Nombre:Carlos Albalat Heredia Grupo: G3-13

Informe individual de actividades del proyecto

# Datos generales

### URL del Repositorio de GitHub: <https://github.com/cramersito/DP2-Repository>

Nombre de usuario en GitHub: cramersito

Participación en el proyecto

He implementado de la HU-6 a la HU-13, todo lo relacionado con las entidades Curso, Seguro y Certificado, es decir, desarrollar modelo,servicio,controlador y repositorio de estas entidades. También añadí todo lo relacionado a la clase Actor, Clerk, para poder comprobar el actor logeado en ese momento en la página etc... El esquema y populate de la base de datos fue realizado de manera conjunta con Jesús. En cuanto a las pruebas he testeado lo que ha desarrollado Jesús, es decir, todas las pruebas empezando por las unitarias y terminando por las de rendimiento de todas las HU relacionadas con la entidad Transporte. Mencionar que también he desarrollado la autenticación del proyecto, tocando los archivos SecurityConfig.java, MyUserDetailsService.java y HomeController.java.

Historias de usuario en las que he participado

He implementado de la HU-6 a la HU-13, ambas inclusive.

Funcionalidad implementada

He implementado los controladores CertificateController, CourseController, InsuranceController, ClerkController y HomeController(Autenticacion).

He añadido el método public void enroll() a la clase OwnerService. He implementado los servicios CertificateService, CourseService e InsuranceService.

Las entidades las hemos creado a medias como el populate y el esquema de la base de datos.

Las vistas que he realizado se encuentran dentro de los directorios src/main/resources/templates/certificate,course,insurance. Y las vistas de login y cerrar sesión.

## Pruebas implementadas

### Pruebas unitarias

He creado test unitarios para 3 servicios(TransportService,OwnerService y ActorService), 3 controladores(TransportController,OwnerController y ClerkController), 3 repositorios(TransportRepository,OwnerRepository,ClerkRepository). Eso hace un total de 9 clases de test unitarios con un total de 27 métodos con @Test.

Pruebas de interfaz de usuario

He creado un caso de prueba positivo por la HU-1, HU-3, HU-4, HU-5. No se ha realizado casos negativos debido a la naturaleza de las historias de usuario (listados,displays...)

Pruebas end-to-end en los controladores de la aplicación (si aplica)

No aplica. Aunque se han realizado una serie de pruebas E2E de manera conjunta para realizar pruebas sobre el correcto funcionamiento del sistema de principio a fin obteniendo así una visión global del proyecto. Hemos implementado las clases CertificateControllerE2ETest y CourseControllerE2ETest.

Pruebas de integración con un servicio externo (si aplica)

No aplica.

Pruebas de rendimiento

Profiling de código (si aplica)

