Nombre: Javier Hidalgo Rodríguez Grupo: G3-14

Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g3-14.git
Nombre de usuario en GitHub: javhidrod8
Rama del usuario en Github: javhidrod1 (Aclaración: No hemos trabajado en nuestras ramas creadas al principio del curso ya que el equipo tomó la decisión de que trabajaríamos utilizando el modelo de ramificación para git, GitFlow)

Participación en el proyecto

Historias de usuario en las que he participado

He implementado completas las historias de usuario HU-3 , HU-4, HU-5, HU-7, HU-8 HU-9, HU-13, HU-14, HU-16, HU-17 y HU-19, además, he desarrollado junto a mi compañero Pedro Pablo Carvajal las historias HU-3 y HU-5, junto a mi compañero Enrique Moreno las historias HU-7, HU-9, HU-16, HU-17, HU-19, junto a mi compañero Pedro Pablo Carvajal y Pedro Jesus Muñoz la historia H8

Además, he implementado las reglas de negocio R2, con mis compañeros Pedro Pablo Carvajal, Enrique Moreno, Jose Francisco Regadera y Pedro Jesus Muñoz, R4, con mi compañero Enrique Moreno y Jose Francisco Regadera, y R5, con mi compañero Enrique Moreno.

Esto hace un total de 11 historias de usuario y 3 reglas de negocio creadas por mi.

Funcionalidad implementada

He implementado los controladores AgenActController, ComentarioActividadController, ComentarioHotelController, HabitacionController, InscripcionHotelController, ReservaActividadController, ReservaVueloController, UserController.
Además, he añadido varios métodos a los servicios AgenActService, ComentarioActividadService, ComentarioHorelService, HabitacionService, InscripcionHotelService, ReservaActividadService, ReservaVueloService, UserService.
Tambien he creado las entidades AgenAct, ComentarioActividad, ComentarioHotel, Habitacion, InscripcionHotel, ReservaActividad, ReservaVuelo, User, y las vistas para agenacts, habitaciones, hoteles, inscripcionhoteles, reservaActividad, reservaVuelo, users, menu.
He implementado los validadores específicos para la contraseña y el dni del usuario, las fechas en las reservas, el numero de tarjeta y el cvc en las reservas que he implementado.
Esto hace un total de 24 clases implementadas por mi y 8 interfaces definidas.

Pruebas implementadas

Pruebas unitarias

He creado tests unitarios para 1 servicio (UserService) y 3 controladores (HabitacionController, AgenActController, CompVueloController). Eso hace un total de 4 clases de test unitarios con un total de 23 métodos anotados con @Test.

Pruebas de Controlador

He creado 1 caso de prueba positivo y 1 negativo1 de controlador para la inscripción de habitaciones, 2 casos de prueba positivos y 2 negativos para la creación y edición de agencAct, 2 casos de prueba positivos y 2 negativos para la creación y edición de compVuelos, 1 caso de prueba positivo para la búsqueda de agenAct y 1 caso de prueba positivo para la búsqueda de compVuelos.

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

Entidad User:

Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/model/User.java

Explicación de responsabilidades de esta entidad: El motivo de esta entidad se encuentra en recoger los datos de cada usuario registrado en la aplicación, así como su historial de reversas realizadas. Además, debe guardar los datos de las actividades, habitaciones y vuelos vinculados a este usuario.

Código: (Omito el código de las importaciones)

```
@Entity
@Table(name = "users")
public class User {
	@Id
	@NotEmpty
	String username;

	@NotEmpty
	@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;

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

	@Column(name = "dni")
	@NotEmpty
	@Pattern(regexp = "^[0-9]{8,8}[A-Za-z]$", message="El DNI debe contener 8
números y una letra mayúscula.")
	private String dni;

	boolean enabled;
```

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

	@ManyToMany(cascade = {
			CascadeType.ALL})
	@JoinTable(
			name = "users_actividades",
			joinColumns = {@JoinColumn(name = "username")},
		inverseJoinColumns = {@JoinColumn(name = "actividades_id")}
			)
	private Set<Actividad> actividades;

