ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

PLATAFORMA PARA LA PREVENCIÓN DE DELITOS

Procesamiento de datos y lógica de negocios

CRIME PREVENTION PLATFORM

SUBTITULO EN INGLES

Data processing and business logic

Realizado por
JUAN PALMA BORDA

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2019

Fecha defensa: de septiembre de 2019

Fdo. El/la Secretario/a del Tribunal

Resumen

Se ha desarrollado un proyecto colaborativo con la participación de la Policía Nacional de Málaga. El objetivo del proyecto es la elaboración de una plataforma web para la visualización de datos y la ejecución de modelos de prevención de delitos generados con datos privados de la policía de Málaga.

La parte del proyecto relativa a este trabajo consiste en la importación, estructuración y exportación de los datos de la policía después de haberlos procesado y limpiado. Para ello se hace uso del lenguaje Python 3 y de la base de datos no relacional MongoDB para el almacenamiento de los datos.

Palabras clave: Prevención de crímenes, ingeniería de datos, limpieza de datos, procesamiento de datos.

Abstract

This collaborative project has been carried out with the National Police Corps of Malaga. The objective of this project is creating a web platform for data visualization and for executing crime prevention models generated by private data of Malaga police.

The part of the project relative to this work is the importation, structuration and exportation of the police data after they have been processed and cleaned. The web platform has been made with Python 3 as the main programming language and MongoDB has been used for storing the data.

Keywords: **Crime prevention, data engineering, data cleaning, data processing.**

Índice

1

Introducción

1.1 Motivación

Las actuaciones policiales frente a delitos de cualquier tipología se deben realizar de una manera rápida y efectiva, sabiendo distribuir los recursos policiales que minimicen el número de delitos o maximicen el número de detenciones de las personas que comenten dichos actos delictivos.

Para conseguir esto la informática puede jugar un papel clave gracias a herramientas de recolección, limpieza, procesamiento, consulta y visualización de datos o estadísticas de estos mismos.

1.2 Objetivos

Por ello el objetivo de este Trabajo de Fin de Grado (TFG) es crear una herramienta web que permita realizar las tareas mencionadas anteriormente con datos de la Policía Nacional de Málaga y que, por ende, permita el análisis y prevención de delitos tanto para la realización de informes como para la optimización del reparto de patrullas en el futuro.

Este trabajo ha sido realizado por un grupo de tres estudiantes por lo que la herramienta web a crear se ha dividido en diferentes partes, cada una de las cuales tiene distintos requisitos y realizados por distintas personas, estas partes son:

- Un módulo para poder visualizar los datos de manera gráfica y así contribuir a una mejor interpretación de éstos.
- Unos servicios para poder transmitir los datos de una manera segura y rápida a la interfaz.
- Un módulo de estandarización y jerarquización de datos en los que se representen de manera clara los detalles de los mismos, tales como las horas de los delitos, dónde sucedieron los hechos, la tipología del hecho, etc.
- Una herramienta o servicio que se encargue de procesar, limpiar y estructurar estos datos.
- Un simulador que permita predecir futuros comportamientos en base a los datos de los que dispone la herramienta.

1.3 Antecedentes

Intentar predecir dónde, con qué frecuencia o quién va a cometer un delito es una acción que se lleva a cabo en todos los cuerpos de seguridad para intentar limitar posibles daños, reducir los delitos o aumentar la eficacia de sus integrantes.

Este campo de la informática es un gran atractivo para multitud de gobiernos, cuerpos de seguridad o empresas y poco a poco se van implementado prototipos y primeras versiones de estos algoritmos todos los años.

Es el caso de *X-law* un algoritmo diseñado en Nápoles por el inspector policía *Elia Lombardo*. Este algoritmo realiza a un estudio de los datos y aplica

modelos criminales de forma que es capaz de dar cada poco tiempo la zona más probable dónde se va a dar un tipo de delito, así como que características van a tener el agresor o la víctima. [1][2]

Otro ejemplo es el trabajo *Investigación y análisis de crímenes en las ciudades de Chicago y Los Ángeles* realizado por *Adrián García Humanes* en esta misma universidad en donde se lleva a cabo un análisis de los *datasets* públicos de los crímenes en Chicago y en Los Ángeles. Para ello se lleva a cabo una limpieza, clasificación y filtrado de estos mismos datos. Para poder después realizar un análisis de los mismos. [3]

Una de las características que diferencia este trabajo de otros en el ámbito educativo, es que en este no se parte de unos *datasets* fijos e inmutables, sino que salvo por unos mínimos, pueden variar enormemente, tanto en número de campos como en el tamaño de los archivos. Esto es debido a que son *datasets* con datos actuales que son generados expresamente por la policía nacional de Málaga.

1.4 Metodología

Para poder llevar a cabo todas estas tareas se ha aplicado una metodología ágil basada en la forma de trabajar y en los métodos de *Scrum*.

Una metodología no es más que un conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal, según la definición extraída de la Real Academia Española (*versión web, 2019*). En la informática, las metodologías tradicionales son demasiado estructuradas y requieren de unos requisitos de los proyectos fijos y con poca variación. Por ello en los últimos años se han ido imponiendo las metodologías ágiles como mejora de las tradicionales en numerosos proyectos informáticos.

Estas mejoras consisten en flexibilizar los requisitos de la aplicación de forma que permita cambios más drásticos, sin afectar tanto al proyecto e integrando de una manera más activa al cliente del proyecto ya sea en forma de continuas reuniones o de jornadas de trabajo en las que el propio cliente pueda estar presente.

La amplia mayoría de las metodologías ágiles para el desarrollo de software dividen el trabajo en iteraciones, contando cada una con un número concreto de fases. Estas suelen ser:

- Una planificación detallada de las tareas a realizar durante la iteración.
- El análisis de requisitos.
- El diseño de las nuevas partes del sistema.
- La implementación de dichas partes.
- Las pruebas.
- La elaboración de la documentación.

El objetivo de esta estructuración es conseguir un mínimo producto viable en cada iteración.

Se escogió la metodología *Scrum* debido a que gran parte de sus características, entre las que destacan el uso de una *Taskboard* o pizarra para planificar cada iteración o *sprint* (como se denomina cada iteración en *Scrum*), la revisión de los mismos de forma retrospectiva y las reuniones diarias entre los distintos integrantes del grupo. En la adaptación que se ha escogido de Scrum no se incluyen los distintos roles de los integrantes del proyecto, debido al número reducido de integrantes.

Finalmente se definieron un total de 4 iteraciones iniciales, cuyos requisitos fueron cambiando con el avance del proyecto, aunque la idea básica de las mismas se mantuvo a lo largo del proyecto.

1.5 Tipología del proyecto

Este proyecto ha sido realizado en grupo con partes distintas salvo pequeñas excepciones. Las partes diferenciadas han sido: la interfaz gráfica a cargo de *Alberto Ramírez Mena*; la generación de modelos con agentes para la prevención de delitos y la seguridad del servidor, realizados por *Sergio Gavilán Prieto*; y la gestión de los datos, que engloba la importación, limpieza y exportación de los datos, que han sido los objetivos de este trabajo.

Por otro lado, la realización de las consultas a la base de datos se ha realizado de manera conjunta, las primeras versiones y planteamiento inicial fue realizada por parte del compañero *Sergio Gavilán Prieto* y las versiones finales y las mejoras de eficiencia fueron parte de este TFG.

Adicionalmente y, como parte de este TFG, se realizaron diversas reuniones con el Cuerpo Nacional de Policía de Málaga, que han desempeñado el rol de clientes. Esto ha supuesto un reto adicional y un aumento de dificultad en el trabajo, pues no se parten con unos requisitos fijos de antemano sino con una idea general de lo que se pretende hacer y en cada reunión se van añadiendo o eliminados requisitos del proyecto.

Estas reuniones se hacían aproximadamente cada dos meses y en ellas se planteaban posibles mejoras, nuevos requisitos, se rechazaban tareas por inviabilidad y se estructuraba la ejecución de que tareas se realizarían con más prioridad.

1.6 Tecnologías y herramientas empleadas

En la parte del proyecto correspondiente a este TFG, se ha usado *Python 3* como lenguaje programación principal, debido principalmente a la simpleza del usó y a las facilidades que tiene a la hora de manejar grandes cantidades de datos no estandarizados.

Para la base de datos principal se ha usado *MongoDB*, una base de datos no relacional que se orienta a documentos, llamados *BSON* (*Binary JSON*). Al ser una base de datos no relacional evita la obligatoriedad de tener totalmente estructurada las entidades de la base de datos, esto se adapta estupendamente a datos que pueden tener distritos atributos, aun representando fundamentalmente lo mismo.

Hay que destacar que un *JSON* se convierte automáticamente a un diccionario de *Python* y de *BSON* a este formato la conversión es casi inmediata. También se puede llevar a cabo lo contrario, es decir, pasar de un diccionario a un documento *JSON* lo que facilita en gran medida la importación de los datos.

Dentro del propio importador se ha usado *SQLite* como base de datos auxiliar. En esta base de datos se han guardado la información necesaria para realizar ciertas partes de la importación de los datos. *SQLite* es una base de datos muy sencilla orientada a no muchos datos y que es relacional, por lo que se usan consultas *SQL* para manejarlas.

Hay otras tecnologías que se han usado pero que no son especialmente relevantes, estas son: *JavaScript*, en su versión de *AngularJS*, este lenguaje más que usarse se ha interpretado parcialmente para realizar algunas mejoras en los servicios *REST*. También se ha utilizado parcialmente *ANTLR4,* una herramienta para la elaboración y uso de gramáticas, para la elaboración de determinadas funcionalidades con los datos.

Para realizar el servidor *REST* se ha usado *Flask*. *Flask* es un framework orientado a microservicios que permite tener muy rápido y de manera sencilla unos servicios *REST* funcionado.

Como *IDEs* se han usado principalmente dos: *Pycharm*, el *IDE* para *Python* de *JetBrains* y, *Visual Studio Code*, un editor de código libre realizado por *Windows*.

Pycharm se ha usado para el proyecto en sí gracias a la gran cantidad de mejoras con *plugins* que obtiene y que permite una compatibilidad rápida con bases de datos o con otras librerías o lenguajes.

Visual Studio Code se ha usado para la interpretación de los datos en *CSV* ya que permite ver su codificación de manera muy rápida y maneja bastante bien grandes cantidades de datos, también permite una edición de los JSON bastante cómoda.

Por último, para la realización de modelos iniciales se han usado *MagicDraw*, con su correspondiente lenguaje *UML* y *yEd Graph Editor* que se ha usado para la elaboración de diagramas más simples.

Mencionar por último que todas las librerías usadas en el proyecto tienen una licencia MIT o alguna licencia muy parecida que permite su modificación y uso de manera gratuita y que no contienen copyleft.

1.7 Estructura de la memoria

La memoria se estructurará siguiendo la ejecución de las iteraciones del proyecto, en cada una de las iteraciones se explicarán de forma resumida sus objetivos y se procederá posteriormente a explicar el desarrollo de cada uno de ellos.

Es importante aclarar destacar que la “Iteración 0” corresponde a una iteración introductoria del proyecto en la que se llevaron a cabo la mayor parte de las investigaciones pertinentes y se plantearon los primeros requisitos del proyecto.

En la “Iteración 1” desde el punto de vista del *Importador*, se trabajó con un sistema de importación independiente del servidor y se centró en la manera de procesar los datos de la policía.

Por otro lado, en la “Iteración 2”, se integró el sistema de importación de datos dentro del servidor y se crearon los algoritmos para generar las festividades y distritos municipales a partir de los datos proporcionados por la policía. También se terminó un módulo auxiliar que permitía transformar rápidamente datos públicos proporcionados en CSV a la base de datos interna del propio *Importador*.

En la última iteración del proyecto, la “Iteración 3”, se corrigieron y minimizaron los problemas de memoria del sistema de importación y se permitió la exportación a CSV de estadísticas y de entidades de la base de datos, así como se facilitó una forma de eliminar todos los datos del sistema.

En el último gran apartado de la memoria se explican las conclusiones del proyecto y posibles mejoras o cambios que se pueden llevar en un futuro para seguir desarrollando la aplicación de la policía.

2

Iteración 0

2.1 Introducción

En esta primera iteración se recopilaron los primeros requisitos del proyecto, así como se llevaron a cabo las tareas de modelado del sistema, la investigación de las tecnologías que se iban a usar en el proyecto y el análisis e interpretación de los primeros *datasets* proporcionados por la policía.

2.2 Desarrollo

El desarrollo de la Iteración 0 estará divido en 4 secciones: la toma de los requisitos del sistema, la investigación de las tecnologías, el análisis y la definición de los datos y el modelado del sistema.

2.2.1 Toma de requisitos del sistema

La toma de requisitos inicial del sistema fue una tarea que no se llevó a cabo de manera totalmente definitiva, ya que al ser un proyecto basado en iteraciones y del que se espera que se continúe en un futuro había bastante incertidumbre en las necesidades de la policía. Por ello en esta iteración se

tomaron las ideas bases del proyecto, sobre las se iban a tomar el resto de los requisitos.

En esta primera toma de requisitos, se diferenciaron muchos tipos de formas de realizar las búsquedas sobre los datos de los *datasets*, muchos de los cuales se rechazaron o terminaron postergados en las distintas iteraciones. Asimismo, se tomaron los requisitos base de visualización del entorno a representar, la importación de datos con formato parcialmente variable desde *CSV* y la creación y simulación de modelos basados en agentes.

2.2.2 Investigación de las tecnologías

Las tecnologías a investigar variaron para cada uno de los integrantes del grupo. Para el *Frontend* se decidió usar *D3* para la visualización gráfica y *AngularJS* para los controladores; mientras que para el *Backend* se optó por *Flask* que es un *framework* de *Python* orientado a microservicios de forma que se pudieran aprovechar todo el servidor si se quisiera realizar una versión móvil de la aplicación en un futuro.

Para la base de datos se optó por *MongoDB* ya que los datos no iban a tener los mismos campos, es decir, iban a tener diferentes atributos no estandarizados. Por último, para los agentes se llevó a cabo una investigación de distintos *frameworks* de simulación, entre los que destacaron *MESA*, *PADE*, *SPADE*, del que al final se escogió *MESA*.

La investigación relativa a la parte del proyecto relevante para el preprocesamiento de datos consistió en la lectura del manual de *MongoDB*, -cuyo enlace web se encuentra en la bibliografía del trabajo-; la creación de una conexión entre *Python* y esta base de datos; la investigación de librerías de lectura de *CSV* en *Python*; el estudio de buenas prácticas de programación en *Python* y, por último, a la lectura de un resumen de la utilización de expresiones regulares y gramáticas.

En el manual de *MongoDB* se describía la estructura de los documentos almacenados en esta base de datos y las maneras más eficientes de estructurar los documentos. En este manual vienen explicadas de qué se tienen que realizar las relaciones de objetos, dependientes unos de otros.

Por ejemplo, un objeto *Modelo de coche* y un objeto *Carrocería,* es mejor organizarlos dentro de un mismo documento y no en dos documentos relacionados por una clave foránea, sobre todo en el caso de que se tenga pensado realizar consultas sobre la clase *Coche* con atributos de la *Carrocería,* ya que si no se tendrían que hacer más de una consulta para obtener el resultado.

En el manual también se indiciaban las distintas formas de realizar consultas a las colecciones de *MongoDB*, desde los *finds*, que tenían una estructura bastante simple, a los *aggregations* que eran más complejos, pero permitían consultas más potentes.

Para la conexión entre *Python* y *MongoDB* se decidió por el uso de *PyMongo*, ya que es una librería que permite conectar de manera extremadamente sencilla el lenguaje de programación y la base de datos.

En la investigación de las librerías se investigó parcialmente *Pandas*, librería para el análisis de grandes cantidades de datos en *Python*, pero se optó por no usarlas ya que existen problemas si no se conoce la ubicación de las exactas del *CSV*. Para poder saberlo hay que realizar una búsqueda por las primeras líneas del fichero y solamente esto, supone la destrucción de una de las grandes ventajas de *Pandas* que es el bajo costo de memoria a la hora de leer un archivo. También hay que destacar que la tarea del importador es limpiar los datos y darles sentido lógico y estructural a estos y no hacer análisis de estos directamente, que es el otro gran beneficio de *Pandas*. Aun así, se ha marcado como posible mejora futura el intento de integración de esta librería con el importador de los datos.

Para el uso de gramáticas se revisó *ANTLR4* y se vio que no tenía sentido su uso para la lectura del fichero, aunque se ha terminado usando en algunos pequeños algoritmos.

2.2.3 Análisis y definición de los datos

Inicialmente, la Policía Nacional de Málaga proporcionó para este TFG tres ficheros *CSV* con datos de las denuncias recibidas por en los meses de octubre de 2017 y junio y octubre de 2018. Se procedió al análisis de los datos observados y se identificaron los siguientes grupos de datos comunes dentro del *CSV*: el *Hecho*, los *Responsables*, las víctimas y testigos *Implicados*, los *Objetos*, la *Actuación policial* y la *Detención del responsable*. Dentro de estos mismos, se identificaron también otros datos comunes tales como los datos relativos a las fechas y lugares donde se llevaron a cabo cada una de las acciones.

De este análisis previo se tomó como identificador principal el *número de la actuación*, aunque después pasaría a serlo la tripleta formada por el *número de la actuación,* el ***año de la fecha de actuación*** y el ***código de la plantilla de actuación***.

Otro aspecto a destacar es la posibilidad de tener hasta cierto punto, objetos variables. Es decir, dos documentos *CSV* no tienen por qué tener el mismo número de campos.

Después de este análisis previo se pensó en cómo se iban a almacenar los datos dentro de *MongoDB* y, siguiendo su manual de recomendaciones, se decidió almacenar en un único documento *BSON* [**Figura 2.2.3.1**], en detrimento de separar los objetos en distintas colecciones, acción que de acuerdo al manual de recomendaciones de *MongoDB* solo se debería llevar a cabo cuando se vaya a replicar excesiva información y cuando no se vayan a llevar a cabo

consultas sobre atributos de estos objetos internos, ya que supondría un aumento del número de consultas necesarias para obtener el resultado esperado.

En el caso de estudio no se replicaba demasiada información ya que la amplia mayoría de los objetos internos eran relativos a la denuncia en cuestión y tenían, por tanto, atributos diferentes. También se desconocía, en un primer momento, todos los atributos sobre los que se iban a realizar consultas, por lo que se decidió mantenerlos integrados en un mismo documento.

Esta decisión permitió que el proyecto fuera escalable en el futuro, añadiendo nuevos atributos sobre los que realizar búsquedas.

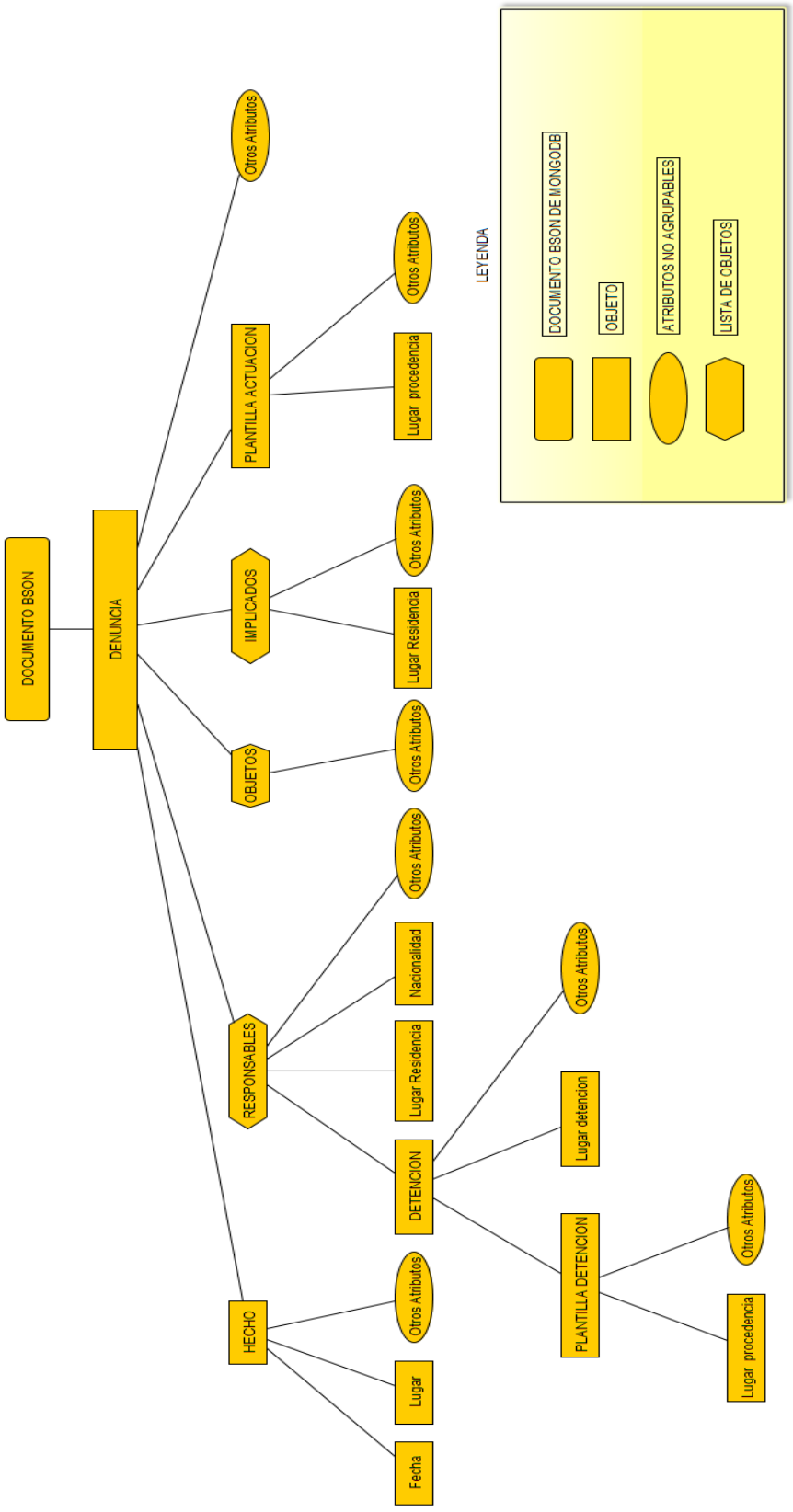


Figura 2.2.3.1 *Estructura del documento BSON*

Inicialmente, se destacaron ciertos atributos, los cuales se consideraban importantes antes de hablar con la Policía, estos eran:

- Tipología del hecho.
- *Modus Operandi*.
- Lugar de ocurrencia (lugar general).
- Lugar de ocurrencia (tipo de establecimiento).
- Nacionalidad, sexo y edad del responsable.
- Nacionalidad, sexo y edad de los implicados.
- Tipo de objetos.

En base a esto, la siguiente decisión fue crear un tipo de documento que recogiese todos estos atributos diferenciados por lugares y fechas señaladas, de forma que se pudieran obtener, de manera rápida, datos resumidos sin tener que hacer todas las consultas necesarias.

Esto llevó a la concreción de otro tipo de documento *BSON* que se denominó *Localización*, y a la colección donde se almacenaron estos documentos se le denominó *Localizaciones*.

Este documento se construiría en el propio *Importador* y tendría los datos resumidos de los atributos que la propia policía destacó y que fueran destacados en el futuro.

Por último, se tomó la decisión de plantear un tercer documento que se organizaría por provincias y que fueran los valores de los desplegables de la interfaz. Esta decisión se tomó por dos razones. La primera, que estos valores no eran fijos, sino que variaban de acuerdo con los valores de la policía y, la segunda, que, si se reunían en un único documento, la carga de estos valores sería casi instantánea.

2.2.4 Modelado del sistema

Se diseño un servidor basado en microservicios *REST* de forma que no fuera dependiente de la interfaz y, por ende, que se pudiera crear, si así se quisiera, versiones para otros dispositivos que reutilicen los servicios de la aplicación.

Los usuarios estarían divididos en dos roles: los administradores y los usuarios normales, ambos requiriendo autentificación previa en el sistema ya que acceden a datos privados. El uso de administradores ha permitido no controlar ciertos errores derivados de un mal uso de la aplicación, principalmente en el *Importador*, en el que no se han tenido en cuenta archivos maliciosos o completamente erróneos.

El modelado del módulo de lectura se separó inicialmente en 3 partes. A saber:

- Una clase para abrir y procesar las líneas de cada uno de los ficheros (denominado *Lector*).
- Una clase para almacenar las distintas entidades (denominada inicialmente como *Diccionario*).
- Una clase para guardar las entidades producto de la lectura de los ficheros (denominada *Denuncia*).

No se procedió a la importación directamente de cada denuncia leída en la base de datos, debido a que la policía había dejado claro que cada denuncia podía tener varias entradas tanto en un mismo fichero como en varios, lo que conllevaba dos posibles soluciones. Una primera, la importación directa de cada una de las *Denuncias* a la base de datos e ir actualizándolas cuando encontrara otra entrada de la misma *Denuncia*, lo cual conllevaba una gran cantidad de tiempo; y otra, el almacenamiento en memoria de las denuncias las cuales se actualizarían por cada una de las entradas en los documentos de la misma. Esta

segunda solución plantea mejoras en el tiempo de importación, así como problemas los de memoria, que se comentarán más adelante.

2.3 Primera reunión con la Policía Nacional de Málaga

La primera reunión con la policía se realizó el 12 de marzo de 2019, en ella se presentaron los prototipos de la interfaz y la idea básica de la estructuración de los datos. Asimismo, se les preguntó acerca de los atributos más importantes sobre los que se iban a realizar las búsquedas y las simulaciones.

Los atributos más relevantes para la policía fueron un subconjunto de los que habíamos decidido nosotros. Fueron concretamente:

- Tipología del hecho.
- *Modus Operandi.*
- Lugar de ocurrencia (tipo de establecimiento).
- Día de la semana.
- Nacionalidad y sexo del responsable.

Cuando se presentó la idea básica de la estructuración de los datos, se preguntó por el identificador de las denuncias, ya que se había identificado únicamente el *Número de Actuación* como el atributo que diferenciaba una denuncia de otra. Esto se descubrió como erróneo cuando se nos informó de que no existía un atributo único como tal, sino que la única manera de identificarlos era por la tripleta de atributos formada por el *número de la actuación*, el *año de la fecha de actuación* y el *código de la plantilla de actuación.*

Por eso mismo, se acordó con la policía que la fecha de actuación iba tener siempre el mismo formato y que los campos pertenecientes al identificador iban a aparecer siempre dentro del CSV generado por ellos, bajo los nombres

[$AA03 NumActuacion].[Numactuacion], [$AA03Plantilla Actuacion].[Plantilla Cod] y *[$AA03 Fecha Actuacion].[Dia Actuacion].*

De cara a la presentación de la interfaz, a la policía les gustó la idea planteada de visualizar inicialmente el mapa de Málaga con los datos resumidos en ellos.

Por último, se comentó el tema de los permisos dentro de la aplicación. Inicialmente la idea que tenía la Policía era que todo el mundo pudiese importar datos a la aplicación. Todo lo cual se habló, decidiendo así el papel de administradores que serían los únicos que pudiesen importar datos.

3

Iteración 1

3.1 Introducción

En la primera iteración de implementación del proyecto se desarrolló una versión básica del *Importador* separada del resto del proyecto y se planteó su posterior inclusión dentro del servidor para la siguiente iteración.

3.2 Desarrollo desde el punto de vista del procesamiento de datos

Para realizar esta primera versión del *Importador* se establecieron un total de 5 requisitos y la realización de las pruebas en base a los 3 *datasets* que teníamos a nuestra disposición:

- Lectura de los datos y limpieza de los datos.
- Creación de la clase *Denuncia.*
- Creación de la clase para almacenar las denuncias.
- Creación de las clases *Localización* y *Cabeceras.*
- Mejoras de rendimiento para el *Importador.*

3.2.1 Lectura de los datos y limpieza de los datos

Los tres ficheros *CSV* tenían estructura muy similar, unas 20.000 líneas en cada fichero y unos 90 atributos por cada entrada, sabiendo de antemano que una Denuncia podía estar representada por varias entradas en las que se especificaban distintos responsables, implicados u objetos.

La codificación de los ficheros era *UTF-8* y se usaba como separador del *CSV* el punto y coma. En los archivos se destacan dos tipos de datos en cuanto a la procedencia de su creación. En primer lugar, los que habían sido seleccionables, cuya característica más destacable es que no tenían caracteres extraños y estaban en mayúscula, y, en segundo lugar, los introducidos a mano por agentes de policía y que contenían caracteres extraños tales como tabuladores o retornos de carro (saltos de línea) y un número de puntos, espacios y comas desconocido tanto al inicio al final como entre las palabras.

Por último, las cabeceras del CSV (nombres de las columnas) no estaban en la primera línea, sino que estaban en alguna línea más avanzada, lo que suponía el tener que encontrar las cabeceras para proceder a leer el fichero. Se supuso que cada uno de los nombres de las columnas estaba separado en grupos separados por un punto y entre un par de corchetes y en el primero de ellos había un símbolo del dólar seguido de un código alfanumérico. Ejemplo de una cabecera:

[$AA01 Origen Actuacion].[Origen Actuacion]

Gracias a esto se creó un método que comprobaba si una palabra era una cabecera comparándola con una expresión regular. En caso de que no se hubiera encontrado la línea de las cabeceras y que esta línea no contuviese cabeceras, se procedía a ignorar la línea y continuar.

Una vez identificada la línea de las cabeceras, se procedió a limpiar de códigos, corchetes y posibles caracteres extraños cada una de ellas, lo que daba como resultado la siguiente cabecera:

Origen Actuacion.Origen Actuacion

Por último, se intentó dar un sentido estructural a lo que significaba cada una de las cabeceras, por lo que se comprobó si esta contenía alguna palabra que la identificase como perteneciente a uno de los objetos principales de la cada Denuncia.

El ejemplo anterior pertenece al objeto ***Actuación*** y su atributo es ***Origen,*** por lo que después de la identificación se procedió a eliminar la palabra Actuación de la cabecera y a eliminar todas las palabras anteriores al último punto. Por último, se procedió a poner el nombre del objeto principal antes del nombre del atributo, separados de éste por un único punto. Siguiendo el ejemplo anterior, esto da como resultado:

Actuacion.Origen

Para estandarizar los nombres de los objetos principales se procedió a almacenar en un diccionario el nombre que se le iba a poner y al identificar que un objeto es uno de estos objetos principales se introduce la palabra almacenada. Las palabras a eliminar por cada objeto principal también variaban, aunque principalmente contenían el nombre en plural del objeto en cuestión y algunas palabras donde también aparecía el nombre de esa entidad.

Inicialmente estas palabras se encontraban en un diccionario en *Python*, pero cuando fueron creciendo en tamaño y se incluyeron los objetos dentro de los objetos principales, se tomó la decisión de almacenarlos dentro de un base de datos interna del propio *Importador*.

