Memoria de proyecto Zoo

Nombre: Oscar Gracia Cobo, Sergio Simón, Sergio Misas.
Curso: 1ºDAM

Resumen

Enunciado y requisitos funcionales

Desarrolla una aplicación en JavaFX para gestionar una lista de animales de un zoo, utilizando funciones CRUD (crear, borrar, modificar, leer). La aplicación debe estar conectada a una base de datos que contenga una tabla "Animales" con los siguientes campos: id (auto incremental), nombre, nombre Científico, fecha Nacimiento, ...

Requerimientos funcionales:

1. La aplicación debe permitir al usuario agregar nuevos animales al zoo, ingresando su nombre, especie y edad. El campo "id" debe ser generado automáticamente por la base de datos.
2. El usuario debe poder ver la lista completa de animales existentes, mostrando su nombre, especie, edad y foto asociada.
3. Se debe implementar la funcionalidad de modificar los datos de un animal existente en la lista, permitiendo cambiar su nombre, especie y otros campos mutables (por ejemplo, fecha nacimiento no).
4. La aplicación debe proporcionar una opción para eliminar un animal de la lista.
5. Se requiere la capacidad de exportar la lista completa de animales en formato JSON.
6. La aplicación debe tener la capacidad de importar una lista de animales desde un archivo JSON y agregarlos a la base de datos.
7. Se pide realización de documentos como memoria y diagramas.
8. Añadir una ventana con información sobre ti.
9. Como ampliación opcional, se puede implementar la gestión de fotos asociadas a los animales, permitiendo al usuario cargar y mostrar imágenes relacionadas con cada animal.

Requerimientos no funcionales:

- La interfaz de usuario debe ser intuitiva y fácil de usar, con controles adecuados para realizar las operaciones CRUD.
- La base de datos utilizada debe contar con una tabla con los campos mencionados y la configuración necesaria para generar el id de forma automática.
- Se recomienda utilizar una biblioteca de Java para manipular archivos JSON, como Jackson o Gson, para facilitar la exportación e importación de datos.
- Se valora el uso de buenas prácticas de programación, como la separación de responsabilidades y la implementación de una arquitectura MVC (Modelo-Vista-Controlador) o similar.

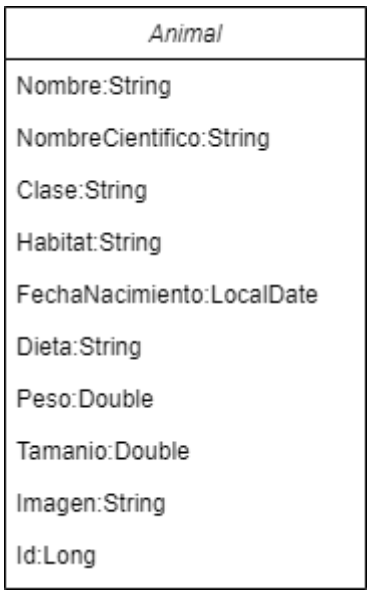
Requisitos de información:

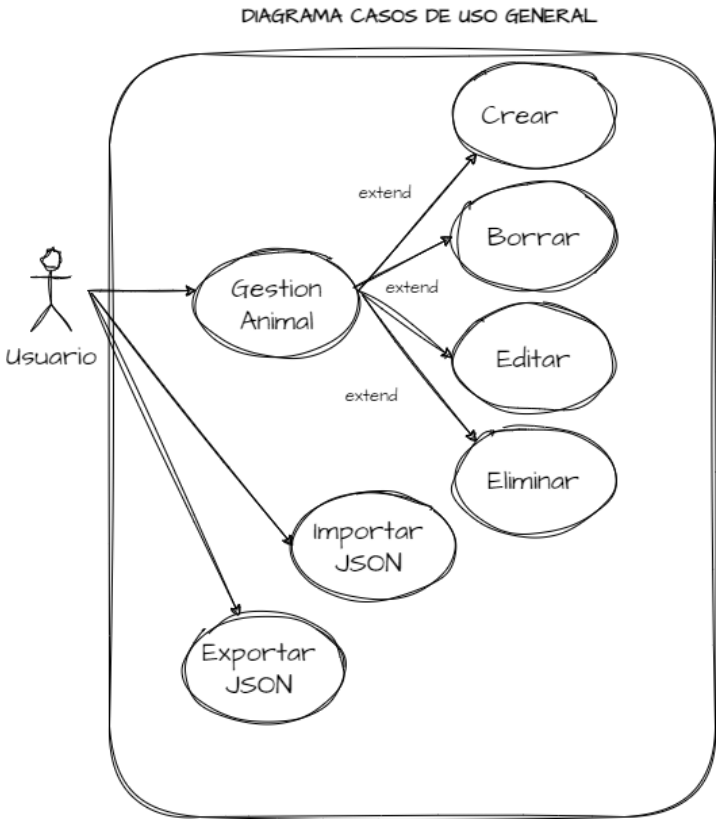
- Campos requeridos para cada animal: nombre, especie, nombre científico, fecha nacimiento, peso, tamaño.