	@ManyToMany(cascade = {
			CascadeType.ALL})
	@JoinTable(

			name = "users_vuelos",
			joinColumns = {@JoinColumn(name = "username")},
		inverseJoinColumns = {@JoinColumn(name = "vuelos_id")}
			)
	private Set<Vuelo> vuelos;
	@ManyToMany(cascade = {
			CascadeType.ALL})
	@JoinTable(
			name = "users_habitaciones",
			joinColumns = {@JoinColumn(name = "username")},
		inverseJoinColumns = {@JoinColumn(name = "habitacion_id")}
			)
	private Set<Habitacion> habitaciones;


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

	public Set<ReservaHabitacion> getReservaHabitacion() {
		return reservaHabitacion;
	}

	public void setReservaHabitacion(Set<ReservaHabitacion> reservaHabitacion)
{
		this.reservaHabitacion = reservaHabitacion;
	}


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

	public Set<ReservaVuelo> getReservaVuelo() {
		return reservaVuelo;
	}

	public void setReservaVuelo(Set<ReservaVuelo> reservaVuelo) {
		this.reservaVuelo = reservaVuelo;
	}


	@OneToMany(cascade = CascadeType.ALL, mappedBy = "user")
	private Set<ReservaActividad> reservaActividad;
```

```
public Set<ReservaActividad> getReservaActividad() {
	return reservaActividad;
}

public void setReservaActividad(Set<ReservaActividad> reservaActividad) {
	this.reservaActividad = reservaActividad;
}




public void setActividades(Set<Actividad> actividades) {
	this.actividades = actividades;
}

public void setVuelos(Set<Vuelo> vuelos) {
	this.vuelos = vuelos;
}

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

public String getUsername() {
	return username;
}

public void setUsername(String username) {
	this.username = username;
}

public String getPassword() {
	return password;
}

public void setPassword(String password) {
	this.password = password;
}

public String getTelefono() {
	return telefono;
}

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

public String getDni() {
	return dni;
}

public void setDni(String dni) {
	this.dni = dni;
}

public boolean isEnabled() {
	return enabled;
}
```

```
	public void setEnabled(boolean enabled) {
		this.enabled = enabled;
	}

	public Set<Authorities> getAuthorities() {
		return authorities;
	}

	public void setAuthorities(Set<Authorities> authorities) {
		this.authorities = authorities;
	}


	protected Set<Habitacion> getHabitacionesInternal() {
		if (this.habitaciones == null) {
			this.habitaciones = new HashSet<>();
		}
		return this.habitaciones;
	}

	protected void setHabitacionesInternal(Set<Habitacion> habitaciones) {
		this.habitaciones = habitaciones;
	}

	public List<Habitacion> getHabitaciones() {
		List<Habitacion> sortedHabitaciones = new
ArrayList<>(getHabitacionesInternal());
		PropertyComparator.sort(sortedHabitaciones, new
MutableSortDefinition("nombre", true, true));
		return Collections.unmodifiableList(sortedHabitaciones);
	}

	public void addHabitacion(Habitacion habitacion) {
		getHabitacionesInternal().add(habitacion);
	}

	public boolean removeHabitacion(Habitacion habitacion) {
		return getHabitacionesInternal().remove(habitacion);
	}



	protected Set<Actividad> getActividadesInternal() {
		if (this.actividades == null) {
			this.actividades = new HashSet<>();
		}
		return this.actividades;
	}

	protected void setActividadesInternal(Set<Actividad> actividades) {
		this.actividades = actividades;
	}

	public List<Actividad> getActividades() {
		List<Actividad> sortedActividades = new
ArrayList<>(getActividadesInternal());
		PropertyComparator.sort(sortedActividades, new
MutableSortDefinition("nombre", true, true));
```

```
		return Collections.unmodifiableList(sortedActividades);
	}

	public void addActividad(Actividad actividad) {
		getActividadesInternal().add(actividad);
	}

	public boolean removeActividad(Actividad actividad) {
		return getActividadesInternal().remove(actividad);
	}



	protected Set<Vuelo> getVuelosInternal() {
		if (this.vuelos == null) {
			this.vuelos = new HashSet<>();
		}
		return this.vuelos;
	}

	protected void setVuelosInternal(Set<Vuelo> vuelos) {
		this.vuelos = vuelos;
	}

	public List<Vuelo> getVuelos() {
		List<Vuelo> sortedVuelo = new ArrayList<>(getVuelosInternal());
		PropertyComparator.sort(sortedVuelo, new
MutableSortDefinition("origen", true, true));
		return Collections.unmodifiableList(sortedVuelo);
	}

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

Entidad ComentarioActividad: (En colaboración con Enrique Moreno)

Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/model/ComentarioActividad.java

Explicación de responsabilidades de esta entidad: Esta entidad basa su funcionalidad en la recopilación de los datos sobre los comentarios de los usuarios en las distintas actividades disponibles. Los parámetros a observar son la puntuación que ese usuario le de a la actividad, así como el mensaje que desee poner como comentario.