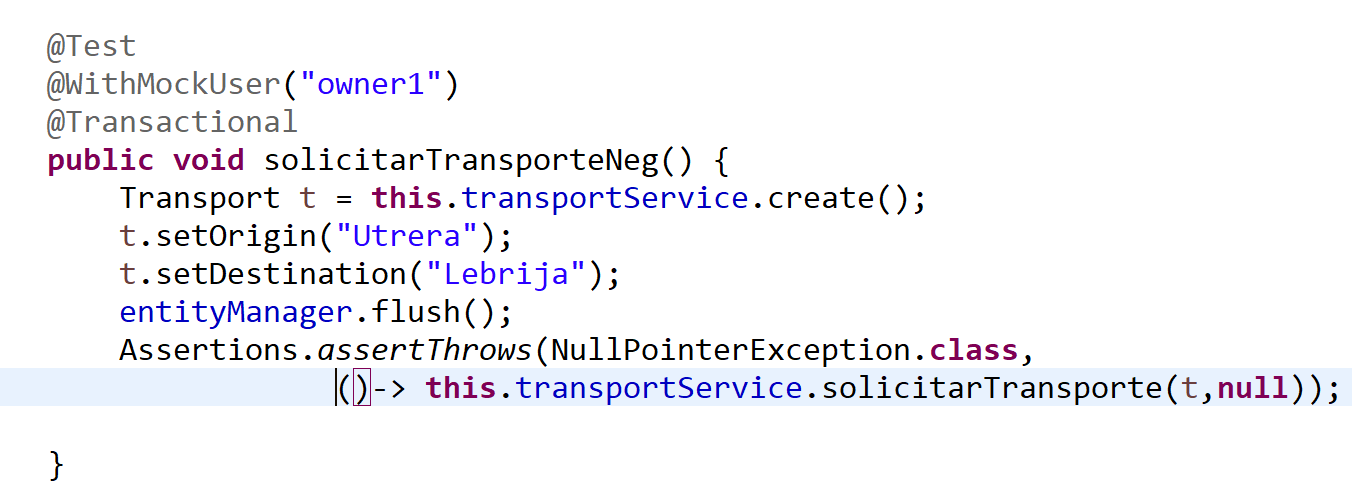
No aplica

Refactorizaciones (si aplica)

