Nombre: Gonzalo Fernández Jiménez 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: gonferjim
Rama del usuario en Github: gonferjim

Participación en el proyecto

Historias de usuario en las que he participado

He desarrollado junto a mi compañero de equipo Javier Grosso las siguientes historias de usuario:

-H4: Publicar Noticias.
-H5: Asignar noticias a partidos
-H9: Asignar y eliminar jugadores a un partido
-H10: Visualización de Noticias
-H16: Editar perfil

Además de haber participado en las siguientes reglas de negocio junto a mi compañero Javier Grosso:

-RN3: máximo número de tarjetas amarillas/rojas para participar
-RN4: no poder arbitrar dos partidos a la misma hora

Funcionalidad implementada

Mi compañero Javier Grosso y yo hemos creado la entidad Noticia y la entidad Equipo junto a sus repositorios.

También hemos implementado el controlador NoticiaController.

Además los métodos de los servicios NoticiaService, EquipoService, y los métodos necesarios para la tarea H9 en PartidoService.

Mi compañero Javier y yo hemos implementado el validador PartidoValidator, necesario para la RN3 y la RN4.

También hemos realizado las vistas de las entidades Noticia y Partido, situadas en la carpeta jsp.

Esto hace un total de 6 clases implementadas y 2 interfaces. Sin tener en cuenta las implementaciones necesarias para el H9 en la clase Partido service.

Nombre: Gonzalo Fernández Jiménez Grupo: G3-08

Pruebas implementadas

Pruebas unitarias

He realizado de manera individual los test unitarios de PartidoService.

En equipo he realizado junto a Javier Grosso 2 tests unitarios para Servicios (UsuarioService, NoticiaService)

Eso hace un total de 3 clases de test unitarios con un total de 16 métodos anotados con @Test.

Pruebas de Controlador

He creado 2 casos de prueba positivos y 1 negativo de controlador para la HU-4.
He creado 2 casos de prueba positivos de controlador para la HU-10.
He creado 2 casos de prueba positivos y 1 negativo de controlador para la HU-9.
He creado 5 casos de prueba positivos y 2 negativo de controlador para la HU-16.

Ejemplos de funcionalidades implementadas

Entidades

Entidad Noticia, está en la ruta src/main/java/org.springframework.samples.petclinic/model
Realizada en colaboración con Javier Grosso.

Consiste en una entidad que implementa en su código una relación many to many con la entidad partidos (debido a la historia de usuario H5 donde debemos de poder relacionar las noticias con los partidos), y posee un parámetro para el título de la noticia (no puede estar vacío), otro parámetro para el desarrollo de la noticia (tampoco puede estar vacío), y un parámetro para la fecha de publicación de la noticia (que solo puede ser fechas pasadas o el presente y está formateada de la forma año/mes/dia).

Noticia.java

```java

package org.springframework.samples.petclinic.model;

import java.time.LocalDate;

@Getter
@Setter
@Entity
@Table(name = "noticias")
public class Noticia extends BaseEntity {

    @NotEmpty
    private String      titulo;

    @NotEmpty
    private String      texto;

    @PastOrPresent
    @DateTimeFormat(pattern = "yyyy/MM/dd")

    private LocalDate   fecha;

    @ManyToMany
    @JoinTable(name = "partidos_noticias", joinColumns = @JoinColumn(name = "noticia_id"),
    inverseJoinColumns = @JoinColumn(name = "partido_id"))
    private List<Partido> partidos;

}

```

Entidad User, está en la ruta src/main/java/org.springframework.samples.petclinic/model
Realizada en colaboración con Javier Grosso

Consiste en una entidad recogida la entidad User nativa, debido a las numerosas implementación y configuración del framework en base a esta entidad ha sido más oportuno recoger esta entidad que crear una nueva y atender a los conflictos. Esta es la entidad base a la hora de implementar los rolas y las autorizaciones.

Consiste en un nombre de usuario y una contraseña que se utiliza para iniciar sesión.
Siendo el nombre de usuario el atributo Identificativo de esta entidad.
Otros atributos con una funcionalidad más documentativa.