- Los animales deben tener asignado un identificador único (id) generado automáticamente.
- La lista de animales debe incluir la foto asociada a cada animal.
- Los datos deben ser exportados en formato JSON para su posterior importación.

Creación de esquemas

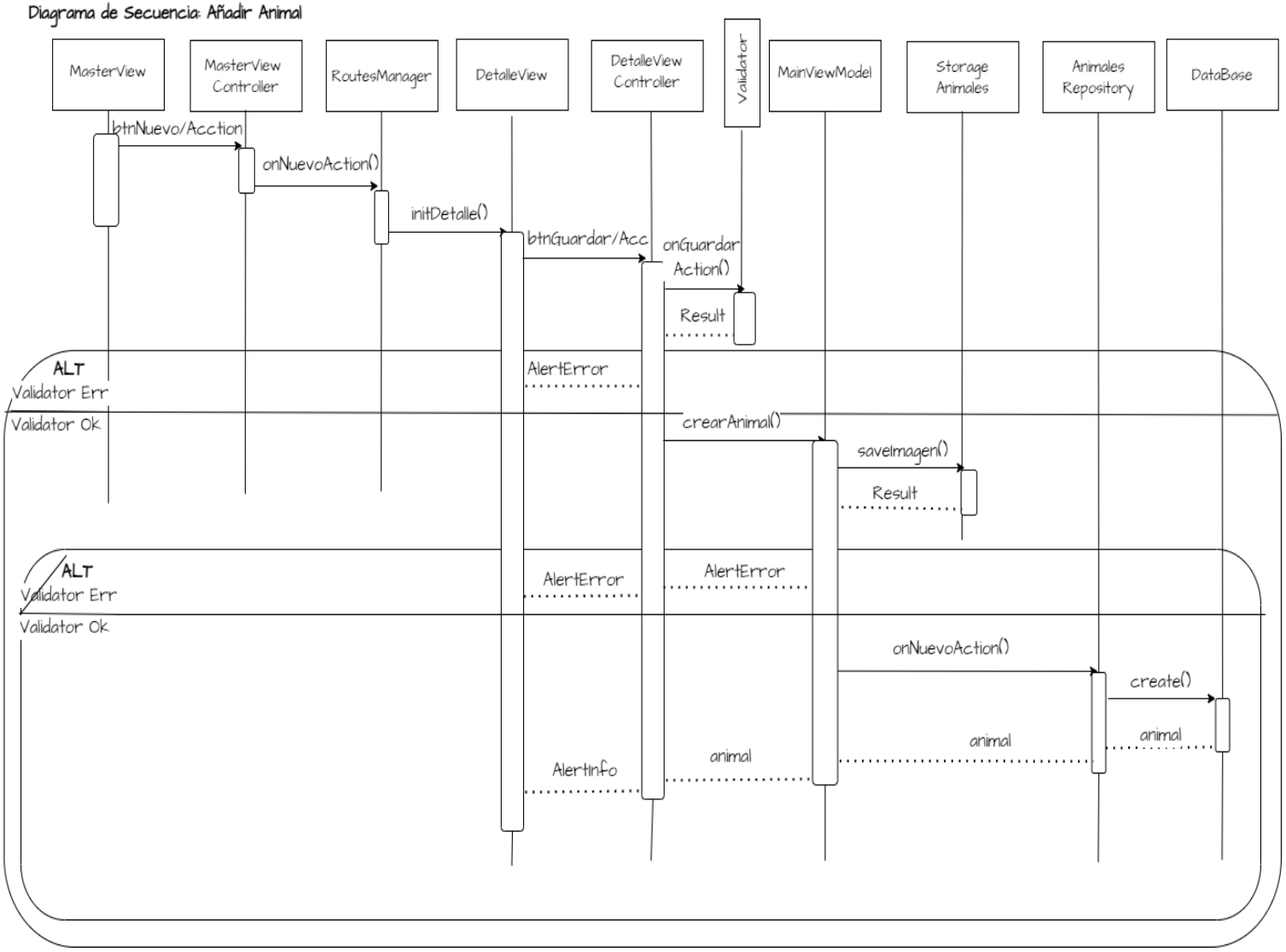
Al tener únicamente que trabajar con los datos de animales. Solo usaremos una tabla en la base de datos, lo cual nos ahorra el hacer un esquema E/R , el esquema de clase será suficiente.





