Nombre: Pedro Jesús Muñoz Cifuentes 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: Pedmuncif y Pedro*
Rama del usuario en Github: pedmuncif (Explicado al final del documento)*

Participación en el proyecto

Historias de usuario en las que he participado

· He implementado completas las historias de usuario HU-6 y HU-8, estas historias las he desarrollado junto a mi compañero Pedro Pablo Carvajal Moreno.

· He implementado completas las historias de usuario HU-12 y HU-20, estas historias las he desarrollado junto a mi compañero José Francisco Regadera Mejías.

· He implementado completas las historias de usuario HU-2 y HU-10, estas historias las he desarrollado junto a mi Francisco Regadera Mejías y Enrique Moreno Vázquez.

Reglas de negocio

· He desarrollado la regla de negocio 3 junto a mi compañero Francisco Regadera Mejías y la 2 con todos mis compañeros.

Funcionalidad implementada

· He implementado el controlador ActividadController, AgenActController, InscripcionHotelController, VueloController, CompVuelosController y he añadido varios métodos al servicio ActividadService, AgenActService, InscripcionHotelService, VueloService y CompVuelosService. También he creado la entidad CompVuelos, Habitacion y las vistas agenacts, compvuelos, inscripcionhoteles y vuelos.

· He implementado un formatter para el tipo LocalDate

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

Esto hace un total de 12 clases implementadas por mí y 6 interfaces definidos.

Pruebas implementadas

Pruebas unitarias

·He creado tests unitarios para 3 servicios (ReservaActividadService, ReservaVueloService, ReservaHotelService), 5 controladores (UserController, ReservaActividadController, ReservaHotelController, ReservaVueloController, VueloController), 0 repositorios , 0

validadores y 3 formatters. Eso hace un total de 8 clases de test unitarios con un total de 39 métodos anotados con @Test.

Pruebas de Controlador

He creado 5 casos de prueba positivo y 5 negativos de controlador para la HU-4, HU-6, HU-15, HU-16, y HU-17 casos positivo para la HU-4, HU-6, HU-15, HU-16, y HU-17.

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

COMPAÑÍA DE VUELOS(COMPVUELOS).

· RUTA: src\main\java\org\springframework\samples\petclinic\model\CompVuelos.java

· La responsabilidad de esta clase es persistir nuestro modelo de dominio a lenguaje java para así tener todas las entidades de nuestro modelo de dominio en el proyecto.

· Aparte de definir los atributos hacemos que compvuelos tenga relación con la entidad vuelos.

```
@Entity
@Table(name = “compvuelos”)
public class CompVuelos extends BaseEntity {

	//Atributos
	@Column(name = “nombre”)
	@NotEmpty
	private String nombre;

	@Column(name = “sede”)
	@NotEmpty
	private String sede;

	@Column(name = “2ais”)
	@NotEmpty
	private String 2ais;

	//Relación con la entidad Vuelos.
	@OneToMany(cascade = CascadeType.ALL, mappedBy = “compVuelo”)
	private Set<Vuelo> vuelos;

	public Set<Vuelo> getVuelos() {
		return vuelos;
	}

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

	public String getNombre() {
		return nombre;
	}

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

```
	}

	public String getSede() {
		return sede;
	}

	public void setSede(String sede) {
		this.sede = sede;
	}

	public String getPais() {
		return 3aís;
	}

	public void setPais(String 3aís) {
		this.pais = 3aís;
	}

	@Override
	public String toString() {
		return "CompVuelos [nombre=" + nombre + ", sede=" + sede + ", 3aís="
+ 3aís + "]";
	}

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

Servicio

COMPAÑÍA DE VUELOS(COMPVUELOSSERVICE)

· Ruta:
src\main\java\org\springframework\samples\petclinic\service\CompVuelosService.java

·La responsabilidad de esta clase es crear métodos que serán nutridos por el repositorio, donde se accederían a los datos.
Estos métodos serán usados en los controladores.

· @Autowired para hacer la inyección de dependencia.

