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: pedcarmor
Rama del usuario en Github: pedcarmor

Participación en el proyecto

Historias de usuario en las que he participado

He implementado completas las historias de usuario HU-1, HU-3, HU-5, HU-6, HU-8, HU-15, además, he desarrollado junto a mi compañero Pedro Jesús Muñoz las historias de usuario HU-6, HU-8 y la regla de negocio RN2.

He implementado junto a mi compañero Javier Hidalgo las historias de usuario HU-3, HU-5 y la regla de negocio RN2.

He implementado junto a mi compañero Enrique Moreno las historias de usuario HU-15 y la regla de negocio RN2.

He implementado junto a mi compañero José Francisco Regadera la regla de negocio RN2.

Funcionalidad implementada

He implementado el controlador AgenActController, ActividadController, ReservaHabitacionController, VueloController, InscripcionHotelController he añadido varios métodos al servicio AgenActService, ActividadService, HotelService, InscripcionHotelService, ReservaHabitacionService, VueloService. También he creado la entidad AgenAct, Actividad, InscripcionHotel, ReservaHabitacion, Vuelo, y las vistas agenacts, actividades, reservaHabitacion, hoteles, inscripcionhoteles, vuelos.

He implementado un formatter para las fechas de la entidad Vuelo.

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

He implementado un formatter para las fechas de la entidad ReservaHabitacion

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

He implementado un formatter para el cvc de la entidad ReservaHabitacion

```
@Pattern(regexp="\\d{3}",message = "Debe contener 3 dígitos")
```

Esto hace un total de 15 clases implementadas por mí y 6 interfaces definidos.

Pruebas implementadas

Pruebas unitarias

He creado tests unitarios para 8 servicios (CompVuelosServiceTests, AgenActsServiceTests, HotelServiceTests, ActividadServiceTests, ComentarioActividadServiceTests, ComentarioHotelServiceTests, InscripcionHotelServiceTest, VueloServiceTests), 7 controladores (HotelControllerTests, ActividadControllerTests, CompVuelosControllerTests, AgenActsControllerTests, ComentarioActividadControllerTests, ComentarioHotelControllerTests, InscripcionHotelControllerTests). Eso hace un total de 15 clases de test unitarios con un total de 62 métodos anotados con @Test.

Pruebas de Controlador

He creado 1 casos de prueba positivo y 1 negativos de controlador para la HU-18.
He creado 1 casos de prueba positivo y 1 negativos de controlador para la HU-7.
He creado 1 casos de prueba positivo y 1 negativos de controlador para la HU-9.
He creado 1 casos de prueba positivo y 1 negativos de controlador para la HU-1.
He creado 2 casos de prueba positivo y 1 negativos de controlador para la HU-2.
He creado 3 casos de prueba positivo y 1 negativos de controlador para la HU-8.

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

- **ReservaHabitacion** se encuentra en la ruta
\src\main\java\org\springframework\samples\petclinic\model\ReservaHabitacion.java
Esta entidad se ha realizado en colaboración con mi compañero Enrique Moreno.

```
@Entity
@Table(name="reservashabitacion")

public class ReservaHabitacion extends BaseEntity{

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

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

            @Column(name = "salida")
            @DateTimeFormat(pattern = "yyyy/MM/dd")
            @Future
          @NotNull
            private LocalDate salida;
```

```
            @Column(name = "numeroTarjeta")
            @CreditCardNumber
            private String numeroTarjeta;

            @Column(name = "cvc")
            @Pattern(regexp="\\d{3}",message = "Debe contener 3
dígitos")
            private String cvc;

            @Column(name="precio")
            private Double precioFinal;

            @ManyToOne
            @JoinColumn(name = "username")
            private User user;

            @OneToOne
            @JoinColumn(name = "habitacion_id")
            private Habitacion habitacion;
```

En la entidad ReservaHabitacion podemos ver cómo están los atributos necesarios para realizar una reserva, como es la fecha de entrada y salida, el número de tarjeta y el cvc, así como las relaciones entre los usuarios y las habitaciones.

