Nombre: Ignacio Sanabria Alonso de Caso 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.git
Nombre de usuario en GitHub: ignsanalo
Rama del usuario en Github: ignsanalo, ignsanalo2, ignsanalo3

Participación en el proyecto

Historias de usuario en las que he participado

He desarrollado junto a mi compañero Juan Luis Muñoz Navarro las historias de usuario:

- **H1** - Creación y eliminación de un equipo en una competición.
- **H3** - Creación, edición y eliminación de una competición.
- **H6** - Inscripción de un equipo en una competición.
- **H7** - Creación, edición y eliminación de equipo.
- **H8** - Creación, edición y eliminación de jugadores.
- **H10** - Visualización de jugadores de otros equipos.
- **H13** - Visualización de los equipos.
- **H17** - Visualización de las competiciones.

Y las siguientes reglas de negocio:

- **RN1** - Entrenador un solo equipo.
- **RN5** - No se pueden registrar dos jugadores con el mismo DNI.

Funcionalidad implementada

He implementado junto a mi compañero Juan Luis Muñoz Navarro las siguientes funcionalidades:

- **Jugador**: entidad, vistas, controlador, servicio, repositorio, formatter y funcionalidades (visualizar, crear, editar, eliminar y buscar).
- **Equipo**: vistas, controlador, servicio, repositorio y funcionalidades (visualizar, crear, editar, eliminar y buscar).
- **Competición**: vistas (salvo list), controlador, servicio, repositorio y funcionalidades (crear, editar y eliminar)

Pruebas implementadas

Pruebas unitarias

He creado junto a mi compañero Juan Luis Muñoz Navarro estas pruebas unitarias:

- **Servicios**: Jugador y Equipo.
- **Controladores**: Jugador y Competición.

Hacen un total de 4 clases de pruebas unitarias con un total de 30 métodos anotados con @Test.

Pruebas de Controlador

- **Jugador**: He creado 5 casos de prueba positivos y 2 negativos de controlador para la HU-8 y 2 casos positivos para la HU-10.
- **Competición**: He creado 5 casos de prueba positivos y 2 negativos de controlador para la HU-3 y 2 casos positivos de visualización.

Ejemplos de funcionalidades implementadas

Entidad (Jugador)

La entidad Jugador extiende de persona (esto significa que hereda sus atributos) y está ubicada en la ruta *src/main/java/org.springframework.samples.petclinic.model*. He creado 4 atributos:

- **tarjetaAmarilla**: Corresponde al número de tarjetas amarillas de un jugador. Es de tipo 'Integer'.
- **tarjetaRoja**: Corresponde al número de tarjetas rojas de un jugador. Es de tipo 'Integer'.
- **lesión**: Si un jugador está o no lesionado. Es de tipo 'Boolean'.
- **equipoId**: Hace referencia al equipo al que pertenece ese jugador mediante su ID. Es de tipo 'Equipo' (clase implementada) y su relación es '@ManyToOne' (Varios jugadores en un solo equipo).

Esta entidad quedaría implementada de la siguiente manera:

```java
package org.springframework.samples.petclinic.model;

import javax.persistence.Column;

@Getter
@Setter
@Entity
@Table(name = "jugador")
public class Jugador extends Person {

    @Column(name = "tarjetaAmarilla")
    protected Integer   tarjetaAmarilla;

    @Column(name = "tarjetaRoja")
    protected Integer   tarjetaRoja;

    @Column(name = "lesion")
    protected Boolean   lesion;

    @ManyToOne
    @JoinColumn(name = "equipoId")
    private Equipo      equipo;

}
```

Servicio (JugadorService)

En el servicio de Jugador interviene *JugadorRepository*, ya que nos hará falta a la hora de implementar algunos métodos pertenecientes a estas clases. *JugadorService* está ubicada en la ruta *src/main/java/org.springframework.samples.petclinic.service*. Esta clase se compone de un total de 6 métodos que explicaremos a continuación:

