Grado en Ingeniería Informática

Trabajo de Final de Grado

Desarrollo de una consulta de explotaciones ganaderas en función del perfil del usuario

Autor:
Jorge Revaliente Lainez

Supervisor:
Javier Vilarroya Villalonga
Tutor académico:
Sergio Barrachina Mir

Fecha de lectura: 13 de julio de 2023
Curso académico 2022/2023

Resumen

En este documento se desarrolla el Trabajo Fin de Grado realizado por Jorge Revaliente Lainez durante su estancia en la empresa Sopra Steria.

El proyecto consiste en el desarrollo de una consulta para el portal agrario, denominado oficialmente Oficina Virtual Veterinaria (OVV), perteneciente a la Generalitat Valenciana. Esta consulta permite visualizar las subexplotaciones ganaderas accesibles por un usuario con permisos de acceso al portal. Esta consta de un bloque de filtros que permite al usuario refinar el resultado de la consulta en base a una serie de criterios de búsqueda. Además, se permite consultar información adicional de cada una las subexplotaciones, así como la exportación en un archivo *Excel* del listado de las subexplotaciones que cumplen con los criterios de búsqueda antes nombrados.

La parte *backend* de la aplicación web ha sido desarrollada haciendo uso del *framework* Spring; un *framework* escrito en el lenguaje de programación *Java*.

La parte *frontend* en cambio, se ha desarrollado utilizando el motor de plantillas *Thymeleaf*, *HTML5* y *ES6 JavaScript* entre otras tecnologías y/o bibliotecas comúnmente utilizadas en el desarrollo *frontend*.

En este documento se describe la planificación previa del proyecto y su análisis, con la explicación de cada uno de los componentes implementados en el proyecto. Por último, se detallan las pruebas realizadas y se exponen las conclusiones finales.

Palabras clave

Consulta de explotaciones, subexplotaciones, Oficina Virtual Veterinaria, *API REST*, *HTTP*, *Excel*, *Spring*, *Thymeleaf*, *AJAX*, punto final.

Keywords

Livestock farming, underexploitation, Virtual Veterinary Office, *API REST*, *HTTP*, *Excel*, *Spring*, *Thymeleaf*, *AJAX*, *endpoint*.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Este proyecto de final de grado se ha desarrollado en la empresa Sopra Steria.[1] Empresa multinacional francesa especializada en la digitalización de empresas y desarrollo de software. Cuenta con nueve oficinas en España, una de las cuales está en Valencia, que es donde he realizado mi estancia.

La empresa, entre la cartera de proyectos que lleva, cuenta con una colaboración con la Dirección General de Tecnologías de la Información y las Comunicaciones (DGTIC), perteneciente a la Generalitat Valenciana. La DGTIC:

"Es el órgano de la Generalitat que asume las competencias, para la Administración de la Generalitat y sus organismos autónomos, en materia de seguridad de la información, planificación, coordinación, autorización y control de las Tecnologías de la Información (TI), las telecomunicaciones, las comunicaciones corporativas y la teleadministración de la Generalitat" [21].

En esta colaboración, la DGTIC propone a Sopra Steria distintos proyectos a realizar, que generalmente son aplicaciones web para distintas consejerías de la Generalitat.

El proyecto de Sopra Steria en el que he trabajado trata sobre la creación de una Oficina Virtual Veterinaria (OVV). Portal web que proporcionará a ganaderos, veterinarios y administrativos una serie de trámites que podrán realizar en línea, lo que les evitará tener que realizarlos presencialmente en las oficinas comarcales agrarias.

Entre las diferentes funcionalidades que se integrarán en la OVV, una de las más importantes y que se ha priorizado en la puesta en marcha del portal es la que tiene por objeto este proyecto de final de grado: la consulta de explotaciones ganaderas en función del perfil del usuario.

[1] `https://www.soprasteria.es/`

1.2. Objetivo y alcance del proyecto

El proyecto se enmarca dentro de los siguientes objetivos ganaderos de la Generalitat Valenciana:

- Mejorar el área de la ganadería valenciana en el ámbito informático.
- Reducir la carga de las oficinas comarcales mediante la automatización de la consulta.

El objetivo principal de este proyecto es diseñar y crear un trámite dentro de la OVV que permita consultar explotaciones ganaderas dependiendo del tipo de usuario. Este objetivo se puede desglosar en los siguientes subobjetivos:

- Proporcionar un entorno gráfico donde el usuario pueda obtener información de sus explotaciones.
- Crear unos bloques de filtros con listas desplegables que permitan filtrar la información mostrada en la consulta.
- Extraer el resultado de la consulta en un fichero de tipo *Excel*.

El alcance del proyecto se ha limitado al desarrollo de código relacionado con la interfaz de usuario o parte *frontend* de la consulta, que a la vez tiene su propia parte trasera o *backend*. Esta parte trasera contiene los componentes necesarios para el funcionamiento de la interfaz de usuario, como por ejemplo, la parte cliente de una *Application Programming Interface (API) Representational State Transfer (REST)* [36], encargada de realizar llamadas a la parte servidor de una *API REST* para conseguir la información necesaria en la interfaz de usuario.

Por otro lado, el alcance no incluye el desarrollo de la parte servidor del *API REST*, ya que fue asignada a un compañero de la empresa.

Alcance funcional

La consulta muestra las subexplotaciones relacionadas con las explotaciones a las que el usuario conectado tiene acceso. Esto reducirá la carga administrativa de las oficinas comarcales y facilitará la vida de los usuarios administrados.

Existen diversos perfiles de usuario, lo que influye en el número de subexplotaciones mostradas.

Además, el usuario puede hacer uso de diversos filtros para acotar la información consultada y descargarla en un fichero de tipo *Excel*.

Alcance organizativo

El equipo que desarrolló la consulta de explotaciones estuvo compuesto por: un jefe de proyecto, que coordinó a las personas del equipo; dos analistas programadores, uno encargado de realizar la parte servidor de la *API REST* y otro encargado de guiarme en el proyecto; y por último, un desarrollador, que fui yo.

Existen tres tipos de usuarios: titular, veterinario de la Agrupación de Defensa Sanitaria Ganadera (ADSG) y administrador. El titular y el veterinario ADSG pueden hacer consultas personalizadas, como por ejemplo, cuáles son las subexplotaciones asociadas solo a las explotaciones a las que tienen acceso. En cambio, el administrador puede consultar las subexplotaciones asociadas a todas las explotaciones existentes en la Comunidad Valenciana.

Otra organización que interviene en el proyecto es la DGTIC, que es quien encargó el proyecto.

Alcance informático

La consulta desarrollada interacciona con la parte servidor de una *API REST*, creada por el compañero encargado de la parte del servidor. Es una *API* creada a partir de los principios de la arquitectura *REST*, encargada de establecer la comunicación cliente-servidor entre el cliente *API REST* que he desarrollado y una base de datos, que almacena la información necesaria para la consulta de explotaciones [36]. Esta base de datos no entra tampoco en el alcance del proyecto, ya que no fue creada por mí.

La comunicación cliente-servidor se realiza mediante los puntos finales o *endpoints* de la *API REST*. Son ubicaciones digitales que reciben solicitudes sobre un recurso en el servidor al que está conectado [25].

La comunicación entre el cliente y servidor de la *API REST* se realiza mediante el protocolo *HTTP (Hypertext Transfer Protocol)* [50]. El cliente de la *API REST* realiza peticiones *HTTP* al servidor *API REST*, que capta la información de la base de datos y la manda de vuelta al cliente.

1.3. Descripción detallada del proyecto

El proyecto consiste en la creación de un trámite adicional en la OVV sobre la consulta de explotaciones ganaderas relacionadas con el perfil del usuario.

Las explotaciones accesibles dependen del tipo de usuario. Los titulares son propietarios de las explotaciones, por lo que solo pueden acceder a aquellas que les pertenezcan. Los veterinarios ADSG pueden pertenecer a una o varias ADSG, por lo tanto, tienen acceso a las explotaciones gestionadas por las ADSG a las que pertenezcan. Los administradores pueden acceder a todas las explotaciones disponibles.

Cada explotación puede tener una o varias subexplotaciones y cada subexplotación se distingue por la familia de la especie animal que contenga.

La primera vez que se entra al trámite, solo muestra las subexplotaciones relacionadas con la explotación a la que se ha conectado.

Las subexplotaciones aparecen en un cuadro, con una fila por subexplotación y 10 columnas, donde muestran información de la subexplotación que representa cada fila y la explotación a la que pertenece. Además, cada fila tiene una columna adicional con un botón que redirige a una vista con más información de la subexplotación que representa.

Una vez dentro del trámite, pueden elegirse qué subexplotaciones mostrar gracias a unos bloques de filtros. Estos bloques contienen listas desplegables con distintos parámetros acerca de las subexplotaciones que se quieren mostrar y de las explotaciones a las que pertenece el usuario. Existen tres tipos de bloques y se muestra uno u otro según el tipo de usuario conectado: titular, veterinario ADSG y administrador.

Por último, el trámite contiene dos botones, uno que exporta las subexplotaciones mostradas en un archivo de tipo *Excel* y otro que permite salir del trámite.

1.3.1. Tecnologías utilizadas

En este apartado se describen las tecnologías utilizadas para el desarrollo del proyecto. Se dividen en seis tipos: lenguajes de programación, *frameworks*, bibliotecas, motores de plantillas, *plugins* y herramientas de desarrollo.

Los lenguajes de programación utilizados son:

- *Java*: se trata de un lenguaje de programación orientado a objetos con múltiples usos. Contiene una sintaxis similar al lenguaje de programación *C++* y utiliza la idea de la máquina virtual del lenguaje *Smalltalk* [11]. Los programas escritos en lenguaje *Java* son independientes del entorno en el que se ejecuten gracias a la Máquina Virtual *Java* [20]. Gran parte de este proyecto ha sido escrito en este lenguaje de programación, utilizando *Spring Boot*.
- *JavaScript*: lenguaje de programación utilizado para crear páginas web dinámicas. Es capaz de modificar el contenido de una página web de forma dinámica, permitiendo añadirle funcionalidades como animaciones o interactividad a sus componentes. También es un lenguaje interpretado, ya que no necesita una compilación previa para ser ejecutado por el propio navegador [17].

Los *frameworks* utilizados se dividen en:

- *Spring*: *framework* utilizado para desarrollar aplicaciones en *Java*. Contiene diferentes módulos que facilitan el desarrollo de una aplicación *Java*. Uno de los módulos es *Spring MVC*, que implementa el patrón de diseño *Modelo-Vista-Controlador (MVC)* [58].

- *Spring Boot*: es una extensión del *framework Spring* que facilita el desarrollo de una aplicación en este *framework*. Algunas de las razones que facilitan su desarrollo son: aportación de las dependencias necesarias para su funcionamiento, un servidor integrado para ejecutar la aplicación, autoconfiguración del *framework*, etc [8].
- *Bootstrap: framework* de *CSS* [49] dirigido al desarrollo web *responsive* y móvil. Contiene *HTML* [51], *CSS* y plantillas de diseño basadas en *JavaScript* para componentes de la interfaz como: botones, navegación, formularios, etc. La tecnología *responsive* optimiza el contenido de la página web en función del tamaño del dispositivo que lo esté visualizando [43]. Ha sido usado en el proyecto para adaptar el tamaño de los componentes de la vista a dispositivos distintos de un computador, como móviles o tabletas.

Las bibliotecas usadas en el proyecto son:

- *jQuery*: biblioteca de *JavaScript* con múltiples funciones. Simplifica la manera de manipular documentos *HTML*, manejar los eventos, animaciones y *AJAX*. También, su código está preparado para ser compatible con cualquier navegador web. Por todo ello, facilita la programación nativa en *JavaScript* [33].
- *Selectize*: consiste en una biblioteca de *JavaScript* que mejora el uso del componente *select* de *HTML*, el cual representa una lista desplegable con distintos elementos [53]. Algunas de las mejoras que aporta son: permite seleccionar varios elementos de la lista, realizar una búsqueda entre ellos o limitar el número de elementos seleccionados [59].

El motor de plantillas utilizado ha sido *Thymeleaf*. Se trata de un motor de plantillas de *Java* para archivos *XML*, *XHTML* y *HTML5*. Es popularmente usado en la vista dentro del patrón *MVC* en los proyectos *Spring*, es por ello que posee un dialecto propio llamado *SpringStandard* que adapta *Thymeleaf* a varias funciones de *Spring* [55].

El único *plugin* utilizado ha sido *DataTables*. *Plugin* de la biblioteca *jQuery* que extiende el funcionamiento del elemento *table* de *HTML* [54]. Alguna de las características que añade son: una paginación que divide las filas del cuadro en diferentes páginas, una barra de búsqueda para filtrar el contenido de las filas y para ordenar las filas por cada una de las columnas del cuadro, entre otras [41].

Finalmente, se muestran las herramientas de desarrollo que se han usado:

- *Spring Tool Suite (STS)*: entorno de desarrollo integrado, o en inglés *integrated development environment (IDE)*, para desarrollar aplicaciones en el *framework Spring* y que está basado en el *IDE Eclipse*. *STS* contiene varias funcionalidades que facilitan el desarrollo de código en *Spring*, como el autocompletado de código o una mejora visual en la edición de archivos de configuración de *Spring* [57].
- *Swagger UI*: herramienta que ofrece una interfaz de usuario donde visualizar los puntos finales de una *API REST*. También permite interactuar con los puntos finales en la propia interfaz y que muestre la información que devuelve cada uno [38]. En el desarrollo del proyecto fue consultada con frecuencia, debido a que los puntos finales creados por el compañero encargado de la *API REST* fueron publicados en esta herramienta.

- *Font Awesome*: biblioteca de iconos para el diseño web. Contiene más de 26.000 iconos divididos en distintas categorías. Además, cada uno contiene estilos basados en *CSS* ya definidos [19]. El proyecto usa la versión gratuita de esta biblioteca.
- *Maven*: herramienta para la gestión de proyectos basados en el lenguaje *Java*. Su objetivo es simplificar el proceso de compilación de un proyecto y establecerle un conjunto de estándares para dicho proceso. Para ello, construye los proyectos usando el *Project Object Model (POM)*, fichero *XML* que contiene información sobre el proyecto, como sus dependencias, que son las bibliotecas que usa el proyecto. *Maven* se encarga de descargar automáticamente las dependencias en el repositorio local, facilitándole al desarrollador esta tarea [48] [27]. Una de las dependencias necesarias es la *API* de *Java APACHE POI*, que permite manejar archivos en formato de *Microsoft Office* [46]. Esta *API* se utilizará para generar los archivos en formato *Excel*.
- *TortoiseSVN*: programa que ofrece una interfaz para facilitar el uso del *software* de control de versiones *Apache Subversion (SVN)*. Esta aplicación se integra en el explorador de archivos de *Windows* [56]. Este programa fue utilizado para gestionar las distintas versiones del proyecto sobre un repositorio *SVN*.

1.4. Estructura de la memoria

Esta memoria está dividida en seis capítulos donde se detalla el trabajo de final de grado en cuestión.

Este primer capítulo proporciona toda la información necesaria para tener un contexto general del proyecto.

El siguiente capítulo es la planificación del proyecto, donde se define una planificación temporal con todas las tareas realizadas en el proyecto. Después, el seguimiento del proyecto, donde se muestran los cambios respecto a la planificación del proyecto acordada. También se estima el coste real del proyecto y la gestión de sus posibles riesgos.

El tercer capítulo incluye el análisis del sistema, donde se desarrollan diagramas para indicar los requisitos del sistema.

En el capítulo cuatro se explican aspectos sobre el diseño de *software* y *hardware* del proyecto, y se muestran capturas de pantalla con las vistas desarrolladas.

El penúltimo capítulo es el de implementación y pruebas. En él se describe todos los aspectos relacionados con el código del proyecto y las pruebas realizadas.

Finalmente, en el capítulo de conclusiones se da una opinión a nivel personal y profesional sobre el proyecto realizado.

En el anexo se muestran diagramas de clases de todas las clases de *Java* creadas en el proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

La metodología usada para el desarrollo de los proyectos de la DGTIC es gvLOGOS.[1] Se trata de una metodología propuesta por la DGTIC para establecer una gestión en los proyectos TIC (Tecnologías de la Información y las Comunicaciones) de la Generalitat Valenciana. Dentro de ella, se contemplan diferentes procesos, como la calidad y la seguridad, entre otros.

Aunque en los proyectos de la DGTIC se utilice la metodología gvLOGOS, para el desarrollo de la consulta de explotaciones decidí utilizar una adaptación del modelo de desarrollo en cascada [32], debido a que se adapta mejor a este trabajo de final de grado. El modelo en cascada original está representado en la figura 2.1.

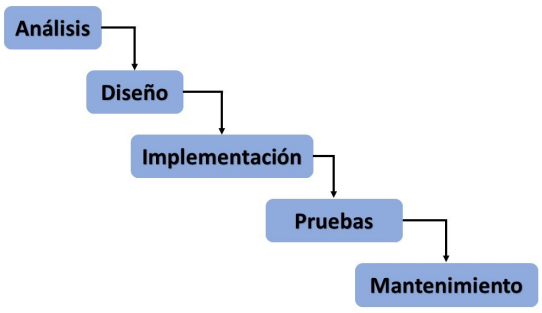


Figura 2.1: Modelo de desarrollo en cascada.

Por contra, la figura 2.2 muestra el modelo en cascada realmente utilizado. Si la comparamos con la figura 2.1, tiene dos diferencias: contiene una fase adicional llamada estudio, donde se estudiaron todas las tecnologías que se iban a usar en el proyecto; y no existe la fase de mantenimiento, porque una vez acabó la fase de pruebas, mi desarrollo en el proyecto finalizó.

[1] `https://dgtic.gva.es/es/metodologia-gvlogos`

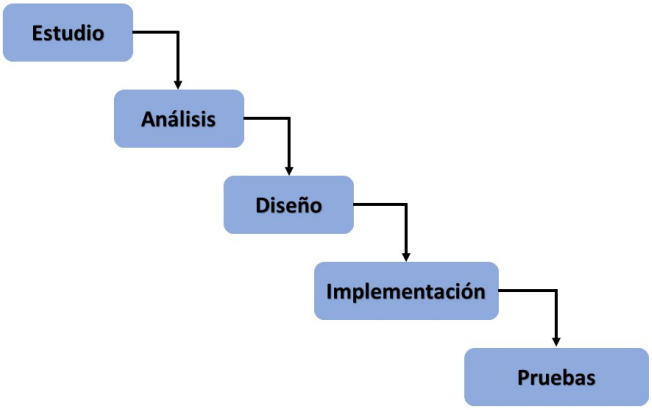


Figura 2.2: Adaptación del modelo en cascada.

2.2. Planificación temporal del proyecto

A continuación, se detallan las tareas desarrolladas dentro del proyecto siguiendo las fases de la adaptación del modelo en cascada representada en la figura 2.2:

- En la fase de estudio, me familiaricé con todas las tecnologías que se iban a usar en el proyecto. Lo que luego me ayudó a avanzar de manera más rápida en la fase de implementación.
- En la fase de análisis definí los requisitos del proyecto. Para ello, estudié un documento de la DGTIC que me proporcionó la empresa donde se explica con detalle la consulta de explotaciones. Algunos de los aspectos que se explican son: capturas de pantalla con las interfaces de usuario a crear, qué campos se muestran en las interfaces y qué usuarios acceden a la consulta, entre otros. Por tanto, en base a este documento creé un diagrama de casos de uso y varios cuadros con los requisitos de datos para definir los requisitos del proyecto [14].
- Continuando con la fase de diseño, primero definí los puntos finales de la *API REST* necesarios. Consistió en definir cuáles iban a ser los puntos finales implementados por el desarrollador de la *API REST*. Después, diseñé los *Data Transfer Objects (DTO)* [35] en diagramas de clases, pertenecientes a los diagramas de Lenguaje Unificado de Modelado (en inglés *UML*) [14]. Los *DTO* han sido implementados en este proyecto con clases *Java*, que almacenan la respuesta de cada punto final. El diseño de los *DTO* se realizó porque al comenzar la fase de diseño no estaban creados los puntos finales, lo que me permitió iniciar la fase de implementación sin esperar a que los puntos finales estuvieran creados.
- En la fase de implementación realicé las siguientes tareas:
  - **Desarrollo del módulo *Data Transfer Objects (DTO)*:** creación de los *DTO* en clases *Java*.

- **Desarrollo del módulo *client***: implementación de las llamadas a los puntos finales de la *API REST*, que proporcionan la información necesaria a la interfaz de usuario.
- **Desarrollo del módulo *service-api*:** creación de las interfaces que definen la lógica de negocio de la consulta.
- **Desarrollo del módulo *service-impl***: implementación de las interfaces creadas en el módulo *service-api*.
- **Desarrollo del módulo *application***:
  - **Desarrollo de la capa controladora:** creación de la clase controladora encargada de manejar las peticiones *HTTP* que permiten a los usuarios interactuar con la aplicación a través de un navegador web.
  - **Desarrollo de la capa vista:**
    - Implementación de las pantallas de la interfaz de usuario.
    - Creación del fichero *JavaScript* que aporta funcionalidades a la interfaz de usuario.

- Al acabar la fase de implementación pasé a la fase de pruebas. En esta fase se realizaron una serie de pruebas para asegurar que la consulta funciona correctamente y está preparada para ser usada.

2.2.1. Estructura de desglose del trabajo

En la figura 2.3 se muestra la Estructura de desglose del trabajo (EDT) [31] donde se representan de manera más visual las tareas descritas en el apartado anterior.

Figura 2.3: Estructura de desglose del trabajo (EDT).

2.2.2. Diagrama de Gantt

El diagrama de *Gantt* mostrado en la figura 2.4 contiene todas las tareas del proyecto organizadas en una línea de tiempo desde la fecha de inicio a fin de las prácticas. Cada fase está indicada con una barra central de color naranja.

Las tareas se planificaron acorde a las 300 horas de prácticas realizadas, con una jornada de siete horas diarias.

La duración de las tareas fueron estimadas antes de comenzar el proyecto, por tanto, no muestra el tiempo real en días que duró cada tarea. En el apartado 2.3 se muestra un diagrama de *Gantt* con la duración real de cada una de las tareas.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
0	**Consulta de explotaciones ganaderas**	**43 días**	**30/01**	**29/03**	
1	**Estudio**	**5 días**	**30/01**	**03/02**	
2	Estudio de las tecnologías usadas en el proyecto	5 días	30/01	03/02	
3	**Análisis**	**2 días**	**06/02**	**07/02**	
4	Estudio del documento de la DGTIC con los requisitos	1 día	06/02	06/02	2
5	Creación del diagrama de casos de uso	0,5 días	07/02	07/02	4
6	Creación de tablas con los requisitos de datos	0,5 días	07/02	07/02	5
7	**Diseño**	**2 días**	**08/02**	**09/02**	
8	Definición de los puntos finales de la API REST	1 día	08/02	08/02	6
9	Diseño de los DTO en diagramas de clases	1 día	09/02	09/02	8
10	**Implementación**	**31 días**	**10/02**	**24/03**	
11	Desarrollo del módulo DTO	3 días	10/02	14/02	9
12	Desarrollo del módulo client	5 días	15/02	21/02	11
13	Desarrollo del módulo service-api	1 día	22/02	22/02	12
14	Desarrollo del módulo service-impl	4 días	23/02	28/02	13
15	**Desarrollo del módulo application**	**18 días**	**01/03**	**24/03**	
16	Desarrollo de la capa controladora	8 días	01/03	10/03	14
17	**Desarrollo de la capa vista**	**10 días**	**13/03**	**24/03**	
18	Implementación de las pantallas de la interfaz de usuario	5 días	13/03	17/03	16
19	Creación del fichero JavaScript	5 días	20/03	24/03	18
20	**Pruebas**	**3 días**	**27/03**	**29/03**	
21	Realización de pruebas	3 días	27/03	29/03	19

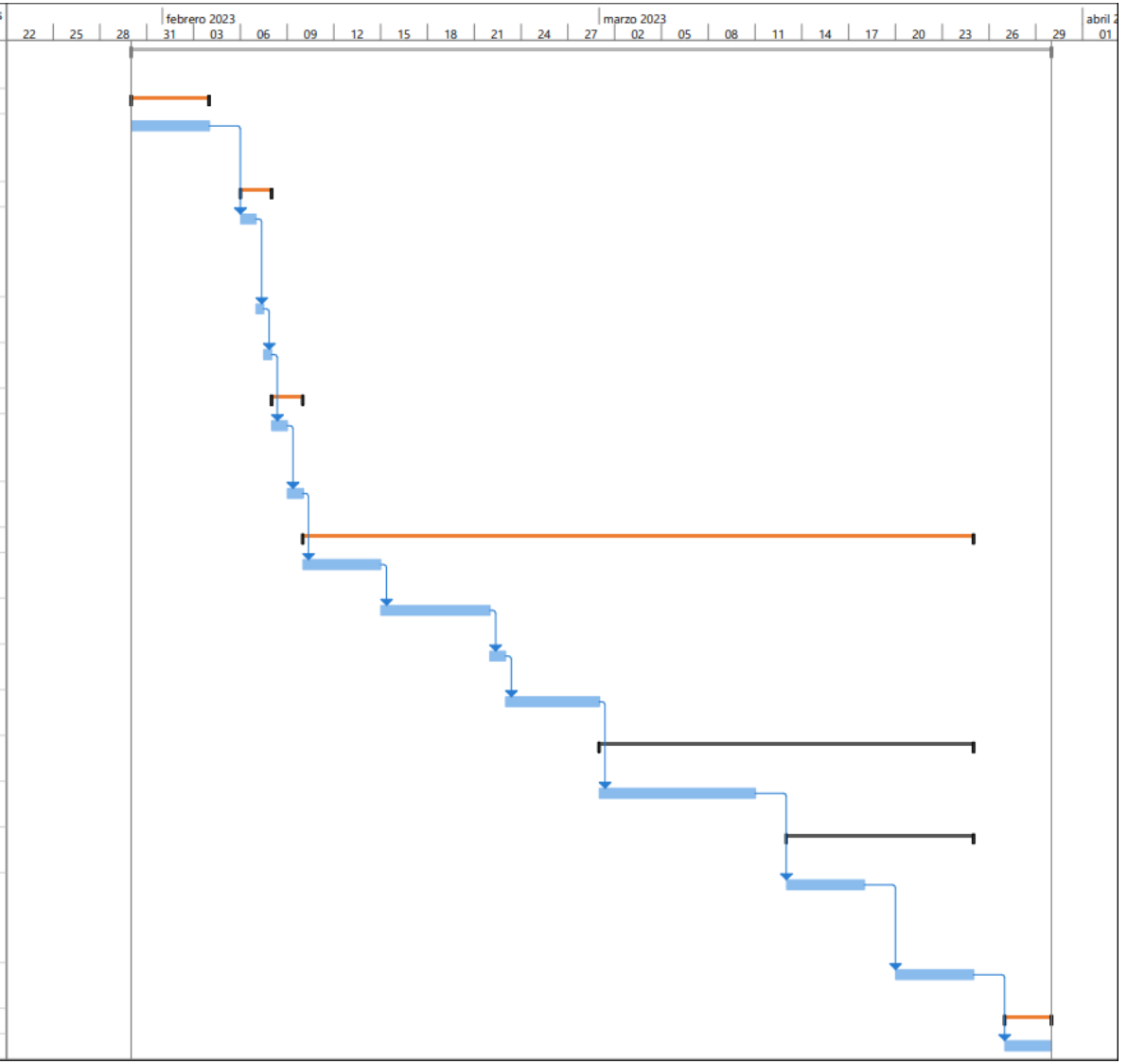


Figura 2.4: Diagrama de *Gantt*.

2.3. Seguimiento del proyecto

En este apartado se describe el seguimiento real que tuvo el proyecto durante su desarrollo y sus desviaciones en comparación con la planificación inicial. Este seguimiento se expone de forma gráfica en el diagrama de *Gantt* en la figura 2.5.

La fase de estudio de las tecnologías tardó un día más de lo previsto, debido a que eran muchas las tecnologías por aprender. Esta pequeña desviación aseguró que no se retrasara la fase de implementación debido a la inexperiencia en las tecnologías.

Siguiendo con la fase de análisis, finalizó un día más tarde, ya que un día no fue suficiente para realizar el diagrama de casos de uso y unos cuadros con los requisitos de datos.

Por otro lado, la fase de diseño se realizó en el tiempo planeado, porque dos días fueron suficientes para definir los puntos finales y diseñar en diagramas de clases todos los *DTO*.

En la fase de implementación hubo una serie de desviaciones que se describen a continuación.

El desarrollo del módulo *DTO* disminuyó en un día, ya que al haberlos representado con diagramas de clases en la fase de diseño facilitó la tarea drásticamente.

La siguiente tarea fue el desarrollo del módulo *client*, que tardé dos días menos respecto a lo planeado. Esto fue debido a que la implementación de las llamadas a los puntos finales fue rápida de realizar.

El tiempo planeado en el desarrollo del módulo *service-api* no cambió al ser la tarea sencilla, en cambio, el módulo *service-impl* tardó un día más en realizarse, al ser la implementación más complicada de lo pensado.

Las tres tareas dentro del desarrollo del módulo *application* fueron realizadas al mismo tiempo y no de manera secuencial, como fue planeado. Esta desviación fue debido a mi falta de experiencia en este tipo de proyectos. Tardé 20 días en acabar las tres tareas.

El desarrollo de las tres tareas tuvo los siguientes pasos: primero, implementaba un método de la capa controladora, después creaba sus componentes asociados en la interfaz de usuario y, por último, si fuera necesario, creaba una o varias funciones dentro del fichero *JavaScript* que ampliaban el funcionamiento de los dos pasos anteriores. Este proceso lo repetí con todos los métodos de la capa controladora, de esta manera, comprobé que las tres tareas funcionaban de manera correcta.

Por ejemplo, después de implementar el método de la capa controladora que descarga el resultado de la consulta de explotaciones en un *Excel*, creé en la capa vista el botón ≪Descargar Excel≫. Una vez creado el botón, creé una función dentro del fichero *JavaScript* que al pulsar el botón, ejecuta el método creado en la capa controladora y descarga en el equipo del usuario el archivo *Excel*.

Por último, la fase de pruebas disminuyó en dos días, debido a que se realizaron pruebas manuales para comprobar el funcionamiento de la consulta.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
0	**Consulta de explotaciones ganaderas**	**43 días**	**30/01**	**29/03**	
1	**Estudio**	**6 días**	**30/01**	**06/02**	
2	Estudio de las tecnologías usadas en el proyecto	6 días	30/01	06/02	
3	**Análisis**	**3 días**	**07/02**	**09/02**	
4	Estudio del documento de la DGTIC con los requisitos	1 día	07/02	07/02	2
5	Creación del diagrama de casos de uso	1 día	08/02	08/02	4
6	Creación de tablas con los requisitos de datos	1 día	09/02	09/02	5
7	**Diseño**	**2 días**	**10/02**	**13/02**	
8	Definición de los puntos finales de la API REST	1 día	10/02	10/02	6
9	Diseño de los DTO en diagramas de clases	1 día	13/02	13/02	8
10	**Implementación**	**31 días**	**14/02**	**28/03**	
11	Desarrollo del módulo DTO	2 días	14/02	15/02	9
12	Desarrollo del módulo client	3 días	16/02	20/02	11
13	Desarrollo del módulo service-api	1 día	21/02	21/02	12
14	Desarrollo del módulo service-impl	5 días	22/02	28/02	13
15	**Desarrollo del módulo application**	**20 días**	**01/03**	**28/03**	
16	Desarrollo de la capa controladora	20 días	01/03	28/03	14
17	**Desarrollo de la capa vista**	**20 días**	**01/03**	**28/03**	
18	Implementación de las pantallas de la interfaz de usuario	20 días	01/03	28/03	14
19	Creación del fichero JavaScript	20 días	01/03	28/03	14
20	**Pruebas**	**1 día**	**29/03**	**29/03**	
21	Realización de pruebas	1 día	29/03	29/03	16;18;19

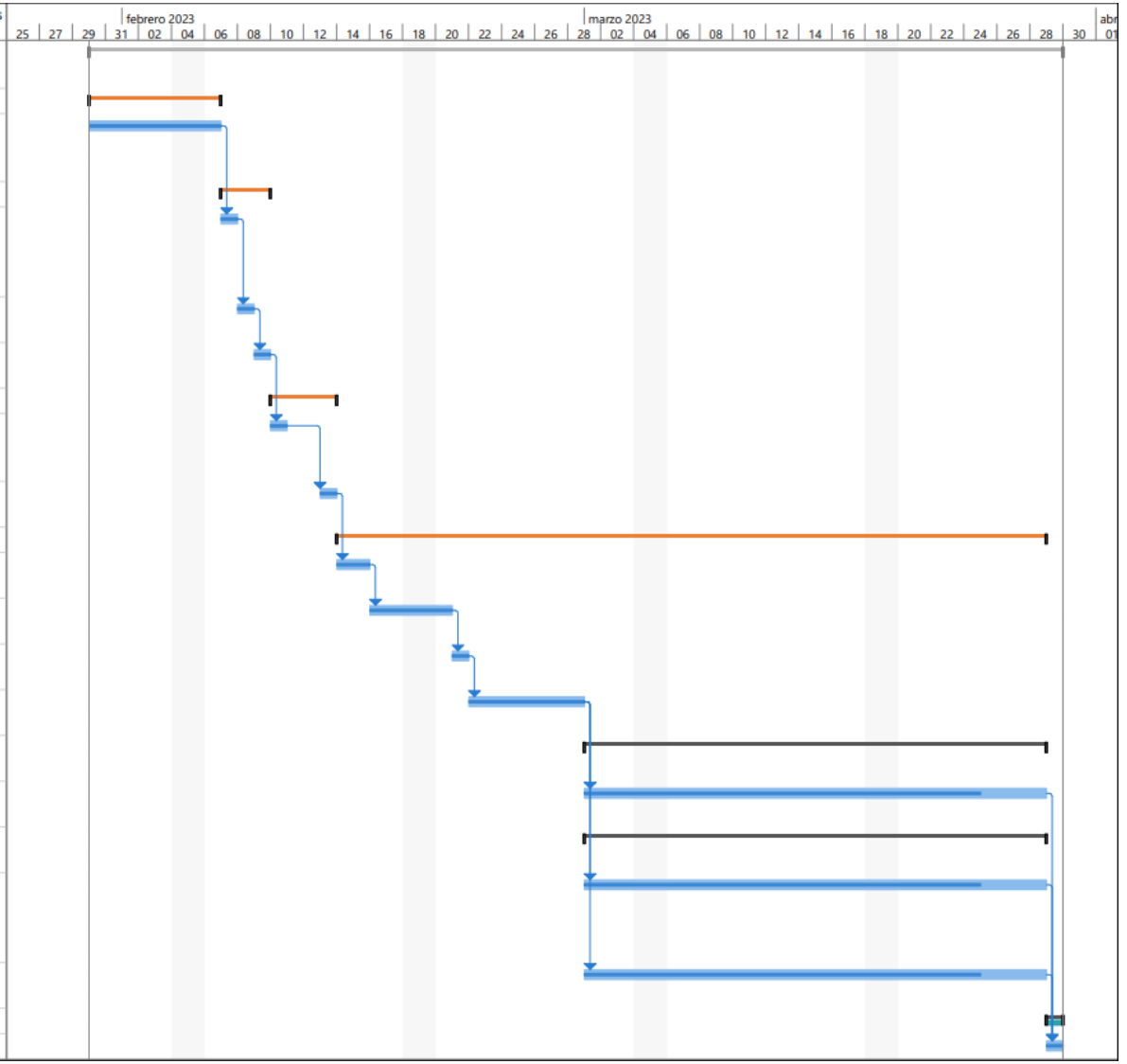


Figura 2.5: Diagrama de *Gantt* con el seguimiento del proyecto.

2.4. Estimación de recursos y costes del proyecto

En este apartado se describen los recursos usados en el proyecto junto a sus costes. Los recursos se dividen en tres tipos: humanos, tecnológicos y en infraestructura y servicios.

