Nombre: Marcos Rodríguez Osorio Grupo: G3-08

Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g3-08
Nombre de usuario en GitHub: marrodoso
Rama del usuario en Github: marrodoso, marrodoso1, marrodoso2

Participación en el proyecto

Historias de usuario en las que he participado

He implementado las historias de usuarios;
H2: Asignar árbitro a un Partido
H12: Visualización de los partidos en una competición
H14: Visualización de los entrenadores de un equipo
H15: Visualización árbitro, jugador y equipo de un partido
La H2 la implementé con la gran ayuda de mi equipo, Javi, Ignacio, Juan Luis y sobre todo Gonzalo.

Además, he implementado la Regla de Negocio 2: Imposibilidad de inscripción de un jugador lesionado, con la ayuda de mi compañero Gonzalo.

Funcionalidad implementada

He implementado el controlador ArbitroController, algunos métodos de CompeticionController y EntrenadorController.
He implementado los servicios ArbitroService, CompeticionService y EntrenadorService.
He creado las entidades Arbitro, Competición y Entrenador.
He creado las vistas de la carpeta árbitros.
He creado la vista competicionesList de la carpeta competiciones.
He creado las vistas de la carpeta entrenadores.
He implementado en la clase PartidoValidator el método para la RN2.

Pruebas implementadas

Pruebas unitarias

Con la ayuda de Gonzalo he realizado los test unitarios para Servicio de Arbitro y Entrenador. (ArbitroService, EntrenadorService).
Además, con la misma ayuda de Gonzalo, he realizado el test unitario para el controlador de Entrenador. (EquipoController)
Eso hace un total de 3 clases de test unitarios con un total de 16 métodos anotados con @Test.

Nombre: Marcos Rodríguez Osorio Grupo: G3-08

Pruebas de Controlador

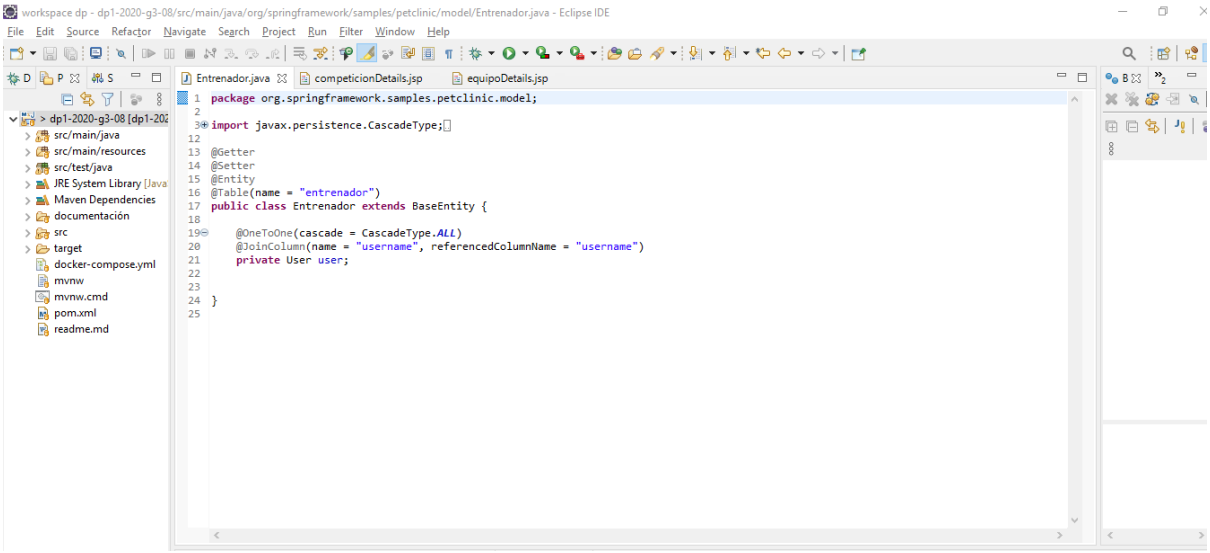
He creado 6 casos de prueba positivo y 2 negativos de controlador para TestEntrenadorController