- **Vuelo** se encuentra en la ruta \src\main\java\org\springframework\samples\petclinic\model\Vuelo.java

```
@Entity
@Table(name = "vuelos")
public class Vuelo extends BaseEntity {

	@Column(name = "billetes")
	private Integer billetes;

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

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

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

	@Column(name = "fechaVuelta")
	@DateTimeFormat(pattern = "yyyy/MM/dd")
```

```
	private LocalDate fechaVuelta;

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

	@Column(name = "numeroPlazas")
	private Integer numeroPlazas;

	@ManyToOne
	@JoinColumn(name = "compvuelo_id")
	private CompVuelos compVuelo;

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


	@OneToOne
	@JoinColumn(name = "reservavuelo")
	private ReservaVuelo reservavuelo;
```

La entidad Vuelo contiene todos los atributos necesarios para poder luego realizar una reserva junto con las relaciones entre las compañías aéreas, los usuarios y las reservas de vuelos.

Servicio

- **InscripcionHotelService** se encuentra en la ruta src\main\java\org\springframework\samples\petclinic\service\InscripcionHotelService.java

```
@Service
public class InscripcionHotelService {

	private InscripcionHotelRepository inscripcionHotelRepository;

	@Autowired
	public InscripcionHotelService(InscripcionHotelRepository
inscripcionHotelRepository) {
		this.inscripcionHotelRepository =
inscripcionHotelRepository;
	}

	@Transactional(readOnly = true)
	public InscripcionHotel findInscripcionHotelById(int id)
throws DataAccessException {
		return inscripcionHotelRepository.findById(id);
	}

	@Transactional
```

```
	public void saveInscripcionHotel(InscripcionHotel
inscripcionHotel) throws DataAccessException {
		inscripcionHotelRepository.save(inscripcionHotel);
	}

	public List<InscripcionHotel> findAll() throws
DataAccessException {
		return inscripcionHotelRepository.findAll();
	}
```

Contiene los métodos correspondientes a guardar las inscripciones en la base de datos, los métodos de buscar una inscripción mediante el ID y encontrar todas las inscripciones disponibles en la base de datos.

Controlador

- **ReservaHabitacionController** se encuentra en la ruta \src\main\java\org\springframework\samples\petclinic\web\ReservaHabitacionController.java

Este controlador se ha realizado con mis compañeros Enrique Moreno y Javier Hidalgo.

```
@Controller
@RequestMapping("/hoteles/{hotelId}/{nhabitacion}/")
public class ReservaHabitacionController {

	private ReservaHabitacionService reservaHabitacionService;
	private HabitacionService habitacionService;
	private UserService userService;
	private static final String
VIEWS_RESERVAHABITACION_CREATE_FORM =
"reservaHabitacion/createReservaHabitacionForm";

	@Autowired
	public ReservaHabitacionController(final
ReservaHabitacionService reservaHabitacionService,final
HabitacionService habitacionService, final UserService userService)
{
		this.reservaHabitacionService = reservaHabitacionService;
		this.habitacionService=habitacionService;
		this.userService= userService;
	}

	@GetMapping(value = "reservaHabitacion/new")
	public String initCreationForm(Map<String, Object> model) {
		ReservaHabitacion reservaHabitacion = new
ReservaHabitacion();
		model.put("reservaHabitacion", reservaHabitacion);
		return VIEWS_RESERVAHABITACION_CREATE_FORM;
```

```
	}

	@PostMapping(value = "reservaHabitacion/new")
	public String processCreationForm(@PathVariable("nhabitacion")
int nhabitacion,
			@Valid ReservaHabitacion reservaHabitacion,
BindingResult result, Map<String, Object> model) {
		reservaHabitacion.setFechaReserva(LocalDate.now());
		if (result.hasErrors()) {
			model.put("reservaHabitacion", reservaHabitacion);
			return VIEWS_RESERVAHABITACION_CREATE_FORM;
		} else {
			Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal(
);
			UserDetails userDetails = null;
			if (principal instanceof UserDetails) {
				userDetails = (UserDetails) principal;
				}
			String userName = userDetails.getUsername();
			User user=
this.userService.findByUsername(userName);
			Habitacion h =
this.habitacionService.findHabitacionByNhabitacion(nhabitacion);
			Set<User> ls = h.getUsers();
			ls.add(user);
			h.setUsers(ls);
			h.setDisponible(false);
			reservaHabitacion.setHabitacion(h);
			reservaHabitacion.setUser(user);
			LocalDate entrada = reservaHabitacion.getEntrada();
			LocalDate salida = reservaHabitacion.getSalida();
			Double dias = (double) DAYS.between(entrada,
salida);
			Double precio = dias*h.getPrecio();
			reservaHabitacion.setPrecioFinal(precio);

	this.reservaHabitacionService.saveReservaHabitacion(reservaHab
itacion);
			return "redirect:"+reservaHabitacion.getId();
		}
	}

	@GetMapping("reservaHabitacion/{reservaHabitacionId}")
	public ModelAndView
showReservaHabitacion(@PathVariable("reservaHabitacionId") int
reservaHabitacionId) {
		ModelAndView mav = new
ModelAndView("reservaHabitacion/reservaHabitacionDetails");
```

```
            mav.addObject("reservaHabitacion",
this.reservaHabitacionService.findReservaHabitacionById(reservaHabit
acionId));
            return mav;
      }
}
```

