# 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: juamunnav
Rama del usuario en Github: juamunnav, juamunnav 1, revisión (esta última también ha sido utilizada por más miembros del grupo)

## Participación en el proyecto

### Historias de usuario en las que he participado

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

### Funcionalidad implementada

He implementado junto a mi compañero Juan Luis Muñoz Navarro:
• Entidad jugador con su respectivo servicio, repositorio y controlador.
• Entidad equipo con su respectivo servicio, repositorio y controlador.
• Competición con todo menos la propia entidad de competición. Estos son sus respectivas vistas, controladores, servicios, repositorios y funcionalidades (visualizar, crear, editar, eliminar, buscar…).

Para la entidad jugador hemos creado también el JugadorFormatter.

## Pruebas implementadas

### Pruebas unitarias

He creado tests unitarios para 2 servicios (Jugador, Equipo) y 2 controladores (Jugador, Competición). Eso hace un total de 4 clases de test unitarios con un total de 28 métodos anotados con @Test.

### Pruebas de Controlador

- Jugador: He creado 5 casos de prueba positivo y 2 negativos de controlador para la HU-8 y 2 casos positivo para la HU-11.
- Competición: He creado 5 casos de prueba positivo y 2 negativos de controlador para la HU-3 y 2 casos positivo para la HU-17.

# Ejemplos de funcionalidades implementadas

## Entidades (máximo de dos ejemplos)

La ruta de la entidad Equipo:
\src\main\java\org\springframework\samples\petclinic\model\Equipo.java

La entidad Equipo extiende de BaseEntity, al estar equipo dentro de una competición se establece una relación ManyToOne, que quiere decir que muchos equipos pueden pertenecer a una competición.
A parte de los atributos propio de equipo, como son nombre y lugar, también tiene una relación OneToMany con la clase Jugador, que significa que un equipo puede tener muchos jugadores.
También tiene una relación con la entidad Entrenador OneToOne, por la que se establece que un Equipo solo puede tener un Entrenador.
Por último, tenemos la relación con la entidad Partido ManyToMany, en la que establecemos que muchos equipos pueden participar en muchos partidos.

Con el @JoinColumn nos traemos el nombre del atributo que esta alojada en la base datos de las clases a la que referencias todas estas relaciones.

```java
import java.util.ArrayList;

@Getter
@Setter
@Entity
@Table(name = "equipos")
public class Equipo extends BaseEntity {

    @ManyToOne
    @JoinColumn(name = "competicionId")
    private Competicion     competicion;

    @Column(name = "nombre")
    @Size(min = 4, max = 50)
    @NotEmpty
    private String  nombre;

    @NotEmpty
    @Column(name = "lugar")
    private String          lugar;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "equipo")
    private Set<Jugador>    jugadores;

    @OneToOne(cascade= CascadeType.ALL)
    @JoinColumn(name = "entrenador_id")
    private Entrenador entrenador;
```

```java
@ManyToMany
@JoinTable(name = "equipo_partidos", joinColumns = @JoinColumn(name = "equipo_id"),
inverseJoinColumns = @JoinColumn(name = "partido_id"))
private Set<Partido> partidos;

protected Set<Jugador> getJugadoresInternal() {
    if (this.jugadores == null) {
        this.jugadores = new HashSet<>();
    }
    return this.jugadores;
}

protected void setJugadoresInternal(final Set<Jugador> jugadores) {
    this.jugadores = jugadores;
}

public List<Jugador> getJugadores() {
    List<Jugador> sortedJugadores = new ArrayList<>(this.getJugadoresInternal());
    PropertyComparator.sort(sortedJugadores, new MutableSortDefinition("name", true, true));
    return Collections.unmodifiableList(sortedJugadores);
}

public void addJugador(final Jugador jugador) {
    this.getJugadoresInternal().add(jugador);
    jugador.setEquipo(this);
}

public boolean removeJugador(final Jugador jugador) {
    return this.getJugadoresInternal().remove(jugador);
}
public Jugador getJugador(final String name) {
    return this.getJugador(name, false);
}

public Jugador getJugadorwithIdDifferent(String dni, final Integer id) {
    dni = dni.toLowerCase();
    for (Jugador jugador : this.getJugadoresInternal()) {
        String compName = jugador.getDni();
        compName = compName.toLowerCase();
        if (compName.equals(dni) && jugador.getId() != id) {
            return jugador;
        }
    }
    return null;
}

public Jugador getJugador(String name, final boolean ignoreNew) {
    name = name.toLowerCase();
    for (Jugador jugador : this.getJugadoresInternal()) {
        if (!ignoreNew || !jugador.isNew()) {
            String compName = jugador.getNombre();
            compName = compName.toLowerCase();
            if (compName.equals(name)) {
                return jugador;
            }
        }
    }
```