Código: (Omito el código de las importaciones)

```
@Entity
@Table(name = "comentariosactividad")
public class ComentarioActividad extends BaseEntity{

	@Column(name = "puntuacion")
	@Range(min=1,max=10)
	private Integer puntuacion;

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

	@ManyToOne
	@JoinColumn(name = "actividad_id")
```

```
	private Actividad actividad;

	public Integer getPuntuacion() {
		return puntuacion;
	}

	public void setPuntuacion(Integer puntuacion) {
		this.puntuacion = puntuacion;
	}

	public String getMensaje() {
		return mensaje;
	}

	public void setMensaje(String mensaje) {
		this.mensaje = mensaje;
	}

	public Actividad getActividad() {
		return actividad;
	}

	public void setActividad(Actividad actividad) {
		this.actividad = actividad;
	}

}
```

Servicio

Servicio UserService:

Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/service/UserService.java

Explicación de responsabilidades de este servicio: En este servicio encontramos métodos que sirven para guardar un usuario a la hora de crearlo o editarlo, encontrar un usuario por su username y borrar un usuario mediante la funcionalidad de eliminar mi cuenta.

Código: (Omito el código de las importaciones)

```
@Service
public class UserService {

	private UserRepository userRepository;

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

	@Transactional
	public void saveUser(User user) throws DataAccessException {
		user.setEnabled(true);
		userRepository.save(user);
	}
```

```
	public Optional<User> findUser(String username) {
		return userRepository.findById(username);
	}

	public User findByUsername(String username) {
		return userRepository.findByUsernameLike(username);
	}

	@Transactional
	public void deleteUser(User user) throws DataAccessException {
		userRepository.delete(user);
	}
}
```

Controlador

Controlador UserController:

Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/web/UserController.java

Explicación de responsabilidades de este controlador: En este controlador tratamos el procesamiento de la creación y edición de un usuario, redireccionando a las vistas correspondientes una vez creados, así como el borrado de un usuario de nuestra aplicación y el tratamiento del historial de reservas de ese usuario.