Este controlador solo contiene dos rutas, la ruta correspondiente a al formulario para poder realizar una reserva de habitación y que te muestre el detalle de esa reserva. A la hora de realizar el formulario de reserva, debemos obtener el usuario que se encuentra loggueado mediante UserDetails, así como obtener la fecha actual para guardarla en la fecha de realización de la reserva. Además, debemos cambiar el estado de la habitación de disponible a no disponible poniendo el atributo Disponible a false y se calculará el precio final mediante la multiplicación entre la diferencia de los días reservados por el precio de la habitación.

Mediante @RequestMapping conseguimos obtener los valores del id del hotel y el número de la habitación que queremos reservar.

Repositorio

- **ReservaHabitacionRepository** se encuentra en la ruta \src\main\java\org\springframework\samples\petclinic\repository\ReservaHabitacionRepository.java

```
public interface ReservaHabitacionRepository extends
JpaRepository<ReservaHabitacion, Integer>{

      ReservaHabitacion findById(int id);
      @Query("select h from ReservaHabitacion h where
h.user.username like %?1")
      Collection<ReservaHabitacion>
findReservaHabitacionByUserLike(String username);
}
```

La función principal de este repositorio es el de buscar una reserva de una habitación realizada por el usuario, pasando como parámetro el nombre de usuario. Ya que extiende de JpaRepository no es necesario implementar los métodos findById ni los métodos CRUD.

Conversor o Formatter (si aplica)

No aplica

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

Validador para establecer una contraseña segura.
@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")

Este validador ha sido implementado con Enrique Moreno, Javier Hidalgo, Pedro Jesús Muñoz y José Francisco Regadera.

Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

No aplica.

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

- **ActividadServiceTests** se encuentra en la ruta \src\test\java\org\springframework\samples\petclinic\service\ActividadServiceTests.java

```
@Test
    @Transactional
    public void shouldInsertActividad() {
      Collection<Actividad> actividades =
this.actividadService.findByNombre("Escalada");
      //No debe existir la actividad con ese nombre en la base de
datos
      int found = actividades.size();
      //Tamaño 0

      System.out.println("==========================================
================");
      System.out.println("Actividad Escalada no encontrada. Found=
"+found);

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

      Actividad actividad = new Actividad();
      actividad.setNombre("Escalada");
      actividad.setDescripcion("Muy buena ruta para realizar con los
amigos y muy facil");
      actividad.setDireccion("Sierra de Grazalema");
      actividad.setProvincia("Sevilla");
      actividad.setValoracion(4);
      actividad.setPrecio(2);

      this.actividadService.saveActividad(actividad);
      System.out.println("assertThat"+actividad.getId());
      assertThat(actividad.getId()).isNotEqualTo(0);

      //Comprobamos que se ha añadido sin problemas
      actividades = this.actividadService.findByNombre("Escalada");
      assertThat(actividades.size()).isEqualTo(found+1);
```

```
      System.out.println("Actividad Escalada encontrada. Found=
"+actividades.size());

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

Esta prueba comprueba que se ha creado una actividad correctamente.

Primero, en la fase de **Arrange** debemos crear la actividad añadiéndole todos los valores válidos a los atributos.
A continuación, en la fase **Act** guardamos la actividad en la base de datos mediante el servicio.
Por último, en la fase **Assert**, comprobamos que el ID de la actividad no sea 0, ya que si es distinto a 0 significa que se ha creado. A continuación, buscamos la actividad por el nombre que se le haya asignado y comprobamos que efectivamente existe en el conjunto.