Posee el atributo genre que es una clase hija de NamedEntity la cual se utiliza para elaborar enumerados, siendo genre un enumerado tal que Genre: {Hombre, Mujer}.
También posee una lista de authorities las cuales son Entidades donde se guarda la información de los roles de ese usuario.

```java
/*
 * Copyright Futvilla Team
 */

package org.springframework.samples.petclinic.model;

import java.time.LocalDate;

@Getter
@Setter
@Entity
@Table(name = "users")
public class User {

    @Id
    @NotEmpty
    private String username;

    @NotEmpty
    private String password;

    private boolean enabled;

    @Column(name = "first_name")
    @NotEmpty
    private String firstName;

    @Column(name = "last_name")
    @NotEmpty
    private String lastName;

    @NotEmpty
    @Email
    private String  email;

    @Column(name = "birth_date")
    @DateTimeFormat(pattern = "yyyy/MM/dd")
    private LocalDate   birthDate;

    @NotEmpty
    @Digits(fraction = 0, integer = 10)
    private String  telephone;

    @ManyToOne
    @JoinColumn(name = "genre_id")
    private Genre   genre;

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
    private Set<Authorities> authorities;

    public boolean isNew() {
        return this.username == null;
    }
}
```

Servicio

Servicio NoticiaService, implementado en colaboración con Javier Grosso y localizado en la ruta src/main/java/org.springframework.samples.petclinic/service

El NoticiaService se encarga de agrupar todas las funcionalidades que se prestan como servicio, siendo quien invoca los métodos del Repository.
En este caso, podemos ver que implementa las invocaciones a los métodos del repositorio necesarios para obtener una colección con todas las noticias, para obtener una noticia según su id en la base de datos, para guardar una noticia en la base de datos y para borrar una noticia en la base de datos.

Este servicio como el resto de servicios tiene un *log* que informa por consola de las acciones transmitidas al repositorio.

```
30 @Service
31 public class NoticiaService {
32
33     private NoticiaRepository noticiaRepository;
34
35     @Autowired
36     public NoticiaService(NoticiaRepository noticiaRepository) {
37         this.noticiaRepository = noticiaRepository;
38     }
39
40     @Transactional(readOnly = true)
41     public Collection<Noticia> findAll() throws DataAccessException {
42         log.info("Se han recogido todas las Noticias");
43         return noticiaRepository.findAll();
44     }
45
46     @Transactional(readOnly = true)
47     public Noticia findById(int id) throws DataAccessException {
48         log.info("Se han recogido una Noticia por id");
49         return noticiaRepository.findById(id);
50     }
51
52     @Transactional
53     public void saveNoticia(Noticia noticia) throws DataAccessException {
54         log.info("Se han guardado una Noticia");
55         noticiaRepository.save(noticia);
56     }
57     @Transactional
58     public void deleteNoticia(Noticia noticia) throws DataAccessException {
59         log.info("Se han eliminado una Noticia");
60         noticiaRepository.delete(noticia);
61     }
```

Servicio UserService, implementado en colaboración con Javier Grosso y localizado en la ruta src/main/java/org.springframework.samples.petclinic/service

El UserService se encarga de agrupar todas las funcionalidades que se prestan como servicio, siendo quien invoca los métodos del Repository.
En este caso, podemos ver que implementa las invocaciones a los métodos del repositorio necesarios para obtener una colección de todos los géneros existentes en la base de datos, una colección con todas los users, para obtener un user según su username en la base de datos, para guardar un user en la base de datos y para borrar un user en la base de datos.

A la hora de guardar un user debemos asegurarnos de que esté activo (setEnable) para evitar que alguien que se acaba de registrar sea incapaz de utilizar su usuario.

Este servicio como el resto de servicios tiene un *log* que informa por consola de las acciones transmitidas al repositorio.

```java
/*
 * Copyright Futvilla Team
 */
package org.springframework.samples.petclinic.service;

import java.util.Collection;

@Slf4j
@Service
public class UserService {

    private UserRepository userRepository;

    @Autowired
    public UserService(UserRepository usuarioRegistradoRepository) {
        userRepository = usuarioRegistradoRepository;
    }

    @Transactional(readOnly = true)
    public Collection<Genre> findGenres() throws DataAccessException {
        return userRepository.findGenres();
    }


    @Transactional(readOnly = true)
    public Collection<User> findAll() throws DataAccessException {
        log.info("Se han recogido todos los Users");
        return userRepository.findAll();
    }

    @Transactional(readOnly = true)
    public User findUserByUsername(String username) {
        log.info("Se han recogido un User por username");
        return userRepository.findByUsername(username);
    }

    @Transactional(readOnly = true)
    public User findUserByUsername(String username) {
        log.info("Se han recogido un User por username");
        return userRepository.findByUsername(username);
    }

    @Transactional
    public void saveUser(User user) throws DataAccessException {
        log.info("Se han guardado un User");
        user.setEnabled(true);
        userRepository.save(user);
    }

    @Transactional
    public void delete(User user) throws DataAccessException {
        log.info("Se han eliminado un User");
        userRepository.delete(user);
    }

}
```

