



Report Item 5

Grupo: 20

Miembro:

José Ángel Domínguez Espinaco

Daniel Lozano Portillo

José Joaquín Rodríguez Pérez

María Ruíz Gutiérrez

Miguel Ternero Algarín

Laura Vera Recacha

Índice

1. Introducción.....	3
2. Pasos que seguir para la implementación.....	3
2.1. Dependencias.....	3
2.2. Configuración del fichero de prueba	3
2.3. Casos de prueba.....	6
2.4. Explicación de un caso de prueba concreto.....	7
3. Ejecución de las pruebas	8
4. Casos de prueba.....	8
4.1. Casos de prueba de las listas.....	8
4.2. Casos de uso de acciones de asistir y dejar de asistir.	9
4.3. Casos de prueba para el display.....	10
4.4. Casos de prueba para editar	10
5. Conclusión	12
6. Referencias.....	12

1. Introducción.

Para la finalización completa del proyecto "Acme-rendezvous 2.0" se pide como requisito un A+ que consiste en la implementación de test para buscar fallos en los controladores, como hemos visto hasta ahora a lo largo del curso podemos hacer pruebas con relativa facilidad, pero estas solo se pueden hacer sobre los repositorios y servicios. Se pide por tanto que se elija el marco de prueba de servlets que se prefiera para testear un controlador.

El marco de prueba de servlets elegido para esta ocasión ha sido MockMvc. Es una herramienta de la cual nos provee Spring. Ha sido elegida debido a la facilidad de realizar test y a su entendimiento.

2. Pasos que seguir para la implementación.

2.1. Dependencias.

Para obtener las dependencias para realizar las pruebas necesitaremos que se encuentre en el pom.xml lo siguiente:

- Spring Framework 4.0.0
- Spring-test-mvc 3.2.4
- JUnit 4.11
- Maven 4.0.0

Actualmente de entrada no necesitaremos modificar el pom.xml puesto que estas dependencias ya están incluidas en el proyecto.

2.2. Configuración del fichero de prueba

Lo siguiente es la implementación de todo lo necesario en el fichero de prueba para poder escribir nuestras pruebas y ejecutarlas.

Comenzaremos con la creación de una clase llamada `RendezvousControllerTest` que es la que se va a encargar de realizar las pruebas al controlador de `RendezvousUserController`. Primero debemos de importar las librerías correspondientes para el funcionamiento correcto, estas son las que se pueden apreciar en la imagen siguiente:

```

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.context.WebApplicationContext;

import utilities.AbstractTest;
import controllers.user.RendezvousUserController;

```

Como se ha dicho anteriormente el controlador escogido para este caso es `RendezvousUserController`. Además de haber añadido las librerías anteriores a `RendezvousControllerTest` tenemos que añadir la configuración necesaria a la clase, esta configuración es muy parecida a la que añadíamos en los anteriores test informales. La configuración que tenemos que añadir es la siguiente:

```

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {
    "classpath:spring/junit.xml"
})
@WebAppConfiguration
@Transactional

```

Esto hay que añadirlo como siempre justo encima de la declaración de la clase. Con “`@RunWith`” estamos diciendo que se va a ejecutar con JUnit y la siguiente anotación indica donde está la configuración de JUnit. La presencia de `@WebAppConfiguration` en una clase de prueba indica que se debe cargar un `WebApplicationContext` para la prueba utilizando un valor predeterminado para la ruta a la raíz de la aplicación web. `@Transactional` indica que nos encontramos dentro de una transacción.

Tenemos en cuenta que estamos configurando el contexto de Spring directamente en el test `RendezvousControllerTest` declarando una clase estática mediante la anotación `@Configuration`. La anotación `@Configuration` es la encargada de definir que la clase es una clase de configuración para el propio framework, es decir Spring lo recogerá automáticamente.