```
@Test
    @Transactional
    public void shouldInsertActividadVacio() {
      Actividad actividad = new Actividad();
      actividad.setNombre("");
      actividad.setDescripcion("");
      actividad.setDireccion("Sierra de Grazalema");
      actividad.setValoracion(4);
      actividad.setPrecio(5);

         assertThat(actividad.getNombre().isEmpty()).isTrue();
      assertThat(actividad.getDescripcion().isEmpty()).isTrue();

    }
```

En esta prueba comprobamos que la actividad contiene campos vacíos, como son el nombre y la descripción.
Primero, en la fase **Arrange**, creamos la actividad añadiendo los campos nombre y descripción vacíos.
En la fase **Act** guardamos la actividad y luego en la fase **Assert** comprobamos que los atributos que hemos establecido como vacíos están realmente vacíos.

Pruebas unitarias parametrizadas (si aplica)

No aplica

Pruebas de controlador

- **HotelControllerTests** se encuentra en la ruta \src\test\java\org\springframework\samples\petclinic\web\HotelControllerTests.java

```
@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)).willRetu
rn(hotelazo);

      }
```

Este método se va a utilizar para la fase **Arrange**, ya que necesitaremos unos datos iniciales y que se van utilizar durante las pruebas.

- **Escenario positivo**

```
@WithMockUser(value = "spring")
      @Test
      void testProcessFindFormSuccess() throws Exception {

      given(this.hotelService.findByNombre("")).willReturn(Lists.new
ArrayList(hotelazo, new Hotel()));


      mockMvc.perform(get("/hoteles")).andExpect(status().isOk()).an
dExpect(view().name("hoteles/hotelesList"));
      }
```

Esta prueba comprueba que a la hora de buscar un hotel con el parámetro nombre vacío, te devuelve la vista con todos los hoteles que se encuentren disponibles en la aplicación.

En la fase **Act** buscamos el hotel con el nombre vacío y en la fase **Assert** comprobamos que devuelve correctamente a la vista donde se encuentra la lista con todos los hoteles.

- **Escenario negativo**

```
      @WithMockUser(value = "spring")
         @Test
      void testProcessCreationFormHasErrors() throws Exception {
            mockMvc.perform(post("/hoteles/new")
                                       .with(csrf())
                                       .param("nombre", "Joe")
                                       .param("direccion", "123
Prueba Street")
```

```
                                        .param("estrellas", "4")
                                        .param("provincia", "Sevilla")
                                        .param("telefono",
"01316763284325234"))
                        .andExpect(status().isOk())

        .andExpect(model().attributeHasErrors("hotel"))

        .andExpect(model().attributeHasFieldErrors("hotel","telefono")
)

        .andExpect(view().name("hoteles/createOrUpdateHotelForm"));
        }
```

En esta prueba comprobaremos que el formulario de creación de un hotel contiene algún parámetro vacío o con errores, lo cual deberá comprobar que el modelo contiene errores y por lo tanto te devuelve de nuevo al formulario de creación.

En este caso, en la fase **Act** establecemos los parámetros que se utilizarán en el formulario con sus valores, y en la fase **Assert** comprobamos que el parámetro teléfono contiene un error, ya que no se corresponde a lo establecido en el modelo y devuelve a la vista correspondiente.

Principales problemas encontrados

Problemas con las pruebas de controlador para comprobar que el formulario de crear una entidad redirija correctamente, ya que muchas veces me salía un error de que el status era 200 OK y no 300 REDIRECTION y no redirigía correctamente a donde se aplica en el controlador.

Luego, en las pruebas de controlador de errores en el formulario he tenido problemas para realizarlas porque no tenía puesto el atributo correctamente a como se correspondía en el controlador.

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 GitFlow, añadiendo ramas a cada funcionalidad que implementamos.