Controlador

Controlador NoticiaController, localizado en la ruta src/main/java/org.springframework.samples.petclinic/web.
Realizado en colaboración con Javier Grosso.
NoticiaController se encarga de responder a los eventos que se producen en la vista de Noticia, 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 una nueva noticia (initCreationForm y processCreationForm) y de como invoca cambios en el

modelo (añadiendo la noticia al repositorio por medio del servicio que ofrece noticiaService).

Cosas a tener en cuenta:

- A la hora de crear una noticia el usuario no tiene acceso a la fecha, si no que la fecha se establece como la fecha de creación de la noticia.

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

import java.time.LocalDate;

@Slf4j
@Controller
public class NoticiaController {

    private static final String     VIEWS_NOTICIA_CREATE_OR_UPDATE_FORM = "noticias/createOrUpdateNoticiaForm";
    private static final String     VIEWS_NOTICIA_SHOW                  = "noticias/noticiaDetails";

    private final NoticiaService    noticiaService;

    private final PartidoService    partidoService;


    @Autowired
    public NoticiaController(NoticiaService noticiaService,PartidoService partidoService) {
        this.noticiaService = noticiaService;
        this.partidoService = partidoService;
    }

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

    @ModelAttribute("partidos")
    public Collection<Partido> populatePartidos() {
        return partidoService.findAll();
    }
```

```java
    @GetMapping(value = "/noticias/new")
    public String initCreationForm(final Map<String, Object> model) {
        log.info("Se ha iniciado la creación de una noticia");
        Noticia noticia = new Noticia();
        model.put("noticia", noticia);
        return NoticiaController.VIEWS_NOTICIA_CREATE_OR_UPDATE_FORM;
    }

    @PostMapping(value = "/noticias/new")
    public String processCreationForm(@Valid final Noticia noticia, final BindingResult result) {
        if (result.hasErrors()) {
            log.warn("Se ha abortado la creación de una noticia");
            return NoticiaController.VIEWS_NOTICIA_CREATE_OR_UPDATE_FORM;
        } else {
            log.info("Se ha finalizado la creación de una noticia");
            LocalDate fechaActual = LocalDate.now();
            noticia.setFecha(fechaActual);
            noticiaService.saveNoticia(noticia);

            return "redirect:/noticias/" + noticia.getId();
        }
    }

    @GetMapping("/noticias/{id}")
    public ModelAndView showNoticia(@PathVariable("id") final int id) {
        log.info("Se ha iniciado la muestra de detalles de una noticia");
        ModelAndView mav = new ModelAndView(NoticiaController.VIEWS_NOTICIA_SHOW);
        mav.addObject(noticiaService.findById(id));
        return mav;
    }

}
```

```
100
101     @GetMapping("/noticias/list")
102     public String showNoticiaList(final Map<String, Object> model) {
103         log.info("Se ha iniciado la vista de mostrar la lista de noticias");
104         Collection<Noticia> noticias = noticiaService.findAll();
105         model.put("noticias", noticias);
106         return "noticias/noticiasList";
107     }
108
109     @GetMapping(value = "/noticias/{id}/delete")
110     public String processDeleteForm(@PathVariable("id") final int Id) {
111         log.info("Se ha iniciado la eliminación de una noticia");
112         Noticia noticia = noticiaService.findById(Id);
113         noticiaService.deleteNoticia(noticia);;
114         return "redirect:/noticias/list";
115     }
116
117 }
118
```

Controlador UserController, localizado en la ruta src/main/java/org.springframework.samples.petclinic/web. Realizado en colaboración con Javier Grosso.
UserController se encarga de responder a los eventos que se producen en la vista de User, 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 user (initCreationForm y processCreationForm) y de como invoca cambios en el modelo (añadiendo el user al repositorio por medio del servicio que ofrece userService).

Cosas a tener en cuenta:

- A la hora de guardar el usuario también es necesario guardar las autoridades para que los permisos de rol “user” se apliquen y puedan acceder a todo lo que su usuario lo permita.
- También es necesario actualizar el SecurityContext para que si un usuario edita su propio usuario los cambios realizados también se guarden en la sesión de la aplicación.