Para compensar la sencillez de estos esquemas realizare todos los esquemas de secuencia, aunque son muy parecidos entre sí.

Diagrama de Secuencia: Añadir Animal



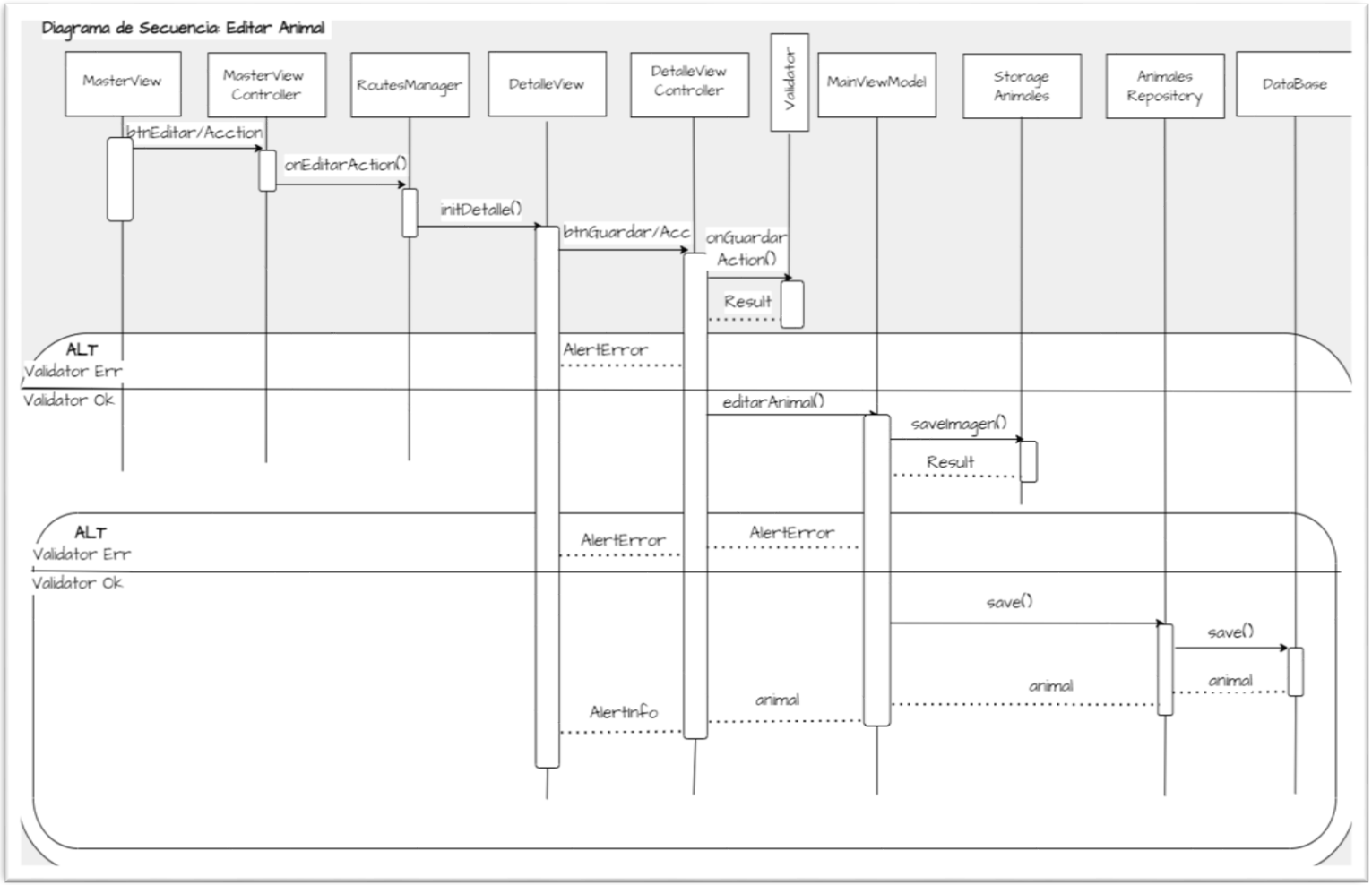


Diagrama de Secuencia: Editar Animal
MasterView
MasterView Controller
RoutesManager
DetalleView
DetalleView Controller
Validator
MainViewModel
Storage Animales
Animales Repository
DataBase
btnEditar/Accion
onEditarAction()
initDetalle()
btnGuardar/Acc
onGuardar Action()
Result
ALT
Validator Err
AlertError
Validator Ok
editarAnimal()
saveImagen()
Result
ALT
Validator Err
AlertError
AlertError
Validator Ok
save()
save()
animal
animal
AlertInfo
animal