No aplica

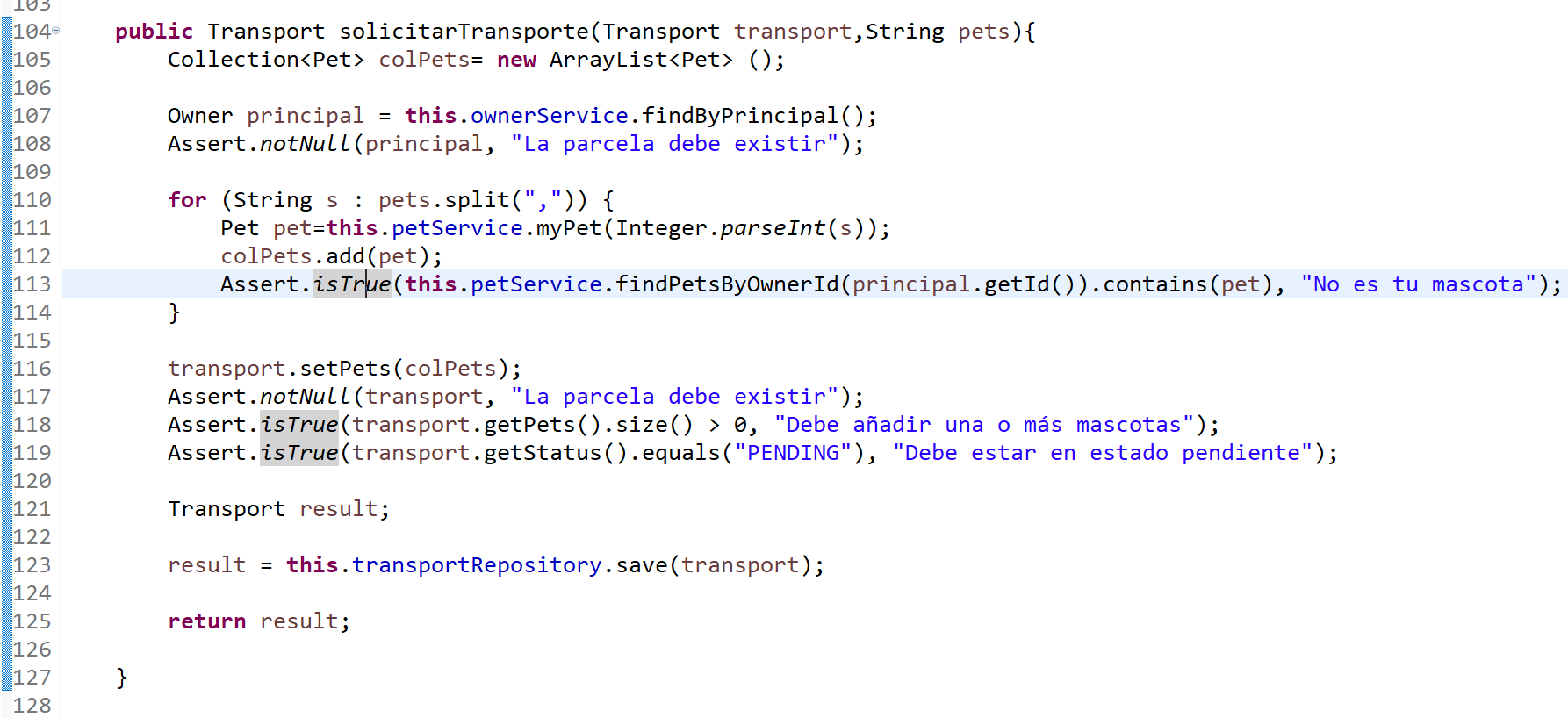
Ejemplos de pruebas implementadas

## Pruebas unitarias (máximo de dos ejemplos)



Aquí se comprueba un caso negativo del método solicitarTranposrte(Transport transport, String pets) de la clase TransportService.

* La sección *Arrange* de esta prueba consiste en simular un contexto de autenticación del usuario “Owner1” bajo el rol “OWNER” y crear un transporte mediante el método Create de la clase TransportService, añadiendole los atributos Origen y Destino posteriormente, preparando lo necesario para ejecutar el test.
* En la etapa *Act* se realiza una llamada al método a testear, en este caso, solicitarTranportes, al cual se le pasa un parámetro nulo(String pets), dando error en la siguiente porción de código de la clase TransportService. En la línea 110 al realizar la operación pets.split(“,”) se lanza una excepción NullPointer recogida en la sección Assert de este test.

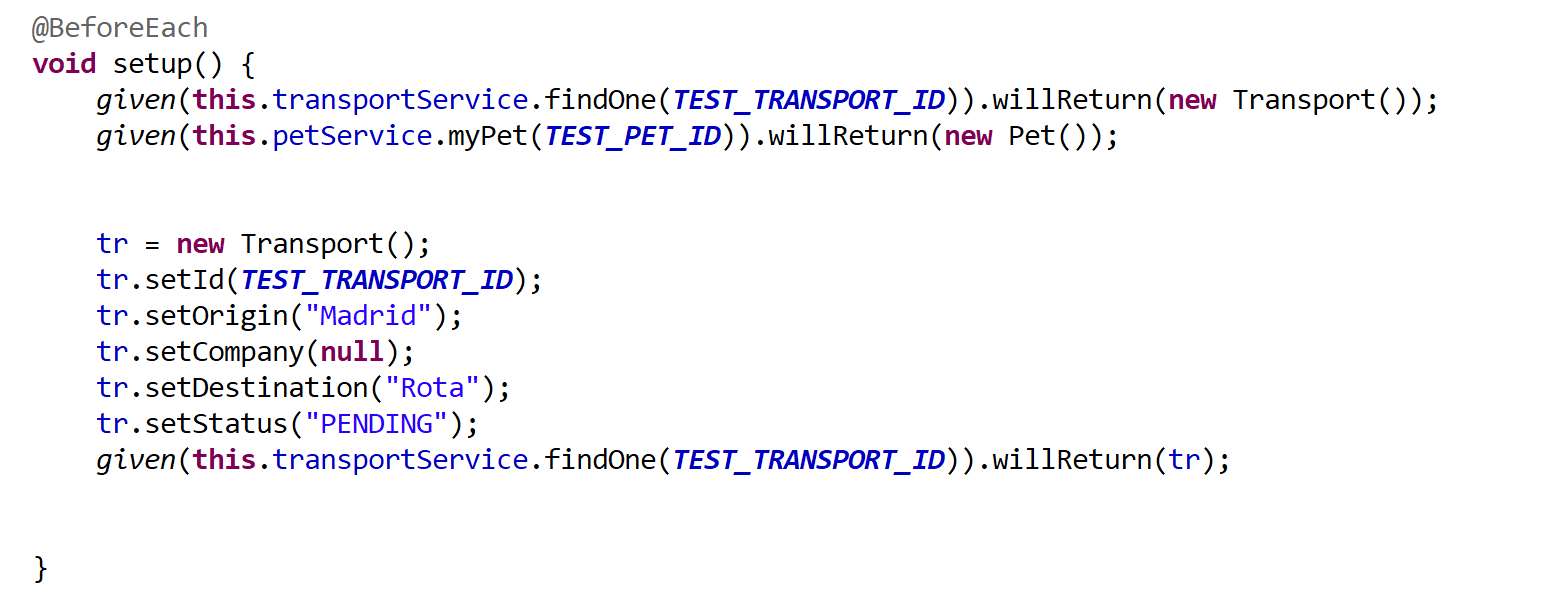


* Por último, en *Assert* se comprueba que efectivamente el método lanza una excepción de tipo NullPointerException al intentar acceder a las mascotas que son nulas. Comportándose el sistema como debería.