2.4.1. Recursos y costes humanos

Los recursos humanos en el desarrollo de la consulta de explotaciones fueron cuatro personas: un jefe de proyecto, dos analistas programadores, uno para desarrollar la parte servidor de la *API REST* y otro que me guió en el proyecto; y un desarrollador *junior*, que fui yo.

Para calcular el coste del jefe de proyecto se usa el salario medio anual en bruto de un jefe de proyecto en España. Según la página *Glassdor* [23], es de 44.598,00 €/año, dando un total de 3.716,50 €/mes. Estimando que su carga semanal son 40 horas, trabaja un total de 160 horas al mes, con un coste de 23,23 €/h.[2] Dedicó 30 minutos al día a coordinar las personas involucradas en el proyecto, entonces el coste prorrateado que tuvo es de 11,62 €/día, la mitad de su coste por hora. Este calculo se muestra en la fórmula 2.1.

$$CosteDiaJefeProyecto = 23{,}23\,€/\text{h} \times \frac{1\,\text{h}}{60\,\text{min}} \times 30\,\text{min/día} = 11{,}62\,€/\text{día} \tag{2.1}$$

Una vez calculado el coste por día del jefe de proyecto, se calcula en la fórmula 2.2 su coste acorde a los 43 días que duró la estancia.

$$CosteJefeProyecto = CosteDiaJefeProyecto \times 43\,\text{días} = 499{,}66\,€ \tag{2.2}$$

Para calcular mi coste en el proyecto se usa el salario medio anual de un programador *junior* en España, que según la página *Glassdor* [24] es de 19.000,00 €/año (1.583,33 €/mes). Mi carga mensual era de 140 horas, por lo tanto el coste por hora fue de 11,31 €/h.[3] En la fórmula 2.3 se calcula mi coste, teniendo en cuenta que la jornada laboral fue de 7h/día durante 43 días.

$$CosteDesarrolladorJunior = 11{,}31\,€/\text{h} \times 7\,\text{h/día} \times 43\,\text{días} = 3404{,}31\,€ \tag{2.3}$$

Siguiendo con el primer analista programador, se aplica el salario medio mensual que tiene un analista programador en España. Según la página *Talent* [45], es de 2.631,00 €/mes. Suponiendo que su trabajo semanal son 40 horas, trabaja 160 horas al mes, siendo su coste por hora: 16,44 €/h.[4]

Este desarrollador solo podía dedicar tres horas al día al desarrollo de la parte servidor de la *API REST*, debido a que trabajaba en varios trámites de la OVV a la vez.

Teniendo en cuenta que trabajaba tres horas diarias durante 43 días, con la fórmula 2.4 se calcula su coste.

$$CosteAnalistaProgramador1 = 16{,}44\,€/\text{h} \times 3\,\text{h/día} \times 43\,\text{días} = 2120{,}76\,€ \tag{2.4}$$

[2] $3716{,}50\,€/\text{mes} \times \frac{1\,\text{mes}}{160\,\text{horas}} = 23{,}23\,€/\text{h}$

[3] $1583{,}33\,€/\text{mes} \times \frac{1\,\text{mes}}{140\,\text{horas}} = 11{,}31\,€/\text{h}$

[4] $2631{,}00\,€/\text{mes} \times \frac{1\,\text{mes}}{160\,\text{horas}} = 16{,}44\,€/\text{h}$

Por último, para calcular el coste del segundo analista programador, se vuelve a usar el salario medio mensual de un analista programador en España, que es de 2.631,00 €/mes. Estimando que su trabajo semanal es de 40 horas, y por tanto, 160 horas al mes, su coste por hora es también 16,44 €/h.

Entonces, teniendo en cuenta que dedicó una hora diaria a guiarme en el proyecto durante 43 días, su coste en el proyecto se calcula con la fórmula 2.5.

$$\mathit{CosteAnalistaProgramador2} = 16{,}44\,€/\mathrm{h} \times 1\,\mathrm{h/día} \times 43\,\mathrm{días} = 706{,}92\,€ \tag{2.5}$$

Habiendo calculado el coste de todos los recursos humanos, en el cuadro 2.1 se muestra la suma de los costes humanos. A cada coste humano se le añaden los gastos de contratación, que considero que son hipotéticamente un 20 %.

Personal	**Coste en bruto**	**Gastos de contratación (20 %)**	**Coste total**
Jefe de proyecto	499,66 €	99,93 €	599,59 €
Desarrollador *Junior*	3.404,31 €	680,86 €	4.085,17 €
Primer analista programador	2.120,76 €	424,15 €	2.544,91 €
Segundo analista programador	706,92€	141,38 €	848,30 €
			8.077,97 €

Cuadro 2.1: Costes humanos

Para calcular el coste humano total, hay que tener en cuenta los costes indirectos del proyecto. Un ejemplo de coste indirecto puede ser: baja por enfermedad, maternidad/paternidad, etc. He considerado que son también un 20 % del total del coste. En la fórmula se muestra el coste humano total, que es la suma de los costes humanos calculados en el cuadro 2.1 y los costes indirectos.

$$\mathit{CosteHumanoTotal} = \mathit{CosteHumano} + (\mathit{CosteHumano} \times 0{,}2) = 8077{,}97\,€ + 1615{,}59\,€ = 9693{,}56\,€ \tag{2.6}$$

2.4.2. Recursos y costes en infraestructura y servicios

El primer coste en infraestructura y servicios estimado es el alquiler de la oficina donde se realizó este proyecto. Para estimarlo, se utiliza el alquiler de una oficina en el mismo edificio donde se realizaron las prácticas. Este alquiler está publicado en la página de la empresa *idealista* con un coste de 14.375 €/mes y 1.150m² de capacidad [26].

Para prorratear los metros cuadrados usados se utiliza el Real Decreto 486/1997, donde explica que se necesitan: «*2 metros cuadrados de superficie libre por trabajador*» [13]. En este caso, se usan cuatro metros cuadrados por trabajador, cumpliendo el mínimo legal.

El calculo total de los metros cuadrados utilizados en la oficina se muestra en la fórmula 2.7, donde se tiene en cuenta el espacio usado por los cuatro trabajadores.

$$\mathit{SuperficieTotal} = \mathit{SuperfTrabajador} \times 4 = 4\,\mathrm{m^2} \times 4 = 16\,\mathrm{m^2} \tag{2.7}$$

En la página web del alquiler se muestra el coste por metro cuadrado de la oficina, siendo 12,50 €/m² [26]. Gracias a este dato, en la fórmula 2.8 se calcula el coste de la superficie utilizada.

$$CosteSuperficie = SuperficieTotal \times 12{,}50\,€/\mathrm{m}^2 = 200\,€/\mathrm{mes} \tag{2.8}$$

Una vez calculado el coste de la superficie, en la fórmula 2.9 se calcula el coste del alquiler prorrateado a un día de trabajo. En el calculo, se supone que un mes tiene 30 días.

$$CosteDiaAlquiler = 200\,€/\mathrm{mes} \times \frac{1\,\mathrm{mes}}{30\,\mathrm{días}} = 6{,}67\,€/\mathrm{día} \tag{2.9}$$

A continuación, en la fórmula 2.10 se calcula el coste total del alquiler, teniendo en cuenta la fórmula 2.9 y los 43 días de desarrollo.

$$CosteTotalAlquiler = CosteDiaAlquiler \times 43\,\mathrm{días} = 286{,}81\,€ \tag{2.10}$$

El segundo coste en infraestructura y servicios a tener en cuenta es la electricidad. En el calculo de su coste, solo se tiene en cuenta la energía utilizada por todos los recursos.

Según la página *Tarifaluzhora* [37], precio medio de la luz en España en la fecha consultada es de 0,1521 €/kWh.

Con la fórmula 2.11 se calcula la energía en kWh utilizada por un tipo de recurso durante toda la estancia. En ella se tiene en cuenta: p, que es la potencia en kW del recurso; que fueron utilizados siete horas diarias durante 43 días y n, el número de recursos del mismo tipo usados.

$$EnergiaRecurso = \mathrm{p} \times 7\,\mathrm{h/día} \times 43\,\mathrm{días} \times \mathrm{n} \tag{2.11}$$

La fórmula 2.11 se utiliza en las filas de la columna ≪kWh totales≫ del cuadro 2.2, que muestra la energía en kWh utilizada por todos los recursos de un mismo tipo.

Con la fórmula 2.12 se calcula el coste de la energía de los recursos de un mismo tipo. Este calculo se utiliza en la última columna del cuadro 2.2.

$$CosteEnergiaRecurso = EnergiaRecurso \times 0{,}1521\,€/\mathrm{kWh} \tag{2.12}$$

n	**Recurso**	***p*** **(kW)**	**kWh totales**	**Coste**
2	Panel	0,02950 [29]	17,76	2,70 €
4	Ordenador	0,08000 [4]	96,32	14,65 €
4	Monitor	0,01382 [6]	16,64	2,53 €
				19,88 €

Cuadro 2.2: Costes en energía de los recursos

Para calcular el coste de la energía utilizada en la iluminación de la oficina, se supone que utiliza unos paneles led con una potencia de 29,5 W [29]. Esta potencia convertida en kilovatios son 0,0295 kW, dividiendo la potencia del panel entre 1000.

Fueron usados dos paneles led de la oficina para los puestos de trabajo de los cuatro trabajadores.

Los ordenadores utilizados fueron portátiles del modelo *HP PROBOOK 650 G8 I5-1135G7 SYST*. Este portátil tiene una potencia de 80 W (0,08 kW) [4].

El último recurso a considerar en el coste de la energía son los monitores utilizados. Fueron utilizados cuatro monitores *Philips 243V7QDSB*, con una potencia de 13,82 W (0,01382 kW) [6].

El tercer coste en infraestructura y servicios es el internet utilizado. Se tiene en cuenta un plan de la empresa *Orange* con fibra óptica hasta 1 Gb, con un coste de 33,9 €/mes [34].

En la fórmula 2.13 se calcula el coste del internet prorrateado a un día de trabajo. En ella, se tiene en cuenta que un mes tiene 30 días.

$$CosteDiaInternet = 33{,}9\,€/\text{mes} \times \frac{1\,\text{mes}}{30\,\text{días}} = 1{,}13\,€/\text{día} \tag{2.13}$$

Una vez calculado el coste por día del internet, en la fórmula 2.14 se calcula el total, contando los 43 días de trabajo.

$$CosteInternet = CosteDiaInternet \times 43\,\text{días} = 48{,}59\,€ \tag{2.14}$$

El último coste en infraestructura y servicios a tener en cuenta es el mobiliario utilizado por los trabajadores. El mobiliario utilizado se presenta en el cuadro 2.3, donde sus costes están prorrateados a los 43 días realizados.

Mobiliario	**Precio**	**Vida útil (años)**	**Coste prorrateado (43 días)**	**Cantidad**	**Total**
Silla	65,99 € [7]	3	2,59€	4	10,36 €
Mesa	88,99 € [2]	3	3,49 €	4	13,96 €
Papelera	9,99 € [1]	3	0,39 €	4	1,56 €
					25,88 €

Cuadro 2.3: Costes del mobiliario

Los costes en la cuarta columna del cuadro 2.3 son calculados con la fórmula 2.15, donde la variable *precio* coincide con los precios mostrados en la segunda columna. Además, se estima que la vida útil del mobiliario es de 3 años.

$$CosteMobiliarioProrrateado = \frac{precio}{3\,\text{años}} \times \frac{1\,\text{año}}{365\,\text{días}} \times 43\,\text{días} \tag{2.15}$$

Los costes totales mostrados en la última columna del cuadro 2.3 son determinados con la fórmula 2.16. En esta fórmula se multiplica el coste prorrateado de un tipo de mobiliario calculado en la fórmula 2.15 por el total de personas implicadas en el proyecto.

$$CosteMobiliarioProrrateadoTotal = CosteMobiliarioProrrateado \times 4 \tag{2.16}$$

Después de realizar todos los cálculos de los costes en infraestructura y servicios, en el cuadro 2.4 se muestra el coste total de los costes indirectos.

Recurso	Coste
Alquiler	286,81 €
Electricidad	19,88 €
Internet	48,59 €
Mobiliario	25,88 €
	381,16 €

Cuadro 2.4: Costes en infraestructura y servicios

2.4.3. Recursos y costes tecnológicos

Los recursos tecnológicos se dividen en dos tipos: *software* y *hardware*. Para calcular el coste tecnológico total que hubo, se determina el coste de los dos tipos de recursos.

Software

Los recursos *software* usados son todas las tecnologías anteriormente citadas en el apartado 1.3.1. Puesto que son gratuitas, su coste total es 0 €.

Hardware

Los recursos *hardware* son todos los materiales tecnológicos que ofreció la empresa para desarrollar este proyecto. En el cuadro 2.5 se definen todos estos recursos con sus costes prorrateados acorde a los 43 días que realicé la estancia en las prácticas. Además, para calcular el coste prorrateado se estima que la vida útil de los recursos es de 3 años.

Recurso	Modelo	Precio	Vida útil (años)	Coste prorrateado (43 días)	Cantidad	Total
Portatil	*HP PROBOOK 650*	1.066,73 € [4]	3	41,89 €	4	167,56 €
Ratón	*HP 435 MLTDVC WRLS*	37,73 € [3]	3	1,48 €	4	5,92 €
Teclado	*Logitech K120*	13,99 € [5]	3	0,55 €	4	2,20 €
Monitor	*Philips 243V7QDSB*	99,99 € [6]	3	3,93 €	4	15,72 €
						191,40 €

Cuadro 2.5: Costes de *hardware*

Los costes prorrateados en la quinta columna del cuadro 2.5 son calculados a partir de la

fórmula 2.17, donde la variable *precio* coincide con los precios mostrados en la tercera columna.

$$CosteHardwareProrrateado = \frac{precio}{3\,\text{años}} \times \frac{1\,\text{año}}{365\,\text{días}} \times 43\,\text{días} \tag{2.17}$$

Los costes totales de *hardware* mostrados en la última del cuadro 2.5 se calculan mediante la fórmula 2.18. En ella se multiplica el coste prorrateado de estos recursos con la cantidad de unidades compradas, que en este caso fueron cuatro por cada uno. Finalmente, en la última fila aparece la suma de todos los costes de *hardware*.

$$CosteHardwareProrrateadoTotal = CosteHardwareProrrateado \times 4 \tag{2.18}$$

2.4.4. Coste total del proyecto

Tras haber calculado todos los costes, en el cuadro 2.6 se obtiene el coste total prorrateado del proyecto.

Recurso	**Coste**
Humano	8.077,97 €
De infraestructura y servicios	381,16 €
Software	0,00 €
Hardware	191,40 €
	8.650,53 €

Cuadro 2.6: Coste total del proyecto

2.5. Gestión de riesgos

En este apartado se identifican los riesgos del proyecto y sus correspondientes medidas. Esta práctica es crucial porque ayuda a que la planificación se desarrolle de la forma acordada y pueda finalizarse el proyecto.

El primer riesgo es el desconocimiento de las tecnologías utilizadas en el proyecto. La medida para este riesgo es asignar un tiempo suficiente a su estudio, además de la resolución de dudas por los demás compañeros del proyecto.

La baja de un miembro del equipo es un riesgo posible. Una medida adecuada es la contratación temporal de una persona capacitada para el puesto durante el tiempo de la baja.

En tercer lugar, el riesgo de accidente en la oficina. Un ejemplo de accidente podría ser la inundación de la oficina por una rotura de las tuberías o un incendio. Su solución es la opción de teletrabajo para todos los empleados.

Otro riesgo a tener en cuenta son los posibles errores de *hardware* y *software* en los equipos. Este riesgo supondría un retraso en el proyecto de varios días, ya que tendría que solicitarse

un nuevo equipo que es enviado desde la oficina de Sopra Steria en Madrid. Además, al nuevo equipo habría que instalarle todas las tecnologías necesarias. La única solución a este problema es esperar a la llegada de un nuevo equipo, ya que legalmente solo se puede trabajar en estos proyectos con portátiles de la empresa. Otra medida que rebajaría la gravedad de este riesgo es el guardado diario del código desarrollado, evitando que se pierda.

Siguiendo con los riesgos, se encuentra el retraso en la creación de los puntos finales de la *API REST*. Este es un problema en la fase de implementación, porque al no estar listos no es posible utilizar la información que llegan de estos. Una posible solución es simular la llegada de la información de estos puntos finales creándola de forma local, de esta manera, puedo comprobar que ciertas funcionalidades implementadas en el proyecto funcionen, como por ejemplo verificar que se muestren subexplotaciones en la consulta.

El penúltimo riesgo es un cambio en los requisitos de la consulta de explotaciones por parte de la DGTIC. Este cambio de requisitos podría aumentar la carga de trabajo en el proyecto, dificultando seguir la planificación acordada. Para solucionarlo entra en juego el jefe de proyecto, que se reuniría con ellos para mediar una solución y procurar que este cambio afecte lo menos posible al desarrollo del proyecto.

El último riesgo es un ataque cibernético a la empresa. Dependiendo de la gravedad del ataque podría hasta parar completamente la evolución del proyecto. La solución al respecto es notificar el ataque lo antes posible a los expertos en ciberseguridad de la empresa para que establezcan medidas y minimicen los daños ocasionados. Otra solución para evitar que ocurra es impartir cursos de seguridad informática a los trabajadores para establecer unas normas que favorezcan la seguridad de sus equipos informáticos.

Capítulo 3

Análisis del sistema

3.1. Diagrama de casos de uso

En la figura 3.1 se muestra un diagrama *UML* de casos de uso. En él se identifican los tres actores que acceden a la consulta: titular, veterinario ADSG y administrador. El cuarto actor ≪usuario≫ se ha creado para simplificar el diagrama, ya que está asociado a los casos de uso que comparten los tres actores que acceden a la consulta. Además, se muestran los casos de uso, que describen las acciones que realiza el sistema para producir un resultado para un actor específico [14].

En los cuadros 3.1, 3.2. 3.3. 3.4, 3.5, 3.6 y 3.7 se detallan los casos de uso de la figura 3.1.

3.2. Requisitos de datos

En este apartado se muestran en los cuadros 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14 y 3.15 los requisitos de datos del proyecto, de entrada y salida. Los de entrada identifican qué información introduce el usuario mediante el uso de los bloques de filtros y los de salida qué información proveniente de la parte servidor de la *API REST* se muestra en la interfaz de la consulta de explotaciones.

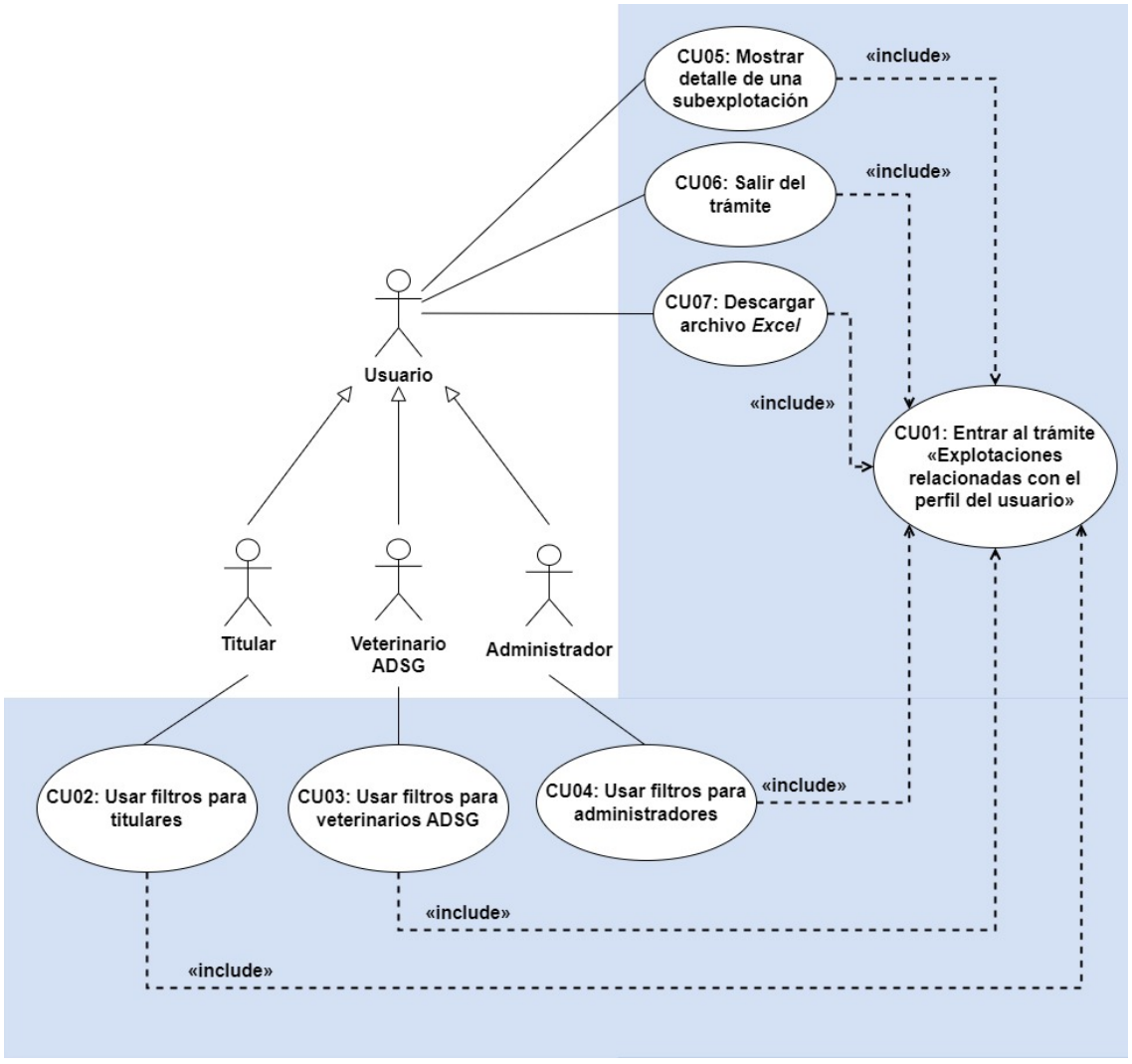


Figura 3.1: Diagrama de casos de uso.

Identificador	CU01
Nombre	Entrar al trámite ≪Explotaciones relacionadas con el perfil del usuario≫
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	Usuario
Descripción	Permite al usuario acceder al trámite ≪Explotaciones relacionadas con el perfil del usuario≫.
Precondición	-
Pasos secuencia normal	1. El usuario selecciona el apartado ≪Consultas≫. 2. El sistema muestra los distintos tipos de trámites dentro del apartado ≪Consultas≫. 3. El titular selecciona el trámite ≪Explotaciones relacionadas con el perfil del usuario≫. 4. El sistema muestra la interfaz de la consulta, que contiene un cuadro donde se muestra una fila por cada subexplotación asociada con la explotación que se ha conectado el usuario.

Cuadro 3.1: Descripción del caso de uso CU01.

Identificador	CU02
Nombre	Usar filtros para titulares
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	Titular
Descripción	Permite al usuario con el rol de titular utilizar el bloque de filtros exclusivo para titulares.
Precondición	-
Pasos secuencia normal	1. El sistema identifica que el usuario conectado es un titular. 2. El sistema muestra la interfaz de la consulta de explotaciones con el bloque de filtros exclusivo para un usuario titular. 3. El titular selecciona datos en las listas desplegables dentro del bloque de filtros. 4. El titular presiona el botón ≪Aplicar≫. 5. El sistema muestra las subexplotaciones relacionadas con los datos introducidos dentro de las listas desplegables.

Cuadro 3.2: Descripción del caso de uso CU02.

Identificador	CU03
Nombre	Usar filtros para veterinarios ADSG
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	veterinario ADSG
Descripción	Permite al usuario con el rol de veterinario ADSG utilizar el bloque de filtros exclusivo para veterinarios ADSG.
Precondición	-
Pasos secuencia normal	1. El sistema identifica que el usuario conectado es un veterinario ADSG. 2. El sistema muestra la interfaz de la consulta de explotaciones con el bloque de filtros para un veterinario ADSG. 3. El veterinario ADSG selecciona datos en las listas desplegables dentro del bloque de filtros. 4. El veterinario ADSG presiona el botón ≪Aplicar≫. 5. El sistema muestra las subexplotaciones relacionadas con los datos introducidos dentro de las listas desplegables.

Cuadro 3.3: Descripción del caso de uso CU03.

Identificador	CU04
Nombre	Usar filtros para administradores
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	Administrador
Descripción	Permite al usuario con el rol de administrador utilizar el bloque de filtros para administradores.
Precondición	-
Pasos secuencia normal	1. El sistema identifica que el usuario conectado es un administrador. 2. El sistema muestra la interfaz de la consulta de explotaciones con el bloque de filtros para un administrador. 3. El administrador selecciona datos en las listas desplegables dentro del bloque de filtros. 4. El administrador presiona el botón ≪Aplicar≫. 5. El sistema muestra las subexplotaciones relacionadas con los datos introducidos dentro de las listas desplegables.

Cuadro 3.4: Descripción del caso de uso CU04.

Identificador	CU05
Nombre	Mostrar detalle de una subexplotación
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	Usuario
Descripción	Permite al usuario consultar información adicional de una subexplotación.
Precondición	Solo podrá acceder al detalle si existe al menos una subexplotación asociada.
Pasos secuencia normal (al entrar al trámite)	1. El usuario selecciona un botón con el símbolo ≪+≫, ubicado en la última columna de cualquier fila del cuadro donde se muestran las subexplotaciones asociadas con la explotación que se ha conectado el usuario. 2. El sistema muestra una nueva interfaz con información adicional de la subexplotación consultada.
Pasos secuencia normal (después de usar los filtros)	1. El usuario selecciona datos en las listas desplegables dentro del bloque de filtros. 2. El usuario presiona el botón ≪Aplicar≫. 3. El sistema muestra en un cuadro las subexplotaciones relacionadas con los datos introducidos dentro de los filtros. 4. El usuario selecciona un botón con el símbolo ≪+≫, ubicado en la última columna de cualquier fila del cuadro. 5. El sistema muestra una nueva interfaz con información adicional de la subexplotación consultada.

Cuadro 3.5: Descripción del caso de uso CU05.

Identificador	CU06
Nombre	Salir del trámite
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	Usuario
Descripción	Permite al usuario salir del trámite ≪Explotaciones relacionadas con el perfil del usuario≫.
Pasos secuencia normal	1. El usuario selecciona el botón ≪Cancelar≫. 2. El sistema redirige el usuario a la pantalla del apartado ≪Consultas≫.

Cuadro 3.6: Descripción del caso de uso CU06.

Identificador	CU07
Nombre	Descargar archivo *Excel*
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	21/05/23
Fecha de revisión	02/06/23
Actor/es	Usuario
Descripción	Permite al usuario tener la información de las subexplotaciones mostradas en la consulta en un archivo de tipo *Excel*.
Precondición	-
Pasos secuencia normal	1. El usuario selecciona el botón ≪Descargar *Excel*≫. 2. El sistema crea y descarga un fichero *Excel* con la información de las subexplotaciones mostradas en la consulta.

Cuadro 3.7: Descripción del caso de uso CU07.

Identificador	RD01
Nombre	Datos de entrada en el bloque de filtros para titulares
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de entrada
Descripción	Identificadores que designan las opciones seleccionadas en las listas desplegables del bloque de filtros para titulares.
Datos	Identificador/es del tipo de explotación, identificador/es de la/s especie/s de las subexplotaciones, identificador/es de la/s clasificación/es zootécnica/s de las subexplotaciones e identificador del estado de la subexplotación.
Detalle	-
Comentarios	Los identificadores de las opciones seleccionadas se mandan desde la interfaz del usuario a la parte servidor de la *API REST*, encapsulados en un *DTO* al presionar el botón ≪Aplicar≫ del bloque de filtros. Pueden seleccionarse más de una opción en los filtros: tipo de explotación, especie y clasificación zootécnica. Por ello, se indica en la fila ≪Datos≫ de este cuadro que pueden ser uno o varios.

Cuadro 3.8: Requisito de datos RD01

Identificador	RD02
Nombre	Datos de salida en el bloque de filtros para titulares
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de salida
Descripción	Información mostrada dentro de las listas desplegables en el bloque de filtros para titulares.
Datos	Tipo/s de explotación/es, especie/s, clasificación/es zootécnica/s y estados.
Detalle	-Estados: contendrá tres valores fijos: alta, baja e inactiva.
Comentarios	Los datos provienen de la parte servidor de la *API REST*, encapsulados en un *DTO* al entrar por primera vez al trámite. Las siguientes listas actúan como una barra de búsqueda en el caso de tener seis o más opciones: tipo/s de explotación/es, especie/s y clasificación/es zootécnica/s.

Cuadro 3.9: Requisito de datos RD02

Identificador	RD03
Nombre	Datos de entrada en el bloque de filtros para veterinarios ADSG
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de entrada
Descripción	Identificadores que designan las opciones seleccionadas en las listas desplegables del bloque de filtros para veterinarios ADSG.
Datos	Código REGA de la explotación, identificador de la ADSG, identificador/es del tipo de explotación, DNI del titular de la explotación, identificador/es de la/s especie/s de las subexplotaciones, identificador/es de la/s clasificación/es zootécnica/s de las subexplotaciones e identificador del estado de la subexplotación.
Detalle	-
Comentarios	Los identificadores de las opciones seleccionadas se mandan desde la interfaz del usuario a la parte servidor de la *API REST*, encapsulados en un *DTO* al presionar el botón ≪Aplicar≫ del bloque de filtros. Pueden seleccionarse más de una opción en los filtros: tipo de explotación, especie y clasificación zootécnica. Por ello, se indica en la fila ≪Datos≫ de este cuadro que pueden ser uno o varios. El dato ≪Código REGA de la explotación≫ puede tener el valor ≪TODAS LAS DE MI PERFIL≫, que indica que se muestren las subexplotaciones de todas las explotaciones asociadas a las ADSG a las que pertenezca el veterinario ADSG.

Cuadro 3.10: Requisito de datos RD03

Identificador	RD04
Nombre	Datos de salida en el bloque de filtros para veterinarios ADSG
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de salida
Descripción	Información mostrada dentro de las listas desplegables en el bloque de filtros para los veterinarios ADSG.
Datos	Código REGA de la explotación, código/s de la/s ADSG, nombre/s de la/s ADSG, tipo/s de explotación/es, el/los DNI del veterinario ADSG, especie/s, clasificación/es zootécnica/s y estados.
Detalle	-Estados: contendrá tres valores fijos: alta, baja e inactiva.
Comentarios	Los datos provienen de la parte servidor de la *API REST*, encapsulados en un *DTO* al entrar por primera vez al trámite. Las siguientes listas actúan como una barra de búsqueda en el caso de tener seis o más opciones: tipo/s de explotación/es, especie/s y clasificación/es zootécnica/s. Por otro lado, la lista con los DNI del veterinario ADSG actúa siempre como un campo de búsqueda. El filtro del dato ≪Código REGA de la explotación≫ contendrá ≪TODAS LAS DE MI PERFIL≫ como una de sus opciones. Seleccionarla indica que se muestren todas las subexplotaciones asociadas a las ADSG a las que pertenezca el veterinario ADSG.

Cuadro 3.11: Requisito de datos RD04

Identificador	RD05
Nombre	Datos de entrada en el bloque de filtros para administradores
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de entrada
Descripción	Identificadores que designan las opciones seleccionadas en las listas desplegables del bloque de filtros para administradores.
Datos	Identificador de la ADSG, identificador/es del/de los tipo/s de explotación/es, DNI del titular de la explotación, identificador/es de la/s especie/s de las subexplotaciones, identificador/es de la/s clasificación/es zootécnica/s de las subexplotaciones, identificador del estado de la subexplotación, DNI del veterinario responsable, DNI del veterinario habilitado y DNI del veterinario ADSG.
Detalle	-
Comentarios	Los identificadores de las opciones seleccionadas se mandan desde la interfaz del usuario a la parte servidor de la *API REST*, encapsulados en un *DTO* al presionar el botón ≪Aplicar≫ del bloque de filtros. Pueden seleccionarse más de una opción en los filtros: tipo de explotación, especie y clasificación zootécnica. Por ello, se indica en la fila ≪Datos≫ de este cuadro que pueden ser uno o varios.

Cuadro 3.12: Requisito de datos RD05

Identificador	RD06
Nombre	Datos de salida en el bloque de filtros para administradores
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de salida
Descripción	Información mostrada dentro de las listas desplegables en el bloque de filtros para los administradores.
Datos	Código/s de la/s ADSG, nombre/s de la/s ADSG, tipo/s de explotación/es, el/los DNI del titular, especie/s, clasificación/es zootécnica/s, estados, el/los DNI del veterinario responsable, el/los DNI del veterinario habilitado y el/los DNI del veterinario ADSG.
Detalle	-Estados: contendrá tres valores fijos: alta, baja e inactiva.
Comentarios	Los datos provienen de la parte servidor de la *API REST*, encapsulados en un *DTO* al entrar por primera vez al trámite. Las siguientes listas actúan como una barra de búsqueda en el caso de tener seis o más opciones: tipo/s de explotación/es, especie/s y clasificación/es zootécnica/s. Por otro lado, las cuatro listas con los DNI actúan siempre como un campo de búsqueda.

Cuadro 3.13: Requisito de datos RD06

Identificador	RD07
Nombre	Subexplotaciones relacionadas con el usuario
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de salida
Descripción	Información mostrada en el cuadro donde se muestran las subexplotaciones consultadas.
Datos	Código REGA de la explotación, titular, especie, estado, calificaciones, tipo, municipio, censo REGA, plazas REGA y censo RIIA.
Detalle	-
Comentarios	Los datos provienen de la parte servidor de la *API REST*, encapsulados en un *DTO*.

Cuadro 3.14: Requisito de datos RD07

Identificador	RD08
Nombre	Detalle de una subexplotación
Versión	1.0
Autor	Jorge Revaliente Lainez
Fecha de creación	23/05/23
Fecha de revisión	02/06/23
Fuente	Consulta de explotaciones
Tipo	Datos de salida
Descripción	Información adicional sobre una subexplotación.
Datos	Código REGA de la explotación, nombre de la explotación, especie, ADSG, nombre de la ADSG, provincia, responsable sanitario, clasificación zootécnica, censo reproductores, plazas reproductores, censo cebo, plazas cebo, categoría, censo y plazas.
Detalle	-
Comentarios	Los datos provienen de la parte servidor de la *API REST* encapsulados en un *DTO*. Los datos categoría, censo y plazas aparecen en un cuadro similar al de la consulta de explotaciones.

Cuadro 3.15: Requisito de datos RD08

Capítulo 4

Diseño del sistema

4.1. Diseño de la arquitectura del sistema

La arquitectura usada en la consulta de explotaciones es la arquitectura cliente-servidor. Esta arquitectura se compone por dos partes: el cliente, que es el encargado de enviar peticiones, y el servidor, que recibe las peticiones y devuelve una respuesta al cliente [30].

En este proyecto, el cliente es un usuario accediendo a la consulta mediante un navegador web, mientras que el servidor de aplicaciones es la plataforma donde está alojada la consulta y la parte servidor de la *API REST*.

En la figura 4.1 se representa de manera gráfica la arquitectura del sistema, con los pasos que determinan su funcionamiento sin entrar en detalles de la implementación.

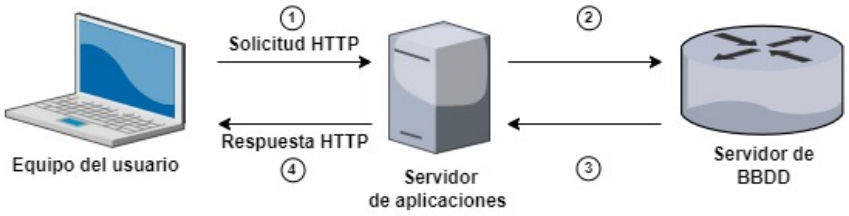


Figura 4.1: Arquitectura del sistema.