## Servicio

En el **EquipoService** hacemos referencia a los atributos de **EquipoRepository** y **PartidoRepository** para poder utilizar los métodos de dichas clases.
En esta clase se pueden ver los diferentes métodos que hemos creado:

- **findEquipoById**(int id) al que le pasamos un id de equipo y hace referencia al método **findById**(id) del **EquipoRepository,** este método lo utilizamos para poder encontrar el equipo que le pasemos su id como parámetro.
- **findEquipoByNombre**(string nombre), hace referencia al método **findByNombre**(nombre) del **EquipoRepository,** este método lo utilizamos para dado un nombre encontrar el equipo.
- **findEquipo()** en el que utilizamos el método **findAll**() del **EquipoRepository,** método con el que obtenemos una lista de todos los equipos registrados**.**
- **saveEquipo**() en el que utilizamos el método sabe() del **EquipoRepository,** este método es usado para guardar un equipo en la base de datos.

- **deleteEquipo(Equipo equipo),** en este método primero se recorre la lista de todos los partidos que se tienen registrados, se comprueba si alguno de los equipos de cada partido coincide con el equipo que queremos eliminar y si es así se elimina dicho partido y posteriormente eliminamos el equipo.

```java
@Service
public class EquipoService {

    private EquipoRepository    equipoRepository;
    private PartidoService      partidoService;


    @Autowired
    public EquipoService(final EquipoRepository equipoRepository, final PartidoService partidoService) {
        this.partidoService = partidoService;
        this.equipoRepository = equipoRepository;
    }

    @Transactional(readOnly = true)
    public Equipo findEquipoById(final int id) throws DataAccessException {
        return this.equipoRepository.findById(id);
    }

    @Transactional(readOnly = true)
    public Collection<Equipo> findEquipoByNombre(final String nombre) throws DataAccessException {
        return this.equipoRepository.findByNombre(nombre);
    }
   @Transactional(readOnly = true)
   public Collection<Equipo> findEquipos() throws DataAccessException {
       return this.equipoRepository.findAll();
   }

   @Transactional
   public void saveEquipo(final Equipo equipo) throws DataAccessException {
       this.equipoRepository.save(equipo);
   }

   @Transactional
   public void deleteEquipo(final Equipo equipo) {
       Collection<Partido> partidos = this.partidoService.findAll();
       for (Partido partido : partidos) {
           if (partido.getEquipo1().getId() == equipo.getId() || partido.getEquipo2().getId() == equipo.getId()) {
               this.partidoService.deletePartido(partido);
           }
       }
       this.equipoRepository.delete(equipo);
   }
```

## Controlador

El controlador EquipoController esta en la ruta:
**\src\main\java\org\springframework\samples\petclinic\web\EquipoController.java**
Declaramos en primer lugar la vista con la que se crean y editan los equipos para que asi sea más fácil referenciarlo.