Aquí se comprueba un caso negativo donde el método Post solicitarTransporte del controlador TransportController no se realiza con éxito ya que el origen y el estado está en blanco.

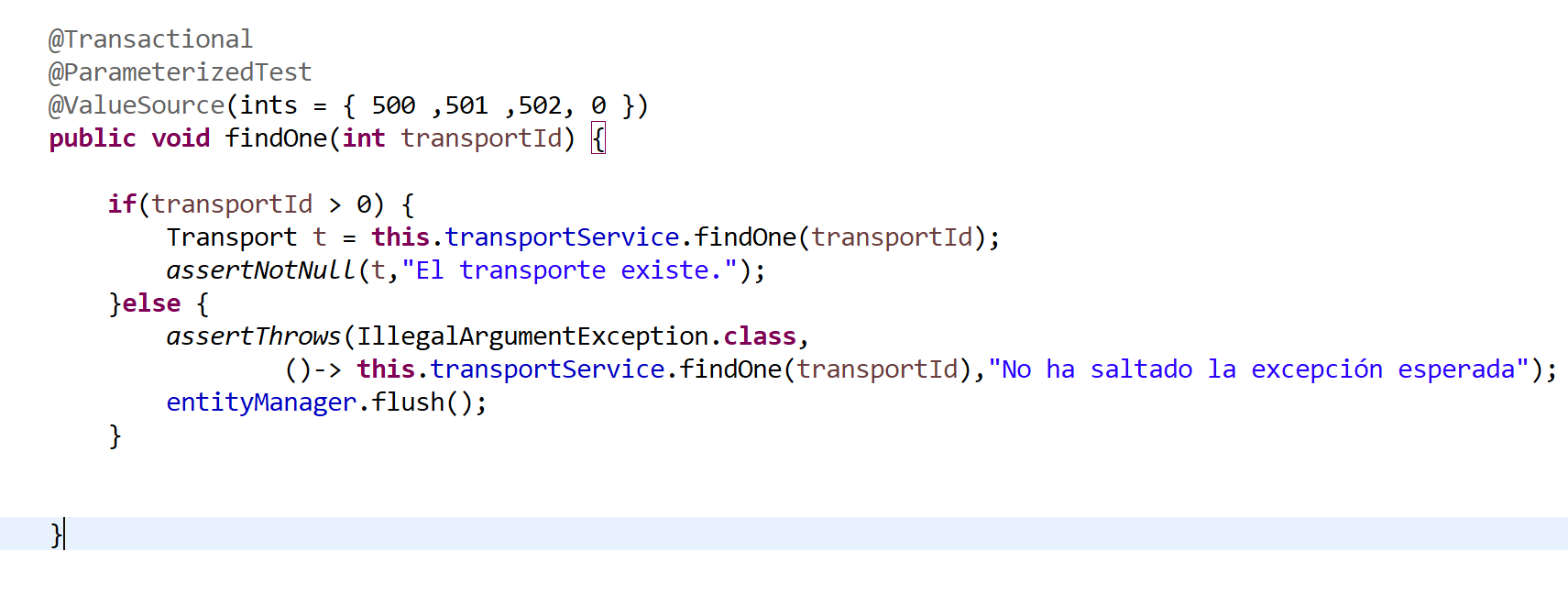
* La sección *Arrange* de esta prueba consiste en simular un contexto de autenticación del usuario “Owner1” bajo el rol “OWNER” mediante la anotación @withMockUser y además el siguiente setUp. Donde al principio de cada test de esta clase se inicializa todo lo necesario para poder llevarlo a cabo.