Diagrama de Secuencia: Borrar Animal

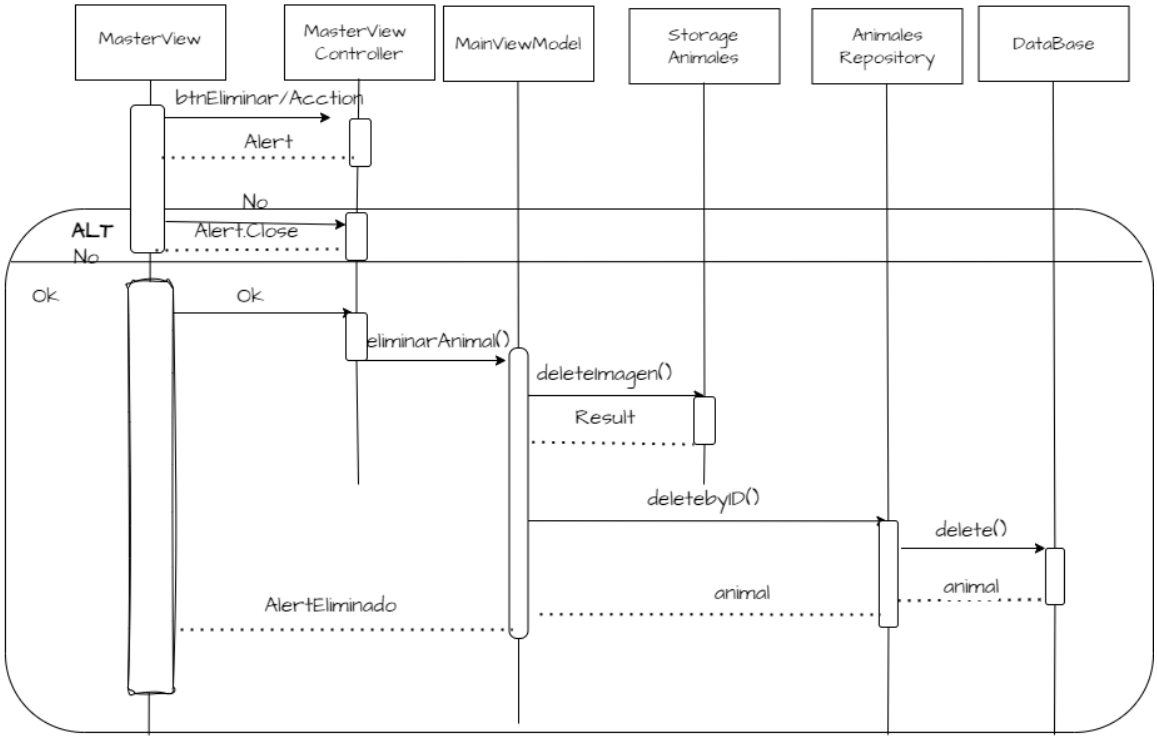


Diagrama de Secuencia: Exportar JSON

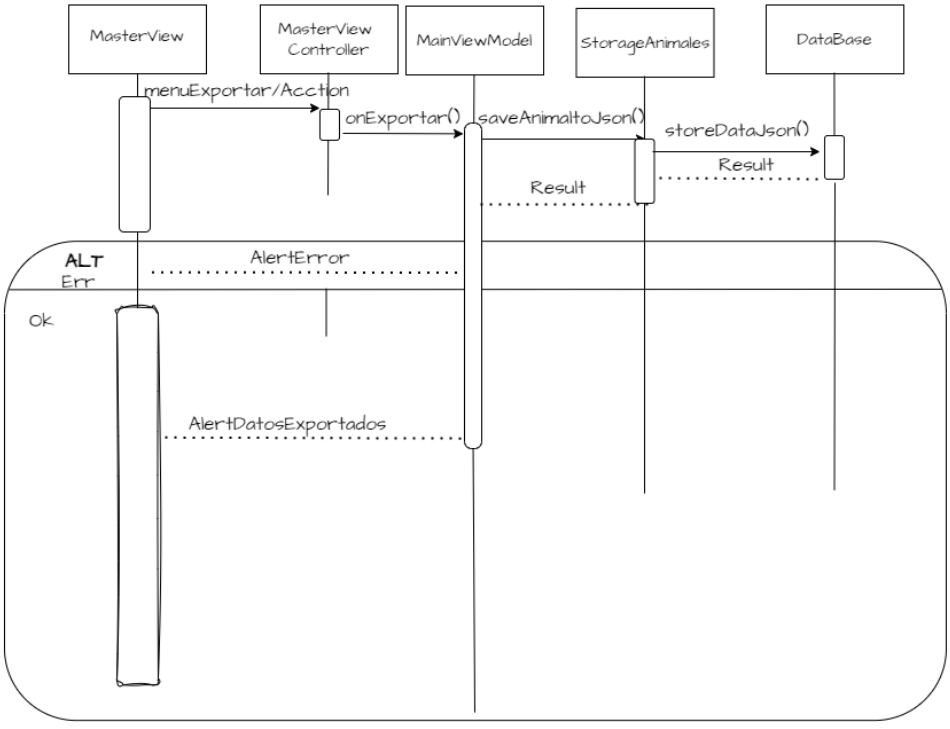