Ejemplos de funcionalidades implementadas

Entidades

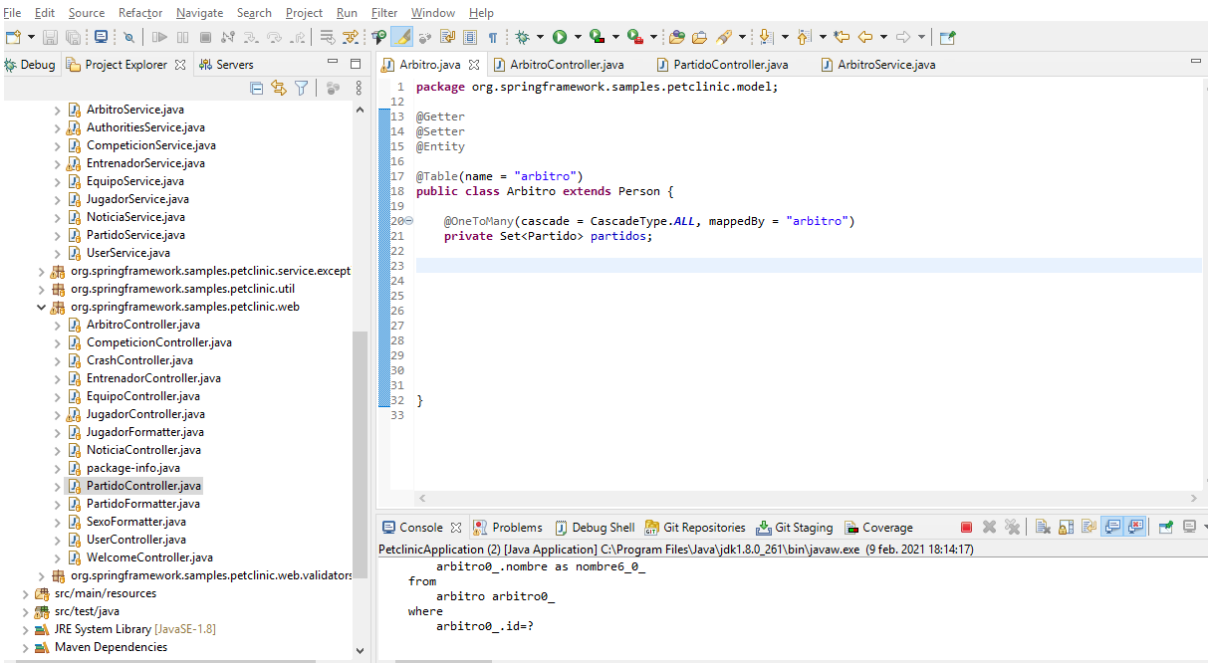
Entidad Entrenador esta en la ruta src/main/java/org.sprintframework.samples.petclinic/model.

Consiste en una entidad que implementa en su código una relación oneToOne con la entidad user de la que hereda todos sus atributos. Cabe destacar que Entrenador está presente en la historia de usuario H14 (Visualización de los entrenadores de un equipo), pero como se puede ver, no se observa la relación oneToOne con Equipo, ya que esa relación la tiene Equipo en su modelo.



Entidad Arbitro esta en la ruta src/main/java/org.sprintframwork.samples.petclinic/model.

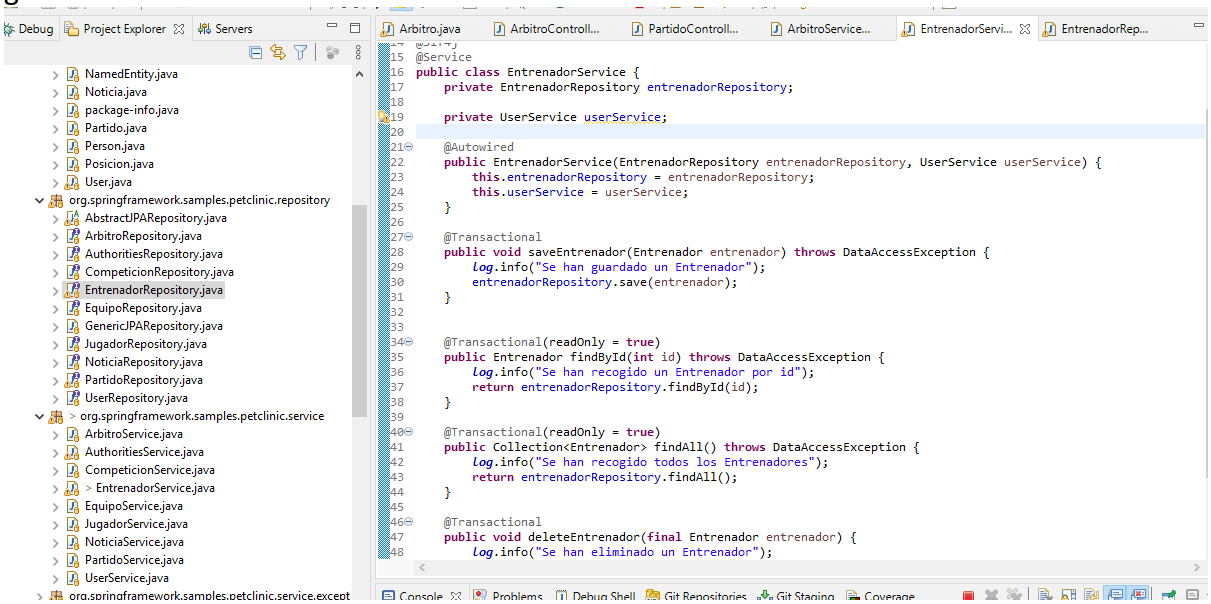
Consiste en una entidad que implementa en su código una relación oneToMany con la entidad Partido, debido a varias historias de usuario (H2 - Asignar árbitro a un partido y H15 - Visualización árbitro, jugador y equipo de un partido). Esta entidad extiende de Person y hereda todos sus atributos.



