Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g3-14
Nombre de usuario en GitHub: JoseFrancisco-uvus
Rama del usuario en Github: josregmej
Nota: No hemos trabajado en nuestras ramas creadas al principio del curso ya que el equipo tomo la decisión de que trabajaríamos utilizando el modelo de ramificación para git, GitFlow.

Participación en el proyecto

Durante el proyecto, ha habido algunas historias de usuario que he realizado sólo y otras historias de usuario que he realizado junto a mis compañeros.

Historias de usuario en las que he participado

He desarrollado completas junto a mi compañero Pedro Jesús Muñoz Cifuentes las historias de usuario:

- HU12 – Búsqueda de hotel y vuelo
- HU20 – Búsqueda de hotel y actividad

He desarrollado completas junto a mi compañero Enrique Moreno Vázquez las historias de usuario:

- HU1 - Búsqueda de hoteles
- HU3 – Baja de hoteles
- HU11 - Búsqueda de actividades
- HU18 - Alta actividad

He desarrollado completas junto a mis compañeros Enrique Moreno Vázquez y Pedro Jesús Muñoz Cifuentes las historias de usuario:

- HU2 - Alta de hoteles
- HU10 – Sistema FAQ

Reglas de negocio en las que he participado

He desarrollado junto a mi compañero Enrique Moreno Vázquez, Javier Hidalgo Rodríguez las reglas de negocio:

- R1 – Privacidad en el proceso de registro
- R4 – Descuento por código

Nombre: José Francisco Regadera Mejías Grupo: G3-14

He desarrollado junto a mi compañero Pedro Jesús Muñoz Cifuentes, Enrique Moreno Vázquez, Javier Hidalgo Rodríguez, Pedro Pablo Carvajal Moreno las reglas de negocio:

- R2 – Restricciones contraseñas

He desarrollado junto a mi compañero Pedro Jesús Muñoz Cifuentes las reglas de negocio:

- R3 – No mostrar hoteles con baja valoración

Funcionalidad implementada

He añadido varios métodos a los controladores:

- ActividadController
- AgenActController
- HabitacionController
- HotelController
- HotelActividadController
- Search2EntitiesController
- VueloController
- UserController
- WelcomeController

He añadido varios métodos al servicio S1:

- ActividadService
- AgenActService
- HabitacionService
- HotelService
- UserService
- VueloService

También he creado las entidades:

- Actividad
- AgenAct
- Habitacion
- Hotel
- User
- Vuelo

Y las vistas:

- actividades
- agenacts
- habitaciones
- hotelActividad
- hoteles
- search
- vuelos

Esto hace un total de 21 clases implementadas por mí y 6 interfaces definidas.

Pruebas implementadas

Pruebas unitarias

He creado tests unitarios para los servicios:

- ActividadServiceTest
- AgenActServiceTest
- HabitacionServiceTest
- HotelServiceTest
- UserServiceTest

He creado tests para controladores:

- CompVuelosControllerTest
- HabitacionControllerTest
- HotelControllerTest

Eso hace un total de 8 clases de test unitarios con un total de 26 métodos anotados con @Test.

Pruebas de Controlador

He creado 0 casos de prueba positivo y 1 negativos de controlador para la HU6 – Búsqueda de vuelos.

He creado 2 casos de prueba positivo y 0 negativos de controlador para la HU1 – Búsqueda de hoteles.

He creado 4 casos de prueba positivo y 3 negativos de controlador para la HU2 – Alta de hoteles.

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

Entidad Hotel

Ruta: /src/main/java/org/springframework/samples/petclinic/model/Hotel

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

import java.util.List;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
```

```
import javax.persistence.Entity;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.Digits;
import javax.validation.constraints.NotEmpty;
import org.hibernate.validator.constraints.Range;


@Entity
@Table(name = "hoteles")
public class Hotel extends BaseEntity{

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

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

	@Column(name = "estrellas")
	@Range(min=1,max=5)
	private Integer estrellas;

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

	@Column(name = "telefono")
	@NotEmpty
	@Digits(fraction = 0, integer = 10, message = "Debe contener 9 dígitos")
	private String telefono;

	@Column(name = "valido")
	private Boolean valido;

	@OneToMany(cascade = CascadeType.ALL, mappedBy = "hotel")
	private Set<Habitacion> habitaciones;


	public Set<Habitacion> getHabitaciones() {
		return habitaciones;
	}

	public void setHabitaciones(Set<Habitacion> habitaciones) {
		this.habitaciones = habitaciones;
	}
```

