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, rendimiento… 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 OwnerController. 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 y Owner así como archivos de configuración del proyecto.

Pruebas implementadas

Pruebas unitarias

He implementado test unitarios de las clases Certificado, seguro y curso cubriendo sus respectivos controladores, servicios y repositorios. 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 comprobar 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

He desarrollado las pruebas de rendimiento que engloba las HU-5 a la 13, es decir el desarrollo que ha hecho mi compañero. *En el documento general de rendimiento se recoge con más detalles dichas pruebas.*

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, un curso seteando sus atributos con valores correctos salvo la fecha de fin y un contexto de autenticación como CLERK. 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 las validaciones del modelo de la entidad Transporte a través del método solicitarTransporte.

- La sección *Arrange* de esta prueba consiste en probar todos los parámetros de entrada con todos los valores posibles para tener un amplio espectro de testeo siempre utilizando un contexto de autenticación como OWNER.
- En la etapa *Act* llamamos al método solicitarTransporte donde se guarda la entidad en la base datos si no tiene errores de persistencia y lógica.
- Por último, en *Assert* podemos verificar y corroborar que cuando se meten todos los valores de forma correcta y coherente se guarda la entidad y luego se comprueba que existe en la DB mediante assert. También verificamos que cuando no ocurre esto el sistema se comporta como debería saltando dichas excepciones.

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 estaríamos 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. Todo esto siempre bajo un contexto de autenticación como OWNER.
- 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

Los test de interfaz de usuarios se sitúan en el directorio src\test\java\com\DP2Spring\test\ui\course .

```
@Test
 public void testHU6UIPositivo() throws Exception {
          driver.get("http://localhost:"+port+"/login");
  driver.findElement(By.linkText("Iniciar sesion")).click();
  driver.findElement(By.id("phone-form1-z")).click();
  driver.findElement(By.id("phone-form1-z")).clear();
  driver.findElement(By.id("phone-form1-z")).sendKeys("clerk1");
  driver.findElement(By.id("message-form1-z")).clear();
  driver.findElement(By.id("message-form1-z")).sendKeys("clerk1");
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  driver.findElement(By.linkText("Publicar curso")).click();
  driver.findElement(By.id("entity")).click();
  driver.findElement(By.id("entity")).clear();
  driver.findElement(By.id("entity")).sendKeys("test 2");
  driver.findElement(By.id("description")).clear();
  driver.findElement(By.id("description")).sendKeys("test 2");
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  driver.findElement(By.id("description")).click();
  driver.findElement(By.id("description")).clear();
  driver.findElement(By.id("description")).sendKeys("test 2");
  driver.findElement(By.xpath("//form[@id='myForm']/div[2]")).click();
  driver.findElement(By.id("price")).clear();
  driver.findElement(By.id("price")).sendKeys("13");
  driver.findElement(By.id("startDate")).click();
  driver.findElement(By.xpath("//div[@id='ui-datepicker-div']/div/a[2]/span")).click();
  driver.findElement(By.linkText("6")).click();
  driver.findElement(By.id("endDate")).click();
  driver.findElement(By.xpath("//div[@id='ui-datepicker-div']/div/a[2]/span")).click();
  driver.findElement(By.linkText("20")).click();
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  assertEquals("test 2", driver.findElement(By.xpath("//td")).getText());
 }


 //Caso negativo con fecha inicio pasada

 @Test
 public void testHU6UINegativo() throws Exception {
          driver.get("http://localhost:"+port+"/login");
  driver.findElement(By.linkText("Iniciar sesion")).click();
  driver.findElement(By.xpath("//section[@id='form1-z']/div[2]/div")).click();
  driver.findElement(By.id("phone-form1-z")).clear();
  driver.findElement(By.id("phone-form1-z")).sendKeys("clerk1");
  driver.findElement(By.id("message-form1-z")).clear();
  driver.findElement(By.id("message-form1-z")).sendKeys("clerk1");
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  driver.findElement(By.linkText("Publicar curso")).click();
  driver.findElement(By.id("entity")).click();
  driver.findElement(By.id("entity")).clear();
  driver.findElement(By.id("entity")).sendKeys("test negativo");
  driver.findElement(By.id("description")).clear();
  driver.findElement(By.id("description")).sendKeys("test negativo");
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  driver.findElement(By.id("description")).click();
  driver.findElement(By.id("description")).clear();
  driver.findElement(By.id("description")).sendKeys("test negativo con fecha inicio pasada");
  driver.findElement(By.xpath("//form[@id='myForm']/div[2]")).click();
  driver.findElement(By.id("price")).clear();
  driver.findElement(By.id("price")).sendKeys("12");
  driver.findElement(By.id("startDate")).click();
  driver.findElement(By.xpath("//div[@id='ui-datepicker-div']/div/div")).click();
  driver.findElement(By.linkText("1")).click();
  driver.findElement(By.id("endDate")).click();
  driver.findElement(By.xpath("//div[@id='ui-datepicker-div']/div/a[2]/span")).click();
  driver.findElement(By.linkText("1")).click();
  driver.findElement(By.xpath("//button[@type='submit']")).click();
  assertEquals("La fecha de inicio debe ser futura.",
driver.findElement(By.xpath("//form[@id='myForm']/div[5]/p")).getText());
 }
```