Servicio

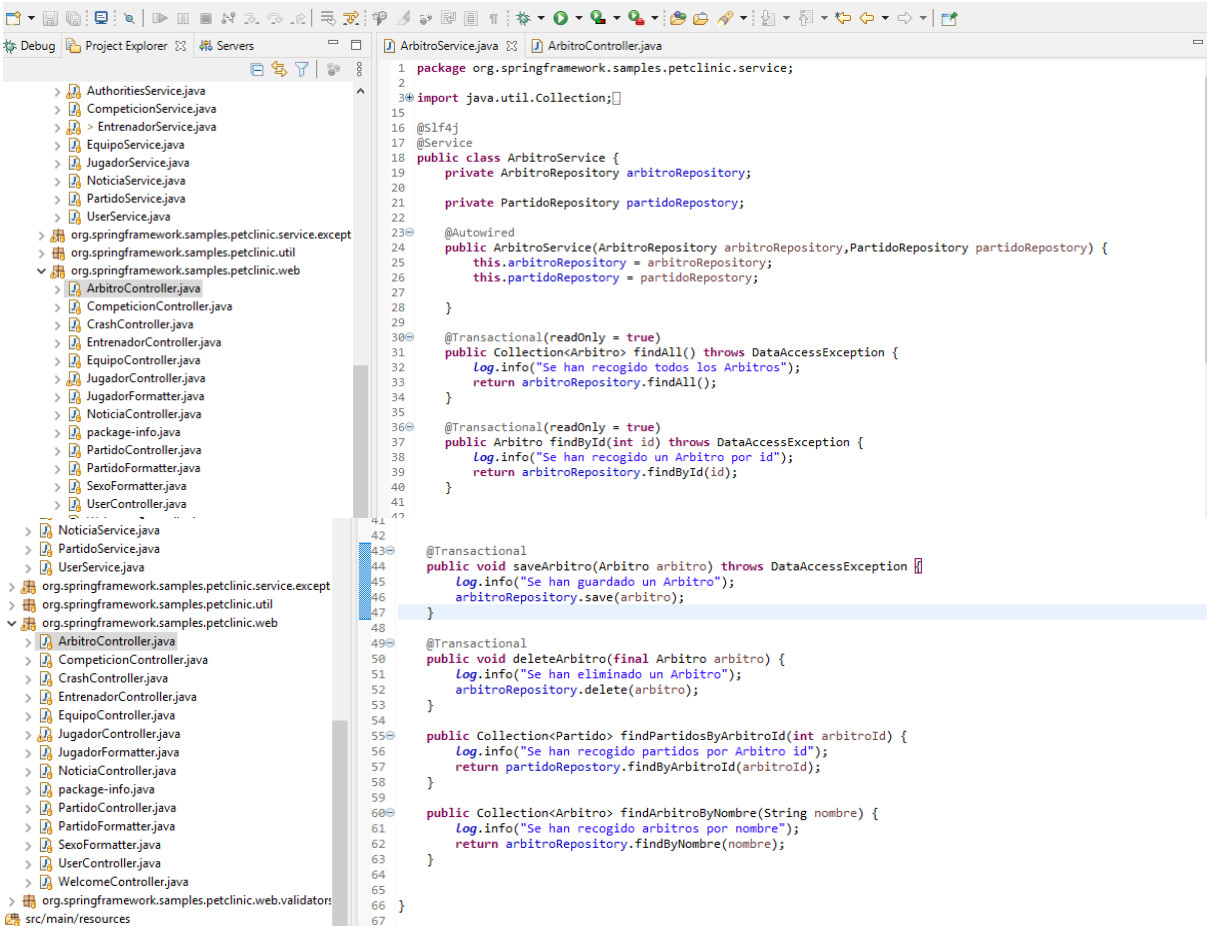
Servicio EntrenadorService, localizado en la ruta src/main/java/org.springframework.samples.petclinic/service

El EntrenadorService se encarga de agrupar todas las funcionalidades que se prestaran como servicio, siendo quien invoca los métodos del Repository.
En este caso, podemos ver que implementa las invocaciones a los métodos del repository necesarios para obtener un entrenador por su ID, una colección de todos los entrenadores y guardar o borrar un entrenador.



