Informe individual de actividades del proyecto

# Datos generales

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

Nombre de usuario en GitHub: jesusrico12

Participación en el proyecto

He implementado las HUs-1 a la 5 y la HU-14 (todas las historias pertenecientes la entidad Transporte). Esto incluye todos los archivos pertenecientes a esta clase (servicios, modelos, repositorios, controladores, vistas, configuración…). El esquema de la base de datos junto con su “populate” se ha realizado de manera conjunta con mi compañero. En el repositorio de Github se encuentra un enlace al diagrama UML del proyecto. En cuanto a las pruebas se han hecho un testeo “cruzado”, por lo tanto he realizado las pruebas oportunas sobre la implementación llevada a cabo por mi compañero. Es decir, todos los test unitarios, IU, integración, End2End… de Curso, Seguro y Certificado. Cabe destacar que se han realizado pruebas que no aplican al nivel 6 como son las End2End o pruebas unitarias parametrizadas para cerciorarnos del correcto funcionamiento de nuestro proyecto. Por último, cabe mencionar que todos los test salvo los de rendimiento se encuentra en el directorio src/test/java del proyecto.

Historias de usuario en las que he participado

He implementado las HUs-1 a la 5 y la HU-14 (todas las historias pertenecientes la entidad

Transporte).

Funcionalidad implementada

He implementado los controladores TransportController, CustomErrorController y HomeController. En cuanto al modelo se ha realizado de manera conjunta, así como su esquema y “populate”. He implementado el servicio, repositorio y vistas de Transporte, así como archivos de configuración del proyecto.

## Pruebas implementadas

### Pruebas unitarias

He implementado test unitarios de las clases CertificateServiceTest y CourseServiceTest donde cubre los servicios y repositorios de Curso, Seguro y Certificado. También hemos realizado pruebas parametrizadas sobre la clase TransporteServiceTest para aprender a realizar comprobaciones de la misma unidad de código en un mismo método con distintos escenarios. Para realizar estas comprobaciones se ha implementado la clase ValidatorTest. He implementado un total de 17 pruebas unitarias contemplando distintos escenarios para cada método no trivial.

Pruebas de interfaz de usuario

He creado un caso positivo para cada historia de usuario y uno negativo para la historia de HU-6 “publicar curso”. Debido a la naturaleza de las historias de usuario en muchos casos no se puede realizar un caso negativo. Estos test se encuentran en el directorio src\test\java\com\DP2Spring\test\ui\course. En total hay 7 tests positivos englobando todas las HUs de cursos, certificado y seguro y uno negativo en relación la publicación de curso.

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

TODO

Profiling de código (si aplica)

No aplica

Refactorizaciones (si aplica)

No aplica

Ejemplos de pruebas implementadas

## Pruebas unitarias (máximo de dos ejemplos)

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

@Test

@Transactional

@WithMockUser("clerk1")

**public** **void** shouldNotPersistCourse() {

Course course = **new** Course();

Certificate certificate = **new** Certificate();

certificate.setDescription("Test cert");

certificate.setEntity("Testt");

course.setDescription("Test negativo");

course.setPrice(12.0);

course.setStartDate(**new** Date(System.*currentTimeMillis*()-1));

course.setEndDate(**new** Date(System.*currentTimeMillis*()-1));

*assertThat*(course.getOwnersRegistered()).isNullOrEmpty();

**try** {

//Check first certificate save

**this**.certificateService.save(certificate);

course.setCertificate(certificate);

**this**.courseService.save(course);

}**catch**(Throwable oops) {

System.***out***.println("=================================================");

System.***out***.println(oops.getMessage());

System.***out***.println("=================================================");

//Checking exception is what expected

*assertThat*(oops.getMessage().equals("La fecha de inicio debe ser anterior a la fecha de fin."));

}

entityManager.flush();

//Id == 0 means that doesn't get persisted

*assertThat*(course.getId() == 0);

}

En primer lugar, este test comprueba que un certificado no debe persistir con fecha fin igual o anterior a la fecha inicio.

* La sección *Arrange* de esta prueba consiste en inicializar un certificado y un curso seteando sus atributos con valores correctos salvo la fecha de fin. Antes de someter nuestra lógica se realiza una pequeña comprobación de que no hay ningún propietario inscrito ya que se acaba de crear.
* En la etapa *Act* se persiste el certificado, se setea al curso y se somete a examen el método de publicar curso “save”.
* Por último, en *Assert* podemos verificar que dicho método no llegar a persistir la entidad Curso debido a que salta la assertions de la fecha. Luego se compruebe que dicho assert del método es contemplado antes de que se persista en la base de datos.

//Clerk nulo

@WithMockUser(username = "clerk1", authorities = {"CLERK"})

@Test

**void** testCreateCourseNoSuccess3() **throws** Exception{

mockMvc.perform(*post*("/course/createCourse")

.with(*csrf*())

.param("price", "10.99")

.param("description", "Description")

.param("startDate", "22/09/2018")

.param("endDate", "22/09/2016")

.param("certificate", "50"))

.andExpect(*model*().attributeHasErrors("course"))

.andExpect(*model*().attributeHasFieldErrors("course", "clerk"))

.andExpect(*status*().isOk())

.andExpect(*view*().name("/course/create"));

## }

En primer lugar, este test comprueba que el controlador de crear curso recibe de forma anómala y maliciosa mediante la vista un curso sin secretario.