- **JugadorService**: Este método nos sirve para llamar a las constante *JugadorRepository* con el fin de utilizar libremente los métodos de dicha clase.
- **saveJugador**: Este método nos sirve para guardar la edición de alguno/s de los atributos descritos en la entidad *Jugador* y *Person*. Cabe mencionar que no se puede guardar un DNI si es igual al de otro jugador (esta restricción está eliminada)
- **findJugadorById**: Con este método podemos buscar un jugador por su ID (cada jugador tiene un ID que no se puede repetir)
- **findJugadorByNombre**: Con este método podemos buscar uno o varios jugadores por su nombre.
- **findJugadores**: Con este método podemos buscar todos los jugadores, independientemente del equipo al que pertenezcan.
- **deleteJugador**: Este método nos sirve para eliminar a un jugador.

Este servicio quedaría implementado de la siguiente forma:

```java
package org.springframework.samples.petclinic.service;

import java.util.Collection;

@Service
public class JugadorService {

    private JugadorRepository jugadorRepository;


    @Autowired
    public JugadorService(final JugadorRepository jugadorRepository) {
        this.jugadorRepository = jugadorRepository;
    }

    @Transactional(rollbackFor = DuplicatedJugadorDNIException.class)
    public void saveJugador(final Jugador jugador) throws DataAccessException, DuplicatedJugadorDNIException {
        Jugador otherJugador = jugador.getEquipo().getJugadorwithIdDifferent(jugador.getDni(), jugador.getId());
        if (StringUtils.hasLength(jugador.getDni()) && otherJugador != null && otherJugador.getId() != jugador.getId()) {
            throw new DuplicatedJugadorDNIException();
        } else {
            this.jugadorRepository.save(jugador);
        }
    }
    @Transactional(readOnly = true)
    public Jugador findJugadorById(final int id) throws DataAccessException {
        return this.jugadorRepository.findById(id);
    }

    @Transactional(readOnly = true)
    public Collection<Jugador> findJugadorByNombre(final String nombre) throws DataAccessException {
        return this.jugadorRepository.findJugadorByNombre(nombre);
    }

    @Transactional(readOnly = true)
    public Collection<Jugador> findJugadores() throws DataAccessException {
        return this.jugadorRepository.findAll();
    }

    @Transactional
    public void deleteJugador(final Jugador jugador) {
        this.jugadorRepository.delete(jugador);
    }

}
```

Controlador (JugadorController)

En el controlador de Jugador interviene *JugadorService* y *EquipoService*, ya que nos hará falta a la hora de implementar algunos métodos presentes en dichas clases. *JugadorController* está ubicada en la ruta *src/main/java/org.springframework.samples.petclinic.web*. Esta clase se compone de un total de 10 métodos que explicaremos a continuación:

- **JugadorController**: Este método nos sirve para llamar a las constantes *JugadorService* y *EquipoService* con el fin de utilizar libremente los métodos de dichas clases.
- **initJugadorBinder**: Este método nos sirve para poder operar con el ID de un jugador.
- **initCreationForm**: Con este método podemos ir al formulario de crear jugador.
- **processCreationForm**: Con este método podemos enviar los datos del formulario de crear jugador.
- **initFindForm**: Con este método podemos ir al buscador de jugadores.
- **processFindForm**: Con este método podemos enviar los datos que hemos escrito en el buscador de jugadores.
- **initUpdateForm**: Con este método podemos acceder al formulario de editar jugador.
- **processUpdateForm**: Con este método podemos enviar los datos del formulario de editar jugador.
- **showJugadorList**: Con este método podemos mostrar una lista de los jugadores.
- **processDeleteForm**: Con este método podemos eliminar a un jugador.