Declararemos dentro de esta clase un vean, para ello basta con escribir la anotación `@Bean`. Cuando Java se encuentra con dicha anotación,

se ejecutará el método y registrará el valor de retorno como un bean dentro de la clase BeanFactory, y así el método estará disponible para spring. Esto lo declararemos al final del test siempre para tener el archivo bien estructurado.

En la siguiente imagen se puede ver cómo queda la configuración del contexto.

```
@Configuration
public static class TestConfiguration {

    @Bean
    public RendezvousUserController rendezvousUserController() {
        return new RendezvousUserController();
    }

}
```

Después de incorporar lo anterior al fichero de prueba nos situamos justo después de la declaración de la clase `RendezvousController` test e incorporamos los diferentes objetos que usaremos. Primero declaramos un objeto `MockMvc` que nos ayudara a llamar al controlador `RendezvousUserController`. La anotación `@Before` junto con el método `setUp()` asegura que el método se invoca antes de cada prueba. En la imagen se puede apreciar todo lo incorporado después de la cabecera, cabe destacar que la clase como siempre hacemos en los test extenderá de `AbstractTest`.

```
public class RendezvousControllerTest extends AbstractTest {

    private MockMvc mockMvc;

    @Autowired
    private WebApplicationContext ctx;

    @Override
    @Before
    public void setUp() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.ctx).build();
    }

}
```

2.3. Casos de prueba.

Con todo lo anterior ya podemos disponernos a escribir o desarrollar nuestros casos de prueba dentro del fichero `RendezvousControllerTest`. Estos casos de prueba lo recomendable es que vayan estructurados de la mejor manera posible, es decir separando listados de ediciones y demás. En la siguiente imagen se pueden apreciar varias pruebas para los listados.

```
@Test
public void getListDeletedPositive() throws Exception {
    //Usuario va a listar las citas que ha eliminado
    this.authenticate("user1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list-deleted"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

@Test
public void getListRendezvousPositive1() throws Exception {
    this.authenticate("user3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/list"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void getListRendezvousNegative1() throws Exception {
    //Se va a intentar acceder al listado de citas creadas por un usuario sin estar logeado
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/list"));
}
```

Como se ha podido observar en los casos de uso creamos casos de prueba positivos y negativos.

Pasamos a explicar cómo crear un caso de prueba positivo. Primero debemos poner que nos encontramos ante un test con la anotación `@Test`, luego declaramos el método devolviendo este siempre un void y al final del nombre de la declaración del método tenemos que poner `throws Exception`, posteriormente dentro del método creamos el caso de prueba, pero lo esencial del contenido de los métodos es donde hacemos la petición al controlador, esta se realiza con lo siguiente:

```
this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/assist" + "?rendezvousId=" + id))
    .andExpect(MockMvcResultMatchers.status().isOk())
    .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
```

Especificamos que la petición es ".get" seguido de la url a la que vamos a realizar la petición más los parámetros que lleva esta petición y luego viene lo que se espera que devuelva la petición. En este caso se espera que la petición devuelva un código de estado de 200 Ok y que la vista devuelta sea el listasis de rendezvous. A parte de realizar peticiones GET se pueden hacer peticiones POST también y varias mas de las que nos provee "MockMvcRequestBuilders".

Para la realización de test negativos el funcionamiento es el mismo con la diferencia de que se va a violar una restricción a la hora de hacer la petición o va a saltar algún error de algún tipo, pues este error debe ir dentro de la anotación `@Test` quedando por ejemplo así:

```
@Test(expected = AssertionError.class)
```

Con este error esperado estamos diciendo que la petición al controlador debe de violar un assert.

2.4. Explicación de un caso de prueba concreto.

Por si no queda del todo claro se va a exponer un caso de prueba positivo y otro negativo de intentar acceder a editar una cita. Veamos el positivo:

```
//Solo puede editarla si no esta en modo final
@Test
public void editRendezvousByUserPositive() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvouse4");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/edit" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}
```

Según los requisitos proporcionados por nuestro cliente un usuario solo puede editar las citas que él ha creado además de que esta no debe estar en modo final.