Código: (Omito el código de las importaciones)

```
@Controller
public class UserController {

	private static final String VIEWS_USER_CREATE_OR_UPDATE_FORM =
"users/createOrUpdateUserForm";
	private static final String VIEWS_USER_DELETE_EXITO= "users/exitoDelete";
	private static final String VIEW_USER_HISTORIAL = "users/historial";
	private final UserService userService;
	private final ReservaActividadService reservaActividadService;
	private final ReservaVueloService reservaVueloService;
	private final ReservaHabitacionService reservaHabitacionService;

	@Autowired
	public UserController(UserService userservice,ReservaActividadService
reservaActividadService,ReservaHabitacionService
reservaHabitacionService,ReservaVueloService reservaVueloService) {
		this.userService = userservice;
		this.reservaActividadService = reservaActividadService;
		this.reservaVueloService = reservaVueloService;
		this.reservaHabitacionService = reservaHabitacionService;

	}

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

	@GetMapping(value = "/users/new")
	public String initCreationForm(Map<String, Object> model) {
		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,
Map<String, Object> model) {
			if (result.hasErrors()) {
				model.put("user", user);
				return VIEWS_USER_CREATE_OR_UPDATE_FORM;
			}
			else {
				this.userService.saveUser(user);
				return "redirect:/users/"+user.getUsername();
			}
		}

		@GetMapping(value = "/users/{username}/edit")
		public String initUpdateForm(@PathVariable("username") String username,
ModelMap model) {
			User user = this.userService.findByUsername(username);
			model.put("user", user);
			return VIEWS_USER_CREATE_OR_UPDATE_FORM;
		}

		@PostMapping(value = "/users/{username}/edit")
		public String processUpdateUserForm(@Valid User user, BindingResult result,
				@PathVariable("username") String username, ModelMap model) {
			if (result.hasErrors()) {
				model.put("user", user);
				return VIEWS_USER_CREATE_OR_UPDATE_FORM;
			}
			else {
				if (user.getUsername().equals(username)) {
					user.setUsername(user.getUsername());
				}else {
					user.setUsername(username);
				}
				this.userService.saveUser(user);
				return "redirect:/users/{username}";
			}
		}

		@GetMapping("/users/{username}")
		public ModelAndView showUser(@PathVariable("username") String username) {
			ModelAndView mav = new ModelAndView("users/userDetails");
			mav.addObject("user", this.userService.findByUsername(username));
			return mav;
		}

		@RequestMapping(value = "/users/{username}/delete")
	    public String deleteUser(@PathVariable("username") final String username,
final ModelMap model) {
	        User user= this.userService.findByUsername(username);
	        this.userService.deleteUser(user);
	        return VIEWS_USER_DELETE_EXITO;
	    }

		@RequestMapping(value = "/users/{username}/historial")
```

```
    public String historialUser(@PathVariable("username") final String username,
Map<String, Object> model) {
        Collection<ReservaHabitacion> hh =
this.reservaHabitacionService.buscarReservaHabitacion(username);
       model.put("selectionsHabitacion", hh);
        Collection<ReservaActividad> ha =
this.reservaActividadService.buscarReservaActividad(username);
        model.put("selectionsActividad", ha);
        Collection<ReservaVuelo> hv =
this.reservaVueloService.buscarReservaVuelo(username);
        model.put("selectionsVuelo", hv);
        return VIEW_USER_HISTORIAL;
    }
}
```

Repositorio

Repositorio UserRepository:

Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/repository/UserRepository.java

Explicación de responsabilidades de este repositorio: Este repositorio sirve para encontrar un usuario de la base de datos cuyo parámetro username concuerde con el username que le demos por parámetro.

Código: (Omito el código de las importaciones)

```
public interface UserRepository extends  JpaRepository<User, String>{

	@Query("select u from User u where u.username like %?1")
	User findByUsernameLike(String username);

}
```

Conversor o Formatter

No aplica

Validador y anotación asociada

Validación de contraseña de un usuario: (Este validador fue implementado junto a todo el grupo de trabajo, excepto Alvaro Sevilla)

Utilizo una expresión regular para comprobar que la contraseña del usuario cumpla con los criterios establecidos en la restricción que ideamos.

```
	@NotEmpty
	@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;
```

Validación de DNI de un usuario:

Utilizo una expresión regular para comprobar que el DNI del usuario concuerda con la definición de un número de DNI válido.

```
@Column(name = "dni")
	@NotEmpty
	@Pattern(regexp = "^[0-9]{8,8}[A-Za-z]$", message="El DNI debe contener 8
números y una letra mayúscula.")
	private String dni;
```

Ejemplos de pruebas implementadas

Pruebas unitarias

Prueba shouldInsertUser: (En colaboración con Jose Francisco Regadera)
Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/repository/UserRepository.java
Explicación de esta prueba: En esta prueba unitaria vamos a probar que se inserte correctamente un usuario dentro de la base de datos con el método saveUser de nuestro servicio UserService.
Código:

```
	@Test
	@Transactional
	public void shouldInsertUser() {

//Esta parte corresponde al Arrange/Fixture

		User usuarios = this.userService.findByUsername("pruebadorInsert");

		User user=new User();
			user.setUsername("pruebador");
			user.setPassword("Contrasen4");
			user.setTelefono("645985985");
			user.setDni("20061859V");
		String found = user.getDni();

//Esta parte corresponde al Act

		this.userService.saveUser(user);

//Esta parte corresponde al Assert

		//Comprobamos que se ha añadido sin problemas
		usuarios = this.userService.findByUsername("pruebador");
		assertThat(usuarios.getDni()).isEqualTo(found);
	}
```