Esta base de datos se hizo en *SQLite* e inicialmente se crearon las tablas de *PalabraClavePrincipal*, *PalabraSecundaria* y *PalabraListaNegra* y los métodos para poder acceder a las traducciones y a las palabras de cada una de ellas. [**Figura 3.2.1.1**]

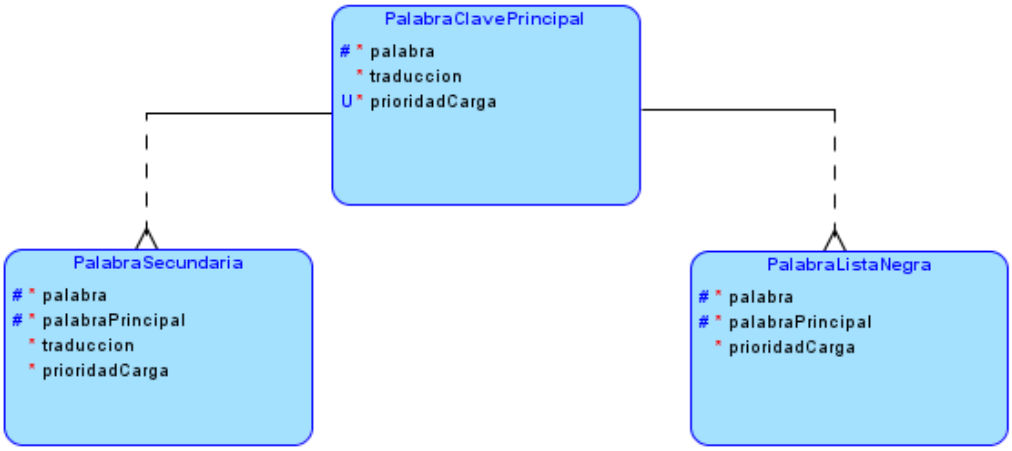


Figura 3.2.1.1 *Modelo entidad relación de las palabras clave de las cabeceras de la base de datos interna del Importador*

Una vez que las cabeceras se encuentran identificadas, limpias y con los arreglos necesarios para que se puedan entender con mayor facilidad, se procedió a la limpieza y estructuración de datos gracias a estas cabeceras. [**Figura 3.2.1.2**]

Para la limpieza de los datos se siguió el principio de menor modificación posible de los mismos, es decir, dejarlos limpios de caracteres extraños pero lo más parecidos posibles a como venían en los *datasets* originales. Por ejemplo, no se cambiaron las palabras en mayúsculas por otras en minúsculas, principalmente ya que, si en un fututo se pidiera la exportación de esos datos para su posterior ingreso en bases de datos de la policía, se conservará su estandarización.

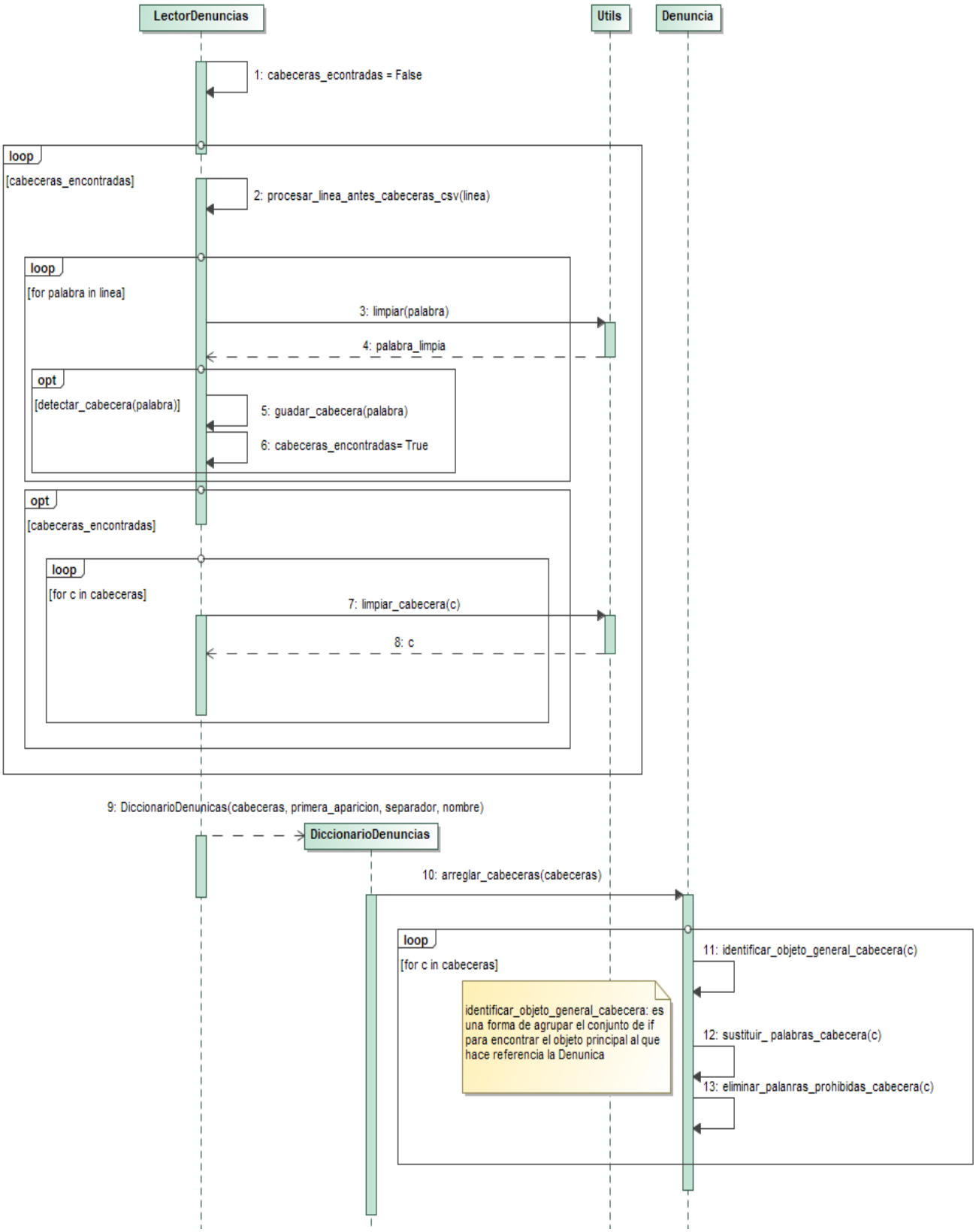


Figura 3.2.1.2 *Fragmento del algoritmo de lectura, dedicada a la identificación, limpieza y procesamiento de las cabeceras.*

3.2.2 Creación de la clase *Denuncia*

La clase *Denuncia* debía ser lo suficientemente flexible para poder almacenar un número indeterminado de atributos dentro de la clase, permitiendo una exportación fácil a *MongoDB.* Por ello, se tomó la decisión de crear un diccionario de *Python* dentro de la clase *Denuncia* que pudiera representar con eficacia su contenido.

Cada uno de los objetos internos dentro de la *Denuncia* sería a su vez otro diccionario, con la salvedad de los *objetos*, *responsables* e *implicados* de la misma que serían listas de diccionarios, puesto que puede haber varios objetos de estas categorías dentro de la misma entidad.

Este formato facilitaba también las consultas sobre atributos de la clase y la exportación de la *Denuncia* a la base de datos, ya que un diccionario *Python* tiene una estructura casi idéntica a un fichero *JSON* y la exportación a *MongoDB* se puede realizar en este formato, ya que su formato base son los ficheros *BSON*, que son una versión de los ficheros *JSON,* pero con un identificador llamado *_id* que es propio de la base de datos.

Se crearon *getters* de los atributos principales de la denuncia, así como el identificador de la denuncia, unificando los tres atributos que según los propios policías formaban el código distintivo de la *Denuncia.* Estos son el *Número de Actuación,* el *Código de la Plantilla de Actuación* y el año de la fecha presente en el atributo *Día de Actuación.*

Después se procedió a crear los nuevos atributos de la *Denuncia, a saber,* el día de la semana y el tramo horario del Hecho. Esto permitía reducir las consultas de los atributos, ya que no habría que buscar en un intervalo horario, sino que se buscaría directamente un valor concreto.

Por último, se procedió a realizar una forma de unificar dos denuncias que tuvieran el mismo identificador, en una misma denuncia.

3.2.3 Creación de la clase para almacenar las *Denuncias*

Para poder tener almacenadas las denuncias de manera temporal y poder ir uniendo las distintas entradas de cada una de ellas, se desechó la idea de introducirlas en la base datos de manera instantánea, debido a que en un fichero *CSV* podría haber normalmente 2, 3 o 4 entradas por cada denuncia y, en algunos casos, pudiendo llegar hasta 300 entradas.

Esto que supondría tener que realizar numerosas consultas a una base de datos para crear y actualizar una misma entidad; esto implicaría también, complicar la creación de objeto resumen, pues se tendrían que recalcular estos objetos el mismo número de veces. Es importante mencionar que estas entradas contienen nuevas entradas de *objetos*, *responsables* o *implicados* dentro de la misma ***Denuncia***.

Por esto se decidió la creación de una clase que se denominó ***DiccionarioDenuncias,*** que sería la encargada de mantener todas las denuncias dentro de un diccionario con el identificador de éstas sirviendo como identificador de la propia estructura de datos. Esto permitirá la unión de denuncias con relativa facilidad, ya que al terminar de leer una Denuncia se comprueba si está ya presente en el diccionario y se actualizan los nuevos valores de las listas mencionadas anteriormente.

Esta clase se usó también para sacar datos estadísticos de la lectura de los ficheros y poder presentarlos y razonarlos en posteriores reuniones con los agentes de la policía. Con la creación de esta clase se terminó el modelo básico de lectura [**Figura 3.2.3.1**].

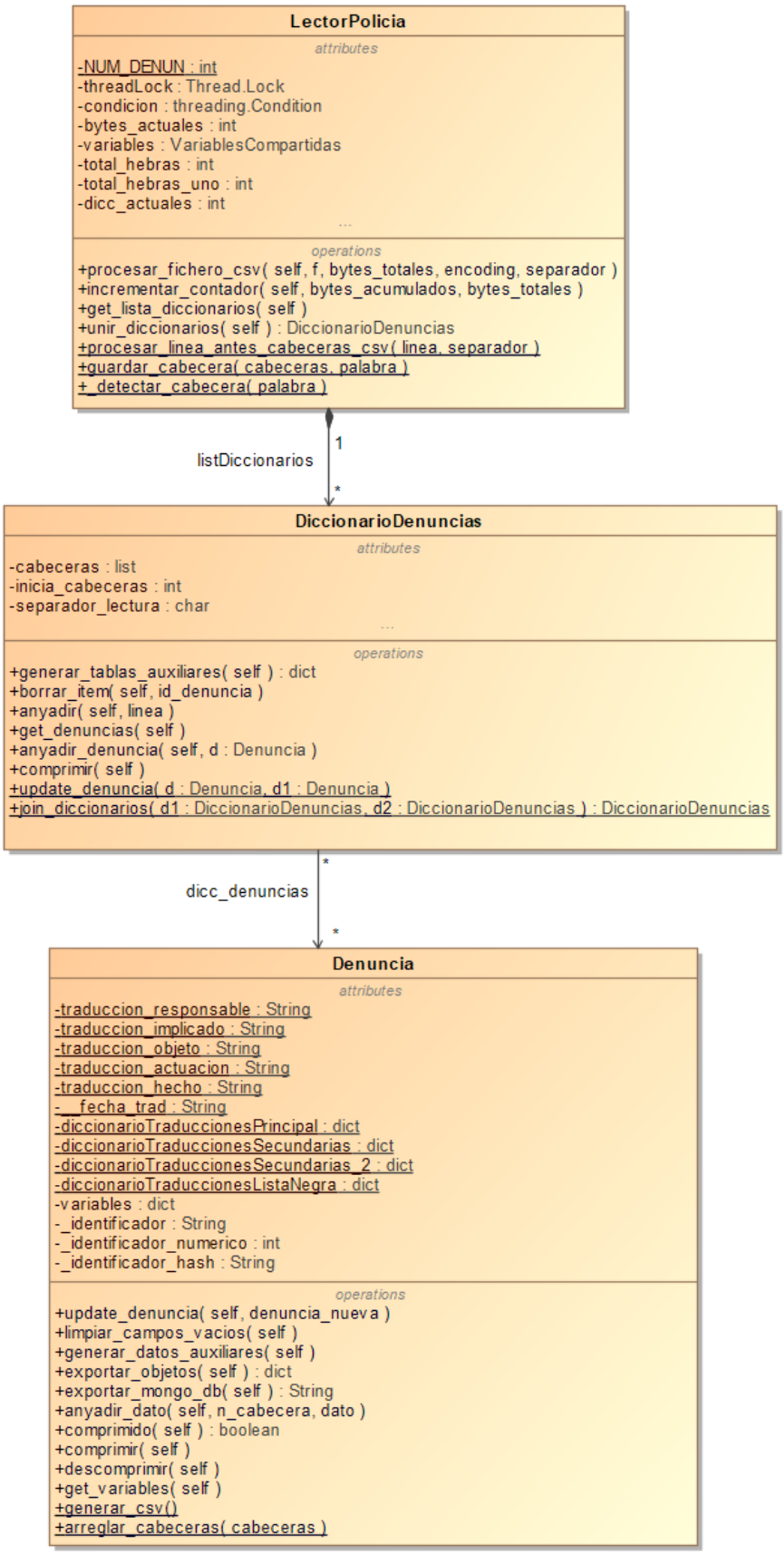


Figura 3.2.3.1 *Diagrama de clases base del importador*

3.2.4 Creación de las clases *Localización* y *Cabeceras*

La primera clase se denominó *Localización* puesto que, en un primer momento, solo iba a tener como identificador el lugar donde se producía cada *Hecho* (Objeto interno de la *Denuncia* que representa qué sucedió, dónde y cuándo) de cada *Denuncia,* ampliándose posteriormente para incluir también la fecha. Esta clase contiene un breve resumen de los datos de los atributos indicados como claves por la policía y sobre los que se iban a realizar las consultas. Estos atributos fueron:

- *Modus operandi* del *Hecho* de la *Denuncia.*
- Tipo de lugar específico del *Hecho* de la *Denuncia.*
- Tipología del *Hecho* de la *Denuncia.*
- Información del género y nacionalidad de los responsables.
- Día de la semana en el que se cometió el *Hecho* de la *Denuncia.*
- Tramo horario en el que se cometió el *Hecho* de la *Denuncia.* Originalmente dividido en *Mañana, Tarde, Noche y Madrugada.*

A esto se le añadían el número total de denuncias o hechos de cada *Localización,* así como sus identificadores principales que eran inicialmente:

- Mes
- Año
- Provincia
- Municipio
- Distrito policial

Estos atributos se combinaban entre sí, junto con caracteres especiales que hacían de separador para formar el identificador de la *Localización.* El único de estos atributos que se considera obligatorio es la provincia, siendo los demás dependientes de lo que se quiera representar dentro de la *Localización* concreta.

Algunos ejemplos de identificadores son:

- **2018-1#MALAGA**: representa el resumen de lo acontecido el mes de enero de 2018 en la provincia de Málaga.
- **MALAGA_MARBELLA_MARBELLA**: representa el resumen de lo acontecido en el distrito policial *MARBELLA* de Marbella (el único distrito que se identifica en esta población) de la provincia de Málaga.
- **@1MALAGA**: representa el resumen de lo acontecido los meses de enero en la provincia de Málaga.

Una vez generadas las *Localizaciones* se procedió a usarlas para generar las entidades de las *Cabeceras*, que son los valores que se introducirían en los seleccionables de la interfaz de la aplicación.

Para ello se seleccionaban únicamente las *Localizaciones* que solo contenían el valor *Provincia* y el valor *Municipio* relleno, y de ahí se extraían los valores de los atributos sobre los que se iban a realizar las consultas.

3.2.5 Mejoras de rendimiento del *Importador*

Se intentó mejorar el tiempo de lectura de los ficheros, para ello se recurrió al uso de *Threads.* A cada uno de los *Hilos* se le asignaba un fichero y después se procedía a juntar los resultados de cada uno en un mismo objeto común sobre el que se generarían las tablas auxiliares. El uso de *Threads* con respecto a la ejecución secuencial mejoró en un 30% el tiempo de ejecución. También se procedió a la creación de numerosas variables de clase que reducían el número de consultas que se realizaban a la base de datos interna del *Importador* y por ende el tiempo de ejecución del mismo.

Por último, se procedió al cálculo de los nuevos atributos solamente en el caso de nuevas denuncias, lo que reducía en aproximadamente un 30% las ejecuciones de esa parte del código.

3.3 Segunda reunión con la Policía Nacional de Málaga

En esta segunda reunión tuvo lugar el 17/05/2019. Se presentó un prototipo funcional con los datos que se tenían hasta ese momento. En ese prototipo solo estaba habilitado el mapa de la provincia de Málaga y solo se permitían consultas simples dentro de un mismo campo, es decir, los seleccionables solo permitían un valor.

Se dio información acerca de los datos que evidenciaban que los campo *Fecha_hasta* y *Hora_hasta* del *Hecho* de la *Denuncia* no tenían relevancia pues sólo en menos del 6% de las entradas los campos eran relevantes y se presentó la división por tramos horarios de las denuncias, dentro del cual se eliminó la *Madrugada*, que se integró dentro de la *Noche*.

También se les preguntó acerca de la posibilidad de introducir nuevamente denuncias que ya estaban en la base de datos y actualizarlas, aunque se indicó que podría llevar bastante tiempo hacer todas estas actualizaciones. Su respuesta fue que no hacía falta y que en el caso de detectar que una denuncia ya se encontraba en la base de datos, se procediera a ignorarla, ya que lo más probable es que no incluyera nueva información y se perdería mucho tiempo.

La Policía planteó la posibilidad de poder buscar por días festivos, recalcaron la importancia de la búsqueda por distritos municipales, y pidieron asimismo que se pudieran hacer consultas con valores múltiples dentro de los seleccionables de la interfaz, confirmando así la necesidad de poder ver información resumida de la zona donde se plantea la búsqueda, con los filtros seleccionados.

Por último, se hizo nos entrega de otros dos *datasets* con información adicional, los cuales tenían otra codificación, otro separador y un formato parcialmente distinto del que nos fue entregado inicialmente.

4

Iteración 2

4.1 Introducción

En la segunda iteración del proyecto se hicieron los ajustes necesarios para integrar los dos nuevos formatos de los ficheros de datos al *Importador*, se unificó éste con el resto de del proyecto y se desarrollaron tres módulos a incluir dentro del *Importador*. Por último, se empezaron a refactorizar las consultas desde *Python* a la base de datos.

4.2 Desarrollo desde el punto de vista del procesamiento de datos

Para realizar esta iteración desde el punto de vista de este TFG se plantearon los siguientes requisitos:

- Refactorizar el *Importador* para los nuevos *datasets.*
- Integrar el *Importador* en el servidor. Para ello:
  - Crear los servicios de lectura.
  - Importación directa a *MongoDB.*
  - Actualización de las tablas auxiliares.

        - Permitir conocer el estado de la lectura, pues iba a ser una tarea que iba a llevar mucho tiempo.
    - Creación del módulo *Festividades.*
        - Días festivos concretos.
        - Periodos festivos.
- Creación de un módulo que permita convertir *CSV* de tamaño reducido a la base de datos interna del *Importador* para poder usar esos datos en el preprocesamiento.
    - Lectura del fichero.
    - Creación de tablas y almacenamiento de entidades.
    - Exportación a *SQL* de la base de datos generada.
- Conseguir el distrito municipal donde sucedió el *Hecho* de la *Denuncia.*

Además, se añadió la refactorización y actualización de los métodos de consulta a *MongoDB,* cuyas primeras versiones fueron realizadas por Sergio Gavilán Prieto. Esto se enmarca en la realización de tareas comunes para el proyecto.

4.2.1 Refactorizar el *Importador* para los nuevos *datasets*

Los nuevos *datasets* incluían la información de todas la Denuncias realizadas a la Policía Nacional en la provincia de Málaga en el año 2018 y todos sus responsables. Estos ficheros estaban codificados en una versión especial de *UTF-8*, denominada *UTF-8-sig*, que es una variante creada por Windows y usaban como separador la coma. Esto implicaba identificar la codificación del fichero y su separador antes de proceder a identificarlo.

Para ello se identificaba primero la codificación mediante la identificación de errores a la hora de decodificar los ficheros; se permitió el uso de tres tipos de codificaciones *UTF-8*, *UTF-8-sig* y *CP1252* (codificación española usada por Microsoft Excel).

Una vez identificado el tipo de codificación se usó el algoritmo empleado por Microsoft Excel para conocer el separador de los *CSV,* el cual consiste en leer las primeras líneas del fichero y contar el número de apariciones de la coma o del punto y coma. En este caso se leen las 400 primeras líneas.

Una vez realizadas estas dos tareas, se procedió a leer y preprocesar los nuevos ficheros y a verificar si su resultado era correcto.

4.2.2 Integrar el *Importador* en el servidor

Para poder integrar el módulo de lectura dentro del servidor se siguieron una serie de pasos que corresponden a los puntos tratados a continuación.

4.2.2.1 Crear los servicios de lectura

Los servicios para permitir la lectura de los ficheros se hicieron en un módulo distinto del resto de servicios, se implementaron bajo el *Blueprint lector/V1/,* de forma que todos los métodos relacionados con la lectura de los ficheros estuviesen agrupados. Para poder acceder a esta dirección es necesario contar con permisos de administrados, lo que permite no tener que controlar totalmente la posibilidad de que se introduzcan archivos maliciosos.

Se crearon dos servicios, uno para la lectura de los ficheros y otro, para devolver el estado del *Importador.* El servicio *REST* para la lectura de los ficheros se limita a 200 megas para no tener ningún problema a la hora de procesar la lectura. El tamaño de un archivo con los datos correspondientes a un año es de aproximadamente 100 megas, por lo que este limitador no supone un realmente un problema El segundo servicio se creó con la idea de poder devolver el estado del *Importador.* Para ello se pensaron en las tres fases del *Importador*: el procesamiento de los datos; la comprobación en la base de datos de las *Denuncias* procesadas y, la generación y la importación de las *Denuncias*

y sus tablas auxiliares. Este método, por lo tanto, devuelve una lista con los valores [% *procesamiento,* % *comprobación,* % *importación].*

4.2.2.2 Importación directa MongoDB

Para la exportación a *MongoDB* desde *Python* y no desde ficheros se crearon dos métodos. Uno para comprobar y eliminar de la importación aquellas *Denuncias* que ya estaban presentes en la base de datos, y otro para generar las tablas auxiliares y proceder a la exportación de éstas junto con las *Denuncias* a la base de datos.

El primer método hace grupos de 1000 Denuncias y comprueba si sus identificadores están presentes en *MongoDB.* De esta forma no hace falta ir comprobando una a una la presencia de cada entidad en la base de datos.

El segundo método genera, en primer lugar, las *Localizaciones* y las *Cabeceras* a importar, las actualiza en la base de datos y procede después a su eliminación, exportando, por último, las denuncias en grupos de 5000.

La exportación desde *Python* a *MongoDB* se hace de forma directa si el objeto exportado es un diccionario, de ahí el uso de éste para almacenar las variables de las denuncias. La limitación del número de denuncias exportada cada vez es consecuencia del uso de la memoria interna de *Python.*

4.2.2.3 Actualización de las tablas auxiliares

La actualización de las tablas auxiliares requiere importar éstas a objetos *Python* nuevamente. Para ello, en primer lugar, se generan las tablas auxiliares que se generarían si la base de datos se encontrara vacía, y se va comprobando si el identificador de cada entidad de estas tablas se encuentra ya presente en *MongoDB.*

En caso de que no se encuentre, se procederá a su exportación, si no se generará el objeto *Python* de la misma y se juntará con la nueva entidad generada para sustituirla por la que está en *MongoDB*.

4.2.2.4 Permitir conocer el estado de la lectura

Para conocer el estado de la lectura se creó una clase llamada *Variables* que iba a contener las variables de lectura del *Importador*. Esta clase está diseñada pensando en que el módulo de lectura hace uso de la concurrencia para preprocesar los datos.

En *Python* la asignación, modificación y consulta de variables de sus tipos básicos está protegido contra el uso de *Threads*, lo que incluye las listas y los diccionarios. Para modificar los atributos de esta clase, que no son más que los porcentajes mencionadas en el punto *4.2.2.1*, se añadieron los métodos correspondientes en tres métodos principales del *Importador*.

En primer lugar, se introdujo en el método que lee los datos de los ficheros, al que se le pasa como parámetro el número de bytes de todos los ficheros a leer. Por otro lado, se mantiene en una variable el peso del total procesado ya por el *Importador*, el cual va siendo modificado por cada una de las hebras. Esto supone que generar el porcentaje del procesamiento de los datos consiste en hacer una simple división.

En segundo lugar, se modificó el método de la comprobación de la base de datos para que el número de denuncias fueran comprobadas de mil en mil, partiendo de que se sabe antes de empezar el número total de *Denuncias* a comprobar. Vuelve a ser una simple división lo que genera el porcentaje de la comprobación total de los datos.

En tercer y último lugar, se dividió en dos fases el cálculo del total de la importación realizada. Por un lado, se encontraba el número de tablas auxiliares

actualizadas y, por otro, el número de *Denuncias* importadas. Esta división es del 50% del tiempo de exportación total para cada una de ellas.

4.2.3 Creación del módulo *Festividades*

Se procedió, en primer lugar, a separar entre festividades de un día o de una noche concreta con periodos festivos, ya que los primeros tienen una fecha concreta dentro del calendario. Es decir, la pregunta que se tiene que realizar es: *¿corresponde esta fecha a un día festivo?,* mientras que en los periodos festivos se tienen que calcular las fechas entre las que se desarrolla esta festividad y después comprobar si una fecha se encuentra dentro del intervalo o no.

4.2.3.1 Días festivos concretos

En el primer grupo se decidió introducir los siguientes días:

- Día del trabajador
- Halloween
- El día de los muertos
- San Juan
- Virgen de la Victoria
- San Valentín
- Día de la Constitución
- Dia de la Inmaculada
- Día de la Hispanidad

Para una mayor adaptabilidad se introdujo una gramática escrita en *ANTLR4* que procesara los valores y su fecha concreta de un fichero de texto llamado *festividades*, de forma que para añadir otra festividad de este grupo solamente haría falta añadir una línea con los datos de esta a este fichero.

4.2.3.2 Periodos festivos

Los periodos festivos son las fiestas que abarcan numerosos días y que tienen días concretos que son fechas clave para muchas personas. Entre los periodos festivos que se celebran en Málaga, se decidió la inclusión de los siguientes:

- Feria de Málaga
- Carnaval
- Navidad
- Semana Santa
- Semana Blanca

Para cada una de estas festividades se definió una clase que devolvía el valor de esta festividad y si era un día concreto dentro de esta, dada una fecha que se le pasaba como parámetro.

Dentro de estos periodos solamente la *Navidad* tiene una fecha que no varía de un año a otro. *Carnaval* y *Semana Santa* se basan ambos en el Domingo de Resurrección para designar los días concretos de fiesta. Para calcular este día existe un algoritmo público que lo calcula, este algoritmo se modificó parcialmente para adaptarlo al resto del código.

Por último, para la *Feria de Málaga* y para la *Semana Blanca*, se fijaron los días que siempre entran dentro de esta festividad y después de estudiar los últimos 10-20 años, se crearon los algoritmos que aproximan la fecha. El cálculo de las fechas la *Feria de Málaga* puede tener un error de ± 2 día, mientras que el error para el cálculo de la *Semana Blanca* es de ± 1 día.

4.2.4 Creación de un módulo auxiliar para pasar de CSV a bases de datos relacionales

Una de las bases de este proyecto es la idea de realizar un producto basado en iteraciones de forma que se pueda ir ampliando el proyecto con nueva información o datos sin la necesidad de rehacer gran parte del código anterior. En esta idea base se enmarca este módulo.

Este módulo permite la traducción desde *CSV* a bases de datos relacionales de ficheros *CSV* cuya estructura no sea especialmente compleja. Su uso está destinado a desarrolladores y no a usuarios finales, debido a que suele ser necesario en primer lugar, añadir funcionalidades para usar esos nuevos datos y, en segundo lugar, se requeriría probablemente adaptar los datos a lo que necesite esa funcionalidad.

Este módulo trabaja con estructuras de datos en formato de árbol, es decir, un objeto raíz del que parten todos los objetos siguientes y es ideal para objetos simples.

La estructura de este módulo se basa en 3 partes:

1. La lectura del fichero, que se reutiliza en gran medida la parte ya programada para la lectura de la *Denuncia*.
2. La creación de las tablas y almacenamiento de las entidades en la base de datos.
3. La exportación a *SQL* de la base de datos generada.

4.2.4.1 Lectura del fichero

Para leer los ficheros *CSV* se abstrajo la parte de la lectura del fichero de la denuncia que no tenía que ver con el caso concreto, de forma que se crearon tres nuevas clases, la clase *Lector*; la clase *Diccionario* y, la clase *OImportable*.

Estas clases pasaron a ser padres de las clases de la lectura de la *Denuncia* y en ellas se encontraban numerosos métodos extrapolables a cualquier fichero *CSV*.

4.2.4.2 Creación de las tablas y almacenamiento de las entidades

Una vez todo el fichero se ha limpiado y estructurado en objetos *Python,* se procedió a crear las tablas y a identificar los tipos de los atributos. Para ello se creó una clase principal llamada *BDRelacional* en la que se van almacenando los datos en entidades de la clase *EntidadTabla* que tiene *AtributosRellenos*. Estas dos clases tienen referencias a la tabla o al atributo al que pertenecen, representados en las clases *Tabla* y *Atributo.* [**Figura 4.2.4.2.1**]

El objeto *Atributo* tiene un enumerado representando el tipo del atributo en *MySQL*, el cual se traduce en los *Escritores* a la base de datos relacional seleccionada. Por último, genera un estado de la base de datos actual por si se quiere importar varios archivos de una misma base de datos.

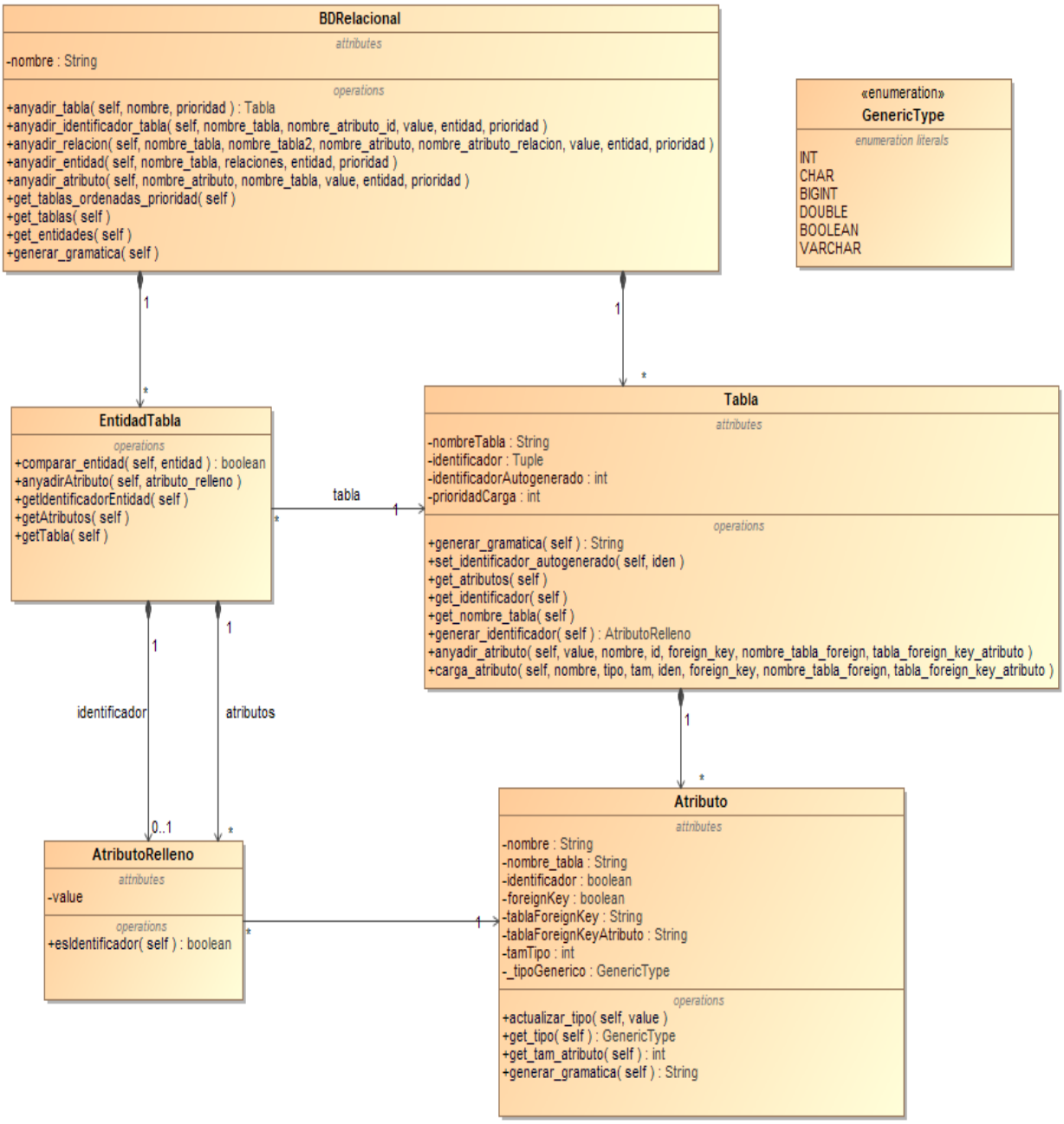


Figura 4.2.5.1 *Diagrama de clases del almacenamiento de datos del módulo auxiliar de exportación a bases de datos relacionales*

4.2.4.3 Exportación a SQL

La exportación a *SQL* se efectúa en las clases *Escritor*, de la cual heredan las clases de *Escritores* de cada base de datos. En la práctica solo están disponibles *MySQL*, lenguaje en el que se desarrollaron las pruebas, y SQLite, base de datos a la que se van a importar los datos para el *Importador.*

Esta escritura permite exportar la creación de una base de datos, la creación de las tablas de las entidades leídas y las inserciones de estas mismas entidades.