A continuación, se explican los pasos de la figura 4.1 cuando el usuario accede al trámite de la consulta de explotaciones: en primer lugar, el usuario ejecuta una solicitud *HTTP* accediendo mediante un navegador web al trámite. Después, el servidor de aplicaciones recibe la solicitud *HTTP* y la envía a la aplicación de la consulta de explotaciones, que la gestiona llamando a los puntos finales de la parte servidor de la *API REST* para conseguir las subexplotaciones pertinentes. Una vez los puntos finales reciben las solicitudes, la parte servidor de la *API REST* llama al servidor de base de datos para conseguir las subexplotaciones relacionadas con la explotación conectada por el usuario. En el tercer paso, el servidor de base de datos realiza

las consultas en lenguaje *SQL* necesarias y envía la información resultante a la parte servidor de la *API REST*. Una vez la parte servidor consigue la información, la entrega dentro de una respuesta *HTTP* a la parte cliente de la *API REST*. Por último, la aplicación de la consulta de explotaciones envía una respuesta *HTTP* al usuario y visualiza la interfaz de la consulta de explotaciones.

4.2. Diseño de la arquitectura de *software*

El patrón de diseño de *software* utilizado en este proyecto es el patrón Modelo-Vista-Controlador (MVC) [10]. Este patrón divide la aplicación en tres módulos independientes:

- Modelo: contiene los datos de la aplicación.
- Vista: incluye los componentes que son mostrados mediante una interfaz de usuario.
- Controlador: es el componente encargado de gestionar las interacciones que se realizan en la interfaz de usuario.

El módulo *Spring MVC* del *framework Spring* implementa este patrón de la siguiente manera:

- Modelo: está representada con la clase *Model* de *Spring*, que es la encargada de enviar datos a la vista. De los datos que se mandan, destacan los *DTO*.
- Vista: utiliza el motor de plantillas *Thymeleaf*, que permite mostrar los componentes *HTML* y los datos mandados al modelo en la interfaz de usuario.
- Controlador: son todas las clases del proyecto que contengan la anotación *@Controller*. Estas clases son las encargadas de gestionar las peticiones *HTTP*.

4.3. Definición de los puntos finales de la *API REST*

En el cuadro 4.1 se muestran los puntos finales de la parte servidor de la *API REST* utilizados en la consulta. Estos puntos finales son similares a los que definí en la fase de diseño, tarea que realicé para posteriormente crear la representación en diagramas de clases de la respuesta de cada punto final. Estas representaciones se muestran en el anexo A.

Existen tres columnas en el cuadro 4.1: la primera define el método *HTTP* que se realiza en cada punto final, la segunda muestra la ruta o *path* de la *URI* que identifica cada punto final [12] y la última, la descripción de cada punto.

4.4. Diseño de las interfaces

En este apartado se van a mostrar el diseño de la vista del proyecto. Por ello, va a dividirse en dos subapartados. El primero va a mostrar el diseño de la vista principal del proyecto y el segundo, el de la vista que muestra el detalle de una subexplotación.

4.4.1. Vista principal

En la figura 4.2 se muestra una captura de pantalla de la vista principal del proyecto. Esta vista principal contiene varios componentes que se detallan a continuación. Algunos de ellos se indican con un recuadro azul y un número.

Método HTTP	Punto final	Descripción
POST	paginaSubexplotaciones/ {pagina}/{size}/ {direction}/{columna}	Obtiene una página de las subexplotaciones que cumplen con los filtros de búsqueda introducidos por el usuario.
POST	subexplotacionesExcelCategorias/	Obtiene las subexplotaciones, con el detalle de las categorías, consultadas por el usuario para exportarlas en un archivo *Excel*.
POST	subexplotacionesExcel/	Obtiene las subexplotaciones, sin el detalle de las categorías, consultadas por el usuario para exportarlas en un archivo *Excel*.
GET	explotaciones/	Obtiene las explotaciones que tiene acceso el usuario.
GET	tiposExplotaciones/	Obtiene los tipos de las explotaciones accesibles por el usuario.
GET	titulares/	Obtiene los titulares de las explotaciones que tenga acceso el usuario.
GET	especies/	Obtiene las especies de las subexplotaciones accesibles por el usuario.
GET	clasificacionesZootecnicas/	Obtiene las clasificaciones zootécnicas de las subexplotaciones accesibles por el usuario.
GET	veterinariosResponsables/	Obtiene los veterinarios responsables de las explotaciones a las que tiene acceso el usuario.
GET	veterinariosHabilitados/	Obtiene los veterinarios habilitados de las explotaciones a las que tiene acceso el usuario.
GET	veterinariosADSG/	Obtiene los veterinarios ADSG de las ADSG a las que pertenezcan las subexplotaciones accesibles por el usuario.
GET	adsgVeterinariosADSG/	Obtiene las ADSG a las que está asociada el veterinario ADSG.
GET	adsgAdministrador/	Obtiene las ADSG asociadas a las subexplotaciones accesibles por el administrador.
GET	detalleSubexplotacion/ {idSubexp}	Obtiene los datos con el detalle de la subexplotación asociada al identificador ≪idSubexp≫.
GET	estados/	Obtiene los tres posibles estados de una subexplotación.

Cuadro 4.1: Puntos finales de la *API REST*

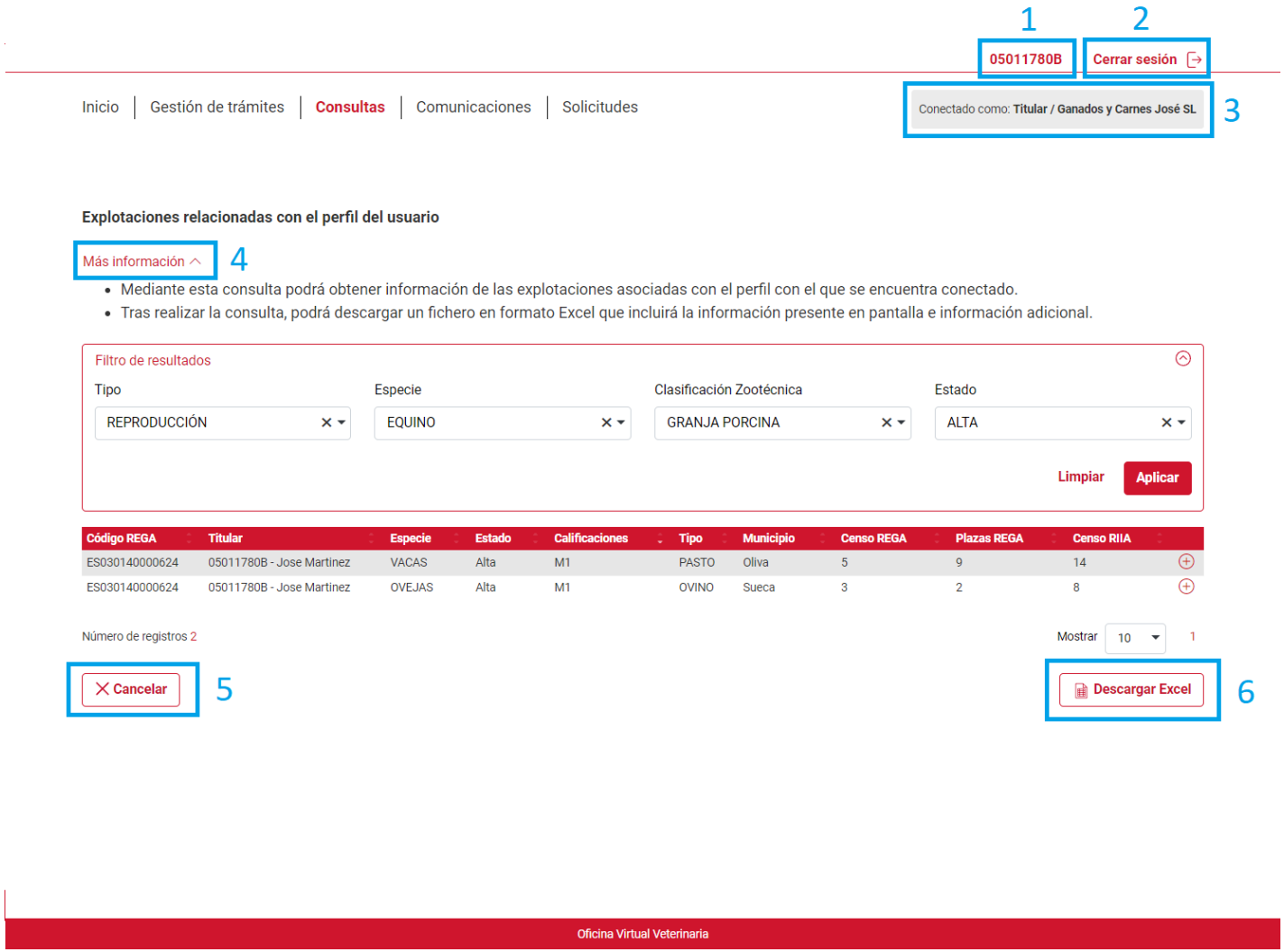


Figura 4.2: Vista principal del proyecto.

El recuadro con el número uno incluye el DNI del usuario conectado.

El segundo es un botón que contiene el texto ≪Cerrar sesión≫. Al presionar este botón se cierra la sesión y se redirige al usuario a la pantalla de inicio de sesión.

Siguiendo con el número tres, es un cuadro donde se muestra el tipo del usuario y el nombre de la explotación con la que se ha conectado.

El cuarto recuadro indica una ventana desplegable con información sobre el trámite. El contenido puede abrirse y cerrarse haciendo clic en el botón ≪Más información≫. Además, justo arriba de la ventana se incluye un título con el nombre del trámite.

El quinto es el botón ≪Cancelar≫ y el sexto el botón ≪Descargar Excel≫. Al hacer clic en el botón ≪Cancelar≫ se redirige al usuario a la pantalla con las consultas disponibles en la OVV.

Por otro lado, al hacer clic en el botón ≪Excel≫ primero aparece una ventana de confirmación, que está mostrada en la figura 4.3. Esta ventana pregunta al usuario si quiere que el *Excel* descargado contenga las subexplotaciones con o sin la información de sus categorías. Además la ventana contiene dos botones, un botón ≪Si≫, para confirmar la pregunta anterior y un botón ≪No≫ para denegarla.

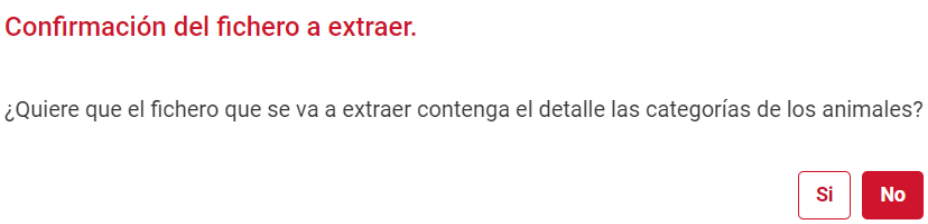
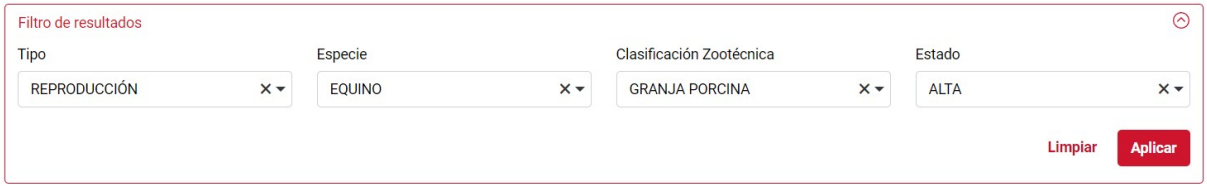


Figura 4.3: Ventana de confirmación para descargar el *Excel*.

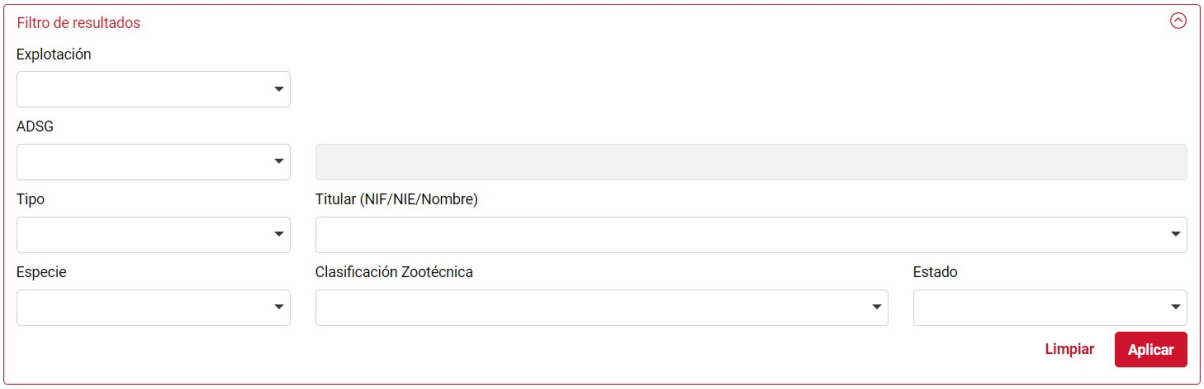
El componente mostrado en la figura 4.4a es el bloque de filtros para los titulares. Este componente cambia según el tipo de usuario conectado, que muestra un tipo de bloque u otro. También existe el bloque de filtros para los veterinarios ADSG en la figura 4.4b y los administradores en la figura 4.4c.

4.4.2. Vista con el detalle de una subexplotación

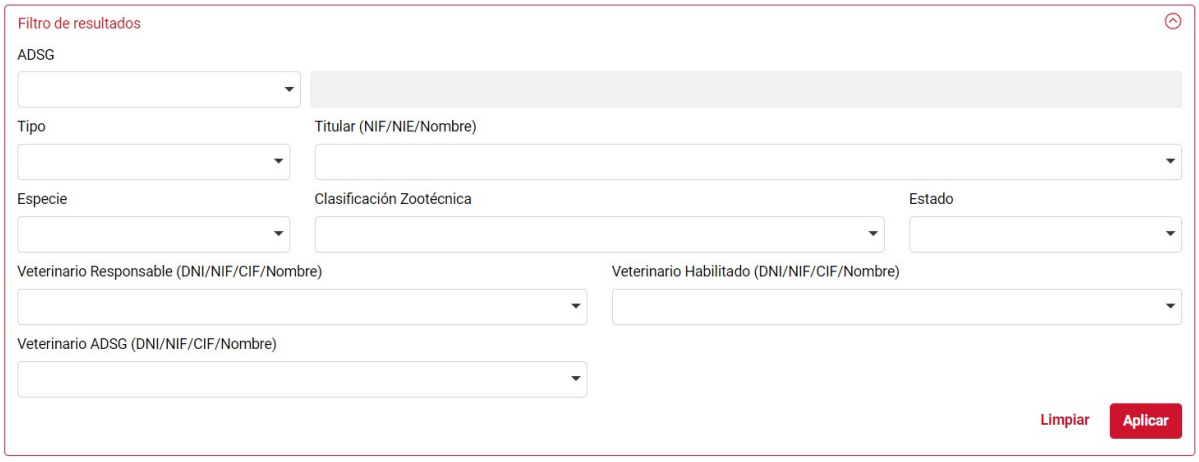
En la figura 4.5 aparece la vista que muestra el detalle de una subexplotación. Esta vista contiene varios componentes que van a explicarse a continuación.



(a) Bloque de filtros para los titulares.



(b) Bloque de filtros para los veterinarios ADSG.



(c) Bloque de filtros para los administradores

Figura 4.4: Bloques de filtros en la vista principal.

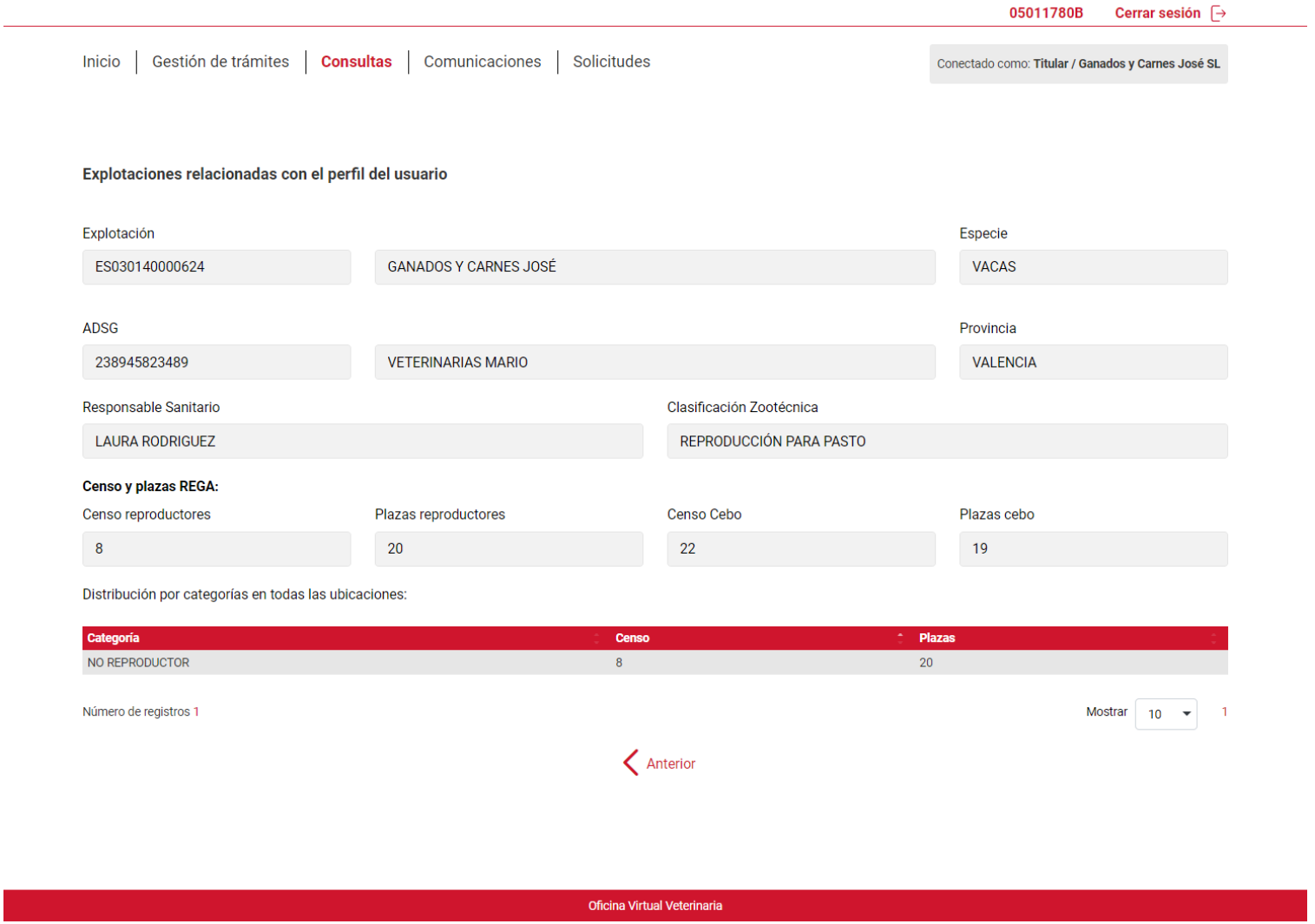


Figura 4.5: Vista con el detalle de una subexplotación.

Los componentes específicos de esta vista son los siguientes. El primero es el grupo de componentes en la figura 4.6. Este grupo de componentes muestra información adicional sobre una subexplotación mostrada en el componente *DataTables* de la vista principal. El único componente que se diferencia en este grupo es un componente *DataTables* que muestra información sobre las categorías de la subexplotación.

Figura 4.6: Grupo de componentes que muestra información sobre una subexplotación.

El siguiente componente específico de esta vista es el botón «Anterior». Al hacer clic en este botón se vuelve a la vista principal.

Capítulo 5

Implementación y pruebas

5.1. Estructura del código

La estructura del proyecto está creada con la tecnología *Maven*, en concreto, utilizando un proyecto multimódulo. Un proyecto multimódulo consiste en un proyecto principal con varios módulos, que son proyectos definidos en el archivo *POM* del proyecto principal y que se ejecutan en grupo [47].

El código del proyecto está dividido entre los distintos módulos, según la función de cada uno. A continuación, se definen los módulos del proyecto:

- Módulo *DTO*: contiene definidas todas las clases *DTO*.
- Módulo *client*: contiene la implementación del cliente de la *API REST*.
- Módulo *service-api*: contiene las interfaces de los servicios que designan la lógica de negocio de la aplicación.
- Módulo *service-impl*: contiene las implementaciones de las interfaces del módulo *service-api*.
- Módulo *application*: contiene todos los componentes de la capa vista, la clase controladora, el fichero *JavaScript* y varios ficheros de configuración del proyecto.

En la figura 5.1 se muestran los módulos citados anteriormente. En ella aparecen los dos componentes que residen en el servidor de aplicaciones, estos son: la consulta de explotaciones creada en este proyecto y la parte servidor de la *API REST*.

Cada módulo al ser un proyecto, contiene un archivo *POM* con su información. En este fichero están definidas las dependencias a los otros módulos del proyecto para así poder compilarse y ejecutarse correctamente.

Estas dependencias están señaladas mediante flechas dentro del componente «INTERFAZ DE USUARIO DE LA CONSULTA DE EXPLOTACIONES» en la figura 5.1, donde aparecen los módulos descritos en este subapartado.

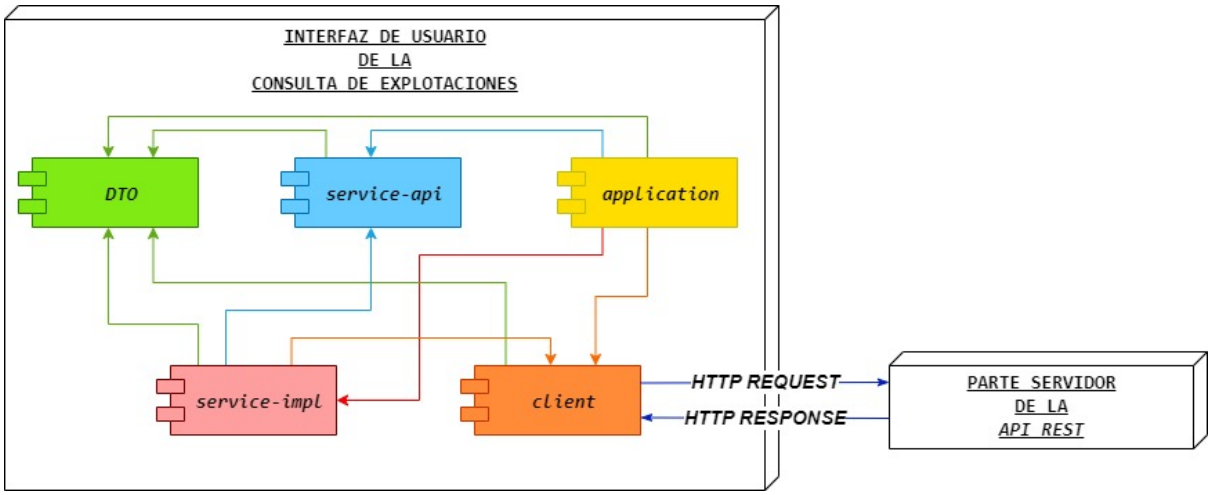


Figura 5.1: Módulos del proyecto.

A continuación se detallan las dependencias entre los módulos:

- Dependencias de *client*:
  - A la parte servidor de la *API REST*: el módulo *client* contiene el cliente de la *API REST*, que realiza las solicitudes *HTTP* a los puntos finales de la parte servidor de la *API REST*.
  - A *DTO*: almacenamiento de los datos dentro del cuerpo de la respuesta *HTTP* capturada en el cliente de la *API REST* en un *DTO*.
- Dependencias de *service-api*:
  - A *DTO*: utilización de los *DTO* en los métodos de las interfaces que contienen la lógica de negocio.
- Dependencias de *service-impl*:
  - A *service-api*: implementación de las interfaces con la lógica de negocio.
  - A *DTO*: utilización de los *DTO* por parte de la implementación de la lógica de negocio.
  - A *client*: acceso al cliente de la *API REST* para obtener los datos necesarios en la implementación de la lógica de negocio.
- Dependencias de *application*:
  - A *client*: desde la capa *application* se configura el cliente de la *API REST* en la capa *client*, proporcionándole desde un fichero de configuración: la *URI* del servidor de la *API REST* y el tiempo de espera máximo de conexión al servidor.

- A *service-api* y *service-impl*: la capa controladora dentro del módulo *application* necesita acceso a la lógica de negocio.

- A *DTO*: se produce un intercambio de información entre la capa de vista incluida en el módulo *application* y la lógica de negocio, que le proporciona los *DTO* con información para mostrar en la vista.

5.2. Descripción técnica de la implementación

En este apartado se describe con detalle cuál es el funcionamiento de cada uno de los módulos del proyecto y de sus ficheros de código. Los ficheros de código están escritos en los lenguajes de programación descritos en el apartado 1.3.1. Además, se muestran diagramas de clases de las clases *Java* más importantes del proyecto.

5.2.1. Módulo *DTO*

Un *DTO* es un objeto de transferencia de datos utilizado exclusivamente para almacenar información. En este proyecto, los *DTO* están escritos en clases *Java*. Para que una clase se considere un *DTO*, debe implementar obligatoriamente la interfaz *Serializable* de *Java*. Esta interfaz permite a *Java* convertir un objeto en una secuencia de bytes, de esta manera puede ser enviado por la red y posteriormente ser reconstruido en el objeto original una vez llegue a su destino [35] [15].

Por otro lado, los *DTO* fueron creados a partir de los cuadros con los requisitos de datos mostrados en el subapartado 3.2.

En el proyecto existen dos tipos de *DTO*:

- ***DTO* de entrada:** encapsulan los datos introducidos por el usuario desde la interfaz gráfica. En concreto, guardan los identificadores de las opciones elegidas dentro de las listas desplegables en los bloques de filtros.

- ***DTO* de salida:** son utilizados para almacenar la información mostrada en los componentes de la interfaz gráfica. Se dividen en dos grupos según su función: los encargados de encapsular los datos provenientes de la *API REST* y los que actúan como un modelo para la vista, los cuales denominaré *DTO REST* y *DTO MV* (*Model View*), respectivamente.

Los dos tipos de *DTO* heredan la clase *BaseDTO*, que almacena información de cada solicitud o respuesta *HTTP*. Tiene como atributos: *mensaje*, con el mensaje ejecutado en la llamada *HTTP*; y *ok*, un atributo *Boolean* que identifica si la llamada ha sido exitosa, el cual es por defecto *true*. La clase *BaseDTO* también implementa la interfaz *Serializable* para cumplir el requisito de *DTO*. En la figura 5.2 aparece un diagrama de clases de este *DTO*.

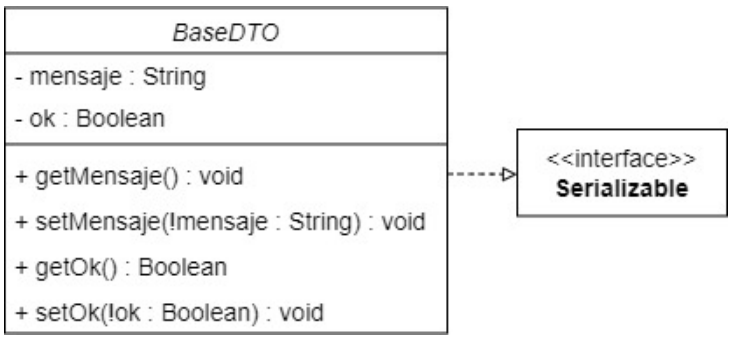


Figura 5.2: Diagrama de clases del *DTO BaseDTO*.

DTO de entrada

Los *DTO* de entrada heredan de la clase *AbstractDTOEntrada*, mostrada en la figura 5.3, que almacena en sus atributos los identificadores de las opciones elegidas dentro de las listas desplegables comunes a los tres tipos de filtros. Sus atributos son de dos tipos: *String*, cuando solo almacena un identificador y una lista de la interfaz de *Java* `List<String>`, cuando almacena más de una.

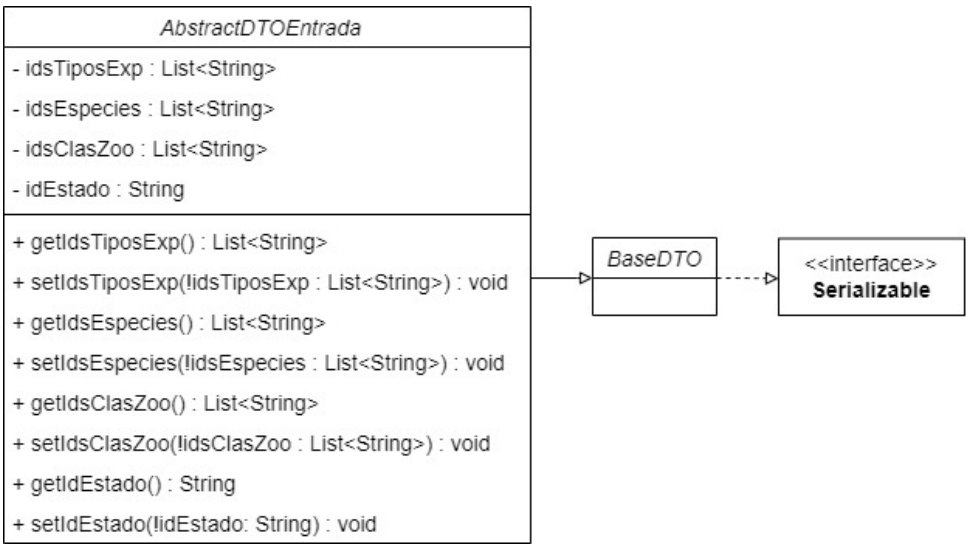


Figura 5.3: Diagrama de clases del *DTO AbstractDTOEntrada*.

En la figura 5.4 se muestra el diagrama de clases del *DTO* de entrada *DTOEntradaTitulares*. Este *DTO* almacena los identificadores de las opciones elegidas en las cuatro listas desplegables del bloque de filtros para titulares, que coinciden con las listas comunes a los tres tipos de filtros. Por tanto, todos sus atributos son heredados de la clase *AbstractDTOEntrada*.

Todas las clases *DTO* del proyecto contienen un método *Getter* y *Setter* por cada uno de sus atributos. El método *Getter* permite conseguir el valor almacenado en el atributo y el *Setter* igualarlo a otro valor.

Figura 5.4: Diagrama de clases del *DTO* de entrada *DTOEntradaTitulares*.

En la figura 5.5 se muestra el diagrama de clases del *DTO* de entrada *DTOEntradaVetADSG*. *DTO* que almacena los identificadores de las opciones elegidas en las listas del bloque de filtros para veterinarios ADSG. Tiene tres atributos no comunes a los demás filtros: *codigoExplotacion*, el código de la explotación elegida; *codigoADSG*, el código de la ADSG elegida; y *titular*, el DNI del titular elegido. Los demás atributos son heredados de *AbstractDTO*Entrada.

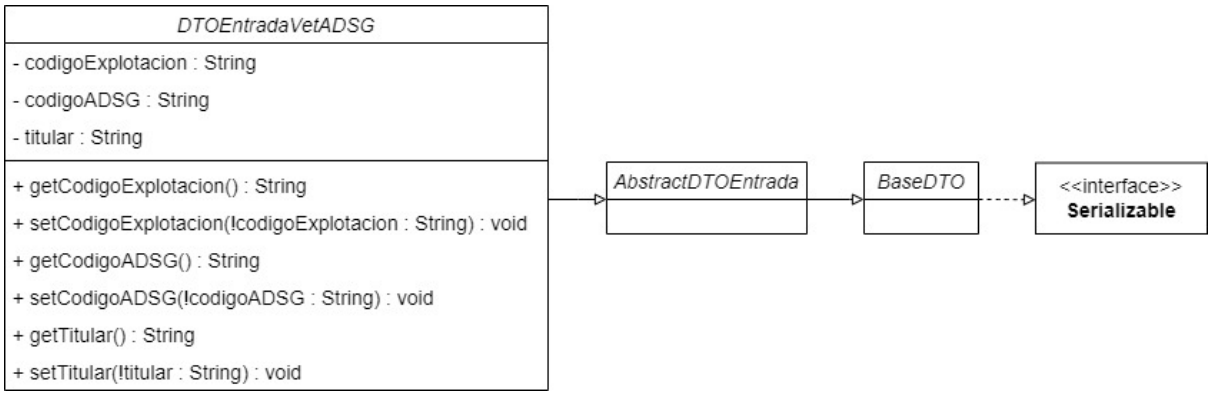


Figura 5.5: Diagrama de clases del *DTO* de entrada *dtoEntradaVetADSG*.

Por último, se presenta en la figura 5.6 el último diagrama de clases del *DTO* de entrada *DTOEntradaAdministradores*. *DTO* que almacena los identificadores de las opciones elegidas en las listas del bloque de filtros para los administradores. Contiene cinco atributos no comunes a los demás *DTO* de entrada: *codigoADSG*, atributo que almacena el código ADSG elegido; *titular*, el DNI del titular elegido; *dniVetResponsable*, el DNI del veterinario responsable elegido; *dniVetHabilitado*, el DNI del veterinario habilitado elegido y *dniVetADSG*, con el DNI del veterinario ADSG elegido. Los atributos no comunes son heredados de la clase *AbstractDTOEntrada*.

DTO de salida

En este apartado, se muestran en diagramas de clases todos los *DTO* de salida del proyecto, que se dividen en dos partes: *DTO REST* y *DTO MV*.

Los datos que se almacenan en los *DTO REST* se consiguen haciendo llamadas a los puntos finales de la *API REST*. Por ejemplo, en la figura 5.7 se muestra el diagrama de clases del *DTO REST EspecieDTO*. Este *DTO REST* almacena la respuesta del punto final ≪/especies≫, que devuelve las especies que tengan las subexplotaciones a las que tiene acceso el usuario.

Por otro lado, los *DTO MV* tienen los datos que se muestran en los componentes de la interfaz gráfica, los cuales son conseguidos de los *DTO REST*. Existen cinco *DTO MV* en el

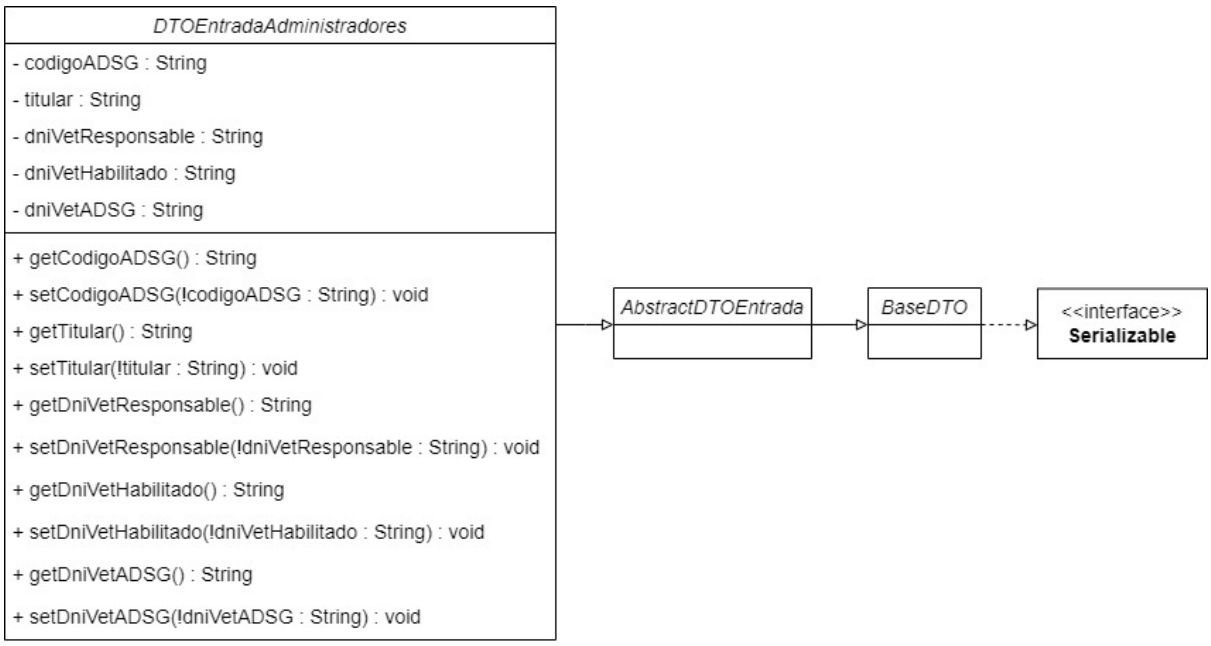


Figura 5.6: Diagrama de clases del *DTO* de entrada *DTOEntradaAdministradores*.

proyecto: *TitularMV*, *VetADSGMV*, *AdministradorMV*, el *DTO REST PaginaSubexplotacionesDTO*, mostrado en la figura A.6; y el *DTO REST DetalleSubexplotacionDTO*, representado en la figura A.23.