Servicio ArbitroService, localizado en la ruta
src/main/java/org.springframework.samples.petclinic/service

El ArbitroService se encarga de agrupar todas las funcionalidades que se prestaran como servicio, siendo quien invoca los métodos del Repository.
En este caso, podemos ver que implementa las invocaciones a los métodos del repository necesarios para obtener un árbitro por su ID, una colección de todos los árbitros, guardar o borrar un árbitro. También tenemos 2 métodos, los cuales, uno devuelve los partidos en lo que está un árbitro determinado y otro que devuelve el árbitro por un nombre dado.

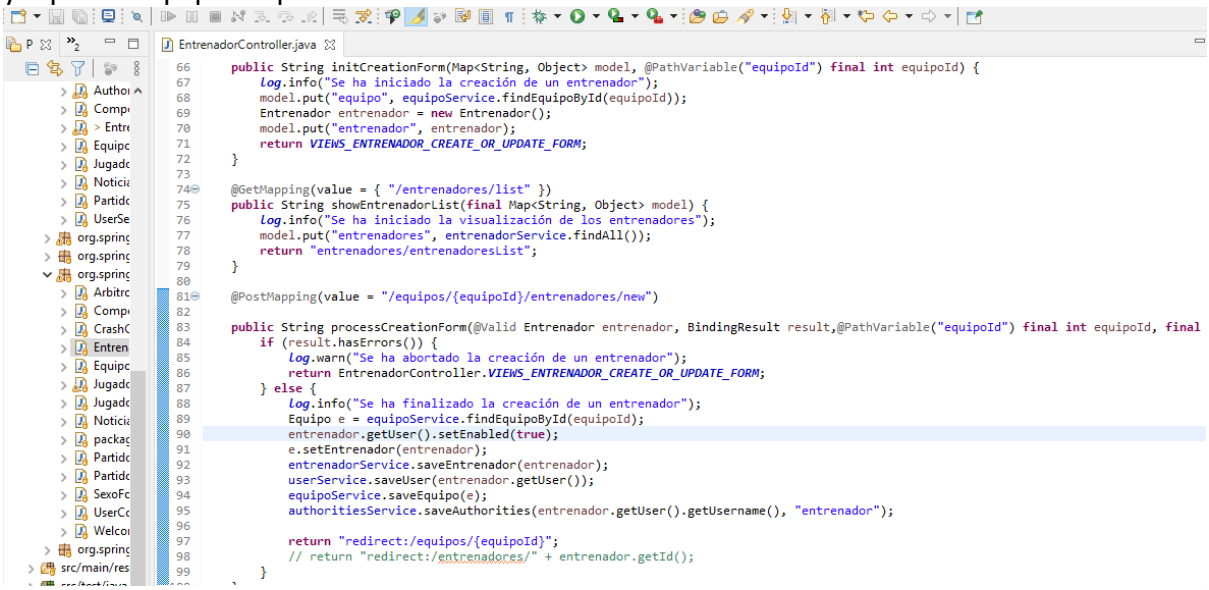


Controlador

Controlador EntrenadorController, localizado en la ruta
src/main/java/org.springframework.samples.petclinic/web.
EntrenadorController se encarga de responder a los eventos que se producen en la vista de Equipo, invocando cambios en el modelo, y mapeando las peticiones HTTP.

Por ejemplo, podemos ver que implementa mapeo para el Get y el Post de cuando creas un nuevo entrenador (initCreationForm y processCreationForm) y de cómo invoca cambios en el modelo. Por ejemplo en el InitCreationForm podemos observar que añadimos un @PathVariable que indica que un parámetro de método debe estar vinculado a una variable de plantilla de URI, en este caso es el equipo al que va a pertenecer el entrenador. Vemos que en el modelo asociamos el equipo con la id del equipo y seguidamente el entrenador.

De la misma forma, en el processCreationForm, añadimos esta @PathVariable. Dentro de este método, una vez definido el entrenador en el equipo, procedemos a guardar en el servicio el entrenador y el equipo. Un entrenador siempre ha de crearse desde un equipo, ya que un equipo no puede estar sin entrenador.



En esta misma captura observamos el método por el cual se muestra la lista que contiene a todos los entrenadores creados.

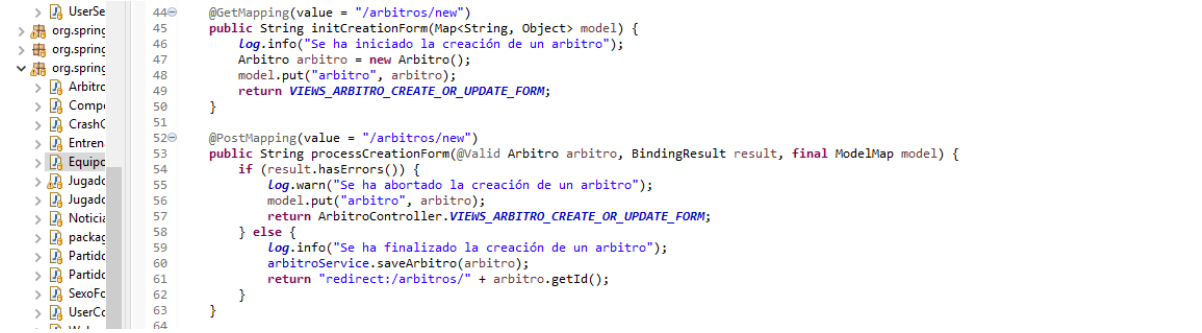
En el controlador de entrenadores también se implementan peticiones Get y Post para editar a un entrenador. Básicamente tenemos un entrenador y un equipo, al entrenador con un set le modificamos el id y al equipo con un set le modificamos el entrenador por el antes modificado, posteriormente, se guarda el nuevo entrenador en el servicio.