4.2.5 Conseguir el distrito municipal donde sucedió el *Hecho* de la *Denuncia*

Para poder conseguir el distrito municipal dada la calle donde sucedió el hecho, se necesitaban primero los datos de las calles de Málaga y los distritos donde están éstas. Para ello, se recurrieron a los datos abiertos del Ayuntamiento de Málaga, donde vienen recogidas diversos ficheros con información de Málaga, entre ellos está la información de las calles junto a su código postal y a su número de distrito.

Una vez encontrados los datos, se procedió a estudiarlos para ver cuál era el formato de los mismos. Los datos venían en un fichero *CSV*, codificados en *UTF-8* y separados por comas. En el caso de los nombres y la tipología de las calles, venían en un formato alejado de la interpretación normal de las mismas. Un ejemplo concreto sería:

Camino de los Albuceros -> CMNO, ALBUCEROS DE LOS

Debido a este formato se añadieron líneas al *CSV* que representaban otras formas de escribir el nombre de la calle.

Camino de los Albuceros -> CMNO, DE LOS ALBUCEROS

Una vez estudiados los datos, se procedió a usar el módulo de exportación a bases de datos relacionales, para introducir estos datos dentro de la base de datos del *Importador*.

Los datos relativos a las calles en los ficheros de la Policía eran muy diferentes unos de otros, ya que dependían de la persona que hubiera redactado cada *Denuncia,* la estructura más típica era *CALLE* o *PLAZA* como tipología de la vía y el nombre con los artículos al inicio.

También se daban circunstancias donde la tipología de la vía venía en el nombre, este es el caso de la *Alameda Principal*, que no es una calle sino una alameda o donde venía reflejado el nombre del lugar como, por ejemplo, *AEROPUERTO* o *AZUCARERA*.

Una vez introducidos los datos públicos en la base de datos del *Importador* junto a los datos de la Policía estudiados, se intentó crear el algoritmo de asignación del distrito de forma que redujera al mínimo los errores o fallas de interpretación entre los dos formatos de datos.

Este algoritmo verifica que el delito a buscar es del municipio de Málaga y que la calle es un campo de la *Denuncia*. En el caso de que la calle sea desconocida se pondrá el distrito 12 que identifica al distrito desconocido. Después comprueba si la calle estaba registrada previamente en un array interno de forma que no tenga que realizar la consulta correspondiente. Si no está registrado se invoca el método *get_distrito.* [**Figura 4.2.5.1**]

El método *get_distrito* recibe como argumento el nombre y el tipo de la vía. Inicialmente se busca únicamente por el nombre de la calle, y solo en el caso de que no haya ningún distrito o de que haya más de uno se realizan las siguientes partes del algoritmo.

En el caso de que no haya ninguno se elimina el tipo de vía y se sustituye por la primera palabra del nombre de la calle, ya que muchas veces la tipología de la calle viene en el nombre. Ejemplo es la *Alameda Principal,* que viene en los datos de la policía como *calle, Alameda Principal (tipología de la vía, nombre de la vía).*

Si antes de este paso hay más de un distrito se lleva a cabo la búsqueda añadiendo la tipología de la calle. Este paso suele reducir el número de distritos que aparecen en la consulta. En el caso de que haya más de un distrito después de esta búsqueda, se escoge uno al azar, ya que la mayoría de las veces son vias que abarcan uno, dos o tres distritos distintos. Por último, se ha de comentar que si hay algún error o no se encuentran distrito se devuelve el distrito 12, representando el distrito desconocido. [**Figura 4.2.5.2**]

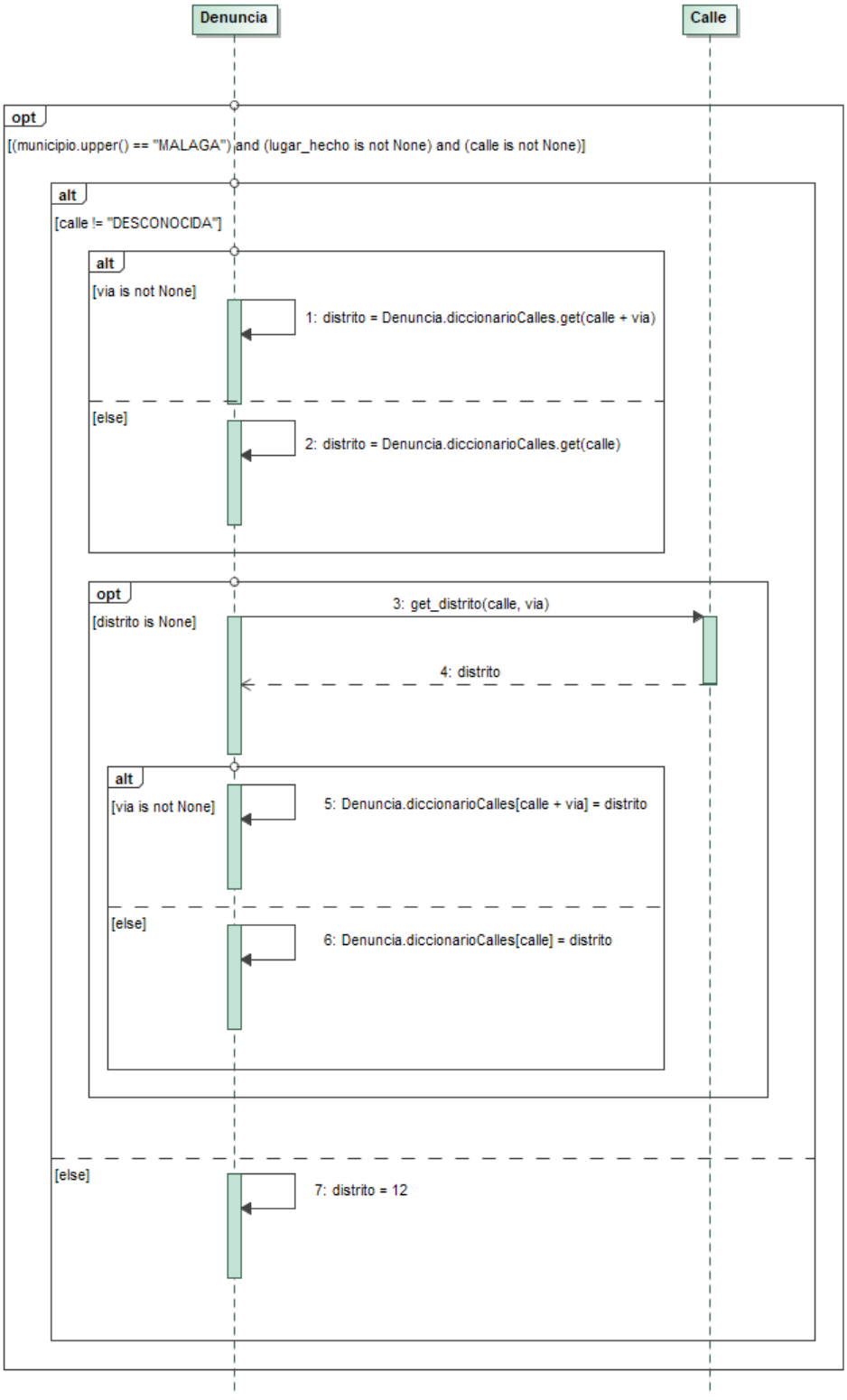


Figura 4.2.5.1 *Algoritmo para conseguir el distrito de una Denuncia*

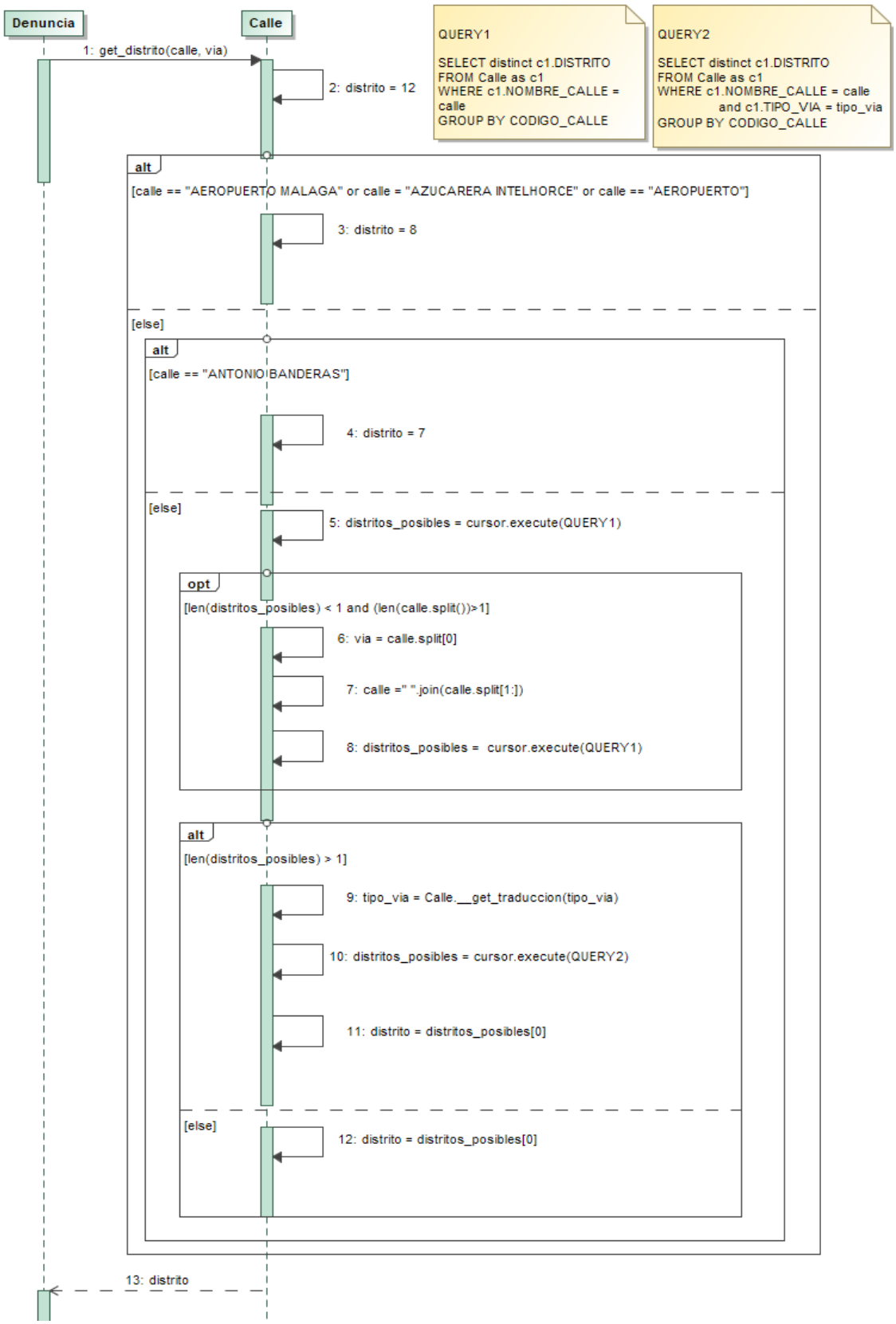


Figura 4.2.5.2 *Método get_distrito que devuelve el distrito correspondiente al nombre de una vía y a su tipo (plaza, calle, ...)*

4.3 Refactorización y mejora de los servicios de los *Filtros*

Está parte del proyecto es compartida con Sergio Gavilán Prieto. Él se encargó de realizar las primeras versiones de los servicios a refactorizar y de proveerles de la seguridad y la recepción de los parámetros, aunque se amplió el número de los parámetros de algunos de ellos.

La aplicación contiene dentro de los servicios de búsquedas básicos, pocos servicios, pero estos son muy flexibles en cuanto a la cantidad de parámetros a recibir, recibiendo estos, listas variables de atributos a introducir en la consulta a la base de datos.

Esta consulta, si se realiza desde *Python*, recibe como argumento un diccionario con los valores a buscar en la *BD* adaptados ya a la nomenclatura usada en *MongoDB*. Las primeras versiones construían estos diccionarios y usaban los métodos *find* para devolver los resultados, pero la mayoría de estos resultados requirieren que se tengan que hacer cientos de consultas con este método para poder representar los valores del mapa en cuestión.

Por ello, este método al requerir demasiado tiempo fue desechado y se utiliza una versión más avanzada del método *find* denominada *aggregate,* el cual tiene la desventaja de que en la actual versión de *MongoDB* no puede usar los índices de los *find*, pero en cambio permite realizar recopilaciones de datos que ahorran cientos de consultas a la base de datos [11].

Por esta razón se mantuvieron métodos que generaban el diccionario para generar la consulta, pero fueron cambiados todos los métodos que se realizaban sobre la consulta para adaptar los parámetros necesarios.

Además, también se completó el método *verZona* con los ajustes requeridos por los agentes de la Policía Nacional de Málaga, los cuales pedían que los

resultados de los filtros de las búsquedas filtraran también los resultados de dichas estadísticas.

También se rehicieron ciertos métodos internos sobre el *verZona* para permitir usar correctamente la colección de Localizaciones. Un ejemplo de la mejora con respecto a la búsqueda en la colección de *Denuncias* es que las consultas realizadas a la primera tardan menos de un segundo en bases de datos con gigas de información, mientras que algunas de las consultas más costosas que se pueden realizar, la cual agrupa cientos de consultas para generar estos datos, pueden elevarse a la orden de algunos minutos.

Por último, se añadieron los nuevos parámetros a las búsquedas y se ajustaron los métodos que creaban el diccionario para permitir la inclusión de los mismos. Estos nuevos atributos eran las Festividades y los distritos municipales.

4.4 Tercera reunión con la Policía Nacional de Málaga

La tercera reunión con los agentes de policía se realizó el 08/07/2019. En ella se presentaron las novedades con respecto a la versión anterior, las cuales permitían la visualización en zonas calientes de los delitos cometidos en los distritos municipales de Málaga, así como la visualización de la interfaz del *Importador* y de los filtros sobre múltiples valores de los seleccionables, entre el que se encontraba el de Festividades. También se mostraron los avances en cuanto a la seguridad de los usuarios y a cómo iba a funcionar la misma y se presentaron las limitaciones del *Importador*, las cuales no tuvieron relevancia en el resto de la reunión.

Por último, en esta reunión se plantearon los requisitos finales a tratar en esta primera versión del proyecto concerniente a este trabajo realizado en grupo.

En estos requisitos finales se incluyeron la exportación a documentos *CSV* de las entidades de la base de datos, la simulación de datos en mapas de calor y permitir la posibilidad de eliminar todos los datos de *MongoDB*.

5

Iteración 3

5.1 Introducción

En la última iteración del proyecto se hicieron los ajustes necesarios para mejorar el uso de memoria del *Importador*, así como se crearon los métodos de exportación nuevamente a *CSV*. Por último, se terminó de documentar y explicar el código perteneciente a lo realizado en este TFG.

5.2 Desarrollo desde el punto de vista del procesamiento de datos

En esta iteración se estudió el uso de la memoria del *Importador* y se intentó aumentar el rendimiento de ésta a costa del tiempo de procesamiento de los datos.

También se procedieron a crear los dos métodos de exportación a *CSV*. Uno que exportaba nuevamente las Entidades importadas, pero con los nuevos valores generados, limpios de caracteres extraños y con una lógica clara en el nombre de las cabeceras del fichero de datos. El segundo método desarrollado exportaba los resúmenes generados para las zonas y filtros seleccionados.

Por último, se creó el método más básico para poder eliminar el contenido de la base de datos.

Por eso, los requisitos que se establecieron para esta iteración son:

- Reducción del uso de la memoria empleada por el *Importador.*
- Exportación a *CSV.*
    - Exportación de estadísticas de las zonas seleccionadas.
    - Exportación de las entidades pertenecientes al filtro seleccionado.
- Permitir la eliminación de la base de datos.

5.2.1 Reducción del uso de la memoria empleada por el *Importador*

El procesamiento de datos al tener que realizarse directamente sobre objetos en memoria ocupa mucho espacio, si se tiene en cuenta que los ficheros están abiertos y que, por cada entidad de la clase *Denuncia,* se tiene un diccionario que contiene a su vez otras listas y otros diccionarios, lo que ocupa una gran cantidad de espacio en memoria. Esta entidad contendría un diccionario más por ser un objeto *Python* si no fuera por el uso del atributo *___slots___* que elimina este diccionario a costa de no permitir nuevas variables en el objeto.

El programa se testeó en un ordenador con 16GB de *RAM*, las cuales no estaban siendo utilizadas únicamente por *Python*, aun así, el programa aguantaba un total de aproximadamente 200MG de datos de los ficheros una vez ya todos ellos preprocesados.

Teniendo en cuenta por cada año hay alrededor de 150.000 denuncias, se podrían superar las 300.000 entidades en memoria si se introducen varios ficheros, en los que haya muchas denuncias que aparezcan en todos ellos.

Por ello se modificó el algoritmo de lectura para que no unificase las denuncias al final, sino que las fuese unificando conforme iba leyendo los datos, Lo que permitió no sobrecargar de información replicada la memoria de la *RAM,* que es la memoria de la que hace uso *Python.*

También se introdujo una funcionalidad al objeto *Denuncia* que permitía convertir el diccionario de variables a un *String* y poderlo reconvertir a un diccionario, de forma que almacenara mucha menos memoria en *Python.* Esta conversión se implementó gracias a la librería *JSON.*

Esta mejora en memoria tiene como contrapartida que empeora la eficiencia de tiempo, ya que tiene que convertir de un diccionario a *String* y de ésta nuevamente a diccionario cada vez que realiza una consulta o que actualiza la *Denuncia.* Por ello esta conversión se realiza en grupos de 5.000 *Denuncias* por *Thread*, para evitar en gran medida que las entradas sucesivas, que generalmente son de una misma *Denuncia,* se tengan que estar aplicando estos métodos una y otra vez.

Esta mejora en memoria permite tratar sin miedo hasta tamaños totales de ficheros cercanos a 500 megas, aunque la limitación se mantiene en 200 megas de forma que la memoria de *Python* no se desborde por causa de alguna situación no prevista.

5.2.2 Exportación a *CSV*

La exportación a *CSV* fue una funcionalidad que nos fue expresamente pedida por la Policía de Málaga, ya que hicieron hincapié en la necesidad de exportar las entidades de la base de datos a *CSV* y de las estadísticas de las zonas con los filtros previamente realizados. Dado que estos dos documentos a exportar no tenían un formato común, se crearon dos métodos para la exportación de ficheros *CSV.* Ambos servicios se crearon bajo otro nuevo *Blueprint,* que no requería de permisos de administrador para usarlo.

5.2.3.1 Exportación de las estadísticas de las zonas seleccionadas

La exportación de las estadísticas tenía que permitir la exportación de tres tipos diversos de formatos que, aún teniendo un formato distinto, tenían gran parte del mismo en común.

En estos tipos de formatos consideramos:

1. Los archivos directamente traídos de la colección *Localizaciones.*
2. Los archivos generados en un intervalo de tiempo y aquellos en los que no se especificaba tiempo.
3. Los archivos pertenecientes a las búsquedas por mes o por año.

El primer y segundo formato de datos difería en la agrupación de *Nacionalidades* y *Géneros* de los responsables. Y el tercer formato no era más que múltiples documentos con los formatos anteriores, organizados por los meses y años de la búsqueda en cuestión.

Por ello se creó un método que generaba las cabeceras correspondientes al formato del fichero que le era pasado como argumento**.**

5.2.3.2 Exportación de las entidades pertenecientes al filtro seleccionado

Para realizar las exportaciones de entidades que no requerían de los mismos atributos, se tenían que averiguar de antemano los valores de los atributos que se iban a exportar al *CSV.*

Para lograr esto se ejecutó un comando para obtener dichos valores e ir guardando dichos datos en un contenedor para posteriormente enviarlo como respuesta del servicio *REST.* Este contenedor tiene una limitación en cuanto a memoria por lo que se limita el número de entradas de la base de datos a exportar.

La exportación *CSV* se realiza con las palabras libres de caracteres raros y sin el uso de los caracteres usados en la exportación de Málaga, esto provoca mayor legibilidad y facilita la importación de estos datos a otros lugares de su almacenamiento, pero bloquea la posibilidad de poder reimportarlos en la base de datos; aun así, cambiar esto solamente requiere de modificar una línea en el código fuente del exportador.

Por otro lado, el formato de los nombres de los atributos es el resultado de aplicar la profundidad del atributo a su propio nombre, en que cada objeto interno al que haga referencia añade el nombre de este seguido por un punto, un ejemplo sería el atributo *Género* de los *Responsables* cuyo nombre del atributo sería:

Responsables.Género.

Por último, debido a que no se puede añadir profundidad a un *CSV*, las entidades con más de un objeto interno dentro de las listas contienen una entrada en el *CSV* por cada uno de los mismos. En algunos casos hay entidades que requieren de cerca de 300 entradas.

5.2.3 Permitir la eliminación de la base datos

Para poder arreglar cualquier problema o para aligerar el contenido de la base de datos, se implementó la opción de poder eliminar la base de datos. Este servicio se creó bajo el mismo *Blueprint* que el importador de datos ya que comparte la necesidad de ser accedido únicamente por un administrador de la aplicación y porque tiene relevancia en la importación de datos.

Una vez creado el servicio se construyó un método que eliminaba de la base de datos todas las entidades de cada una de las colecciones de la misma.

6

Conclusiones, dificultades y trabajos futuros

6.1 Conclusiones

Se ha desarrollado una plataforma web que permite estructurar un conjunto de datos de entrada sobre delitos proporcionados por la policía nacional, a la vez que realizar consultas básicas de los mismos. Los datos obtenidos se visualizan en mapas de calor de la provincia y del municipio de Málaga. Al permitir la exportación se puede trasladar de forma sencilla los resultados de una búsqueda o mediante la elaboración de resúmenes estadísticos o de la ejecución de modelos se puede ayudar a prevenir delitos que se realicen en la zona.

De igual manera hay que mencionar que el trabajo está pensado para ser una base sobre la que se puedan seguir haciendo ajustes y añadiendo otras funcionales. Esta es una de las grandes razones de intentar mantener todos los datos posibles dentro la base de datos, ya que si se quieren añadir nuevos

parámetros a las búsquedas solo habría que modificar estas y no modificar de ninguna manera el importador o reimportar gran cantidad de datos.

Cabe destacar que al haberle dado forma a los datos sería fácil refactorizar el modelo actual a otros modelos con objetos para representar cada una de las clases o de importarlos o expórtalos por separado.

Por último, hay que mencionar que solamente la visualización de los datos ya constituye una gran mejoría a la situación de la policía de Málaga, y que al haber limpiado los datos solamente el paso de estos por el proyecto realizado ya mejora inherentemente el uso de los mismos en un futuro ya que se encuentran sin caracteres extraños como tabuladores, saltos de líneas o múltiples espacios entre palabras.

6.2 Dificultades encontradas

A lo largo del trabajo se han ido encontrando diversos problemas. Si nos centramos en el apartado de la importación y exportación de datos, los principales problemas encontrados han sido la falta de estandarización de los *datasets*, el retraso en la obtención de los mismos y la poca disponibilidad y mal estado de datos abiertos públicos que existen en el repositorio de open data del Ayuntamiento de Málaga.

6.2.1 *Datasets* de la policía

A lo largo del proyecto la Policía Nacional nos ha proporcionado tres tipos de *datasets*, si bien en todos ellos se mantenían la designación de las cabeceras mediante los caracteres " *[$* ". Los primeros conjuntos de datos que nos fueron entregados contenían la descripción de los delitos de 3 meses, estos estaban separados en 3 ficheros de datos distintos y estaban en codificados en *utf-8-sig.* En ellos venía la información de los responsables e implicados de cada una de las denuncias presentadas en ese periodo.

Por ello, en la estructuración inicial de los datos, se consideraron dos actores: *Implicados* y *Responsables;* los primeros no aparecen en el resto de los ficheros y los segundos aparecen o no aparecen según el fichero de datos.

La distribución de los datos por meses también motivó la necesidad de utilizar hebras, ya que permitían leer por separado cada uno de los ficheros en los que se conocía a priori que, debido a que eran de meses distintos, no habría gran cantidad de delitos que se repitieran.

Cuando se recibieron los siguientes *datasets*, al final de la "Iteración 1" después de la primera reunión, se recibieron en otro formato: un fichero para todas las denuncias de un año y otro con los delitos con *Responsables* asociados. Estos dos ficheros estaban en el mismo formato, pero contenían una diferencia crucial que se tardó en bastante en detectar.

Esta diferencia consistía en que mientras los primeros ficheros tenían unas primeras líneas vacías y los datos empezaban más abajo, en estos ficheros iniciaban directamente en la primera línea. Esta diferencia que de primeras no parece relevante, cobra importancia cuando se descubre que la codificación *utf-8-sig* contiene un número de *bytes* fijos en el comienzo del fichero. Esto provocaba que no se detectaran bien las cabeceras, puesto que en la comparación había caracteres no visibles implicados.

Ambos cambios hicieron que el importador tuviera que adaptarse y que al no disponer de una estandarización en el formato de los *datasets* y al tener la información dividida en dos tipos de ficheros, gran parte de las medidas que se hubieran podido tomar para mejorar la eficiencia de la importación de los datos, tanto en tiempo como en memoria, no fueran factibles.

Finalmente, los últimos conjuntos de datos, las denuncias entre 2010 y 2017, no cambiaron tanto en la estructuración dentro de los propios ficheros; sí que añadían una distinción: no se disponían datos de *Responsables* lo que hizo

de cara a la interfaz que hubiera gran disonancia en las búsquedas que los incluían y en las que no. El gran problema de estos datos fue el tiempo, ya que llegó a mediados de la última iteración, lo que supuso una gran problemática a la hora de probar como respondía la aplicación a varios *gigabytes* de datos.

6.1.2 Datos públicos

Los datos públicos usados en este proyecto fueron los censos del Instituto Nacional de Estadística y el callejero de Málaga del propio ayuntamiento de la ciudad.

Los primeros únicamente se trataron en este trabajo en la fase de investigación, pero se encontró que la API para la obtención de las series del censo por municipios de Málaga venía mal explicada y se tardó más de lo debido en poder obtener los datos necesarios.

Por otro lado, en el tema del callejero se disponía solo en *CSV* por lo que se realizó un módulo auxiliar para procesar ese fichero y otros futuros si es que se encontraban. La problemática de este fichero es que, si bien no presentaba errores, los nombres de las calles como ya se ha mencionado en el apartado *4.2.5* venían de una manera un tanto esperpéntica y un ejemplo sería el ya mencionado ***Camino de los Albuceros***, aparecía como ***CMNO, ALBUCEROS DE LOS***. Los artículos de los nombres de muchas calles venían al final del nombre lo que supuso añadir registros de las calles con ambos nombres.

Por último, no se han encontrado datos que relacionen las calles de Málaga con los barrios de las mismas, por lo que no se pudo realizar esta tarea que hubiera permitido visualizar mejor los puntos calientes de zonas más reducidas en el mapa.

6.3 Trabajos futuros

6.3.1 Posibles mejoras en el procesado de datos

En el apartado del procesado de datos se podrían mejorar los siguientes aspectos si así se requirieran.

6.3.1.1 Reducción en el uso de la memoria

Se podría reducir el uso de la memoria si se requiriera la importación de una mayor cantidad de datos. Para ello se podría añadir una base de datos intermedia para ir introduciendo los datos cuando fuera necesario e ir extrayéndolos después en bloques para actualizarlos. Esta mejora, sin embargo, supondría un alto coste en tiempo.

También se podría intentar adaptar la librería *Pandas* al proyecto, ya que esta librería consume menos memoria al abrir ficheros extremadamente grandes.

Por último, como ha sido mencionada ya en esta memoria, se podrían importar directamente las Denuncias procesadas a la base de datos e ir actualizándolas poco a poco, para al final recalcular las tablas auxiliares. Se desaconseja esta opción por el coste en tiempo necesario.

6.3.1.2 Mejor control de errores

Dado que es una funcionalidad usada por los administradores y en la que no se destruyen datos ya almacenados en la base de datos, no se tiene un control excesivamente férreo de los posibles errores a la hora de importar, por ejemplo, un fichero erróneo.

Si se quisiera prevenir esto se modificaría únicamente el servicio y no sería necesario modificar el módulo del *Importador*.

6.3.2 Nuevas funcionalidades que se podrían añadir en el proyecto

Dado que es un proyecto que se tiene interés en continuar y que se ha desarrollado iterativamente, hay requisitos que se podrían haber introducido o que se podrían recuperar de los rechazados por falta de tiempo.

6.3.2.1 Visualización de los puntos calientes de las calles de Málaga

Este requisito se pidió originalmente, pero debido a que los nombres de las calles de Málaga no contienen formatos comunes, se requeriría de la transformación de estos a un mismo formato. Una idea sobre la que empezar sería intentar cambiar el formato las calles de los datos de la policía por el formato de las calles del ayuntamiento y desde ahí encontrar alguna manera de localizar estas calles con ese formato dentro de un mapa de Málaga.

6.3.2.2 Visualización de los puntos calientes por barrios de Málaga

Este requisito sería bastante fácil de implementar si el ayuntamiento dispusiera de las calles que forman cada barrio. Actualmente se dispone del nombre de la calle, aunque este tiene un formato propio, el distrito policial, el distrito municipal y el código postal de cada uno de los *Hechos* de las *Denuncias* en la aplicación

6.3.2.3 Aumento de los parámetros para realizar las consultas

Siempre se podrían añadir nuevos parámetros a la hora de realizar las búsquedas en la aplicación, esto supondría tener que modificar levemente la interfaz, los servicios *REST* de la aplicación, así como los métodos que generan la consulta a la base de datos.

6.3.2.4 Eliminación parcial de datos de la aplicación

Para realizar eliminaciones parciales de la base de datos, se tendría que crear, en primer lugar, una interfaz para seleccionar qué datos se quieren

eliminar y después se tendrían que recalcular los valores de las colecciones auxiliares de la base de datos.