```
@OneToMany(cascade = CascadeType.MERGE, mappedBy = "hotel")
private List<ComentarioHotel> comentarios;


public List<ComentarioHotel> getComentarios() {
	return comentarios;
}

public void setComentarios(List<ComentarioHotel> comentarios) {
	this.comentarios = comentarios;
}

public String getNombre() {
	return nombre;
}

public void setNombre(String nombre) {
	this.nombre = nombre;
}

public String getDireccion() {
	return direccion;
}

public void setDireccion(String direccion) {
	this.direccion = direccion;
}

public Integer getEstrellas() {
	return estrellas;
}

public void setEstrellas(Integer estrellas) {
	this.estrellas = estrellas;
}

public String getProvincia() {
	return provincia;
}

public void setProvincia(String provincia) {
	this.provincia = provincia;
}

public String getTelefono() {
	return telefono;
```

```
	}

	public void setTelefono(String telefono) {
		this.telefono = telefono;
	}

   public Boolean getValido() {
		return valido;
	}

	public void setValido(Boolean valido) {
		this.valido = valido;
	}

}
```

Esta entidad indica que datos de hotel van a guardarse: nombre, dirección, estrellas, provincia, teléfono. Además cuenta con una relación con habitaciones y con comentarioHotel que sirve para obtener la valoración sobre ese hotel.

Entidad Actividad

Ruta: src/main/java/org/springframework/samples/petclinic/model/Actividad

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

import java.time.LocalDate;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.validation.constraints.FutureOrPresent;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.Range;
import org.springframework.format.annotation.DateTimeFormat;
```

```
@Entity
@Table(name = "actividades")
public class Actividad extends BaseEntity{


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

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

	@Column(name = "valoracion")
	@Range(min=1,max=5)
	private Integer valoracion;

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

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

	@Column(name = "precio")
	@NotNull
	private Integer precio;

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

	@ManyToOne
	@JoinColumn(name = "agenact_id")
	private AgenAct agenact;

	@ManyToMany(mappedBy = "actividades")
	private Set<User> users;

	@OneToMany(cascade = CascadeType.MERGE, mappedBy = "actividad")
	private List<ComentarioActividad> comentarios;

	@OneToOne
	@JoinColumn(name = "reservaactividad")
	private ReservaActividad reservaactividad;
```

```
public List<ComentarioActividad> getComentarios() {
	return comentarios;
}

public void setComentarios(List<ComentarioActividad> comentarios) {
	this.comentarios = comentarios;
}

public String getNombre() {
	return nombre;
}

public void setNombre(String nombre) {
	this.nombre = nombre;
}

public String getDescripcion() {
	return descripcion;
}

public void setDescripcion(String descripcion) {
	this.descripcion = descripcion;
}

public Integer getValoracion() {
	return valoracion;
}

public void setValoracion(Integer valoracion) {
	this.valoracion = valoracion;
}

public String getDireccion() {
	return direccion;
}

public void setDireccion(String direccion) {
	this.direccion = direccion;
}

public Integer getPrecio() {
	return precio;
}

public void setPrecio(Integer precio) {
	this.precio = precio;
```

```
	}

	public String getProvincia() {
		return provincia;
	}

	public void setProvincia(String provincia) {
		this.provincia = provincia;
	}

	public LocalDate getFecha() {
		return fecha;
	}

	public void setFecha(LocalDate fecha) {
		this.fecha = fecha;
	}

	public AgenAct getAgenact() {
		return agenact;
	}

	public void setAgenact(AgenAct agenact) {
		this.agenact = agenact;
	}

	protected Set<User> getUsersInternal() {
		if (this.users == null) {
			this.users = new HashSet<>();
		}
		return this.users;
	}

	public Set<User> getUsers() {
		return users;
	}

	public void setUsers(Set<User> users) {
		this.users = users;
	}

	public void setUsersInternal(Set<User> users) {
		this.users = users;
	}

	public ReservaActividad getReservaactividad() {
		return reservaactividad;
```

```
	}

	public void setReservaactividad(ReservaActividad reservaactividad) {
		this.reservaactividad = reservaactividad;
	}



}
```

Esta entidad indica que se debe guardar en actividad: nombre, descripción, valoración, dirección, provincia, precio, fecha. Además cuenta con la relación con AgenAct, con comentarioActividad y con reservaActividad que mantiene el registro de la reserva de esa actividad. Esta entidad fue realizada junto con Enrique Moreno Vázquez.