Los tres primeros *DTO MV* mencionados almacenan la información que se muestra dentro de las listas desplegables de cada bloque de filtros disponibles para el usuario. Por otro lado, *DetalleSubexplotacionDTO* contiene la información que se muestra en el detalle de una subexplotación, y *PaginaSubexplotacionesDTO* la información de las subexplotaciones que se muestran en el cuadro del componente *DataTables*.

PaginaSubexplotacionesDTO y *DetalleSubexplotacionDTO* se tratan también como un *DTO MV* porque ya contienen los atributos necesarios para almacenar la información que muestran en sus respectivos componentes, de este modo, evité crear dos *DTO MV* que fueran iguales a estos dos *DTO*.

TitularMV, *VetADSGMV* y *AdministradorMV* heredan de la clase *AbstractMV* mostrada en la figura 5.8, que almacena la información que se muestra dentro de las listas desplegables comunes a los tres tipos de filtros.

La clase *AbstractMV* tiene cuatro atributos con una lista de la interfaz *List* de *Java*. Sus atributos son: *explotacionTipos*, que almacena tantos objetos del *DTO REST TipoExplotacionDTO* como número de tipos tengan las explotaciones del titular; *explotacionEspecies*, que guarda un número de objetos del *DTO REST EspecieDTO* igual a las distintas especies que tengan las subexplotaciones asociadas a las explotaciones que el titular sea propietario; *clasificacionesZoo*, otra lista que contiene tantos objetos del *DTO REST ClasZootecnicaDTO* como clasificaciones zootécnicas tengan las subexplotaciones pertenecientes a las explotaciones del titular; y por último, *estadosSubexp*, que contiene tres objetos del *DTO REST EstadoSubexpDTO* con los

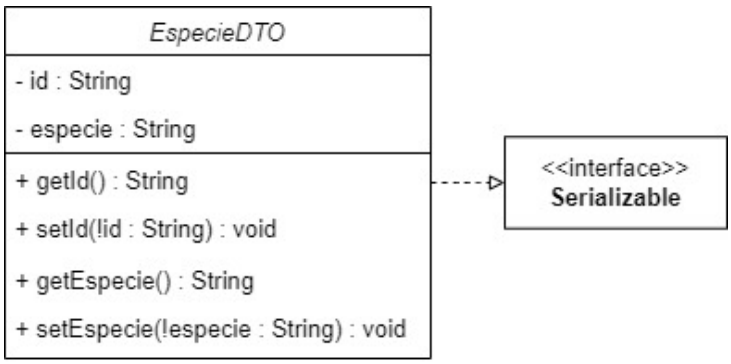


Figura 5.7: Diagrama del *DTO REST EspecieDTO*.

posibles estados de una subexplotación.

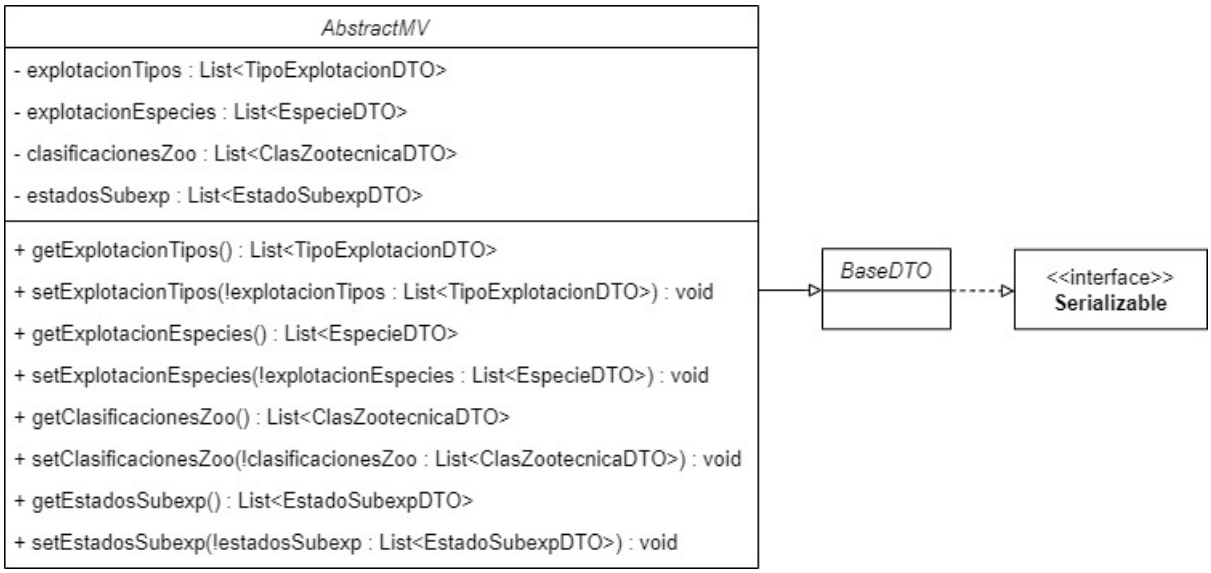


Figura 5.8: Diagrama de clases del *DTO MV AbstractMV*.

En la figura 5.9 se presenta el *DTO MV* del filtro para los titulares. Este *DTO* hereda todos sus atributos y métodos de la clase *AbstractMV*.

El siguiente *DTO MV* es *VetADSGMV*, expuesto en la figura 5.10. Es el encargado de almacenar la información que se muestra en las listas desplegables del bloque de filtros para los veterinarios ADSG. Los atributos no comunes a *TitularMV* y *AdministradorMV* en este *DTO* son: *listaExp*, lista que almacena tantos objetos del *DTO REST ExplotacionDTO* como explotaciones estén asociadas a las ADSG a las que pertenezca el veterinario; *listaADSG*, lista que contiene objetos del *DTO REST ADSGDTO*, cuyo número corresponde con las ADSG a las que está asociado el veterinario; y *listaTitulares*, que almacena una cantidad de objetos del *DTO REST PersonaDTO* igual al número de los distintos titulares propietarios de las explotaciones asociadas a las ADSG a las que pertenece el veterinario. Los demás atributos son heredados de la clase *AbstractMV*.

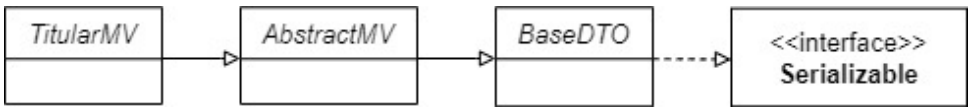


Figura 5.9: Diagrama de clases del *DTO MV TitularMV*.

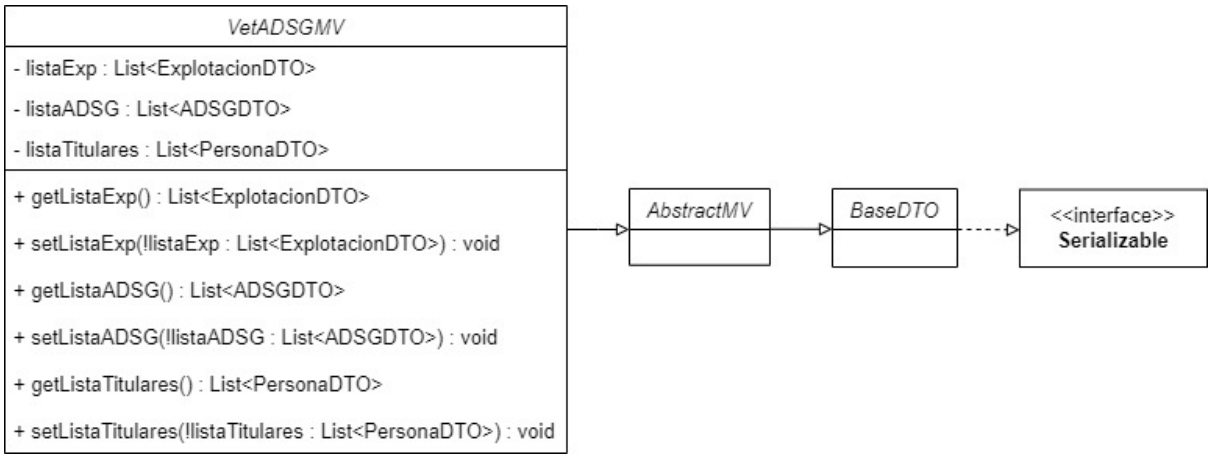


Figura 5.10: Diagrama de clases del *DTO MV VetADSGMV*.

Por último, en la figura 5.11 se muestra el *DTO MV AdministradorMV*, que guarda la información de las listas desplegables en el bloque de filtros para los administradores. Los atributos no comunes a *TitularMV* y *VetADSGMV* son: *listaADSG*, lista que contiene los *DTO REST ADSGDTO* con las ADSG a las que está asociado el administrador; *listaTitulares*, que almacena los *DTO REST PersonaDTO* con los titulares de las explotaciones a las que el administrador tiene acceso; *listaVetResp*, lista con los *DTO REST PersonaDTO* que contienen los veterinarios responsables de las explotaciones a las que el administrador tiene acceso; *listaVetHabilitado*, lista que almacena los *DTO REST PersonaDTO* con los veterinarios habilitados de las explotaciones a las que el administrador tiene acceso; y *listaVetADSG*, que incluye los *DTO REST PersonaDTO* con los veterinarios ADSG de las ADSG a las que pertenezcan las subexplotaciones que tiene acceso el administrador.

5.2.2. Módulo *client*

El módulo *client* contiene el cliente de la *API REST*, encargado de realizar las solicitudes *HTTP* a los puntos finales del servidor de la *API REST* y encapsular los datos que nos devuelve en los *DTO REST*.

Para implementar el cliente de la *API REST*, se creó una interfaz en lenguaje *Java* llamada *ClientRestService*, que contiene la definición de los métodos que realizan las llamadas a los puntos finales.

Cada método tiene el objetivo de realizar una solicitud *HTTP* a un punto final diferente. Es por ello que la interfaz tiene definidos 15 métodos, número que coincide con los puntos finales

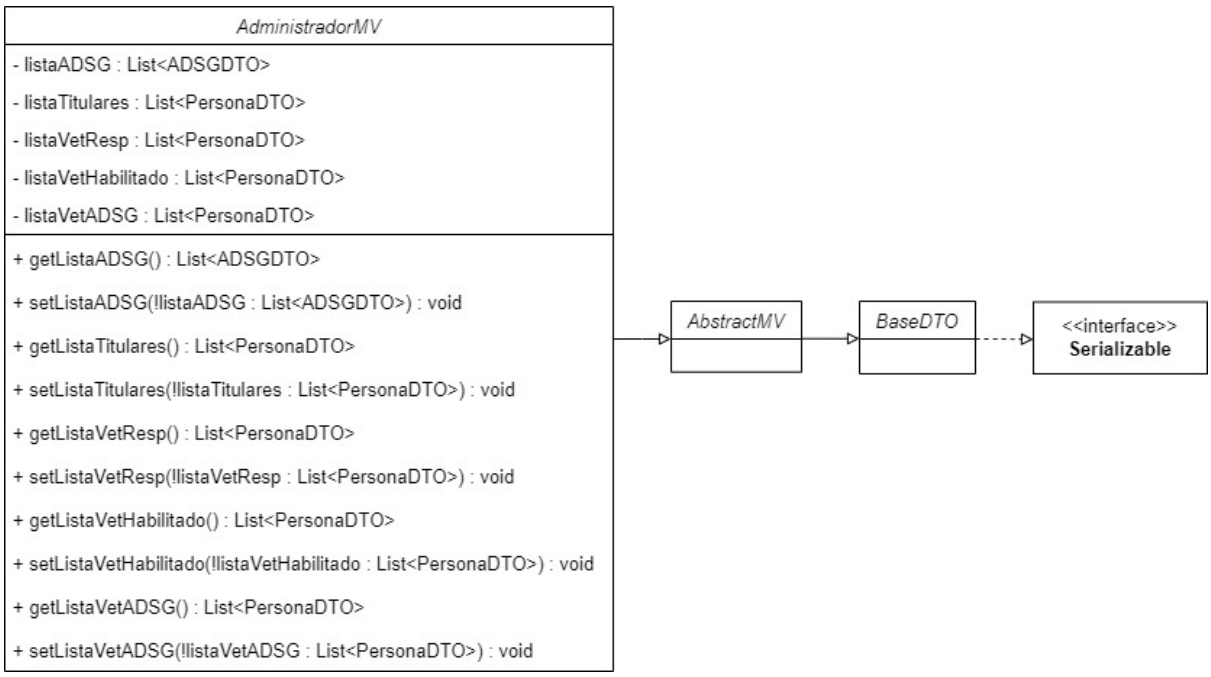


Figura 5.11: Diagrama de clases del *DTO MV AdministradorMV*.

indicados en el cuadro 4.1. La interfaz se muestra en la figura 5.12.

También se creó la clase *ClientRestServiceImpl*, que contiene la implementación de los métodos en la interfaz *ClientRestService*. El diagrama de clases de *ClientRestServiceImpl* está expuesto en la figura 5.13.

El constructor de esta clase tiene tres parámetros: *direccion*, cadena con la *URI* del servidor de la *API REST*; *tiempoEspera*, que contiene el tiempo en milisegundos de espera en la conexión y recepción del servidor de la *API REST*; y por último, *restTemplateBuilder*, objeto de la clase *RestTemplateBuilder* de *Spring Boot* para configurar y crear una instancia de la clase *RestTemplate*. La clase *RestTemplate* de *Spring*, es la encargada de realizar las solicitudes *HTTP* al servidor de la *API REST*.

La clase *ClientRestServiceImpl* hereda de la clase *RestServiceImpl*, que tiene el objetivo de configurar e instanciar su atributo *restTemplate*, de la clase *RestTemplate*.

Para configurar e instanciar su atributo *restTemplate* se siguen estos pasos: primero en el constructor de *ClientRestServiceImpl* se realiza una llamada al constructor de *RestServiceImpl*, pasándole tres argumentos: *direccion*, *tiempoEspera* y *restTemplateBuilder*.

Después, en el constructor de *RestServiceImpl*, si el parámetro *tiempoEspera* no es nulo, se llaman a los métodos *setConnectTimeout()* y *setReadTimeout()* del parámetro *restTemplateBuilder*. Al llamarlos, se les pasa el parámetro *tiempoEspera* como argumento, de este modo, se establece el tiempo máximo de espera en milisegundos que tendrán las solicitudes *HTTP* con el objeto *RestTemplate*.

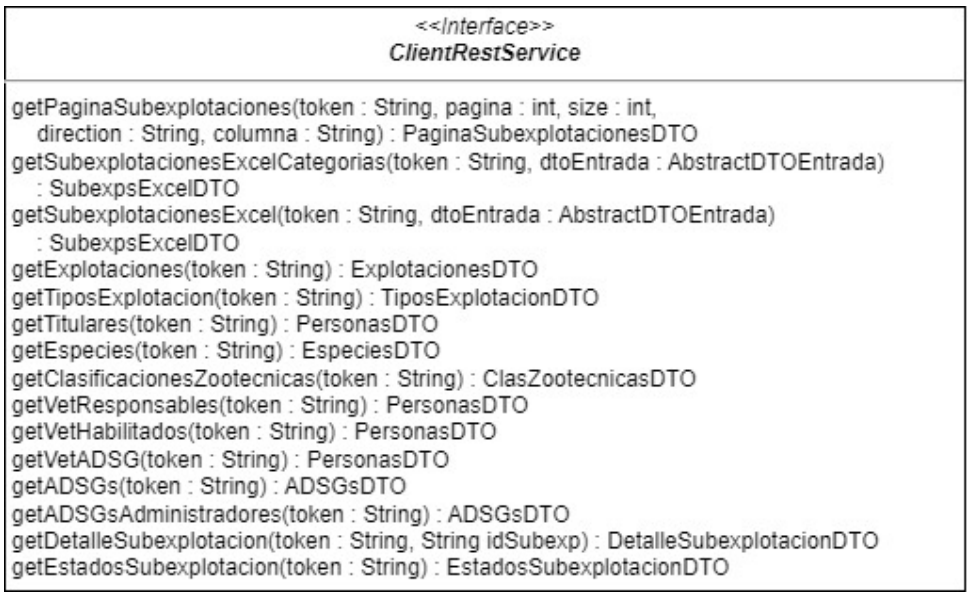


Figura 5.12: Diagrama de clases de la interfaz *ClientRestService*.

Una vez establecido el tiempo de espera, el atributo *restTemplateBuilder* llama a su método *build()*, que configura y devuelve una instancia de la clase *RestTemplate* [40]. Por último, la instancia devuelta se establece al atributo *restTemplate* de *RestServiceImpl*, de esta manera ya puede ser accedido con su método *getRestTemplate()*.

Por otro lado, en el constructor de *RestServiceImpl* también se establece su atributo *direccion* con el parámetro *direccion*. Así, la *URI* de la parte servidor de la *API REST* puede ser consultada con el método heredado *getDireccion()*.

Los métodos implementados en la clase *ClientRestServiceImpl* utilizan el método *exchange()* del atributo *restTemplate*, para realizar las solicitudes *HTTP* a los puntos finales y encapsular las respuestas *HTTP* en un objeto de la clase *ResponseEntity* de *Spring* [39].

Este método necesita una serie de parámetros: una cadena *String* que representa la plantilla *URI* que identifica el punto final llamado, un objeto de la clase *HttpMethod* que indica el tipo de método *HTTP* utilizado en la solicitud, un objeto de la clase *HttpEntity* que contiene las cabeceras y el cuerpo de la solicitud *HTTP*, la clase a la que se quiere convertir la respuesta y por último, de forma opcional, una o más variables que se expanden en la plantilla de la *URI* [39].

Una plantilla *URI* es una secuencia de caracteres para representar una *URI* mediante la expansión del contenido de las expresiones incluidas en la plantilla. Una plantilla puede tener cero o más expresiones y cada expresión se identifica por el texto entre las llaves «{» y «}» [18].

La expansión es la cadena resultante que se obtiene al procesar cada expresión en la plantilla *URI*. Este procesamiento lo realiza el procesador de la plantilla, que es el programa o biblioteca que transforma la plantilla *URI* en una *URI*, sustituyendo cada una de las expresiones en la plantilla por su expansión correspondiente [18].

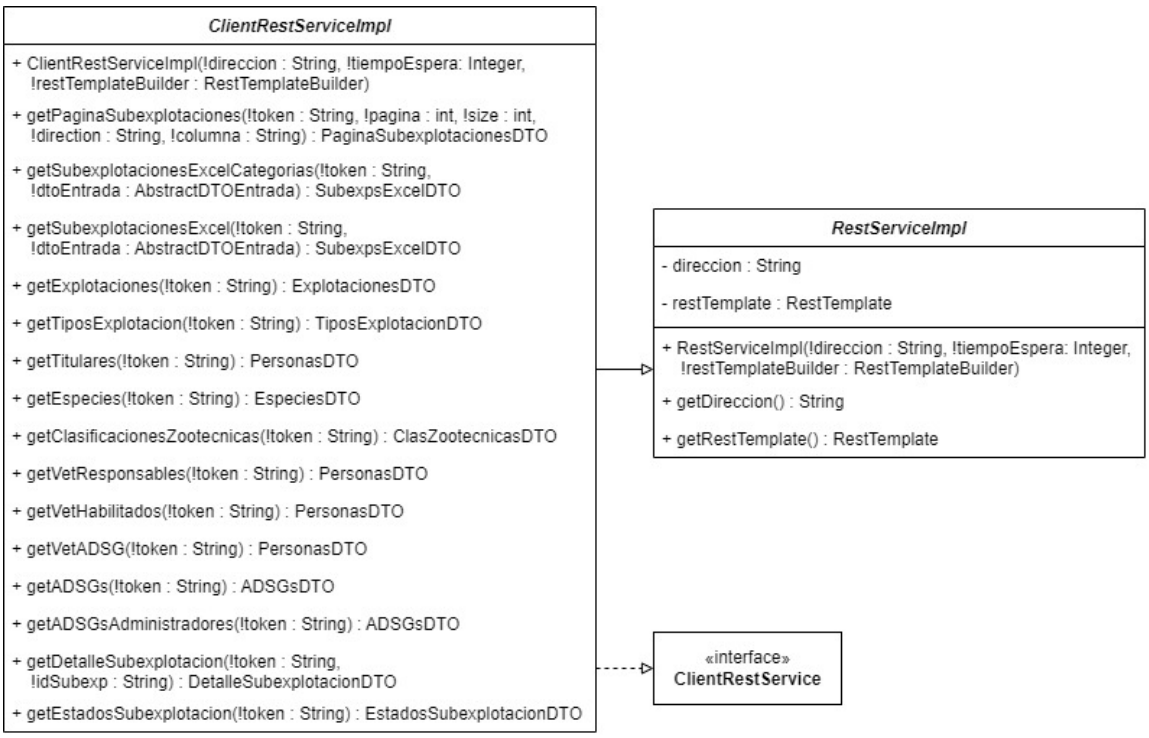


Figura 5.13: Diagrama de clases de la clase *ClientRestServiceImpl*.

La plantilla *URI* de cada uno de los métodos está formada por dos partes: la primera parte es la dirección del servidor de la *API REST*, proporcionada por el método *getDireccion()* y la segunda es la *URI* del punto final al que se envía la solicitud *HTTP*, que está definida en la columna ≪Punto final≫ del cuadro 4.1.

Existen dos puntos finales que contienen variables de expansión, estos son: ≪*paginaSubexplotaciones/{pagina}/{size}/{direction}/{columna}*≫ y ≪*detalleSubexplotacion/{idSubexp}*≫.

La llamada al primer punto final se realiza dentro del método *getPaginaSubexplotaciones()* de la clase *ClientRestServiceImpl*, en concreto, en el método *exchange()*. El primer argumento en la llamada es una cadena con la siguiente plantilla *URI*: ≪*getDireccion() + "paginaSubexplotaciones/{pagina}/{size}/{direction}/{columna}"*≫. La plantilla contiene cuatro expresiones, que son expandidas con los valores de las variables opcionales pasadas como argumentos en el método *exchange()*, que son los parámetros *pagina*, *size*, *direction* y *columna* del método *getPaginaSubexplotaciones()*.

El siguiente punto final que contiene una expresión en su plantilla *URI* es ≪*detalleSubexplotacion/{idSubexp}*≫, llamado con *exchange()* dentro del método *getDetalleSubexplotacion()*. La expresión ≪{*idSubexplotacion*} se sustituye por el parámetro *idSubexplacion* al pasarlo como argumento al método *exchange()*.

Los demás puntos finales no contienen ninguna expresión en su plantilla *URI*, es por ello que sus llamadas al método *exchange()* (dentro del método de la clase *ClientRestServiceImpl* que le corresponda a cada uno) no contienen ninguna variable opcional como argumento que se

expanda en su plantilla *URI*.

El objeto de la clase *HttpEntity* en los parámetros del método *exchange()* contiene las cabeceras y el cuerpo de la solicitud *HTTP*. Las cabeceras son información adicional adjunta a la solicitud o respuesta *HTTP*. Cada cabecera está formada por un nombre, seguido de dos puntos y un valor. Por tanto, su estructura es: ≪clave : valor≫ [52].

Una de las cabeceras añadidas en el objeto de la clase *HttpEntity* es *Authorization*. La cabecera *Authorization* contiene información sobre el usuario que realiza la solicitud *HTTP*. Esta información está codificada en un *JSON Web Token (JWT)* [28]. El *JWT* es obtenido a partir de un servicio externo que se ejecuta antes de entrar a la OVV y al que delegamos la autenticación. Este servicio muestra una pantalla de inicio de sesión con autenticación por DNI y contraseña. Si la autenticación es correcta, el servicio almacena el *JWT* del usuario como una *cookie* en el navegador.

El *JWT* contiene información sobre el usuario, tales como: el tipo de usuario, DNI, etc. Esta información es tratada por la parte servidor de la *API REST* para identificar qué información consultar en la base de datos. Una vez consultada, la codifica en formato *JavaScript Object Notation (JSON)* [44], la introduce en el cuerpo de la respuesta *HTTP* y la envía al cliente de la *API REST*.

La cabecera *Authorization* explicada es añadida dentro del objeto de la clase *HttpEntity* creado en el método, que a su vez se pasa como argumento al método *exchange()*. La cabecera tiene como clave ≪Authorization≫ y su valor es el parámetro *token*, que es una cadena con el *JWT*.

Por otro lado, existen tres métodos en la clase *ClientRestServiceImpl* que añaden el *DTO* de entrada *dtoEntrada* llegado por parámetro dentro del cuerpo de la solicitud *HTTP* realizada por el método *exchange()*. Esto se consigue añadiendo el *DTO* de entrada dentro del objeto de la clase *HttpEntity* creado en cada método. Los tres métodos son: *getPaginaSubexplotaciones()*, *getSubexplotacionesExcelCategorias()* y *getSubexplotacionesExcel()*.

Estos tres métodos introducen el *DTO* de entrada en el cuerpo de su solicitud *HTTP* porque la *API REST* necesita la información dentro de este *DTO* para saber qué subexplotaciones devolver en la respuesta *HTTP* a dichos métodos.

Después de realizar la solicitud *HTTP* con el método *exchange()*, el propio método recibe la respuesta *HTTP* y usa la biblioteca *Jackson* [22] para convertir el *JSON* incluido en el cuerpo de la respuesta en un objeto de la clase *DTO REST* indicada como argumento en el método *exchange()*.

Una vez realizada la conversión, el método *exchange()* devuelve un objeto de la clase *ResponseEntity*. Luego, se llama al método *getBody()* del objeto devuelto, que a su vez devuelve el *DTO REST* incluido en el cuerpo de la respuesta *HTTP* representada por el objeto de la clase *ResponseEntity*. Finalmente, cada *DTO REST* es devuelto en su respectivo método de la clase *ClientRestServiceImpl*.

5.2.3. Módulo *service-api*

El módulo *service-api* contiene dos interfaces con la lógica de negocio del proyecto: *ServiceEPU* y *ExportServiceEPU*.

El objetivo de cada método de la interfaz *ServiceEPU*, mostrada en el diagrama de clases de la figura 5.14 es crear y devolver un *DTO MV* con toda la información que se muestra en uno o varios componentes de la vista.

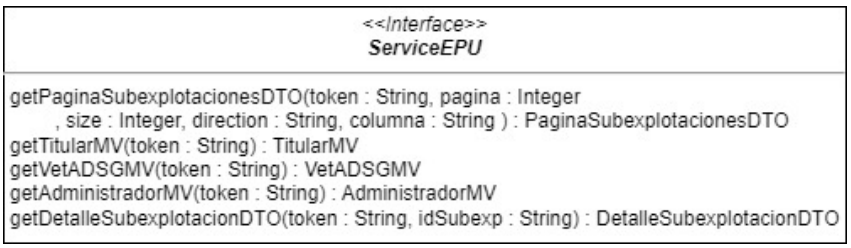


Figura 5.14: Diagrama de clases de la interfaz *ServiceEPU*.

El método *getPaginaSubexplotacionesDTO()* devuelve el *DTO MV PaginaSubexplotacionesDTO*, que contiene las subexplotaciones que se muestran en el componente *DataTables*.

Los tres siguientes métodos contienen la información a mostrar dentro de las listas desplegables en los tres tipos de bloques de filtros disponibles. El método *getTitularMV()* devuelve el *DTO MV TitularMV*, que contiene la información del bloque de filtros para los titulares; *getVetADSGMV()* devuelve el *DTO MV VetADSGMV*, con la información del bloque de filtros para los veterinarios ADSG y *getAdministradorMV()* devuelve el *DTO MV AdministradorMV*, con la información del bloque de filtros para los administradores.

Por último, *getDetalleSubexplotacionDTO()* devuelve el *DTO MV DetalleSubexplotacionDTO*, que contiene la información que se muestra en la vista del detalle de una subexplotación.

Por otro lado, el objetivo de cada método de la interfaz *ExportServiceEPU* es devolver una lista de bytes con el contenido de un archivo *Excel*. Esta interfaz está representada en el diagrama de clases de la figura 5.15, que tiene definidos dos métodos.

El método *getExcelCategorias()* devuelve el contenido de un *Excel* que contiene las subexplotaciones consultadas con la información de sus categorías. Estas categorías están mostradas en el componente *DataTables* de la vista del detalle de una subexplotación.

En cambio, *getExcelSinCategorias()* devuelve el contenido de un *Excel* con las subexplotaciones consultadas sin sus categorías.

5.2.4. Módulo *service-impl*

El siguiente módulo, *service-impl* contiene la implementación de las dos interfaces creadas en el módulo *service-api*. Estas dos implementaciones son las clases: *ServiceEPUImpl*, que im-

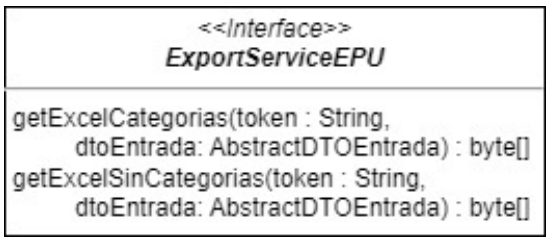


Figura 5.15: Diagrama de clases de la interfaz *ExportServiceEPU*.

plementa a la interfaz *ServiceImpl*; y *ExportServiceEPUImpl*, que implementa a la otra interfaz *ExportServiceEPU*.

ServiceEPUImpl

El diagrama de clases de *ServiceEPUImpl* se encuentra reflejado en la figura 5.16.

El único atributo de esta clase es *clientRestService*, objeto de interfaz *ClientRestService*, que se utiliza para captar los *DTO REST* necesarios de la parte servidor de la *API REST*. Su constructor *ServiceEPUImpl()* solo asigna a su atributo *clientRestService* el parámetro *clientRestService*.

Los métodos privados de la clase *ServiceEPUImpl* tienen el objetivo de llamar a los métodos del atributo *clientRestService* para adquirir el *DTO REST* de la parte servidor de la *API REST* que contiene el atributo que devuelve el método privado.

Por ejemplo, el método privado *getListaEspecies()* llama con el atributo *clientRestService* al método *getEspecies()* para conseguir el *DTO REST EspeciesDTO*. Una vez conseguido, accede a su atributo *listaEspecies* del tipo `List<EspecieDTO>` con su método *getListaEspecies()* y lo devuelve.

Estos métodos privados son llamados por los métodos públicos de *ServiceEPUImpl*: *getTitularMV()*, *getVetADSGMV()* y *getAdministradorMV()*. Estos métodos públicos devuelven en un *DTO MV* la información que se muestra en las listas desplegables del bloque de filtros que representa cada uno. En concreto, *getTitularMV()* almacena las listas del bloque para los titulares, *getVetADSGMV()* para los veterinarios ADSG y *getAdministradorMV()* para los administradores.

Cada uno de estos tres métodos públicos realiza los siguientes pasos: primero, crea el objeto *DTO MV* que devuelve y después, se llama uno a uno a varios métodos privados de la clase *ServiceEPUImpl*, que consiguen la información necesaria para cada una de sus listas desplegables. En cada llamada a los métodos privados se les pasa por argumento el objeto *DTO MV* creado y la cadena *token*. Una vez que se ejecuta cada método privado, se establece la información devuelta mediante un *Setter* en el *DTO MV*.

En el algoritmo 1 se enseña el pseudocódigo del método *getTitularMV()*, que sirve de referencia para los otros métodos públicos: *getVetADSGMV()* y *getAdministradorMV()*. Además,

en el algoritmo se siguen los pasos descritos en el párrafo anterior.

Algorithm 1 Obtener el *DTO MV TitularMV* con el método *getTitularMV()*

```
1: procedure getTitularMV(token)
2:     Crear un DTO MV TitularMV vacío llamado mv
3:     Obtención de listaTiposExplotacion con getListaTiposExplotacion(token, mv)
4:     Asignación de listaTiposExplotacion a mv con mv.setExplotacionTipos(listaTiposExplotacion)
5:     Obtención de listaEspecies con getListaEspecies(token, mv)
6:     Asignación de listaEspecies a mv con mv.setExplotacionEspecies(listaEspecies)
7:     Obtención de listaClasZootecnicas con getListaClasZootecnicas(token, mv)
8:     Asignación de listaClasZootecnicas a mv con mv.setClasificacionesZoo(listaClasZootecnicas)
9:     Obtención de listaEstadosSub con getListaEstadosSub(token, mv)
10:    Asignación de listaEstadosSub a mv con mv.setEstadosSubexp(listaEstadosSub)
11:    Devolver mv
12: end procedure
```

Por ejemplo, *getTitularMV()*, llama a los métodos privados: *getListaTiposExplotacion()*, *getListaEspecies()*, *getListaClasZootecnicas()* y *getListaEstadosSub()*. De esta manera consigue la información para sus cuatro listas desplegables.

La estructura de los métodos privados de la clase *ServiceEPUImpl* es similar, por ello, se muestra en el algoritmo 2 el pseudocódigo del método *getListaExplotaciones()*.

Algorithm 2 Obtener la lista de explotaciones con el método *getListaExplotaciones()*

```
1: procedure getListaExplotaciones(token, model)
2:     Crear una lista vacía llamada listaExplotaciones
3:     if model.getOk() es verdadero then
4:         Obtención de explotaciones con clienteRestService.getExplotaciones(token)
5:         if explotaciones.getDescripcionResultado() está vacío then
6:             Asignar explotaciones.getExplotaciones() a listaExplotaciones
7:         else
8:             Asignar falso al atributo ok del parámetro model con model.setOk(false)
9:         end if
10:    end if
11:    Devolver listaExplotaciones
12: end procedure
```

La condición en la línea tres del algoritmo 2 se realiza para verificar que el atributo *ok* es verdadero. Este atributo indica si ha habido algún error en las llamadas a la *API REST* por parte de los demás métodos privados de la clase *ServiceEPUImpl* ejecutados dentro del método público correspondiente.

En la línea cinco del algoritmo 2 se verifica si el atributo *descripcion* del *DTO REST* capturado en la línea cuatro está vacío. Si no está vacío, la solicitud a la parte servidor de la *API REST* ha sido errónea y se asigna el valor falso al atributo *ok* del parámetro *model*, de esta manera, las llamadas restantes a los métodos privados dentro del método público no ejecutarán ninguna solicitud a los puntos finales de la parte servidor de la *API REST*.

ExportServiceEPUImpl

La clase *ExportServiceEPUImpl* es la clase encargada de exportar el contenido de las subexplotaciones consultadas en un archivo *Excel*. Esta clase está representada en el diagrama de clases de la figura 5.17.

Su único atributo es *clientRestService*, que pertenece a la clase *ClientRestService*. Este atributo es usado para llamar a los métodos *getSubexplotacionesExcelCategorias()* y *getSubexplotacionesExcel()*, que consiguen la información de las subexplotaciones que se van a exportar en el *Excel*.

Su constructor *ExportServiceEPU* solo asigna a su atributo *clientRestService* el parámetro *clientRestService*.