Bibliografía

[1] **BBC NEWS MUNDO (2018, Noviembre, 19). El ladrón que fue atrapado en Italia gracias a un nuevo algoritmo para predecir delitos inventado en Nápoles.** Disponible: https://www.bbc.com/mundo/noticias-46261759

[2] **F. Rodella (2018, Noviembre, 28). El policía que aprendió a programar para predecir crímenes.** Disponible: https://elpais.com/tecnologia/2018/11/26/actualidad/1543224868_536310.html

[3] **A. García, "Investigación y análisis de crímenes en las ciudades de Chicago y Los Ángeles.", Trabajo de Fin de Máster, Universidad de Málaga, Málaga, MA, España, 2019.**

A. Ruiz, "SIG, CRIMEN Y SEGURIDAD. ANÁLISIS, PREDICCIÓN Y PREVENCIÓN DEL FENÓMENO CRIMINAL", Trabajo de Fin de Máster, Universidad Complutense de Madrid, Madrid, M, España, 2011/2012.

Manual de Mongo 4.2. Disponible: https://docs.mongodb.com/manual/

Datos abiertos Ayto. Málaga. Disponible: **https://datosabiertos.malaga.eu/**

Documentación de SQLite. Disponible: **https://www.sqlite.org/docs.html**

Documentation de Flask. Disponible: **https://flask.palletsprojects.com/en/1.1.x/**

Documentation de PyMongo 3.9.0. Disponible**:**

https://api.mongodb.com/python/current/

Apéndice A

Manual de Usuario

A.1. Acerca del programa

Casandra es una página web cuya finalidad es proporcionar ayudar a los cuerpos de la policía nacional de Málaga a comparar los datos sobre distintos delitos y realizar predicciones conforme a ellos. La página cuenta con un mapa de málaga provincial y otro de la capital, que servirán para representar los datos de las consultas en forma de mapas de calor. Además, también añade otros medios de visualización, como la representación de los datos en un formato estructurado en árbol o en forma de gráficas de barras, disco o circular.

A.2. Guía de uso

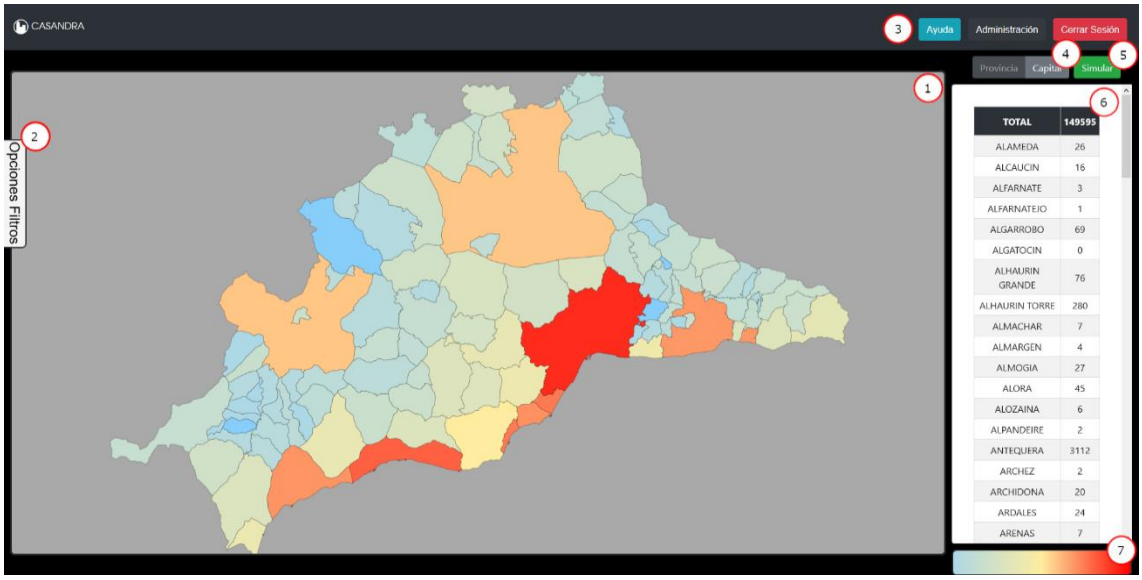


Figura 2 *Interfaz Principal*

1.- Mapa de calor 2.- Pestaña de filtro3.- Opciones de usuario

4.- Opciones de mapa 5.- Simulación 6.- Tabla de datos

7.- Escala

A.2.1. Realizar búsqueda

Para realizar una búsqueda con parámetros se deberá acceder a la pestaña de Opciones de Filtros (Campo 2 [Figura 2.1]).

Dentro de la pestaña de filtros se tendrá acceso a dos opciones de filtrado:

1. Filtrado por Intervalo (Campo 3 [Figura 2.1]).
2. Filtrado por Mes y Año (Campo 4 [Figura 2.1]).

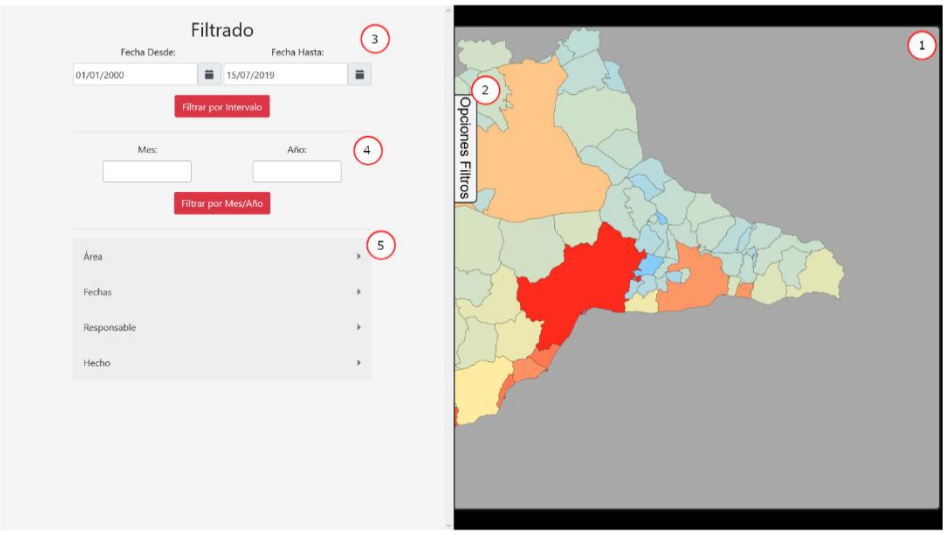


Figura 2.1 *Pestaña de Filtros*

1.- Mapa de calor 2.- Cerrar pestaña 3.- Opciones de Intervalo

4.- Opciones de Mes y Año 5.- Opciones de filtrado

A.2.1.1 Filtrado por Intervalo

Para el primer tipo será necesario aportar una fecha de inicio del intervalo y otra de fin del intervalo (Campos "Fecha Desde" y "Fecha Hasta" respectivamente [Figura 2.1.1]). Al seleccionar entonces la opción de filtrado (botón "Filtrar por Intervalo" [Figura 2.1.1]) el mapa (Campo 1 [Figura 2.1]) se actualizará con los datos relativos a dicho intervalo.

La búsqueda por intervalo se verá en todo momento restringida por estas reglas:

- El Campo "Fecha Desde" no puede ser mayor que el campo "Fecha Hasta"
- El valor de "Fecha Desde" no puede ser menor al 1 de enero de 1950.
- El valor de "Fecha Hasta" no puede sobrepasar la fecha del dispositivo.

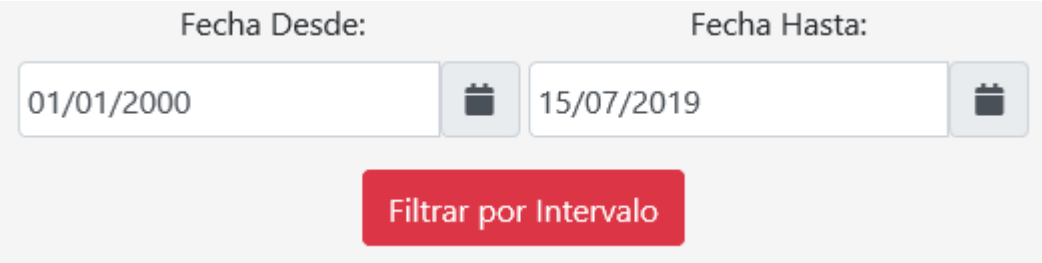


Figura 2.1.1 *Filtrar por Intervalo*

A.2.1.2 Filtrado por Mes y Año

A la hora de filtrar por Mes y Año se deberá suministrar por qué meses (Campo "Mes" [Figura 2.1.2]) y que años (Campo "Año" [Figura 2.1.2]) se deseará realizar el filtro, pueden seleccionar tanta cantidad como se prefiera. Al seleccionar entonces la opción de filtrado (botón "Filtrar por Mes/Año" [Figura 2.1.2]) el mapa (Campo 1 [Figura 2.1]) se actualizará con los datos relativos a dicho intervalo.

Figura 2.1.2 *Filtrar por Mes y Año*

A.2.1.3 Filtros

Es posible añadir algunos filtros a nuestras búsquedas para limitar la cantidad de datos a obtener relativos a ciertos parámetros. Para ver estos campos se debe de pulsar en alguna de las opciones suministradas [Figura 2.1.3].

NOTA: Cuando se incluye algún tipo de filtro, la cabecera de su menú se actualiza para indicar la cantidad de elementos que hay seleccionados (Campos "Área" y "Responsable" [Figura 2.1.3]).

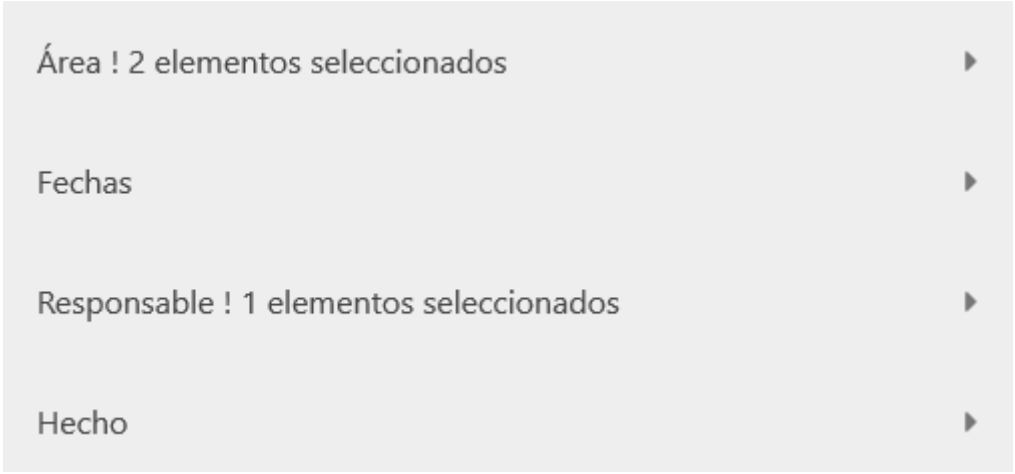


Figura 2.1.3 *Opciones de filtrado*

A.2.1.3.1 Opciones de área

Aquí podrán seleccionarse filtros relativos al lugar dónde se produjeron los hechos [Figura 2.1.3.1.1]. Las opciones comprenden el municipio o distrito donde se realizó la denuncia (Campo "Municipio" [Figura 2.1.3.1.1] o "Distrito" [Figura 2.1.3.1.2]) y el tipo de lugar donde pudo producirse el hecho (Campo "Lugar" [Figura 2.1.3.1.1] y [Figura 2.1.3.1.2]).

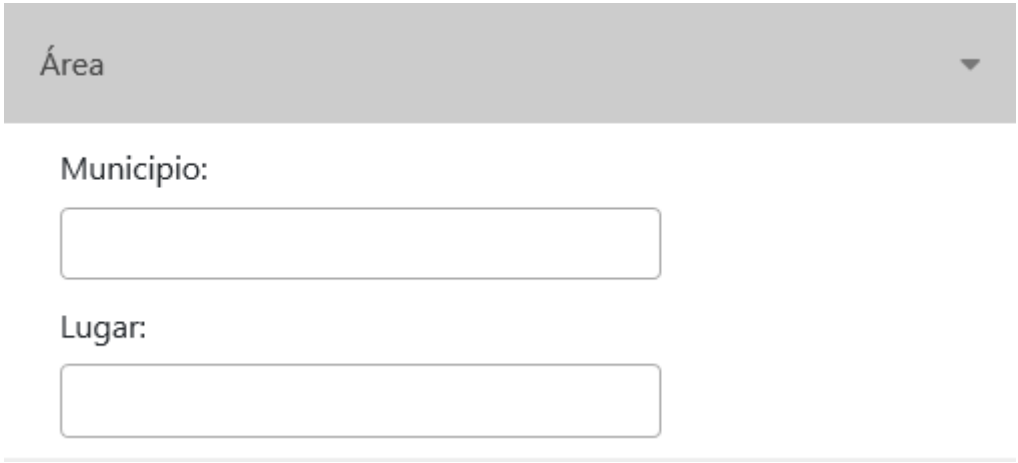


Figura 2.1.3.1.1 *Opciones de área (Provincia)*

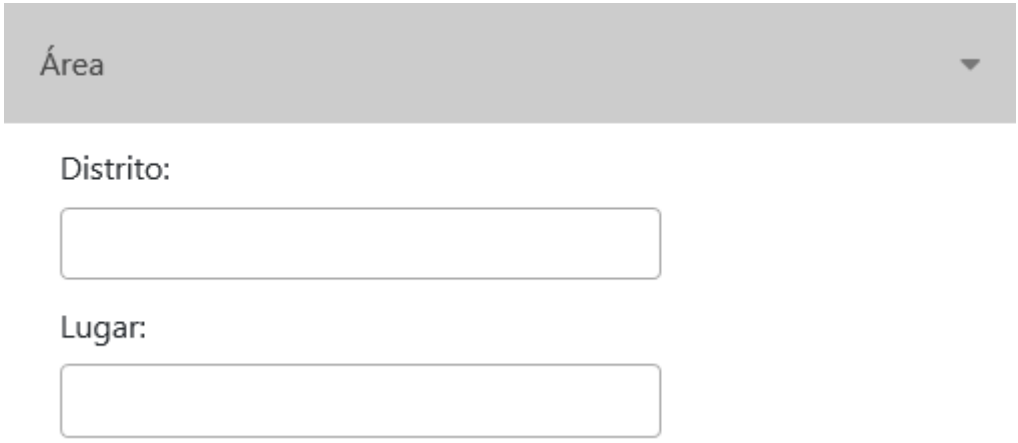


Figura 2.1.3.1.2 *Opciones de área (Capital)*

A.2.1.3.2 Opciones de fecha

Aquí podrán seleccionarse filtros relativos a cuando se produjeron los hechos [Figura 2.1.3.2]. Las opciones comprenden el día de la semana cuando se produjo el hecho (Campo "Día Semana" [Figura 2.1.3.2]), el tramo horario (Campo "Tramo Horario" [Figura 2.1.3.2]) o la posible festividad donde se ocurrió el hecho (Campo "Festividades" [Figura 2.1.3.2]).

Figura 2.1.3.2 *Opciones de Fecha*

A.2.1.3.3 Opciones de responsable

Corresponde a los filtros relativos a quien produjo el hecho [Figura 2.1.3.3]. Incluye el sexo del responsable (Campo "Sexo Responsable" [Figura 2.1.3.3]) y su nacionalidad (Campo "Nacionalidad Responsable" [Figura 2.1.3.3]).

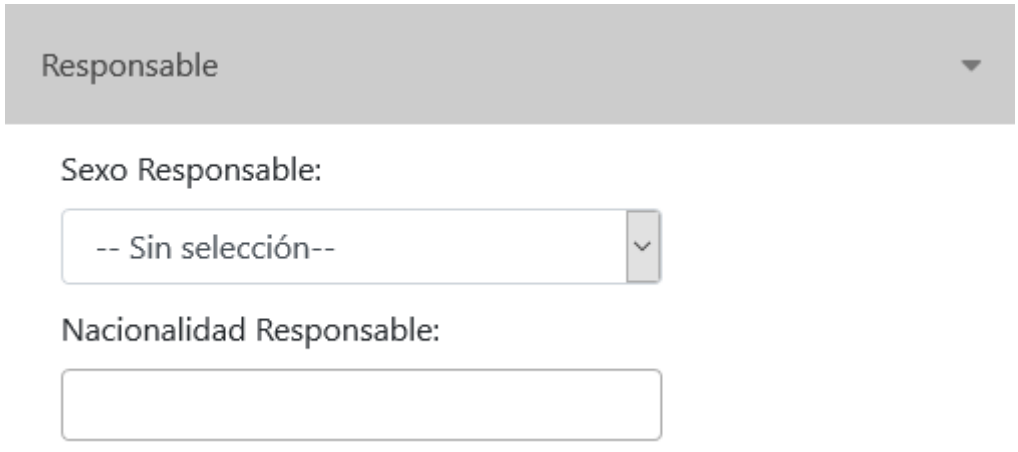


Figura 2.1.3.3 *Opciones de Responsable*

A.2.1.3.4 Opciones de hecho

Corresponde a los filtros relativos al hecho en sí mismo [Figura 2.1.3.4]. Incluye el tipo de hecho (Campo "Hecho" [Figura 2.1.3.4]) y la forma en la que se produjo (Campo "Modus Operandi" [Figura 2.1.3.4]).

Figura 2.1.3.4 *Opciones de Hecho*

A.2.2. Consultar datos y estadísticas

Existen diversas formas de consultar los datos actuales de un mapa de calor. El primero de ellos es mediante la tabla de datos que hay siempre a la derecha del mapa (Campo 6 [Figura 2]). Esta tabla puede ocultarse si el tamaño de la pantalla es demasiado pequeño, en dicho caso podrá alternarse entre el mapa y la propia tabla mediante la pulsación de un solo botón (Botón "Tabla/Mapa" [Figura 2.2.1]).

Si se prefiere, también es posible ver los datos de un área concreta manteniendo el cursor sobre el área en cuestión [Figura 2.2.2].

No obstante, si se prefiere consultar los datos de una forma más exhaustiva, al seleccionar un área del mapa se desplegará una pestaña con una representación en árbol de los datos de la zona seleccionada [Figura 2.2.3]. La

estructura de los datos puede variar dependiendo del tipo de filtro realizado (Intervalo o Mes/Año) [Figura 2.2.4].

De forma adicional, es posible seleccionar más de un área para poder ver sus datos a la vez [Figura 2.2.5]. Para hacer esto se debe seleccionar una zona y luego las siguientes de formas sucesivas. En el caso que no se quiera borrar un registro de selecciones, bastará con pulsar el botón "Borrar Todo" [Figura 2.2.3] [Figura 2.2.4] [Figura 2.2.5].

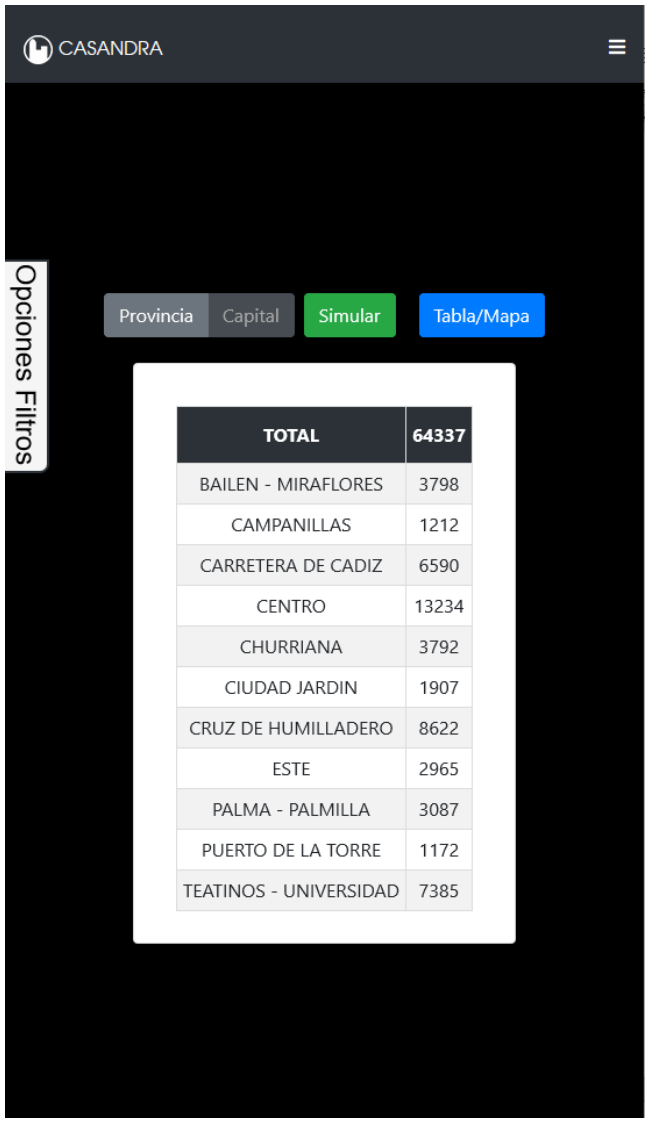


Figura 2.2.1 *Tabla (Pantallas pequeñas)*

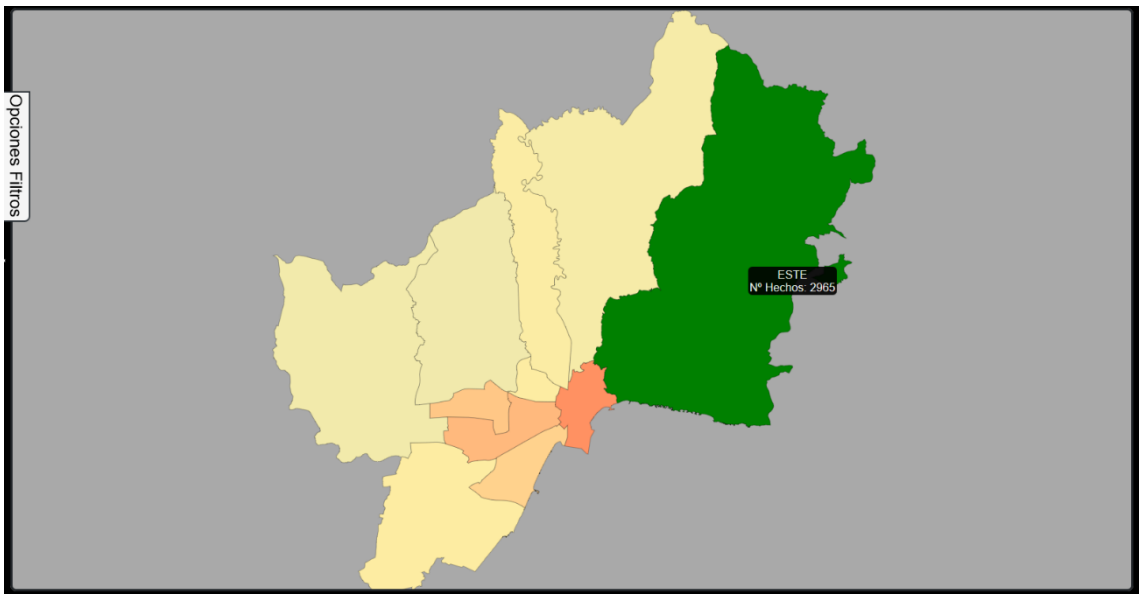


Figura 2.2.2 *Datos de un área determinada*

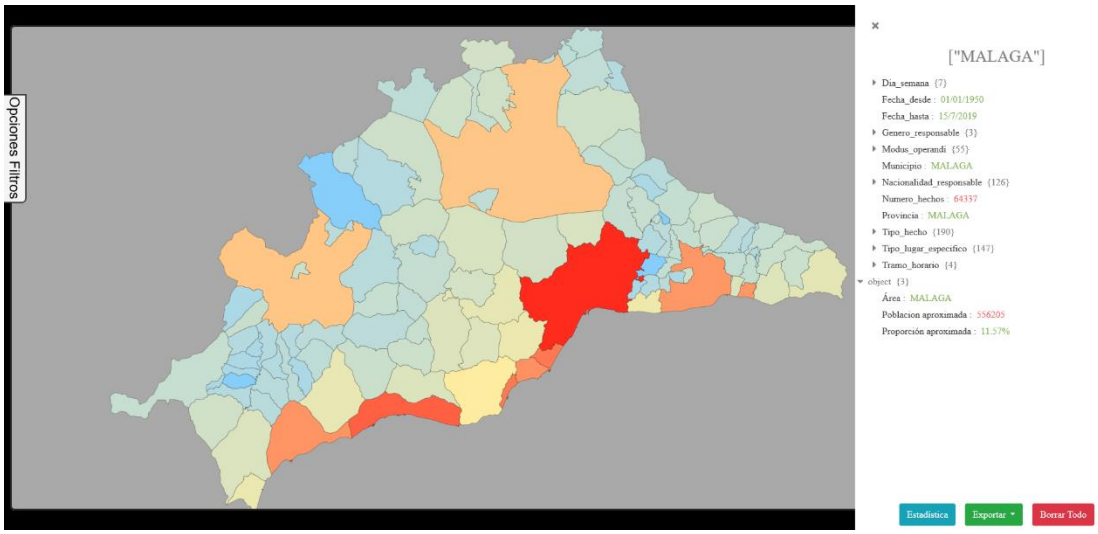


Figura 2.2.3 *Representación en árbol*

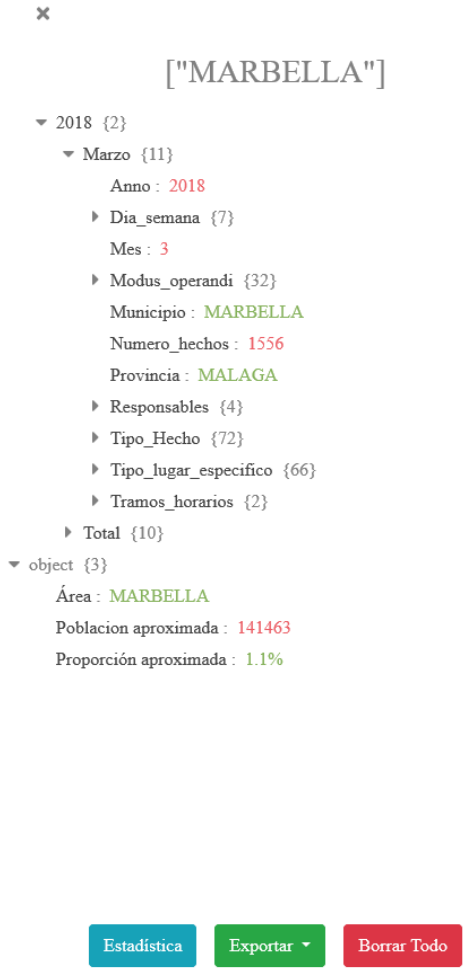


Figura 2.2.4 *Representación alternativa*

Figura 2.2.5 *Representación de dos zonas*

Con los datos que se tengan desplegados en ese momento será posible exportarlos en formato .csv o JSON (Botón "Exportar" [Figura 2.2.3] [Figura 2.2.4] [Figura 2.2.5]), o será posible verlos en forma de gráficas (Botón "Estadísticas" [Figura 2.2.3] [Figura 2.2.4] [Figura 2.2.5]).

A.2.2.1 Estadísticas

El la ventana de estadísticas será posible ver los datos de una o varias zonas en forma de gráficas [Figura 2.2.1.1]. Si se seleccionaron varias áreas se podrán alternar entre estas pulsando los botones a ambos lados de la pantalla

(Campos 4 y 5 [Figura 2.2.1.1]). Es posible que alguna de las gráficas se saturen debido a la gran cantidad de datos en ellas, en ese caso se puede variar el porcentaje mínimo de datos que se puede mostrar mediante el campo 3 [Figura 2.2.1.1].

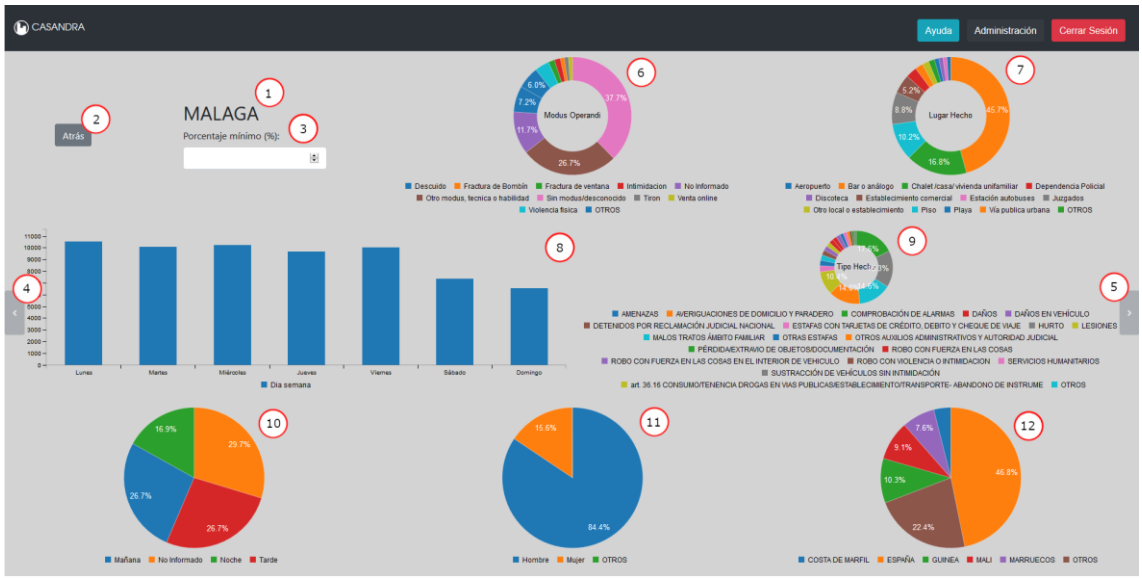


Figura 2.2.1.1 *Estadísticas de una zona*

1.- Zona 2.- Volver atrás 3.- Cambiar porcentaje mínimo

4.- Cambiar a zona previa 5.- Cambiar a zona posterior 6.- Modus Operandi (Disco)

7.- Lugar del hecho (Disco) 8.- Día de la semana (Barras) 9.- Tipo de hecho (Disco)

10.- Tramo horario (Circular) 11.- Sexo responsable (Circular)

12.- Nacionalidad responsable (Circular)

A.2.3. Simulación y predicción

Accediendo a la opción de simular (Campo 5 [Figura 2]) la página web cambiará a la ventana de simulación [Figura 2.3].

Desde esta ventana será posible realizar simulaciones en base a los datos guardados en la aplicación. Para comenzar una simulación solo habrá que pulsar en la pestaña "Opciones Simulación" [Figura 2.3] para abrir el menú de simulación.

Figura 2.3 *Ventana de Simulación*

A.2.3.1 Predicción

Cuando el menú esté en la opción de predicción (Campo 1 [Figura 2.3.1.1]) se mostrarán las opciones para una simulación de tipo "Predicción" [Figura 2.3.1.1].

Las opciones que poseen son las de usar un intervalo de fechas (se aplican las mismas restricciones que en el apartado "*2.1.1 Filtrado por Intervalo*") o un conjunto de festividades (no es posible usar ambas a la vez, y además, usar festividades bloqueará la opción de filtro por "Día de la semana"), un intervalo

de horas (de nuevo, la hora de inicio no puede ser mayor a la hora de fin del intervalo), si se quiere simular en la provincia o en la capital (Se mostrará el campo "Municipio" o "Distrito" respectivamente) y las múltiples opciones de filtrado (Comprenden las mismas opciones que el apartado"*2.1.3 Filtros*").