* En la etapa *Act* se simula el formulario que un usuario ha podido rellenar, pero sin origen y destino.
* Por último, en *Assert* se verifica que efectivamente el atributo origen y status de la entidad Transporte contiene errores (en este caso de persistencia debido a los tags proporcionados por Spring). Como consecuencia de esto, el controlador actúa como debería enviándote a la misma vista señalando los errores, para poder realizar el transporte de nuevo con todos los campos de manera correcta.

## Pruebas unitarias parametrizadas (si aplica)

Las pruebas unitarias siguientes se encuentran en el directorio src\test\java\com\DP2Spring\test\controllers y src\test\java\com\DP2Spring\test\services .



En primer lugar, este test comprueba que la base de datos está populada correctamente, comprobando que esos identificadores se encuentran en ella.

* La sección *Arrange* de este test se encuentra en la etiqueta @ValueSource, donde se pasan varios identificadores como parámetros para realizar el test sobre éstos mismos (Primero se comprueba si existe el transporte con identificador 500, después 501...).
* En la etapa *Act* se comprueba el método findOne de la clase TransportService para encontrar un transporte dado su identificador en la base de datos. Comprobando que efectivamente no es nulo y existe.
* Por último, en *Assert* se comprueba que efectivamente si el identificador es menor que cero debe saltar una excepción, contemplada en este Assert, es decir, no existe ningún transporte con identificador 0.

## Pruebas de integración con la base de datos

Cabe mencionar que al principio del proyecto se tomó la decisión de usar la base de datos MySQL por lo tanto al realizar cualquier prueba estaría probando dicha integración. Para cerciorarnos, he realizado los test de integración de la siguiente clase:TransportControllerIntegrationTest. Estas se encuentran el directorio src\test\java\com\DP2Spring\test\web\integration.

En TransportControllerIntegrationTest encontramos el siguiente test:



En primer lugar, este test comprueba el correcto funcionamiento del sistema a la hora de solicitar un transporte de animales a nivel de controlador sin mockear, es decir, consultando a las clases Servicio reales y por tanto a los repositorios reales accediendo a nuestra base de datos MySQL directamente.

* La sección *Arrange* de esta prueba consiste en simular un contexto de autenticación del usuario “Owner1” bajo el rol “OWNER”. También se incializa un transporte seteandole los atributos Origen y Destino. Acto seguido se comprueba en el BindingResult que no hay no problemas de validación. Recordar que a diferencia de los test unitarios(mockeados) de controladores, no hay un setUp con la anotacion @BeforeEach ya que accedemos a la base de datos real.
* En la etapa *Act* se realiza una llamada al método a testear, solicitarTransporte, pasándole por parámetro el transporte creado y el binding creado, ambos en la sección Arrange, junto a dos identificadores de animales en tipo String.
* Por último, en *Assert* se comprueba que efectivamente te redirige a la vista my-pets, ya que el método se ha realizado de manera satisfactoria, en caso contrario, volverías al formulario indicando los errores que obtuviste.