Explicación breve de cada método en el propio código.

```
@Service
public class CompVuelosService {

	private CompVuelosRepository compVuelosRepository;

	@Autowired
	public CompVuelosService(CompVuelosRepository compVuelosRepository) {
		this.compVuelosRepository = compVuelosRepository;
	}
	//Encontrar compañía de vuelos por el ID
	@Transactional(readOnly = true)
	public CompVuelos findCompVuelosById(int id) throws DataAccessException {
		return compVuelosRepository.findById(id);
	}
	 //Guardar una compañía de vuelos
```

```
	@Transactional
	DataAccessException {
		compVuelosRepository.save(compVuelos);
	}
	 //Encontrar una compañía de vuelos por el nombre.
	@Transactional(readOnly = true)
	public Collection<CompVuelos> findByNombre(String name) throws
	DataAccessException {
		return compVuelosRepository.findByNombreLike(name);
	}
```

VUELOSERVICE

· RUTA: src\main\java\org\springframework\samples\petclinic\service\VueloService.java

· La responsabilidad de esta clase es la creación de métodos para acceder a la base de datos mediante el repositorio. También se ha implementado reglas de negocio(validaciones semánticas).
Estos métodos serán usados en el controlador VueloController.

· Explicación de los métodos mediante comentarios en el código.

```
@Service
public class VueloService {

	private VuelosRepository vueloRepository;

	@Autowired
	public VueloService(VuelosRepository vuelosRepository) {
		this.vueloRepository = vuelosRepository;
	}
	//Encontrar un vuelo por el ID
	@Transactional(readOnly = true)
	public Vuelo findVueloById(int id) throws DataAccessException {
		return vueloRepository.findById(id);
	}
	 //Guardar un vuelo
	@Transactional
	public void saveVuelo(Vuelo vuelo) throws DataAccessException {
		vueloRepository.save(vuelo);
	}
	 //Encontrar un vuelo por el origen
	@Transactional(readOnly = true)
	public Collection<Vuelo> findByOrigen(String origen) throws
DataAccessException {
		return vueloRepository.findByOrigenLike(origen);
	}
	//Encontrar un vuelo por el destino
	@Transactional(readOnly = true)
	public Collection<Vuelo> findByDestino(String destino) throws
DataAccessException {
		return vueloRepository.findByDestinoLike(destino);
	}
	//Muestra todos los vuelos
	@Transactional(readOnly = true)
	public List<Vuelo> findAllDestinos() throws DataAccessException {
		return vueloRepository.findAllDestinos();
```

```
	}
```

Controlador

AGENACTCONTROLLER

RUTA: src\main\java\org\springframework\samples\petclinic\web\AgenActController.java

· Explicación: En este controlador podemos ver una serie de métodos que nos permiten manejar las peticiones HTTP. Mediante @Autowired realizamos las inyecciones de dependencia ya que necesitamos el servicio de los vuelos.
Explicación de los métodos en ellos mediante comentarios.