Servicio

Servicio Actividad

Ruta: /src/main/java/org/springframework/samples/petclinic/service/ActividadService

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

import java.util.Collection;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.samples.petclinic.model.Actividad;
import org.springframework.samples.petclinic.model.Vuelo;
import org.springframework.samples.petclinic.repository.ActividadRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;


@Service
public class ActividadService {

	private ActividadRepository actividadRepository;

	@Autowired
	public ActividadService(ActividadRepository actividadRepository) {
		this.actividadRepository = actividadRepository;
	}

	@Transactional(readOnly = true)
	public Actividad findActividadById(int id) throws DataAccessException {
		return actividadRepository.findById(id);
```

```
	}

	@Transactional
	public void saveActividad(Actividad actividad) throws DataAccessException {
		actividadRepository.save(actividad);
	}

	@Transactional(readOnly = true)
	public Collection<Actividad> findByNombre(String nombre) throws
DataAccessException {
		return actividadRepository.findByNombreLike(nombre);
	}

	@Transactional(readOnly = true)
	public List<Actividad> findAllActividades() throws DataAccessException {
		return actividadRepository.findAllActividades();
	}
   @Transactional(readOnly = true)
	public Collection<Actividad> findByPrecio(Integer precio) throws
DataAccessException{
		return actividadRepository.findByActividadPrecioLessThanEqual(precio);
	}
}
```

En esta clase se encuentran los métodos que son usados por los controladores, cabe destacar los métodos: findByNombre que devuelve la actividad con el nombre dado, findAllActividades que devuelve todas las actividades y findByPrecio que devuelve todas las actividades cuyo precio sea menor o igual al dado. El método findByPrecio fue realizado junto con mi compañero Pedro Jesús Muñoz Cifuentes.

Controlador

Controlador Actividad

Ruta: /src/main/java/org/springframework/samples/petclinic/web/ActividadController

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

import java.util.Collection;
import java.util.Map;
import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.samples.petclinic.model.Actividad;
import org.springframework.samples.petclinic.service.ActividadService;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class ActividadController {

	private static final String VIEWS_ACTIVIDAD_CREATE_OR_UPDATE_FORM =
"actividades/createOrUpdateActividadForm";
	private final ActividadService actividadService;

	@Autowired
	public ActividadController(ActividadService actividadService) {
			this.actividadService = actividadService;
	}

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

	@GetMapping(value = "/actividades/new")
	public String initCreationForm(Map<String, Object> model) {
		Actividad actividad = new Actividad();
		model.put("actividades", actividad);
		return VIEWS_ACTIVIDAD_CREATE_OR_UPDATE_FORM;
	}

	@PostMapping(value = "/actividades/new")
	public String processCreationForm(@Valid Actividad actividad, BindingResult
result, Map<String, Object> model) {
		if (result.hasErrors()) {
			model.put("actividades", actividad);
			return VIEWS_ACTIVIDAD_CREATE_OR_UPDATE_FORM;
		}
		else {
			this.actividadService.saveActividad(actividad);
	       return "redirect:/actividades/"+actividad.getId();
		}
	}

	@GetMapping(value = "/actividades/{actividadId}/edit")
	public String initUpdateForm(@PathVariable("actividadId") int actividadId,
ModelMap model) {
		Actividad actividad = this.actividadService.findActividadById(actividadId);
```

```
		model.put("actividades",actividad);
		return VIEWS_ACTIVIDAD_CREATE_OR_UPDATE_FORM;
	}

	@PostMapping(value = "/actividades/{actividadId}/edit")
	public String processUpdateActividadForm(@Valid Actividad actividad,
BindingResult result,
			@PathVariable("actividadId") int actividadId,ModelMap model) {
		if (result.hasErrors()) {
			model.put("actividades",actividad);
			return VIEWS_ACTIVIDAD_CREATE_OR_UPDATE_FORM;
		}
		else {
			actividad.setId(actividadId);
			this.actividadService.saveActividad(actividad);
			return "redirect:/actividades/{actividadId}";
		}
	}

	@GetMapping(value = "/actividades/find")
	public String initFindForm(Map<String, Object> model) {
		model.put("actividades", new Actividad());
		return "actividades/findActividades";
	}


	@GetMapping(value = "/actividades")
	public String processFindForm(Actividad actividad, BindingResult result,
Map<String, Object> model) {

		if (actividad.getNombre() == null) {
			actividad.setNombre(""); // empty string signifies broadest possible
search
		}

		Collection<Actividad> results =
this.actividadService.findByNombre(actividad.getNombre());
		if (results.isEmpty()) {
			result.rejectValue("nombre", "notFound", "not found");
			return "actividades/findActividades";
		}
		else if (results.size() == 1) {
			actividad = results.iterator().next();
			return "redirect:/actividades/" + actividad.getId();
		}
		else {
			model.put("selections", results);
```

```
                        return "actividades/actividadesList";
                }
        }

        @GetMapping(value = "/actividades/findActividadesPrecio")
        public String initFindFormPrecio(Map<String, Object> model) {
                model.put("actividad", new Actividad());
                return "actividades/findActividadesPrecio";
        }

        @GetMapping(value = "/actividades/precio")
        public String processFindFormPrecio(Actividad actividad, BindingResult result,
Map<String, Object> model) {

                if (actividad.getPrecio() == null) {
                        actividad.setPrecio(999999); // empty string signifies broadest
possible search
                }

                Collection<Actividad> results =
this.actividadService.findByPrecio(actividad.getPrecio());
                if (results.isEmpty()) {
                        result.rejectValue("precio", "notFound", "not found");
                        return "actividades/findActividadesPrecio";
                }
                else if (results.size() == 1) {
                        actividad = results.iterator().next();
                        return "redirect:/actividades/" + actividad.getId();
                }
                else {
                        model.put("selections", results);
                        return "actividades/actividadesListPrecio";
                }
        }

        @GetMapping("/actividades/{actividadId}")
        public ModelAndView showActividad(@PathVariable("actividadId") int
actividadId) {
                ModelAndView mav = new ModelAndView("actividades/actividadDetails");
                mav.addObject("actividades",
this.actividadService.findActividadById(actividadId));
                return mav;
        }
}
```

