UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES
ESCUELA DE INGENIERÍA CIVIL INFORMÁTICA

# Proyecto base PetClinic-rest. Testing

## Ingeniería de Software, 2019-2

Cabe recordar que el proyecto base PetClinic-rest, el cual corresponde a un backend, define clases en cuatro capas principales:

- Rest, donde residen las clases que implementan los controladores en los cuales se definen los servicios rest de la aplicación
- Service, la cual contiene las clases que satisfacen las necesidades de los servicios definidos en la capa Rest. Lo anterior implica que los métodos de las clases controladoras invocan métodos de estas clases service.
- Repository, que contiene las clases que interactúan con la base de datos a través de Spring-Hibernate. Los métodos de estas clases son invocados desde los métodos de las clases service.
- Model, capa que incluye las clases que representan las entidades o beans de datos asociados a las tablas de la base de datos.

La Figura 1 muestra el diagrama de clases correspondiente a la capa Model.

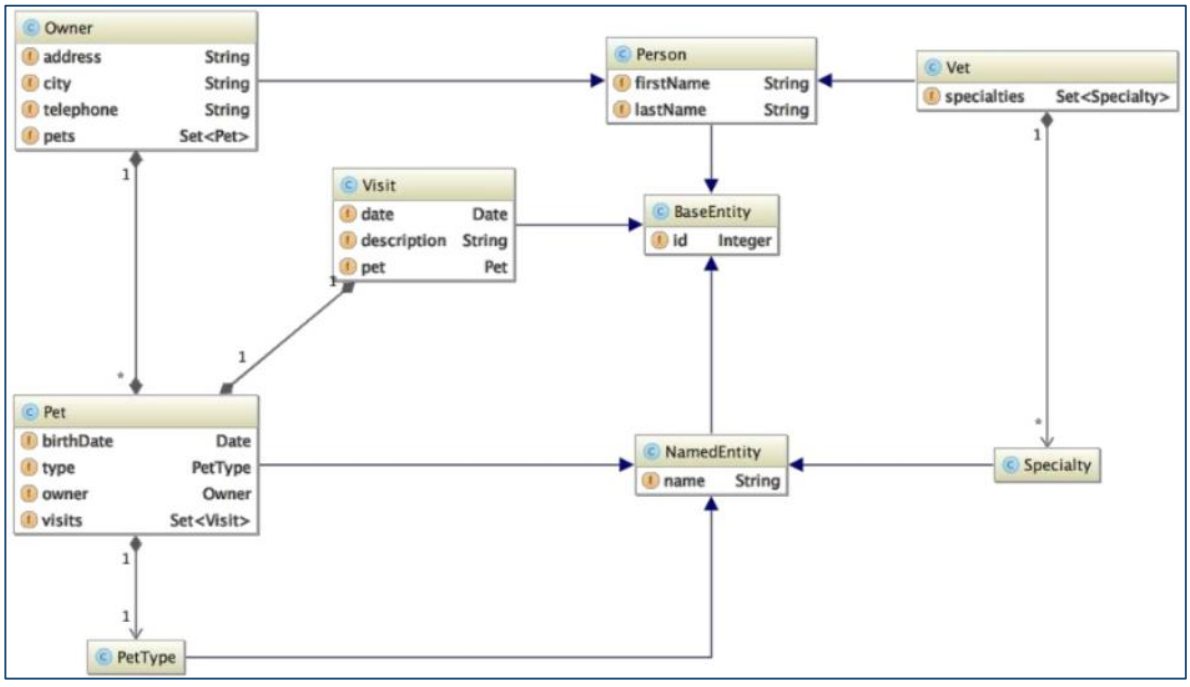


Figura 1. Diagrama clases – Modelo del Dominio

Por otra parte, la estructura del proyecto cuenta con los paquetes de código que incluyen los tests y recursos necesarios para ejecutarlos. Esto se muestra en la Figura 2.

