

Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: <https://github.com/gii-is-DP1/dp1-2020-g2-07>

Nombre de usuario en GitHub: FranciscoJRE14

Rama del usuario en Github: franX

Participación en el proyecto

Historias de usuario en las que he participado

He implementado completas las historias de usuario H15-Crear Circuito, H16-Eliminar Circuito, H17-Editar Circuito, H18-Mostrar Circuitos

He desarrollado junto a mi compañero las historias de usuario H19-Añadir un horario a un empleado, H20- Añadir sesiones a un empleado en un determinado horario, H21-Ver Horarios trabajador.

Funcionalidad implementada

Entidades creadas en solitario: Horario.java, Circuito.java.

Repositorios creados en solitario: CircuitoRepository.java.

Controladores creados en solitario: CircuitoController.java.

Servicios creados en solitario: CircuitoService.java.

He añadido métodos a los siguientes servicios y controladores: SalaService.java, SalaController.java, HorarioController.java, HorarioRepository.java.

Vistas

creadas: circuitosListing, createOrUpdateCircuitosForm, horariosForm, employeeSchedule.

He añadido elementos a la vista employeeDetails.

Esto hace un total de 5 clases implementadas por mi y 4 interfaces definidas.

Pruebas implementadas

Pruebas unitarias

Test de los servicios: SalaService(RoomMockTest.java), CircuitoService(CircuitMockTest.java)

Test de los controladores: CircuitoController(CircuitoControllerTest.java)

Test de los formatters: SalaFormatter(SalaFormatterTest.java)

Pruebas de Controlador

Controlador CircuitoController:

- Historia de Usuario H15-Crear Circuito
1 caso positivo, 1 caso negativo
- Historia de Usuario H16-Editar Circuito
1 caso positivo, 1 caso negativo

- Historia de Usuario H17-Eliminar Circuito
1 caso positivo, 1 caso negativo
- Historia de Usuario H17-Mostrar Circuito
1 caso positivo

Pruebas de Service

- Service SalaService (4 casos positivos, 1 negativo)
- Service CircuitoService(5 casos positivos, 1 negativo)

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

Circuitos(/dp1-2020-g2-

07/src/main/java/org/springframework/samples/petclinic/model/Circuito.java)

Esta entidad es relevante, ya que es un objeto del modelo de dominio en donde se guarda la informacion de las salas que componen un circuito y sus características.

```

1 package org.springframework.samples.petclinic.model;
2 import java.util.List;
11
12 @Entity
13 @Data
14 @Table(name="circuitos")
15 public class Circuito extends NamedEntity {
16
17     @JoinTable(
18         name = "rel_circuito_salas",
19         joinColumns = @JoinColumn(name="FK_Circuito"),
20         inverseJoinColumns = @JoinColumn(name="FK_Sala")
21     )
22
23
24     @ManyToMany
25     @Size(min = 2, message = "The circuit has to be composed of two rooms minimum.")
26     private List<Sala> salas;
27
28     private Integer aforo;
29
30     @Size(min= 10, message = "The description needs to have at least ten letters.")
31     @Column(name = "descripcion", length=1024)
32     private String descripcion;
33
34
35     public Sala getSala(String name) {
36         return getSala(name);
37     }
38
39     public void addSala(Sala s){
40         salas.add(s);
41     }
42
43     public void addSalas(List<Sala> s){
44         salas.addAll(s);
45     }
46
47

```

Horarios(/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/model/Horario.java)