Para realizarlo introducimos la anotación @Test sin esperar ningún error, posteriormente la cabecera del método acompañado de throws Exception, luego introducimos la casuística de la prueba, es decir nos logeamos como "user1" buscamos la "rendezvouse4" y hacemos una petición GET a la url rendezvous/user/edit?rendezvousId=id esperando como respuesta el código de estado 200 ok y la vista edit de rendezvous. Por último, saldremos de la sesión.

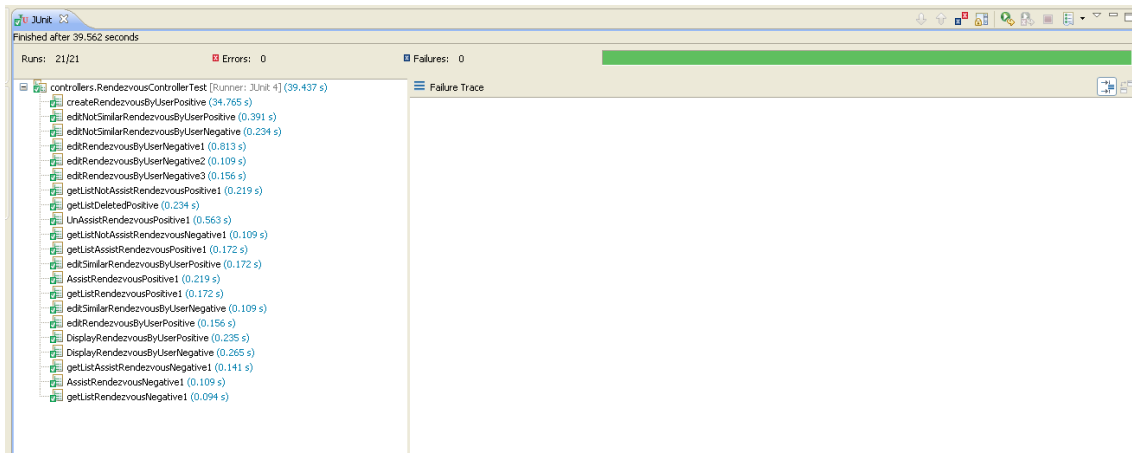
Veamos el caso de prueba para comprobar lo mismo pero esta vez será negativo el resultado.

```
@Test(expected = AssertionError.class)
public void editRendezvousByUserNegative1() throws Exception {
    //Un usuario va a intentar editar una cita que esta en modo final.
    this.authenticate("user1");
    int id = this.getEntityId("rendezvouse1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/edit" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}
```

Para este caso hemos establecido que la respuesta obtenida es la violación de una restricción, para realizarlo nos logeamos como "user1" e intentamos editar la cita1 que se encuentra en modo final, como se encuentra en modo final no nos debería dejar editarla por eso salta el assert.

3. Ejecución de las pruebas

Para comprobar todos los casos de prueba debemos dirigirnos al botón de play situado arriba de la vista de eclipse abrir el desplegable y hacer clic a run as y luego a JUnit test. Esto hará que se ejecuten los casos de prueba declarados en la clase. Si todo va correcto se deberá apreciar que la barra resultante de la ejecución es entera verde. La siguiente imagen demuestra cómo se debería ver la correcta ejecución de las pruebas además de que estas den el resultado esperado.