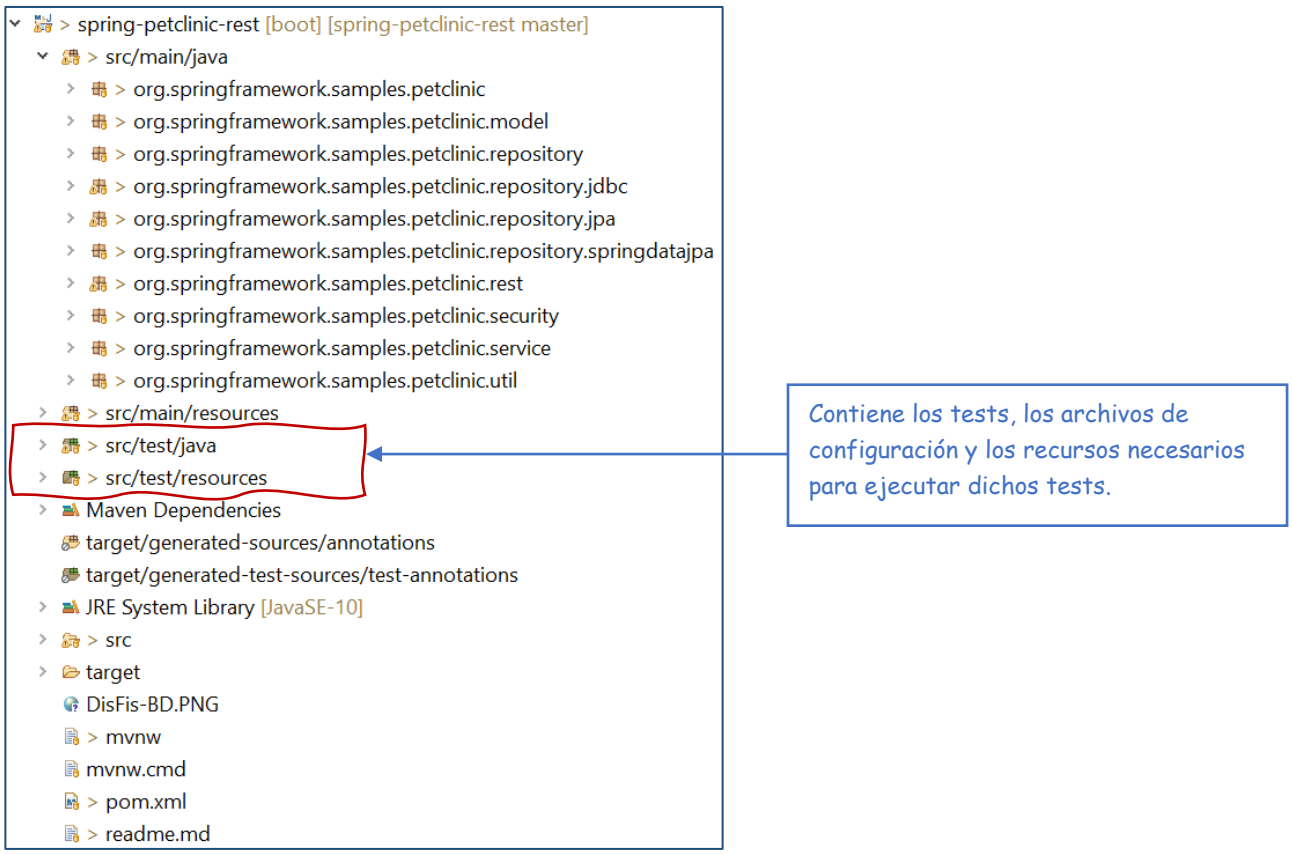


Figura 2. Estructura del proyecto

El paquete de código resources contiene los archivos que preparan la base de datos para realizar pruebas en la capa de servicios. Dichas pruebas son de tipo componente o feature, pues hacen uso de las clases colaboradoras de la capa inferior de persistencia.