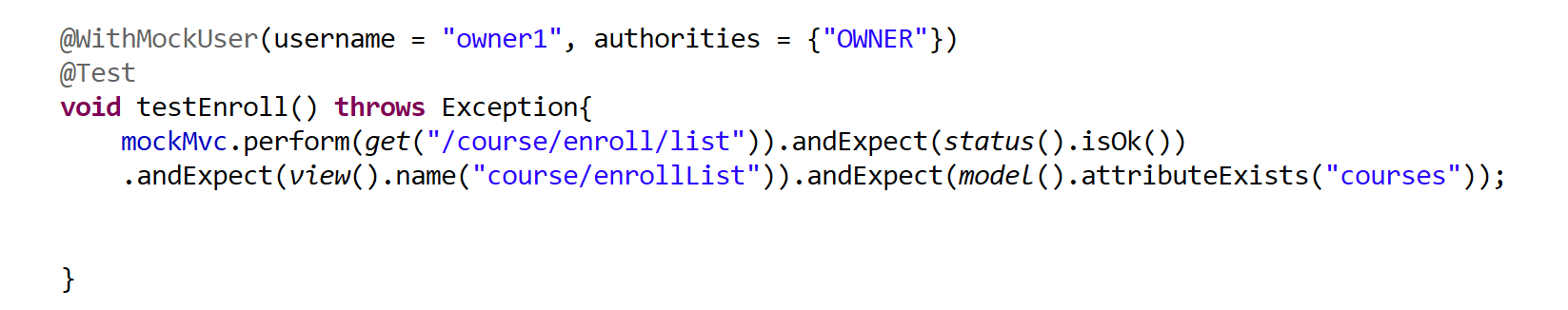
## Pruebas de interfaz de usuario

## 

En este test de interfaz de usuario se simula la interacción de un usuario el cual quiere cumplir la HU3, es decir, ver de manera sencilla aquellos transportes pendientes. Dicho de otra forma, ver aquellos animales que tienen un transporte pendiente. Haciendo uso de la herramienta Katalon se logea como clerk1, haciendo click en el enlace Transportes Pendientes del menú. Comprobando por último que el elemento h2 de la página se corresponde a Transportes Pendientes. Siendo este un test completamente satisfactorio.

Pruebas end-to-end en los controladores de la aplicación (si aplica)

Señalar que nosotros no teníamos que realizar este tipo de test debido a que vamos a por el nivel 6. En cambio, visto que era casi idéntico a los unitarios de controladores, la única diferencia es que se accede directamente a los servicios( no son mockeados), se han realizado varias pruebas como por ejemplo esta:



Pruebas de integración con un servicio externo (si aplica)

No aplica.

Pruebas de rendimiento

Esto es un extracto del archivo solicitarTranporte2Escenarios. En éste se puede comprobar que se enfrenta un escenario positivo, en el cual todos los datos son correctos para realizar dicho transporte, frente a uno negativo en el cual no tiene ni origen ni destino. Por lo tanto, debe saltar la excepción correspondiente al tag @NotBlank implementado en el modelo de la clase Transport.

He desarrollado las pruebas de rendimiento para las HU de Transporte desarrolladas previamente por Jesús.



Profiling de código (si aplica)

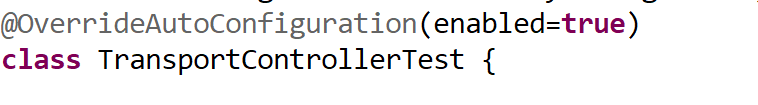
No aplica.

Refactorizaciones (si aplica)

No aplica.

Principales problemas encontrados

Nos hemos encontrado varios problemas a lo largo de la asignatura con el proyecto, pero siendo la gran mayoría provocados por el principal problema que consiste en lo siguiente. Decidimos no utilizar el proyecto base Pet Clinic proporcionado por Spring y en la asignatura, debido a que no nos parecieron congruentes varias cosas a nivel de lógica de la aplicación. Por ejemplo, un Owner podía tocar datos sensibles de otros Owner entre otras cosas. Entonces creamos un proyecto de Spring Boot con Thymeleaf para las vistas en lugar de JSP desde cero, utilizando una base de datos (MySQL) externa no como se daba en la asignatura en un primer momento con una base de datos incrustada(H2). Este cambio desde el principio acarreó otros problemas como menciono al principio, por ejemplo, a la hora de realizar los test unitarios de controladores. Al crear el contexto para que se pudiera realizar ese test no servía con las anotaciones dadas en la asignatura porque estaba enfocado a usar la base de datos H2, entonces después de investigar ambos al añadir esta anotación se pudo pasar a realizar estos tests.



Tuvimos más problemas por tener el proyecto desde cero, pero fueron de menos importancia y se solventaron casi al instante, he destacado este porque nos paró varios días al principio de la asignatura.

# Otros comentarios

La autenticación implementada es diferente a la seguida en la asignatura (proyecto base de Pet Clinic), ya que, nosotros tenemos una configuración basada en una clase MyUserDetailsService que esta implementa una interfaz de Java llamada UserDetailsService. Mediante una consulta por username se permite el inicio de sesión a los usuarios. En cambio en la asignatura se seguía una autenticacion jdbc, insertando esta consulta por username directamente en el archivo de configuración SecutiryConfig.java