Entrenador no tiene una petición Get de borrado ya que un entrenador se elimina si se elimina el equipo en el que está.

Nombre: Marcos Rodríguez Osorio Grupo: G3-08

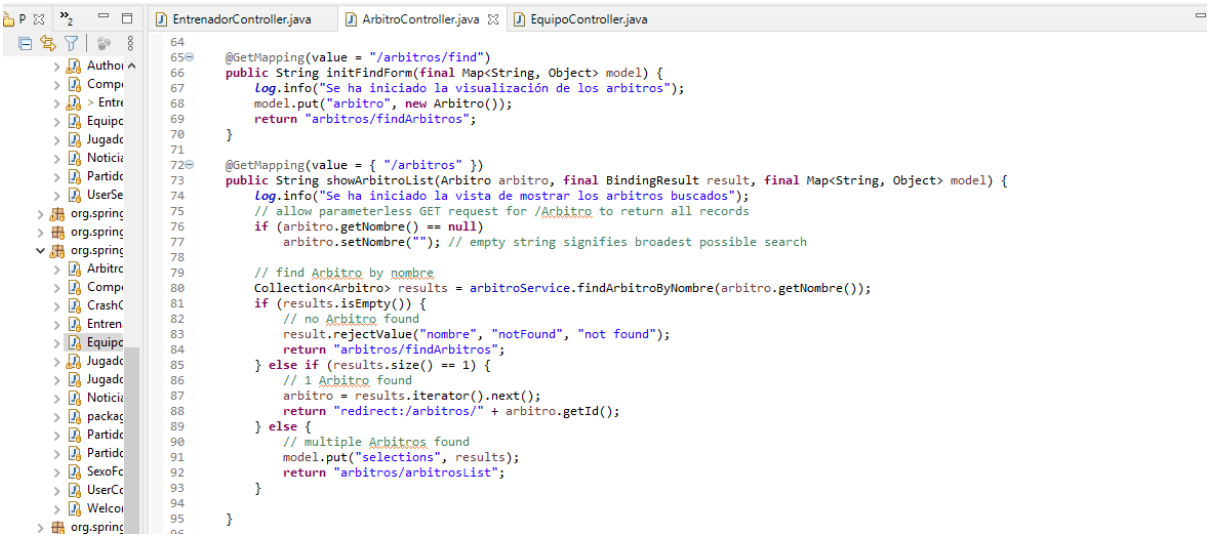
Controlador ArbitroController, localizado en la ruta src/main/java/org.springframework.samples.petclinic/web.
ArbitroController se encarga de responder a los eventos que se producen en la vista de Arbitro, invocando cambios en el modelo, y mapeando las peticiones HTTP.

Por ejemplo, podemos ver que implementa mapeo para el Get y el Post de cuando creas un nuevo árbitro (initCreationForm y processCreationForm) y de cómo invoca cambios en el modelo. Finalmente, añade en el repositorio, a través del servicio, el árbitro.



```
@GetMapping(value = "/arbitros/new")
public String initCreationForm(Map<String, Object> model) {
    log.info("Se ha iniciado la creación de un arbitro");
    Arbitro arbitro = new Arbitro();
    model.put("arbitro", arbitro);
    return VIEWS_ARBITRO_CREATE_OR_UPDATE_FORM;
}

@PostMapping(value = "/arbitros/new")
public String processCreationForm(@Valid Arbitro arbitro, BindingResult result, final ModelMap model) {
    if (result.hasErrors()) {
        log.warn("Se ha abortado la creación de un arbitro");
        model.put("arbitro", arbitro);
        return ArbitroController.VIEWS_ARBITRO_CREATE_OR_UPDATE_FORM;
    } else {
        log.info("Se ha finalizado la creación de un arbitro");
        arbitroService.saveArbitro(arbitro);
        return "redirect:/arbitros/" + arbitro.getId();
    }
}
```

Además podemos ver la implementación del método find y del showList, estos 2 métodos los realicé con ayuda de Gonzalo.
Básicamente, creamos un buscador que encuentre el árbitro que determinemos por su nombre y mostraremos una lista de todos los árbitros disponibles.

