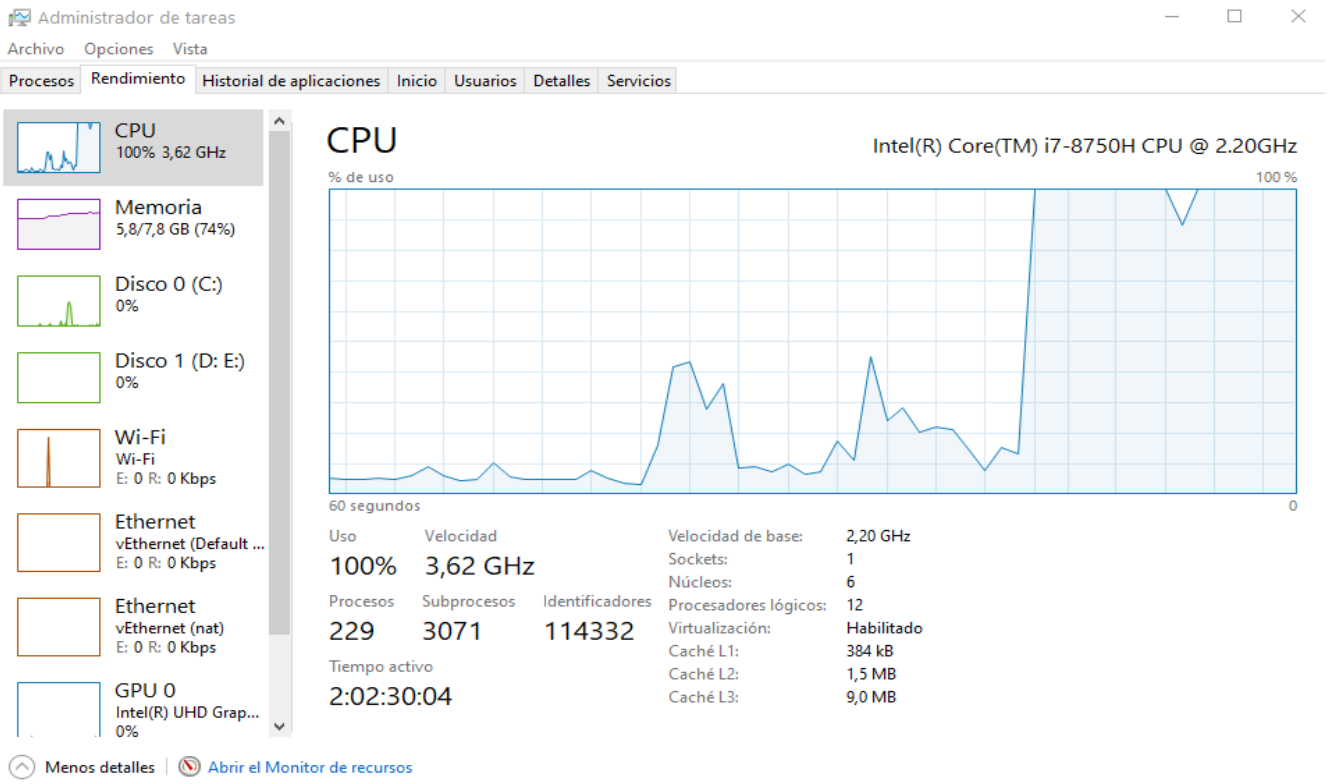
```
        "Upgrade-Insecure-Requests" -> "1")

    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("CreateDiseaseFormPetOwn
er1")
            .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", "${stoken}"))
    .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)
     )
}
```