La Figura 3 muestra todas las clases que han sido definidas para realizar 169 tests, unitarios y de feature, del proyecto.

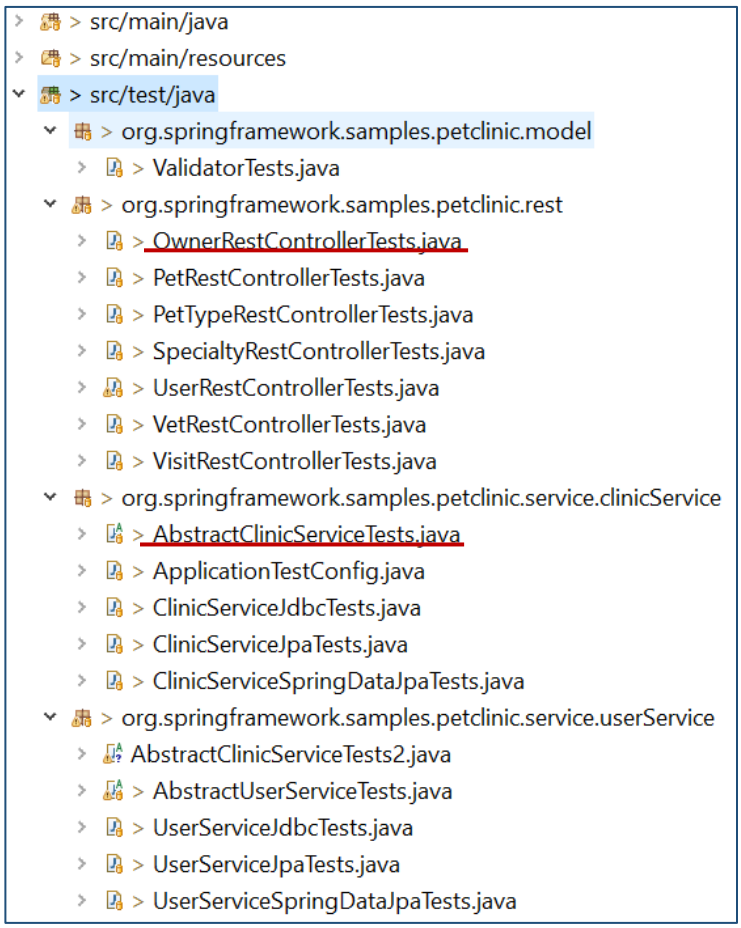


Figura 3. Estructura del proyecto desde la perspectiva del testing

## Programación de tests

A continuación se presentan pruebas automatizadas escritas con JUnit, Mockito y clases de Spring para realizar pruebas, entre ellas MockMvc, que se inlcuyen en el proyecto base petclinic-rest.

Se han omitido las líneas que contienen package e import por razones de espacio.

### Capa Service

La clase que se presenta a continuación, AbstactClinicServiceTest, contiene los tests (de feature) definidos para la clase ClinicService (en realidad ClinicServiceImpl, pues ClinicService es una interface Java) de la capa Service.

**public abstract class AbstractClinicServiceTests {**

```
@Autowired
protected ClinicService clinicService;

@Before
public void init() {
   MockitoAnnotations.initMocks(this);
}


@Test
public void shouldFindOwnersByLastName() {
   // Act (Caso exitoso)
   Collection<Owner> owners = this.clinicService.findOwnerByLastName("Davis");
   // Assert
   assertThat(owners.size()).isEqualTo(2);

  // Act (Caso fallido)
   owners = this.clinicService.findOwnerByLastName ("Daviss");
  // Assert
   assertThat(owners.isEmpty()).isTrue();
}

@Test
public void shouldFindSingleOwnerWithPet() {
   // Act
   Owner owner = this.clinicService.findOwnerById(1);
   // Assert
   assertThat(owner.getLastName()).startsWith("Franklin");
   assertThat(owner.getPets().size()).isEqualTo(1);
   assertThat(owner.getPets().get(0).getType()).isNotNull();
   assertThat(owner.getPets().get(0).getType().getName()).isEqualTo("cat");
}
```

Objeto (SUT) cuyos métodos se probarán, se pide a Spring que cree e inyecte el código en esta clase.