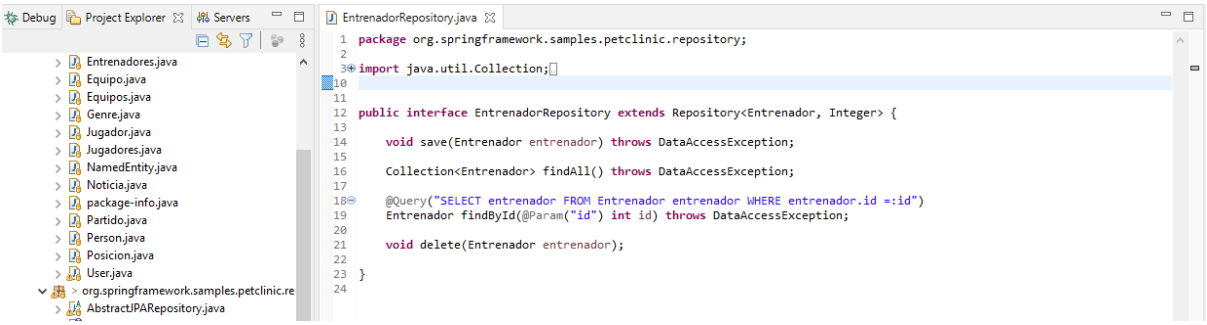


```
@GetMapping(value = "/arbitros/find")
public String initFindForm(final Map<String, Object> model) {
    log.info("Se ha iniciado la visualización de los arbitros");
    model.put("arbitro", new Arbitro());
    return "arbitros/findArbitros";
}

@GetMapping(value = { "/arbitros" })
public String showArbitroList(Arbitro arbitro, final BindingResult result, final Map<String, Object> model) {
    log.info("Se ha iniciado la vista de mostrar los arbitros buscados");
    // allow parameterless GET request for /Arbitro to return all records
    if (arbitro.getNombre() == null)
        arbitro.setNombre(""); // empty string signifies broadest possible search

    // find Arbitro by nombre
    Collection<Arbitro> results = arbitroService.findArbitroByNombre(arbitro.getNombre());
    if (results.isEmpty()) {
        // no Arbitro found
        result.rejectValue("nombre", "notFound", "not found");
        return "arbitros/findArbitros";
    } else if (results.size() == 1) {
        // 1 Arbitro found
        arbitro = results.iterator().next();
        return "redirect:/arbitros/" + arbitro.getId();
    } else {
        // multiple Arbitros found
        model.put("selections", results);
        return "arbitros/arbitrosList";
    }

}
```

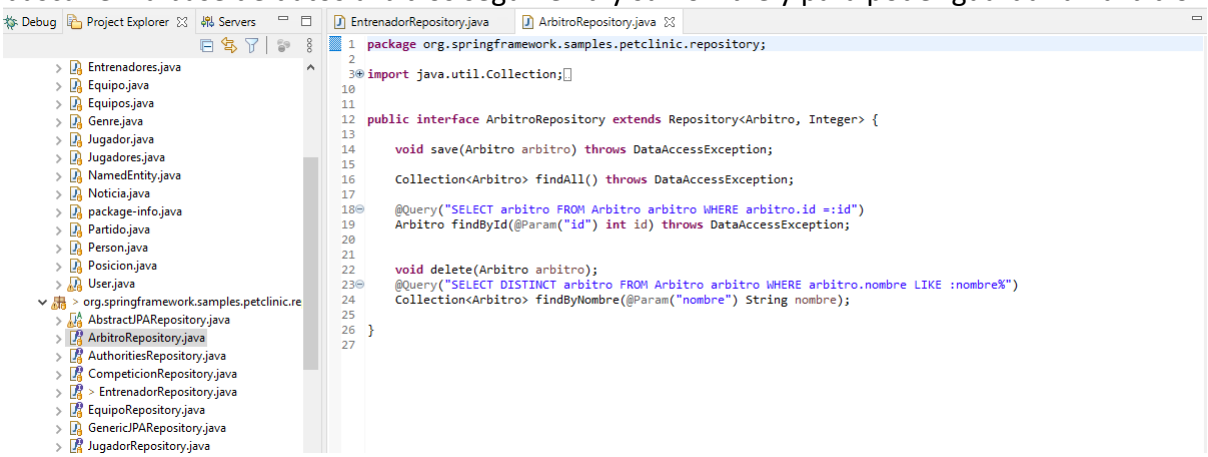
Nombre: Marcos Rodríguez Osorio Grupo: G3-08

Repositorio

EntrenadorRepository, localizado en la ruta src/main/java/org.springframework.samples.petclinic/repository
EntrenadorRepository es una clase que contiene la lógica requerida para acceder a los datos del repositorio de Entrenador en la base de datos, haciendo las consultas a esta de forma transparente.
Tiene implementada la lógica necesaria para poder obtener todos los entrenadores, para poder buscar en la base de datos entrenadores según el Id y para poder guardar y borrar un entrenador.