Quiero subrayar que 'init' hace referencia al 'get', es decir, accede a la URL descrita. En cambio, 'process', indica el envío de datos escritos a través de un 'post'. Este controlador quedaría implementado de la siguiente forma:

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

import java.util.Collection;

@Controller
public class JugadorController {

    private static final String     VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM = "jugadores/createOrUpdateJugadorForm";

    private final JugadorService    jugadorService;
    private final EquipoService     equipoService;


    @Autowired
    public JugadorController(final JugadorService jugadorService, final EquipoService equipoService) {
        this.jugadorService = jugadorService;
        this.equipoService = equipoService;
    }

    @InitBinder("jugador")
    public void initJugadorBinder(final WebDataBinder dataBinder) {
        dataBinder.setDisallowedFields("id");
    }

    @GetMapping(value = "/equipos/{equipoId}/jugadores/new")
    public String initCreationForm(final ModelMap model, @PathVariable("equipoId") final int equipoId) {
        Jugador jugador = new Jugador();
        jugador.setEquipo(this.equipoService.findEquipoById(equipoId));
        model.put("jugador", jugador);
        return JugadorController.VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM;
    }
```

```java
    @PostMapping(value = "/equipos/{equipoId}/jugadores/new")
    public String processCreationForm(@Valid final Jugador jugador, final BindingResult result, final ModelMap model,
        @PathVariable("equipoId") final int equipoId) {

        if (result.hasErrors()) {
            model.put("jugador", jugador);
            return JugadorController.VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM;
        } else {
            Equipo e = this.equipoService.findEquipoById(equipoId);
            try {
                e.addJugador(jugador);
                this.jugadorService.saveJugador(jugador);
                this.equipoService.saveEquipo(e);
            } catch (DuplicatedJugadorDNIException ex) {
                result.rejectValue("name", "duplicate", "already exists");
                return JugadorController.VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM;
            }
            return "redirect:/equipos/{equipoId}";
        }
    }

    @GetMapping(value = "/jugadores/find")
    public String initFindForm(final Map<String, Object> model) {
        model.put("jugador", new Jugador());
        return "jugadores/findJugadores";
    }

    @GetMapping(value = "/jugadores")
    public String processFindForm(final Jugador jugador, final BindingResult result, final Map<String, Object> model) {

        // allow parameterless GET request for /Jugadores to return all records
        if (jugador.getNombre() == null) {
            jugador.setNombre(""); // empty string signifies broadest possible search
        }

        // find Jugadores by nombre
        Collection<Jugador> results = this.jugadorService.findJugadorByNombre(jugador.getNombre());
        if (results.isEmpty()) {
            // no Jugadores found
            result.rejectValue("nombre", "notFound", "not found");
            return "jugadores/findJugadores";
        } else {
            // multiple Jugadores found
            model.put("selections", results);
            return "jugadores/jugadoresList";
        }
    }

    @GetMapping(value = "/equipos/{equipoId}/jugadores/{jugadorId}/edit")
    public String initUpdateForm(@PathVariable("jugadorId") final int jugadorId, final ModelMap model) {
        Jugador jugador = this.jugadorService.findJugadorById(jugadorId);
        model.put("jugador", jugador);
        return JugadorController.VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM;
    }

    @PostMapping(value = "/equipos/{equipoId}/jugadores/{jugadorId}/edit")
    public String processUpdateForm(@Valid final Jugador jugador, final BindingResult result,
        @PathVariable("jugadorId") final int jugadorId, @PathVariable("equipoId") final int equipoId, final ModelMap model) {
        if (result.hasErrors()) {
            model.put("jugador", jugador);
            return JugadorController.VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM;
        } else {
            try {
                Equipo e = this.equipoService.findEquipoById(equipoId);
                jugador.setId(jugadorId);
                jugador.setEquipo(e);
                this.jugadorService.saveJugador(jugador);
            } catch (DuplicatedJugadorDNIException ex) {
                result.rejectValue("name", "duplicate", "already exists");
                return JugadorController.VIEWS_JUGADOR_CREATE_OR_UPDATE_FORM;
            }
            return "redirect:/equipos/{equipoId}";
        }
    }

    @GetMapping("/jugadores/list")
    public String showJugadorList(final Map<String, Object> model) {
        Jugadores jugadores = new Jugadores();
        jugadores.getJugadorList().addAll(this.jugadorService.findJugadores());
        model.put("jugadores", jugadores);
        return "jugadores/jugadoresList";
    }