```
18
19 import java.util.Collection;
45
46 @Slf4j
47 @Controller
48 public class UserController {
49
50     private static final String            VIEWS_USER_CREATE_OR_UPDATE_FORM    = "users/createOrUpdateUserForm";
51     private static final String            VIEWS_USER_SHOW                     = "users/userDetails";
52
53     private final UserService   userService;
54     private final AuthoritiesService authoritiesService;
55
56
57     @Autowired
58     public UserController(UserService userService, AuthoritiesService authoritiesService) {
59         this.userService = userService;
60         this.authoritiesService = authoritiesService;
61     }
62
63     @InitBinder
64     public void setAllowedFields(WebDataBinder dataBinder) {
65         //dataBinder.setDisallowedFields("id");
66     }
67
68     @ModelAttribute("genres")
69     public Collection<Genre> populateGenres() {
70         return userService.findGenres();
71     }
```

```java
@GetMapping(value = "/users/new")
public String initCreationForm(Map<String, Object> model) {
    log.info("Se ha iniciado la creación de un user");
    User user = new User();
    model.put("user", user);
    return VIEWS_USER_CREATE_OR_UPDATE_FORM;
}

@PostMapping(value = "/users/new")
public String processCreationForm(@Valid User user, BindingResult result) {
    if (result.hasErrors()) {
        log.warn("Se ha abortado la creación de un user");
        return VIEWS_USER_CREATE_OR_UPDATE_FORM;
    } else {
        log.info("Se ha finalizado la creación de un user");
        userService.saveUser(user);
        authoritiesService.saveAuthorities(user.getUsername(), "user");
        return "redirect:/users/" + user.getUsername();
    }
}

@GetMapping(value = "/users/{username}/edit")
public String initUpdateUserForm(@PathVariable("username") String username, Model model) {
    log.info("Se ha iniciado la edición de un user");
    User user = userService.findUserByUsername(username);
    model.addAttribute(user);
    return VIEWS_USER_CREATE_OR_UPDATE_FORM;
}
```

```java
@PostMapping(value = "/users/{username}/edit")
public String processUpdateUserForm(@Valid User user, BindingResult result, @PathVariable("username") String user
    if (result.hasErrors()) {
        log.warn("Se ha abortado la edición de un user");
        return VIEWS_USER_CREATE_OR_UPDATE_FORM;
    } else {
        log.info("Se ha finalizado la edición de un user");
        if(user.getUsername().equals(username))
            userService.saveUser(user);
        else {
            userService.saveUser(user);
            authoritiesService.saveAuthorities(user.getUsername(), "user");
            // Eliminamos el Usuario y su autoridad con el username abandonado
            Authorities auto = authoritiesService.findAllAuthorities().stream().
                filter(x->x.getUser().getUsername().equals(username)).findFirst().get();
            authoritiesService.deleteAuthorities(auto);
            userService.delete(userService.findUserByUsername(username));
            //Actualizamos el Authentication
            Authentication auth = SecurityContextHolder.getContext().getAuthentication();

            org.springframework.security.core.userdetails.User updatedUser = new org.springframework.security.cor
            Authentication newAuth = new UsernamePasswordAuthenticationToken(updatedUser, auth.getCredentials(),

            SecurityContextHolder.getContext().setAuthentication(newAuth);
        }

        return "redirect:/users/" + user.getUsername();
    }
}
```

Repositorio

NoticiaRepository, localizado en la ruta src/main/java/org.springframework.samples.petclinic/repository.
Realizado en colaboración con Javier Grosso.

NoticiaRepository es una interfaz que contiene la lógica requerida para acceder a los datos del repositorio de Noticia en la base de datos, haciendo las consultas a esta de forma transparente.
Tiene implementada la lógica necesaria para poder obtener todas las noticias, para poder buscar en la base de datos noticias según el Id y para poder guardar y borrar una noticia.