Diagrama de Secuencia: Importar JSON

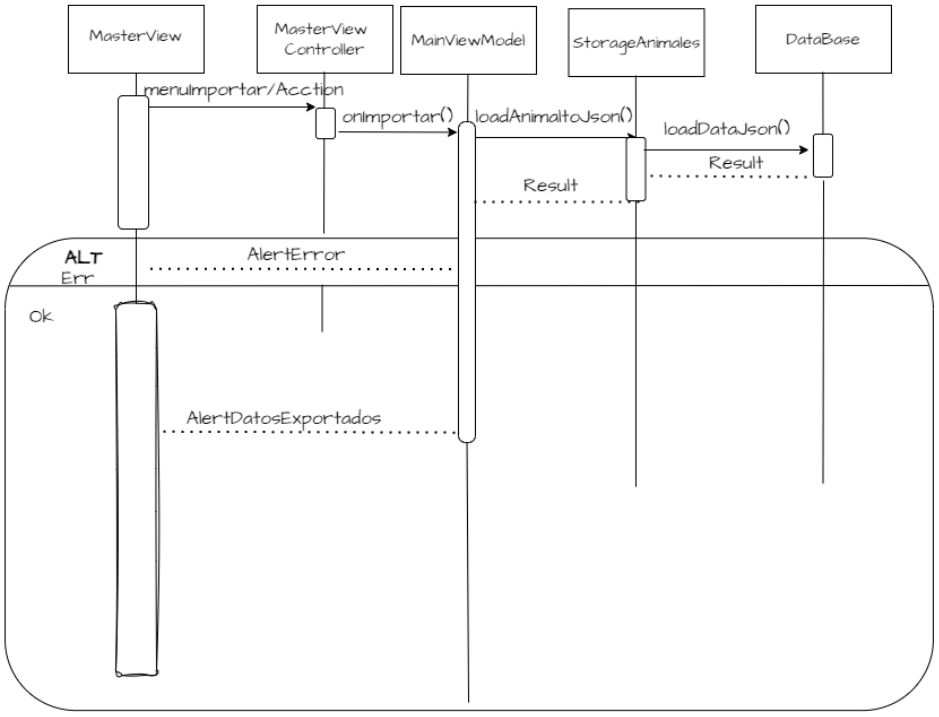
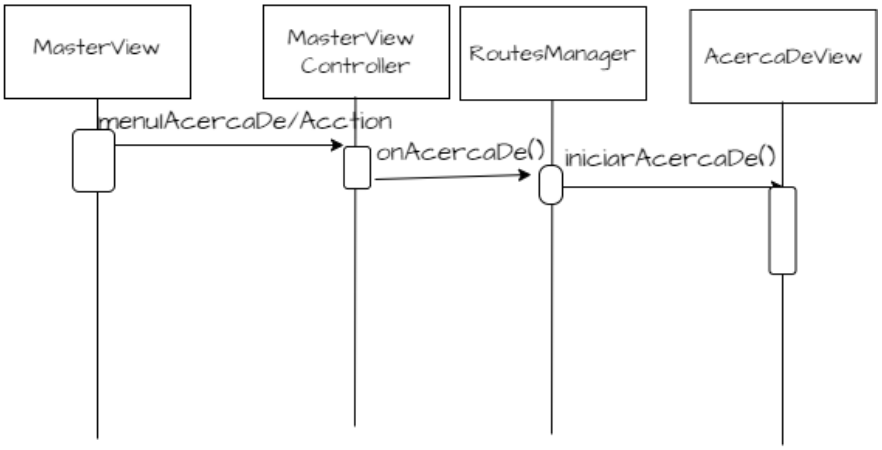