ArbitroRepository, localizado en la ruta src/main/java/org.springframework.samples.petclinic/repository
ArbitroRepository es una clase que contiene la lógica requerida para acceder a los datos del repositorio de Arbitro en la base de datos, haciendo las consultas a esta de forma transparente.
Tiene implementada la lógica necesaria para poder obtener todos los árbitros, para poder buscar en la base de datos árbitros según el Id y su nombre y para poder guardar un árbitro.



Validador y anotación asociada (si aplica, máximo de dos ejemplos)

PartidoValidator, realizado por mis compañeros Gonzalo Fernandez y Javier Grosso.
Situado en la ruta src/main/java/org.springframework.samples.petclinic/web/validators
Es un validador necesario para los requisitos de negocio, a parte de otros, RN2.

La validación consiste en comprobar que en un partido no puede jugar un jugador que esté lesionado.

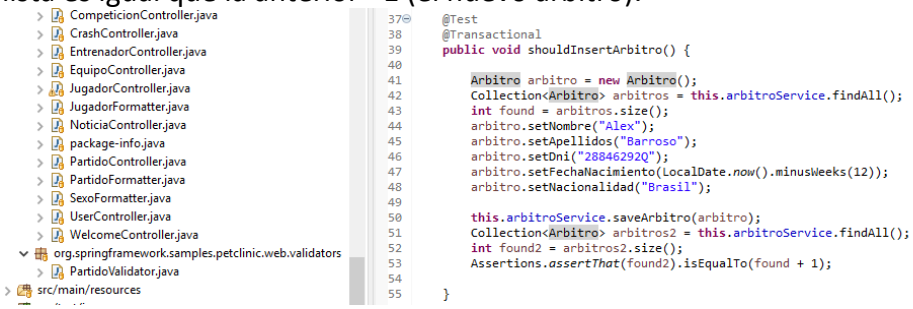
```java
package org.springframework.samples.petclinic.web.validators;

import java.util.Collection;

@Slf4j
public class PartidoValidator implements Validator{


    private PartidoService partidoService;

    public PartidoValidator(PartidoService partidoService) {
        this.partidoService=partidoService;
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return Partido.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        log.info("Se ha iniciado el validador de Partido");
        Collection<Partido> restoDePartidos = partidoService.findAll();
                }
           // Este codigo realiza la regla de negocio 2, Un jugador que esté lesionado no
           // podrá ser inscrito en un
           // partido, hasta que no le hayan dado el alta de su lesión.
           log.info("Se van a validar las lesiones de los jugadores");
           for (Jugador jug : jugadores)
               if (jug.getLesion()) {
                   errors.rejectValue("jugadoresParticipantes",
                       "Un jugador que esté lesionado no podrá ser inscrito en un partido",
                       "Un jugador que esté lesionado no podrá ser inscrito en un partido");
               }

```

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

Test 1: realizado a ArbitroService dentro de la clase ArbitroServiceTest en la ruta src/test/java/org.springframework.samples.petclinic/service
Realizado en colaboración con Gonzalo Fernandez

Esta prueba consiste en crear un árbitro, guardarlo en la base de datos por medio del servicio, y comprobar que se ha guardado bien en la base de datos.
Para ello, creamos un arbitro vacío y obtenemos el tamaño de la lista de todos los árbitros. Añadimos unos atributos aleatorios y guardamos el árbitro en la base de datos. Obtenemos el tamaño de la nueva lista de árbitros (ahora con un arbitro más) y comprobamos que esta lista es igual que la anterior +1 (el nuevo arbitro).

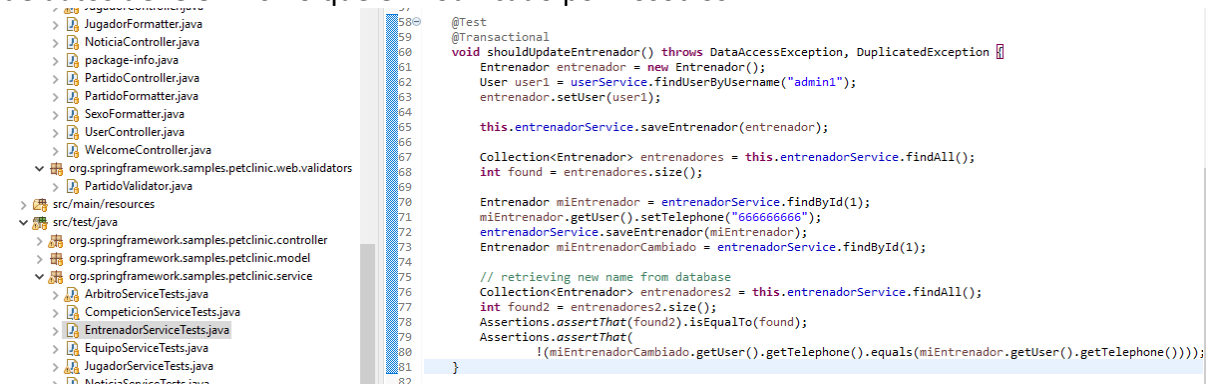



```java
@Test
@Transactional
public void shouldInsertArbitro() {

    Arbitro arbitro = new Arbitro();
    Collection<Arbitro> arbitros = this.arbitroService.findAll();
    int found = arbitros.size();
    arbitro.setNombre("Alex");
    arbitro.setApellidos("Barroso");
    arbitro.setDni("28846292Q");
    arbitro.setFechaNacimiento(LocalDate.now().minusWeeks(12));
    arbitro.setNacionalidad("Brasil");

    this.arbitroService.saveArbitro(arbitro);
    Collection<Arbitro> arbitros2 = this.arbitroService.findAll();
    int found2 = arbitros2.size();
    Assertions.assertThat(found2).isEqualTo(found + 1);

}
```