A continuación se explican los métodos privados de la clase *ExportServiceEPUImpl*:

- *getListaSubexpExcelCategorias()*: utiliza el atributo *clientRestService* para llamar al método *getSubexplotacionesExcelCategorias()*, que obtiene de la parte servidor de la *API REST* las subexplotaciones con sus categorías que se van a exportar en el fichero *Excel*. En el caso de que la solicitud sea errónea, el método *getListaSubexpExcelCategorias()* devuelve una lista vacía, en caso contrario, la lista con las subexplotaciones con sus categorías.
- *getListaSubexpExcel()*: utiliza el atributo *clientRestService* para llamar al método *getSubexplotacionesExcel()*, que obtiene de la parte servidor de la *API REST* las subexplotaciones sin sus categorías que se van a exportar en el fichero *Excel*. En el caso de que la solicitud sea errónea, el método *getListaSubexpExcel()* devuelve una lista vacía, en caso contrario, la lista con las subexplotaciones sin sus categorías.
- *crearEstiloCabecera()*: define el estilo de las celdas que representan la cabecera y el título de la *Excel*. Esto es realizado con la clase *CellStyle* de la biblioteca *APACHE POI*. Una vez hecho, devuelve el objeto utilizado de esa clase.
- *crearTituloExcel()*: crea la primera fila del archivo *Excel* y una celda dentro de ella, que se extiende hasta abarcar el rango de columnas que contiene la cabecera. Esta celda representa el título de la *Excel* y contiene el texto ≪Explotaciones relacionadas con el perfil del usuario≫.
- *crearCabecera()*: crea la segunda fila del archivo *Excel* y sus celdas correspondientes. Para ello, usa el parámetro *mapaCabeceras*, que es un mapa de la interfaz *Map* de *Java* donde cada valor almacenado dentro de él se escribe en una celda de la segunda fila. Estas celdas representan las cabeceras, que son los nombres que identifican la información que se muestra en cada columna del *Excel*.
- *getCabecerasExcelCategorias()*: este método crea la cabecera y el título del archivo *Excel*, además, crea y devuelve un mapa con las cabeceras utilizadas en el *Excel* que incluye la información de las subexplotaciones con sus categorías. Primero, el método crea un mapa, que va a almacenar cadenas *String* en sus claves y valores. A este mapa se le añaden los nombres de las cabeceras. Después, crea una variable llamada *estilo* llamando al método *crearEstiloCabecera()*. Luego, se llama al método *crearTituloExcel()* y *crearCabecera()* proporcionándoles sus argumentos necesarios. Finalmente, devuelve el mapa creado.

- *getCabecerasExcelSinCategorias()*: este método realiza lo mismo que el método *getCabecerasExcelCategorias()*, lo único que al mapa que crea se le añaden las cabeceras que tiene el *Excel* que muestra información de las subexplotaciones sin sus categorías.

- *switchContenidoComun()*: el método utiliza una estructura *switch* para evaluar el parámetro *cabecera*. Este parámetro indica qué atributo del parámetro *subexpDTO*, objeto que contiene la información de una subexplotación, debe escribirse dentro del parámetro *celda*, que contiene la celda del *Excel* donde se va a escribir. Esta estructura *switch* solo evalúa los nombres de las cabeceras comunes tanto para el *Excel* que contiene las categorías como para el que no.

- *switchContenidoCategoria()*: el método utiliza un *switch* para evaluar el parámetro *cabecera*. Este parámetro indica qué atributo de los objetos incluidos dentro del parámetro *categorias*, que contiene una lista con cada categoría de la subexplotación, tiene que escribirse dentro del parámetro *celda*. También tiene un parámetro con el nombre *pos*, que indica el índice del objeto la lista *categorias* al que se quiere acceder.

- *switchContenidoSinCategoria()*: este método es similar a *switchContenidoComun()*, la única diferencia es que la estructura *switch* solo evalúa los nombres de las cabeceras que son específicos para el *Excel* que no contiene las categorías de las subexplotaciones.

- *crearContenidoExcelCategorias()*: método encargado de construir el contenido del *Excel* que contiene las categorías de las subexplotaciones. Este método se representa mediante pseudocódigo en el algoritmo 3.

- *crearContenidoExcelSinCategorias()*: método encargado de construir el contenido del *Excel* que contiene las subexplotaciones consultadas sin sus categorías. Este método se describe con pseudocódigo en el algoritmo 4.

- *excelABytes()*: este método escribe el *Excel* representado en el parámetro *libro* en un objeto de la clase *ByteArrayOutputStream* de *Java*. Después, se devuelve el contenido de este objeto en una lista de bytes.

El método en pseudocódigo representado en el algoritmo 3 contiene tres parámetros: *hoja*, que representa la hoja de cálculo de la *Excel*; *mapaCabeceras*, mapa con las cabeceras del tipo de *Excel* que contiene la información de las subexplotaciones con sus categorías; y *listaSubexp*, lista que almacena las subexplotaciones con sus categorías y que se van a escribir en el archivo *Excel*.

Las líneas dos y tres inicializan las variables *numFila* y *numColumna*, respectivamente. La variable *numFila* indica el número de la fila en la hoja de la *Excel* donde se va a realizar la escritura, mientras que *numColumna* indica el número de la columna en la que se va a llevar a cabo la escritura.

En la línea cuatro se ejecuta un bucle *for*, que recorre cada objeto de la clase *SubexpExcelDTO* incluido dentro de la lista recibida por parámetro *listaSubexp*. Además, cada objeto recorrido se incluye en el objeto *subexplotacionExcel* indicado en la línea cuatro.

Después, en la línea cinco se almacena en una variable llamada *categorias* la lista con las categorías almacenadas en el objeto *subexplotacionExcel*. La lista con las categorías se consigue

Algorithm 3 Creación contenido del *Excel* con las categorías con el método *crearContenidoExcelCategorias()*

```
1: procedure crearContenidoExcelCategorias(hoja, mapaCabeceras, listaSubexp)
2:     Inicializar numFila a 2
3:     Inicializar numColumna a 0
4:     for cada subexplotacionExcel en listaSubexp do
5:         Obtener la lista de categorías categorias de subexplotacionExcel
6:         Inicializar i a 0
7:         repeat
8:             Crear una nueva fila row en hoja en la posición numFila
9:             for cada entrada en mapaCabeceras do
10:                Crear una nueva celda cell en la posición numColumna de la fila row
11:                Llamar a switchContenidoComun con los argumentos entrada.getKey(), cell y
   subexplotacionExcel
12:                Llamar a switchContenidoCategoria con los argumentos entry.getKey(), cell,
   categorias e i
13:                Incrementar numColumna en 1
14:            end for
15:            Incrementar numFila en 1
16:            Incrementar i en 1
17:            Asignar un 0 a numColumna
18:        until i es mayor o igual que el tamaño de categorias
19:        Asignar un 0 a numColumna
20:    end for
21: end procedure
```

llamando al método *getCategoriasExcel()* del objeto *subexplotacionExcel*. Una vez hecho, en la línea seis se crea una variable *i* con el número cero, que indica el índice del objeto de la clase *CategoriaDTO* accedido dentro de la lista *categorias*.

Desde la línea siete hasta la 18 existe un bucle *do-while*, que se ejecuta una vez por cada categoría dentro de la lista *categorias*.

En la línea ocho dentro del bucle *do-while* se crea una variable llamada *row* de la clase *Row*, incluida en la biblioteca de *APACHE POI*. Esta variable se crea al llamar con el parámetro *hoja* a su método *createRow()*, pasándole como argumento la variable *numFila*. Este método crea una fila dentro de la hoja de cálculo del archivo *Excel* en la posición indicada por el argumento *numFila*.

A continuación, desde la línea nueve a la 14 se ejecuta un bucle *for*, que recorre todas las cabeceras incluidas dentro del parámetro *mapaCabeceras*. Cada cabecera recorrida se incluye en la variable *entrada* de la línea nueve. Dentro de ese bucle, en la línea 10 se crea una variable *cell* de la clase *Cell*, también perteneciente a la biblioteca de *APACHE POI*. Esta variable se crea llamando con la variable *row* al método *createCell()*, pasándole como argumento la variable *numColumna*. Con el método *createCell()* se crea una celda en la hoja del archivo *Excel*, que se posiciona en la columna indicada por la variable *numColumna* de la fila representada por la variable *row*.

Una vez concluido, en la línea 11 se llama al método privado *switchContenidoComun()* con los argumentos: *entrada.getKey()*, *cell* y *subexplotacionExcel*. Si el nombre de la cabecera indicada en el argumento *entrada.getKey()* coincide con alguna de las condiciones evaluadas dentro del *switch* de *switchContenidoComun*, se escribe el contenido del atributo de *subexplotacionExcel* asociado a dicha cabecera dentro de la celda indicada en el argumento *cell*.

Por ejemplo, si el valor de *entrada.getKey()* en la llamada al método *switchContenidoComun()* fuera «provincia», que es una cabecera común a los dos tipos de *Excel*, se obtendría el valor del atributo *provinciaExcel* de *subexplotacionExcel* utilizando su método *getProvinciaExcel()*, y después se escribiría ese valor dentro de la celda en el objeto *cell*.

En la línea 12 se llama al método privado *switchContenidoCategoria()*, con los argumentos: *entry.getKey()*, *cell*, *categorias* e *i*. Si el nombre de la cabecera indicada en el argumento *entry.getKey()* coincide con alguna de las condiciones evaluadas en el *switch* del método privado, se escribe el contenido del atributo asociado a dicha cabecera, que está ubicado en el objeto de la posición *i* de la lista *categorias*.

Un ejemplo en la llamada al método privado *switchContenidoCategoria()* podría ser cuando el valor de *entrada.getKey()* fuera «descripcion» y el valor de *i* un cero. En ese caso, se accedería al valor del atributo *descripcion* llamando al método *getDescripcion()* del objeto en el índice cero de la lista *categorias*. Una vez obtenido, se escribiría el valor dentro de la celda en el objeto *cell*.

Una vez acaba el bucle *for*, en la línea 14, ya se han escrito todas los valores necesarios en las celdas de la fila actual, por tanto, se debe empezar a escribir en la siguiente fila. Para conseguirlo, en la línea 15 se incrementa en uno el valor de la variable *numFila*.

La línea 16 incrementa la variable *i* en uno para acceder al objeto en la siguiente posición de la lista *categorias*. Antes de acceder a ella, en la línea 18 se realiza la siguiente comprobación para verificar si quedan más posiciones en la lista por acceder: si la variable *i* es mayor o igual que el tamaño de la lista *categorias*, indica que ya se han escrito todas las categorías de la lista y por tanto, el bucle acaba.

En caso contrario, aún quedan categorías por escribir, por tanto, se ejecuta otra iteración del bucle *do-while*, escribiendo de nuevo el contenido de las cabeceras comunes a los dos tipos de *Excel* y a continuación, la información de la nueva categoría almacenada en la posición *i* de la lista *categorias*.

En el caso de que queden categorías por escribir en el archivo *Excel*, se vuelve a escribir desde la columna en la posición cero de la hoja *Excel*. Para conseguirlo, en la línea 17 se asigna el valor cero a la variable *numColumna*.

En la línea 20 se acaba el bucle *for* principal. En el caso de que queden subexplotaciones por escribir en la lista del parámetro *listaSubexp*, se ejecuta otra iteración desde la línea cuatro. En caso contrario, se sale del bucle y el método *crearContenidoExcelCategorias()* finaliza.

Algorithm 4 Creación contenido del *Excel* sin las categorías con el método *crearContenidoExcelSinCategorias()*

```
1:  procedure crearContenidoExcelSinCategorias(hoja, mapaCabeceras, listaSubexp)
2:      Inicializar numFila a 2
3:      Inicializar numColumna a 0
4:      for cada subexplotacionExcel en listaSubexp do
5:          Crear una nueva fila row en hoja en la posición numFila
6:          for cada entrada en mapaCabeceras do
7:              Crear una nueva celda cell en la posición numColumna de la fila row
8:              Llamar a switchContenidoComun con los argumentos entrada.getKey(), cell y
    subexplotacionExcel
9:              Llamar a switchContenidoSinCategoria con los argumentos entry.getKey(), cell y
    subexplotacionExcel
10:             Incrementar numColumna en 1
11:         end for
12:         Incrementar numFila en 1
13:         Asignar un 0 a numColumna
14:     end for
15: end procedure
```

El método en pseudocódigo representado en el algoritmo 4 contiene los mismos parámetros que el algoritmo 3: *hoja*, que representa la hoja de la *Excel*; *mapaCabeceras*, mapa con las cabeceras del tipo de *Excel* que contiene la información de las subexplotaciones sin sus categorías; y *listaSubexp*, lista que almacena las subexplotaciones sin sus categorías y que se van a escribir en el archivo *Excel*.

Las líneas dos y tres inicializan las variables *numFila* y *numColumna*, respectivamente. Estas variables se utilizan de la misma manera que las variables con esos nombres en el método *crearContenidoExcelCategorias()*, explicado en el algoritmo 3.

En la línea cuatro se ejecuta un bucle *for*, que recorre cada objeto de la clase *SubexpExcelDTO* incluido dentro de la lista recibida por parámetro *listaSubexp*. Cada objeto recorrido se incluye en la variable *subexplotacionExcel*.

En la línea cinco se crea una variable llamada *row* de la clase *Row*. Esta variable se crea al llamar con el parámetro *hoja* a su método *createRow()*, pasándole como argumento la variable *numFila*.

A continuación, desde la línea seis a la 11 se ejecuta un bucle *for*, que recorre todas las cabeceras incluidas dentro del parámetro *mapaCabeceras*. Dentro de ese bucle, en la línea siete se crea una variable *cell* de la clase *Cell*. Esta variable se crea llamando con la variable *row* al método *createCell()*, pasándole como argumento la variable *numColumna*.

Una vez creada la celda, en la línea ocho se llama al método privado *switchContenidoComun()* con los argumentos: *entrada.getKey()*, *cell* y *subexplotacionExcel*.

En la línea nueve se llama al método privado *switchContenidoSinCategoria()*, con los argumentos: *entry.getKey()*, *cell*, *categorias* y *subexpDTO*. Si el nombre de la cabecera indicada en el argumento *entrada.getKey()* coincide con alguna de las condiciones evaluadas dentro del *switch* de *switchContenidoSinCategoria()*, se escribe el contenido del atributo de *subexplotacionExcel* asociado a dicha cabecera dentro de la celda en el objeto *cell*.

Una vez escrito el valor correspondiente en la celda, se incrementa en uno el valor de la variable *numColumna* para crear una nueva celda en la columna siguiente.

Al acabar en la línea 11 el bucle *for* ya se han escrito todos los valores necesarios en las celdas de la fila actual, por tanto, se debe empezar a escribir en la siguiente fila. Para conseguirlo, en la línea 12 se incrementa en uno el valor de la variable *numFila*.

Por otro lado, si quedan subexplotaciones por recorrer, tiene que empezar a escribirse su contenido desde la primera columna de la hoja *Excel*, por ello, en la línea 13 se asigna un cero a la variable *numColumna*.

En la línea 14 se acaba el bucle *for* principal. En el caso de que queden subexplotaciones por escribir en la lista del parámetro *listaSubexp*, se ejecuta otra iteración desde la línea cuatro. En caso contrario, se sale del bucle y el método *crearContenidoExcelSinCategorias()* finaliza.

Una vez explicados todos los métodos privados, se explican los métodos públicos de *ExportServiceEPUImpl*.

El método *getExcelCategorias()* se encarga de exportar en una lista de bytes el contenido de un *Excel* con la información de las subexplotaciones consultadas con sus categorías. Para ello, primero llama al método *getListaSubexpExcelCategorias()*, pasándole como argumento el parámetro *token*. La lista devuelta por el método *getListaSubexpExcelCategorias()* se asigna a la variable *listaExcel*. Si el contenido de la *listaExcel* es nulo, el método *getExcelCategorias()* devuelve una lista de bytes vacía. En cambio, si no es nulo, el método sigue su ejecución.

Después, se instancia el objeto *libro* de la clase *Workbook*, que pertenece a la biblioteca *APACHE POI* y representa el archivo *Excel*. Una vez instanciado el objeto *libro* se crea una

variable *hoja*, de la clase *Sheet* también de *APACHE POI*. Esta variable se crea llamando al método *createSheet()* del atributo *libro*, que crea una nueva hoja en el archivo *Excel*.

Tras eso, se crea un mapa en la variable *header* llamando al método *getCabecerasExcelCategorias()*, que contiene todas las cabeceras para el tipo de *Excel* que se va a crear. A este método se le pasa como argumentos las variables *libro* y *hoja*.

Como penúltimo paso se llama al método *crearContenidoExcelCategorias()* pasándole como argumentos las variables: *hoja*, *mapaCabeceras* y *listaExcel*.

Finalmente, se devuelve el resultado de la llamada al método *excelABytes()*, pasándole como argumento el objeto *libro*.

El método *getExcelSinCategorias()* exporta en una lista de bytes el contenido de un *Excel* con la información de las subexplotaciones consultadas pero sin la información de sus categorías. Este método es prácticamente igual que *getExcelCategorias()* pero contiene dos diferencias: el mapa *header* se crea llamando al método *getCabecerasExcelSinCategorias()* y se ejecuta el método *crearContenidoExcelSinCategorias()* en lugar del método *crearContenidoExcelCategorias()*.

5.2.5. Módulo *application*

En este subapartado van a explicarse tres componentes importantes dentro del módulo *application*. Estos son: la clase controladora del proyecto, el fichero *JavaScript* creado y los componentes de la capa vista.

ControllerEPU

El *framework Spring* hace uso del patrón de diseño *Front Controller* o Controlador Frontal para el manejo de las peticiones *HTTP*, donde un solo controlador es el encargado de manejarlas. En concreto, las maneja un *Servlet* central de *Spring* llamado *DispatcherServlet*, que dirige las peticiones *HTTP* a los controladores de la aplicación [9].

La clase que designa el controlador del proyecto es *ControllerEPU*. Esta clase contiene la anotación *@Controller* de *Spring* para indicarle a *Spring* que pertenece a los controladores de la aplicación.

Además, la clase utiliza la anotación *@RequestMapping* para indicar la ruta base de la *URI* a la que responderán todos los métodos del controlador. Esta dirección se indica dentro del parámetro *path* de *@RequestMapping*.

Cada uno de los métodos del controlador se identifican por un método *HTTP* y una subruta, que tiene como ruta base el parámetro *path* de *@RequestMapping*. Para identificar la *URI* a la que va a responder cada método del controlador he utilizado las anotaciones *@GetMapping* o *@PostMapping*, según si el método responde a una solicitud *HTTP* que utiliza el método *GET* o *POST*.

En la figura 5.18 se muestra el diagrama de clases de *ControllerEPU*. Esta clase contiene dos atributos: *serviceEPU*, de la clase *ServiceEPU*, y *exportServiceEPU*, de la clase *ExportServiceEPU*.

El primer atributo, *serviceEPU*, se utiliza para conseguir los *DTO MV* necesarios. Mientras que el segundo atributo, *exportServiceEPU*, se emplea para poder descargar el archivo *Excel* con la información de las subexplotaciones.

Su constructor *ControllerEPU()* asigna a sus atributos *serviceEPU* y *exportServiceEPU* los parámetros *serviceEPU* y *exportServiceEPU*, respectivamente.

Seguidamente, se explican los métodos de la clase *ControllerEPU*.

El método *mostrarVistaPrincipal()* es el encargado de renderizar el fichero *HTML* ≪vistaPrincipal.html≫, que muestra la vista principal de la consulta de explotaciones. Contiene la anotación *@GetMapping*, que indica que esté método responde a solicitudes *HTTP* con el método *GET*. La *URI* con la que responde este método es ≪/7≫, que se indica en el parámetro *path* de la anotación.

Este método va a representarse mediante un pseudocódigo en el algoritmo 5.

Algorithm 5 Método *mostrarVistaPrincipal()*

```
1:  @GetMapping( path = "/7")
2:  procedure mostrarVistaPrincipal(@CookieValue(value = "JWT") token, model, sesion)
3:      Obtención de perfil con sesion.getAttribute("perfilDTO")
4:      Obtención de perfilId con perfil.getId()
5:      Creación de una instancia dtoMv de la clase BaseDTO
6:
7:      if perfil tiene el identificador del titular then
8:          Asignación en dtoMV con serviceEPU.getTitularMV(token)
9:          Almacenamiento en model con model.addAtribute("uriTabla", /titularDt)
10:     else
11:         if perfil tiene el identificador del veterinario ADSG then
12:             Asignación en dtoMV con serviceEPU.getVetADSGMV(token)
13:             Almacenamiento en model con model.addAtribute("uriTabla", /vetADSGDt)
14:         else
15:             Asignación en dtoMV con serviceEPU.getAdministradorMV(token)
16:             Almacenamiento en model con model.addAtribute("uriTabla", /administradorDt)
17:         end if
18:     end if
19:     if dtoMv.getOk() es verdadero then
20:         Almacenamiento en model con model.addAtribute("mv", dtoMv)
21:         Almacenamiento en model con model.addAtribute("cancelar", "/cancel")
22:         return instancia de la clase ModelAndView("vistaPrincipal.html")
23:     end if
24:     Asignar falso al atributo ok del parámetro model con dtoMv.setOk(false)
25:     return instancia de la clase ModelAndView("error500.html")
26: end procedure
```

Los parámetros de *mostrarVistaPrincipal()* son: *token*, el *JWT* que contiene información del usuario; *model*, objeto de la clase *Model* de *Spring* que permite almacenar información para que sea usada por la vista y además, actúa como un mapa de *Java*, asociando cada valor almacenado con una clave; y *sesion*, objeto de la clase *HttpSession* de *Java* que almacena información en la sesión actual, también en forma de mapa.

La clase de retorno de este método es *ModelAndView*, que indica al motor de plantillas *Thymeleaf* que procese la plantilla indicada como argumento. Posteriormente, *Thymeleaf* la procesa y se devuelve el *HTML* resultante.

En primer lugar, el método *mostrarVistaPrincipal()* obtiene la instancia de un *DTO* llamado *PerfilDTO*, que contiene información sobre qué tipo de usuario se ha conectado: titular, veterinario ADSG o administrador.

La instancia de este *DTO* se obtiene de la sesión, llamando al método *getAttribute()* del parámetro *sesion* con el argumento ≪perfilDTO≫. Dicho método devuelve el objeto vinculado al nombre ≪perfilDTO≫ en la sesión, que en este caso es el *DTO* de la clase *PerfilDTO*. Una vez devuelto, se asigna al objeto con el nombre *perfil*.

Cabe destacar que tanto la definición de la clase *DTO* llamada *PerfilDTO* como el código que guarda la instancia de este *DTO* en sesión ya estaban creados. Por lo tanto, forman parte del desarrollo del portal web de la OVV pero no de la consulta de explotaciones.

Continuando con el método, una vez obtenida la instancia de la clase *PerfilDTO*, se almacena su atributo *id* en una variable llamada *perfilId*. Este atributo contiene el identificador que define el tipo del usuario conectado. Para almacenarlo primero se accede a él, llamando con la variable *perfil* a su método *getId()*.

A continuación, según el tipo del usuario identificado, se llama con el atributo *serviceEPU* a uno de los tres métodos que devuelven un *DTO MV* con los datos que se muestran dentro del bloque de filtros. Si el usuario es un titular, se llama al método *getTitularMV()*. Por otro lado, si es un veterinario ADSG, se llama al método *getVetADSGMV()*, y si es un administrador, a *getAdministradorMV()*.

El *DTO MV* devuelto se asigna a un objeto de la clase *BaseDTO* llamado *dtoMv*, debido a que hereda de esta clase.

Una vez conseguido el *DTO MV*, se almacena con el parámetro *model* una de las tres *URI* indicadas dentro del parámetro *path* en las anotaciones *@PostMapping* de los siguientes métodos del controlador: *datatablesTitulares()* si el usuario es un titular; *datatablesVetADSG()* si es un veterinario ADSG y *datatablesAdministradores()* si es un administrador. El almacenamiento de la *URI* en el parámetro *model* tiene la clave ≪uriTabla≫.

El almacenamiento de información dentro del parámetro *model* se realiza llamando a su método *addAttribute()*, que recibe dos argumentos: el primero, una cadena *String* con la clave del objeto a almacenar; y el segundo, el objeto que se quiere almacenar.

Posteriormente, se comprueba que el atributo *ok* del *DTO MV* es verdadero con su método *getOk()* para verificar que contiene toda la información necesaria. Si es así, se almacena el *DTO*

MV en el parámetro *model* con la clave ≪mv≫. Este *DTO MV* es utilizado posteriormente por la vista inyectar su contenido dentro de la lista desplegable en el bloque de filtros que le corresponda.

Acto seguido, se almacena con la clave ≪cancelar≫ la *URI* con la que responde el método *cancelar()* del controlador. Esta *URI* es utilizada por el componente que representa el botón ≪Cancelar≫ en la vista, que al pulsarlo la llama.

Por último, el método *mostrarVistaPrincipal()* devuelve una nueva instancia de la clase *ModelAndView* de *Spring*, que tiene como argumento la dirección de la plantilla con la vista principal que va a renderizar *Thymeleaf*. Esta plantilla se encuentra en el fichero *HTML* ≪vistaPrincipal.html≫.

En cambio, si el atributo *ok* del *DTO MV* es falso, indica que no contiene la información necesaria a mostrar en la vista. Por tanto, el método *mostrarVistaPrincipal()* devuelve una instancia de la clase *ModelAndView*, que tiene como argumento la dirección de una plantilla que muestra una vista con un mensaje de error.

Los siguientes métodos que se van a explicar del controlador están relacionados con el componente *DataTables*, que representa el cuadro donde se muestra la información sobre las subexplotaciones relacionadas con el usuario.

Este componente tiene activada la opción del procesamiento del lado del servidor. Esta opción permite que el servidor sea el encargado de manejar todos los datos consultados en la base de datos y que sean enviados de manera parcial al componente *DataTables* según los necesite. Esta opción es esencial cuando la cantidad manejada de datos por parte del servidor es muy grande [42].

Con esta opción activada, cuando se realice una acción en el componente, como por ejemplo, ordenar las filas mostradas por una columna o mostrar una página del componente, el propio componente envía una solicitud *AJAX* al controlador, que es el responsable de llamar a la *API REST* para conseguir los datos resultantes de la acción realizada en el componente. Una vez los consigue, son enviados de vuelta al controlador, que se encarga de mostrarlos en el *DataTables*.

El primer método del controlador relacionado con el *DataTables* es *datatablesTitulares()*. Este método se llama cuando un usuario titular realiza una acción en el componente *DataTables*. Una vez ejecutada la acción, el componente manda una solicitud *AJAX* del tipo *POST* con la *URI* indicada en el parámetro *path* de la anotación *@PostMapping* en el método *datatablesTitulares()*. Esta llamada *AJAX* llama al método *dataTablesTitulares()*.

El primer parámetro de *datatablesTitulares()* es la cadena *token*, que contiene el *JWT*. Este parámetro tiene la anotación *@CookieValue* de *Spring*, que se encarga de conseguir la *cookie* almacenada en el navegador que contiene el *JWT*. Para ello, se indica el nombre de la *cookie* en el parámetro *value* de la anotación.

El segundo parámetro es *columnasDatatables*, objeto de la clase *ColumnasDatatables*. Esta clase contiene información sobre las columnas del componente *DataTables* que realiza la petición *AJAX*. Algunos de los datos que almacena son: el nombre de la columna, si la columna se encuentra ordenada y en qué dirección (ascendente o descendente).

Siguiendo con el tercer parámetro, *paginadoDatatables*, se trata de un objeto de la clase *PaginadoDatatables*, encargada de almacenar la información relacionada con la paginación del *DataTables* que realiza la petición *AJAX*. La paginación es una funcionalidad del *DataTables* que le permite dividir sus filas de datos en distintas páginas.

El cuarto parámetro, *draw*, contiene el número de la solicitud *AJAX* realizada por el *DataTables*. Este número garantiza que la respuesta a dicha solicitud pueda devolverse en un orden, debido a que las llamadas *AJAX* son asíncronas y por su naturaleza, pueden llegar fuera de orden.

El parámetro *draw* contiene la anotación *@RequestParam*, con el valor ≪draw≫ en el parámetro *value* de la anotación. Esta anotación permite extraer el valor de la *query* con la clave *draw* dentro de la *URI* que llama al método *datatablesTitulares()*, y una vez extraído, asignárselo al parámetro *draw* [12].

Los dos últimos parámetros son: *sesion*, de la clase *HttpSession* y *dtoEntrada*, *DTO* de entrada de la clase *DTOEntradaTitulares*.

El parámetro *dtoEntrada* almacena las opciones elegidas dentro de las listas desplegables del filtro para titulares. Estas opciones son enviadas dentro de la petición *AJAX* al método *datatablesTitulares()* y posteriormente, son inyectadas en cada atributo del *DTO* de entrada.

El tipo de retorno del método *datatablesTitulares()* es un objeto de la clase *ResponseEntity*, que representa la respuesta *HTTP* enviada al componente *DataTables*. El cuerpo de la respuesta *HTTP* representada por este objeto a su vez almacena un objeto de la clase `DataTablesConvertido<PaginaSubexplotacionDTO>`. Esta clase contiene toda la información de las subexplotaciones adaptada de manera que pueda visualizarse en el componente *DataTables*.

Las clases *ColumnasDatatables*, *PaginadoDatatables* y *DataTablesConvertido* son clases ya creadas y utilizadas en otros trámites de la OVV. Por lo tanto, no han sido creadas por mí, tan solo han sido usadas para crear este controlador.

El objeto *ResponseEntity* que se devuelve en el método *datatablesTitulares()* se consigue llamando al método privado del controlador *procesarDatatables()*. En la llamada a este método privado, se pasan como argumentos todos los parámetros de *datatablesTitulares()*.

Los demás métodos del controlador relacionados con el componente *DataTables* son *datatablesVetADSG()* y *datatableAdministradores()*. Estos dos métodos son prácticamente iguales al método explicado *datatablesTitulares()* pero con dos diferencias.

La primera diferencia de *datatablesVetADSG()* frente a *datatablesTitulares()* es que el primer método contiene en el parámetro *path* de su anotación *@PostMapping* la *URI* que envía el componente *DataTables* cuando es usado por un veterinario ADSG. La segunda es que su parámetro *dtoEntrada*, que es un *DTO* de entrada, es de la clase *DTOEntradaVetADSG*.

Por otro lado, *datatablesAdministradores()* tiene también dos diferencias similares frente a *datatablesTitulares()*. La primera es que el parámetro *path* en su anotación *@PostMapping* contiene la *URI* que envía el componente *DataTables* al ser usado por un administrador. La

segunda es que su parámetro *dtoEntrada* es de la clase *DTOEntradaAdministradores*.

Acto seguido, va a explicarse el método privado del controlador *procesarDatatables()*. Este método va a representarse en pseudocódigo en el algoritmo 6.

Algorithm 6 Método *procesarDatatables()*

1: **procedure** *procesarDatatables*(*token*, *columnasDatatables*, *paginadoDatatables*, *draw*, *sesion*, *dtoEntrada*)
2: **if** *dtoEntrada.getOk()* es verdadero **then**
3: Almacenamiento en *sesion* con *sesion.setAttribute("entradaDTO", dtoEntrada)*
4: Obtención de *paginaDTO* con *serviceEPU.getPaginaSubexplotacionesDTO(token, pagina, size, direction, columna)*
5: **if** *paginaDTO.getOk()* es verdadero **then**
6: **return** instancia de la clase *ResponseEntity* con un cuerpo que tenga una instancia de *DataTablesConvertido* con los argumentos: *paginaDTO.getListaSubexplotaciones()*, *draw* y *columnasDatatables*.
7: **end if**
8: **return** instancia de la clase *ResponseEntity* con un cuerpo que tenga una instancia de *DataTablesConvertido* con los argumentos: lista vacía, *draw* y *columnasDatatables*.
9: **end if**
10: **return** instancia de la clase *ResponseEntity* con el cuerpo vacío
11: **end procedure**

Primero, el método comprueba si el atributo *ok* del parámetro *dtoEntrada* es verdadero para verificar que contiene toda la información necesaria. Si no es verdadero, no puede ejecutarse el método en condiciones y se devuelve un objeto de la clase *ResponseEntity* con un cuerpo vacío. Por otro lado, si es verdadero, el método continua.

Una vez comprobado que el atributo *ok* es verdadero, se almacena en sesión el parámetro *dtoEntrada* con la clave ≪entradaDTO≫. Esto se realiza porque va a ser usado posteriormente por los métodos *exportarExcelCategorias()* y *exportarExcelSinCategorias()*.

Después, se llama con el atributo *serviceEPU* al método *getPaginaSubexplotacionesDTO()* para conseguir el *DTO REST PaginaSubexplotacionesDTO*, que contiene en su atributo *listaSubexplotaciones* la lista con la información de las subexplotaciones que se van a enviar al componente *DataTables*. El objeto devuelto por este método se asigna a una variable llamada *paginaDTO* en el método.

A la llamada al método *getPaginaSubexplotacionesDTO()* se le pasan como argumentos: el parámetro *token*, el atributo *pagina* de la clase *PaginadoDatatables*, que contiene el número de la página que se quiere recibir; el atributo *size* de la clase *PaginadoDatatable*, que almacena el número máximo de filas mostradas; el atributo *direction* de la clase *PaginadoDatatable*, cadena que identifica si la ordenación es ascendente o descendente; y finalmente, el atributo *columna* de la clase *PaginadoDatatable*, que contiene el nombre de la columna por la que se va a realizar la ordenación.

Los atributos de las clase *PaginadoDatatable* pasados como argumento al método *getPaginaSubexplotacionesDTO()* son conseguidos con sus respectivos *Getters*. Para simplificar esta

sentencia en la línea cuatro del pseudocódigo indicado en el algoritmo 6 se han indicado los nombres de estos atributos en lugar de sus *Getters*.

Posteriormente, se comprueba que el *DTO REST* devuelto en el método *getPaginaSubexplotacionesDTO()* contenga la información solicitada. Esto se realiza comprobando su atributo *ok*.

Si el atributo *ok* es falso, se devuelve un objeto de la clase *ResponseEntity* que contiene en su cuerpo un objeto de la clase *DataTablesConvertido* instanciado con los siguientes argumentos: una lista vacía, que indica no se muestre ninguna subexplotación en el *DataTables*; el parámetro *draw*, y por último, el parámetro *columnasDatatables*.

En cambio, si el atributo *ok* es verdadero, se devuelve un objeto de la clase *ResponseEntity* que contiene en su cuerpo un objeto de la clase *DataTablesConvertido* instanciado con los argumentos: la lista con las subexplotaciones, obtenida llamando con la variable *paginaDTO* al método *getListaSubexplotaciones()*; el parámetro *draw* y finalmente, el parámetro *columnasDatatables*.

El método *exportarExcelCategorias()* tiene el objetivo de devolver un objeto de la clase *ResponseEntity*, que almacena en su cuerpo una lista de bytes con el contenido del archivo *Excel* que contiene la información de las subexplotaciones con sus categorías. Contiene los siguientes parámetros: *token*, con el *JWT*; *model*, objeto de la clase *Model*; y *sesion*, objeto de la clase *HttpSession*.

En el algoritmo 7 se enseña el pseudocódigo del método *exportarExcelCategorias()*.