Luego como necesitaremos métodos tanto de **EquipoService** y como de **CompeticionService** los inicializamos también.
El método **setAllowedFields** lo utilizamos para poder tratar con el parametro “id”

Con el método **initCreationForm** accedemos al formulario para la creación del equipo.

```java
@Controller
public class EquipoController {

    private static final String VIEWS_EQUIPO_CREATE_OR_UPDATE_FORM  = "equipos/createOrUpdateEquiposForm";

    private final EquipoService equipoService;
    private final CompeticionService competicionService;

    @Autowired
    public EquipoController(final EquipoService equipoService,final CompeticionService competicionService, final UserService user
        this.equipoService = equipoService;
        this.competicionService= competicionService;
    }

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

    @GetMapping(value = "/competiciones/{competicionId}/equipos/new")
    public String initCreationForm(final Map<String, Object> model, @PathVariable("competicionId") final int competicionId) {
        Equipo equipo = new Equipo();
        equipo.setCompeticion(this.competicionService.findCompeticionById(competicionId));
        model.put("equipo", equipo);
        return EquipoController.VIEWS_EQUIPO_CREATE_OR_UPDATE_FORM;
    }
```

Con el método **processCreationForm** procedemos a guardar y enviar los datos del formulario de creación de un equipo.
Mediante el método **initFindForm** accedemos a la vista donde se encuentra el formulario para la búsqueda de un equipo.

```java
    @PostMapping(value = "/competiciones/{competicionId}/equipos/new")
    public String processCreationForm(@Valid final Equipo equipo, final BindingResult result,final ModelMap model ,@PathVariable('
        if (result.hasErrors()) {
            model.put("equipo", equipo);
            return EquipoController.VIEWS_EQUIPO_CREATE_OR_UPDATE_FORM;
        } else {
            Competicion e = this.competicionService.findCompeticionById(competicionId);
                e.addEquipo(equipo);
                this.equipoService.saveEquipo(equipo);
                this.competicionService.saveCompeticion(e);

            return "redirect:/competiciones/{competicionId}";
        }
    }


    @GetMapping(value = "/equipos/find")
    public String initFindForm(final Map<String, Object> model) {
        model.put("equipo", new Equipo());
        return "equipos/findEquipos";
    }
```

Con el método **processFindForm** si no introducimos ningún valor el formulario de búsqueda nos devolverá una lista con todos los equipos que hay en la base datos, por el contrario, si introducimos el nombre de un equipo existente en la base de datos nos mostrará solo ese equipo e introducimos un nombre que no este en la base de datos no mostrará un mensaje de error.

Con el método **initUpdateEquipoForm** accedemos a la vista donde se encuentra el formulario para la edición de los parametros de un equipo.

```java
@GetMapping(value = "/equipos")
public String processFindForm(Equipo equipo, final BindingResult result, final Map<String, Object> model) {
    // allow parameterless GET request for /Equipos to return all records
    if (equipo.getNombre() == null) {
        equipo.setNombre(""); // empty string signifies broadest possible search
    }
    // find Equipos by nombre
    Collection<Equipo> results = this.equipoService.findEquipoByNombre(equipo.getNombre());
    if (results.isEmpty()) {
        // no Equipos found
        result.rejectValue("nombre", "notFound", "not found");
        return "equipos/findEquipos";
    } else if (results.size() == 1) {
        // 1 Equipo found
        equipo = results.iterator().next();
        return "redirect:/equipos/" + equipo.getId();
    } else {
        // multiple Equipos found
        model.put("selections", results);
        return "equipos/equiposList";
    }
}
@GetMapping(value = "/competiciones/{competicionId}/equipos/{equipoId}/edit")
public String initUpdateEquipoForm(@PathVariable("equipoId") final int equipoId,@PathVariable("competicionId") final int comp
    Equipo Equipo = this.equipoService.findEquipoById(equipoId);
    model.addAttribute(Equipo);
    return EquipoController.VIEWS_EQUIPO_CREATE_OR_UPDATE_FORM;
}
```

Con el método **processUpdateForm** procedemos a guardar y enviar los datos del formulario de la edición de los parámetros del equipo.