Esta entidad es importante, ya que es un objeto del modelo de dominio en donde se guarda la fecha en la que un empleado cumple su horario laboral.

```
1 package org.springframework.samples.petclinic.model;
2
3 import java.time.LocalDate;
4
5 @Entity
6 @Table(name = "horario")
7 public class Horario extends BaseEntity {
8
9     @NotNull(message = "Date can't be null")
10    @Column(name = "fecha")
11    @DateTimeFormat(pattern = "yyyy/MM/dd")
12    private LocalDate fecha;
13
14    @NotNull
15    @ManyToOne
16    @JoinColumn(name = "employee_id")
17    private Employee employee;
18
19    @OneToMany(mappedBy = "horario", cascade = CascadeType.ALL)
20    private List<Sesion> sesiones;
21
22    public Horario() {
23        super();
24    }
25
26    public Horario(@NotNull(message = "Date can't be null") LocalDate fecha, @NotNull Employee employee,
27        List<Sesion> sesiones) {
28        super();
29        this.fecha = fecha;
30        this.employee = employee;
31        this.sesiones = sesiones;
32    }
33
34    public LocalDate getFecha() {
35        return fecha;
36    }
37
38    public Employee getEmployee() {
39        return employee;
40    }
41
42    public void setFecha(LocalDate fecha) {
43        this.fecha = fecha;
44    }
45
46    public void setEmployee(Employee employee) {
47        this.employee = employee;
48    }
49
50    public List<Sesion> getSesiones() {
51        return sesiones;
52    }
53
54 }
```

Servicio

SalaService(/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/service/SalaService.java)

En este servicio se encuentran los metodos que hacen eficiente la aplicacion a la hora de crear/editar salas, asi tambien la inyeccion de dependencia de esta clase en el controlador de Circuitos, asi correcto su funcionamiento. Tambien se han añadido la excepcion DuplicatedNameException.

```

1 package org.springframework.samples.petclinic.service;
2 import java.util.Collection;
3
12
13 @Service
14 public class SalaService {
15
16     SalaRepository salaRepo;
17
18     @Autowired
19     public SalaService(SalaRepository salaRepo) {
20         this.salaRepo = salaRepo;
21     }
22
23
24
25     public Collection<Sala> findAll(){
26         return salaRepo.findAll();
27     }
28
29     public Optional<Sala> findById(int id){
30         return salaRepo.findById(id);
31     }
32
33     // public List<Sala> findByIdLista(int id){
34     //     return salaRepo.findById(id);
35     // }
36
37     @Transactional
38     public void delete(Sala sala) {
39         salaRepo.deleteById(sala.getId());
40     }
41
42     @Transactional
43     public void save(@Valid Sala sala) {
44         salaRepo.save(sala);
45     }
46     public Sala getRoomwithIdDifferent(String name,Integer id) {
47         name=name.toLowerCase();
48         for (Sala sala: getRooms()) {
49             String compName=sala.getName();
50             compName=compName.toLowerCase();
51             if (compName.equals(name) && sala.getId()!=id) {
52                 return sala;
53             }
54         }
55         return null;
56     }
57     public Collection<Sala> getRooms(){
58         return salaRepo.findAll();
59     }
60     @Transactional(rollbackOn = {DuplicatedSalaNameException.class})
61     public void saveSala(Sala sala) throws DuplicatedSalaNameException {
62         Sala otherRoom=getRoomwithIdDifferent(sala.getName(), sala.getId());
63         if (StringUtils.hasLength(sala.getName()) && (otherRoom!= null && otherRoom.getId()!=sala.getId())) {
64             throw new DuplicatedSalaNameException();
65         }else
66             salaRepo.save(sala);
67     }
68 }

```

Controlador

CircuitoController(/dp1-2020-g2-

07/src/main/java/org/springframework/samples/petclinic/web/CircuitoController.java)