Este controlador permite la búsqueda de actividades tanto por nombre, devolviendo las que coincidan con el nombre o todas si está vacío, como por el

precio, devolviendo cuyo precio sea menor o igual al dado. También permite la creación y modificación de actividades y el showActividad muestra los detalles de la actividad. Los métodos de initFindFromPrecio y processFindFromPrecio fueron realizadas junto con mi compañero Pedro Jesús Muñoz Cifuentes.

Repositorio

Repositorio Actividad

Ruta:
/src/main/java/org/springframework/samples/petclinic/repository/ActividadRepository

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

import java.util.Collection;
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.samples.petclinic.model.Actividad;

public interface ActividadRepository extends JpaRepository<Actividad, Long>  {

	@Query("select u from Actividad u where u.nombre like %?1")
	Collection<Actividad> findByNombreLike(String nombre);

	@Query("select a from Actividad a")
	public List<Actividad> findAllActividades();

	Actividad findById(int id);

	@Query("select a from Actividad a where a.precio <= ?1")
	Collection<Actividad> findByActividadPrecioLessThanEqual(Integer precio);
}
```

Aquí se realizan las consultas a la base de datos, cabe destacar: findAllActividades que devuelve todas las actvidiades, findActividadByNombreLike que devuelve la actividad con el nombre indicado y findActividadesPrecioLessThanEqual. Este último método fue hecho junto con mi compañero Pedro Jesús Muñoz Cifuentes.

Conversor o Formatter (si aplica)

No aplica.

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

Criterios de seguridad en la contraseña

```
@Pattern(regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)[a-zA-Z\\d]{8,}$", message="La contraseña debe tener 8 cáracteres minimo, una letra mayúscula y una minúscula, al menos un número")
String password;
```

Este validador hace que solo se permitan contraseñas con un mínimo de 8 caracteres, una minúscula, una mayúscula y un número. Este validador fue realizado junto a mis compañeros Pedro Pablo Carvajal Moreno, Javier Hidalgo Rodriguez, Enrique Moreno Vázquez y Pedro Jesús Muñoz Cifuentes.

Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

No aplica.

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

Ruta: /src/test/java/org/springframework/samples/petclinic/service/HotelServiceTests

Caso insertar hotel