· La responsabilidad de esta clase es la de manejar las peticiones HTTP y redirigir a sus vistas correspondientes.

```
@Controller
public class AgenActController {

	private static final String VIEWS_AGENACTS_CREATE_OR_UPDATE_FORM =
"agenacts/createOrUpdateAgenActForm";
	private final AgenActService agenActService;

	@Autowired
	public AgenActController(AgenActService agenActService) {
			this.agenActService = agenActService;
	}
	@InitBinder
	public void setAllowedFields(WebDataBinder dataBinder) {
		dataBinder.setDisallowedFields("id");
	}

	//Recibe HTTP(GET). Inicia la creación del objeto.
	@GetMapping(value = "/agenacts/new")
	public String initCreationForm(Map<String, Object> model) {
		AgenAct agenAct = new AgenAct();
		model.put("agenact", agenAct);
		return VIEWS_AGENACTS_CREATE_OR_UPDATE_FORM;
	}

	//Recibe HTTP(POST). Si el modelo no tiene errores, usa el servicio para
	guardarlo en la base de datos, si tiene errores te reenvía al formulario de
	creación.
	@PostMapping(value = "/agenacts/new")
	public String processCreationForm(@Valid AgenAct agenAct, BindingResult
	result, Map<String, Object> model) {
		if (result.hasErrors()) {
		model.put("agenact",agenAct);
		return VIEWS_AGENACTS_CREATE_OR_UPDATE_FORM;
	  }
	  else {
	      this.agenActService.saveAgenAct(agenAct);
	      return "redirect:/agenacts/"+agenAct.getId();
	  }
    }
```

```
	//Obtiene el objeto que va a ser modificado a través del servicio y te lo
	reenvía a la vista. Obtener un formulario para editar una agencia de
	actividades por el id.
	@GetMapping(value = "/agenacts/{agenactId}/edit")
	public String initUpdateForm(@PathVariable("agenactId") int agenActId,
	ModelMap model) {
		AgenAct agenAct = this.agenActService.findAgenActById(agenActId);
		model.put("agenact",agenAct);
		return VIEWS_AGENACTS_CREATE_OR_UPDATE_FORM;
	}

	//En el momento que ya se ha editado comprueba que tiene errores y se ha
	editado bien, si está bien te redirige a la vista de los detalles de la
	agencia. Si tiene error te reenvía a la vista anterior.
	@PostMapping(value = "/agenacts/{agenactId}/edit")
	public String processUpdateAgenActForm(@Valid AgenAct agenAct,
	BindingResult result,
			@PathVariable("agenactId") int agenactId,ModelMap model) {
		if (result.hasErrors()) {
			model.put("agenact",agenAct);
			return VIEWS_AGENACTS_CREATE_OR_UPDATE_FORM;
		}
		else {
			agenAct.setId(agenactId);
			this.agenActService.saveAgenAct(agenAct);
			return "redirect:/agenacts/{agenactId}";
		}
	}

	@GetMapping(value = "/agenacts/find")
	public String initFindForm(Map<String, Object> model) {
		model.put("agenact", new AgenAct());
		return "agenacts/findAgenActs";
	}

		//Hemos desarrollado un filtro para obtener las agencias por el
	nombre, si está vacío las muestra todas.
	@GetMapping(value = "/agenacts")
	public String processFindForm(AgenAct agenact, BindingResult result,
	Map<String, Object> model) {

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

		Collection<AgenAct> results =
		this.agenActService.findByNombre(agenact.getNombre());
		if (results.isEmpty()) {
			result.rejectValue("nombre", "notFound", "not found");
			return "agenacts/findAgenActs";
		}
		else if (results.size() == 1) {
			agenact = results.iterator().next();
			return "redirect:/agenacts/" + agenact.getId();
		}
		else {
			model.put("selections", results);
			return "agenacts/agenActsList";
```

```
		}
	}
	//Nos enseña la agencia que le entra por la url mediante agenactId.
	@GetMapping("/agenacts/{agenactId}")
	public ModelAndView showAgenAct(@PathVariable("agenactId") int agenactId) {
		ModelAndView mav = new ModelAndView("agenacts/agenactDetails");
			mav.addObject("agenact",
	this.agenActService.findAgenActById(agenactId));
		return mav;
	}
```

Repositorio

VUELOSREPOSITORY

RUTA:
src\main\java\org\springframework\samples\petclinic\repository\VuelosRepository.java

· La responsabilidad de esta clase es la obtención de datos de la base mediante sentencias JPQL. Vamos a pasar a explicar cada sentencia arriba de los métodos mediante comentarios.

```
public interface VuelosRepository extends JpaRepository<Vuelo, Integer>  {

	 //Seleccionar un vuelo con origen igual al que se pasa por parametro
	@Query("select v from Vuelo v where v.origen like %?1")
	public Collection<Vuelo> findByOrigenLike(String origen);

	 //Seleccionar un vuelo con destino igual al que se pasa por parametro
	@Query("select v from Vuelo v where v.destino like %?1")
	public Collection<Vuelo> findByDestinoLike(String destino);

	 //Todos los vuelos
	@Query("select v from Vuelo v")
	public List<Vuelo> findAllDestinos();

	Vuelo findById(int id) throws DataAccessException;
}
```

Conversor o Formatter (si aplica)

==========

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

· Hemos realizado entre todos los componentes este patrón para las contraseñas.

@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")

Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

=======

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

(====================NOMBRE====================)
Delimita el fin.

RESERVAHABITACIONSERVICETEST.

RUTA:
src\test\java\org\springframework\samples\petclinic\service\ReservaHabitacionServiceTests.java

La prueba se ha realizado con mi compañero Enrique Moreno Vázquez.

```
@Test
	void shouldFindReservaHabitacionByName() {

		ReservaHabitacion reservaHabitacion = new ReservaHabitacion();
		reservaHabitacion.setFechaReserva(LocalDate.now());
		reservaHabitacion.setEntrada(LocalDate.of(2021, 3, 3));
		reservaHabitacion.setSalida(LocalDate.of(2021, 3, 22));
		reservaHabitacion.setNumeroTarjeta("371449635398431");
		reservaHabitacion.setCvc("123");
		reservaHabitacion.setPrecioFinal(20.0);
		//Creamos usuario
		User usuario = new User();
		usuario.setUsername("enrmorvaz");
		reservaHabitacion.setUser(usuario);
		//Creamos hotel
		Hotel hotel = new Hotel();
		hotel.setNombre("HOTEL 2");
		hotel.setDireccion("Calle Cano");
		hotel.setEstrellas(2);
		hotel.setProvincia("Sevilla");
		hotel.setTelefono("322222222");

		//Creamos habitacion
		Habitacion habitacion = new Habitacion();
		habitacion.setNhabitacion(123);
		habitacion.setPrecio(23);
		habitacion.setDisponible(true);
		habitacion.setNcamas(2);
		habitacion.setHotel(hotel);

		//Creamos un set añadimos la habitacion
		Set<Habitacion> habitaciones= new HashSet<Habitacion>();
		habitaciones.add(habitacion);
		hotel.setHabitaciones(habitaciones);
		//Añadimos la habitacion
		reservaHabitacion.setHabitacion(habitacion);
```