4. Casos de prueba.

4.1. Casos de prueba de las listas.

```
@Test
public void getListDeletedPositive() throws Exception {
    //Usuario va a listar las citas que ha eliminado
    this.authenticate("user1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list-deleted"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void getListDeletedNegative() throws Exception {
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list-deleted"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
}
```

El caso de prueba "getListDeletePositive()" tiene como objetivo comprobar que podemos acceder al listado de citas borradas del usuario que esta logeado y nos devuelve la vista listasis que es la que usamos como referencia para representar las listas de citas. Para el caso negativo sobre este caso de prueba intentamos acceder al listado sin estar logeado lo que debería de devolver un error al violar un assert.

```
@Test
public void getListRendezvousPositive1() throws Exception {
    this.authenticate("user3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/list"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void getListRendezvousNegative1() throws Exception {
    //Se va a intentar acceder al listado de citas creadas por un usuario sin estar logeado
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/list"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/list"));
}
```


En este caso se intenta acceder al listado de las citas que ha creado un usuario que se corresponde con la petición list y esta nos debería de devolver la vista list. Para el caso negativo la respuesta debe ser un error del tipo AssertionError pues se está intentando acceder a ese listado sin estar logeado.

```
@Test
public void getListNotAssistRendezvousPositivel() throws Exception {
    //Se va a acceder al listado de citas que aun no ha confirmado asistencia el usuario
    this.authenticate("user3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/listnotasis"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void getListNotAssistRendezvousNegativel() throws Exception {
    //Se va a acceder al listado de las citas que un usuario aun no va a asistir sin estar logeado
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/listnotasis"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
}
```

En este caso se intenta acceder al listado de las citas a las que no se asiste aun un usuario. En el primer caso no debe devolver ningún error pues estamos logeados como user3 y en el segundo caso al no estar logeado no podremos acceder a la lista y se devolverá la violación de un assert.

```
@Test
public void getListAssistRendezvousPositivel() throws Exception {
    //Se va a acceder al listado de citas que el usuario ha confirmado su asistencia
    this.authenticate("user2");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/listasis"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void getListAssistRendezvousNegativel() throws Exception {
    //El administrador va a intentar acceder al listado de las citas que va a asistir cuando el admin no puede asistir a ninguna
    this.authenticate("admin");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/listasis"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

~
```

Para estos dos casos de prueba tanto el positivo como el negativo son iguales que lo descrito en la anterior imagen solo que aquí se intenta acceder al listado de las citas a las que un usuario es asistente.

4.2. Casos de uso de acciones de asistir y dejar de asistir.

```
@Test
public void AssistRendezvousPositivel() throws Exception {
    //El usuario va a confirmar asistencia una cita
    this.authenticate("user1");
    int id = this.getEntityId("rendezvou3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/assist" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void AssistRendezvousNegativel() throws Exception {
    //El admin va a intentar asistir a una rendezvous
    this.authenticate("admin");
    int id = this.getEntityId("rendezvou3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/assist" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}
```

En estos casos se intentar ser asistente de la cita 3, en el caso positivo se intenta como user3 y la cita esta disponible para asistir por tanto no debe de violar ninguna restricción y en el segundo caso se intenta acceder desde admin y esta acción no está disponible para un admin lo cual debe de devolver un error de violación de assert.

```

@Test
public void UnAssistRendezvousPositive1() throws Exception {
    //El usuario va a cancelar su asistencia a la una cita.
    this.authenticate("user3");
    int id = this.getEntityId("rendezvous3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/notassist" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void UnAssistRendezvousNegative1() throws Exception {
    this.authenticate("admin");
    int id = this.getEntityId("rendezvous3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/notassist" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/listasis"));
}

```

Para estos casos lo que se pretende es dejar de asistir a una cita, el primer caso no debe devolver error pues el user3 es asistente de la cita 3, para el caso negativo un administrador no puede ejecutar esa acción y debe saltar un error de assert.

4.3. Casos de prueba para el display

```

// Display -----
//Solo pueden verlas usuarios que tengan mas de 18 años de edad.

@Test
public void DisplayRendezvousByUserPositive() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvous3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/display" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/display"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void DisplayRendezvousByUserNegative() throws Exception {
    //Un usuario menor de edad va a intentar mostrar una cita para adultos.
    this.authenticate("user5");
    int id = this.getEntityId("rendezvous4");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/display" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/display"));
    this.unauthenticate();
}

```

En estos casos de prueba se prueba que se pueda hacer correctamente la acción de ver una cita, el primer caso no debe devolver ningún error puesto que hacemos el display desde el user3 a una cita que existe y se puede ver y el segundo caso debe violar una restricción assert puesto que el user5 es menor de edad y la cita4 es para adultos.