Este controlador permite varias funcionalidades para la manipulación de circuitos, así como tal muestras, en el caso que se comentan, todo tipo de excepciones.

```

1 package org.springframework.samples.petclinic.web;
2 import java.util.Collection;[]
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 @Slf4j
24 @Controller
25 @RequestMapping("/circuitos")
26 public class CircuitoController {
27
28     public static final String CIRCUITOS_FORM = "circuitos/createOrUpdateCircuitosForm";
29     public static final String CIRCUITOS_LISTING = "circuitos/CircuitosListing";
30
31     private final CircuitoService circuitosServices;
32     private final SalaService salasServices;
33
34
35     @Autowired
36     public CircuitoController(CircuitoService circuitosServices, SalaService salasServices) {
37         this.circuitosServices = circuitosServices;
38         this.salasServices = salasServices;
39     }
40
41     @GetMapping
42     public String circuitosListing(ModelMap model) {
43         Collection<Circuito> c = circuitosServices.findAll();
44         for(Circuito i:c) {
45             i.setAforo(circuitosServices.getAforo(i));
46         }
47         model.addAttribute("circuitos",c);
48         return CIRCUITOS_LISTING;
49     }
50
51     @GetMapping("/{id}/edit")
52     public String editCircuito(@PathVariable("id") int id, ModelMap model) {
53         Optional<Circuito> circuito = circuitosServices.findById(id);
54         if(circuito.isPresent()) {
55             Circuito c = circuito.get();
56             c.setAforo(circuitosServices.getAforo(c));
57             model.addAttribute("circuito",c);
58             model.addAttribute("salas", salasServices.findAll());
59             return CIRCUITOS_FORM;
60         } else {
61             model.addAttribute("message", "We could not find the circuit you are trying to edit.");
62         }
63         return CIRCUITOS_LISTING;
64     }
65
66     @PostMapping("/{id}/edit")
67     public String editCircuito(@PathVariable("id") int id, @Valid Circuito modifiedCircuito, BindingResult binding, ModelMap model) {
68         Optional<Circuito> circuito = circuitosServices.findById(id);
69         if(binding.hasErrors()) {
70             model.put("circuito", modifiedCircuito);
71             model.put("salas", salasServices.findAll());
72             Log.warn("Some mistakes here..");
73             return CIRCUITOS_FORM;
74         } else {
75             BeanUtils.copyProperties(modifiedCircuito, circuito.get(), "id", "aforo", "salas", "descripcion");
76             try{
77                 modifiedCircuito.setAforo(circuitosServices.getAforo(modifiedCircuito));
78                 this.circuitosServices.saveCircuito(modifiedCircuito);
79             } catch (DuplicatedCircuitoNameException ex){
80                 model.put("circuito", modifiedCircuito);
81                 model.put("salas", salasServices.findAll());
82                 binding.rejectValue("name", "duplicate", "already exists");
83                 Log.warn("Circuit with ID " + id + " has a duplicate name");
84                 return CIRCUITOS_FORM;
85             }
86             Log.info("Circuit with ID " + id + " was edit");
87             model.addAttribute("message", "The circuit was updated successfully.");
88             return circuitosListing(model);
89         }
90     }
91
92
93     @GetMapping("/{id}/delete")
94     public String deleteCircuito(@PathVariable("id") int id, ModelMap model) {
95         Optional<Circuito> circuito = circuitosServices.findById(id);
96         if(circuito.isPresent()) {
97             circuitosServices.delete(circuito.get());
98             model.addAttribute("message", "The circuit was deleted successfully.");
99             Log.info("Circuit with ID " + id + " was deleted");
100             return circuitosListing(model);
101         } else {
102             model.addAttribute("message", "We could not find the circuit you are trying to delete.");
103             Log.warn("Circuit with ID " + id + " does not exist");
104             return circuitosListing(model);
105         }
106     }
107
108     @GetMapping("/new")
109     public String newCircuito(ModelMap model) {
110         model.addAttribute("circuito", new Circuito());
111         model.addAttribute("salas", salasServices.findAll());
112         return CIRCUITOS_FORM;
113     }
114 }

```

```

6 @PostMapping("/new")
7 public String saveNewCircuito(@Valid Circuito circuito, BindingResult binding, ModelMap model) {
8     if(binding.hasErrors()) {
9         model.put("circuito", circuito);
10        model.put("salas", salasServices.findAll());
11        Log.warn("Some mistakes here..");
12        return CIRCUITOS_FORM;
13    }
14    else {
15        try{
16            circuito.setAforo(circuitosServices.getAforo(circuito));
17            this.circuitosServices.saveCircuito(circuito);
18        }catch(DuplicatedCircuitoNameException ex){
19            model.put("circuito", circuito);
20            model.put("salas", salasServices.findAll());
21            binding.rejectValue("name", "duplicate", "already exists");
22            Log.warn("This circuit has a duplicate name");
23            return CIRCUITOS_FORM;
24        }
25        model.addAttribute("message", "The circuit was created successfully.");
26        Log.info("You have created a new circuit!");
27        return circuitosListing(model);
28    }
29 }
30 }
31 }
32 }
33 }