Inicializa mocks de Mockito

Resultado obtenido

Llamada al método que se prueba

Dato del caso de prueba

Resultado esperado

```
@Test
public void shouldFindPetWithCorrectId() {
   // Act
   Pet pet7 = this.clinicService.findPetById(7);
   // Assert
   assertThat(pet7.getName()).startsWith("Samantha");
   assertThat(pet7.getOwner().getFirstName()).isEqualTo("Jean");
}
```

```
@Test
@Transactional
public void shouldAddNewVisitForPet() {
   // Arrange
   Pet pet7 = this.clinicService.findPetById(7);
   int found = pet7.getVisits().size();
   Visit visit = new Visit();
   pet7.addVisit(visit);
   visit.setDescription("test");

   // Act
   this.clinicService.saveVisit(visit);
   this.clinicService.savePet(pet7);

   // Assert
   pet7 = this.clinicService.findPetById(7);
   assertThat(pet7.getVisits().size()).isEqualTo(found + 1);
   assertThat(visit.getId()).isNotNull();
}
```

Eliminará los cambios realizados en el test al finalizar su ejecución

Almacena la nueva visit

Graba la relación entre el pet y la nueva visit

```
@Test
@Transactional
public void shouldDeletePet(){
   // Arrange
   Pet pet = this.clinicService.findPetById(1);
   // Act
   this.clinicService.deletePet(pet);
   // Assert
   try {
        pet = this.clinicService.findPetById(1);
        } catch (Exception e) {
              pet = null;
        }
   assertThat(pet).isNull();
}
```

## Capa Rest (controllers)

Las siguientes son pruebas unitarias, en las que se mockea el colaborador de los controllers, esto es, ClinicService y se utiliza MockMvc de Spring para levantar la aplicación en modo de prueba e invocar los servicios de interés.

```
@SpringBootTest
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=ApplicationTestConfig.class)
@WebAppConfiguration
public class OwnerRestControllerTests {

    @Autowired
    private OwnerRestController ownerRestController;

    @MockBean
    private ClinicService clinicService;

    private MockMvc mockMvc;

    private List<Owner> owners;

    @Before
    public void initOwners(){
        this.mockMvc = MockMvcBuilders.standaloneSetup(ownerRestController)
                .setControllerAdvice(new ExceptionControllerAdvice())
                .build();
        // Crea una lista y la llena de objetos owner (ArrayList que es de tipo Collection)
        owners = new ArrayList<Owner>();

        Owner owner = new Owner();
        owner.setId(1);
        owner.setFirstName("George");
        owner.setLastName("Franklin");
        owner.setAddress("110 W. Liberty St.");
        owner.setCity("Madison");
        owner.setTelephone("6085551023");
        owners.add(owner);

        owner = new Owner();
        owner.setId(2);
        owner.setFirstName("Betty");
        owner.setLastName("Davis");
        owner.setAddress("638 Cardinal Ave.");
        owner.setCity("Sun Prairie");
        owner.setTelephone("6085551749");
        owners.add(owner);
```

Si desea utilizar MockMvc

Código bajo prueba

Indica que la siguiente variable será un mock, pide a Mockito lo cree y lo injecte al SUT

Variable que almacenará un MockMvc que levanta un servidor de aplicaciones simple para tests

Se crea un MockMvc staandalone y se le indica el controlador bajo prueba

```
        owner = new Owner();
        owner.setId(3);
        owner.setFirstName("Eduardo");
        owner.setLastName("Rodriquez");
        owner.setAddress("2693 Commerce St.");
        owner.setCity("McFarland");
        owner.setTelephone("6085558763");
        owners.add(owner);

        owner = new Owner();
        owner.setId(4);
        owner.setFirstName("Harold");
        owner.setLastName("Davis");
        owner.setAddress("563 Friendly St.");
        owner.setCity("Windsor");
        owner.setTelephone("6085553198");
        owners.add(owner);
    }
```