4.4. Casos de prueba para editar

```

// Edition -----
@Test
public void createRendezvousByUserPositive() throws Exception {
    this.authenticate("user1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/create"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void createRendezvousByAdmin() throws Exception {
    this.authenticate("admin");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/create"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}

```

En estos casos se intenta crear una cita, en el caso positivo como user1 y en el segundo caso como admin y este actor no puede crear citas.

```
//Solo puede editarla si no esta en modo final
@Test
public void editRendezvousByUserPositive() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvous4");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/edit" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void editRendezvousByUserNegative1() throws Exception {
    //Un usuario va a intentar editar una cita que esta en modo final.
    this.authenticate("user1");
    int id = this.getEntityId("rendezvous1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/edit" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}
```

En estos casos lo que se intenta es editar una cita existente. El primer caso no debe violar ninguna restricción puesto que el user1 es propietario de la cita 4 y esta no esta en modo final aun ni ha sido organizada ya. En el segundo caso se intenta editar una cita que esta en modo final y esto no debe ser posible.

```
@Test(expected = AssertionError.class)
public void editRendezvousByUserNegative2() throws Exception {
    //Un usuario va a intentar editar una cita que esta borrada.
    this.authenticate("user5");
    int id = this.getEntityId("rendezvous5");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/edit" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void editRendezvousByUserNegative3() throws Exception {
    //Un usuario va a intentar editar una que el no ha creado
    this.authenticate("user1");
    int id = this.getEntityId("rendezvous5");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/edit" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/edit"));
    this.unauthenticate();
}
```

Como el editar de citas tiene mucha casuística se decidió hacer mas test para el edit. Estos casos los dos deben lanzar una excepción, el primero se intenta editar una cita que esta borrada, el segundo se intenta editar una cita que no ha sido creada por el user1.

```
@Test
public void editSimilarRendezvousByUserPositive() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvous1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/editSimilar" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/editNotSimilar"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void editSimilarRendezvousByUserNegative() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvous3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/editSimilar" + "?rendezvousId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/editNotSimilar"));
    this.unauthenticate();
}
```

En estos casos se esta probando el asociar una cita con otra accediendo a la vista de editNotSimilar que saldrán las que no están asociadas aún. El primer caso no se debe lanzar ninguna excepción puesto que la cita1 es creada por el user1, el segundo caso debe lanzar una excepción puesto que el user1 esta intentando hacer esto a una cita que no ha creado el.

```

@Test
public void editNotSimilarRendezvousByUserPositive() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvouse1");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/editNotSimilar" + "?rendezvouseId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/editSimilar"));
    this.unauthenticate();
}

@Test(expected = AssertionError.class)
public void editNotSimilarRendezvousByUserNegative() throws Exception {
    this.authenticate("user1");
    int id = this.getEntityId("rendezvouse3");
    this.mockMvc.perform(MockMvcRequestBuilders.get("/rendezvous/user/editNotSimilar" + "?rendezvouseId=" + id))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.view().name("rendezvous/editSimilar"));
    this.unauthenticate();
}

```

Para estos casos se actúa de la misma forma que la anterior solo que se van a desenlazar citas de la cita elegida.

5. Conclusión

El hecho de realizar pruebas a los controladores nos proporciona una mayor cobertura de la que disponíamos ya, a través de la realización de estos test se pueden encontrar distintos fallos a los encontrados en los test de servicios y repositorios lo cual nos resulta de mucha ayuda.

6. Referencias.

- <https://platzi.com/jee/tutoriales/pruebas-usando-junit-mockito-y-mockmvc/>