```

Repositorio

CircuitoRepository(/dp1-2020-g2-

07/src/main/java/org/springframework/samples/petclinic/repository/CircuitoRepository.java)

Tenemos bastantes metodos asi para poder encontrar circuitos o circuito especifico o incluso para obtener salas de un circuito.

```

1 package org.springframework.samples.petclinic.repository;
2 import java.util.Collection;
3
11
12 public interface CircuitoRepository extends Repository<Circuito, Integer>, CrudRepository<Circuito, Integer> {
13
14     Collection<Circuito> findAll();
15     Optional<Circuito> findById(int id);
16     Optional<Circuito> findById(Integer id);
17
18     @Query("SELECT sala FROM Sala sala ORDER BY sala.name")
19     public Collection<Sala> getRoomsByCircuito();
20
21 }
22
23

```

Conversor o Formatter (si aplica)

No aplica.

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

No aplica.

Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

No aplica.

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

RoomMockTest(/dp1-2020-g2-

07/src/test/java/org/springframework/samples/petclinic/service/RoomMockTest.java)

En el arrange creamos la sala, en el act probamos el service y comprobamos mediante assert si devuelve lo que se pide. En el ultimo metodo probamos que salte la excepcion de DuplicatedName

```

29 @RunWith(MockitoJUnitRunner.class)
30 public class RoomMockTest {
31
32     @Mock
33     private SalaRepository salasRepo;
34
35     @Mock
36     private SalaService salaService;
37     private Sala s;
38     private List<Circuito> circuitos;
39     private Circuito c1;
40     private Circuito c2;
41     private Collection<Sala> salas;
42     private Optional<Sala> salaOpt;
43
44     @Before
45     public void setUp() {
46         salaService = new SalaService(salasRepo);
47         s = new Sala();
48         c1 = new Circuito();
49         c2 = new Circuito();
50         circuitos = new ArrayList<Circuito>();
51         salas = new ArrayList<Sala>();
52
53         c1.setAforo(7);
54         c1.setDescripcion("Prueba de circuito");
55         c1.setId(1);
56         c1.setName("Circuito1");
57
58         c2.setAforo(7);
59         c2.setDescripcion("Prueba de circuito");
60         c2.setId(1);
61         c2.setName("Circuito2");
62
63         circuitos.add(c1); circuitos.add(c2);
64         s.setCircuitos(circuitos);
65
66         s.setId(1);
67         s.setName("Jacuzzi");
68         s.setAforo(7);
69
70         salas.add(s);
71
72         salaOpt = Optional.of(s);
73
74         when(salasRepo.findAll()).thenReturn(salas);
75         when(salasRepo.findById(1)).thenReturn(salaOpt);
76         when(salasRepo.save(s)).thenReturn(s);
77
78     }
79
80     @Test
81     public void shouldFindAll() {
82         Collection<Sala> salaExample = this.salaService.findAll();
83
84         assertThat(salaExample).hasSize(1);
85         assertThat(salaExample.iterator().next().getName()).isEqualTo("Jacuzzi");
86     }
87
88     @Test
89     public void shouldFindById() {
90         Optional<Sala> optionalSalaExample1 = salaService.findById(1);
91         assertTrue(optionalSalaExample1.isPresent());
92
93         Optional<Sala> optionalSalaExample2 = salaService.findById(2);
94         assertFalse(optionalSalaExample2.isPresent());
95     }
96
97     @Test
98     public void shouldSave() {
99         salaService.save(s);
100     }
101
102     @Test
103     public void shouldDeleteRoom() {
104         salaService.delete(s);
105     }
106
107     @Test
108     public void shouldThrowExceptionInsertingRoomsInTheSameName() {
109         s = new Sala();
110         s.setId(1);
111         s.setName("Jacuzzi");
112         s.setAforo(7);
113         try {
114             salaService.saveSala(s);
115         } catch (DuplicatedSalaNameException e) {
116             // The room already exist
117             e.printStackTrace();
118         }
119         Sala anotherRoomWithTheSameName = new Sala();
120         anotherRoomWithTheSameName.setId(2);
121         anotherRoomWithTheSameName.setName("Jacuzzi");
122         anotherRoomWithTheSameName.setAforo(7);
123         Assertions.assertThrows(DuplicatedSalaNameException.class, () -> {
124             salaService.saveSala(anotherRoomWithTheSameName);
125         });
126     }
127 }