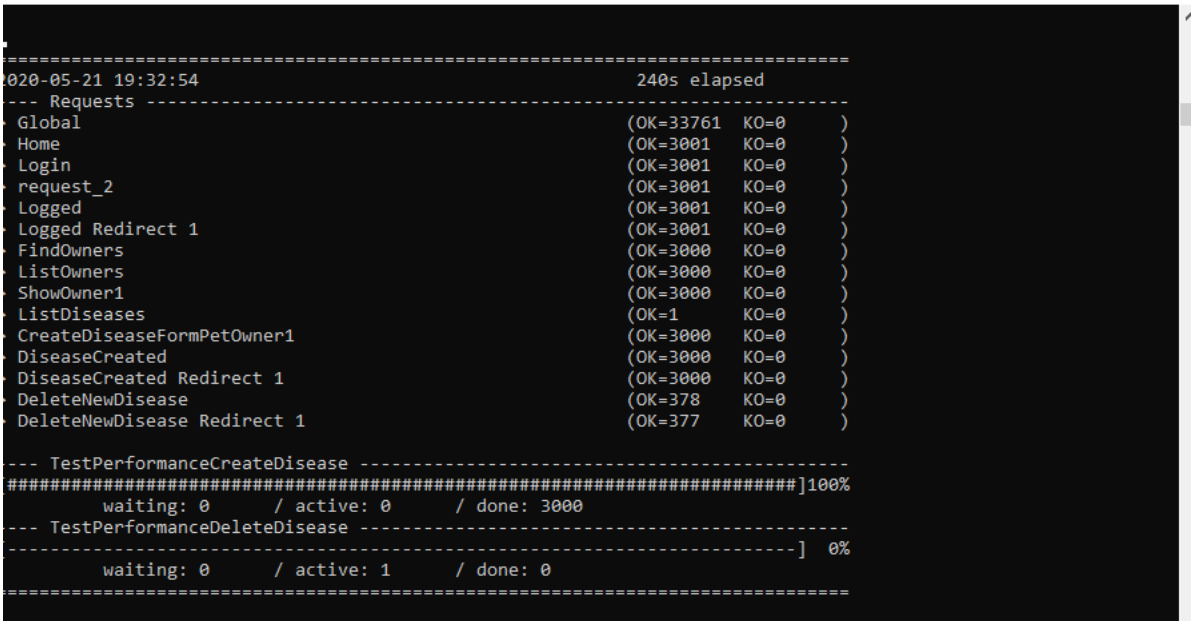
Se puede ver que el tiempo global ha sido sustituido por el máximo de peticiones falladas que no deben superar el 5%.