Diagrama de Secuencia: ACERDA DE



Informe de Caso de Uso – Añadir Animal (RF-01):

Resumen: El caso de uso "Crear" permite al actor usuario insertar un animal en el sistema gestor (bbdd)

Secuencia de pasos
1. El usuario selecciona la opción de crear
2. El sistema muestra la nueva vista donde insertar los valores del nuevo animal
3. El usuario presiona guardar.
4. El sistema valida datos.
5. El sistema almacena el nuevo animal

Flujo Alternativo:
1. El usuario decide no guardar los cambios.
2. la ventana se cierra y vuelve a mostrar la principal

Informe de Caso de Uso – Editar Animal (RF-03):

Resumen: El caso de uso "Editar" permite al actor usuario editar un animal en el sistema gestor (bbdd)

Secuencia de pasos
1. El usuario selecciona la opción de editar
2. El sistema muestra la nueva vista donde editar los valores del animal
3. El usuario presiona guardar.
4. El sistema valida datos.
5. El sistema almacena el animal modificado

Flujo Alternativo:
1. El usuario decide no guardar los cambios.
2. la ventana se cierra y vuelve a mostrar la principal

Informe de Caso de Uso – Eliminar Animal (RF-04):

Resumen: El caso de uso "Eliminar" permite al actor usuario eliminar un animal en el sistema gestor (bbdd)

Secuencia de pasos
1. El usuario selecciona la opción de eliminar
2. El sistema muestra ventana emergente pidiendo confirmación
3. El usuario presiona confirmar.
4. El sistema busca por id y elimina el animal.

Flujo Alternativo:
1. El usuario decide no guardar los cambios.
2. la ventana se cierra y vuelve a mostrar ventana principal

Informe de Caso de Uso – Exportar Animal (RF-05):

Resumen: El caso de uso "Exportar" permite al actor usuario exportar a Json todos los animales de la bbdd

Secuencia de pasos
1. El usuario selecciona la opción menuExportar.
2. El sistema busca todos los animales
3. El sistema crea el json y guarda los valores

Flujo Alternativo:
1. El archivo no se pudo crear por algún motivo

Informe de Caso de Uso – Importar Animal (RF-06):