A continuación, algunas de pruebas unitarias definidas en OwnerRestControllerTest, las cuales utilizan un doble de prueba para ClinicService y el doble de prueba mockMvc.

```
@Test
@WithMockUser(roles="OWNER_ADMIN")
public void testGetOwnerSuccess() throws Exception {

    given(this.clinicService.findOwnerById(1)).willReturn(owners.get(0));

    this.mockMvc.perform(get("/api/owners/1")
        .accept(MediaType.APPLICATION_JSON_VALUE))

    .andExpect(status().isOk())
    .andExpect(content().contentType("application/json;charset=UTF-8"))
    .andExpect(jsonPath("$.id").value(1))
    .andExpect(jsonPath("$.firstName").value("George"));
}
```

Establece que se usará un usuario (user) mockedado con el rol indicado, el que tiene acceso al método bajo prueba.

Se programa el comportamiento del doble de prueba clinicService

Invoca el servicio bajo prueba. Retorna un objeto de la clase **ResultActions**

En el test anterior se verifica que:

a) El método get con uri terminada en /api/owners/1 retorno un código correcto (de la serie 200).
b) El contenido de lo retornado se encuentre en notación json y use el set de caracteres UTF-8.
c) El contenido retornado ($ es la denominación del elemento u objeto raíz) posee un dato/atributo llamado id, cuyo valor asociado sea 1.
d) El contenido retornado ($) posea un dato/atributo llamado firstName, cuyo valor asociado sea "George".

```
@Test
@WithMockUser(roles="OWNER_ADMIN")
public void testGetOwnerNotFound() throws Exception {
	given(this.clinicService.findOwnerById(-1)).willReturn(null);

	this.mockMvc.perform(get("/api/owners/-1")
			.accept(MediaType.APPLICATION_JSON))
		.andExpect(status().isNotFound());
}

@Test
@WithMockUser(roles="OWNER_ADMIN")
public void testGetAllOwnersSuccess() throws Exception {
	owners.remove(0);
	owners.remove(1);
	given(this.clinicService.findAllOwners()).willReturn(owners);

	this.mockMvc.perform(get("/api/owners/")
			.accept(MediaType.APPLICATION_JSON))

		.andExpect(status().isOk())
		.andExpect(content().contentType("application/json;charset=UTF-8"))
		.andExpect(jsonPath("$.[0].id").value(2))
		.andExpect(jsonPath("$.[0].firstName").value("Betty"))
		.andExpect(jsonPath("$.[1].id").value(4))
		.andExpect(jsonPath("$.[1].firstName").value("Harold"));
}

@Test
@WithMockUser(roles="OWNER_ADMIN")
public void testGetAllOwnersNotFound() throws Exception {
	owners.clear();
	given(this.clinicService.findAllOwners()).willReturn(owners);

	this.mockMvc.perform(get("/api/owners/")
			.accept(MediaType.APPLICATION_JSON))
		.andExpect(status().isNotFound());
}

@Test
@WithMockUser(roles="OWNER_ADMIN")
public void testCreateOwnerSuccess() throws Exception {
	Owner newOwner = owners.get(0);
	newOwner.setId(999);
	ObjectMapper mapper = new ObjectMapper();
	String newOwnerAsJSON = mapper.writeValueAsString(newOwner);
```

```
        this.mockMvc.perform(post("/api/owners/")
            .content(newOwnerAsJSON).accept(MediaType.APPLICATION_JSON_VALUE)
            .contentType(MediaType.APPLICATION_JSON_VALUE))

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

Apoyo para comprender MockMvc: https://blog.marcnuri.com/mockmvc-introduccion-a-spring-mvc-testing/