El método **showEquipo** nos va a permitir mediante la clase **ModelAndView** mostramos los detalles de un equipo haciendo una llamada a la vista **equipoDetails.jsp**

```java
@PostMapping(value = "/competiciones/{competicionId}/equipos/{equipoId}/edit")
public String processUpdateEquipoForm(@Valid final Equipo equipo, final BindingResult result, @PathVariable("equipoId") final
    if (result.hasErrors()) {
        return EquipoController.VIEWS_EQUIPO_CREATE_OR_UPDATE_FORM;
    } else {
        Competicion e = this.competicionService.findCompeticionById(competicionId);
        equipo.setId(equipoId);
        equipo.setCompeticion(e);
        this.equipoService.saveEquipo(equipo);

        return "redirect:/equipos/{equipoId}";
    }
}

@GetMapping("/equipos/{equipoId}")
public ModelAndView showEquipo(@PathVariable("equipoId") final int equipoId) {
    ModelAndView mav = new ModelAndView("equipos/equipoDetails");
    mav.addObject(this.equipoService.findEquipoById(equipoId));
    return mav;
}
```

El método **showEquipoList** nos va a permitir mostrar la vista con todos los equipos registrados (**equiposList.jsp**), ya que guardamos en una variable todos los equipos sacados mediante el método **findEquipos** de **EquipoService.**

El método **processDeleteEquipo** primero identifica el equipo y la competición en la que está asignado, luego elimina el equipo de la competición y finalmente eliminamos el equipo de la base de datos.

```java
    @GetMapping("/equipos/list")
    public String showEquipoList(final Map<String, Object> model) {
        Equipos equipos = new Equipos();
        equipos.getEquipoList().addAll(this.equipoService.findEquipos());
        model.put("equipos", equipos);
        return "equipos/equiposList";
    }

    @GetMapping(value = "/competiciones/{competicionId}/equipos/{equipoId}/delete")
    public String processDeleteEquipo(@PathVariable("equipoId") final int equipoId, @PathVariable("competicionId") final int compe

        Equipo equipo = this.equipoService.findEquipoById(equipoId);
        Competicion competicion= this.competicionService.findCompeticionById(competicionId);
        competicion.removeEquipo(equipo);
        this.equipoService.deleteEquipo(equipo);
        return "redirect:/competiciones/{competicionId}";
    }

}
```

## Repositorio

El repositorio EquipoRepository está la ruta:
\src\main\java\org\springframework\samples\petclinic\repository\EquipoRepository.java

Esta interfaz cuenta con los métodos de **save(Equipo equipo)** con el que guardamos los equipos en la base de datos.
Tambien, el método **findAll**() que nos devolvería una Collection con todos los equipos que hay en la base de datos.
Mediante Querys obtenemos los métodos **findById**(@Param(“id”) int id) y **findByNombre**(@Param(“nombre”) int nombre), por el cual en el primer caso obtenemos el equipo que queremos pasándole por parámetro el id y en el segundo pasándole por parámetros el nombre del equipo que queremos obtener.
Y por último, el método **delete**(Equipo equipo) para eliminar el equipo que pasamos por parámetro.

```java
public interface EquipoRepository extends Repository<Equipo, Integer> {

    void save(Equipo equipo) throws DataAccessException;

    Collection<Equipo> findAll() throws DataAccessException;

    @Query("SELECT DISTINCT equipo FROM Equipo equipo WHERE equipo.nombre LIKE :nombre%")
    Collection<Equipo> findByNombre(@Param("nombre") String nombre);

    @Query("SELECT equipo FROM Equipo equipo WHERE equipo.id =:id")
    Equipo findById(@Param("id") int id) throws DataAccessException;

    void delete(Equipo equipo);

}
```