La simulación se realizará al pulsar el correspondiente botón de simulación (Campo 7 [Figura 2.3.1.1]) respecto únicamente a las opciones seleccionadas en el menú. No obstante, es posible autorellenar el resto de los campos marcando la opción "Autocompletar" (Campo 6 [Figura 2.3.1.1]), al hacerlo se abrirá una pestaña para que se seleccione la prioridad que seguirá el autocompletar [Figura 2.3.1.2], las prioridades van desde 0 (máxima prioridad) a 6 (mínima prioridad) y no pueden repetirse.

Los datos simulados se representarán a partir de dos mapas [Figura 2.3.1.3], el primero correspondiendo a los datos empíricos de la simulación y el segundo a los resultados de la predicción. Igual que con el mapa del apartado "*2.2 Consultar datos y estadísticas*" se podrá ver los porcentajes manteniendo el curso sobre un área del mapa o pulsando en la pestaña "Datos Simulación" para verlos en una estructura de árbol [Figura 2.3.1.4].

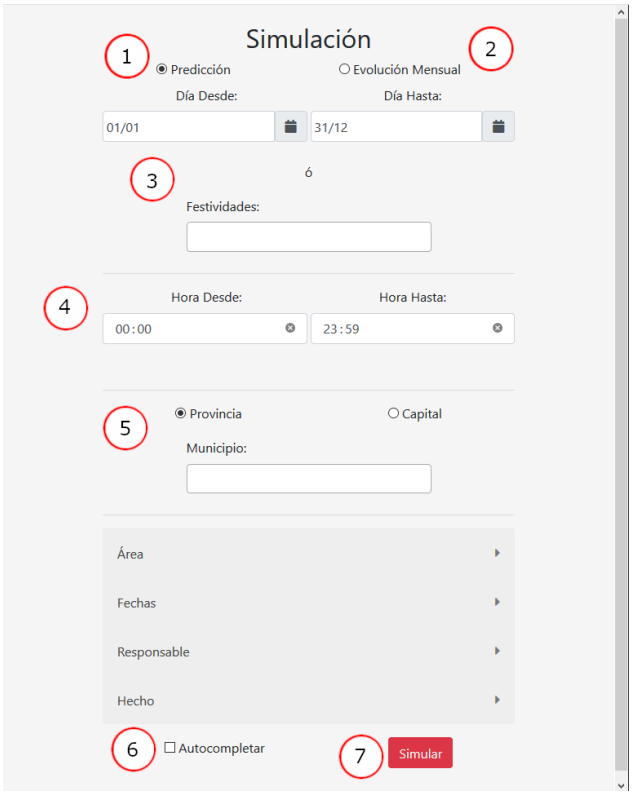


Figura 2.3.1.1 *Menú de Predicción*

1.- Opción de Predicción 2.- Opción de Evolución mensual

3-. Filtros de fechas 4.- Filtros de hora 5.- Filtros de zona

6.- Autocompletar 7.- Iniciar simulación

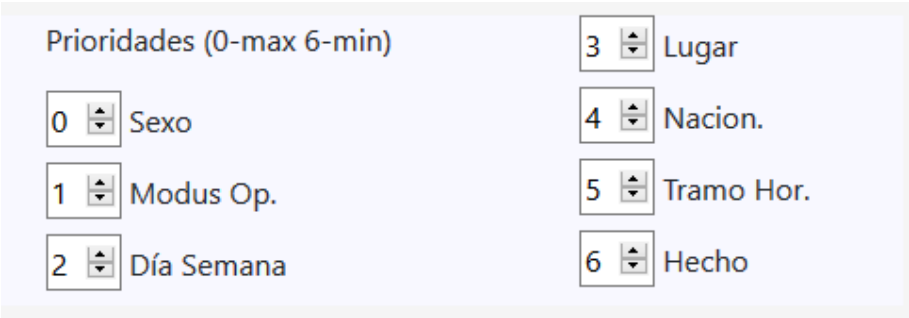


Figura 2.3.1.2 *Prioridades autocompletar*

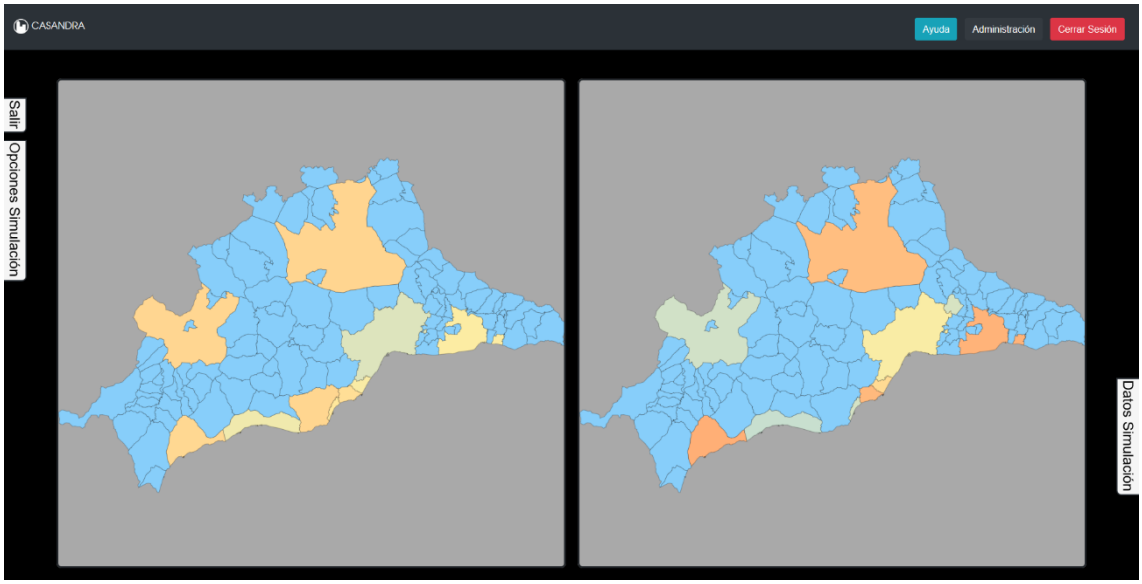


Figura 2.3.1.3 *Datos Empíricos y Simulados (Mapas)*

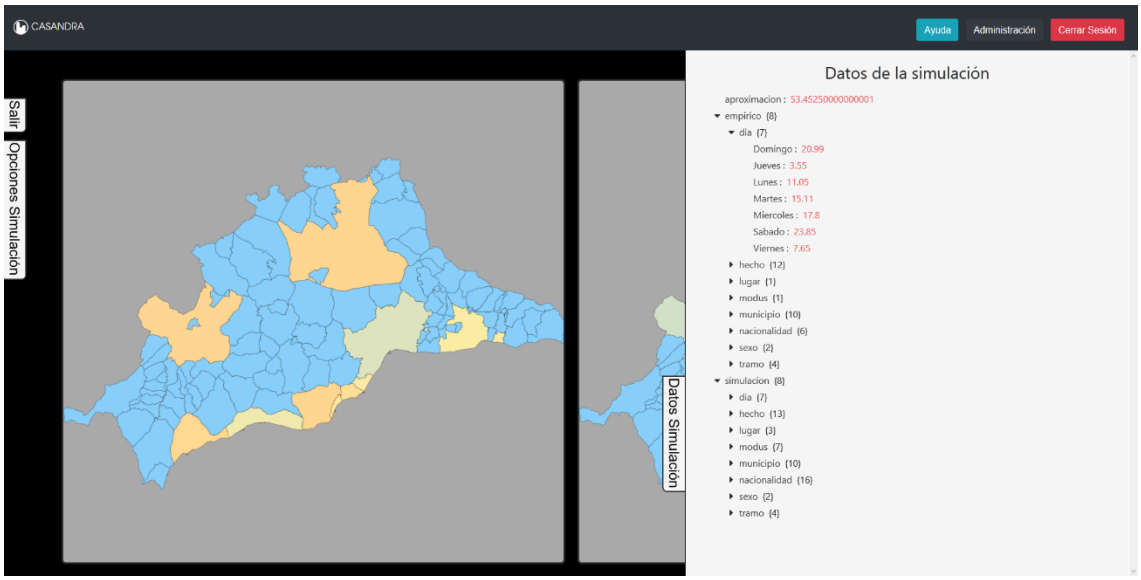


Figura 2.3.1.4 *Datos Empíricos y Simulados (Árbol)*

A.2.3.2. Evolución mensual

Con la opción de evolución mensual será posible ver cómo podrían evolucionar la cantidad de delitos a lo largo de un año o algunos meses. Para acceder a esta opción basta con seleccionar el campo "Evolución Mensual" (Campo 2 [Figura 2.3.1.1]).

El menú de evolución mensual [Figura 2.3.2.1] es similar al de predicción salvo por varios detalles: las fechas se eligen con el día y el mes por separado, solo se puede seleccionar una festividad y no hay opción de intervalo de horas ni de autocompletar.

Al pulsar entonces sobre el botón de "Simular" se cargará un solo mapa con la opción de poder cambiar el mes del cual se hace la representación [Figura 2.3.2.2]. Los datos de cada zona se muestran manteniendo el cursor sobre un área del mapa y en una tabla a la derecha de este. Si la pantalla es muy pequeña, la tabla se oculta en la pestaña "Datos Simulación".

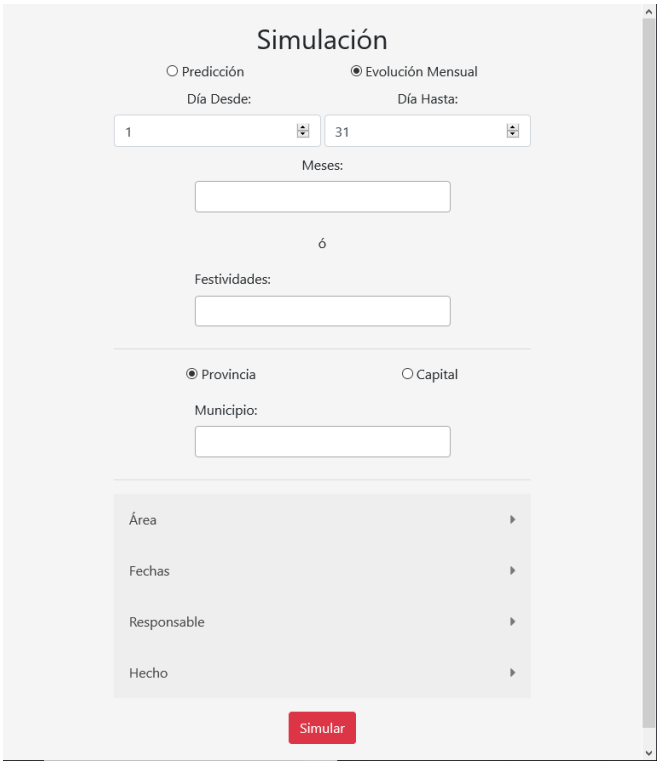


Figura 2.3.2.1 *Menú de Evolución Mensual*

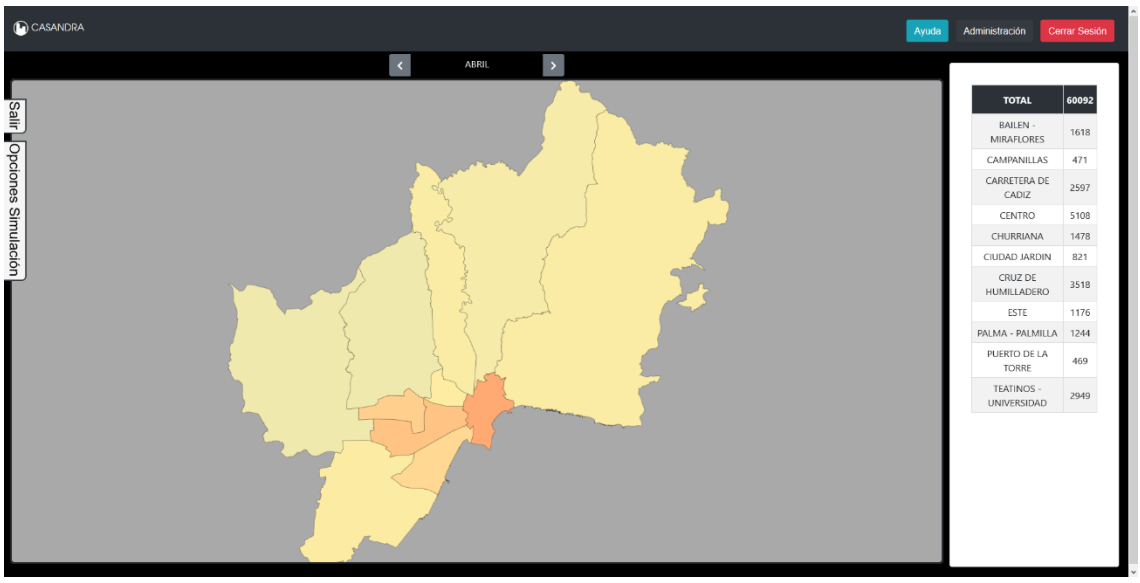


Figura 2.3.2.2 *Mapa de Evolución Mensual*

A.2.4. Opciones de usuario

A.2.4.1. Acceso a la aplicación

Al entrar en la aplicación se requerirá del nombre del usuario y su contraseña para poder acceder. No hay límite de intentos, pero si el usuario no recuerda su contraseña podrá recuperarla a través del enlace "He olvidado mi contraseña". Desde esta pantalla [Figura 2.4.1.1] y a través del nombre de usuario, se le proporcionará una nueva contraseña a la cuenta de correo electrónico asociada, la contraseña es aleatoria y se recomienda cambiarla tan pronto como sea posible.

Nota: Al acceder a la aplicación se pone en constancia que se está de acuerdo con el almacenamiento de *cookies* en su equipo.

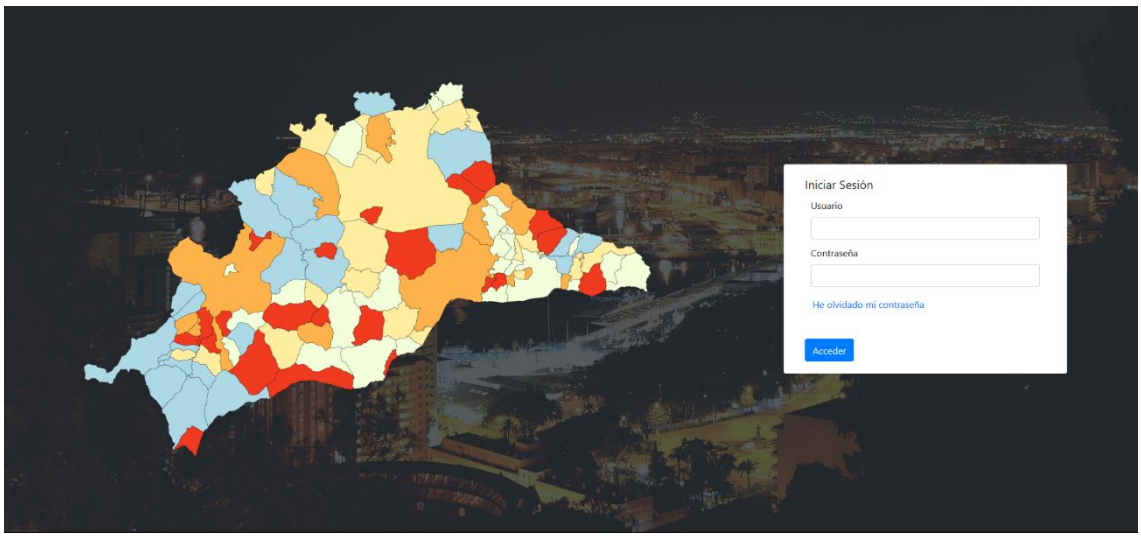


Figura 2.4.1 *Acceso a usuarios*

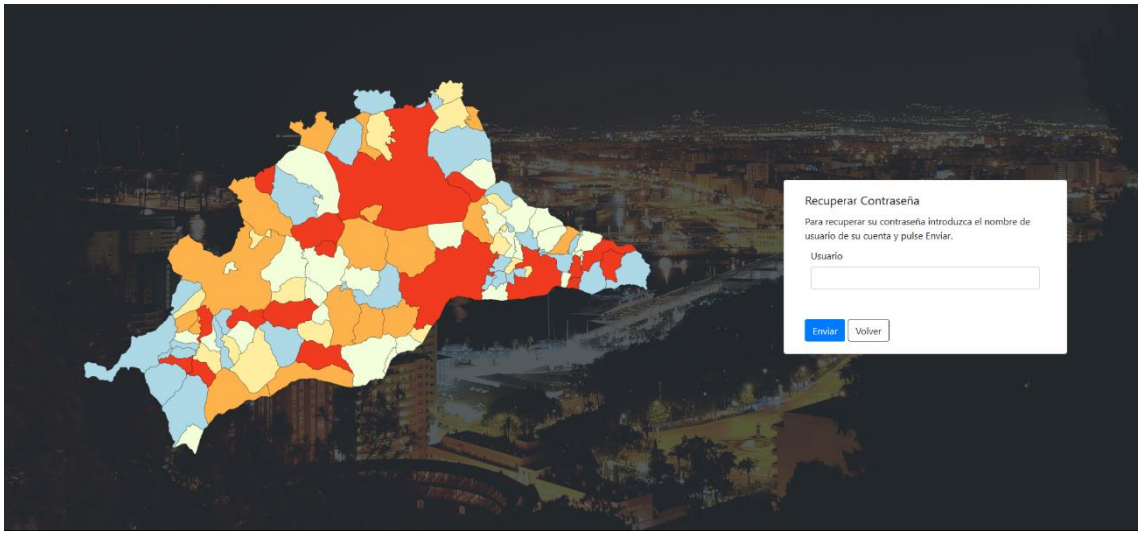


Figura 2.4.1.1 *Recuperar contraseñas*

A.2.4.2. Ayuda

Desde la opción de ayuda, la cual está siempre disponible en la cabecera de la aplicación [Figura 2.4.2.1], se podrá acceder a una serie de opciones para facilitar la experiencia del usuario en la aplicación.

Estas opciones incluyen conocer el contexto de la aplicación (Botón "Acerca de" [Figura 2.4.2.2]), acceder a este mismo manual de usuario (Botón "Manual de Usuario" [Figura 2.4.2.2]) o cambiar la contraseña de la cuenta.

Para cambiar la contraseña bastará con proporcionar la contraseña actual y repetir dos veces la nueva. Si no existen problemas, se notificará del éxito de la operación y ya será posible usar la nueva clave [Figura 2.4.2.3].

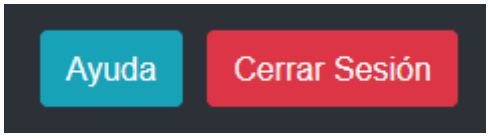


Figura 2.4.2.1 *Botón de Ayuda*

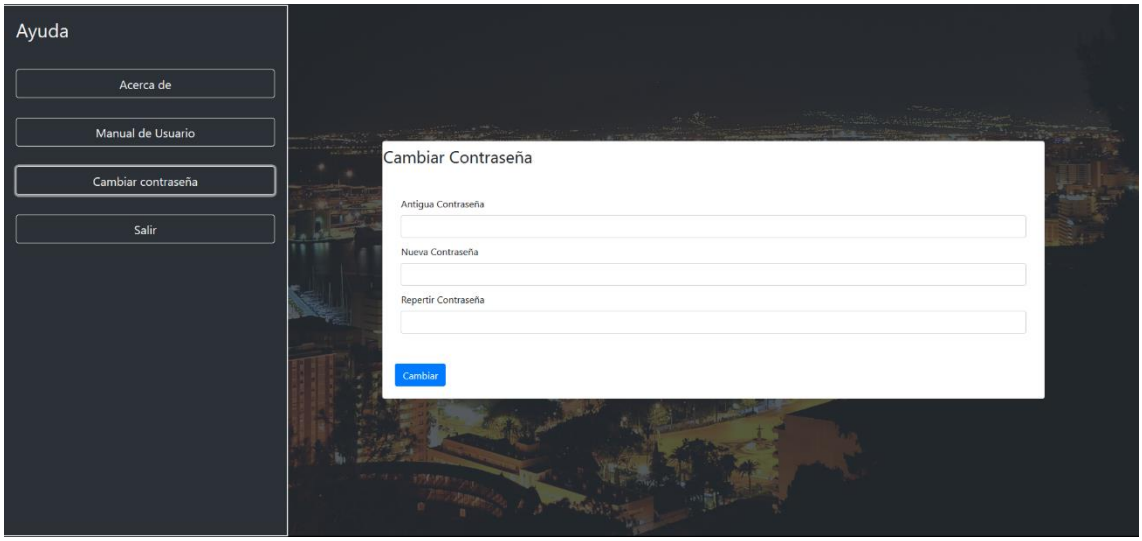


Figura 2.4.2.2 *Ventana de ayuda*

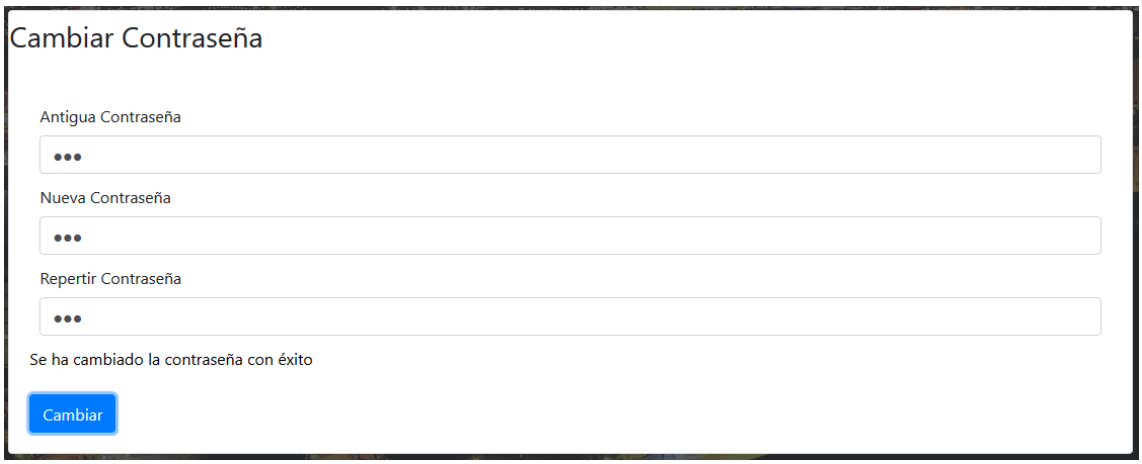


Figura 2.4.2.3 *Confirmación del cambio de contraseña*

A.2.5. Opciones de administrador

Los administradores poseen ciertos privilegios a la hora de gestionar la aplicación. Si un usuario es administrador, se le mostrará el botón "Administración" en la cabecera de la página. Pulsándolo se les dará acceso a las opciones de administrador.

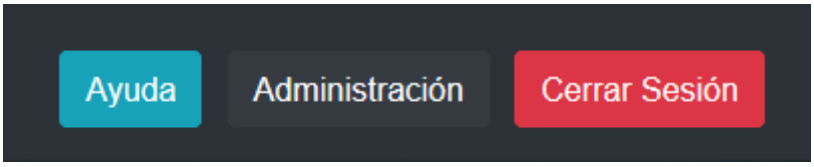


Figura 2.5 *Botón de Administración*

A.2.5.1. Importar datos

Desde esta pantalla el administrador por introducir nuevos datos en la base de datos de la aplicación. Podrá hacerlo examinando los archivos de sus dispositivos o arrastrándolos al área marcado con "Arrastrar aquí". Solo pueden subirse hasta seis archivos a la vez, con un peso máximo de 200 Megabytes.

Estos archivos deberán estar en CSV y deben contener la estructura en la cabecera que se muestra a continuación.

[$AA03 NumActuacion].[Numactuacion]

Y deben contener obligatoriamente el número de la actuación, el código de la plantilla de actuación y la fecha de actuación en el formato *1 de Enero de 2000*.

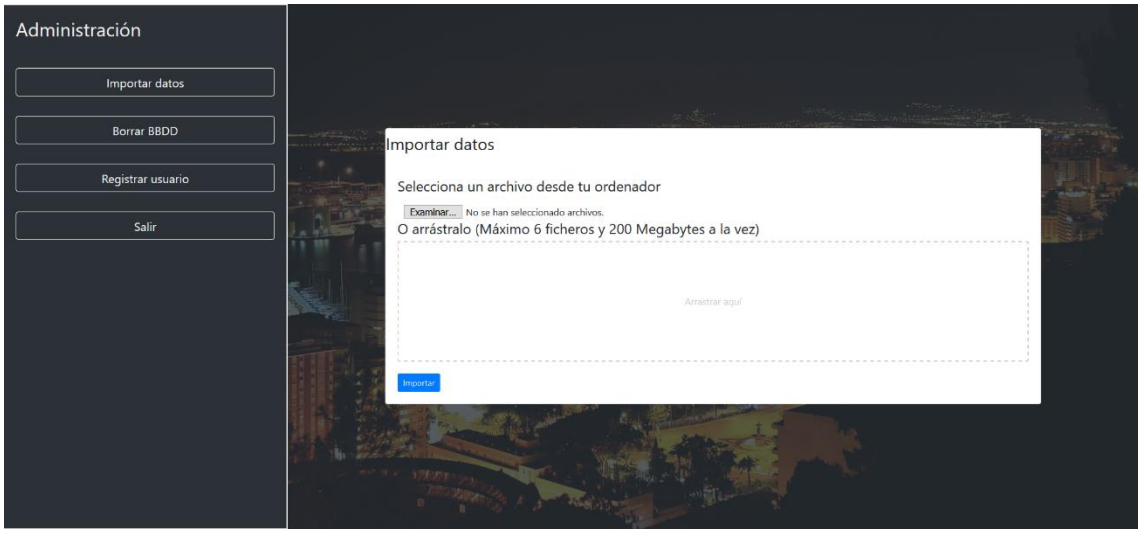


Figura 2.5.1 *Importar datos*

A.2.5.2. Borrar BBDD

Desde este apartado se podrá borrar la totalidad de los datos de la aplicación. Se recomienda hacerlo solo cuando sea necesario ya que no existe la posibilidad de recuperar los datos una vez borrados.

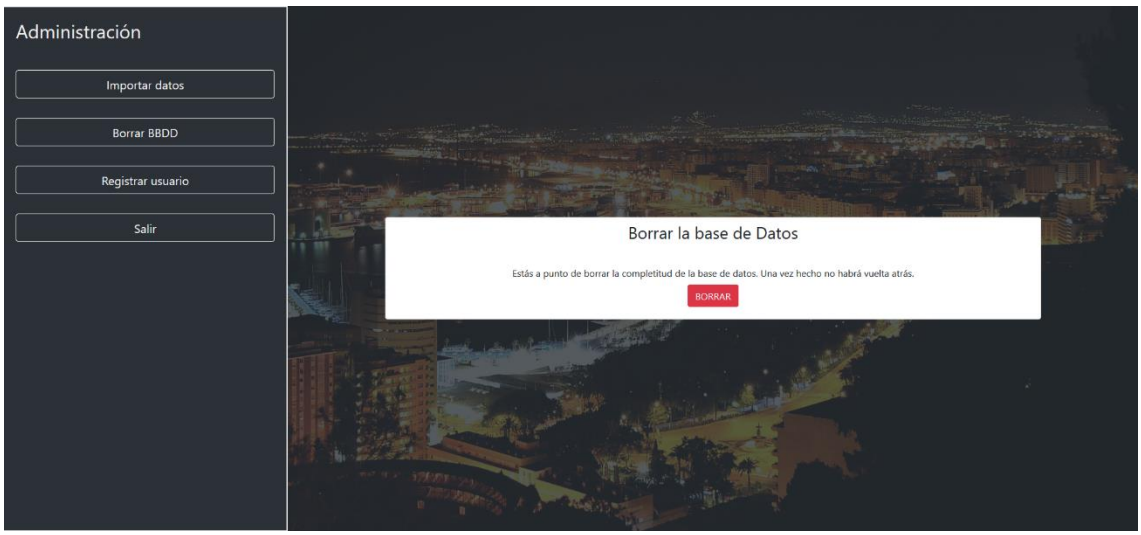


Figura 2.5.2 *Borrar Base de Datos*

A.2.5.3. Registrar usuario

Desde aquí, un administrador podrá registrar a un nuevo usuario. Para ello deberá suministrar un nombre de usuario (que no esté ya registrado), un correo electrónico (se utilizará para poder recuperar una contraseña olvidada), una contraseña (se debe repetir para evitar errores) y si se desea que el nuevo usuario sea administrador o no.

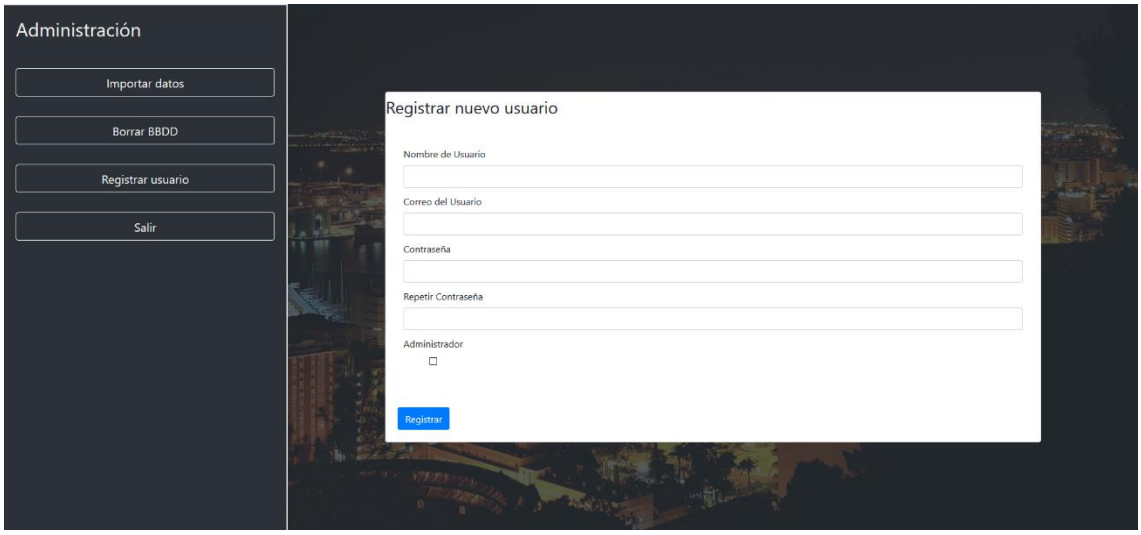


Figura 2.5.3 *Registrar Usuario*

Apéndice B

Manual para el programador

B.1. Introducción:

El siguiente manual para la programación del sistema contiene un análisis detallado del funcionamiento, parámetros y salidas de cada método del servidor. El orden de presentación será el siguiente:

- **Servicios de importación**: servicios para importar ficheros CSV.
- **Lector e importador:** importa datos de ficheros CSV a MongoDB.
- **Módulo auxiliar de importación a bases de datos relacionales:** importa ficheros CSV a bases de datos relacionales.
- **Filtros:** aplicación de filtros a las búsquedas del usuario.
- **Exportación a ficheros:** exportación de filtros aplicados a CSV.
- **Cabeceras:** carga rápida de los filtros aplicables.
- **Usuarios:** manejo y control de usuarios.
- **Seguridad del sistema,** basada en JWT.
- **Agentes:** simulación de futuros eventos.

Los autores de este manual son: Juan Palma Borda y Sergio Gavilán Prieto.

B.2 Servicios del importador

Se dispone únicamente de dos servicios que requieren permisos de administrador, ***cargar_ficheros*** y ***comprobar_lectura***. Ambos están bajo un ***blueprint*** con nombre ***CASANDRA_ficheros*** con URL **/lector/v1**.

- ***cargar_ficheros*: s**ervicio para poder importar datos a la aplicación. No tiene excesivo control de errores para detectar ficheros erróneos. Uno de los motivos por lo que se requiere permiso de administrador.