    @GetMapping(value = "/equipos/{equipoId}/jugadores/{jugadorId}/delete")
    public String processDeleteForm(@PathVariable("jugadorId") final int jugadorId, @PathVariable("equipoId") final int equipoId) {

        Jugador jugador = this.jugadorService.findJugadorById(jugadorId);
        Equipo equipo = this.equipoService.findEquipoById(equipoId);
        equipo.removeJugador(jugador);
        this.jugadorService.deleteJugador(jugador);
        return "redirect:/equipos/{equipoId}";
    }
}
```

Repositorio (JugadorRepository)

El repositorio de Jugador está ubicado en la ruta *src/main/java/org.springframework.samples.petclinic.repository*. Esta clase se compone de un total de 5 métodos que explicaremos a continuación:

- **save**: Este método guarda al jugador.
- **findById**: Este método busca a un jugador por su ID.
- **findJugadorByNombre**: Este método busca a uno o varios jugadores por su nombre.
- **findAll**: Este método encuentra a todos los jugadores.
- **delete**: Este método borra al jugador seleccionado.

Este repositorio quedaría implementado de la siguiente manera:

```
package org.springframework.samples.petclinic.repository;

import java.util.Collection;

public interface JugadorRepository extends Repository<Jugador, String> {

    void save(final Jugador jugador) throws DataAccessException;
    @Query("SELECT jugador FROM Jugador jugador WHERE jugador.id =:id")
    Jugador findById(@Param("id") int id) throws DataAccessException;

    @Query("SELECT DISTINCT jugador FROM Jugador jugador WHERE jugador.nombre LIKE :nombre%")
    Collection<Jugador> findJugadorByNombre(@Param("nombre") String nombre);

    Collection<Jugador> findAll() throws DataAccessException;

    void delete(Jugador jugador);

}
```

Formatter (JugadorFormatter)

En el formatter de Jugador interviene *JugadorService*, ya que nos hará falta a la hora de implementar algún método presente en la clase. *JugadorFormatter* está ubicado en la ruta *src/main/java/org.springframework.samples.petclinic.web*. Esta clase se compone de un total de 3 métodos que explicaremos a continuación:

- **JugadorFormatter**: Este método nos sirve para llamar a la constante *JugadorService* con el fin de utilizar libremente los métodos de dicha clase.
- **print**: Este método nos sirve para imprimir por pantalla el nombre completo de un jugador.
- **parse**: Este método imprime por pantalla el nombre completo de un jugador cuando es buscado por su nombre. Si no lo encuentra, devuelve que el jugador no ha sido encontrado.

Este formatter quedaría implementado de la siguiente manera:

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

import java.text.ParseException;

@Component
public class JugadorFormatter implements Formatter<Jugador> {

    private final JugadorService jugadorService;

    @Autowired
    public JugadorFormatter(final JugadorService jugadorService) {
        this.jugadorService = jugadorService;
    }

    @Override
    public String print(final Jugador jugador, final Locale locale) {
        return jugador.getNombre()+" "+jugador.getApellidos();
    }

    @Override
    public Jugador parse(final String text, final Locale locale) throws ParseException {
        Collection<Jugador> findJugadores = jugadorService.findJugadores();
        for (Jugador jugador : findJugadores)
            if ((jugador.getNombre()+" "+jugador.getApellidos()).equals(text))
                return jugador;
        throw new ParseException("jugador no encontrado: " + text, 0);
    }

}
```

Ejemplos de pruebas implementadas

Pruebas unitarias (EquipoServiceTests)

Las pruebas unitarias de Equipo están ubicadas en la ruta *src/test/java/org.springframework.samples.petclinic.service*. A continuación, vamos a explicar dos de sus métodos:

- **shouldFindEquipoById**: Este método prueba la búsqueda del equipo que tiene ID=1.
- **shouldUpdateEquipo**: Este método prueba la actualización del nombre del equipo. En este caso, al nombre antiguo le añade una X.