Algorithm 7 Método *exportarExcelCategorias()*

1: **procedure** *exportarExcelCategorias*(*token*, *columnasDatatables*, *paginadoDatatables*, *draw*, *sesion*, *dtoEntrada*)
2: Obtención de *dtoEntrada* con *sesion.getAttribute("dtoEntrada")*
3: Obtención de *contenidoExcel* con *exportService.getExcelCategorias(token, dtoEntrada)*
4: **if** *contenidoExcel* es una lista de *bytes* vacía **then**
5: **return** instancia de la clase *ResponseEntity* con el código *HTTP 500* y una lista de bytes vacía en su cuerpo.
6: **end if**
7: Obtención de *cabeceras* con *getCabecerasExcel()*
8: **return** instancia de la clase *ResponseEntity* con las cabeceras en la variable *cabeceras* y la lista *contenidoExcel* en su cuerpo
9: **end procedure**

Primero, el método utiliza el parámetro *sesion* para conseguir el *DTO* de entrada almacenado en sesión en el método *procesarDatatables()* del propio controlador. Después, con el atributo *exportService* se llama al método *getExcelCategorias()*, pasándole como argumentos el parámetro *token* y el *DTO* de entrada conseguido de sesión. Este método devuelve la lista de bytes con el contenido del *Excel* y lo asigna a una variable llamada *contenidoExcel*.

Acto seguido, se comprueba si la lista de bytes devuelta está vacía, si es así, indica que ha habido un error al solicitar la lista del bytes y devuelve un objeto de la clase *ResponseEntity*

con el código *HTTP 500*, que indica un error interno en el servidor, y la lista vacía en su cuerpo.

En caso contrario, se llama al método del controlador *getCabecerasExcel()*, que devuelve un objeto de la clase *HttpHeaders* con las cabeceras necesarias para la respuesta *HTTP* que va a contener el contenido del *Excel*. Además, se le llama con dos argumentos: el nombre del fichero *Excel* y la longitud de la lista devuelta. Las cabeceras se asignan en una variable con el nombre *cabeceras*.

Una vez conseguido el objeto de la clase *HttpHeaders*, se crea un objeto *ResponseEntity* con las cabeceras indicadas en la variable *cabeceras* y un cuerpo con la lista de bytes en la variable *contenidoExcel*. Finalmente, este objeto es devuelto en el método.

El método *exportarExcelSinCategorias()* es igual que *exportarExcelCategorias()*, la única diferencia es que con el atributo *exportService* se llama al método *getExcelSinCategorias()* en lugar de *getExcelCategorias()*.

Uno de los métodos restantes por explicar del controlador es *getDetalleSubexpDTO()*, el cual tiene la anotación *@ModelAttribute* en la definición de su método. La función de este método es devolver el *DTO MV DetalleSubexplotacionDTO*, que es posteriormente inyectado en uno de los parámetros del método *mostrarDetalle()* en el controlador.

El método *getDetalleSubexpDTO()* se representa con el pseudocódigo en el algoritmo 8.

Algorithm 8 Método *getDetalleSubexpDTO()*

```
1: @ModelAttribute
2: procedure getDetalleSubexpDTO(@CookieValue(value = "JWT") token, @PathVariable(value =
   "subexpId") subexpId)
3:     Creación de una instancia dtoDetalle de la clase DetalleSubexplotacionDTO
4:     if subexpId no es nulo then
5:         Asignación de dtoDetalle con serviceEPU.getDetalleSubexplotacionDTO(token, subexpId)
6:     end if
7:     return dtoDetalle
8: end procedure
```

Los parámetros de *getDetalleSubexpDTO()* son: *token*, cadena que capta el *JWT* almacenado con la anotación *@CookieValue*; y *subexpId*, cadena que contiene la anotación *@PathVariable* con el valor ≪subexpId≫ en el parámetro *value* de la anotación.

La anotación *@PathVariable* indica que el parámetro *subexpId* va a almacenar el identificador de la subexplotación indicada en la expresión ≪{subexpId}≫. Esta expresión está contenida en la *URI* que llama al método *mostrarDetalle()*.

Por otro lado, va a describirse el método *mostrarDetalle()*, que se relaciona con el método *getDetalleSubexpDTO()*. El método *mostrarDetalle()* se muestra en pseudocódigo en el algoritmo 9.

La anotación del método *mostrarDetalle()* es *@GetMapping*, que contiene la *URI* ≪/detalle/{subexpId}/≫.

El primer parámetro de *mostrarDetalle()* es *token*, cadena que capta el *JWT* almacenado

Algorithm 9 Método *mostrarDetalle()*

```
1: @GetMapping(path = "/detalle/{subexpId}")
2: procedure mostrarDetalle(@CookieValue(value = "JWT") token, @ModelAttribute detalleSubexpDTO)
3:     if detalleSubexpDTO.getOk() es falso then
4:         return "~{}"
5:     end if
6:     return "fragmentoDetalle :: detalle"
7: end procedure
```

con la anotación *@CookieValue*. El segundo parámetro es *detalleSubexpDTO*, objeto del *DTO MV DetalleSubexplotacionDTO*. Este parámetro contiene la anotación *@ModelAttribute*.

A continuación va a explicarse el funcionamiento del método *mostrarDetalle()*.

Primero, el controlador recibe una solicitud *HTTP* con la *URI* del método *mostrarDetalle()* y a continuación, almacena el identificador de la subexplotación en la expresión ≪{subexpId}≫ de la *URI* recibida en el parámetro *subexpId* de *getDetalleSubexpDTO()*. Este almacenamiento es debido a que tanto *getDetalleSubexpDTO()* como el parámetro *subexpId* contienen la anotación *@ModelAttribute*.

Una vez almacenado el identificador, comienza la ejecución del método *getDetalleSubexpDTO()*.

La primera acción del método *getDetalleSubexpDTO()* es crear una instancia del *DTO MV DetalleSubexplotacionDTO* con el nombre *dtoDetalle*. Después, se comprueba que el parámetro *subexpId* no es nulo. Si no lo es, se llama al método *getDetalleSubexplotacionDTO()* con el atributo *serviceEPU* y los parámetros *token* y *subexpId*. La llamada a este método se asigna a la instancia *dtoDetalle*.

Una vez conseguido el *DTO MV*, el método *getDetalleSubexpDTO()* lo devuelve y automáticamente se inyecta en el parámetro *detalleSubexpDTO* del método *mostrarDetalle()*. Además, se guarda en el modelo para que pueda ser posteriormente usado en la vista.

Una vez se realiza la inyección del *DTO MV*, el método *mostrarDetalle()* comienza a ejecutarse.

El primer paso que ejecuta *mostrarDetalle()* es comprobar si el atributo *ok* del parámetro *detalleSubexpDTO* es falso. Si lo es, el *DTO MV* no contiene la información necesaria y devuelve la cadena ≪~{}≫, que indica un fragmento de *Thymeleaf* vacío.

En cambio, si el atributo *ok* es verdadero, se devuelve una cadena con la dirección de una plantilla de *Thymeleaf* que contiene un fragmento con los componentes *HTML* específicos de la vista que muestra el detalle de una subexplotación.

La última función del controlador es *cancelar()*, el cual tiene el objetivo de redirigir al usuario fuera del trámite de la consulta de explotaciones y borrar un dato añadido en sesión.

La anotación del método es *@GetMapping*. Además, este método solo tiene un único parámetro *sesion*, de la clase *HttpSession*.

Al ejecutar el método *cancelar()* se elimina de la sesión el *DTO* de entrada almacenado con la clave «entradaDTO» en el método *procesarDatatables()*. Esto se realiza llamando con el parámetro *sesion* al método *removeAttribute()*, que recibe como argumento la clave «entradaDTO».

Finalmente, el método *cancelar()* devuelve una instancia de la clase *RedirectView* de *Spring*. La instancia creada recibe como argumento una cadena con la *URI* «/consultas». Al devolver esta instancia se redirige al usuario a la *URI* «/consultas» que muestra los distintos trámites en el apartado consultas de la OVV.

Fichero *JavaScript* del proyecto

El fichero *JavaScript* creado en el proyecto contiene diferentes funciones que se complementan con el controlador para que produzca resultados en la vista de la consulta. En esta parte van a explicarse estas funciones.

La primera función del fichero es *iniciarDatatables()*, que se ejecuta al mostrar la vista principal de la consulta de explotaciones. Esta función envía una llamada *AJAX* con el método *POST* a una de las tres *URI* asignadas a los métodos del controlador según el tipo de usuario que realice una acción en el componente *DataTables*: *datatablesTitulares()*, si realiza la acción un titular; *datatablesVetADSG*, en el caso de un veterinario ADSG; y *datatablesAdministradores()* en el caso de los administradores.

Esta llamada *AJAX* envía una serie de valores. Primero, cada uno de los identificadores de las opciones elegidas dentro de las listas desplegables en el bloque de filtros del usuario que realiza la acción. Además, la información del paginado y de las columnas del *DataTables*.

La segunda función se ejecuta al presionar el botón «Aplicar», que se encuentra dentro del bloque de filtros. Esta función indica al cuadro del *DataTables* que recargue los datos del cuadro realizando otra llamada *AJAX* al servidor, de la misma manera que se ha realizado en la función *iniciarDatatables()*.

La tercera función se ejecuta al presionar el botón «Limpiar» en el bloque de filtros. Esta función limpia el contenido seleccionado en cada una de las listas desplegables del bloque de filtros y refresca el cuadro realizando una petición *AJAX* mediante el componente *Datatables*.

La cuarta función se ejecuta al presionar alguno de los botones «Si» o «No» que se muestran en la pantalla de confirmación al presionar el botón «Descargar Excel» en la consulta de explotaciones.

Si se presiona el botón «Si», indica que se va a descargar el fichero *Excel* con el detalle de las categorías en las subexplotaciones, por tanto la función realiza una llamada *AJAX* con el método *GET* a la *URI* indicada en el método *exportarExcelCategorias()* del controlador. Esta llamada *AJAX* permite que el fichero *Excel* se descargue en el navegador.

En cambio, si se presiona el botón ≪No≫, se va a descargar el fichero *Excel* sin el detalle de las categorías. Entonces, la función realiza una llamada *AJAX* con el método *GET* a la *URI* indicada en el método *exportarExcelSinCategorias()* del controlador. Con esta llamada, el fichero *Excel* se descarga en el navegador.

La cuarta función se ejecuta al presionar el botón ≪Cancelar≫ de la vista principal. Al presionarlo, se realiza una llamada *AJAX* a la *URI* del método del controlador *cancelar()*. Gracias a esta llamada, el usuario sale del trámite de la consulta de explotaciones.

La quinta función muestra el detalle de una subexplotación al presionar alguno de los botones con el símbolo ≪+≫ mostrado en las filas del componente *Datatables*. Una vez presionado este botón, se captura el identificador de la subexplotación en la fila. Una vez capturado, se realiza una petición *AJAX* del tipo *GET* a la *URI* indicada en el método *mostrarDetalle()* del controlador. Esta *URI* va a contener el identificador de la subexplotación capturada.

Si la petición *AJAX* es satisfactoria, se procede a mostrar el detalle de la subexplotación. Para ello, primero se ocultan los componentes *HTML* específicos de la vista principal y se hace visible un botón con el título ≪Anterior≫, que permite volver a la vista principal. Después, se muestran los componentes específicos del detalle de una subexplotación.

La sexta y última función se ejecuta al presionar el botón ≪Anterior≫ contenido en la vista del detalle de una subexplotación. Al presionarlo, se ocultan los componentes que muestran el detalle de una subexplotación y se vuelven a mostrar los componentes de la vista principal.

Componentes de la capa vista

La capa vista se compone de dos vistas. La primera es la vista que se muestra al entrar al trámite de la consulta de explotaciones, donde aparecen las subexplotaciones relacionadas con el usuario. Por otro lado, la segunda vista es la que muestra información adicional sobre una de las subexplotaciones mostradas en el componente *DataTables*.

Estas dos vistas comparten una estructura de bloques, que se divide en tres partes: una cabecera, un bloque principal y un pie de página. La cabecera y el pie de página son bloques de información que no cambian según la vista que se esté mostrando. En cambio, el bloque principal contiene contenido específico de la vista que está mostrando.

La cabecera contiene los componentes *HTML* mostrados en la parte frontal de la vista. Estos componentes son: un menú con los distintos trámites de la OVV, un cuadro con información del usuario y otro cuadro con el DNI del usuario y un botón para cerrar la sesión.

El segundo bloque es el bloque principal, que va a renderizar unos u otros componentes *HTML* según la vista mostrada.

El tercer bloque es el pie de página, que contiene una barra vertical con el texto ≪Oficina Virtual Veterinaria≫.

En la figura 5.19 se muestra una captura de la vista donde se muestran las subexplotaciones,

donde cada bloque está identificado con un número: un uno para la cabecera, un dos para el bloque principal y un tres para el pie de página.

Por otro lado, en la figura 5.20 se presenta una captura con la vista que muestra el detalle de una subexplotación. En esta figura también están indicados los distintos bloques con números, al igual que la figura 5.19.

Para representar la capa vista como una estructura de bloques se han utilizado las plantillas de *Thymeleaf*.

Una plantilla *Thymeleaf* es un archivo *HTML* que contiene elementos propios de *HTML* y, dentro de ellos, atributos y expresiones de *Thymeleaf*. El contenido de *Thymeleaf* incluido es dinámico, ya que el motor de plantillas de *Thymeleaf* lo sustituye al procesar la plantilla en el servidor. Después de procesarse, la plantilla completa se muestra en el navegador web.

Existe una plantilla de *Thymeleaf* principal, ubicada en el fichero «vistaPrincipal.html», que va a mostrar los dos tipos de vistas y que contiene la estructura de bloques ya mencionada.

El contenido de cada uno de los bloques va a estar definido dentro un fragmento de una plantilla de *Thymeleaf* diferente y va a tomar el nombre de fragmento.

Por lo tanto, la plantilla de *Thymeleaf* principal va a tener definidos componentes *HTML* que al renderizarse por *Thymeleaf* serán reemplazados por los componentes dentro de los fragmentos.

En la figura 5.21 se enseña un ejemplo del código de una plantilla de *Thymeleaf* con un fragmento. Este fragmento está definido dentro del componente *div* de *HTML* en la línea ocho, que contiene un atributo con la expresión «th:fragment="fragmento1"». Esta expresión indica que el componente *div* y todo su contenido incluido es un fragmento de *Thymeleaf* con el nombre «fragmento1».

Este fragmento se va a reemplazar en el código mostrado en la figura 5.22, que contiene un ejemplo del código de una plantilla de *Thymeleaf* que actúa como una plantilla principal.

La línea ocho en el código mostrado en la figura 5.22 contiene un componente *div* con la expresión «th:replace="ejemploFragmento :: fragmento1"». Esta expresión reemplaza el elemento donde se encuentra por un fragmento, definido de la siguiente manera: «ruta :: nombre». La dirección del fragmento está indicada en «ruta» y el nombre del fragmento en «nombre».

A continuación se listan los ficheros donde están definidos los fragmentos de *Thymeleaf* utilizados en la vista principal:

- «cabecera.html»: contiene el fragmento con los componentes *HTML* de la cabecera.
- «explotaciones.html»: contiene el fragmento con los componentes *HTML* de vista donde se muestran las subexplotaciones.
- «detalle.html»: contiene el fragmento con los componentes *HTML* de la vista con el detalle de una subexplotación.
- «piePagina.html»: contiene el fragmento con los componentes *HTML* del pie de página.

Los cuatro fragmentos se incluyen dentro de la plantilla en el fichero «vistaPrincipal.html». En ella, los fragmentos definidos en los ficheros «cabecera.html» y «explotaciones.html» siempre se van a mostrar. En cambio, los fragmentos en «explotaciones.html» y «detalle.html» se irán alternando según la vista que se muestre.

Dentro del bloque principal de la figura 5.19, que contiene los componentes del fragmento «explotaciones.html», se encuentra el bloque de filtros, que está rodeado con un círculo azul.

Los componentes de cada bloque de filtros están definidos como un fragmento dentro de una plantilla *Thymeleaf* diferente. Estos fragmentos están incluidos en la plantilla «explotaciones.html», que renderizará uno de ellos según el tipo del usuario conectado.

Los ficheros con los fragmentos que almacenan los componentes de un bloque de filtros son:

- «filtroTitulares.html»: almacena el fragmento con los componentes *HTML* del bloque de filtros para titulares.
- «filtroVetADSG.html»: almacena el fragmento con los componentes *HTML* del bloque de filtros para los veterinarios ADSG.
- «filtroAdministradores.html»: almacena el fragmento con los componentes *HTML* del bloque de filtros para los administradores.

Otro aspecto importante utilizado de *Thymeleaf* es el acceso a la información añadida en el modelo, el cual está representado con la clase *Model* de *Spring*.

Esta cualidad permite mostrar el contenido de los *DTO MV* dentro de las componentes en la vista. En concreto, permiten introducir los datos de los *DTO MV* de las clases *TitularMV*, *VetADSGMV* y *AdministradorMV* dentro las listas desplegables en el bloque de filtros que corresponde a cada uno.

También permite añadir el contenido del *DTO MV* de la clase *DetalleSubexplotacionDTO* dentro de los componentes de la vista que muestra el detalle de una subexplotación.

En la figura 5.23 se muestra un ejemplo de código de una plantilla de *Thymeleaf* que utiliza el modelo para acceder a un dato almacenado en él y plasmarlo en una lista desplegable y un campo de entrada.

El modelo contiene un objeto de la clase *Pruebas*, que tiene dos atributos: una cadena *String* con el nombre *campoInput* y una lista llamada *lista* que contiene objetos de la clase *Prueba*.

Por otro lado, la clase *Prueba* tiene dos atributos: una cadena *id* y otra cadena *descripcion*.

La línea ocho del código accede al objeto de la clase *Pruebas* almacenado en el modelo. En esta línea, el atributo «th:object» de *Thymeleaf* indica que dentro de su componente *div* de *HTML* se puede acceder al objeto con el nombre *objetoModelo*. Este objeto ha sido incluido anteriormente en su controlador, usando el método *addAtribute()* de la clase *Model* de *Spring*, que representa el modelo.

Dentro de la línea 10 a la 14 se encuentra un componente *select* de *HTML*, que representa una lista desplegable. La expresión ≪th:each≫ en la línea 11 recorre cada elemento del atributo *lista* de *objetoModelo* y crea un componente *option* dentro del *select* por cada elemento recorrido.

Cada *option* creado tiene una expresión ≪th:value≫, que establece el valor de la opción indicada en el atributo *id*, y ≪th:text≫, que establece el contenido del atributo *descripcion* como el texto mostrado en cada opción.

Por otro lado, en la línea 17 existe un componente *input*, que representa un campo de entrada. Un ejemplo de campo de entrada son los componentes en la vista del detalle de una subexplotación donde se muestra un atributo del *DTO MV DetalleSubexplotacionDTO*.

El *input* en la línea 17 tiene la expresión ≪th:field≫, que permite mostrar el atributo *campoInput* de *objetoModelo* dentro del *input*.

5.3. Pruebas

En la fase de pruebas se realizaron un tipo de pruebas funcionales, las llamadas pruebas de humo. Una prueba de humo es la realización de una serie de acciones manuales por parte del desarrollador para comprobar el correcto funcionamiento del programa. En concreto, estas acciones fueron probar todos los componentes de la interfaz de usuario [16].

Primero, se hicieron pruebas de humo en la vista principal y después en la vista con el detalle de una subexplotación. En la vista principal se comprobaron todos los componentes dentro de los tres tipos de bloques de filtros. Los primeros componentes probados fueron las listas desplegables, desde las que tienen una barra de búsqueda en ellas a las que permiten una selección simple o múltiple de opciones dentro de ellas.

Después, se utilizaron los botones de ≪Limpiar≫ y ≪Aplicar≫. El botón consiguió borrar las opciones seleccionadas dentro de las listas desplegables y el segundo mostrar dentro del componente *DataTables* las subexplotaciones que cumplen los criterios indicados en las listas desplegables.

Después, se verificaron que las subexplotaciones aparecen de manera correcta en el componente *DataTables*, además de que la paginación y la ordenación de las columnas de este componente también funcionaran.

En las siguientes pruebas de humo se hicieron clic en los botones ≪Cancelar≫ y ≪Descargar Excel≫. El primer botón redirigió al usuario a la pantalla con las consultas de la OVV y el segundo botón descargó los dos tipos de archivo *Excel*.

La última prueba en la vista principal fue hacer clic en el botón ≪+≫ dentro de alguna fila mostrada del componente *Datatables*, que consiguió mostrar la vista con el detalle de una subexplotación.

Las única prueba de humo en la vista del detalle fue presionar el botón ≪Anterior≫, que

vuelve a mostrar la vista principal.

La realización de todas estas pruebas de humo dio como resultado una aplicación web libre de errores y funcional.

Se realizó únicamente este tipo de pruebas porque es el sistema llevado a cabo por el equipo donde trabajé en este proyecto. Según su sistema, después de la fase de pruebas, el proyecto se despliega para su uso por parte de los usuarios finales. En cambio, mi proyecto no se desplegó porque aún no estaba finalizada la OVV. Esto ocasionó que no hubiera una validación real por los usuarios finales.

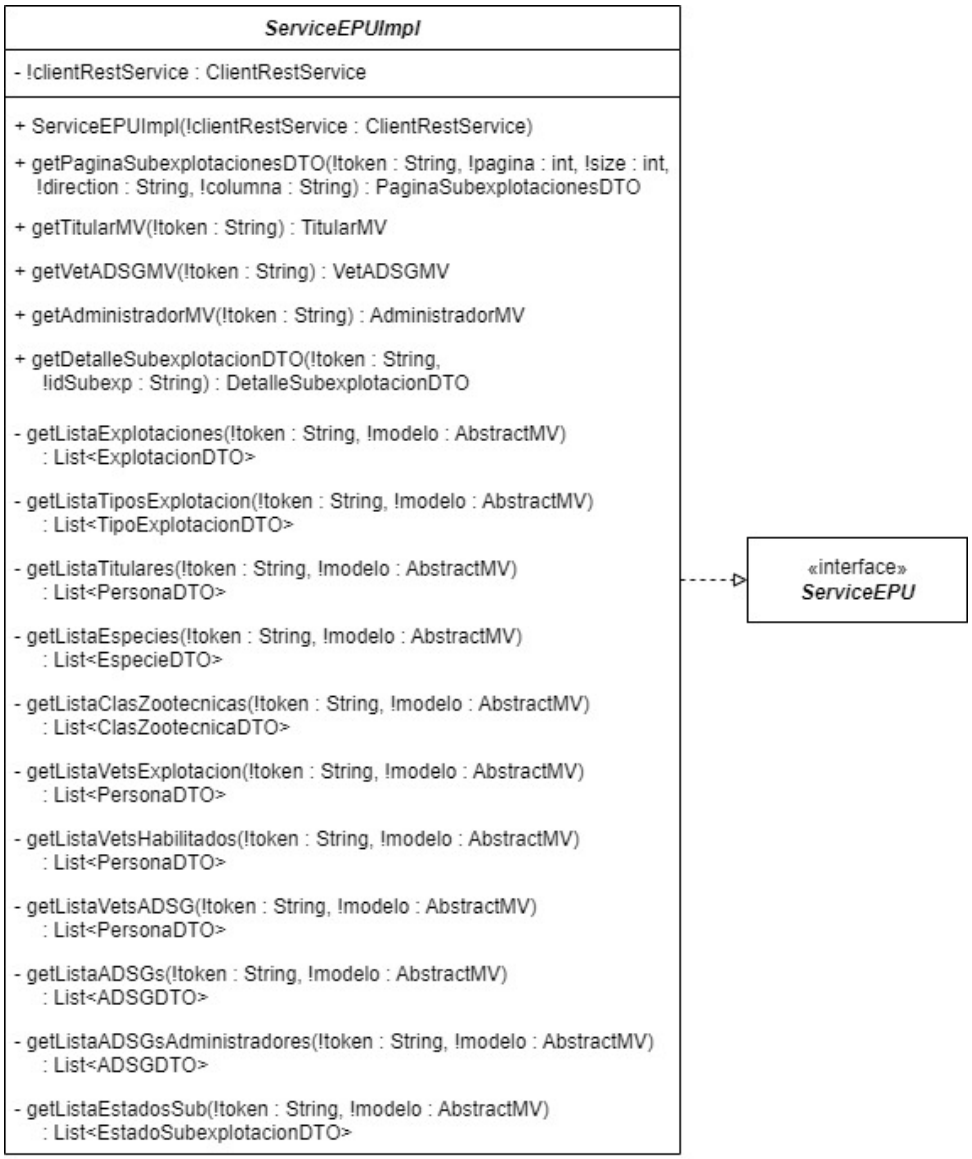


Figura 5.16: Diagrama de clases de la clase *ServiceEPUImpl*.

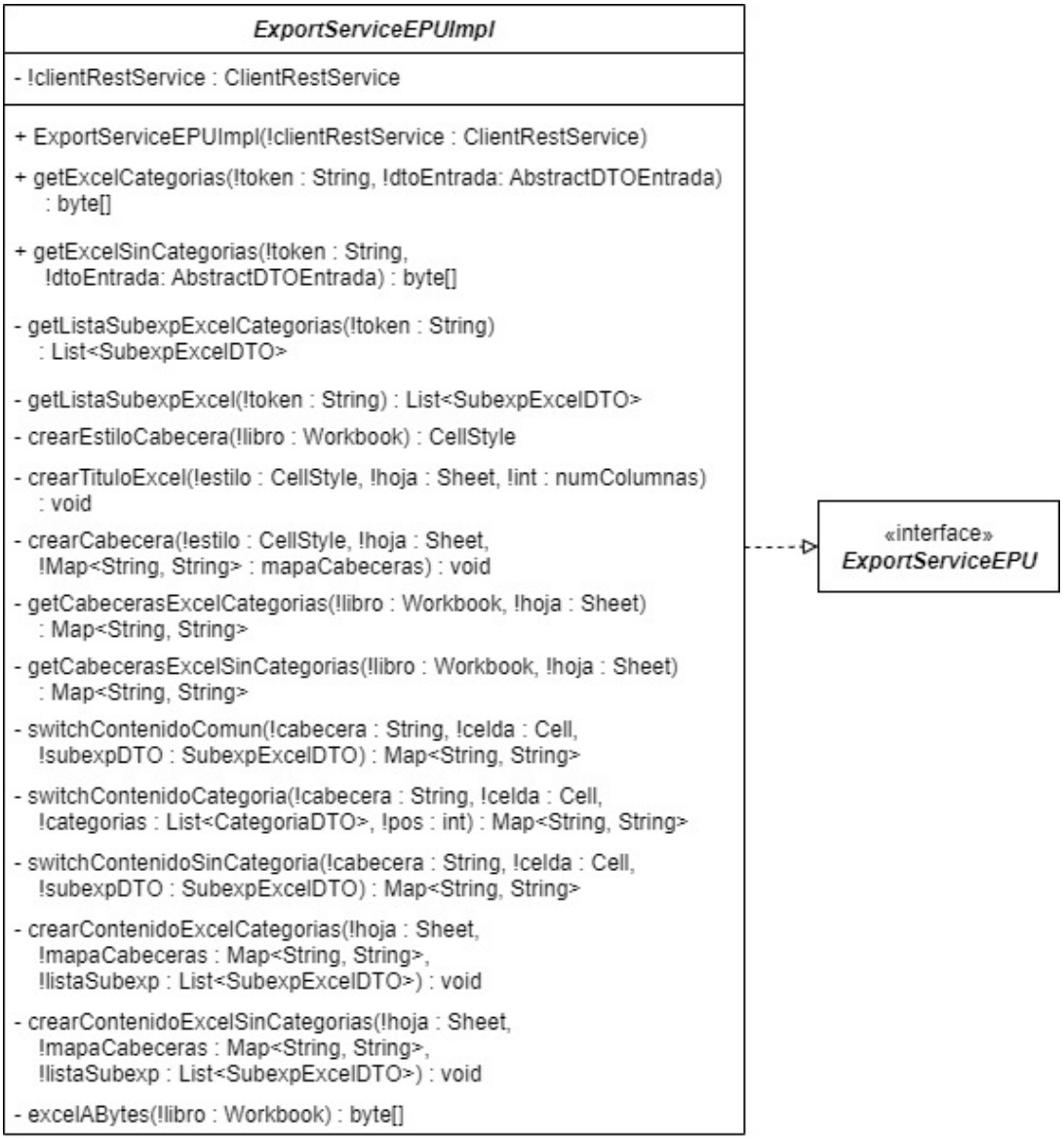


Figura 5.17: Diagrama de clases de la clase *ExportServiceEPUImpl*.

Figura 5.18: Diagrama de clases de la clase *ControllerEPU*.

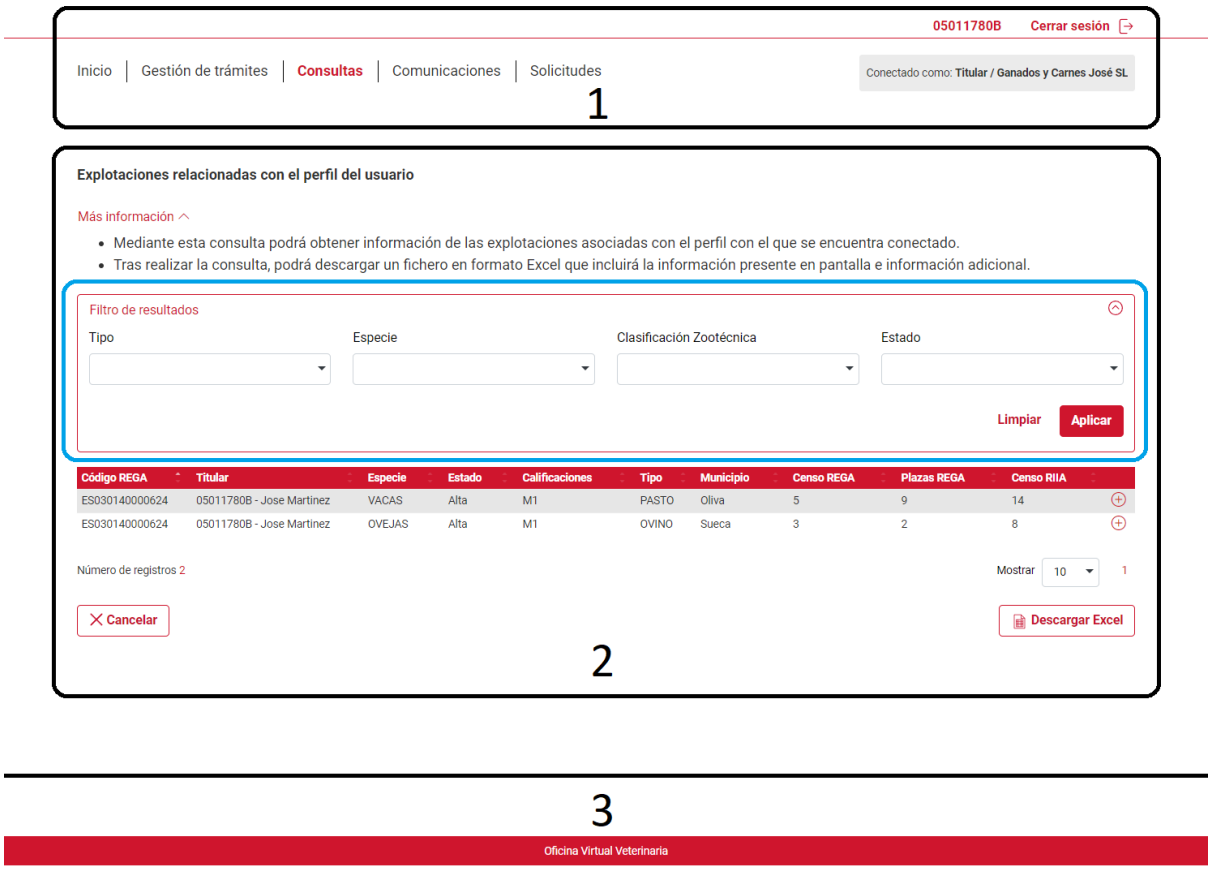


Figura 5.19: Bloques de componentes en la vista donde se muestran las subexplotaciones.