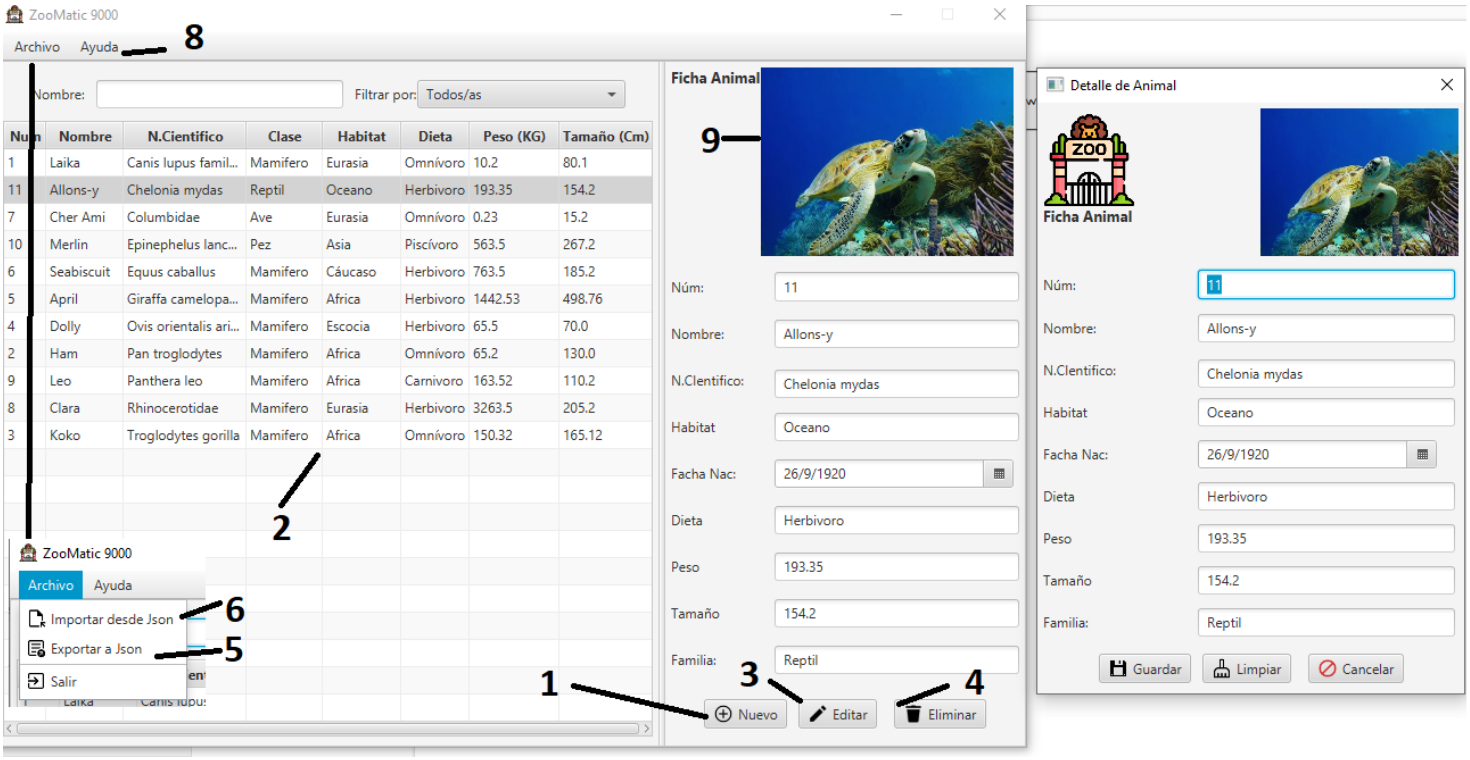
Resumen: El caso de uso "Importar" permite al actor usuario Importar de un Json todos los animales a la bbdd

Secuencia de pasos
1. El usuario selecciona la opción menuImportar.
2. El sistema busca el archivo
3. El sistema borra la bbdd
4.El sistema carga los datos del Json

Flujo Alternativo:
1. El archivo no se pudo cargar por algún motivo

Interfaz

En este apartado muestro captura de la aplicación y asocio cada parte a sus requisitos funcionales



Como se puede ver cumplimos todos los puntos añadiendo un visor lateral de información para solo tener que abrir la view detalle si se ha de modificar o crear un animal.

Decisiones sobre el código y solución de errores

Existen varias decisiones que tenemos que tomar antes de empezar con el código ¿Qué base de datos usamos? ¿Qué librería para JSON?

```
CREATE TABLE IF NOT EXISTS AnimalTable (
    id                INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre            TEXT NOT NULL,
    nombreCientifico  TEXT    NOT NULL,
    clase             TEXT NOT NULL,
    habitat           TEXT NOT NULL,
    fechaNacimiento   TEXT NOT NULL,
    dieta             TEXT NOT NULL,
    peso              REAL NOT NULL,
    tamanio           REAL NOT NULL,
    imagen            TEXT NOT NULL
);
```

```
// SQLDeLight
implementation("app.cash.sqldelight:sqlite-driver:2.0.0-alpha05")
implementation("org.xerial:sqlite-jdbc:3.41.2.1")
```