```
@Test                                                                    A
        @Transactional                                                   R
        public void shouldInsertHotel() {                                R
                Collection<Hotel> hoteles =                              A
this.hotelService.findByNombre("Sunset");                                N
                //No debe existir el hotel con ese nombre en la base de  G
datos                                                                    E
                int found = hoteles.size();
                //Tamaño 0

        System.out.println("==================================
====================");
                System.out.println("Hotel Sunset no encontrado. Found=
"+found);

        System.out.println("==================================
====================");

                Hotel hotel = new Hotel();
```

hotel.setNombre("Sunset");
hotel.setDireccion("Calle Overdrive");
hotel.setEstrellas(4);
hotel.setProvincia("Granada");
hotel.setTelefono("666555111");
System.out.println("HOTEL: "+hotel);
System.out.println("ID HOTEL: "+hotel.getId());
// hotel.setId(1234);
System.out.println("ID HOTEL: "+hotel.getId());
//Opcional, como está relacionado con habitaciones le creo 2 habitaciones
Habitacion hab1=new Habitacion();
hab1.setNhabitacion(001);
hab1.setNcamas(2);
hab1.setPrecio(541);
hab1.setDisponible(true);
hab1.setHotel(hotel);
System.out.println(hab1);
 	
Habitacion hab2=new Habitacion();
hab2.setNhabitacion(002);
hab2.setNcamas(2);
hab2.setPrecio(541);
hab2.setDisponible(true);
hab2.setHotel(hotel);
 	
System.out.println(hab2);
 	
Set<Habitacion> habitaciones=new HashSet<Habitacion>();
habitaciones.add(hab1);
habitaciones.add(hab2);
 	
//Añado las habitaciones
hotel.setHabitaciones(habitaciones);	

this.hotelService.saveHotel(hotel);	**A C**

System.out.println("assertThat"+hotel.getId()); assertThat(hotel.getId()).isNotEqualTo(0);	**ASSERT**

//Comprobamos que se ha añadido sin problemas
hoteles = this.hotelService.findByNombre("Sunset");
	ACT

assertThat(hoteles.size()).isEqualTo(found+1);
System.out.println("Hotel Sunset encontrado. Found= "+hoteles.size());
 	
System.out.println("==========");
}
 	
	ASSERT

En el ARRANGE de este caso positivo, comprobamos que se guarda el hotel con todos los datos necesarios, como hotel tiene relación con habitaciones se crearon unas habitaciones que se añadieron al hotel antes de guardarlas. En el ACT se guarda el hotel y se busca ese hotel para comprobar en los ASSERT que antes de guardarlo no existía y que después de guardarlo sí. Esta prueba se hizo para comprobar que los hoteles se guardaban correctamente, por eso no se deja ningún campo vacío o con valores nulos.

Caso borrar hotel

@Test
void shouldDeleteHotel() {
Hotel hotel = new Hotel();
hotel.setNombre("Sunset");
hotel.setDireccion("Calle Overdrive");
hotel.setEstrellas(4);
hotel.setProvincia("Granada");
 	
hotel.setTelefono("666555111");
System.out.println("HOTEL: "+hotel);
System.out.println("ID HOTEL: "+hotel.getId());
// hotel.setId(1234);
	ARRANGE

System.out.println("ID HOTEL: "+hotel.getId());
//Opcional, como está relacionado con habitaciones le creo 2 habitaciones
Habitacion hab1=new Habitacion();
hab1.setNhabitacion(001);
hab1.setNcamas(2);
hab1.setPrecio(541);
hab1.setDisponible(true);
hab1.setHotel(hotel);
System.out.println(hab1);
 	
Habitacion hab2=new Habitacion();
hab2.setNhabitacion(002);
hab2.setNcamas(2);
hab2.setPrecio(541);
hab2.setDisponible(true);
hab2.setHotel(hotel);
 	
System.out.println(hab2);
 	
Set<Habitacion> habitaciones=new HashSet<Habitacion>();
habitaciones.add(hab1);
habitaciones.add(hab2);
 	
//Añado las habitaciones
hotel.setHabitaciones(habitaciones);	

this.hotelService.saveHotel(hotel);
Collection<Hotel> hoteles = this.hotelService.findByNombre("Sunset");
int found = hoteles.size();
//Tamaño 1
 	
System.out.println("==========");
System.out.println("Hotel Sunset encontrado. Found= "+found);
	ACT

System.out.println("==========
==========");
 	
this.hotelService.deleteHotel(hotel);
hoteles = this.hotelService.findByNombre("Sunset");	

assertThat(hoteles.size()).isEqualTo(found-1);
System.out.println("Hotel Sunset no encontrado. Found=
"+hoteles.size());
}
	ASSERT

En el ARRANGE de esta prueba se crea hotel con los datos necesarios y como tiene relación con habitaciones se crearon unas habitaciones que se añadieron al hotel antes de guardarlas; en el ACT, se guarda el hotel y posteriormente se borra; y en el ASSERT, se comprueba que ese hotel ha sido borrado. Esta prueba se hizo para comprobar que la operación de eliminar hotel funcionaba bien.

Pruebas unitarias parametrizadas(si aplica)

No aplica

Pruebas de controlador

Ruta: /src/test/java/org/springframework/samples/petclinic/web/HotelControllerTests

ARRANGE