# Ejemplos de pruebas implementadas

## Pruebas unitarias (máximo de dos ejemplos)

Las pruebas unitarias del JugadorService están la ruta:
**\src\test\java\org\springframework\samples\petclinic\service\JugadorServiceTests.java**
Pruebas unitarias JugadorService.
En **shouldFindJugadorWithCorrectId**() nos aseguramos que el jugador con id 1, su nombre Daniel y que pertenezca al Betis.

```
package org.springframework.samples.petclinic.service;
import java.time.LocalDate;

@DataJpaTest(includeFilters = @ComponentScan.Filter(Service.class))
class JugadorServiceTests {

    @Autowired
    protected JugadorService    jugadorService;

    @Autowired
    protected EquipoService     equipoService;

    @Test
    void shouldFindJugadorWithCorrectId() {
        Jugador jugador1 = this.jugadorService.findJugadorById(1);
        assertThat(jugador1.getNombre()).startsWith("Daniel");
        assertThat(jugador1.getEquipo().getNombre()).isEqualTo("Betis");

    }
```

En **shouldInsertJugadorIntoDatabaseAndGenerateId**() creamos un nuevo jugador con todos sus atributos y nos aseguramos que la el tamaño de la lista de jugadores se ha incrementa en una unidad. Luego guardamos el jugador en un equipo y nos aseguramos de que el tamaño de la lista de jugadores de ese equipo también se ha visto incrementado en una unidad y que el id del jugador no es nulo.

```
    @Test
    @Transactional
    public void shouldInsertJugadorIntoDatabaseAndGenerateId() {
        Equipo equipo1 = this.equipoService.findEquipoById(1);
        int found = equipo1.getJugadores().size();

        Jugador jugador = new Jugador();
        jugador.setNombre("Paco");
        jugador.setApellidos("Perez");
        jugador.setDni("29517543X");
        jugador.setFechaNacimiento(LocalDate.now().minusYears(12));
        jugador.setNacionalidad("España");
        jugador.setLesion(false);
        jugador.setTarjetaAmarilla(0);
        jugador.setTarjetaRoja(0);
        equipo1.addJugador(jugador);
        assertThat(equipo1.getJugadores().size()).isEqualTo(found + 1);

        this.equipoService.saveEquipo(equipo1);

        equipo1 = this.equipoService.findEquipoById(1);
        assertThat(equipo1.getJugadores().size()).isEqualTo(found + 1);
        // checks that id has been generated
        assertThat(jugador.getId()).isNotNull();
    }
```

## Pruebas de controlador

Las pruebas de controlador de Competición están el ruta:
**\src\test\java\org\springframework\samples\petclinic\controller\CompeticionControllerTests.java**
En **testShowListCompeticiones()** prueba mediante Mock que se visualiza la lista de competiciones.

En **testDeleteCompeticiones()** prueba mediante Mock que se elimina una competición.

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

@WithMockUser(value = "admin1")
@Test
void testDeleteCompeticion() throws Exception {
    this.mockMvc
            .perform(MockMvcRequestBuilders.get("/competiciones/{competicionId}/delete",
                    CompeticionControllerTests.TEST_COMPETICION_ID))
            .andExpect(MockMvcResultMatchers.model().attributeDoesNotExist("competicion"))
            .andExpect(MockMvcResultMatchers.view().name("redirect:/competiciones/list"));
}
```

## Principales problemas encontrados

*En término general el equipo de trabajo creo que ha estado trabajando muy bien durante todos los sprint, aunque tengo que destacar que a partir del Sprint 3 la participación e implicación en el trabajo por parte de Stefan disminuyo notablemente ya que hubo momentos en los que tuvimos que ayudar a Marcos(su pareja de trabajo) los demás integrantes ya que era demasiado para una sola persona, ya que Stefan no hizo las tareas que se habían asignado, ni nos comunicó con tiempo que no iba a poder realizarlas. Por lo que nos repartimos algunas de sus tareas, para que Marcos no tuviese una carga tan alta de trabajo.*

## Otros comentarios

*Aparte de lo comentado anteriormente también he estado implicado en las revisiones antes de las entregas de los sprints. También he sido el encargado de ir creando las tareas en el clockify y de sacar los "reports" de las horas trabajadas por los miembros del grupo.*
*En cuanto a la documentación:*

- *Descripción general del sistema, de los perfiles de usuario, de las historias de usuario y de las reglas de negocio (junto a mi compañero Ignacio).*
- *Diagrama de capas (junto a mi compañero Ignacio).*