· **Arrange/Fixture**: Creamos una nueva reserva. Para ello debemos crear un usuario, una nueva habitación y un nuevo hotel. Para la creación establecemos unos valores correctos.

==========**ARRANGE/FIXTURE**==========

```
	habitacionService.saveHabitacion(habitacion);
	hotelService.saveHotel(hotel);
	this.reservaHabitacionService.saveReservaHabitacion(reservaHabitacion);
		Collection<ReservaHabitacion> reservaVuelos =
	this.reservaHabitacionService.buscarReservaHabitacion("enrmorvaz");
```

· **Act**: Una vez establecido todo realizamos la operación de guardar la habitación, el hotel y la reserva. Creamos la reserva con ID enrmorvaz.

==========**ACT**==========

```
		assertThat(reservaVuelos.size()).isEqualTo(1);
	}
```

· **Assert**: Comprobamos que el conjunto de las reservas tiene tamaño 1, que en este caso sería enrmorvaz

==========**ASSERT**==========

ReservaVueloServiceTest

Ruta:
src\test\java\org\springframework\samples\petclinic\service\ReservaVueloServiceTests.java

· La prueba se ha realizado con mi compañero Enrique Moreno Vázquez.

· Queremos comprobar que se puede encontrar una Reserva de vuelo por su nombre.

```
@Test
	void shouldFindReservaVueloByName() {
		Vuelo vuelo = new Vuelo();
		vuelo.setBilletes(451);
		vuelo.setDestino("Cadiz");
		vuelo.setOrigen("Sevilla");
		vuelo.setPrecio(12);
		LocalDate fechaIda=LocalDate.of(2021, 10, 26);
		vuelo.setFechaIda(fechaIda);
		LocalDate fechaVuelta=LocalDate.of(2021, 11, 4);
		vuelo.setFechaIda(fechaVuelta);
		vuelo.setBilletes(3);

		ReservaVuelo reservaVuelo = new ReservaVuelo();
		reservaVuelo.setFechaReserva(LocalDate.now());
		reservaVuelo.setIda(vuelo.getFechaIda());
		reservaVuelo.setVuelta(vuelo.getFechaVuelta());
		reservaVuelo.setNumeroTarjeta("371449635398431");
		reservaVuelo.setCvc("123");
		reservaVuelo.setPrecioFinal(20.0);
		User usuario = new User();
```

```
            usuario.setUsername("enrmorvaz");
            reservaVuelo.setUser(usuario);
```

· **Arrange/Fixture**: Creamos una nueva Reserva de Vuelo, para ello debemos de crear un vuelo y un usuario.

```
==================================ARRANGE/FIXTURE==================================

            this.reservaVueloService.saveReservaVuelo(reservaVuelo);
                        Collection<ReservaVuelo> reservaVuelos =
            this.reservaVueloService.buscarReservaVuelo("enrmorvaz");
```

· **Act**: Una vez establecido todo realizamos la operación de guardar la reserva de vuelo. Creamos un conjunto en el que se añaden las reservas de vuelos por el nombre enrmorvaz.

```
====================================ACT=======================================

            assertThat(reservaVuelos.size()).isEqualTo(1);
      }
```

· **Assert**: Comprobamos que el conjunto de las reservas tiene tamaño 1, que en este caso sería enrmorvaz.

```
==================================ASSERT======================================
```

Pruebas unitarias parametrizadas (si aplica)

Pruebas de controlador

UserControllerTests.