Ahora mostrare el avance que ha tenido ejecutar este test:



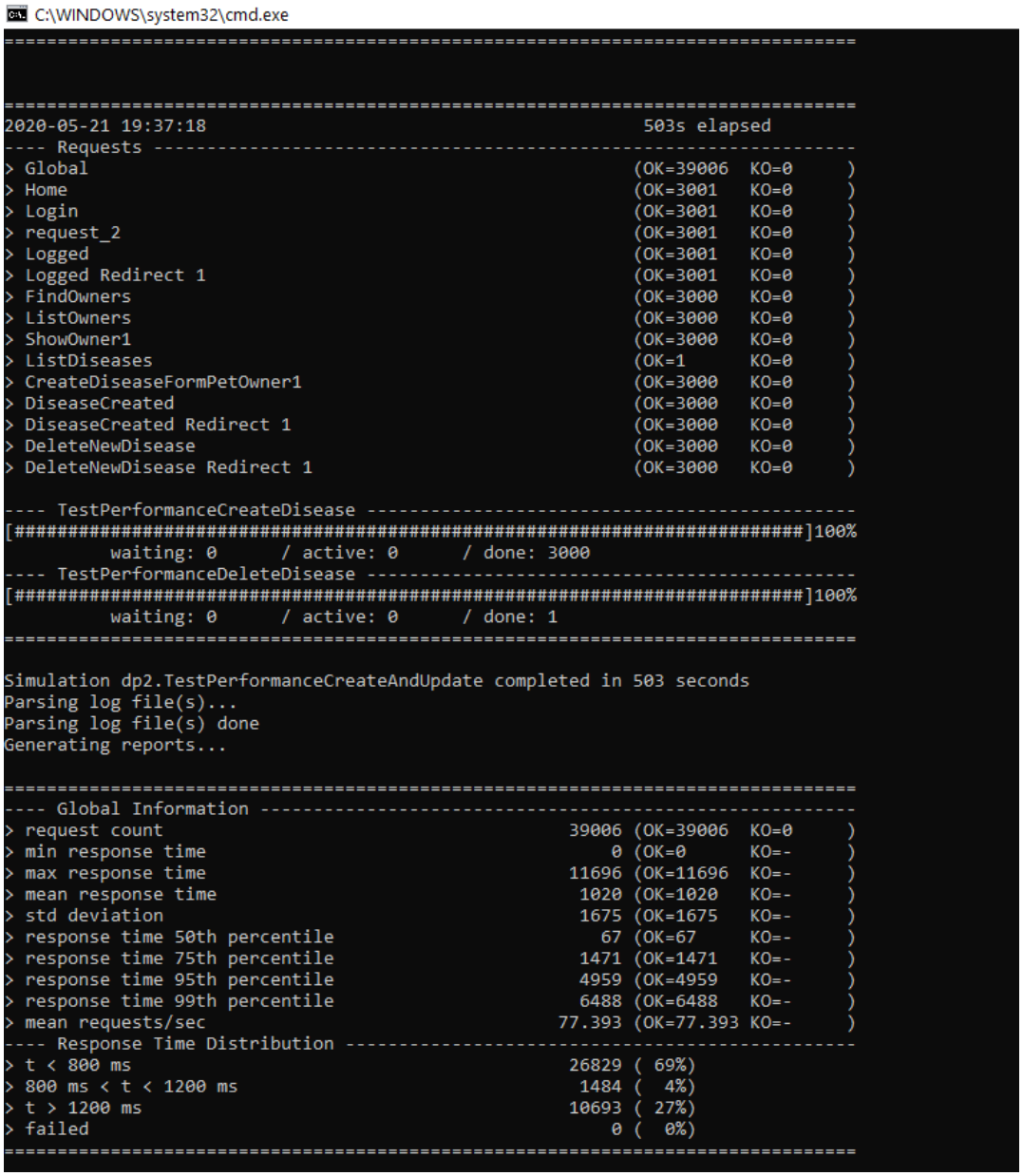
En este momento es cuando nos encontramos ejecutando 1050 usuarios activos.