```java
import java.util.Collection;

public interface NoticiaRepository extends Repository<Noticia, String> {

    void save(Noticia noticia) throws DataAccessException;

    Collection<Noticia> findAll() throws DataAccessException;

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

    void delete(Noticia noticia) throws DataAccessException;

}
```

UserRepository, localizado en la ruta
src/main/java/org.springframework.samples.petclinic/repository.
Realizado en colaboración con Javier Grosso.

UserRepository es una interfaz que contiene la lógica requerida para acceder a los datos del repositorio de User en la base de datos, haciendo las consultas a esta de forma transparente.
Tiene implementada la lógica necesaria para poder obtener todos los users, para poder buscar en la base de datos users según el username, para poder obtener todos los tipos de géneros (genre) y para poder guardar y borrar un user.
Esta es la única entidad que tiene un identificador de tipo String y por tanto el tipo de repositorio del que extiende deber ser correspondiente a un tipo String.

```java
import java.util.Collection;

public interface UserRepository extends Repository<User, String> {

    void save(User user) throws DataAccessException;

    void delete(User user) throws DataAccessException;

    @Query("SELECT genre FROM Genre genre ORDER BY genre.name")
    List<Genre> findGenres() throws DataAccessException;

    Collection<User> findAll() throws DataAccessException;

    @Query("SELECT user FROM User user WHERE user.username = ?1")
    User findByUsername(@Param("username") String username) throws DataAccessException;


}
```

Formatter

El SexoFormatter es un formater de la clase Genre para referirse y visualizarse por su nombre en vez de por su número de identificación.
Realizado en colaboración con Javier Grosso.

```java
@Component
public class SexoFormatter implements Formatter<Genre> {

    private final UserService userService;

    @Autowired
    public SexoFormatter(final UserService userService) {
        this.userService = userService;
    }

    @Override
    public String print(final Genre sexo, final Locale locale) {
        return sexo.getName();
    }

    @Override
    public Genre parse(final String text, final Locale locale) throws ParseException {
        Collection<Genre> findSexos = this.userService.findGenres();
        for (Genre sexo : findSexos) {
            if (sexo.getName().equals(text)) {
                return sexo;
            }
        }
        throw new ParseException("sexo not found: " + text, 0);
    }

}
```

El PartidoFormatter es un formatter de la clase Partido para referirse y visualizarse por su lugar y fecha en vez de por su número de identificación.
Realizado en colaboración con Javier Grosso.

```java
@Component
public class PartidoFormatter implements Formatter<Partido> {


    private final PartidoService partidoService;

    @Autowired
    public PartidoFormatter(final PartidoService partidoService) {
        this.partidoService = partidoService;
    }

    @Override
    public String print(final Partido partido, final Locale locale) {
        return partido.getLugar()+" "+partido.getFecha().toString();

    }

    @Override
    public Partido parse(final String text, final Locale locale) throws ParseException {
        Collection<Partido> findPartidos = partidoService.findAll();
        for (Partido partido : findPartidos)
            if (print(partido,locale).equals(text))
                return partido;
        throw new ParseException("partido not found: " + text, 0);
    }

}
```

Validador

El PartidoValidator ha sido realizado en colaboración con Javier Grosso y otros compañeros que han participado en otras reglas de negocio.

Nombre: Gonzalo Fernández Jiménez Grupo: G3-08

La parte realizada por mi compañero Javier y yo han sido las reglas de negocio de las tarjetas amarillas/rojas y árbitros.

En esta clase realizamos validaciones más complejas que las que se permiten en anotaciones sobre los atributos de las entidades. Aún así, al hacer un @override del validate será necesario incluir el @notEmpty del lugar.

Se realiza un try catch para evitar nullPointerExceptions a la hora de hacer un partido.getJugadoresParticipantes(). En caso de que los jugadores del partido sean null que se establezcan como una lista vacía para evitar posteriores errores.

Luego se recorren todos los jugadores seleccionados en la vista y se comprueba si cumplen con las reglas de negocio de las tarjetas amarillas y rojas.
Luego se recorren todos los partidos y se comprueba que el árbitro seleccionado no está arbitrando otro partido el mismo día.

La otra validación correspondiente a la lesión corresponde a otros individuos del grupo.