* La sección *Arrange* de esta prueba consiste en simular un contexto de autenticación del usuario “Clerk1” bajo el rol “CLERK”.
* En la etapa *Act* simula la acción de envío de formulario con sus correspondientes valores de los atributos sin el creador de este.
* Por último, en *Assert* podemos verificar que la propiedad “clerk” tiene un error de modelo en la entidad curso. También se comprueba que el controlador responde correctamente mediante una respuesta con código 200 y devolviendo a la vista de creación de curso.

## Pruebas unitarias parametrizadas (si aplica)

Aunque al nivel al que aspiramos no aplica se han implementado varias pruebas parametrizadas en pareja. Estas se localizan en la clase src\test\java\com\DP2Spring\test\services\TransportServiceTest.java .

//VALIDACION (AVANZADO)

@ParameterizedTest

@CsvSource({

"Utrera, Lebrija,PENDING,'80,81'",

"Plaza duque, Utrera,ESTADONUEVO,'80,81'",

"Segovia, Utrera,PENDING,'81'",

"Sevilla, ,PENDING,'80,81'",

", Utrera,PENDING,'80,81'",

", ,PENDING,'80,81'",

})

@Transactional

@WithMockUser("owner1")

**void** solicitarTransporteValidacion(String origin,String destination,String status, String pets) {

Transport t = **new** Transport();

t.setOrigin(origin);

t.setDestination(destination);

t.setStatus(status);

Validator validator = createValidator() ;

Set<ConstraintViolation<Transport>> constraintViolations = validator.validate(t);

**if**(constraintViolations.size()== 0) {

**this**.transportService.solicitarTransporte(t,pets);

entityManager.flush();

*assertTrue*(t.getId() > 0);

}**else** {

**for**(ConstraintViolation<Transport> c: constraintViolations) {

**if**(c.getPropertyPath().toString().contentEquals(("origin"))) {

*assertThat*(c.getMessage()).isEqualTo("no puede estar vacío");

}**else** **if**(c.getPropertyPath().toString().contentEquals(("destination"))) {

*assertThat*(c.getMessage()).isEqualTo("no puede estar vacío");

}**else** **if**(c.getPropertyPath().toString().contentEquals(("status"))) {

*assertTrue*(c.getMessage().contentEquals("no puede estar vacío")|| c.getMessage().contentEquals("tiene que corresponder a la expresión regular \"^PENDING|TRANSPORTED$\""));

}

}

}

}

En primer lugar, este test comprueba que el controlador de crear curso recibe de forma anómala y maliciosa mediante la vista un curso sin secretario.

* La sección *Arrange* de esta prueba consiste
* En la etapa *Act* simula
* Por último, en *Assert* podemos verificar

## 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 las siguientes clases: CertificateControllerIntegrationTest y CourseControllerIntegrationTest. Estas se encuentran el directorio src\test\java\com\DP2Spring\test\web\integration.

En CourseControllerIntegrationTest encontramos el siguiente test:

@WithMockUser(username = "clerk1", authorities = {"CLERK"})

@Test

**void** saveCoursePos() **throws** Exception {

Course c= **this**.courseService.create();

Certificate cer = **this**.certificateService.create();

cer.setDescription("sample");

cer.setEntity("sample");

**this**.certificateService.save(cer);

c.setStartDate(**new** Date(System.*currentTimeMillis*()+200000));

c.setEndDate(**new** Date(System.*currentTimeMillis*()+2000000000));

c.setCertificate(cer);

c.setDescription("sample");

BindingResult bindingResult=**new** MapBindingResult(Collections.*emptyMap*(),"");

ModelAndView result=courseController.save(c, bindingResult);

String view = result.getViewName();

*assertEquals*(view,"redirect:/clerk/listCourses");

}

En primer lugar, este test comprueba el correcto funcionamiento de la publicación de un curso. Para ello, es necesario la asignación de un certificado.

* La sección *Arrange* de esta prueba consiste en persistir un certificado añadiéndole los valores necesarios para realizar el test como son entidad y descripción. También se inicializa un curso seteando valores correctos para sus atributos y añadiendo dicho certificado a este. Inicializamos nuestro BindingResult sin errores ya que los datos introducidos son correctos.
* En la etapa *Act* llamamos a nuestro método de controlador de curso. Este permite publicar un curso si no hay errores o llevarte a la misma vista si existen errores.
* Por último, en *Assert* podemos verificar mediante assertEquals que se ha realizado correctamente la publicación del curso ya que redirige a la vista adecuada.

## Pruebas de interfaz de usuario

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

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.

Principales problemas encontrados

Al principio de la asignatura nos dimos cuenta que el esquema del Pet-Clinic original tenía cierta lógica que era incongruente con las implementaciones futuras queíbamos a realizar. Por tanto, tomamos la decisión de hacer un modelo entero nuevo, así como una distinta autenticación a la ya implementada <https://www.baeldung.com/get-user-in-spring-security> para tener un correcto control de authenticacion de usuarios con distintos roles comoes en nuetro caso “Owner” y “Clerk”. También dedicimos aprender una nueva tecnología “Thymeleaf” que permite conectar nuestros controladores con nuestras vistas Html. Pocas semanas después obtuvimos la suficiente soltura para implementar nuestras historias de usuario. Estamos contento con la decisión pues se han realizado las diferentes HUs de manera satisfactoria y en un corto periodo de tiempo.

# Otros comentarios

Authenticacion nueva, nueva tecnología thymeleaf