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

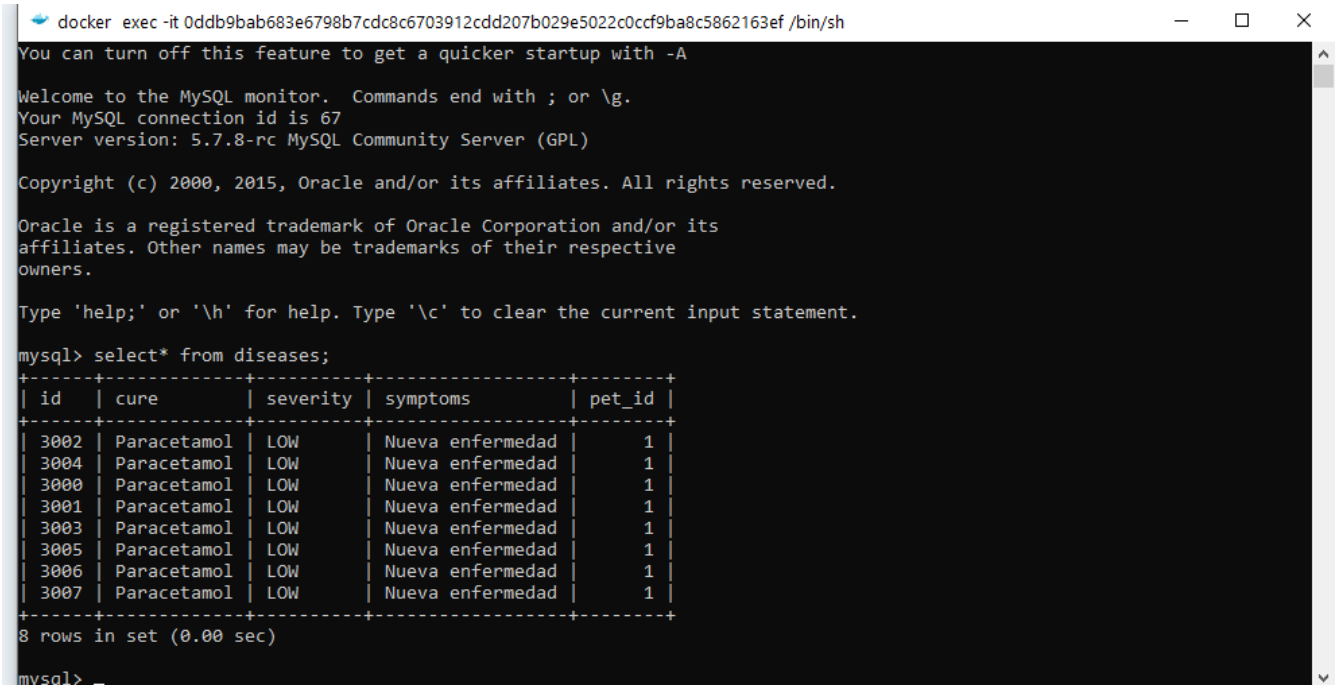


Vemos que con 3000 usuarios no ha habido ninguna petición fallida, en cambio el encargado de borrar todavía está realizando su tarea. En estos momentos la cpu se ha aliviado un poco.

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



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



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.

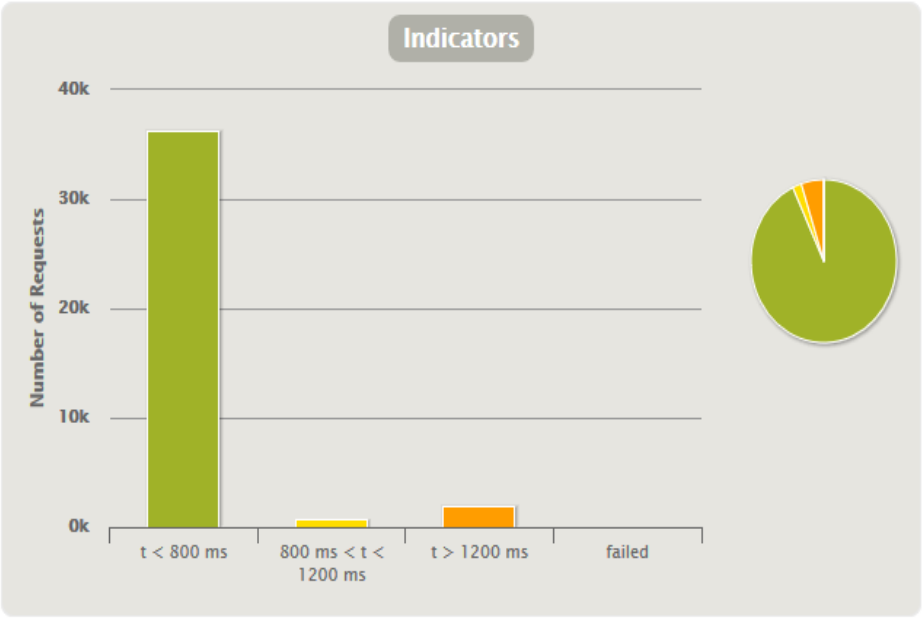
Por lo tanto, ha borrado bien los elementos que se han ido creando.

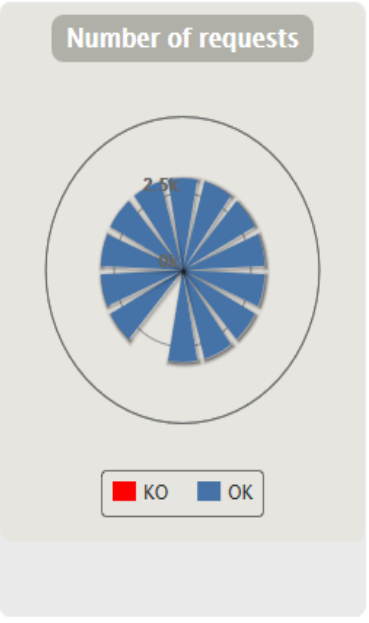
También podemos apreciar que ha cumplido las restricciones que le puse:

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:





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.

Se ha intentado realizar una captura con un máximo de usuarios en el que el sistema empezase a fallar, pero en ese momento el ordenador se quedo congelado, no pudiendo hacer nada.