Esta clase quedaría implementada de la siguiente manera:

```
@Test
void shouldFindEquipoById() {
    Equipo equipo = this.equipoService.findEquipoById(1);
    Assertions.assertThat(equipo != null);
}

@Test
@Transactional
void shouldUpdateEquipo() {
    Equipo Equipo = this.equipoService.findEquipoById(1);
    String oldNombre = Equipo.getNombre();
    String newNombre = oldNombre + "X";

    Equipo.setNombre(newNombre);
    this.equipoService.saveEquipo(Equipo);

    // retrieving new name from database
    Equipo = this.equipoService.findEquipoById(1);
    Assertions.assertThat(Equipo.getNombre()).isEqualTo(newNombre);
}
```

Pruebas de controlador (JugadorControllerTests)

Las pruebas de controlador de Jugador están ubicadas en la ruta *src/test/java/org.springframework.samples.petclinic.controller*. A continuación, vamos a explicar dos de sus métodos:

- **testShowListJugadores**: Este método prueba la visualización de una lista de jugadores.
- **testDeleteJugador**: Este método prueba la eliminación de un jugador.

Esta clase quedaría implementada de la siguiente manera:

```
@WithMockUser(value = "admin1")
@Test
void testShowListJugadores() throws Exception {
    this.mockMvc.perform(MockMvcRequestBuilders.get("/jugadores/list")).andExpect(MockMvcResultMatchers.status()
        .isOk()).andExpect(MockMvcResultMatchers.model().attributeExists("jugadores"))
        .andExpect(MockMvcResultMatchers.view().name("jugadores/jugadoresList"));
}

@WithMockUser(value = "admin1")
@Test
void testDeleteJugador() throws Exception {
    this.mockMvc.perform(MockMvcRequestBuilders.get("/equipos/{equipoId}/jugadores/{jugadorId}/delete",
        JugadorControllerTests.TEST_EQUIPO_ID, JugadorControllerTests.TEST_JUGADOR_ID))
        .andExpect(MockMvcResultMatchers.model().attributeDoesNotExist("jugador"))
        .andExpect(MockMvcResultMatchers.view().name("redirect:/equipos/{equipoId}"));
}
```

Principales problemas encontrados

Por lo general, el desarrollo del proyecto ha ido de forma notable en todos los aspectos. Solamente quería mencionar un aspecto negativo que nos ha acompañado durante todo el cuatrimestre. Uno de los compañeros, Stefan, no se ha implicado ni ha trabajado lo suficiente. Puedo "entender" que durante las reuniones no esté participativo, pero no que deje a un lado las tareas que se le ha asignado. O si las hace, que la haga mal. Es decir, sin preocuparse en la redacción o en lo que pide realmente el ejercicio. Esto me ha afectado, sobre todo en el Sprint 3, ya que tuvimos que ayudar a su compañero Marcos con tareas que, por motivos que no sé si son ciertos, Stefan no pudo realizar. Seguramente Marcos pueda explicar mejor la sobrecarga de trabajo que ha llevado él a lo largo de la asignatura.

Otros comentarios

Además de las implementaciones, he llevado acabo otras tareas de organización, documentación y revisión.

En cuanto a organización, siempre he estado implicado en proponer y realizar un reparto equitativo de tareas a cada subgrupo del proyecto. Estas tareas fueron designadas por destrezas o de forma aleatoria.

Con respecto a revisión por Sprint, junto a otros compañeros, nos hemos encargado de revisar que todo vaya perfectamente, así como mejorar algunos aspectos de entregas anteriores.

Por último, para la documentación se me han asignado las siguientes tareas:

- Descripción general del sistema, de los perfiles de usuario, de las historias de usuario y de las reglas de negocio (junto a mi compañero Juanlu).
- Redacción del *Documento Fin de Sprint* (de todas las entregas).
- Estética, formato y tabla de contenido de la documentación enviada.
- Diagrama de capas (junto a mi compañero Juanlu)