Me decante por usar Sqldeligth por ser la que más uso y no tener problemas de tiempo con imprevistos sorpresa.

```
    // Gson
    implementation("com.google.code.gson:gson:2.10.1")
override fun storeDataJson(file: File, data: List<Animal>): Result<Long, AnimalError> {
    logger.debug { "Guardando datos en fichero $file" }
    return try {
        val gson = GsonBuilder().setPrettyPrinting().create()
        val jsonString = gson.toJson(data.toDto())
        file.writeText(jsonString)
        Ok(data.size.toLong())
    } catch (e: Exception) {
        Err(AnimalError.SaveJson("Error al escribir el JSON: ${e.message}"))
    }

}
```

En cuanto a la librería use Gson por que da menos problemas que Moshi para JavaFx y por su sencillez, al usar Dto no se necesitan ni adaptadores ni nada. Maravilloso

En cuanto al diseño usamos una arquitectura por capas como se nos pide y usamos un sistema de carpetas donde diferenciamos cada parte de nuestra aplicación. Usamos también railway programming para manejar los errores de una manera funcional.

El código esta comentado con JavaDoc para que sea más fácil su comprensión

Alguno de los errores que sufrí durante el proyecto fueron.

Problemas con el módulo JavaFx y los imports – solucionado añadiendo imports y opens

Problema que bloqueaba los campos de texto y no dejaba editarlos. -- Estaban con edición en falso en StageBuilder

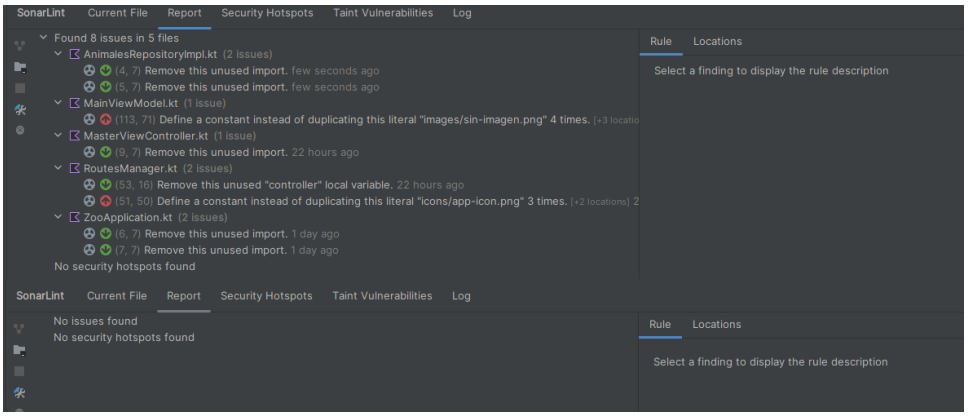
Las imágenes mueven los cuadros inferiores al cambiar – bloquear tamaño de la barra. Ahora la foto dimensiona por debajo. (se podría bloquear también pero no me resulta molesto, me parece peor que no conserve proporciones y deforme imagen)

Los datos se modifican aun dando a el botón cancelar—para este fallo pensé en almacenar los valores iniciales cuando se inicia view y si se pulsa cancelar volver a sobrescribir los valores con los iniciales, pero tras horas dando vueltas veo que me falta soltura para esta clase de errores.

Para la realización de los test igual no fui capaz de mockear la basededatos. Me saltaba siempre que es null

Unas cuantas horas desperdiciadas.

El Sonar solo marco unos cuantos imports que ya no usaba, para que los borrara y me pedía crear un par de constantes para la dirección del icono de la app y la imagen por defecto ya que se repetía 3 veces.



Conclusiones

Pues más de 40 horas de trabajo y aún le quedan algunos errores y los test. Pero en general todo salió, solucione la mayoría de los fallos y los esquemas la verdad que me costó hacerlos, pero creo que quedaron bien Con algo más de soltura para el próximo examen de entornos. Pero apenado por tener aun que preparar el examen y no poder dedicarle un poco más de tiempo para pulirlo.

Aun que el Lunes empecé con ganas y el miércoles ya estaba casi terminado. Pulir fallos, crear la documentación , añadir JavaDoc y la realización de esta memoria me ocupo mas tiempo del deseado. Y bueno el fracaso rotundo con los test a bbdd.

"Sed fugit interea, fugit irreparabile tempus"