  Ruta: /lector/v1/leerFichero

  Método: POST

  Procedimiento:

  1. Elimina ficheros antiguos si es que los hubiera.
  2. Copia los nuevos ficheros.
  3. Crea el importador.
  4. Ejecuta el importador con las rutas de los ficheros.
  5. Borra los ficheros del servidor.

  Parámetros en la petición: ficheros a importar.

  Devuelve: fin del lector.

- ***comprobar_lectura*: s**ervicio para poder visualizar el estado del lector.

  Ruta: /lector/v1/tiempo

  Método: GET

  Parámetros en la petición: no tiene.

  Devuelve: lista de enteros con la siguiente estructura [% leído, % comprobado, % importado en MongoDB].

B.2.1. Clase Importador

Clase principal del importador de datos, tiene dos métodos ***importar_archivos*** y ***get_contador***.

Métodos

- ***importar_archivos***: método base del importador de datos, maneja todo el proceso de importación de datos.

  Procedimiento:

  1. Lanza una hebra por cada fichero.
  2. Une los diccionarios si hubiera alguno que no estuviera unificado.
  3. Comprueba que denuncias están la base de datos.
  4. Importa las denuncias y sus tablas auxiliares a MongoDB.

  Argumentos:

  - archivos: lista con las rutas de los archivos.

  Devuelve: no devuelve nada.

- ***get_contador***: método que devuelve el estado del lector.

  Argumentos: no tiene.

  Devuelve: lista de enteros con la siguiente estructura [% leído, % comprobado, % importado en MongoDB].

B.3. Lector e importador de denuncias

Módulo para la lectura, procesado y posterior importación de denuncias de la Policía Nacional de Málaga en formato **CSV** a **MongoDB**.

Se divide en varios módulos:

- **Lector**
- **TablasMongoPolicia**
- **MongoActualizacion**
- **Base de datos SQLite**
- **Festividades**

Y tiene la clase **utils.py** con algunos métodos generales de todos ellos.

B.3.1. Módulo Lector:

Módulo principal del Importador. Pensado para exportar limpiar, procesar e importar denuncias a la colección **Denuncias** de la base de datos de MongoDB. También genera las colecciones auxiliares **Localizaciones** y **Cabeceras**. Para ello hace uso de las clases *LectorDenuncias* y *DiccionarioDenuncias* [**Figura B.3.1.**]

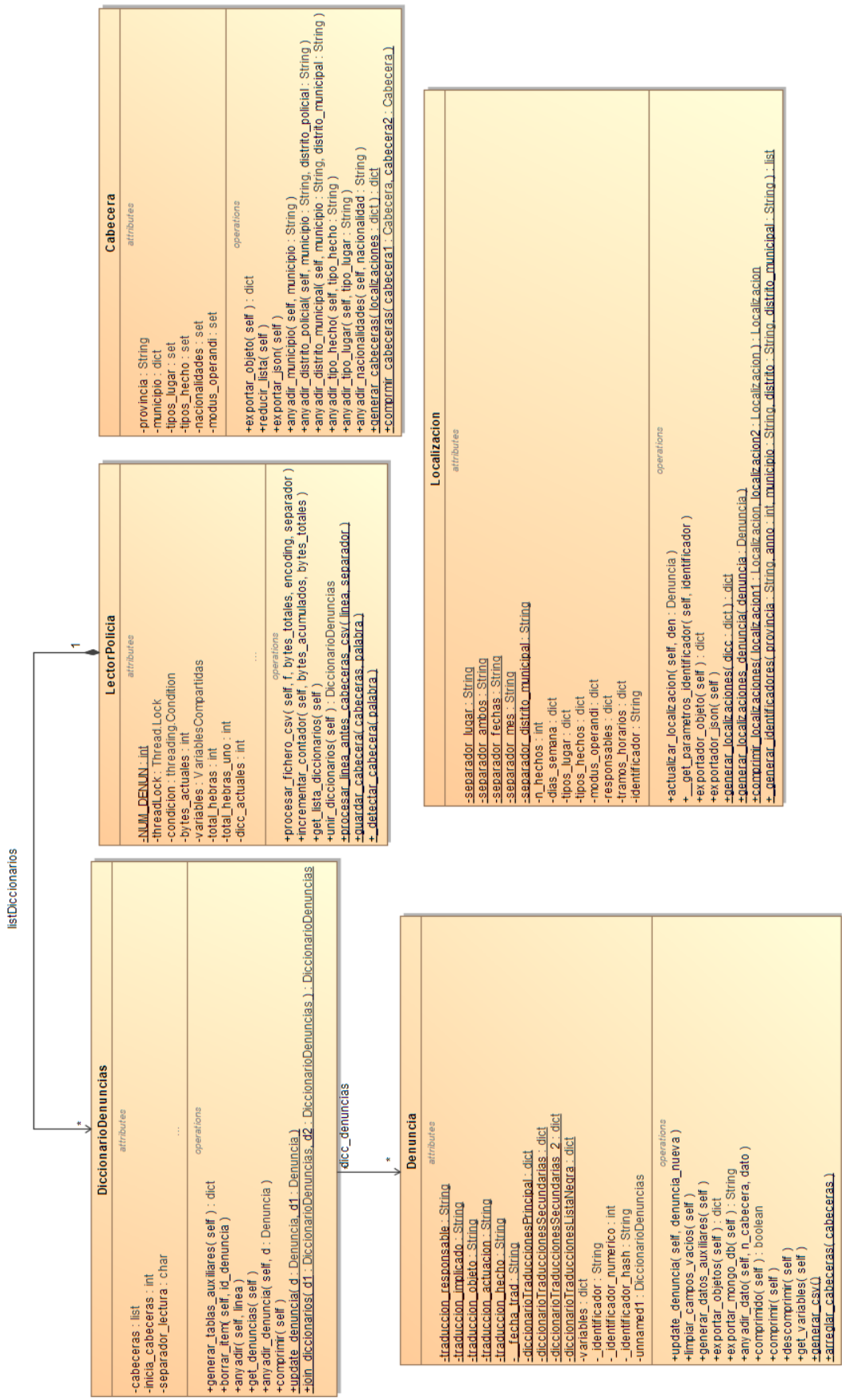


Figura B.3. *Diagrama de clases del Lector de denuncias.*

B.3.1.1. Clase LectorDenuncias

Clase que lee los ficheros, no se ha usado la librería Panda porque da problemas con la estructura de los CSV de la policía. Hereda de la clase *Lector* que es una versión simplificada de este.

Cada lector procesa un único fichero a la vez, pero está pensado para que se leen simultáneamente distintos ficheros en lectores de hebras separadas, ya que irán almacenando y juntando los diccionarios que generan hasta formar un único diccionario.

Métodos

- ***procesar_fichero_csv***: Método base del **LectorDenuncias**, se encarga de leer, encontrar las cabeceras e ir enviando las líneas al **DiccionarioDenuncias** para ir generando *Denuncias*. Tiene sincronización de hebras, ya que está pensado para que haya varios trabajando simultáneamente. Procesa el fichero en un diccionario de denuncias.

  Argumentos:

  - f: ruta del fichero.
  - bytes_totales: número de bytes del total de todos los ficheros.
  - encoding: encoding del fichero, se intenta detectar.
  - separador: separador del csv, sino se intenta detectar.

  Devuelve*:* no devuelve nada*.*

- ***procesar_linea_antes_cabeceras_csv*:** Procesa líneas anteriores a la cabecera del CSV mientras busca las cabeceras.

  Argumentos:

  - línea: línea a procesar.

- separador: separador de la línea.

Devuelve:

- *None* si no es la línea de las cabeceras.
- Tupla (Cabeceras, primera aparición de las mismas).

- ***_detectar_cabecera***: detecta las cabeceras siguiendo un patrón de [$

  Argumentos:

  - palabra: palabra a comprobar.

  Devuelve*:* True si es una palabra perteneciente a las cabeceras, False si no
- ***get_lista_diccionarios*** (property): devuelve la lista de diccionarios compartida por todas las hebras.

  Argumentos: no tiene.

  Devuelve*:* lista de diccionarios de las hebras.
- ***unir_diccionarios***: unifica todos los diccionarios del lector, cada uno de una hebra.

  Argumentos: no tiene.

  Devuelve*:* diccionario final que contiene todas las denuncias.

B.3.1.2. Clase *Lector*

Clase extremadamente simplificada del LectorDenuncias (Uso exclusivo para desarrolladores), que sirve para leer pequeños ficheros CSV, normalmente que contienen datos públicos, para su posterior uso en el propio lector.

Contiene los métodos ***comprobar_encoding*** y ***comprobar_separador***, así como versiones simplificadas de los métodos de la clase que se redefinen en la clase hijo.

Métodos

- ***comprobar_encoding***: intenta detectar el encoding del fichero, detecta *utf8, utf8-sig y cp1252.*

Argumentos:

- f: ruta del fichero

Devuelve*:* codificación del fichero

- ***comprobar_separador***: intenta detectar el separador del fichero, detecta comas y puntos y coma. Solamente lee las 400 primeras líneas

  Argumentos:

  - f: ruta del fichero
  - encoding: codificación del fichero

  Devuelve*:* Separador del CSV

B.3.1.3. Clase DiccionarioDenuncias

Clase que se encarga de almacenar las denuncias, para ello usa un diccionario con la estructura de {identificador: denuncia}. Existe también la clase *Diccionario* que es una versión simplificada de este.

Métodos

- ***anyadir:*** genera una denuncia con la línea seleccionada y la añade al diccionario *dicc_denuncias.*

  Argumentos:

  - línea: línea CSV con la denuncia.

  Devuelve*:* no devuelve nada.

- ***get_denuncias*** (property): Devuelve el diccionario de denuncias.

  Argumentos: no tiene.

  Devuelve*:* dicc_denuncias, formato {id:denuncia}.

- ***borrar_item:*** borra la denuncia del diccionario.

  Argumentos:

  - id_denuncia: identificador de la denuncia a borrar.

  Devuelve*:* no devuelve nada.

- ***generar_tablas_auxiliares***: devuelve un diccionario con las tablas auxiliares ya creadas.

  Argumentos: no tiene.

  Devuelve*:* {"localizaciones": localizaciones, "cabeceras": cabeceras}.
- ***comprimir***: comprime las denuncias del diccionario, convirtiendo su diccionario de variables a una cadena de caracteres.

  Argumentos: no tiene.

  Devuelve*:* no devuelve nada.
- ***join_diccionarios:*** unifica dos diccionarios, actualizando las denuncias correspondientes. Destruye los diccionarios en el proceso.

  Argumentos:

  - d: diccionario 1

  - d2: diccionario 2

  Devuelve*:* Diccionario unificado

B.3.1.4. Clase Diccionario

Versión simplificada del *DiccionarioDenuncias* pensada para ser usada junto a la clase *Lector* para poder leer y más tarde exportar, mediante el módulo de exportación a bases de datos relacionales, información relevante para el lector como, por ejemplo, el archivo *main_calles.sql*.

Está pensado para ser usado solo por desarrolladores y de ninguna manera enlazarlo con la parte de uso de los usuarios.

Contiene los mismos métodos básicos de *anyadir* que el *DiccionarioDenuncias* y también los métodos de exportación a relacionales, que son el ***exportar_mysql*** y el ***exportar_sqlite*** que sirven para exportar el contenido del diccionario a un fichero con la sintaxis del lenguaje de SQL seleccionado.

B.3.2. TablasMongoPolicia

Clases principales del importador, que son a su vez las entidades representadas en las colecciones de *MongoDB*.

B.3.2.1. Clase Denuncia

Clase principal del lector hereda de la clase *OEImplementación*, hereda de esta clase por su similitud a la hora de tratar los datos, ya que unifica todas las variables, salvo los identificadores, de la Denuncia en un diccionario.

Se realiza de esta forma porque la denuncia, salvo unos atributos concretos, no tiene atributos fijos, sino que estos son variables.

También hay que destacar que la mayoría de los métodos están basados en la idea de que hay denuncias que no tienen una única entrada, sino que tienen muchas.

Atributos de clase estáticos

Atributos de clase que evitan multitud de llamadas a la base de datos **Cassandra_Lector.sqlite**, y que contienen las traducciones y listas de palabras secundarias y prohibidas de la base de datos.

- **diccionarioTraduccionesPrincipal**: Diccionario y traducciones de la tabla palabraClavePrincipal.
- **diccionarioTraduccionesSecundarias**: Diccionario y traducciones de la tabla PalabraSecundaria. {(palabraPrincipal, PalabraSecundaria): traduccion}
- **diccionarioTraduccionesSecundarias_2**: Diccionario y traducciones de la tabla PalabraSecundaria. {palabraPrincipal: (PalabraSecundaria, traduccion)}

- **diccionarioTraduccionesListaNegra**: Diccionario con todas las palabras de lista negra asociada a su palabra principal.
- **traduccion_responsable**: Traducción de la palabra responsable.
- **traduccion_implicado**: Traducción de la palabra implicado.
- **traduccion_objeto**: Traducción de la palabra objeto.
- **traduccion_actuacion**: Traducción de la palabra actuación.
- **traduccion_hecho**: Traducción de la palabra hecho.
- **___fecha_trad**: Traducción de la palabra fecha dentro del Hecho.

Traducción de cabeceras

Las cabeceras se traducen antes de empezar a crear cada una de las denuncias; para ello es necesario una limpieza y adaptación de las mismas.

El criterio seguido es identificar los distintos objetos internos de la denuncia a los que hacen referencia cada una de estas cabeceras y adaptar su nombre para una mejor interpretación. El resultado de una traducción es una cadena de caracteres con los nombres de los objetos separados por puntos y en el que el elemento final es el nombre del atributo. Ejemplo:

Traducción de *"Hecho Fecha".Hecho_Fecha_Dia*: **Hecho.Fecha.Dia**

Método

- **arreglar_cabeceras**: estandariza las cabeceras en función a los valores establecidos para los objetos principales en la SQLite.

  Argumentos:

  - cabeceras: cabeceras sin procesar, pero limpia de caracteres extraños

  Devuelve: cabeceras procesadas

Estructuración del diccionario *variables*

La estructuración básica del diccionario *variables* (que es lo que representa a una denuncia) es:

Denuncia: diccionario que contiene las siguientes claves y valores:

- Responsable: lista de diccionarios, en el que cada diccionario representa a un responsable.
- Implicado: lista de diccionarios, en el que cada diccionario representa a un implicado.
- Objeto: lista de diccionarios, en el que cada diccionario representa a un objeto.
- Actuación: diccionario representado el hecho de la denuncia.
- Hecho: diccionario representado el hecho de la denuncia.
- Resto de atributos de la denuncia, que no se clasifican.

Compresión del diccionario *variables*

La clase Denuncia contiene tres métodos relativos a la transformación del diccionario *variables* a una cadena de caracteres para ahorrar espacio de memoria.

- ***comprimido***: devuelve si el diccionario "variables" está comprimido o no.
  Argumentos: No tiene.
  Devuelve: True si el diccionario está comprimido, False si no
- ***comprimir***: convierte el diccionario de variables a una cadena de caracteres con formato JSON, con el método *json.dumps*.
  Argumentos: no tiene.
  Devuelve*:* no devuelve nada.
- ***descomprimir***: convierte el diccionario de variables a un diccionario desde una cadena de caracteres que representa el diccionario en formato JSON.

Argumentos: no tiene.

Devuelve*:* no devuelve nada.

Importación desde CSV

Todo se basa en el método **anyadir_dato**, que añade un atributo al diccionario "*variables*".

- ***anyadir_dato**:* añade un atributo a la denuncia.

  Argumentos:

  - n_cabecera: número de la cabecera.

  - dato: dato en cuestión.

  Devuelve*:* no devuelve nada.

Atributos imprescindibles del CSV

Estos atributos forman el identificador de la denuncia, son tres.

- Numero_Actuacion: Número de la actuación sobre esta denuncia.
- Actuacion.Plantilla_Actuacion.Cod: Código de la plantilla de la Actuación de la denuncia.
- Actuacion.Dia: Fecha en la que se realiza la Actuación (fecha separada por *espacios* y se interpretará el año).

Identificadores

Se distinguen tres identificadores de la denuncia. Uno en formato de cadena de caracteres, resultado de la unión de los tres parámetros anteriores, otro en hexadecimal después de aplicar la función ***sha224*** y otro numérico con la representación numérica del hexadecimal.

Atributos calculados

Se calculan una serie de datos a partir de atributos de la denuncia para facilitar las consultas a la base de datos.

Estos son:

- A partir de la fecha desde del hecho:
  - Día de la semana
  - Día
  - Mes
  - Año
  - Festividad general (Módulo festividades)
  - Festividad concreta (Módulo festividades)
- A partir de la calle y el tipo de vía del hecho:
  - Distrito municipal
- A partir de las horas desde y horas hasta del hecho:
  - Horas desde
  - Horas hasta

Atributos importantes

Conjunto de atributos importantes del hecho, de aquí surgen los atributos que se almacenan en las **Localizaciones** y en las **Cabeceras**. Se disponen *getters* de todos ellos, al invocarlos se descomprime el diccionario si este estuviera comprimido.

Listado:

- Atributos relativos al lugar del hecho: *Provincia*, *Municipio*, ...
- Atributos relativos a la fecha del hecho: *Horas desde*, *Año*, *Mes*, ...

- Objetos internos de la denuncia: *Responsables*, *Implicados*, ...

Simplificación de listas

Una vez leída la denuncia se procede a eliminar elementos no existentes en la misma, como un responsable en el que todos sus atributos son desconocidos. Esto se lleva a cabo con el método ***limpiar_campos_vacios*** que llama a ***___borrar_array*** por cada una de las tres listas del objeto.

- ***limpiar_campos_vacios:*** limpia los arrays de objetos vacíos.

  Argumentos*:* no tiene.

  Devuelve*:* no devuelve nada.

- ***___borrar_array:*** borra el objeto de una lista si no contiene otro valor que no sean diccionarios, valores vacíos o con el valor "No asociado" o "No informado".

  Argumentos*:*

  - lista: lista a la que aplicar el borrado.

  Devuelve*:* no devuelve nada.

Unión de denuncias

Método que junta una denuncia con otra que tenga el mismo identificador, actualiza solamente objetos, implicados y objetos y les pasa antes un filtro para ver si son distintos a los que tiene ya la denuncia, este filtro consiste en tener un atributo distinto a los que ya tiene en su haber los de las listas de la denuncia.

- **update_denuncia**: actualiza una denuncia con nuevos responsables, implicados u objetos, de otra denuncia.

  Argumentos:

  - dicc: denuncia_nueva.

Devuelve*:* no devuelve nada.

Exportación a MongoDB

Hay dos tipos de exportación a MongoDB:

- Directamente a MongoDB (**exportar_objetos**).
- A varios documentos que serán exportados a MongoDB (**exportar_mongo_db**) (*Desarrolladores*).

- ***exportar_objetos****:* exporta el objeto a un diccionario, llama al método superior que devuelve el JSON del diccionario "_variables_" y le añade los tres identificadores internos. Este método descomprime el diccionario si es necesario, y lo comprime al final.

  Argumentos*:* no tiene.

  Devuelve*:* diccionario representativo de la denuncia.
- ***exportar_mongo_db***: genera un JSON en una línea para la posterior exportación a MongoDB.

  Argumentos*:* no tiene.

  Devuelve*:* JSON adaptado a BSON representando la denuncia.

Exportación a CSV

Método

- ***generar_csv:*** método principal a la hora de exportar un grupo de denuncias a CSV, utiliza a su vez los métodos *generar_csv_dict*, *generar_csv_list* y *___generar_csv_dict_list*. Divide el CSV en dos: *parámetros únicos* y *parámetros múltiples*, los primeros se almacenan en *csv_relle* y los segundos en *csv_lista_relle*.

  Argumentos:

  - lista: lista de denuncias, no especialmente grande para no tener problemas con la memoria. De 500 s 1000. Si se provee que el número de

denuncias es excesivamente grande, se recomienda partir el grupo de la denuncia en varios CSV.

- csv_relle: dicc a actualizar, opcional.

- csv_lista_relle: dicc de listas a actualizar, opcional.

- id_i: id actual, opcional.

Devuelve*:* CSV, csv_lista, id_actual.

Métodos

- ***___generar_csv_dict***: genera el CSV de un diccionario interno, método recursivo.

  Argumentos:

  - csv: diccionario con el CSV actual.

  - t: diccionario interno.

  - nombre_variable: nombre del diccionario.

  - id: id actual.

  Devuelve*:* no devuelve nada.

- ***___generar_csv_list***: genera el CSV de una lista interna.

  Argumentos:

  - csv: csv_lista: diccionario con las listas del CSV a exportar.

  - id_d: id actual.

  - lista_param: lista.

  - nombre_variable: nombre de la lista.

  Devuelve*:* no devuelve nada.

- ***___generar_csv_dict_list***: genera el CSV de un diccionario de una lista interna, método recursivo.

  Argumentos*:*

  - csv: csv_lista: diccionario con las listas del CSV a exportar

  - id_d: id actual

- lista_param: lista

- dicc_lista: diccinario de la lista

- nombre_variable: nombre del diccionario

- id_d: id actual

- id_lista: id de la lista

Devuelve*:* no devuelve nada.

B.3.2.2. Clase Localización

Clase auxiliar de la colección denuncia; contiene los datos resumidos de las zonas y fechas a la que representa. La idea es tener los datos precargados y no tener que hacer las consultas más costosas, como las búsquedas generales por un año, mes, municipio, ...

Todas tienen un identificador único que representa qué zona o fecha representa la localización.

Identificador representa:

- **Provincia**: Provincia (**Obligatorio**).
- **Año**: Anno.
- **Mes**: Mes.
- **Municipio**: Municipio.
- **Distrito policial**: Distrito_policial.
- **Distrito municipal**: Distrito_municipal.

Separadores

- Separador del lugar: __
- Separador de la fecha y el lugar: #
- Separador de la fechas: -
- Separador de un mes sin año: @

- Separador del distrito municipal: /

Ejemplos actuales de identificadores:

- **MALAGA**: Provincia de Málaga.
- **@1#MALAGA**: Provincia de Málaga en el mes de enero.
- **2018#MALAGA**: Provincia de Málaga en el año 2018.
- **2018-1#MALAGA**: Provincia de Málaga en el mes de enero del año 2018.
- **MALAGA_MARBELLA**: Provincia de Málaga, municipio de Marbella.
- **MALAGA_MARBELLA_MARBELLA**: Provincia de Málaga, municipio de Marbella en el distrito policial Marbella (Único distrito del municipio de Marbella).
- **MALAGA_MALAGA/Centro**: Provincia de Málaga, municipio de Málaga en el distrito municipal Centro.
- **2018-1#MALAGA_MALAGA/Centro**: Provincia de Málaga, municipio de Málaga, distrito municipal Centro en el mes de enero del año 2018.

Resto de atributos

- **Modus operandi**
- **Número de hechos**
- **Dias de la semana**
- **Tipo de lugar específico**
- **Tipología del hecho**
- **Responsables**: Por género y nacionalidad.
- **Tramos horarios**

Métodos

- ***generar_localizaciones:*** método para crear las localizaciones a partir del diccionario de Denuncias del diccionario.

  Argumentos:

  - dicc: diccionario de denuncias.

  Devuelve*:* Diccionario de localizaciones. {identificador: *Localizacion*}

- ***generar_localizaciones_denuncia:*** genera los identificadores de las localizaciones a crear o modificar de una denuncia.

  Argumentos:

  - denuncia: denuncia de la que generar los identificadores.

  Devuelve*:* Lista de identificadores de la denuncia.

- **___get_parametros_identificador:** método que se llama en el constructor de la localización. Genera los parámetros principales de la localización a través de su identificador.

  Argumentos:

  - identificador: identificador de la localización.

  Devuelve*:* no devuelve nada.

- ***actualizar_localizacion:*** método que sirve para actualizar el resto de los atributos de la localización a partir de una denuncia que le es pasada como parámetro. Actualiza la localización con una denuncia determinada.

Argumentos:

- den: denuncia para actualizar la localización.

Devuelve*:* no devuelve nada.

B.3.2.3. Clase Cabeceras

Clase auxiliar de la colección denuncia, contiene los conjuntos de palabra que se pondrán en los seleccionables de la interfaz. Se crea a partir de las localizaciones y se organiza en provincias.

Atributos que representa

- **Provincia**
- **Municipio**: con sus distritos municipales y policiales asociados.
- **Modus operandi** (*Set*)
- **Tipo de lugar específico** (*Set*)
- **Tipología del hecho** (*Set*)
- **Nacionalidades de los responsables** (*Set*)

Métodos importantes

- ***generar_cabeceras***: genera las cabeceras a partir de un diccionario de Localizaciones.

Argumentos:

- localizaciones: diccionario con las Localizaciones

Devuelve*:* Diccionario de cabeceras. {Provincia: cabecera}

- ***comprimir_cabeceras***: junta dos cabeceras en dos.

Argumentos:

- cabecera1: Cabecera 1

- cabecera2: Cabecera 2

Devuelve*:* Cabecera1 unificada con la cabecera2

B.3.3. MongoActualizacion

Módulo para la comprobación e insertado de valores a la base de datos de MongoDB.

Contiene dos métodos: **cargar_entidades** y **anyadir_denuncias_mongo_DB**. Este primer método comprueba qué valores están ya introducidos en la base de datos; si encuentra que una denuncia ya está en la base de datos, esta se elimina.

El otro método genera las tablas las tablas auxiliares llamando a los métodos pertinentes, las actualiza e introduce las denuncias en la base de datos.

- ***cargar_entidades:*** comprueba qué denuncias no están en la base de datos comprobándolo a partir del Identificador numérico de la *Denuncia*. Las denuncias ya existentes se eliminan según el requisito dado por la policía de Málaga.

  Argumentos:

  - dic: diccionario con las denuncias.

  - variables: contador del lector.

  Devuelve*:* no devuelve nada.

- ***anyadir_denuncias_mongo_DB:*** inserta las denuncias en MongoDB; antes de eso genera las *Localizaciones* y las *Cabeceras* y las modifica o añade a la base de datos.

  Argumentos:

  - diccionario: diccionario con denuncias no existentes en la base de datos

  - variables: contador del lector

  Devuelve*:* no devuelve nada.

B.3.4. Base de datos SQLite

Contiene una base de datos denominada **Cassandra_Lector.sqlite**, con los parámetros necesarios para el correcto funcionamiento del lector.

También están situados los ficheros SQL para generar la base de datos.

B.3.4.1. Conexión

La conexión a las tablas Calle, *PalabraClavePrincipal*, *PalabraListaNegra* y *PalabraSecundaria*, se hace a través de la misma conexión la cual tiene deshabilitado la comprobación del uso de la misma hebra, ya que no se realiza ninguna operación de insertar, actualizar o borrar.

B.3.4.2. Tabla Calle

Tabla generada por el módulo auxiliar de lectura de bases de datos, después del procesado de los datos públicos del ayuntamiento de Málaga. Esta tabla se usa para obtener el distrito municipal de cada denuncia.

Atributos:

```
ID INTEGER NOT NULL,
CODIGO_CALLE INTEGER,
TIPO_VIA TEXT,
NOMBRE_CALLE TEXT,
PRIMER_NUM_TRAMO INTEGER,
ULTIMO_NUM_TRAMO INTEGER,
CODIGO_POSTAL INTEGER,
DISTRITO INTEGER
```

B.3.4.3. Tabla PalabraClavePrincipal

Tabla creada tras el análisis de los datos de denuncias. Esta tabla se usa para la traducción e identificación de palabras principales en el fichero CSV de denuncias.

Atributos:

```
palabra VARCHAR (100) PRIMARY KEY,
traduccion VARCHAR (150) NOT NULL,
prioridadCarga INTEGER UNIQUE NOT NULL
```

B.3.4.4. Tabla PalabraListaNegra

Tabla creada tras el análisis de los datos de denuncias. Esta tabla se usa para la identificación de palabras prohibidas en las cabeceras del fichero CSV con las denuncias. Se procede a eliminarlas después de su identificación.

Atributos:

```
palabra VARCHAR (100),
palabraPrincipal VARCHAR(100) NOT NULL,
prioridadCarga INTEGER NOT NULL
```

B.3.4.5. Tabla PalabraSecundaria

Tabla creada tras el análisis de los datos de denuncias. Esta tabla se usa para la identificación y traducción de palabras secundarias a las principales en las cabeceras del fichero CSV con las denuncias.

Atributos:

```
palabra VARCHAR (100),
palabraPrincipal VARCHAR(100) NOT NULL,
traduccion VARCHAR(150) NOT NULL,
prioridadCarga INTEGER NOT NULL
```

B.3.5. Clases de las entidades de las tablas

Clases que representan las distintas entidades de las tablas de la base de datos SQLite del importador, contienen los métodos para su uso.

B.3.5.1. Clase Calle

Métodos

- ***get_distrito:*** devuelve el distrito dado el nombre de la calle y el tipo de vía.

  Procedimiento:

  1. Comprueba si como nombre de calle esta una zona especialmente conocida que no es una calle, por ejemplo, *AEROPUERTO* y devuelve su distrito asociado si hay coincidencia.
  2. Busca todos los distritos que tengan asociado un tipo de vía con el nombre de calle seleccionado.
  3. Comprueba el número de resultados:
     - 3.1. Si hay un solo distrito -> lo devuelve.
     - 3.2. Si no hay distrito -> vuelta al punto 2 pero quitando la primera palabra del nombre de la calle, ya que muchas calles como **Alameda Principal**, no se llaman así, sino que la primera parte del nombre es realmente la tipología de la vía.
     - 3.3. Si hay más de un distrito -> se busca de nuevo, pero traduciendo y añadiendo la tipología de la vía.
       - 3.3.1. Si hay un resultado -> se devuelve.
       - 3.3.2. Si no hay resultado -> se devuelve distrito desconocido.
       - 3.3.3. Si hay más de uno -> se devuelve uno de ellos.

- ***get_calles:*** devuelve todas las calles de la base de datos.

B.3.5.2. Clase PalabraClavePrincipal

Palabras que representan un objeto dentro de la denuncia, tales como responsable, hecho u objeto.

Métodos

- ***get_palabras*** (método estático): devuelve Todas las palabras ordenadas por PRIORIDAD.

  Argumentos: no tiene.

  Devuelve*:* lista con todas las palabras.
- ***get_traduccion*** (método estático): devuelve la traducción de la palabra que se le pasa como argumento. No tiene control de errores.

  Argumentos*:*

  - pal: palabra a traducir

  Devuelve*:* traducción de la palabra que se le pasa como parámetro.
- ***get_palabras_traduccion*** (método estático): devuelve todas las palabras y sus traducciones de la tabla palabraClavePrincipal ordenadas por prioridad.

  Argumentos: no tiene.

  Devuelve*:* diccionario con todas las palabras con sus traducciones. {palabra: traduccion}

B.3.5.3. Clase PalabraSecundaria

Palabra que representan objetos internos de los objetos de la denuncia, van asociados a la *PalabraClavePrincipal* que representan el objeto al que pertenece dicho objeto interno.

Métodos

- ***get_palabras*** *(*método estático): devuelve todas las palabras secundarias ordenadas por prioridad y asociadas a una palabra principal.

  Argumentos*:*

- principal: palabra principal.

Devuelve*:* Lista con todas las palabras secundarias.

- ***get_traduccion*** (método estático): devuelve la traducción de la palabra y de su principal. No tiene control de errores.

  Argumentos*:*

  - pal: palabra a traducir.

  - pal2: palabra principal de la palabra a traducir.

  Devuelve*:* traducción de la palabra que se le pasa como parámetro que está asociada a determinada palabra principal.

- ***get_palabras_traduccion*** (método estático): devuelve todas las palabras y sus traducciones de la entidad *PalabraSecundaria*.

  Argumentos*:* no tiene.