```

CircuitMockTest(/dp1-2020-g2-

07/src/test/java/org/springframework/samples/petclinic/service/CircuitMockTest.java)

En el arrange creamos la sala, en el act probamos el service y comprobamos mediante assert si devuelve lo que se pide. En el ultimo metodo probamos que salte la excepcion de DuplicatedName y Tambien probamos en el metodo aforoCorrect si devuelve el minimo del aforo de las salas.

```

1 package org.springframework.samples.petclinic.service;
2 import static org.assertj.core.api.Assertions.assertThat;
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 @ExtendWith(MockitoExtension.class)
27 @RunWith(MockitoJUnitRunner.class)
28 public class CircuitoMockTest {
29
30     @Mock
31     private CircuitoRepository circuitoRepo;
32
33
34     @Mock
35     private SalaRepository salaRepo;
36
37     @Mock
38     private CircuitoService circuitoService;
39     private Circuito c;
40     private Sala s;
41     private Sala s2;
42 // private List<Sala> salas;
43 private Collection<Circuito> circuitos;
44 private Optional<Circuito> circuitoOpt;
45
46
47 @Before
48 public void setUp() {
49     circuitoService = new CircuitoService(circuitoRepo);
50     c = new Circuito();
51     s = new Sala();
52     s2 = new Sala();
53     salas = new ArrayList<Sala>();
54     circuitos = new ArrayList<Circuito>();
55
56     s.setName("Jacuzzi");
57     s.setAforo(7);
58     s.setDescripcion("Prueba sobre las salas");
59     s.setId(1);
60
61     s2.setName("Relax Pool");
62     s2.setAforo(7);
63     s2.setDescripcion("Prueba sobre las salas");
64     s2.setId(2);
65
66     salas.add(s); salas.add(s2);
67     c.setSalas(salas);
68
69     c.setName("Circuito");
70     c.setDescripcion("Circuito lleno de relajacion");
71     c.setAforo(circuitoService.getAforo(c));
72     c.setId(1);
73
74     circuitos.add(c);
75
76     circuitoOpt = Optional.of(c);
77
78     when(circuitoRepo.findAll()).thenReturn(circuitos);
79     when(circuitoRepo.findById(1)).thenReturn(circuitoOpt);
80     when(circuitoRepo.save(c)).thenReturn(c);
81
82 }
83
84 @Test
85 public void shouldFindAll() {
86     Collection<Circuito> circuitoExample = this.circuitoService.findAll();
87
88     assertThat(circuitoExample).hasSize(1);
89     assertThat(circuitoExample.iterator().next().getName()).isEqualTo("Circuito");
90 }
91
92 @Test
93 public void shouldFindById() {
94     Optional<Circuito> optionalCircuitoExample1 = circuitoService.findById(1);
95     assertThat(optionalCircuitoExample1.isPresent());
96
97     Optional<Circuito> optionalCircuitoExample2 = circuitoService.findById(2);
98     assertThat(optionalCircuitoExample2.isPresent());
99
100 }
101
102