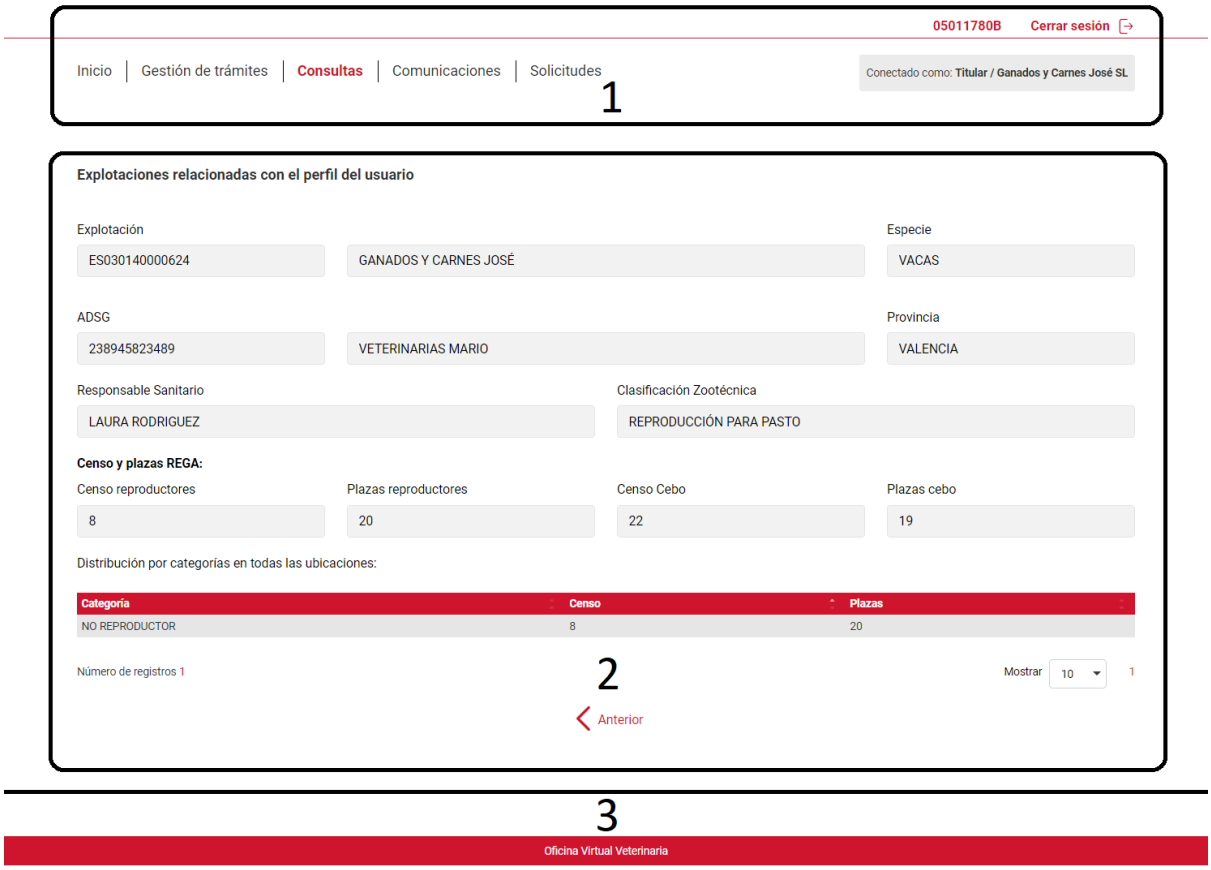


Figura 5.20: Bloques de componentes en la vista con el detalle de una subexplotación.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="ISO-8859-1">
5      <title>Título</title>
6  </head>
7  <body>
8      <div th:fragment="fragmento1">
9          <h1>Hola mundo!</h1>
10     </div>
11 </body>
12 </html>
```

Figura 5.21: Ejemplo de plantilla *Thymeleaf* con un fragmento.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>Título</title>
</head>
<body>
    <div th:replace="ejemploFragmento :: fragmento1">
        <h1>Voy a ser reemplazado!</h1>
    </div>
</body>
</html>
```