  Devuelve*:* diccionario con todas las palabras y su principal asociada con sus traducciones. {(palabraPrincicipal, palabra): traduccion}

- ***get_palabras_traduccion*** (método estático): devuelve todas las palabras y sus traducciones de la entidad *PalabraSecundaria*.

  Argumentos*:* no tiene.

  Devuelve*:* diccionario con todas las palabras principales asociada con sus palabras secundarias y a sus traducciones. {*palabraPrinicipal*: (*palabraSecundaria*, traduccion)}

B.3.5.4. Clase PalabraListaNegra

Palabras que serán eliminadas de la cabecera una vez identificada su palabra principal asociada.

Métodos

- ***get_palabras*** (método estático): devuelve todas las palabras de la lista negra ordenadas por prioridad y asociadas a una palabra principal.

  Argumentos*:*

  - principal: palabra principal.

  Devuelve*:* lista con todas las palabras de la lista negra.

- ***get_todas_palabras*** (método estático): devuelve todas las palabras y sus traducciones de la tabla *palabraClavePrincipal* ordenadas por prioridad.

  Argumentos*:* no tiene.

  Devuelve*:* diccionario de listas con las palabras en la lista negra con su palabra principal asociada. {*palabraPrincipal*: [lista de palabras de la lista negra]}

B.3.6. Festividades

Pequeño modulo creado para saber si una fecha con el formato día/mes/año es un día de fiesta en Málaga.

Método

- ***get_festividad:*** método principal del módulo, devuelve la representación de si es un día festivo dado un día, mes y año.

  Argumentos*:*

  - día: entero.

  - mes: entero.

  - anno: entero.

  Devuelve*:*

  - Festividad genérica, día concreto de la festividad.
  - None, None en caso de que no sea una festividad.

Implementación:

```
def get_festividad(dia: int, mes: int, anno: int):
    dia_f, dia_c = get_festividad_unico_dia(dia, mes)
    if not dia_f:
        dia_f, dia_c = devolver_navidad(dia, mes)
        if not dia_f:
            if mes <= 2:
                dia_f, dia_c = devolver_semana_blanca(dia, mes, anno)
                if not dia_f:
                    dia_f, dia_c = devolver_carnaval(dia, mes, anno)
            else:
                dia_f, dia_c = devolver_dia_semana_santa(dia, mes, anno)
                if not dia_f:
                    dia_f, dia_c = devolver_feria_malaga(dia, mes, anno)
    return dia_f, dia_c
```

Ejemplos:

- get_festividad (31,12,2018)

  Return Navidad, Dia_navidad

- get_festividad(26,12,2018)

  Return Navidad, Navidad

- get_festividad(1,5,2018)

  Return Día_del_trabajador, Día_del_trabajador

- get_festividad (15,8,2018)

  Return Feria_malaga, Feria_malaga

- get_festividad (30,8,2018)

  Return None, None

B.3.6.1. Festividades presentes en el sistema

- Navidad (22/12/x al 6/1/x).
  - Nochebuena (24/12/x).
  - Día de Navidad (25/12/x).
  - Día de los Santos Inocentes (28/12/x).
  - Nochevieja (31/12/x).
  - Año Nuevo (1/6/x).
  - Cabalgata de Reyes (5/6/x).

    - Día de Reyes (6/6/x).
- Semana Santa (intervalo calculado con el Domingo de Pascua, algoritmo público).
    - Viernes de Dolores.
    - Sábado de Pasión.
    - Domingo de Ramos.
    - Lunes Santo.
    - Martes Santo.
    - Miércoles Santo.
    - Jueves Santo.
    - Viernes Santo.
    - Sábado Santo.
    - Domingo de Pascua o de Resurrección.
    - Lunes de Pascua.
- Semana Blanca (Intervalo calculado con el uso del Día de Andalucía (28/2/x))
    - Día de Andalucía (28/2/x).
- Carnaval (Intervalo calculado con el uso del Domingo de Pascua, algoritmo público).
- Feria de Málaga
    - (Intervalo aproximado del intervalo de la feria de Málaga, +-2 días de error; se calcula en base al 15/8/x).
- Otras:
    - Día del trabajador (1/5/x).
    - Halloween y Día de los muertos (31/10/x - 1/11/x).
    - San Juan (23/6/x - 24/6/x).
    - Virgen de la Victoria (8/9/x).

- San Valentín (14/2/x).
- Día de la Constitución (6/12/x).
- Día de la Inmaculada (8/12/x).
- Día de la Hispanidad (12/10/x).

B.3.6.2. Añadir más festividades

Cada una de las fiestas señaladas en el punto anterior tiene una clase de *Python* asociada, que calcula si una fecha está dentro del rango al que pertenece la festividad que representa.

Las fiestas dentro de la categoría de *Otras* se cargan de un fichero *festividades.txt* el cual se carga en el sistema usando una gramática definida en *ANTLR4*.

Expresión de la gramática:

```
Nombre_separado_por_guiones_bajos -> dia/mes;
```

Ejemplo: *día del trabajador, que se produce el 1 de mayo*

```
Día_del_trabajador -> 1/5;
```

Así pues, si se quieren señalar más festividades, se podrían introducir en ese fichero o se podría crear una clase Python para representar cada festividad y modificar el método **get_festividad** de la clase *Festividades* para que sea tomada en cuenta.

B.3.7. Clase Utils

Clase con métodos auxiliares para el lector.

- Contiene métodos de limpieza de palabras tales como *limpiar*.
- Traductores tales como *get_distrito* o *traducir_mes*.

Métodos para conseguir información dados unos valores como *get_dia_semana* o *get_tramo_horario*.

B.4. Módulo de importación a bases de datos relacionales desde objetos Python

Módulo complementario a la importación, pensado para ser ejecutado por desarrolladores de la aplicación para añadir más datos a la base de datos interna del lector. Ejemplo datos públicos con información de los barrios de Málaga.

El módulo está pensado para poder crear rápidamente un programa que permita la importación de documentos *CSV* sin estructuras lógicas a bases de datos relacionales tanto *SQLite* como *MySQL*.

También contiene una gramática escrita en ANTLR4 que permite enviar más datos a una base de datos ya creada, de forma que no haya que reimportar todos los datos.

Este módulo encapsula la importación a bases de datos relacionales en forma de árbol, con unas interfaces realizadas de objetos. Para ello el módulo contiene las siguientes clases principales: *BDRelacional*, *Objetoexportable*, *OEImplementación*, *FactoryEscritores*, *Escritores* y otras clases internas representadas en el siguiente diagrama de clases [**Figura B.4**].

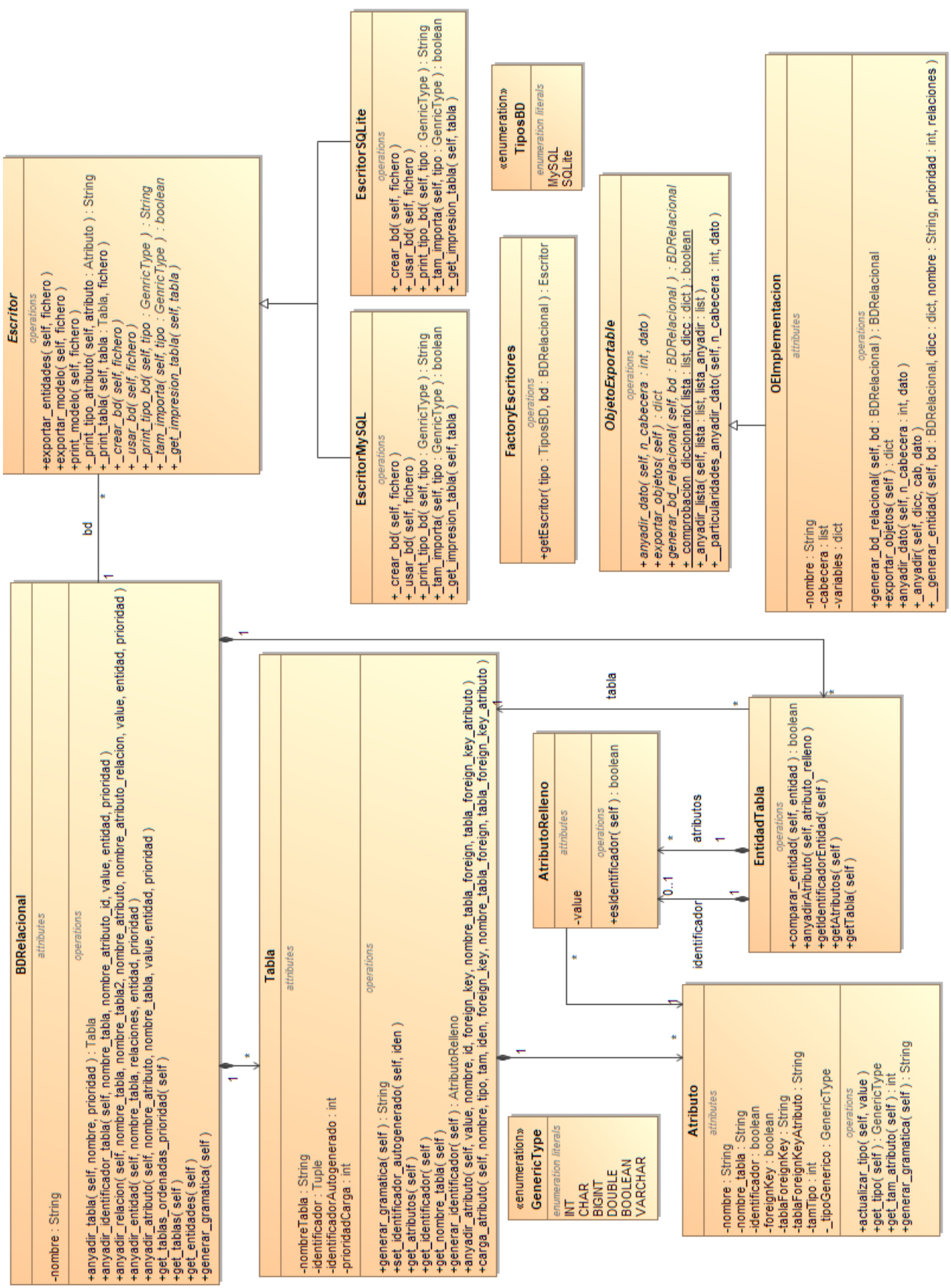


Figura B.4. *Diagrama de clases del módulo de importación a bases de datos relacionales.*

B.4.1. Clase BDRelacional

Clase principal de la librería, encapsula la funcionalidad de una base de datos.

Métodos

- ***anyadir_tabla***: Añade una tabla a la base de datos.

  Argumentos:

  - nombre: nombre de la base de datos

  - prioridad: prioridad de carga a la hora de exportarla

  Devuelve*:* tabla creada.

- ***anyadir_identificador_tabla***: añade un atributo de identificador a la base de datos, creando la tabla sino existe.

  Argumentos:

  - nombre_tabla: tabla a la que hace referencia el atributo

  - nombre_atributo_id: nombre del atributo identificador de la tabla.

  - value: valor del atributo para inferir el tipo de este.

  - entidad: entidad a la que pertenece el atributo.

  - prioridad: prioridad de la tabla si esta lo necesitara.

  Devuelve*:* no devuelve nada.

- ***anyadir_relacion***: añade una relación entre dos tablas de la base de datos, creando cualquiera de las dos tablas si alguna de ellas no existe.

  Argumentos:

  - nombre_tabla: tabla a la que pertenece el atributo de la relación.

- nombre_tabla2: tabla a la que pertenece el atributo al que hace referencia el atributo de la relación.

- nombre_atributo: nombre del atributo en la tabla 'nombreTabla'.

- nombre_atributo_relacion: nombre real del atributo al que se hace referencia.

- value: valor del atributo para inferir el tipo de este.

- entidad: entidad a la que pertenece el atributo.

- prioridad: prioridad de la tabla si esta lo necesitara.

Devuelve*:* no devuelve nada.

- ***anyadir_entidad:*** añade una entidad a la base de datos, creando la tabla de la entidad e introduciendo en esta todos los atributos y relaciones de esta.

  Argumentos:

  - nombre_tabla: nombre de la tabla de la entidad.

  - relaciones: diccionario con las tablas que tiene las relaciones, siempre harán referencia al ID de la tabla.

  - entidad: diccionario con los nombres de los atributos y los valores de los mismos, no se permiten diccionarios, listas u objetos.

  - prioridad: prioridad de carga a la hora de exportarla.

  Devuelve*:* entidad creada.

- ***anyadir_atributo:*** añade un atributo a la tabla.

  Argumentos:

  - nombre_atributo: nombre del atributo.

  - nombre_tabla: nombre de la tabla de la entidad.

- value: valor del atributo para inferir el tipo de este.

- entidad: entidad a la que pertenece el atributo.

- prioridad: prioridad de la tabla si esta lo necesitara.

Devuelve*:* no devuelve nada.

- ***get_tablas*** (propiedad): devuelve las tablas de la base de datos.

  Argumentos: no tiene.

  Devuelve*:* diccionario con las Tablas de la base de datos.

- ***get_entidades*** (propiedad): devuelve las entidades de la base de datos

  Argumentos: no tiene.

  Devuelve*:* lista con las Entidades de la base de datos.

- ***get_tablas_ordenadas_prioridad*** (propiedad): devuelve las tablas de la base de datos ordenadas por mayor prioridad.

  Argumentos: no tiene.

  Devuelve*:* lista con las Tablas de la base de datos ordenadas por prioridad.

- ***generar_gramatica:*** genera un documento escrito siguiendo una gramática escrita en *ANTLR4* que permite la recarga de la base de datos en otro momento.

  Argumentos: no tiene.

  Devuelve*:* no devuelve nada.

B.4.2. Clases de la estructura interna

- Tabla: representa una tabla de una base de datos relacional.

- EntidadTabla: representa una entidad de una tabla de la base de datos relacional.
- Atributo: representa un atributo de una tabla de la base de datos relacional.
- AtributoRelleno: representa un atributo relleno de una entidad de la base de datos relacional.
- Generictype: representa el tipo de un atributo de la base de datos relacional. Los tipos que representa son:

  VARCHAR, INT, BIGINT, DOUBLE, BOOLEAN y CHAR.

Carga de más datos

Para cargar exportar más datos que sean de la misma base de datos que alguna otra exportada anteriormente, la base de datos puede generar un fichero con una gramática generada en ANTLR4, que con el método *generar_gramatica* se guarda en la carpeta *documentosGramaticas* con el nombre de *nombreBD.data*, que posteriormente se puede cargar con el método *cargar_bd.*

- ***cargar_bd:*** carga un objeto Base de datos relacional a partir de la gramática definida en este mismo proyecto.

  Argumentos:

  - fichero: dirección del fichero que contiene con la gramática.

  Devuelve*:* BDRelacional cargada del fichero.

B.4.3. ObjetoExportable

Clase abstracta con los métodos necesarios para la exportación a bases de datos relacionales

Métodos

- ***anyadir_dato***: añade el dato al objeto.

  Argumentos:

  - n_cabecera: numero de la cabecera a la que hace referencia

  - dato: dato al que hace referencia

  Devuelve*:* no devuelve nada.

- ***exportar_objetos:*** exporta el objeto en un diccionario con una estructura de Json.

  Argumentos: no tiene.

  Devuelve*:* Diccionario con formato Json del objeto en cuestion.

- ***generar_bd_relacional:*** genera las tablas y entidades del objeto en la base de datos.

  Argumentos:

  - bd: base de datos a la que añadir el objeto, default *None*.

  Devuelve*:* BDRelacional con el objeto insertado.

B.4.4. OEImplementacion

Implementación de un *ObjetoExportable* de manera básica, estructura de árbol sin seguir ninguna la lógica a la hora de agrupar los objetos.

Clase usada para las importaciones de datos públicos, al poder adaptarse a cualquier *CSV*.

B.4.5. FactoryEscritores

Factory para generar Escritores a partir de una BDRelacional y de un enumerado en la clase TiposBD, que contiene los valores (MySQL y SQLite).

Método

- ***getEscritor***: devuelve un escritor para la base de datos seleccionada.

  Argumentos:

  - tipo: tipo de la base de datos (Enum TiposBD)
  - bd: objeto base de datos sobre el que se va a hacer la escritura.

  Devuelve*:* objeto Escritor creado para esa base de datos.

B.4.6. Escritor

Objeto abstracto para la escritura en el lenguaje de una base de datos relacionales, su implementación está en las clases *Escritor*(*tipoBD*)

Métodos

- ***exportar_entidades***: Exporta las entidades de la base de datos en el lenguaje relacional seleccionado al fichero.

  Argumentos:

  - fichero: fichero sobre el que se va a escribir.

  Devuelve*:* no devuelve nada.

- ***print_modelo***: escribe el modelo de la base de datos el fichero abierto que recibe como parámetro.

  Argumentos:

  - fichero: fichero abierto (`with open (fichero, 'w') as f:`), el valor por defecto es System.out.

  Devuelve*:* no devuelve nada.

- ***exportar_modelo***: exporta el modelo de la base de datos al fichero recibido como parámetro.

Argumentos:

- fichero: dirección del fichero donde se va a escribir el modelo.

Devuelve*:* no devuelve nada*.*

B.5. Módulo para los filtros

Módulo encargado de filtrar datos de acuerdo con unas especificaciones concretas.

B.5.1 Archivos:

- Servicios mapa.
- Filtros middle.
- Filtros fin.
- Filtros útil.

B.5.2 Servicios mapa:

Contiene los servicios REST para hacer consultas de datos aplicando filtros específicos.

Métodos

- ***ver_zona***: Servicio que genera información con la zona detallada donde se está aplicando.

  Método: GET.

  Ruta: /verZona.

  Permisos necesarios: sesión iniciada .

  Parámetros de la petición

  - Provincia : provincia para la búsqueda.
  - Municipio : municipio para la búsqueda.
  - Distrito_municipal : distrito municipal para la búsqueda.
  - Distrito_policial : distrito policial para la búsqueda.
  - resp_sexo : género del responsable.
  - dias_semana : días de la semana.
  - resp_nacionalidad : nacionalidad o nacionalidades del responsable.

- tramo_horario : tramo o tramos horarios.

- tipo_lugar : tipos de lugar. Ejemplo: Farmacia, Vía urbana.

- tipo_hecho: tipos de hecho.

- modus_op : modus operandi.

- mes : mes del que se desea visualizar los datos.

- año : año del que se desea visualizar los datos.

- Festividades: festividades. Ejemplo: carnaval, Navidad, Feria de Málaga.

- fecha_desde: fecha desde la que aplicar el filtro.

- fecha_hasta: fecha hasta la que aplicar el filtro.

Devuelve: si los datos de entrada son correctos, devuelve estado de HTTP 200 junto con el resultado del filtro. En caso contrario, devuelve estado HTTP 400.

- ***carga_mapa:*** devuelve el número de hechos del año actual en la zona especificada y sus subzonas.

  Método: GET.

  Ruta: /cargaMapa.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

  - Provincia: provincia dónde aplicar el filtro.

  - Municipio: municipio dónde aplicar el filtro.

  Devuelve: si los datos son correctos, devuelve estado HTTP 200 junto con el resultado del filtro. En caso contrario, estado HTTP 400.

- ***busqueda_intervalo:*** servicio para filtrado en intervalo, desde una fecha Desde a una fecha Hasta.

  Método: GET.

  Ruta: /fechaInter.

Permisos necesarios: sesión iniciada.

Parámetros de la petición:

- Valor: variable de tipo Integer para informar al servidor sobre dónde realizar la búsqueda, en la provincia o el municipio de Málaga.

- Provincia : provincia a la que aplicar el filtro.

- Municipio : municipio al que aplicar el filtro.

- Distrito_municipal : distrito municipal al que aplicar el filtro.

- Distrito_policial : distrito policial al que aplicar el filtro.

- resp_sexo : género del responsable.

- dias_semana : días de la semana.

- resp_nacionalidad : nacionalidad o nacionalidades del responsable.

- tramo_horario : tramo o tramos horarios.

- tipo_lugar : tipos de lugar. Ejemplo: Farmacia, Vía urbana.

- tipo_hecho: tipos de hecho.

- modus_op : modus operandi.

- Festividades : festividades. Ejemplo: Feria de Málaga, Carnaval, Navidad.

- fecha_desde : fecha desde la que aplicar el filtro.

- fecha_hasta : fecha hasta la que aplicar el filtro.

Devuelve: si los datos son correctos, estado HTTP 200 junto con el resultado del filtro. En caso contrario, estado HTTP 400.

- ***busqueda_dia_mes_anno:*** servicio para obtener el número de delitos con meses y años específicos.

  Método: GET

  Ruta: /fechaDMA.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

- Valor: variable de tipo Integer para informar al servidor sobre dónde realizar la búsqueda, en la provincia o el municipio de Málaga.

- Provincia: provincia a la que aplicar el filtro.

- Municipio: municipio al que aplicar el filtro.

- Distrito_municipal : distrito municipal al que aplicar el filtro.

- Distrito_policial : distrito policial al que aplicar el filtro.

- resp_sexo : género del responsable.

- dias_semana : días de la semana.

- mes: mes sobre el que aplicar el filtro.

- año: año sobre el que aplicar el filtro.

- resp_nacionalidad: nacionalidad o nacionalidades del responsable.

- tramo_horario : tramo o tramos horarios.

- tipo_lugar : tipos de lugar. Ejemplo: Farmacia, Vía urbana.

- tipo_hecho: tipos de hecho.

- modus_op: modus operandi.

- Festividades: festividades. Ejemplo: Feria de Málaga, Carnaval, Navidad.

Devuelve: si los datos son correctos, estado HTTP 200 junto con el resultado del filtro. En caso contrario, estado 400.

B.5.3 Filtros middle

Contiene los métodos intermediarios entre los servicios REST y los que conectan la base de datos. Redireccionan a los métodos apropiados según los datos recibidos en los servicios, pues algunos servicios soportan varios formatos de entrada.

Métodos

- ***denuncia_fecha_intervalo:*** método para búsqueda de resultados en un intervalo de fechas.

  Parámetros:

  - Valor: variable de tipo Integer para informar al servidor sobre dónde realizar la búsqueda, en la provincia o el municipio de Málaga.

  - Provincia : provincia a la que aplicar el filtro.

  - Municipio : municipio al que aplicar el filtro.

  - Distrito_municipal : distrito municipal al que aplicar el filtro.

  - Distrito_policial : distrito policial al que aplicar el filtro.

  - resp_sexo : género del responsable.

  - dias_semana : días de la semana.

  - resp_nacionalidad : nacionalidad o nacionalidades del responsable.

  - tramo_horario : tramo o tramos horarios.

  - tipo_lugar : tipos de lugar. Ejemplo: Farmacia, Vía urbana.

  - tipo_hecho : tipos de hecho.

  - modus_op : modus operandi.

  - Festividades : festividades. Ejemplo: Feria de Málaga, Carnaval, Navidad.

  - fecha_desde : fecha desde la que aplicar el filtro.

  - fecha_hasta : fecha hasta la que aplicar el filtro.

  Devuelve: número de coincidencias en la base de datos

- ***denuncia_dma***: método para búsqueda de resultados en un Año y/o mes específicos

  Parámetros:

  - valor: valor para indicar si se debe buscar sobre municipios o sobre distritos

- mes: mes sobre el que buscar.

- anyo: año sobre el que buscar.

- provincia: provincia.

- municipio: municipio.

- distrito_policial: distritos policiales.

- distrito_municipal: distritos municipales.

- tipo_lugar: tipos de lugar. Ejemplo: Farmacia, Vía urbana.

- tipo_hecho: tipos de hecho.

- modus_op: modus operandi.

- resp_sexo: género del responsable

- resp_nacionalidad: nacionalidad o nacionalidades de los responsables.

- festividades: festividades. Ejemplo: Carnaval, Feria de Málaga, Navidad.

- tramo_horario: tramo o tramos horarios.

- dia_semana: días de la semana.

Devuelve: número de coincidencias en la base de datos.

- ***localizacion_dma***: método para búsqueda de resultados en un Año y/o mes específicos y con consultas simples, de solo un parámetro auxiliar de búsqueda.

  Parámetros:

  - mes: mes sobre el que buscar.

  - anyo: año sobre el que buscar.

  - provincia: provincia.

  - municipio: lista de municipios.

  - distrito_policial: distritos policiales.

  - distrito_municipal: distritos municipales.

- objeto: parámetro a buscar.

- nombre_objeto: nombre del parámetro a buscar.

- valor: valor para indicar si se debe buscar sobre municipios o sobre distritos.

Devuelve: número de coincidencias en la base de datos.

- ***carga_inicial_mapa***: devuelve el número de hechos por municipio, provincia o distrito en el año especificado, o, en su defecto, en el último año recogido en la base de datos.

  Parámetros:

  - provincia: provincial.

  - municipio: muncicipio.

  Devuelve: lista de diccionarios.

- ***ver_zona_middle***: devuelve las zonas pasadas como parámetro en una lista de JSON, este método no acepta fechas.

  Parámetros

  - provincia: provincia.

  - municipio: municipios.

  - distrito_policial: distritos policiales.

  - distrito_municipal: distritos municipales.

  - tipos_lugar: tipos de ligar. Ejemplo: Farmacia, Vía urbana.

  - tipos_hecho: tipos de hecho.

  - modus_op: modus operandi.

  - resp_sexo: género del responsable

  - resp_nacionalidad: nacionalidad o nacionalidades de los responsables.

  - festividades: festividades. Ejemplo: Carnaval, Feria de Málaga, Navidad.

- tramo_horario: tramo o tramos horarios.

- dias_semana: días de la semana.

Devuelve: número de coincidencias en la base de datos.

- ***ver_zona_middle_intervalos***: devuelve las zonas pasadas como parámetro en una lista de JSON, este método acepta intervalos de fechas.

  Parámetros:

  - Provincia : provincia a la que aplicar el filtro.

  - Municipio : municipio al que aplicar el filtro.

  - Distrito_municipal : distrito municipal al que aplicar el filtro.

  - Distrito_policial : distrito policial al que aplicar el filtro.

  - resp_sexo : género del responsable.

  - dias_semana : días de la semana.

  - resp_nacionalidad : nacionalidad o nacionalidades del responsable.

  - tramo_horario : tramo o tramos horarios.

  - tipo_lugar : tipos de lugar. Ejemplo: Farmacia, Vía urbana.

  - tipo_hecho : tipos de hecho.

  - modus_op : modus operandi.

  - Festividades : festividades. Ejemplo: Feria de Málaga, Carnaval, Navidad.

  - fecha_desde : fecha desde la que aplicar el filtro.

  - fecha_hasta : fecha hasta la que aplicar el filtro.

  Devuelve: número de coincidencias en la base de datos.

- ***ver_zona_middle_dma***: devuelve las zonas pasadas como parámetro en una lista de JSON, este método acepta listas de meses y de años.

Parámetros

- mes: mes sobre el que buscar.

- anyo: año sobre el que buscar.

- provincia: provincia.

- municipio: municipio.

- distrito_policial: distritos policiales.

- distrito_municipal: distritos municipales.

- tipo_lugar: tipos de lugar. Ejemplo: Farmacia, Vía urbana.

- tipo_hecho: tipos de hecho.

- modus_op: modus operandi.

- resp_sexo: género del responsable

- resp_nacionalidad: nacionalidad o nacionalidades de los responsables.

- festividades: festividades. Ejemplo: Carnaval, Feria de Málaga, Navidad.

- tramo_horario: tramo o tramos horarios.

- dia_semana: días de la semana.

Devuelve: número de coincidencias en la base de datos

B.5.4 Filtros fin

Se encarga de hacer los datos legibles y operables para la base de datos.

Métodos

- ***generar_diccionario_parametros***: genera el diccionario genérico sobre el que se construirán las consultas a la base de datos.

  Parámetros:

  - provincia: Provincia

  - municipio: Municipios - Lista

- distrito_policial: Distritos policiales - Lista
- distrito_municipal: Distritos municipales - Lista
- tipo_lugar: Tipo lugar - Lista
- tipo_hecho: Tipo hecho - Lista
- modus_op: Modus operandi - Lista
- resp_sexo: Sexo del responsable
- resp_nacionalidad: Nacionalidades de los responsables - Lista
- festividades: Festividades - Lista
- tramo_horario: Tramo horario - Lista
- dia_semana: Dia de la semana - Lista

Devuelve: tupla de diccionario y valor

- ***rellenar_intervalos_diccionario:*** genera las consultas ***$or*** para buscar sobre intervalos de fechas.

  Parámetros:

  - diccionario: diccionario con las consultas ya definidas
  - fecha_hasta: fecha hasta la que se quiere buscar
  - fecha_desde: fecha desde la que se quiere buscar

  Devuelve: tupla de diccionario y valor

- ***_dma_ver_zona_meses_anyos_localizaciones:*** genera el JSON para las consultas sobre día/mes/año que sean simples, accede a la colección de localizaciones

  Parámetros: no recibe parámetros.

  Devuelve: no devuelve nada.

- ***___ver__zona__dma__no__simple***: lanza hebras para generar el JSON de las búsquedas de ver zona que no sean simples. Solo tiene activos un total de hebras igual a la capacidad del procesador menos 1.

  Parámetros: no recibe parámetros.

  Devuelve: JSON con los valores

- ***__threads__dma__denuncia***: método protegido para usar las hebras.

  Parámetros: no recibe parámetros.

  Devuelve: diccionario.

- ***__generar__query__intervalo__ver__zona***: método que genera una consulta sobre la colección denuncia y genera el JSON de la zona en cuestión.

  Parámetros: no recibe parámetros.

  Devuelve: JSON.

- ***__generar__querys__ver__zona***: método que separa las consultas en simples (colección localizaciones) y complejas (colección denuncias), está pensado para consultas sin fechas.

  Parámetros: no recibe parámetros.

  Devuelve: no devuelve nada.

- ***__generar__query__dma__ver__zona***: método que separa las consultas en simples (colección localizaciones) y complejas (colección denuncias), está pensado para consultas con meses y con años.

  Parámetros: no recibe parámetros.

  Devuelve: no devuelve nada.

- ***__ver__zona__middle__denuncia***: método que genera una consulta sobre la colección denuncia y genera el JSON de la zona en cuestión. Pensado para ser sin fechas de ningún tipo.

  Parámetros: no recibe parámetros.

  Devuelve: no devuelve nada.

- ***__localizacion__adapt__query***: adapta la consulta para poder buscar sobre la tabla localizaciones, poniendo el parámetro a agrupar y la suma en cuestión.

  Parámetros: no recibe parámetros.

  Devuelve: no devuelve anda.

- ***__denuncia__dma__adapt__query***: método que genera el diccionario de la consulta y lo adapta para consultas día/mes/año, para después hacer la consulta al método *denuncia_dma_create_n_make_query*.

  Parámetros: no recibe parámetros.

  Devuelve: la consulta realizada.

B.5.5 Filtros útil

Métodos

- ***denuncia__fecha__intervalo***: genera el diccionario de búsqueda para buscar en la colección denuncias.

  Parámetros:

  - dic: diccionario con los parámetros de la consulta.

  Devuelve: lista con los filtros.

- ***__es__simple***: devuelve True si es una consulta simple

  Parámetros:

  - nombres: nombre de los parámetros.

  - params: lista de parámetros.

  Devuelve: True si es simple, False si no.

- ***__generar__identificador__localizaciones***: genera el identificador para buscar sobre localizaciones.

  Parámetros:

  - provincia: provincia a buscar.

  - municipio: municipio a buscar.

  - anyo: año a buscar.

  - mes: mes a buscar.

  - distrito_policial: distrito policial a buscar.

  - distrito_municipal: distrito municipal a buscar.

  Devuelve: identificador para buscar en la base de datos "localizaciones".

Notas

Existen métodos no incluidos en este documento que realizan funciones básicas usadas a nivel profundo de la aplicación. Algunas de estas funciones incluyen búsqueda en diccionarios, copias específicas de listas y más.