En primer lugar, este test comprueba la HU correspondiente a publicar curso. Se verifica un escenario positivo con los datos introducidos correctamente y uno negativo añadiendo una fecha de inicio pasada. En cada test se comprueba con asserts que el sistema se comporta como debería.

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

```
@WithMockUser(username = "clerk1", authorities = {"CLERK"})
@Test
void testCreateCourseNoSuccess() throws Exception{

		mockMvc.perform(post("/course/createCourse")
					.with(csrf())
					.param("price", "10.99")
					.param("description", "Description")
					.param("startDate", "22/09/2015")
					.param("endDate", "22/09/2016")
					.param("clerk", "200"))
		.andExpect(model().attributeHasErrors("course"))
		.andExpect(model().attributeHasFieldErrors("course", "certificate"))
		.andExpect(status().isOk())
		.andExpect(view().name("/course/create"));



}
```

En primer lugar, este test comprueba que el controlador de crear curso recibe todos los parámetros correctamente salvo que no se ha asociado ningún certificado al curso.

- La sección *Arrange* de esta prueba consiste en simular un contexto de autenticación del usuario “Clerk1” bajo el rol “CLERK” sin mockear los servicios oportunos.
- En la etapa *Act* simula la acción de envío de formulario con sus correspondientes valores de los atributos sin asignarle un certificado de la DB.
- Por último, en *Assert* podemos verificar que la propiedad “certificate” 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 de integración con un servicio externo (si aplica)

No aplica.

Pruebas de rendimiento

Esto es un extracto del archivo DosEscenarios2.scala. Este test se basa en comparar el rendimiento de la HU6 (Publicar un curso) frente a la HU8 (Inscribirse en un curso). Mencionar que la HU6 la realiza el rol CLERK y la HU8 la realiza el rol OWNER. En el documento de diagnosis se puede ver con más detalles la particularidad del test.

```
val publicarCurso = scenario("Publicar").exec(Home.home,
                                IniciarSesion.iniciarSesion,
                                IniciadoClerk.iniciadoClerk,
                                CrearCertificado.crearCertificado,
                                CertificadoCreado.certificadoCreado,
                                CursoCreado.cursoCreado,
                                CerrarSesion.cerrarSesion,
                                CerradoSesion.cerradoSesion)

val verCursos = scenario("ListadoCursos").exec(Home.home,
                                    IniciarSesion.iniciarSesion,
                                    IniciadoOwner.iniciadoOwner,
                                    CursosDisponibles.cursosDisponibles,
                                    CerrarSesion.cerrarSesion,
                                    CerradoSesion.cerradoSesion)



setUp(
   publicarCurso.inject(rampUsers(550) during (100 seconds)),
   verCursos.inject(rampUsers(550) during (100 seconds))
).protocols(httpProtocol)
 .assertions(
   global.responseTime.max.lt(5000),
   global.responseTime.mean.lt(1000),
   global.successfulRequests.percent.gt(95)
 )
```

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 que íbamos a realizar. Por tanto, tomamos la decisión de hacer un modelo entero nuevo, así como una distinta autenticación a la ya implementada para tener un correcto control de autenticación de usuarios con distintos roles como son “OWNER” y “CLERK” y seguridad. 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