Figura 5.22: Ejemplo de una plantilla principal de *Thymeleaf*.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="ISO-8859-1">
    <title>Título</title>
</head>
<body>
    <div th:object="${objetoModelo}">
        <div>
            <select id="atributo1">
                <option th:each="cadaObjeto : *{lista}"
                        th:value="${cadaObjeto.id}"
                        th:text="${tipoExp.descripcion}"
            </select>
        </div>
        <div>
            <input th:field="*{campoInput}" type="text">
        </div>
    </div>
</body>
</html>
```

Figura 5.23: Ejemplo de una plantilla de *Thymeleaf* donde se accede a un dato en el modelo.

Capítulo 6

Conclusiones

Tanto el proyecto desarrollado como la redacción de este Trabajo de Final de Grado ha sido un gran reto, del cual he aprendido habilidades valiosas tanto a nivel profesional, técnico y personal.

En cuanto al nivel profesional, gracias a la estancia de prácticas en la empresa Sopra Steria España, he conocido la experiencia de trabajar en el ámbito de la informática. Gracias a ello, he podido vivir como es trabajar en un proyecto en equipo, donde habilidades como la comunicación o coordinación juegan un papel fundamental.

El desarrollo de un proyecto donde desconocía la mayoría de las tecnologías ha aumentado mi conocimiento sobre ellas. Un ejemplo de ellas es el *framework Spring*. Mi opinión sobre el desarrollo de un proyecto en *Spring* ha sido muy positiva, debido a que es un *framework* muy potente y no ha sido complicado aprenderlo, teniendo en cuenta mi conocimiento previo del lenguaje de programación *Java*. Otras tecnologías como el motor de plantillas *Thymeleaf* o *JavaScript* fueron un reto mayor al ser completamente nuevas para mí. Aún así, aunque fuera un reto grande, se consiguieron cubrir todos los objetivos definidos.

A nivel personal, una lección adquirida en la creación de este documento es la mejora en la capacidad de redacción, porque a nivel profesional es una habilidad más que necesaria. También aprendí el papel que juega la paciencia en la redacción de un documento de este tipo, debido a la gran cantidad de horas invertidas.

Por otro lado, he disfrutado del aprendizaje de como escribir un documento en LaTeX, tecnología que tengo pensado utilizar para documentos futuros.

Bibliografía

[1] *Amazon.com, Inc.* (s.f.). *CABLEPELADO Papelera Malla Metalica 12 litros.* https://www.amazon.es/CABLEPELADO-Papelera-Malla-Metalica-litros/dp/B08W2PXZWP/ref=sr_1_1_sspa?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=papelera+oficina&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1. Recuperado el: 17 de mayo de 2023.

[2] *Amazon.com, Inc.* (s.f.). *HOMCOM Mesa de Ordenador.* https://www.amazon.es/HOMCOM-Escritorio-Ajustables-Mobiliario-120x60x76cm/dp/B07VBCWZ4T/ref=sr_1_4_sspa?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=mesa+oficina&sr=8-4-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1. Recuperado el: 17 de mayo de 2023.

[3] *Amazon.com, Inc.* (s.f.). *HP 435 MLTDVC WRLS Mouse.* https://www.amazon.es/HP-435-Multi-Device-Wireless-Mouse/dp/B09NB3Y4QP. Recuperado el: 12 de mayo de 2023.

[4] *Amazon.com, Inc.* (s.f.). *HP PROBOOK 650 G8 I5-1135G7 SYST.* https://www.amazon.es/HP-PROBOOK-650-I5-1135G7-SYST/dp/B08QYZZWC3/ref=sr_1_2?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=HP+PROBOOK+650+G8+I5-1135G7+SYST&sr=8-2. Recuperado el: 12 de mayo de 2023.

[5] *Amazon.com, Inc.* (s.f.). *Logitech K120.* https://www.amazon.es/Logitech-K120-Business-Teclado-conectividad/dp/B0056C7W1C/ref=sr_1_17?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=teclado+oficina&sr=8-17. Recuperado el: 12 de mayo de 2023.

[6] *Amazon.com, Inc.* (s.f.). *Philips Monitors - Monitor 243V7QDSB/00- 24".* https://www.amazon.es/Philips-243V7QDSB-00-resoluci%C3%B3n-tecnolog%C3%ADa/dp/B06Y13QBYL/ref=sr_1_6?keywords=pantalla&sr=8-6. Recuperado el: 12 de mayo de 2023.

[7] *Amazon.com, Inc.* (s.f.). *Silla de Oficina Giratoria Escritorio con Soporte Lumbar Sillon Ruedas Despacho.* https://www.amazon.es/T-LoVendo-TLV-S1-Giratoria-Escritorio-Despacho/dp/B079F5Z3RN/ref=sr_1_3_sspa?keywords=sillas+ergonomicas+oficina&sr=8-3-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1. Recuperado el: 17 de mayo de 2023.

[8] *Baeldung* (23 de noviembre de 2022). *A comparison between Spring and Spring Boot.* https://www.baeldung.com/spring-vs-spring-boot#:~:text=Spring%20Boot%20is%20basically%20an,and%20more%20efficient%20development%20ecosystem. Recuperado el: 25 de abril de 2023.

[9] *Baeldung* (31 de marzo de 2022). *An Intro to the Spring DispatcherServlet — Baeldung*. https://www.baeldung.com/spring-dispatcherservlet. Recuperado el: 20 de junio de 2023.

[10] Bascón Pantoja, E. *El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing*. *Acta Nova*, 2, 12 2004.

[11] Belmonte Pérez, Ó. y Granell Canut, C. y Erdozain Navarro, M. D. C. *Desarrollo de proyectos informáticos con tecnología Java*, (s.f.). https://www3.uji.es/~belfern/libroJava.pdf.

[12] Berners-Lee, T. y Fielding, R. T. y Masinter, L. M. *Uniform Resource Identifier (URI): Generic Syntax*. *RFC* 3986, 2005.

[13] *Boletín Oficial del Estado, 97* de mayo de 2023. Real Decreto 486/1997, de 14 de abril, por el que se establecen las disposiciones mínimas de seguridad y salud en los lugares de trabajo. https://www.boe.es/buscar/act.php?id=BOE-A-1997-8669&p=20230512&tn=1.

[14] Campos Sancho, C. y Grangel Seguer, R. y Nebot Romero, V. *Fonaments d'enginyeria del programari*. Publicacions de la Universitat Jaume I, 2017.

[15] *Chuwiki* (25 de junio de 2022). *Serialización de objetos en java*. https://chuidiang.org/index.php?title=Serializaci%C3%B3n_de_objetos_en_java. Recuperado el: 27 de mayo de 2023.

[16] *crehana* (s.f.). *¿Qué es una prueba de humo?* https://www.crehana.com/blog/transformacion-digital/prueba-de-humo/. Recuperado el: 21 de junio de 2023.

[17] Eguíluz Pérez, J. *Introducción a JavaScript*, 2012. https://biblus.us.es/bibing/proyectos/abreproy/12051/fichero/libros%252Flibro-javascript.pdf.

[18] Fielding, R. T. y Nottingham, M. y Orchard, D. y Gregorio, J. y Hadley, M. *URI Template*. *RFC* 6570, 2012.

[19] *Fonticons* (s.f.). *Font Awesome*. https://fontawesome.com/. Recuperado el: 10 de mayo de 2023.

[20] García Carballeira, F. y Menchaca Méndez, R. Arquitectura de la Máquina Virtual *Java*. *Revista Digital Universitaria*, 1(2), 2000. https://repositorio.unam.mx/contenidos/5040348.

[21] Generalitat Valenciana. (s.f.). *¿Qué es la DGTIC?* https://dgtic.gva.es/es/que-es.

[22] *GitHub, Inc.* (s.f.). *github - fasterxml/jackson: main portal page for the jackson project*. https://github.com/FasterXML/jackson. Recuperado el: 15 de junio de 2023.

[23] *Glassdor, Inc.* (9 de mayo de 2023). *Sueldos para el puesto de Jefe De Proyecto en España*. https://www.glassdoor.es/Sueldos/jefe-de-proyecto-sueldo-SRCH_KO0,16.htm. Recuperado el: 11 de mayo de 2023.

[24] *Glassdor, Inc.* (9 de mayo de 2023). *Sueldos para el puesto de Programador Júnior en España*. https://www.glassdoor.es/Sueldos/espa%C3%B1a-programador-junior-sueldo-SRCH_IL.0,6_IN219_KO7,25.htm. Recuperado el: 11 de mayo de 2023.

[25] *HubSpot, Inc* (27 de julio de 2022). *What is an API endpoint?* https://blog.hubspot.com/website/api-endpoint. Recuperado el: 29 de mayo de 2023.

[26] *Idealista* (s.f.). *Alquiler de Oficina en calle de l'Antiga Senda d'En Senent, 11.* https://www.idealista.com/inmueble/96074972/. Recuperado el: 15 de mayo de 2023.

[27] *jcodepoint* (22 de mayo de 2023). *Agregar dependencias en Apache Maven.* https://jcodepoint.com/maven/agregar-dependencias-en-apache-maven/#:~:text=La%20gesti%C3%B3n%20de%20dependencias%20es,autom%C3%A1ticamente%20a%20su%20repositorio%20local. Recuperado el: 26 de mayo de 2023.

[28] Jones, M. y Bradley, J. y Sakimura, N. *JSON Web Token (JWT). RFC* 7519, 2015.

[29] *Leroy Merlin España S.L.U* (s.f.). *Panel led Backligh de 29.5 w cuadrado.* https://www.leroymerlin.es/productos/iluminacion/iluminacion-interior/paneles-led/panel-led-backligh-de-29-5-w-cuadrado-82858852.html. Recuperado el: 16 de mayo de 2023.

[30] Lizama, O. y Kindley, G. y Morales, J. J. y Gonzales, A. . Redes de computadores: Arquitectura cliente-servidor. *Universidad Tecnica Federico Santa Maria*, 2016. http://profesores.elo.utfsm.cl/~agv/elo322/1s16/projects/reports/Proyecto%20Cliente%20-%20Servidor.pdf.

[31] Loor, M. G. A. Estructura de desglose de trabajo como herramienta para la planificación de proyectos. *Revista de Investigaciones en Energía, Medio Ambiente y Tecnología: RIEMAT*, 1(2):1–4, 2016.

[32] Molina Montero, B. y Vite Cevalos, H. y Dávila Cuesta, J. Metodologías ágiles frente a las tradicionales en el proceso de desarrollo de software. *Espirales: Revista Multidisciplinaria de Investigación*, 2(17), 2018. https://gc.scalahed.com/recursos/files/r161r/w25597w/438760423-269-823-1-PB-pdf.pdf.

[33] *OpenJS Foundation* (s.f.). *jQuery.* https://jquery.com/. Recuperado el: 25 de abril de 2023.

[34] *Orange* (s.f.). *Fibra Pro 1Gb + llamadas.* https://www.orange.es/empresas/internet-fibra-optica/fibra-pro-1gb. Recuperado el: 17 de mayo de 2023.

[35] Pantaleev, A. y Rountev, A. *Identifying Data Transfer Objects in EJB Applications. Institute of Electrical and Electronics Engineers*, pages 5–5, 2007. https://ieeexplore.ieee.org/abstract/document/4273462.

[36] *Red Hat, Inc.* (8 de mayo de 2020). *¿Qué e una API de REST?* https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[37] *Selectra* (s.f.). *Consulta el precio de la luz hoy: Detalles y Evolución de la tarifa PVPC.* https://www.leroymerlin.es/productos/iluminacion/iluminacion-interior/paneles-led/panel-led-backligh-de-29-5-w-cuadrado-82858852.html. Recuperado el: 2 de junio de 2023.

[38] *SmartBear Software* (s.f.). *Swagger UI.* https://swagger.io/tools/swagger-ui/. Recuperado el: 9 de mayo de 2023.

[39] *Spring Framework* (s.f.). *RestTemplate (Spring Framework 6.0.9 API).* https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html#exchange(java.lang.String,org.springframework.http.HttpMethod,org.springframework.http.HttpEntity,java.lang.Class,java.lang.Object...). Recuperado el: 12 de junio de 2023.

[40] *Spring Framework* (s.f.). *RestTemplateBuilder (Spring Framework 3.1.0 API).* https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/web/client/RestTemplateBuilder.html#build()). Recuperado el: 12 de junio de 2023.

[41] *SpryMedia Ltd.* (s.f.). *DataTables.* https://datatables.net/. Recuperado el: 25 de abril de 2023.

[42] *SpryMedia Ltd.* (s.f.). *Server-side processing.* https://datatables.net/. Recuperado el: 21 de junio de 2023.

[43] J. Spurlock. *Bootstrap: Responsive Web Development.* O'Reilly Media, 2013.

[44] Srinivas Sriparasa, S. *JavaScript and JSON Essentials.* Packt Publishing Ltd., 2013.

[45] *Talent* (s.f.). *Salario para analista programador en España, 2023.* https://es.talent.com/salary?job=analista+programador#:~:text=El%20salario%20analista%20programador%20promedio,hasta%20%E2%82%AC%2040.000%20al%20a%C3%B1o. Recuperado el: 5 de mayo de 2023.

[46] *The Apache Software Foundation* (s.f.). *APACHE POI.* https://poi.apache.org/. Recuperado el: 29 de abril de 2023.

[47] *The Apache Software Foundation* (s.f.). *POM Reference.* https://maven.apache.org/pom.html#Aggregation_.28or_Multi-Module.29. Recuperado el: 26 de mayo de 2023.

[48] *The Apache Software Foundation* (s.f.). *What is Maven?* https://maven.apache.org/what-is-maven.html. Recuperado el: 26 de mayo de 2023.

[49] *the Mozilla Foundation* (s.f.). *CSS.* https://developer.mozilla.org/es/docs/Web/CSS. Recuperado el: 22 de abril de 2023.

[50] *the Mozilla Foundation* (s.f.). *Generalidades del protocolo HTTP.* https://developer.mozilla.org/es/docs/Web/HTTP/Overview. Recuperado el: 29 de mayo de 2023.

[51] *the Mozilla Foundation* (s.f.). *html.* https://developer.mozilla.org/es/docs/Web/html. Recuperado el: 22 de abril de 2023.

[52] *the Mozilla Foundation* (s.f.). *HTTP Headers.* https://developer.mozilla.org/es/docs/Web/HTTP/Headers. Recuperado el: 15 de junio de 2023.

[53] *the Mozilla Foundation* (s.f.). *select - HTML: Lenguaje de etiquetas de hipertexto.* https://developer.mozilla.org/es/docs/Web/HTML/Element/select. Recuperado el: 9 de mayo de 2023.

[54] *the Mozilla Foundation* (s.f.). *tabla - HTML: Lenguaje de etiquetas de hipertexto.* https://developer.mozilla.org/es/docs/Web/HTML/Element/table. Recuperado el: 9 de mayo de 2023.

[55] *The Thymeleaf Team.* (s.f.). *Tutorial: Using Thymeleaf.* https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#what-is-thymeleaf.

[56] *The Tortoise Team* (s.f.). *TortoiseSVN.* https://tortoisesvn.net/about.html. Recuperado el: 10 de mayo de 2023.

[57] *Tiwari V.* (s.f.). *Spring Tool Suite.* https://www.codingninjas.com/codestudio/library/spring-tool-suite. Recuperado el: 9 de mayo de 2023.

[58] *VMware, Inc.* (s.f.). *Spring Framework.* https://spring.io/projects/spring-framework. Recuperado el: 22 de abril de 2023.

[59] Xie, Y. (28 de junio de 2017). *Using Selectize Input.* https://shiny.rstudio.com/articles/selectize.html.

Anexo A

Diagrama de clases DTO

En este anexo se muestran las figuras con todos los diagrama de clases de los *DTO* utilizados en el proyecto. En la figura A.1 se muestra el diagrama del *DTO BaseDTO*, del cual heredan todos.

Por otro lado, de la figura A.2 a la A.5 aparecen los diagramas de clases de los *DTO* de entrada.

Los *DTO REST*, que pertenecen a los *DTO* de salida aparecen desde la figura A.6 a la A.25.

Los *DTO MV*, que también se incluyen en los *DTO* de salida se muestran desde la figura A.26 a la A.29.

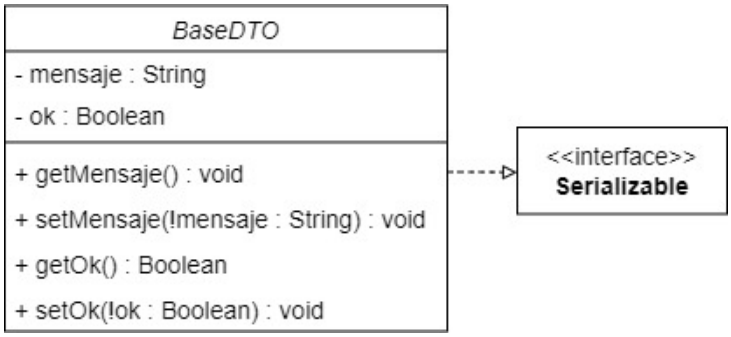


Figura A.1: Diagrama de clases del *DTO BaseDTO*.

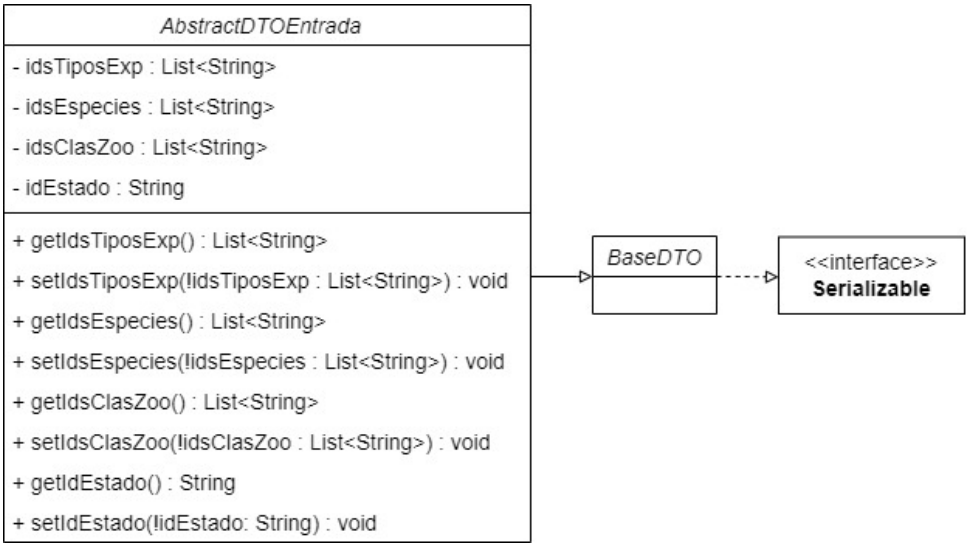


Figura A.2: Diagrama de clases del *DTO AbstractDTOEntrada*.

Figura A.3: Diagrama de clases del *DTO* de entrada *DTOEntradaTitulares*.

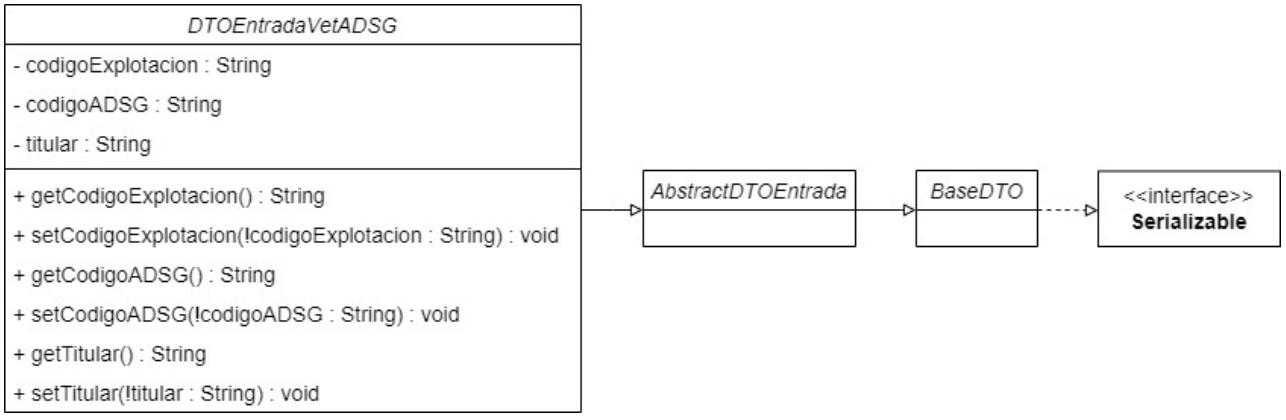


Figura A.4: Diagrama de clases del *DTO* de entrada *DTOEntradaVetADSG*.

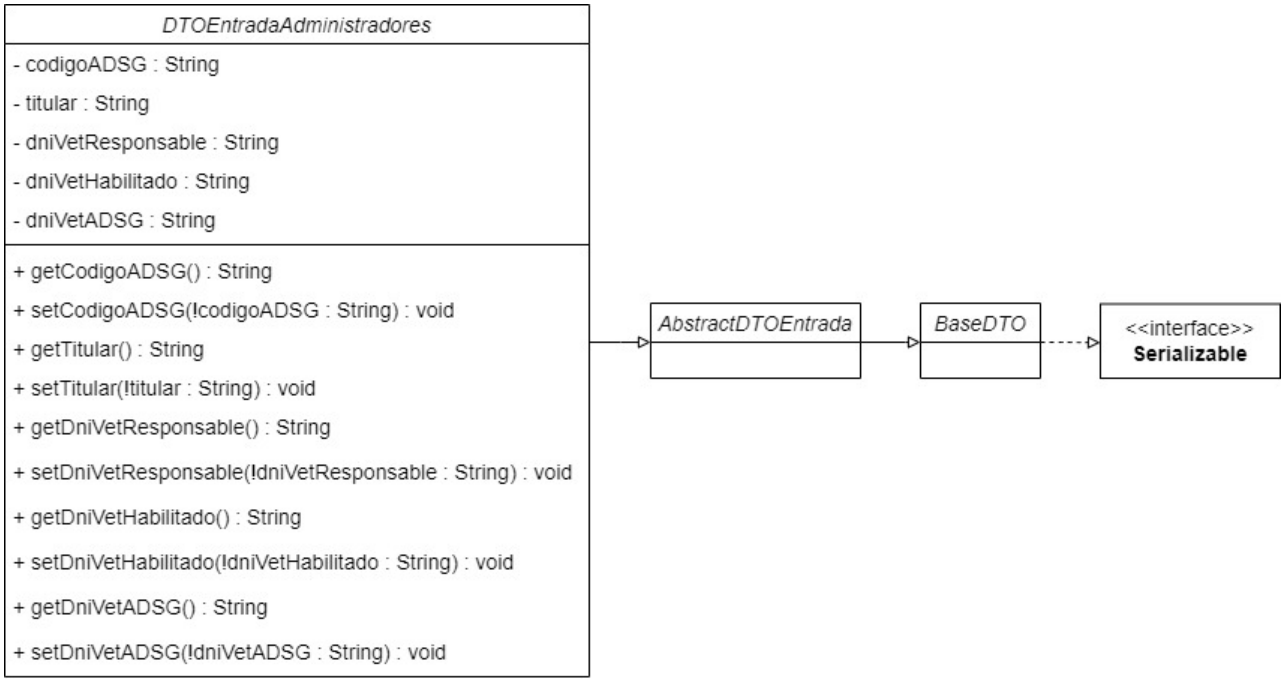


Figura A.5: Diagrama de clases del *DTO* de entrada *DTOEntradaAdministradores*.

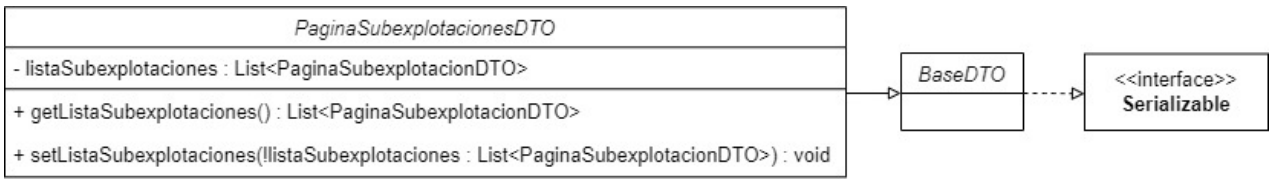


Figura A.6: Diagrama de clases del *DTO REST PaginaSubexplotacionesDTO*.

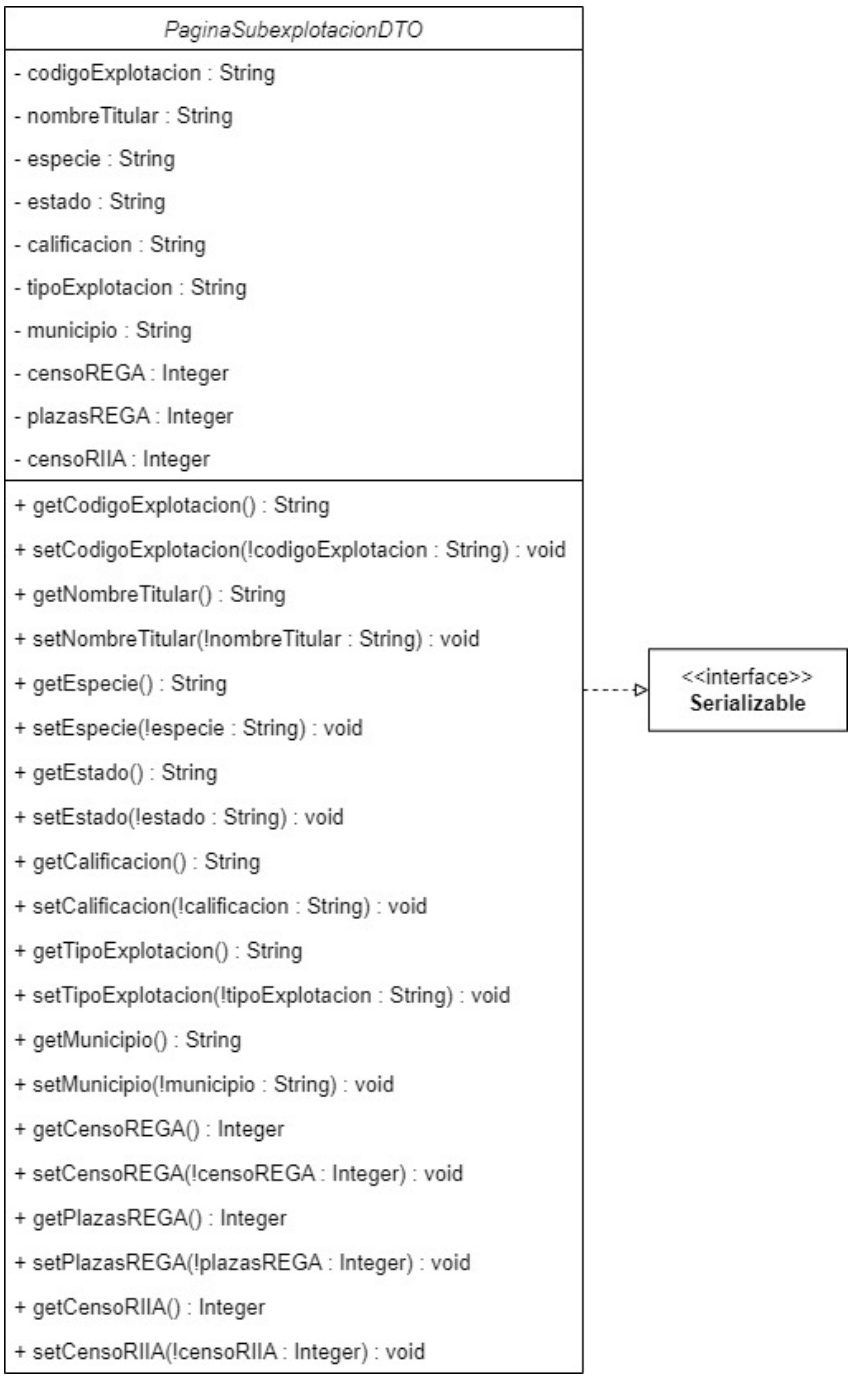


Figura A.7: Diagrama de clases del *DTO REST PaginaSubexplotacionDTO*.

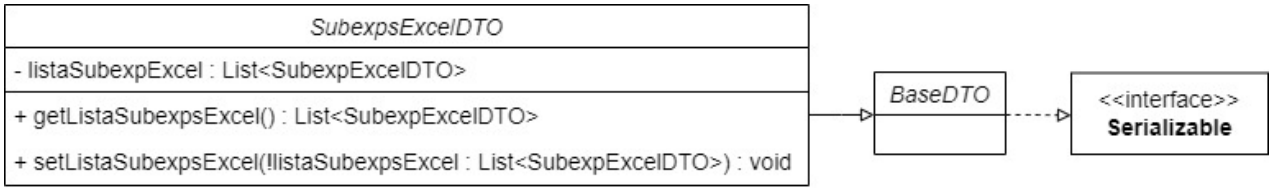


Figura A.8: Diagrama de clases del *DTO REST SubexpsExcelDTO*.

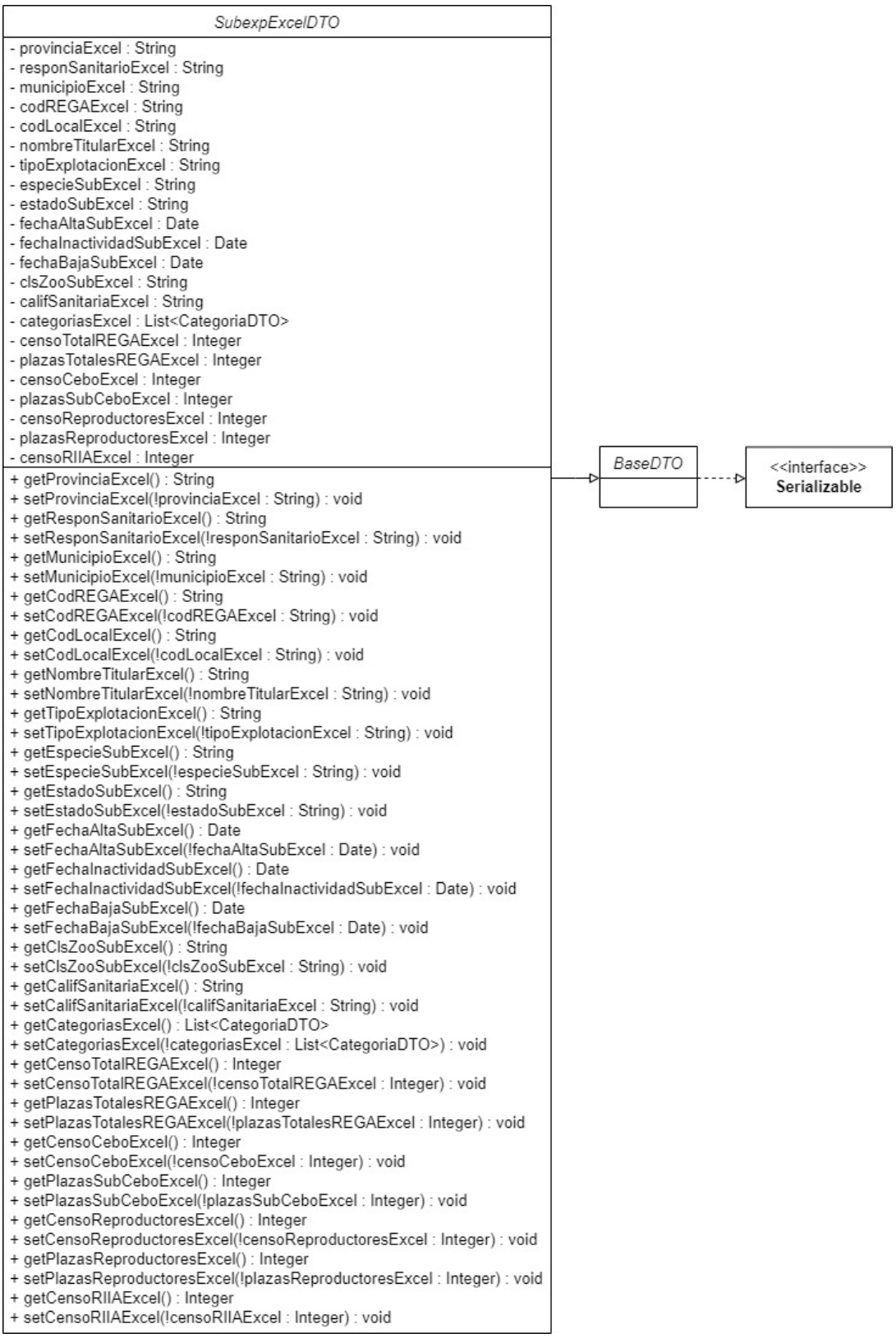


Figura A.9: Diagrama de clases del *DTO REST SubexpExcelDTO*.

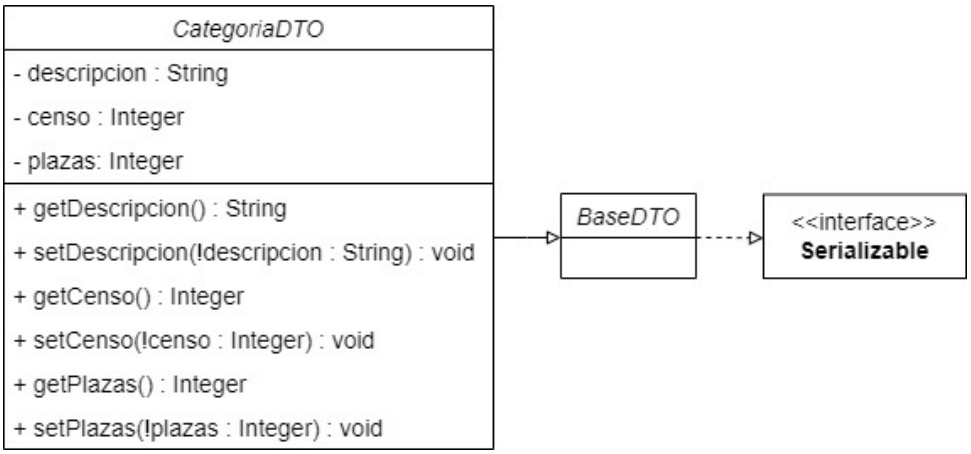


Figura A.10: Diagrama de clases del *DTO REST CategoriaDTO*.

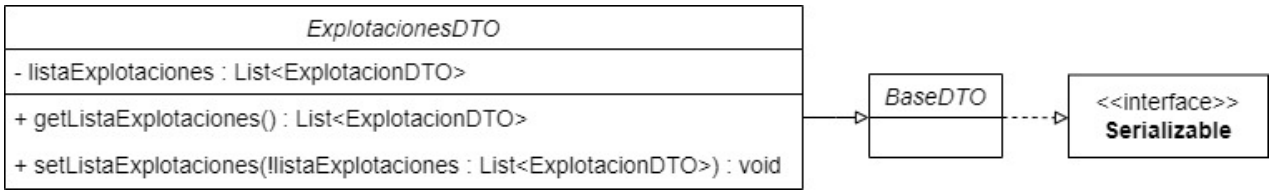


Figura A.11: Diagrama de clases del *DTO REST ExplotacionesDTO*.

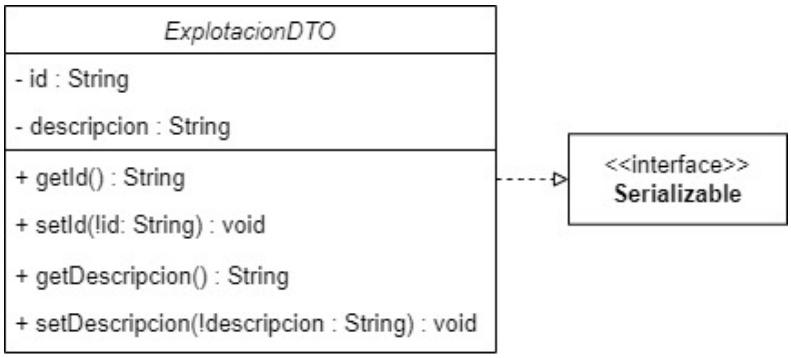


Figura A.12: Diagrama de clases del *DTO REST ExplotacionDTO*.

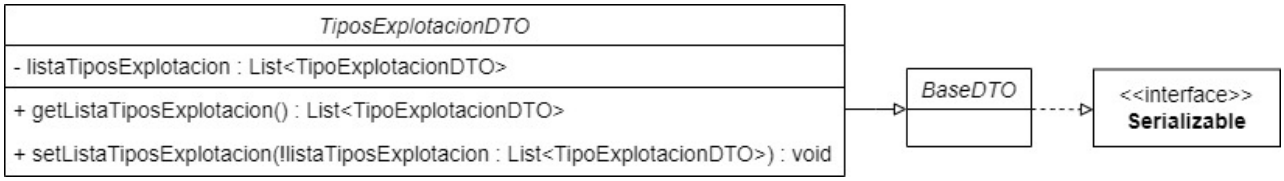


Figura A.13: Diagrama de clases del *DTO REST TiposExplotacionDTO*.

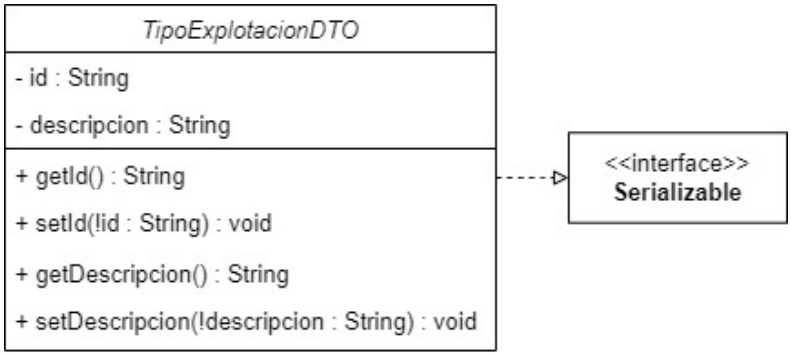


Figura A.14: Diagrama de clases del *DTO REST TipoExplotacionDTO*.

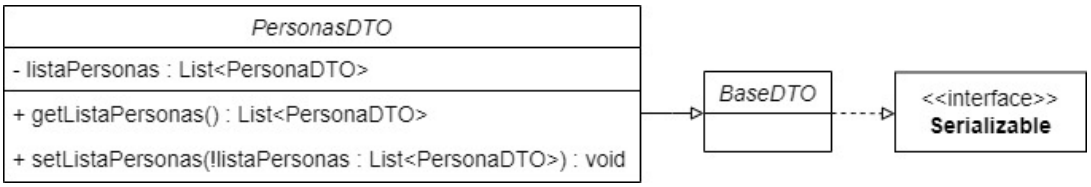


Figura A.15: Diagrama de clases del *DTO REST PersonasDTO*.

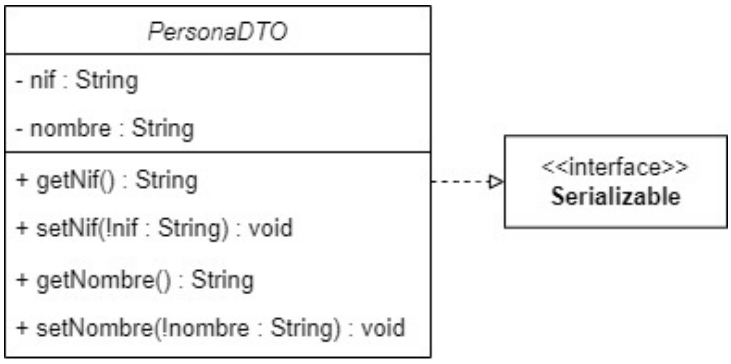


Figura A.16: Diagrama de clases del *DTO REST PersonaDTO*.

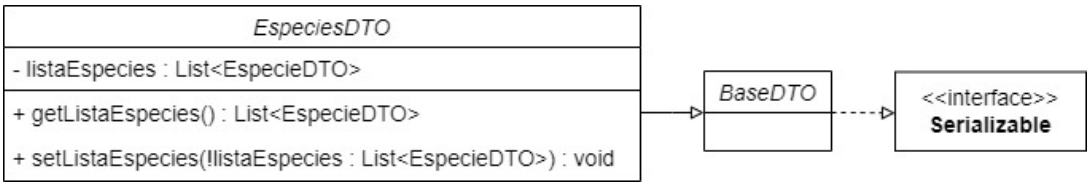


Figura A.17: Diagrama de clases del *DTO REST EspeciesDTO*.

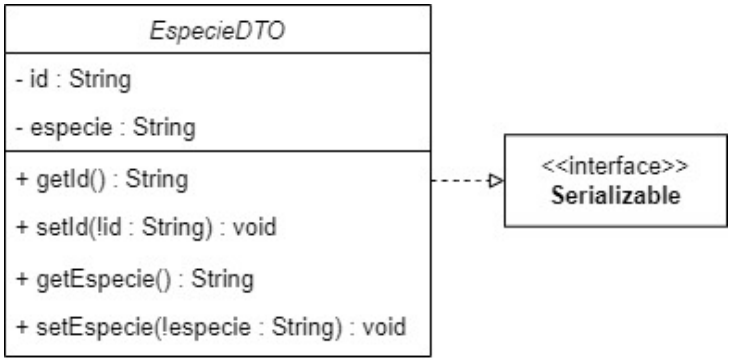


Figura A.18: Diagrama de clases del *DTO REST EspecieDTO*.

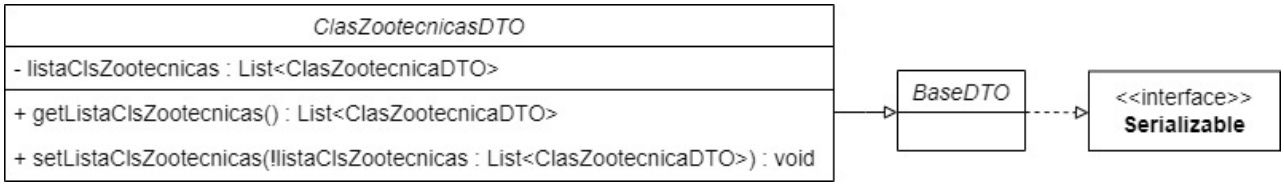


Figura A.19: Diagrama de clases del *DTO REST ClasZootecnicasDTO*.

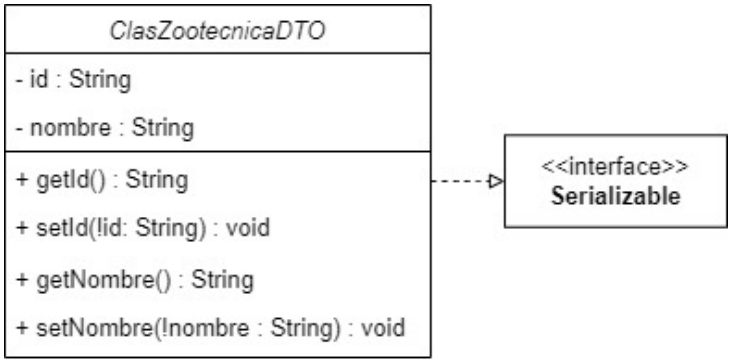


Figura A.20: Diagrama de clases del *DTO REST ClasZootecnicaDTO*.

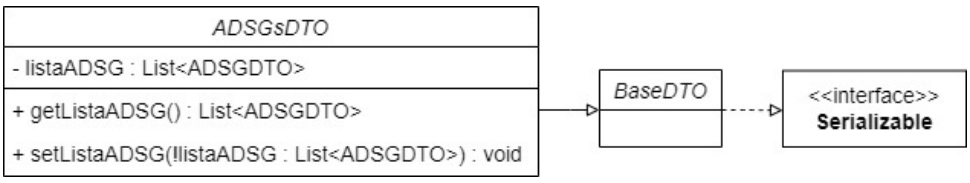


Figura A.21: Diagrama de clases del *DTO REST ADSGsDTO*.

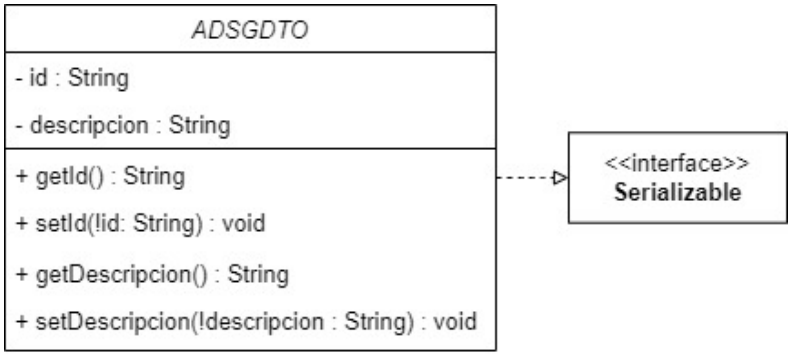


Figura A.22: Diagrama de clases del *DTO REST ADSGDTO*.

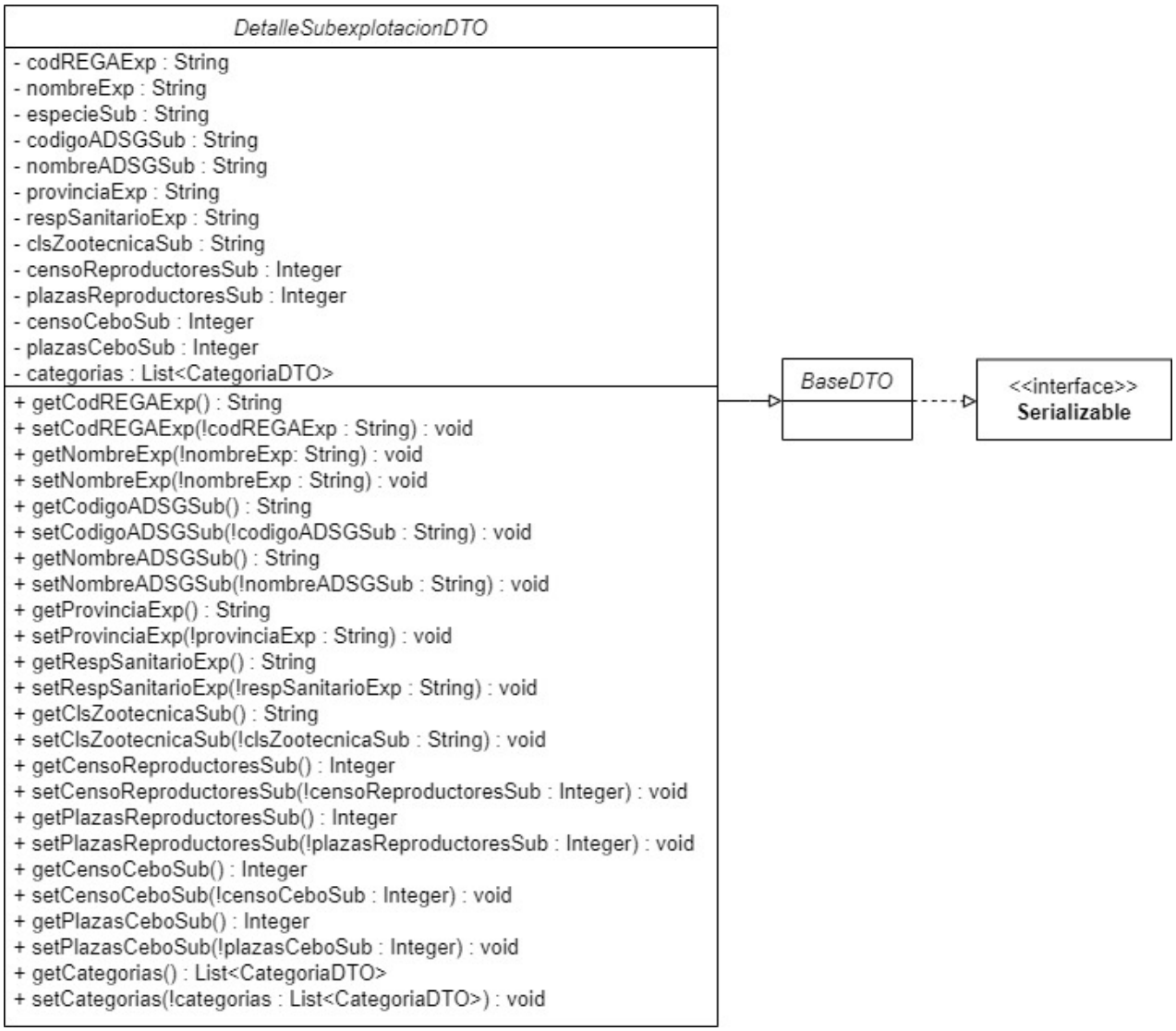


Figura A.23: Diagrama de clases del *DTO REST DetalleSubexplotacionDTO*.

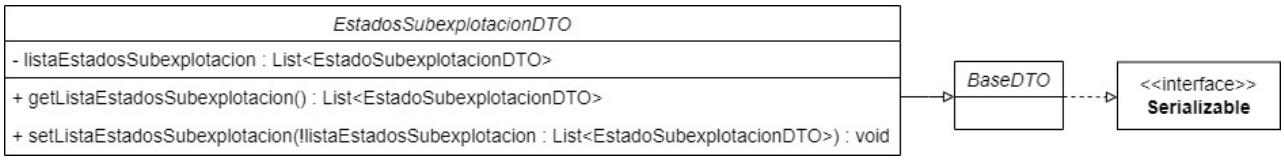


Figura A.24: Diagrama de clases del *DTO REST EstadosSubexplotacionDTO*.

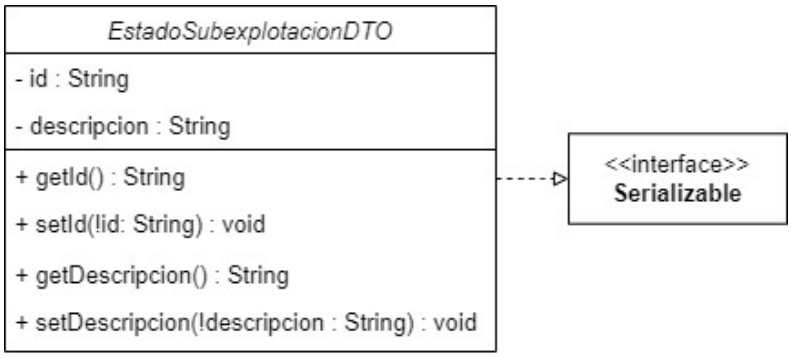


Figura A.25: Diagrama de clases del *DTO REST EstadoSubexplotacionDTO*.

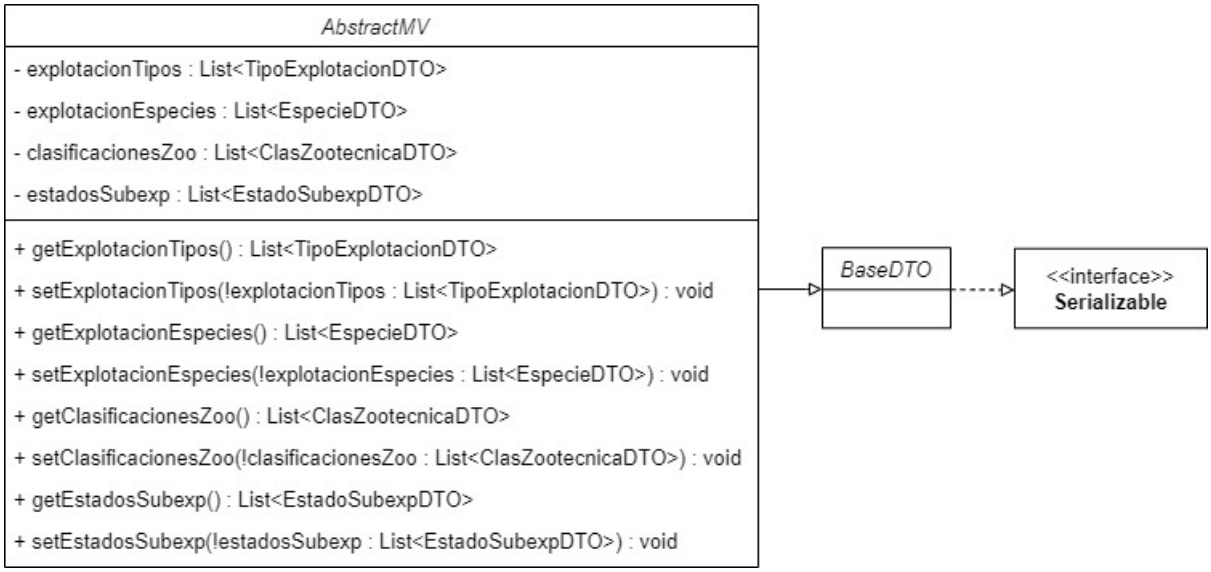


Figura A.26: Diagrama de clases del *DTO MV AbstractMV*.

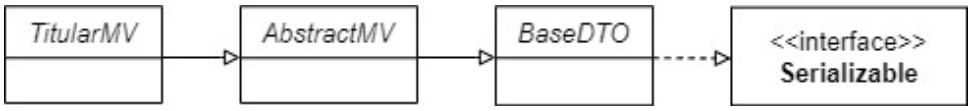


Figura A.27: Diagrama de clases del *DTO MV TitularMV*.

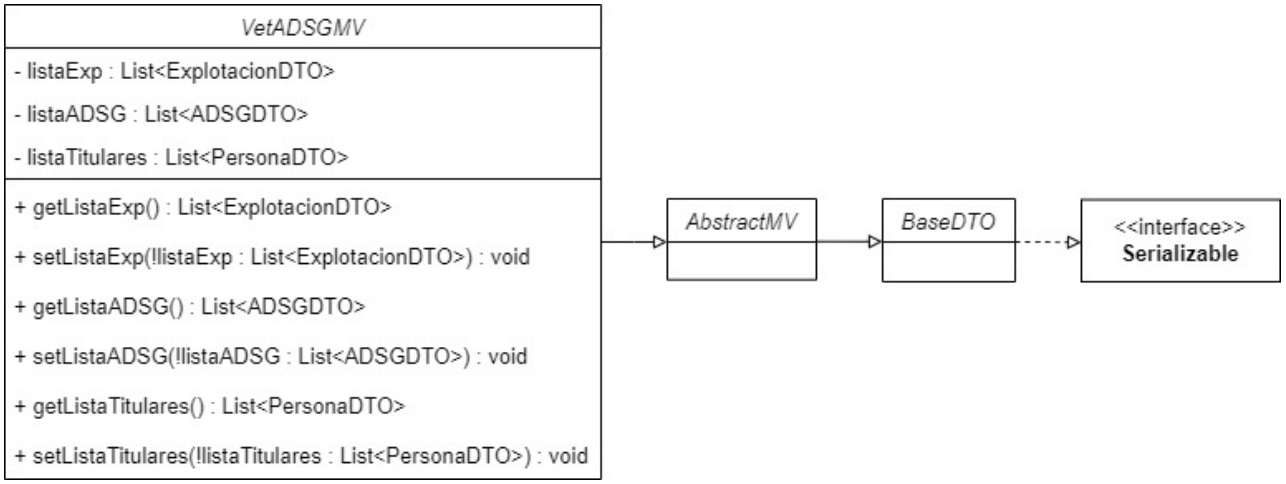


Figura A.28: Diagrama de clases del *DTO MV VetADSGMV*.

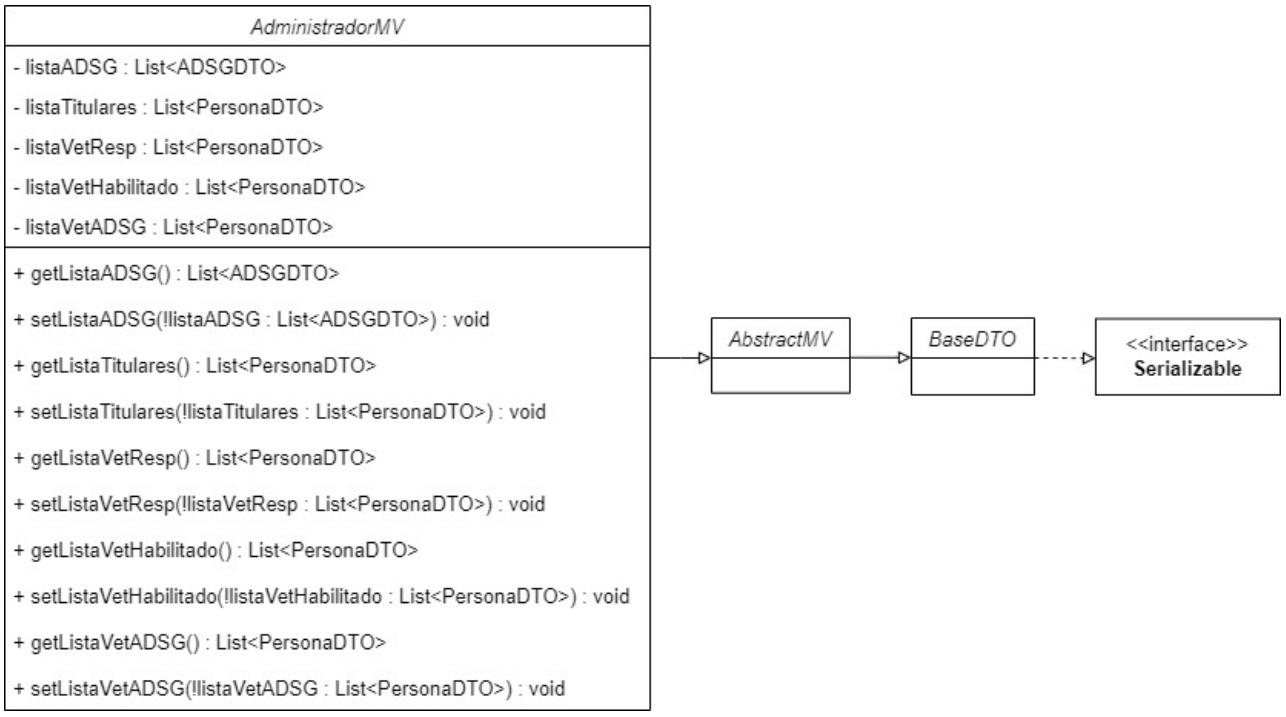


Figura A.29: Diagrama de clases del *DTO MV AdministradorMV*.

Anexo B

Diagrama de clases de la implementación

En este anexo se muestran los diagrama de clases de los siguientes módulos de la implementación:

- *client*: de la figura B.1 a la B.2.
- *service-api*: de la figura B.3 a la B.4.
- *service-impl*: de la figura B.5 a la B.6.
- *application*: la figura B.7.

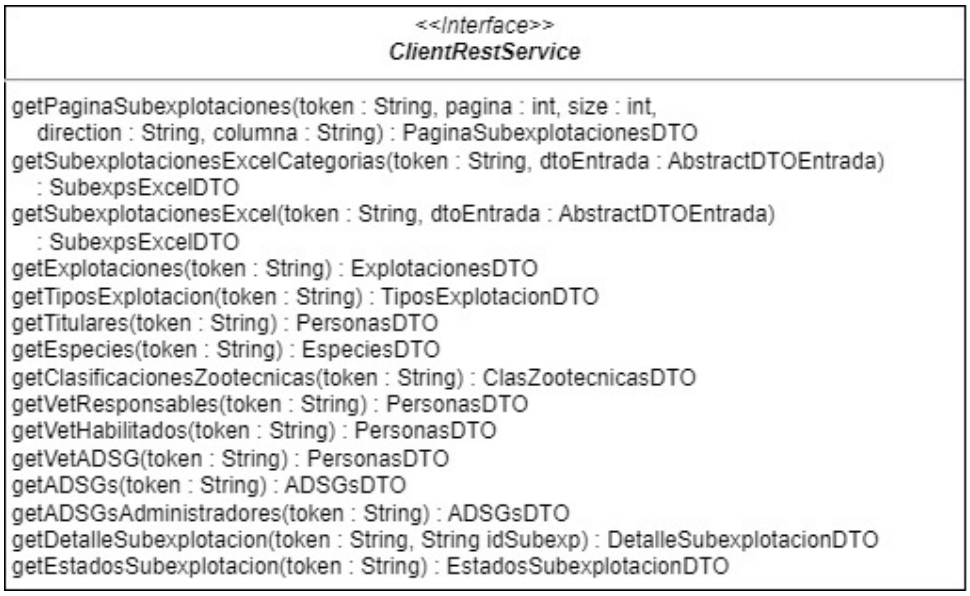


Figura B.1: Diagrama de clases de la interfaz *ClientRestService*.

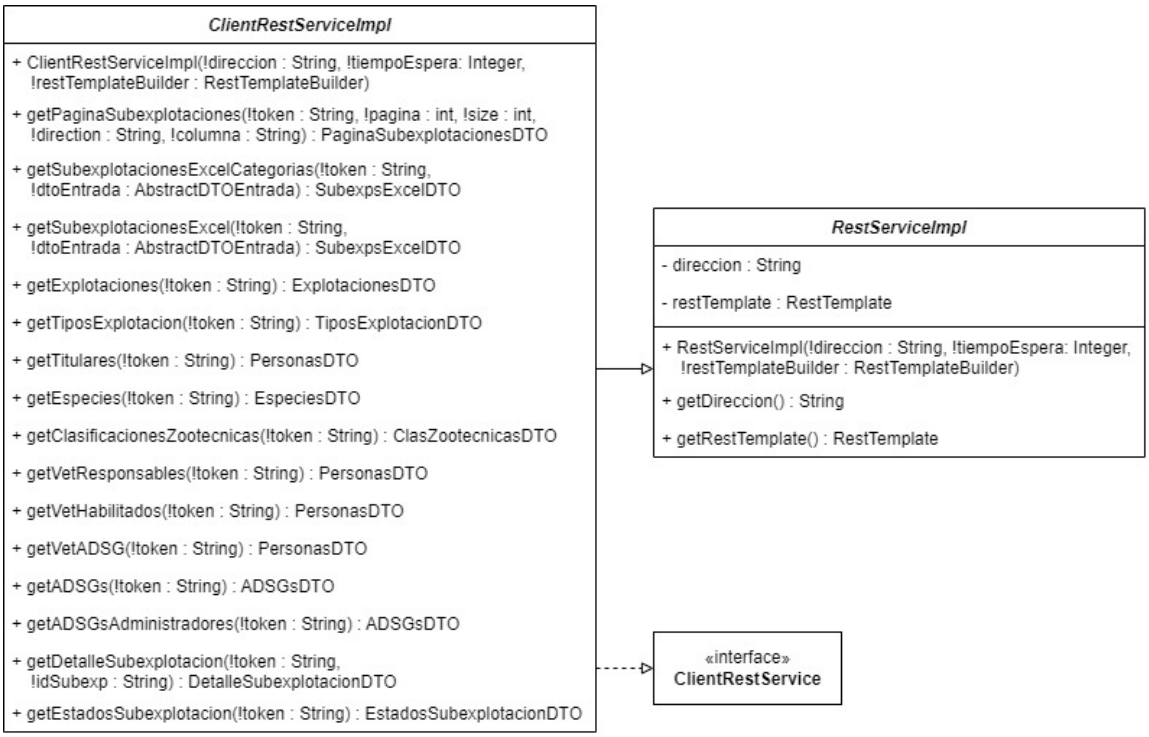


Figura B.2: Diagrama de clases de la clase *ClientRestServiceImpl*.

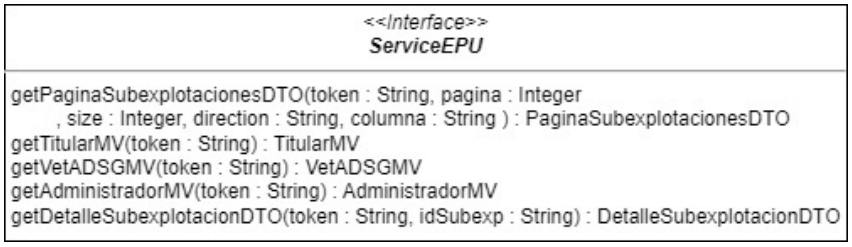


Figura B.3: Diagrama de clases de la interfaz *ServiceEPU*.

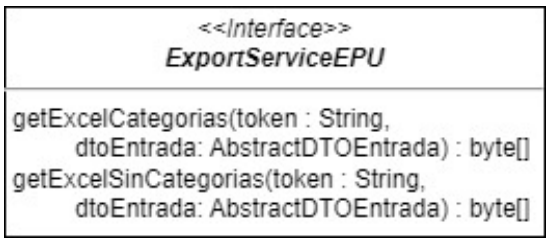


Figura B.4: Diagrama de clases de la interfaz *ExportServiceEPU*.

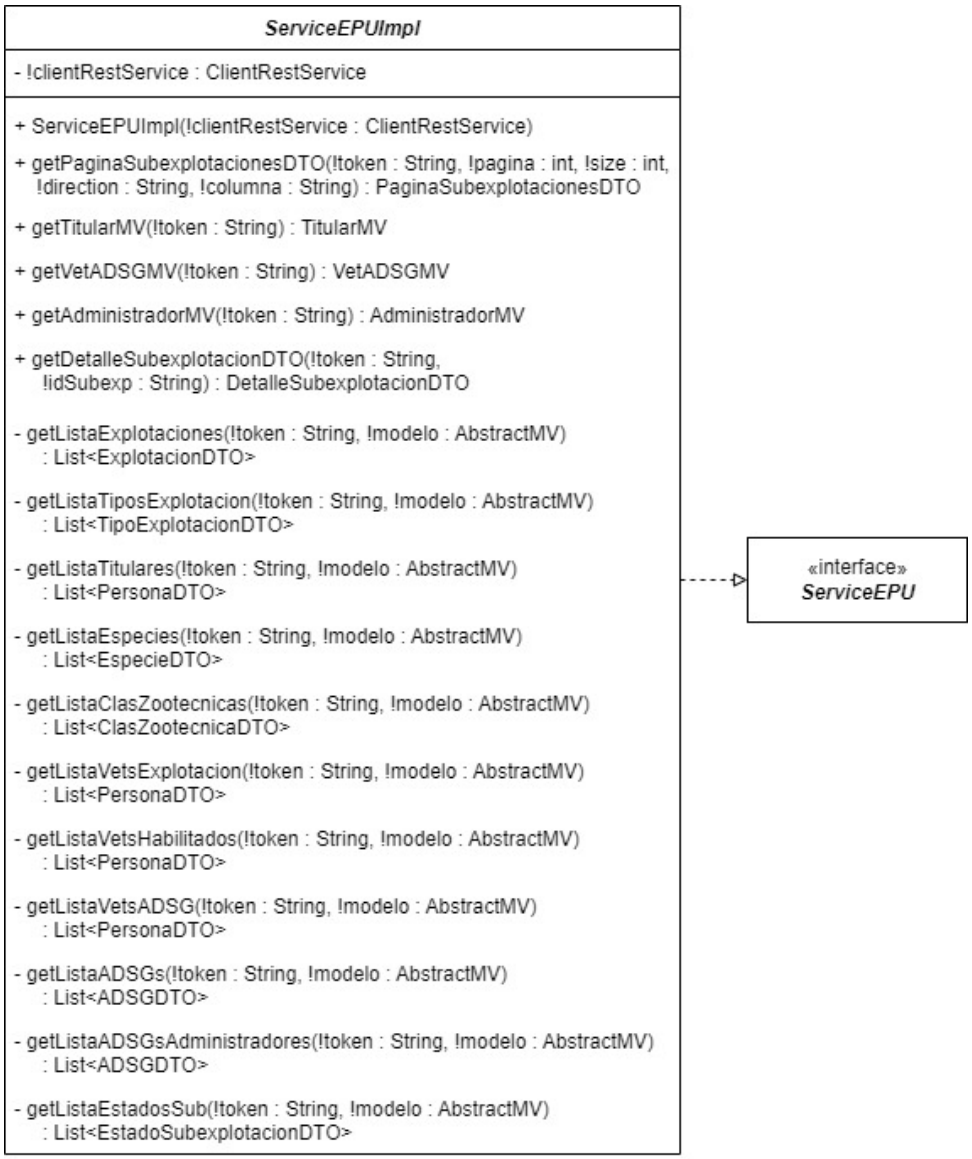


Figura B.5: Diagrama de clases de la clase *ServiceEPUImpl*.

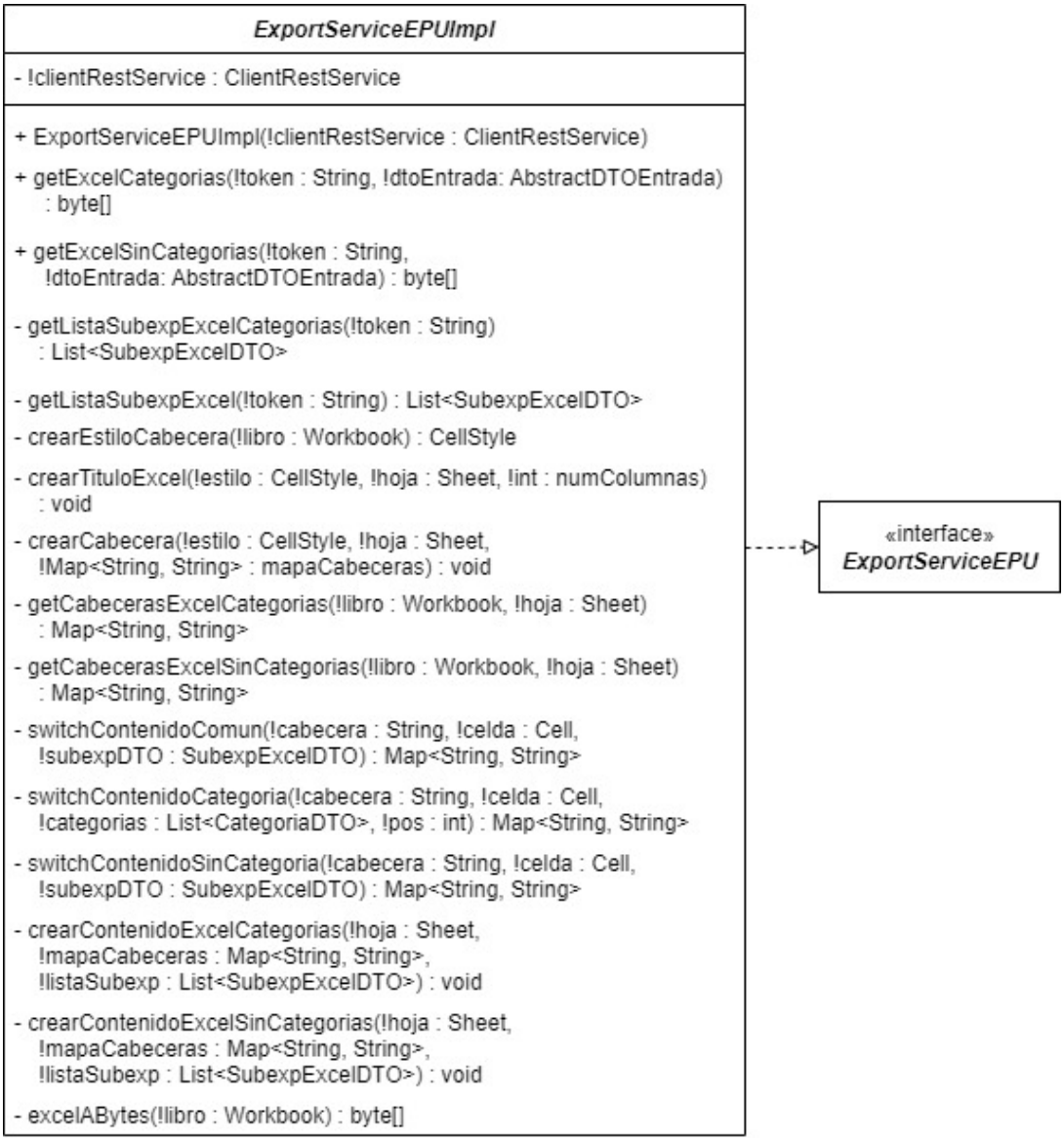


Figura B.6: Diagrama de clases de la clase *ExportServiceEPUImpl*.

Figura B.7: Diagrama de clases de la clase *ControllerEPU*.