```



```

1      }
2
3  @Test
4  public void aforoCorrect() {
5      c = new Circuito();
6      s = new Sala();
7      s2 = new Sala();
8      salas = new ArrayList<Sala>();
9      circuitos = new ArrayList<Circuito>();
10
11      s.setName("Jacuzzi");
12      s.setAforo(7);
13      s.setDescripcion("Prueba sobre las salas");
14      s.setId(1);
15
16      s2.setName("Relax Pool");
17      s2.setAforo(7);
18      s2.setDescripcion("Prueba sobre las salas");
19      s2.setId(2);
20
21      salas.add(s); salas.add(s2);
22      c.setSalas(salas);
23
24      c.setName("Circuito");
25      c.setDescripcion("Circuito lleno de relajacion");
26      c.setAforo(circuitoService.getAforo(c));
27      c.setId(1);
28
29      assertEquals(7, 7);
30
31  }
32
33  @Test
34  public void shouldSave() {
35      circuitoService.save(c);
36  }
37
38  @Test
39  public void shouldDeleteCircuit() {
40      circuitoService.delete(c);
41  }
42
43  @Test
44  public void shouldThrowExceptionInsertingCircuitsWithTheSameName() {
45      c = new Circuito();
46      s = new Sala();
47      s2 = new Sala();
48      salas = new ArrayList<Sala>();
49      circuitos = new ArrayList<Circuito>();
50
51      s.setName("Jacuzzi");
52      s.setAforo(7);
53      s.setDescripcion("Prueba sobre las salas");
54      s.setId(1);
55
56      s2.setName("Relax Pool");
57      s2.setAforo(7);
58      s2.setDescripcion("Prueba sobre las salas");
59      s2.setId(2);
60
61      salas.add(s); salas.add(s2);
62      c.setSalas(salas);
63
64      c.setId(1);
65      c.setName("Circuito");
66      c.setDescripcion("Circuito lleno de relajacion");
67      c.setAforo(circuitoService.getAforo(c));
68
69      try {
70          circuitoService.saveCircuito(c);
71      } catch (DuplicatedCircuitoNameException e) {
72          // The room already exist
73          e.printStackTrace();
74      }
75      Circuito anotherCircuitWithTheSameName = new Circuito();
76      anotherCircuitWithTheSameName.setId(2);
77      anotherCircuitWithTheSameName.setName("Circuito");
78      anotherCircuitWithTheSameName.setAforo(7);
79      Assertions.assertThrows(DuplicatedCircuitoNameException.class, () -> {
80          circuitoService.saveCircuito(anotherCircuitWithTheSameName);
81      });
82  }
83

```

Pruebas unitarias parametrizadas (si aplica)

No aplica

Pruebas de controlador

CircuitControllerTest(/dp1-2020-g2-

07/src/test/java/org/springframework/samples/petclinic/web/CircuitControllerTest.java)