Test 2: realizado a EntrenadorService con ayuda de Gonzalo.

Esta prueba consiste en modificar un entrenador, guardarlo en la base de datos por medio del servicio, y comprobar que se ha guardado bien en la base de datos.
Para ello, creamos un entrenador vacío y le cambiamos su User por uno determinado. Guardamos el entrenador en la base de datos. Obtenemos el tamaño de la lista de entrenadores. Creamos un nuevo entrenador que es el entrenador que anteriormente se había creado y le modificamos el teléfono. Guardamos en la base de datos el entrenador

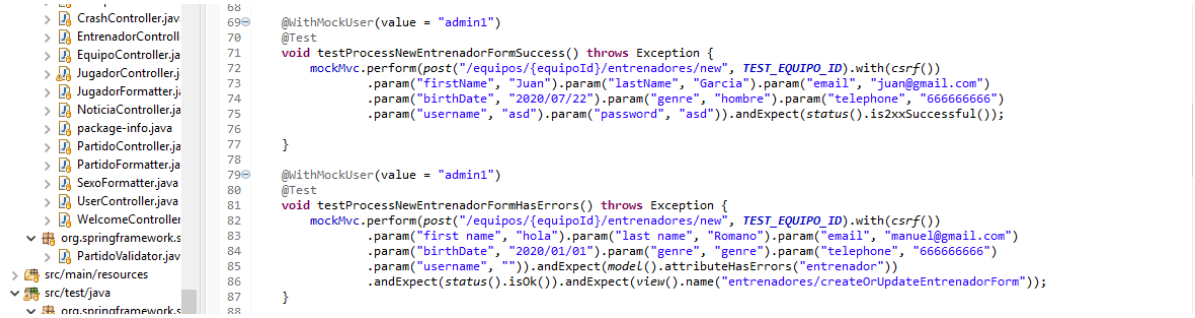
modificado y comprobamos que la lista original es igual que la actual (con el entrenador nuevo). Posteriormente, comprobamos que el atributo del entrenador obtenido de la base de datos tiene el mismo que el modificado por nosotros.



Pruebas de controlador

Test de controlador realizado en EntrenadorControllerTest
src/test/java/org.springframework.samples.petclinic/controller

- Caso positivo: comprueba que si al rellenar correctamente el formulario para crear un nuevo entrenador, sea satisfactoria.
- Caso negativo: comprueba que si al no rellenar correctamente el formulario para crear un nuevo entrenador, por ejemplo dejando el nombre vacío, esta, nos redirige de nuevo a la vista de la creación de un entrenador.



Principales problemas encontrados

El único problema que me he encontrado desarrollando este proyecto, ha sido la falta de implicación de mi pareja Stefan. En los 2 primeros Sprint estuvo un poco más activo pero en el Sprint 3 y 4 ha estado prácticamente ausente, justificando que era por motivos personales. Yo siempre he estado implicado en el proyecto y comunicándome con él para realizar las tareas, pero he notado su falta de interés. Puedo entender que te ocurran desgracias en tu vida personal pero el mínimo gesto sería comentar al grupo que no estas disponible para realizar el trabajo y así el equipo organizarse a la hora de afrontar el proyecto, cosa que no hizo. Causa de esto, he llevado una sobrecarga de trabajo mayor sumado a la inexperiencia de desarrollo del proyecto, en este punto debo agradecer a mis demás compañeros del grupo de trabajo, Gonzalo Fernández, Ignacio Sanabria, Javier Grosso y Juan Luis Muñoz, ya que sin ellos no podría haber realizado mis tareas, me han ayudado constantemente en la ausencia de mi pareja, y gracias a ellos he podido sobrellevar el trabajo.

Nombre: Marcos Rodríguez Osorio Grupo: G3-08

Otros comentarios

Ninguno.