```
3  import java.util.Collection;
14
15 @Slf4j
16 public class PartidoValidator implements Validator{
17
18
19     private PartidoService partidoService;
20
21     public PartidoValidator(PartidoService partidoService) {
22         this.partidoService=partidoService;
23     }
24
25     @Override
26     public boolean supports(Class<?> clazz) {
27         return Partido.class.isAssignableFrom(clazz);
28     }
29
30     @Override
31     public void validate(Object target, Errors errors) {
32         log.info("Se ha iniciado el validador de Partido");
33         Collection<Partido> restoDePartidos = partidoService.findAll();
34
35         // Este codigo realiza la regla de negocio 3, donde se comprueba que en un
36         // partido no pueda entrar jugadores sancionados
37         Partido p = (Partido) target;
38
39         if(p.getLugar().isEmpty()) {
40             errors.rejectValue("lugar",
41                 "no puede estar vacio",
42                 "no puede estar vacio");
```

```
44
45          Set<Jugador> jugadores = new HashSet<Jugador>();
46          try {
47              jugadores = p.getJugadoresParticipantes();
48              if (jugadores == null) {
49                  jugadores = new HashSet<Jugador>();
50              }
51          } catch (NullPointerException e) {
52
53          } finally {
54              log.info("Se van a validar las Tarjetas Rojas y Amarillas");
55              for (Jugador j : jugadores)
56                  if (j.getTarjetaAmarilla() >= 5) {
57                      log.warn("Ha saltado la excepción de tarjetas amarillas");
58                      errors.rejectValue("jugadoresParticipantes",
59                          "El jugador no puede participar si tiene 5 o mas tarjetas amarillas",
60                          "El jugador no puede participar si tiene 5 o mas tarjetas amarillas");
61                  } else if (j.getTarjetaRoja() > 0) {
62                      log.warn("Ha saltado la excepción de tarjetas rojas");
63                      errors.rejectValue("jugadoresParticipantes", "El jugador no puede participar si tiene ta
64                          "El jugador no puede participar si tiene tarjeta roja");
65                  }
66              // Este codigo realiza la regla de negocio 2, Un jugador que esté lesionado no
67              // podrá ser inscrito en un
68              // partido, hasta que no le hayan dado el alta de su lesión.
69              log.info("Se van a validar las lesiones de los jugadores");
70              for (Jugador jug : jugadores)
71                  if (jug.getLesion()) {
72                      errors.rejectValue("jugadoresParticipantes",
73                          "Un jugador que esté lesionado no podrá ser inscrito en un partido",
74                          "Un jugador que esté lesionado no podrá ser inscrito en un partido");
75                  }
```

```
76
77              // Regla de negocio 4
78              log.info("Se van a validar que hay un mismo arbitro arbitrando dos partidos al mismo tiempo");
79              for (Partido partido : restoDePartidos)
80                  if (!p.getId().equals(partido.getId()) && p.getArbitro().equals(p.getArbitro()) && partido.g
81                      errors.rejectValue("arbitro", "Un arbitro no puede arbitrar dos partidos en una misma fe
82                          "Un arbitro no puede arbitrar dos partidos en una misma fecha");
83                  }
84
85          }
86
87
88      }
89
90  }
```

Ejemplos de pruebas implementadas

Pruebas unitarias

Test 1: realizado a UserService dentro del archivo UsuarioServiceTests en la ruta src/test/java/org.springframework.samples.petclinic/service
Realizado en colaboración con Javier Grosso

Este test consiste en eliminar un usuario, primero se crea el usuario y se guarda, comprobado que funciona con un test anterior. Una vez guardado se comprueba cuántos usuarios hay guardados en la base de datos.
Por último se elimina el usuario y se comprueba si la cantidad de usuario en base de datos después de realizar la acción de eliminar es inferior por una unidad a la cantidad de usuarios antes de eliminar.

```
129
130    public void shouldDeleteUser() {
131
132
133        User user = new User();
134        user.setUsername("Sam");
135        user.setPassword("Sam");
136        user.setEnabled(true);
137        user.setFirstName("Sammy");
138        user.setLastName("Gammy");
139        user.setEmail("saga@correo.com");
140        user.setTelephone("954345723");
141        user.setBirthDate(LocalDate.now().minusYears(20));
142
143        this.userService.saveUser(user);
144        Collection<User> usuarios = this.userService.findAll();
145        int found = usuarios.size();
146        userService.delete(user);
147        Collection<User> usuarios2 = this.userService.findAll();
148        int found2 = usuarios2.size();
149        Assertions.assertThat(found2).isEqualTo(found - 1);
150    }
151
152 }
```

Test 2: realizado a NoticiaService dentro del archivo NoticiaServiceTests en la ruta src/test/java/org.springframework.samples.petclinic/service
Realizado en colaboración con Javier Grosso

Este test consiste en actualizar una noticia, primero se crea la noticia y se guarda, comprobado que funciona con un test anterior. Una vez guardado se comprueba cuántas noticias hay guardadas en la base de datos.
Por último se recoge la noticia anterior y se modifica, para luego volverse a guardar. Para luego comprobar si la cantidad de noticias en base de datos después de realizar la acción de guardar no ha cambiado, significando que la noticia se ha actualizado y no se ha creado otra noticia similar a la anterior y también se comprueba que la noticia ya no tiene el título inicial.