```
@BeforeEach
	void setup() {

		hotelazo = new Hotel();
		hotelazo.setId(TEST_HOTEL_ID);
		hotelazo.setNombre("Hotelazo");
		hotelazo.setDireccion("Calle normal");
		hotelazo.setProvincia("Cadiz");
		hotelazo.setEstrellas(5);

		hotelazo.setTelefono("945122241");
		//Deberá devolver el hotel
```

```
        given(this.hotelService.findHotelById(TEST_HOTEL_ID)).willReturn(hot
elazo);

        }
```

Caso positivo

@WithMockUser(value = "spring") @Test void testProcessUpdateFormSuccess() throws Exception { mockMvc.perform(post("/hoteles/{hotelId}/edit", TEST_HOTEL_ID) .with(csrf()) .param("nombre", "Joe") .param("direccion", "123 Prueba Street") .param("estrellas", "4") .param("provincia", "Sevilla") .param("telefono", "013167648"))	**A** **C** **T**
.andExpect(status().is3xxRedirection()) .andExpect(view().name("redirect:/hoteles/{hotelId}")); } 	**A** **S** **S** **E** **R** **T**

En esta prueba de controlador, comprobamos que cuando editamos un hotel, se redirige a la página que muestra los detalles de ese hotel. Todos los valores editados en el ACT están puestos cumpliendo con las restricciones por lo que se comprueba que se edita de forma correcta.

Caso Negativo

@WithMockUser(value = "spring")	**A**

@Test void testProcessUpdateFormHasErrors() throws Exception { mockMvc.perform(post("/hoteles/{hotelId}/edit", TEST_HOTEL_ID) .with(csrf()) .param("nombre", "Joe") .param("direccion", "123 Prueba Street") .param("estrellas", "4") .param("telefono", "013167648"))	**C T**
.andExpect(model().attributeHasErrors("hotel")).andExpect(status().isOk()) .andExpect(view().name("hoteles/createOrUpdateHotelForm")); }	**A S S E R T**

En esta prueba de controlador, comprobamos que cuando editamos un hotel que contenga errores durante su modificación, se redirige a la página que permite la edición de los datos. Todos los valores editados en el ACT están puestos cumpliendo con las restricciones salvo provincia que no aparece y por tanto no tiene valor, por lo que se comprueba que si no están todos los datos correctamente puestos durante la edición es devuelto a la página de editar hotel.

Principales problemas encontrados

Los principales encontrados durante el proyecto han sido:

1. No poder usar select en el jsp para realizar el filtrado.
    a) La solución tomada fue redirigir a otra lista que: mostrase sólo los hoteles cuya provincia coincidiera; lo mismo se hizo con vuelo pero con el destino.
    b) Para las actividades, se creó un buscador que al escribirle una cifra, buscase todas las actividades cuyo precio fuese menor o igual al dado.

2. El borrado de hoteles con valoración baja.
    a) Al no poder borrar el hotel con valoraciones bajas porque generaba conflictos si un usuario ya había reservado ese hotel, se optó por

ocultarlo de la lista de hoteles y solo mostrar aquellos hoteles cuya valoración fuese media o alta; de esta forma aquellos usuarios que hubiesen reservado un hotel con valoración baja conservaban su reserva.

Otros comentarios

No hemos trabajado en nuestras ramas creadas al principio del curso ya que el equipo tomo la decisión de que trabajaríamos utilizando el modelo de ramificación para git, GitFlow. Es decir, para cada funcionalidad/prueba/regla de negocio/etc se crearon sus ramas correspondientes.