Prueba shouldDeleteUser: (En colaboración con Jose Francisco Regadera)
Ruta del fichero: /dp1-2020-g3-14/src/main/java/org/springframework/samples/petclinic/repository/UserRepository.java
Explicación de esta prueba: En esta prueba unitaria vamos a probar que se borre correctamente un usuario de nuestra aplicación gracias al método deleteUser de nuestro servicio UserService.
Código:

```
	@Test
	@Transactional
	public void shouldDeleteUser() {

//Esta parte corresponde al Arrange/Fixture

		User usuarios = this.userService.findByUsername("pruebadorDelete");
		User user=new User();
```

```
                    user.setUsername("pruebador");
                    user.setPassword("Contrasen4");
                    user.setTelefono("645985985");
                    user.setDni("20061859V");

//Esta parte corresponde al Act

                    this.userService.saveUser(user);
            this.userService.deleteUser(user);

//Esta parte corresponde al Assert

            usuarios = this.userService.findByUsername("pruebador");

            assertThat(usuarios).isNull();
        }

}
```

Pruebas unitarias parametrizadas

No aplica

Pruebas de controlador

Prueba testProccessCreationFormSuccess:

Ruta del fichero: /dp1-2020-g3-14/src/test/java/org/springframework/samples/petclinic/web/HabitacionControllerTests.java

Explicación de esta prueba: En esta prueba unitaria vamos a probar que se cree correctamente una habitación dentro de un hotel. Para ello, comprobamos que tras introducir unos parámetros correctos dentro del formulario se nos redirecciona a la vista del hotel en el que hemos creado esta habitación.

Código:

```
//Esta parte corresponde al Arrange/Fixture

void setup() {

            habitacion = new Habitacion();
            habitacion.setNhabitacion(TEST_HABITACION_ID);
            habitacion.setNcamas(2);
            habitacion.setPrecio(20);
            habitacion.setDisponible(true);

            //Deberá devolver la habitacion

      given(this.habitacionService.findHabitacionByNhabitacion(TEST_HABITACION_ID
)).willReturn(habitacion);

      }

//Esta parte corresponde al Act

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

```
	mockMvc.perform(post("/hoteles/1/habitaciones/new").param("nhabitacion",
"666")
                                        .with(csrf())
                                        .param("ncamas", "1")
                                        .param("precio", "60")
                                        .param("disponible", "true"))

//Esta parte corresponde al Assert

                        .andExpect(status().is3xxRedirection());
	}
```

Prueba testProccessCreationFormHasError:
Ruta del fichero: /dp1-2020-g3-14/src/test/java/org/springframework/samples/petclinic/web/HabitacionControllerTests.java
Explicación de esta prueba: En esta prueba unitaria vamos a probar que, tras introducir unos parámetros erróneos para la creación de una habitación, nuestra aplicación nos manda de nuevo al formulario de creación. Además, comprobamos que se nos indica que el parámetro incorrecto tiene errores.
Código:

```
//Esta parte corresponde al Arrange/Fixture

void setup() {

		habitacion = new Habitacion();
		habitacion.setNhabitacion(TEST_HABITACION_ID);
		habitacion.setNcamas(2);
		habitacion.setPrecio(20);
		habitacion.setDisponible(true);

		//Deberá devolver la habitacion

	given(this.habitacionService.findHabitacionByNhabitacion(TEST_HABITACION_ID
)).willReturn(habitacion);

	}

//Esta parte corresponde al Act

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

	mockMvc.perform(post("/hoteles/2/habitaciones/new").param("nhabitacion",
"666")
                                        .with(csrf())
                                        .param("ncamas", "78")
                                        .param("precio", "60"))

//Esta parte corresponde al Assert

	.andExpect(status().isOk())

	.andExpect(model().attributeHasErrors("habitacion"))

	.andExpect(view().name("habitaciones/createOrUpdateHabitacionForm"));
	}
```

Principales problemas encontrados

Los principales problemas que he encontrado han sido a la hora de la creación de comentarios para las actividades y los hoteles, y a la hora de la creación de reservas para los distintos servicios que ofrecemos en la aplicación. El problema que encontré se basaba en que no conseguía diferenciar de qué entidad provenía un comentario o una reserva. Mi compañero Enrique Moreno y yo ideamos una solución para ambos temas que consistía en separar los comentarios de actividad y hoteles en dos entidades distintas, y trabajar con ellas más tarde para su creación. Igualmente, para las reservas, separamos en 3 entidades distintas las reservas para hoteles, vuelos y actividades.

Otros comentarios

No aplica