```
87     @Test
88     @Transactional
89     void shouldUpdateNoticias() {
90         Noticia not = new Noticia();
91         not.setFecha(LocalDate.now());
92         not.setTexto("Hola soy un ejemplo");
93         not.setTitulo("Primicia: el ejemplo");
94
95         this.noticiaService.saveNoticia(not);
96         Collection<Noticia> usuarios = this.noticiaService.findAll();
97         int found = usuarios.size();
98
99         Noticia not2 = this.noticiaService.findById(2);
100        not2.setTitulo("Ya no es primicia");
101
102        this.noticiaService.saveNoticia(not2);
103
104        // retrieving new name from database
105        Collection<Noticia> noticias2 = this.noticiaService.findAll();
106        int found2 = noticias2.size();
107        Assertions.assertThat(found2).isEqualTo(found);
108        Noticia not3 = this.noticiaService.findById(2);
109        Assertions.assertThat(not3.getTitulo()).isEqualTo("Ya no es primicia");
110    }
111
112 }
113
```

Pruebas de controlador

Test 1: realizado a UserController dentro del archivo UserControllerTests en la ruta src/test/java/org.springframework.samples.petclinic/controller

Este test consiste en crear un usuario. Se introduce todo correctamente sin dejar ningún valor vacío o erroneo. Por lo que se espera una redirección a los detalles del usuario creado consecuencia de una correcta creación de un usuario.

```
56    @WithMockUser(value = "admin1")
57    @Test
58    void testProcessNewUserFormSuccess() throws Exception {
59        mockMvc.perform(post("/users/new").with(csrf())
60            .param("firstName", "Juan").param("lastName", "Garcia").param("email", "juan@gmail.com")
61            .param("birthDate", "2020/07/22").param("genre", "hombre").param("telephone", "666666666")
62            .param("username", "asd").param("password", "asd")).andExpect(status().is3xxRedirection())
63        .andExpect(view().name("redirect:/users/asd"));
64    }
65
```

Test 2: realizado a UserController dentro del archivo UserControllerTests en la ruta src/test/java/org.springframework.samples.petclinic/controller

Este test consiste en crear un usuario igual que el test anterior pero esta vez esperando un error surgido como consecuencia de que el nombre de usuario (username) esté vacío. Por lo que se espera que el atributo “user” del modelo tenga un error y que se vuelva a la vista de crear usuario para que se vuelva a intentar introducir los valores correctamente.

```
66    @WithMockUser(value = "admin1")
67    @Test
68    void testProcessNewUserFormHasErrors() throws Exception {
69        mockMvc.perform(post("/users/new").with(csrf())
70            .param("firstName", "hola").param("lastName", "Romano").param("email", "manuel@gmail.com")
71            .param("birthDate", "2020/01/01").param("genre", "genre").param("telephone", "666666666")
72            .param("username", "")).andExpect(model().attributeHasErrors("user"))
73        .andExpect(status().isOk()).andExpect(view().name("users/createOrUpdateUserForm"));
74    }
75
```

Principales problemas encontrados

Ninguno.

Otros comentarios

En el proyecto, a parte de lo anteriormente mencionado, también he participado en:

-Hacer los diferentes Mockups (diagrama de dominio, y Modelo de Datos) de en equipo con mi compañero Javier Grosso.
-Participación en los documentos junto al resto de los compañeros (decidir entidades, escribir y detallar historias de usuario, los documentos de diseño del sistema, ect etc)
-Implementar la opción de registrarse como entrenador al usuario, en equipo con Gonzalo Fernandez.
-Correcciones en el código de funciones implementadas por otros compañeros, o correcciones ortográficas de lo que se muestra al usuario en la capa de aplicación.