Ruta:
src\test\java\org\springframework\samples\petclinic\web\UserControllerTests.java

ARRANGE/FIXTURE

```
@WebMvcTest(controllers=UserController.class,
excludeFilters = @ComponentScan.Filter(type = FilterType.ASSIGNABLE_TYPE, classes
= WebSecurityConfigurer.class),
excludeAutoConfiguration= SecurityConfiguration.class)
public class UserControllerTests {

      private static final String TEST_USER_ID = "luigi";

      @Autowired
      private UserController userController;

      @MockBean
      private UserService userService;
```

```
    @MockBean
    private ReservaActividadService reservaActividadService;
    @MockBean
    private ReservaHabitacionService reservaHabitacionService;
    @MockBean
    private ReservaVueloService reservaVueloService;

      @Autowired
      private MockMvc mockMvc;

      private User user;

      //Creamos el usuario
      @BeforeEach
      void setup() {

            user = new User();
            user.setUsername("mario");
            user.setPassword("Elpepe013");
            user.setTelefono("956444876");
            user.setDni("44068802R");
            //Deberá devolver el hotel

      given(this.userService.findByUsername(TEST_USER_ID)).willReturn(user);

      }
```

· Arrange/Fixture: Creamos un nuevo usuario con parámetros válidos. Comienza en el comentario del código //Creamos el usuario.

TEST NEGATIVO.

```
@Test
      void testProcessCreationFormHasErrors() throws Exception {
            mockMvc.perform(post("/users/new").param("username", "mario")
                  .with(csrf())
                  .param("password", "Elpepe013")
                  .param("telefono", "")
                  .param("dni", "44068802R"))
```

· Act: Para este caso negativo vamos a utilizar el usuario ya creado con el número de teléfono vacío. Para así comprobar que no nos debería de dejar crear el usuario, ya que ese atributo no puede estar vacío.

```
==============================ACT================================================

                  .andExpect(status().isOk())
                  .andExpect(model().attributeHasErrors("user"))
                  .andExpect(model().attributeHasFieldErrors("user","telefono"))
                  .andExpect(view().name("users/createOrUpdateUserForm"));
      }
```

· Assert: Aquí le indicamos que el modelo user tiene un campo con error, en este caso teléfono para que no nos cree el nuevo usuario y nos reenvíe al mismo formulario para indicar mediante el mensaje de una validación sintáctica que rellene el campo.

```
============================ASSERT=============================================
```

TEST POSITIVO.

```
@Test
	void testProcessUpdateFormSuccess() throws Exception {
		mockMvc.perform(post("/users/{username}/edit", TEST_USER_ID)
				.with(csrf())
				.param("username", "mario")
				.param("password", "Elpepe013")
				.param("telefono", "633897503")
				.param("dni", "44068802R"))
```

·Act: Utilizamos el usuario ya creado anteriormente con todos sus parámetros válidos para comprobar que se crea correctamente.

==========ACT==========

```
				.andExpect(status().is3xxRedirection())
				.andExpect(view().name("redirect:/users/{username}"));
	}
```

·Assert: Comprobamos que lo crea y nos redirige a la página del usuario.

==========ASSERT==========

Principales problemas encontrados

1.Solicitud de inscripción de un hotel:

Corresponde a una decisión de diseño ya que nos enfrentamos al problema yo y Pedro Pablo Carvajal Moreno.
Realizar una solicitud para que un propietario de hotel pueda enviar los datos de su hotel y que además se muestren las distintas solicitudes realizadas por todos los propietarios.

La primera vez que quisimos implementarlo decidimos probar con la entidad hotel, tras probar a crear métodos en el controlador y varios servicios, a la hora de realizar la solicitud de la inscripción, se nos añadía directamente a la lista de hoteles ya inscritos por lo que para solucionar este problema decidimos crear la entidad SolicitudInscripciónHotel.

Tras esta nueva solución se nos hizo mucho más fácil implementar todo lo que queríamos ya que desde el propio controlador de la entidad y servicios pudimos crear los métodos fácilmente.

Así, logramos crear una nueva lista en la que tenemos todas las solicitudes de una inscripción de hoteles.

Como una historia de usuario hemos decidido optar por la alternativa 4.b, porque nos parece más sencillo de implementar, ya que tenemos una entidad solamente para las inscripciones y se encuentra separada de las demás entidades. A la hora de intentar implementar las inscripciones desde la entidad Hotel nos resultó muy tedioso el mostrar las inscripciones separadas de los hoteles, ya que salían en la misma vista y no conseguimos separarlas.

2. Borrado de hoteles con valoración baja.

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

*En mi eclipse, por defecto se me guarda el usuario de mi cuenta que está en mi pc y no me había dado cuenta, por eso varios commits que aparecen como Pedro son míos con diferente nombre.

*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 GitFlow, creando una rama por cada funcionalidad implementada.