En el arrange creamos todos los objetos necesarios, en el caso del circuito, las salas son una colección. En los métodos vamos comprobando si el circuito existe, si tiene posibles fallos en los campos a rellenar en su formulario.

```

1 package org.springframework.samples.petclinic.web;
2
3 import java.util.ArrayList;
4
5 @WebMvcTest(controllers = CircuitoController.class, excludeFilters = @ComponentScan.Filter(type = Filter.Type.AUTOWIRE))
6 @SecurityConfiguration
7 public class CircuitoControllerTest {
8
9     private static final int TEST_CIRCUITO_ID=1;
10
11     @Autowired
12     private MockMvc mockMvc;
13     private Circuito c;
14     private Optional<Circuito> circuitoOpt;
15     private Sala s;
16     private Sala s2;
17     private Collection<Circuito> circuitos;
18
19     @MockBean
20     private CircuitoService circuitoService;
21
22     @MockBean
23     private SalaService salaService;
24
25     @BeforeEach
26     private void setUp() {
27
28         c = new Circuito();
29         s = new Sala();
30         s2 = new Sala();
31         List<Sala> salas = new ArrayList<Sala>();
32         //circuitos = new ArrayList<Circuito>();
33
34         s.setName("Jacuzzi");
35         s.setAforo(7);
36         s.setDescription("Prueba sobre las salas");
37         s.setId(1);
38
39         s2.setName("Relax Pool");
40         s2.setAforo(7);
41         s2.setDescription("Prueba sobre las salas");
42         s2.setId(2);
43
44         salas.add(s);
45         salas.add(s2);
46         c.setSalas(salas);
47
48         c.setName("Circuito");
49         c.setDescription("Circuito lleno de relajacion");
50         c.setAforo(circuitoService.getAforo(c));
51         c.setId(1);
52
53         circuitos = new ArrayList<Circuito>();
54         circuitos.add(c);
55         circuitoOpt = Optional.of(c);
56         given(this.circuitoService.findAll()).willReturn(circuitos);
57         given(this.circuitoService.findById(TEST_CIRCUITO_ID)).willReturn(circuitoOpt);
58         given(this.circuitoService.findRoomByCircuit()).willReturn(salas);
59     }
60
61     @WithMockUser(value="spring")
62     @Test
63     public void testCreationCircuit() throws Exception{
64         mockMvc.perform(get("/circuitos/new").andExpect(status().isOk()))
65             .andExpect(view().name("circuitos/createOrUpdateCircuitosForm"))
66             .andExpect(model().attributeExists("circuito"))
67             .andExpect(model().attributeExists("salas"));
68     }
69
70     @WithMockUser(value="spring")
71     @Test
72     public void testCreationCircuitSuccess() throws Exception{
73         mockMvc.perform(post("/circuitos/new").with(csrf()).param("name", "Circuito")
74             .param("salas", "Jacuzzi", "Relax Pool")
75             .param("aforo", "7")
76             .param("descripcion", "Circuito lleno de relajacion"))
77             .andExpect(status().isOk())
78             .andExpect(model().attribute("message", "The circuit was created successfully. "))
79             .andExpect(view().name("circuitos/CircuitosListing"));
80     }
81
82     @WithMockUser(value="spring")
83     @Test
84     public void testCreationCircuitHasErrors() throws Exception{
85         mockMvc.perform(post("/circuitos/new").with(csrf()).param("name", "Circuito")
86             .param("salas", "")
87             .param("aforo", "siete").param("descripcion", "Circuito lleno de relajacion"))
88             .andExpect(model().attributeHasFieldErrors("circuito"))
89             .andExpect(model().attributeHasFieldErrors("circuito", "aforo"))
90             .andExpect(model().attributeHasFieldErrors("circuito", "salas"))
91             .andExpect(status().isOk())
92             .andExpect(view().name("circuitos/createOrUpdateCircuitosForm"));
93     }
94
95     @WithMockUser(value="spring")
96     @Test
97     public void testEditCircuit() throws Exception{
98         mockMvc.perform(get("/circuitos/{id}/edit", TEST_CIRCUITO_ID))
99             .andExpect(status().isOk())
100             .andExpect(view().name("circuitos/createOrUpdateCircuitosForm"))
101             .andExpect(model().attributeExists("circuito"))
102             .andExpect(model().attributeExists("salas"));
103     }
104
105     @WithMockUser(value="spring")
106     @Test
107     public void testEditCircuitSuccess() throws Exception{
108         mockMvc.perform(post("/circuitos/{id}/edit", TEST_CIRCUITO_ID).with(csrf()).param("name", "Circuito")
109             .param("salas", "Jacuzzi", "Sauna")
110             .param("aforo", "6").param("descripcion", "Circuito lleno de relajacion"))
111             .andExpect(status().isOk())
112             .andExpect(model().attribute("message", "The circuit was updated successfully. "))
113             .andExpect(view().name("circuitos/CircuitosListing"));
114     }
115
116     @WithMockUser(value="spring")
117     @Test
118     public void testEditCircuitFailure() throws Exception{
119         mockMvc.perform(get("/circuitos/{id}/edit", 3).with(csrf()))
120             .andExpect(status().isOk())
121             .andExpect(model().attribute("message", "We could not find the circuit you are trying to edit. "))
122             .andExpect(view().name("circuitos/CircuitosListing"));
123     }
124
125     @WithMockUser(value="spring")
126     @Test
127     public void testDeleteCircuitSuccess() throws Exception{
128         mockMvc.perform(get("/circuitos/{id}/delete", TEST_CIRCUITO_ID))
129             .andExpect(status().isOk())
130             .andExpect(view().name("circuitos/CircuitosListing"))
131             .andExpect(model().attribute("message", "The circuit was deleted successfully. "));
132     }
133
134     @WithMockUser(value="spring")
135     @Test
136     public void testDeleteCircuitFailure() throws Exception{
137         mockMvc.perform(get("/circuitos/{id}/delete", 2))
138             .andExpect(status().isOk())
139             .andExpect(view().name("circuitos/CircuitosListing"))
140             .andExpect(model().attribute("message", "We could not find the circuit you are trying to delete. "));
141     }
142 }

```

Principales problemas encontrados

El principal problema que me he encontrado al desarrollar el proyecto fue el diseño de una lógica de negocio que satisfaga la funcionalidad de circuitos y salas. El desconocimiento de la relación ManytoMany y el correcto uso de ella.

Otros comentarios

No procede.