B.6. Exportador a CSV

El módulo de exportación a *CSV* contiene los métodos y servicios para exportar a este formato las estadísticas generadas en la aplicación y de las entidades que han producido estas estadísticas.

B.6.1. Servicios de exportación

- ***exportar_estadistica:*** servicio que genera el CSV dado un JSON generado por el *verZona*.

  Ruta: /exportador/v1/exportJson.

  Método: GET.

  Parámetros en la petición:

  - frase: cadena de caracteres, descripción de la búsqueda realizada para generar el JSON.

  - json: JSON a exportar

  Devuelve: fichero CSV con los datos del JSON

- ***exportar_entidades:*** Servicio para exportar entidades de las denuncias de la base de datos nuevamente a ficheros CSV. No soporta más de 200.000 entidades.

  Ruta: /exportador/v1/exportEntidades

  Método: POST.

  Parámetros en la petición:

  - Ficheros a importar.

  Devuelve*:* documento CSV que contienen las entidades seleccionadas para la exportación.

B.6.2 Métodos para realizar la exportación

Métodos

- ***exportar_json***: convierte uno o varios JSON a CSV.

  Argumentos:

  - dicc: diccionario con los JSON.

  - param: frase descriptiva.

  - numero_json: número de JSON en el diccionario.

  Devuelve*:* CSV.

- ***exportar_denuncias:*** *p*ara generar el diccionario para la consulta, se usa el método generar_diccionario_parametros, el método generar_dict_denuncia y el método rellenar_intervalos_diccionario.

  Argumentos:

  - diccionario: diccionario para la consulta

  Devuelve*:* CSV

B.7. Cabeceras

Módulo encargado de buscar las cabeceras, es decir, todos los tipos de filtros posibles aplicables a un delito.

B.7.1. Archivos:

- Servicios.
- Cabeceras middle.

B.7.2. Servicios mapa

Contiene los servicios REST para obtener todas las cabeceras existentes en la base de datos.

Métodos

- ***get_cabeceras*:** Es el primer servicio llamado al iniciar la aplicación tras el inicio de sesión correcto.

  Se entenderá por cabecera todo aquello que el usuario pueda seleccionar como un filtro, por ejemplo, tipos de hecho, tipos de lugar, modus operandi, municipios, etc.

  Método: GET.

  Ruta: /inicio.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

  - Provincia: provincia. Su valor por defecto es Málaga.

  Devuelve: si los datos son correctos: estado HTTP 200 junto con el resultado del Filtro. En caso contrario, estado HTTP 400.

B.7.3. Cabeceras middle

Conecta con la base de datos, obteniendo todos los posibles filtros.

Método

- ***obtener_cabeceras***: devuelve las cabeceras de la provincia.

  Parámetros de la petición:

  - Provincia: provincia sobre la que buscar.

  Devuelve: diccionario con todas las cabeceras.

B.8. Módulo de usuarios

Módulo encargado de gestionar a los usuarios de CASANDRA. Cada usuario tiene cinco atributos:

- ID: valor autogenerado y único.
- Nombre de usuario: valor único.
- Email.
- Contraseña.
- Rol: sus posibles valores son *ROLE_USER* o *ROLE_ADMIN*.

El nombre de usuario y la contraseña se encuentran encriptados en la base de datos, y cada vez que se desean comprobar los datos, estos se encriptan y se comparan la encriptación.

B.8.1. Archivos:

- usuarios**.**
- usuarios_middle**.**

B.8.2. Usuarios

Contiene los servicios para permitir la comunicación entre *front* y *back-end*.

Métodos

- ***login*:** realiza el login del usuario.

  Método: POST.

  Ruta: /Login.

  Permisos necesarios: ninguno.

  Parámetros de la petición:

  - Nombre de usuario.
  - Contraseña del usuario.

Devuelve: si los datos introducidos son correctos, devuelve estado HTTP 200 junto con el token de tipo JWT de autenticación. En caso contrario, devuelve estado HTTP 400 junto con el mensaje de error correspondiente.

- ***check_admin:*** comprueba si el usuario es administrador o no.

  Método: POST.

  Ruta: /checkAdmin.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

  - ID del usuario.

  Devuelve: si el usuario es administrador, devuelve estado HTTP 200. En caso contrario, devuelve estado HTTP 400.

- ***logout*:** Realiza el *logout* de la aplicación. De esta forma, recoge el token de autenticación del usuario de las cabeceras HTTP y lo almacena en la lista negra de la base de datos.

  Método: POST.

  Ruta: /logout.

  Parámetros de la petición: no recibe parámetros.

  Devuelve: estado HTTP 200.

- ***crear*:** crea un usuario.

  Método: POST

  Ruta: /crear.

  Permisos necesarios: sesión iniciada y ser administrador.

  Parámetros de la petición:

  - Nombre de usuario a registrar.
  - Email del nuevo usuario.
  - Contraseña del nuevo usuario.
  - Administrador (será *True* si es administrador).

Devuelve: si el nombre de usuario es nuevo, estado HTTP 200. En caso contrario, estado HTTP 400.

- ***cambiar_pass***: cambia la contraseña antigua del usuario por la nueva.

  Método: POST.

  Ruta: /cambiar.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

  - ID del usuario.
  - Contraseña antigua.
  - Contraseña nueva.

  Devuelve: di la contraseña antigua coincide con la registrada actualmente, devuelve estado HTTP 200. En caso contrario, devuelve estado HTTP 400.

- ***recuperar***: envía un correo al usuario con una contraseña provisional poder acceder a la aplicación.

  Método: POST.

  Ruta: /recuperar.

  Parámetros de la petición:

  - Nombre de usuario.

  Devuelve: si el correo se envía correctamente, devuelve estado HTTP 200. En caso contrario, estado HTTP 400.

B.8.3. Usuarios middle

Contiene los métodos de conexión con la base de datos.

Métodos

- ***crear_usuario***: crea un usuario nuevo en la base de datos con el usuario y contraseña encriptados mediante una función *hash*, junto con el email y su rol.

Parámetros de la petición:

- Nombre de usuario.
- Email del usuario.
- Contraseña del usuario.
- Rol del usuario.

Devuelve: si no existe otro usuario con el mismo nombre de usuario, devuelve *True*. En caso contrario, devuelve *False*.

- ***comprobar_login***: realiza el login del usuario. Comprueba que el usuario existe y que la contraseña es la suya.

  Parámetros de la petición:

  - Nombre de usuario.
  - Contraseña del usuario.

  Devuelve: si los datos son correctos, devuelve estado HTTP 200. En caso contrario, estado HTTP 400.

- ***comprobar_password***: comprueba que la contraseña enviada es válida y si lo es, le actualiza a la contraseña. En caso contrario, lanza una excepción.

  Parámetros de la petición:

  - ID del usuario.
  - Contraseña antigua del usuario.
  - Contraseña nueva para el usuario.

  Devuelve: si los datos son correctos, actualiza la contraseña sin devolver nada. En caso contrario, lanza una excepción del tipo *WrongUserData*.

- ***update_pass***: actualiza a la contraseña del usuario por la nueva recibida. Acepta actualizaciones tanto por ID de usuario como por nombre de usuario, ya que ambos son identificadores únicos del mismo.

  Parámetros de la petición:

  - Contraseña nueva para el usuario.

- ID del usuario (opcional).

- Nombre del usuario (opcional).

Devuelve: no devuelve nada.

- ***get_email***: devuelve el correo electrónico del usuario a partir de su nombre.

  Parámetros de la petición:

  - Nombre del usuario.

  Devuelve: email del usuario.

- ***id_admin***: comprueba si el usuario es administrador o no.

  Parámetros de la petición:

  - ID del usuario.

  Devuelve: *True* si es administrador. *False* en caso contrario.

- ***insertar_blacklisttoken***: Inserta el token te autenticación en la lista negra.

  Parámetros de la petición:

  - JWT de autenticación.

  Devuelve: no devuelve nada.

- ***comprobar_blacklisttoken***: comprueba si el token de autenticación está en la lista negra de la base de datos.

  Parámetros de la petición:

  - JWT de autenticación.

  Devuelve: si el token ya está almacenado, devuelve el token almacenado. En caso contrario, devuelve *None*.

- ***insertar_refreshblacklisttoken***: inserta el token de refresco en la lista negra.

  Parámetros de la petición:

- JWT de refresco.

Devuelve: no devuelve nada.

- ***comprobar_refreshblacklisttoken***: comprueba si el token de refresco está en la lista negra de la base de datos.

  Parámetros de la petición:

  - JWT de refresco.

  Devuelve: si existe en la base de datos, devuelve su valor en la base da datos. En caso contrario, *None*.

- ***_comprobar_usuario***: comprueba si el usuario existe en la base de datos.

  Parámetros de la petición:

  - Nombre de usuario.

  Devuelve: si existe, devuelve el usuario con toda su información. En caso contrario, *None*.

- ***_usuarios_getnew_id***: devuelve el máximo id actual para los usuarios registrado en la base de datos.

  Parámetros de la petición: no recibe parámetros.

  Devuelve: último ID registrado, más 1.

- ***_refresh_getnew_id***: devuelve el máximo id actual para los tokens de refresco registrados en la base de datos.

  Parámetros de la petición: no recibe parámetros.

  Devuelve: último ID registrado, más 1.

- ***_auth_getnew_id***: devuelve el máximo id actual para los tokens de autenticación registrados en la base de datos.

  Parámetros de la petición: no recibe parámetros.

  Devuelve: último ID registrado, más 1.

B.9. Módulo de seguridad

La seguridad en el proyecto está basada en el uso de JWTs (JSON Web Tokens). No solo utiliza JWT para la autentificación, sino que aplica la idea de los *Refresh Tokens* para garantizar una mayor seguridad.

B.9.1. Archivos:

- Key
- JWT_util
- Middleware

B.9.2 Key

Se encarga de generar la clave privada y única para el servidor, que será utilizad para firmar los JWT.

Métodos

- ***get_key***: Autogenera una clave alfanumérica cifrada con hash3 de 256bits.

  Argumentos: no recibe parámetros.

  Devuelve: valor de la clave en formato *String*.

B.9.3. JWT_util

Contiene las claves del servidor y los métodos para codificar y decodificar los JWT.

Variables globales

- **auth_key**: clave alfanumérica cifrada con hash3 de 256 bits utilizada para firmar los JWT de autenticación.
- **refresh_key**: clave alfanumérica cifrada con hash3 de 256bits utilizada para firmar los JWT de refresco.

Métodos

- ***encode_auth_token***: genera un token de autenticación con el ID del usuario autenticado. Este token tendrá una validad de 6 horas.

  Argumentos:

  - user_id: ID del usuario.

  Devuelve: token de autenticación.

- ***encode_refresh_token***: genera un token de refresco con una validez de 1 minuto.

  Argumentos: no recibe parámetros.

  Devuelve: token de refresco.

- ***decode_auth_token***: decodifica el token de autenticación.

  Argumentos:

  - auth_token: token de autenticación del usuario.

  Devuelve: En caso de que se decodifique correctamente, devolveremos el id del usuario al que pertenece. En caso de error de decodificación o si el token ya había sido utilizado, se lanzará una excepción.

- ***decode_refresh_token***: decodifica el token de refresco.

  Argumentos:

  - refresh_token: token de refresco del usuario

  Devuelve: en caso de que se decodifique correctamente, devolveremos *True*.

  En caso de error de decodificación o si el token ya había sido utilizado, se lanzara una excepción

B.9.4. Middleware

Aquí yacen los métodos middleware encargados de la verificación de validez de los tokens y de comprobar los permisos del usuario

Métodos

- ***login_required***: capta el token de autenticación e intenta decodificarlo. Si la decodificación es válida, es decir, el token sigue estando vigente, la ejecución continúa hasta el *endpoint* para el que fuese la petición.

  Argumentos: no recibe parámetros.

  Devuelve: no devuelve nada.

- ***admin_required***: capta el token de autenticación e intenta decodificarlo. Si la decodificación es válida, comprueba si el usuario tiene permisos de administrador o no. Si es administrador, la ejecución continua hasta el endpoint deseado. En cualquier otro caso, devolverá 403 - *Forbidden*.

  Argumentos: no recibe parámetros.

  Devuelve: no devuelve nada.

- ***middleware_salida***: capta el token de refresco justo antes de responder al cliente e intenta decodificarlo. Si la decodificación es válida, se responde al cliente. Si no lo es, se le genera otro nuevo token de refresco al cliente y se le envía junto al contenido de la respuesta, pues se da por hecho que no el JWT de autenticación es válido actualmente, ya que ha superado el *login_required*. En cualquier otro caso, devolverá 403 - Forbidden.

  Argumentos: no recibe parámetros.

  Devuelve: no devuelve nada.

B.10. Módulo de agentes

Módulo de simulación de delitos basada en agentes. Se diferencian dos partes principales:

- Simulación de delitos por agentes

- Evolución de delitos a lo largo del tiempo

B.10.1. Archivos

- Servicios
- Agentes_middle
- Agentes

B.10.2. Servicios:

Contiene los servicios para permitir la comunicación entre *front* y *back-end*.

Métodos

- **Agentes:** *endpoint* para la simulación de delitos basada en agentes.

  Método: GET.

  Ruta: /agentes.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

  - Elecciones : parámetros elegidos por el usuario para simular.
  - Municipios : municipios que simular.
  - Distritos : distritos que simular.
  - Festividad : festividades que simular.
  - por_distrito : variable para simular en distritos o municipios.
  - auto : variable para indicar si la simulación debe simular otras opciones aparte de las especificadas por el usuario..

- mes_desde : mes desde el que se simulan los datos.

- mes_hasta : mes hasta el que se simulan los datos.

- dia_desde : día desde el que se simulan los datos.

- dia_hasta : día hasta el que se simulan los datos.

- hora_desde : hora desde la que se simulan los datos.

- hora_hasta : hora hasta la que se simulan los datos.

Devuelve: satos resultantes de la simulación.

- ***evolucion_service*:** *endpoint* para la evolución de delitos a lo largo de los 12 meses del año, o de los especificados.

  Método: GET.

  Ruta: /evolución.

  Permisos necesarios: sesión iniciada.

  Parámetros de la petición:

  - Provincia: provincia. Su valor por defecto es Málaga.

  - Municipio: municipio.

  - Distrito_municipal: distrito municipal.

  - resp_sexo : género del responsable.

  - dias_semana : días de la semana.

  - resp_nacionalidad : nacionalidad o nacionalidades de los responsables.

  - tramo_horario : tramo o tramos horarios.

  - tipo_lugar : tipos de lugares. Ejemplo: Farmacia, Vía urbana.

  - tipo_hecho : tipos de hecho.

  - modus_op : modus operandi.

  - meses : meses en los que ver la evolución. Si no se especifica, se hará sobre todos los meses del año.

  - festividades: festividades. Por ejemplo: Carnaval, Feria de Málaga,

Navidad.

- por_distrito : variable para indicar si hacer la evolución mensual para la provincia o el municipio de málaga.

- auto : variable para indicar si la simulación debe simular otras.

- dia_desde : día del mes desde el que se visualizarán los datos.

- dia_hasta : día del mes hasta el que se visualizarán los datos.

Devuelve: si la provincia ha sido enviada, devuelve los datos de resultantes de la simulación, En cualquier otro caso, devuelve estado HTTP 400.

- ***___limpiar_datos_predicción***: limpia los datos de "elecciones" recibidos para la simulación de forma que sea operable por el servidor.

  Parámetros de la petición:

  - elecciones: parámetros elegidos por el usuario, en orden de preferencia y rellenos.

  Devuelve: una lista de datos limpios y listos para ser operados por el servidor.

B.10.3. Agentes_middle

Contiene los servicios para permitir la comunicación entre *front* y *back-end.*

Métodos

- **Evolucion**: aplica los filtros seleccionados a cada mes del año. En caso de que se hayan seleccionado varios meses adrede, se aplicará solo a estos meses.

  Parámetros de la petición:

  - tipo_lugar: tipos de lugar. Ejemplo: Farmacia, Vía urbana.

  - tipo_hecho: tipos de hecho.

- modus_op: modus operandi.

- resp_nacionalidad: nacionalidad o nacionalidades de los responsables.

- distrito_municipal: distritos municipales.

- dia_desde: día desde el que ver la evolución.

- dia_hasta: día hasta el que ver la evolución.

- resp_sexo: género del responsable.

- provincia: provincia.

- municipio: municipios.

- tramo_horario: tramo o tramos horarios.

- dias_semana: días de la semana.

- festividades: festividades. Por ejemplo: Carnaval, Feria de Málaga, Navidad.

- meses: meses en los que ver la evolución.

- por_distrito: variable para ver la evolución por distritos o municipios.

Devuelve: diccionario JSON con los datos resultantes de la búsqueda para cada mes del año o para cada mes seleccionado.

- ***agentes_simulacion***: genera un modelo de agentes para cada año registrado en la base de datos con las elecciones recibidas. Si el autocompletado está desactivado, busca con los parámetros recibidos únicamente. Si está activado, busca los más relevantes para cada característica no rellena de los hechos.

  Todo esto se hace siguiendo un árbol de decisiones, de forma que cada elección repercute en el resultado de las siguientes.

  Parámetros de la petición:

  - municipios: municipios.

  - distritosm: distritos municipales.

  - eleccion: parámetros seleccionados por el usuario para la simulación.

- mes_desde: mes desde el que simular.

- mes_hasta: mes hasta el que simular.

- dia_desde: día desde el que simular.

- dia_hasta: día hasta el que simular.

- hora_desde: hora desde la que simular.

- hora_hasta: hora hasta la que simular.

- festividad: festividad. Por ejemplo: Carnaval, Feria de Málaga, Navidad.

- por_distrito: buscar por distritos o por municipios.

- auto: autocompletar con datos relevantes o no.

Devuelve: diccionario JSON con los datos resultantes de la simulación.

- ***__calcular__dia***: dentro del árbol de decisiones, calcula los porcentajes de los días de la semana seleccionados para el filtro acumulado. Si es automático, busca en todos los días de la semana y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

  - mydict: diccionario con los filtros acumulados.

  - siguientes: lista con las elecciones restantes.

  - auto: autocompletar o no. Por defecto, es *False*.

  Devuelve*:* no devuelve nada.

- ***__calcular__tramo***: dentro del árbol de decisiones, calcula los porcentajes de los tramos horarios seleccionados para el filtro acumulado. Si es automático, busca en todos los tramos horarios y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

  - mydict: diccionario con los filtros acumulados.

- siguientes: lista con las elecciones restantes.

- auto: autocompletar o no. Por defecto, es *False*.

Devuelve: no devuelve nada.

- ***__calcular__hecho***: dentro del árbol de decisiones, calcula los porcentajes de los hechos seleccionados para el filtro acumulado. Si es automático, busca en todos los hechos y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

  - mydict: diccionario con los filtros acumulados.

  - siguientes: lista con las elecciones restantes.

  - auto: autocompletar o no. Por defecto, *False*.

  Devuelve: no devuelve nada.

- ***__calcular__modus***: dentro del árbol de decisiones, calcula los porcentajes de los modus operandi seleccionados para el filtro acumulado. Si es automático, busca en todos los modus operandi y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

  - mydict: diccionario con los filtros acumulados.

  - siguientes: lista con las elecciones restantes.

  - auto: autocompletar o no. Por defecto, *False*.

  Devuelve: no devuelve nada.

- ***__calcular__nacionalidad***: dentro del árbol de decisiones, calcula los porcentajes de las nacionalidades seleccionadas para el filtro acumulado. Si es automático, busca en todas las nacionalidades y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

- mydict: diccionario con los filtros acumulados.

- siguientes: lista con las elecciones restantes.

- auto: autocompletar o no. Por defecto, *False*.

Devuelve: no devuelve nada.

- ***__calcular__sexo***: dentro del árbol de decisiones, calcula los porcentajes de los sexos seleccionadas para el filtro acumulado. Si es automático, busca en todos los sexos y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

  - mydict: diccionario con los filtros acumulados.

  - siguientes: lista con las elecciones restantes.

  - auto: autocompletar o no. Por defecto, *False*.

  Devuelve: no devuelve nada.

- ***__calcular__lugar***: dentro del árbol de decisiones, calcula los porcentajes de los lugares seleccionadas para el filtro acumulado. Si es automático, busca en todos los lugares y extrae los más relevantes.

  Parámetros de la petición:

  - modelo: HistoricoModel donde se almacenan los datos generados.

  - mydict: diccionario con los filtros acumulados.

  - siguientes: lista con las elecciones restantes.

  - auto: autocompletar o no. Por defecto, *False*.

  Devuelve: no devuelve nada.

- ***complejos***: busca en la base de datos los grupos de datos complejos (Nacionalidad del responsable y Sexo del responsable). Si es automático, busca los más relevantes respecto a los filtros especificados. En caso contrario, busca específicamente las nacionalidades o sexos especificados y los ordena por relevancia.

Parámetros de la petición:

- clave: identificador del atributo en la base de datos.
- mydict: diccionario con los filtros acumulados.
- auto: marca si la búsqueda debe ser automática o no.

Devuelve: diccionario con las ocurrencias ordenadas de mayor a menos relevancia.

- ***___encontrar__característica***: Busca específicamente los valores del diccionario agrupándolos por la clave en cuestión. El resultado viene ordenado de mayor a menor relevancia.

  Parámetros de la petición:

  - clave: identificador del atributo en la base de datos.
  - mydict: diccionario con los filtros acumulados.
  - auto: marca si la búsqueda debe ser automática o no.

  Devuelve: no devuelve nada.

- ***__relevantes__segun***: busca las características más relevantes de forma automática en la base de datos.

  Parámetros de la petición:

  - clave: identificador del atributo en la base de datos.
  - mydict: diccionario con los filtros acumulados.
  - auto: marca si la búsqueda debe ser automática o no.

  Devuelve: no devuelve nada.

- ***___mas__relevantes***: se encarga de ordenar los valores recibidos de mayor a menos relevancia. Si criba, solo coge los que pertenezcan al 90% de los resultados más relevantes. En caso contrario, únicamente ordenará.

  Parámetros de la petición:

  - característica: valores a ordenar.
  - cribar: decide si cribamos o no. Por defecto a *True*.

- auto: marca si la búsqueda debe ser automática o no.

Devuelve: Si cribamos u ordenamos, devuelve los valores ordenados. Si el valor obtenido para cribar es 0, devuelve un diccionario vacío. Si característica tiene longitud 0 y no es un filtrado automático, devuelve una excepción del tipo *AgenteException*.

B.10.4. Agentes

Contiene a los agentes (HechoAgent) y los distintos tipos de modelos que hacen uso de ellos. Encontramos 3 modelos dentro de esta jerarquía:

- **DelitoModel**
- **HistoricoModel**
- **SimulacionModel**

Clases

- **HechoAgent**: Representa a un agente.

  Atributos:

  - hecho: Tipo de hecho.
  - lugar: Tipo de lugar.
  - modus: Tipo de modus operandi.
  - dia_semana: Día de la semana.
  - nacionalidad: Nacionalidad.
  - tramo_horario: Tramo horario.
  - sexo: Sexo.
  - municipio: Municipio.
  - distrito: Distrito.

  Métodos:

  - ***set_atributo***: permite hacer set de varios atributos.
  - ***get_atributo***: permite obtener el valor de varios atributos.

- **DelitoModel:** Es la superclase de todos los modelos. Contiene un conjunto de delitos, representados por agentes.

  Predecesora: no tiene clase predecesora.

  Atributos:

  - porcentajes: diccionario con el conjunto de porcentajes para cada característica de un delito.

  - num_agentes: número de agentes que contiene.

  - schedule: planificación - por defecto Aleatoria.

  - agentes: lista con todos los agentes que posee.

  Métodos:

  - asignar_caraceristica: le asigna una característica (ej. tipo de hecho) a cada agente. Esta asignación se realiza de forma aleatoria en base a unos porcentajes de relevancia asignados a cada característica específica.

  - ***_calcular_porcentaje:*** calcula la medida porcentual en que cada característica ha sido asignada a los agentes.

- **HistoricoModel:** contiene un conjunto de delitos, representados por agentes, en un año específico.

  Predecesora: DelitoModel.

  Atributos:

  - myid: identificador único del modelo, para poder distinguirlo del resto.

  - anyo: año que representa el modelo.

  Métodos: no tiene métodos nuevos ni sobrescritos.

- **SimulacionModel:** Contiene todos los modelos HistoricoModel creados, y a partir de ellos obtiene unos porcentajes aproximados de ocurrencia de las distintas características de un delito.

Predecesora: DelitoModel.

Atributos

- aproximación: Diccionario con la aproximación porcentual de cada característica y municipio o distrito donde ocurrirá.

- aproximacion_total: Porcentaje de aproximación entre la simulación y el último año registrado en la base de datos.

Métodos:

- ***__calcular__porcentaje__simulacion***: calcula la medida porcentual en que cada característica ha sido asignada a los agentes.
- ***asignar__caracteristica***: difiere del método de su superclase en que aquí "valores" es un diccionario, no una lista. Le asigna una característica (ej. tipo de hecho) a cada agente de forma aleatoria, en base a unos porcentajes de relevancia asignados a cada característica específica.
- **comparar:** compara las aproximaciones generadas con el modelo del último año registrado y genera una aproximación media.

Apéndice C

Casos de uso

C.1. Introdución

Aquí se especifican los casos de uso más importantes relativos a este TFG, ligados a la importación y exportación de datos y a la realización de las búsquedas en los mapas de la aplicación. Siendo el primero relativo a la provincia de Málaga y el segundo a los distritos de la capital de provincia.

Cada caso de uso incluye diagramas de secuencia e imágenes de la interfaz para un mayor entendimiento.

C.2 Importación de ficheros

Título	Importación de ficheros
Descripción	Un administrador importar uno o varios ficheros al sistema.
Pre-condición	El usuario tiene que ser administrador.
Post-condición	Los datos tienen que estar en la base de datos
Prioridad	Alta
Autor(es)	Juan Palma Borda
Escenario principal	
1. Un administrador introduce uno o varios ficheros en el cuadro de importación. 2. El administrador pulsa el botón para proceder a la importación de datos. 3. Los ficheros se copian en el *Backend*. 4. Se crea el *LectorDenuncias*. 5. Se crea una hebra para cada fichero. 6. Para cada fichero se busca la línea de las cabeceras. 7. Se encuentran las cabeceras. 8. Se crea el *DiccionarioDenuncias* con las cabeceras. 9. Se crea una *Denuncia* por cada línea del fichero. 9.1 Se comprueba si la *Denuncia* existe. 9.2 La *Denuncia* no existe, por lo que se introduce en el *DiccionarioDenuncias.* 10. Se comprueba si las *Denuncias* no existen en la Base de datos. 11. Se generan las *Localizaciones* y las *Cabeceras* de las *Denuncias.* 12. Se importan las *Localizaciones* y las *Cabeceras.*	

13. Se eliminan de memoria las *Localizaciones* y las *Cabeceras.*

14. Se importan las *Denuncias* a la Base de datos.

15. Se devuelve el control al administrador, notificándole éxito en la importación.

Escenario alternativo

1.a No se introducen ficheros en el cuadro de importación.

2.a El administrador pulsa el botón para proceder a la importación de datos.

3.a El sistema notifica que no hay ficheros para importar y le devuelve el control al administrador.

7.c No se encuentran las cabeceras.

8.c Se notifica que la importación no ha tenido fallos, pero no ejecuta los pasos de importación visibles en la interfaz

9.2.d La *Denuncia existe.*

9.3.d Se actualiza la *Denuncia* existente en memoria con los datos de esta.

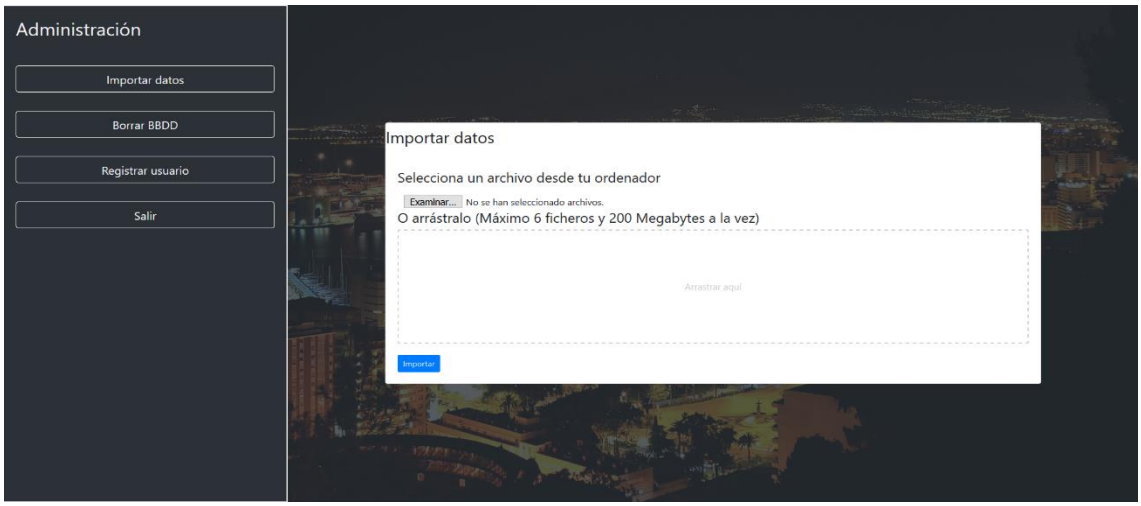
9.4.d Se elimina la última *Denuncia* leída.

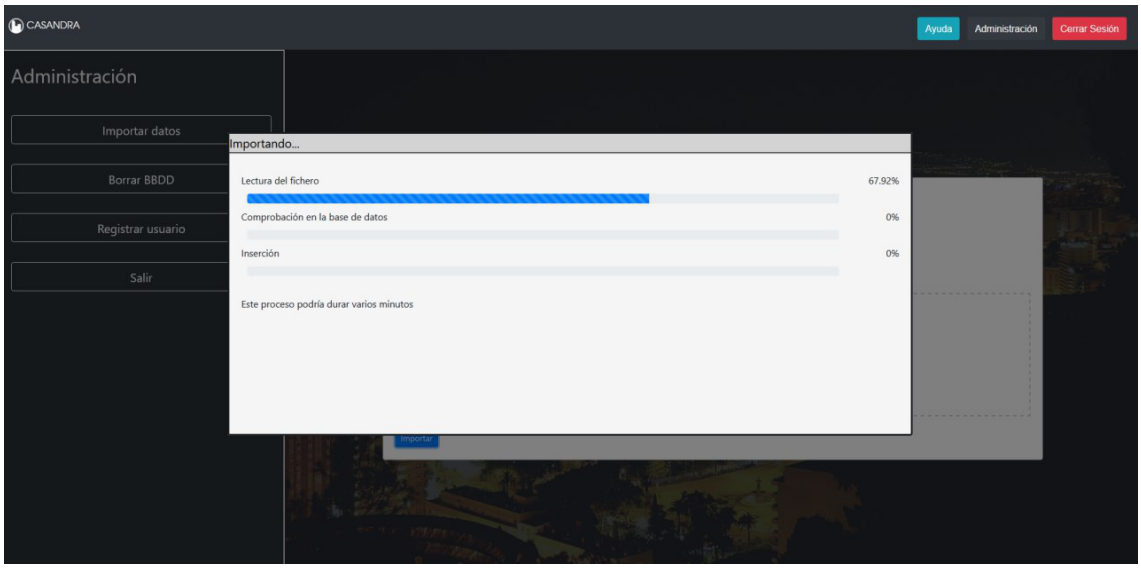
9.5. Se continua con la lectura de las *Denuncias.*

Clases de análisis

A. Clases de entidad	*Denuncia, Localización* y *Cabcera.*
B. Clases de control	*Importador*, *LectorDenuncias* y *DiccionarioDenuncias,*
C. Clases de interfaz	*Controlador interfaz*

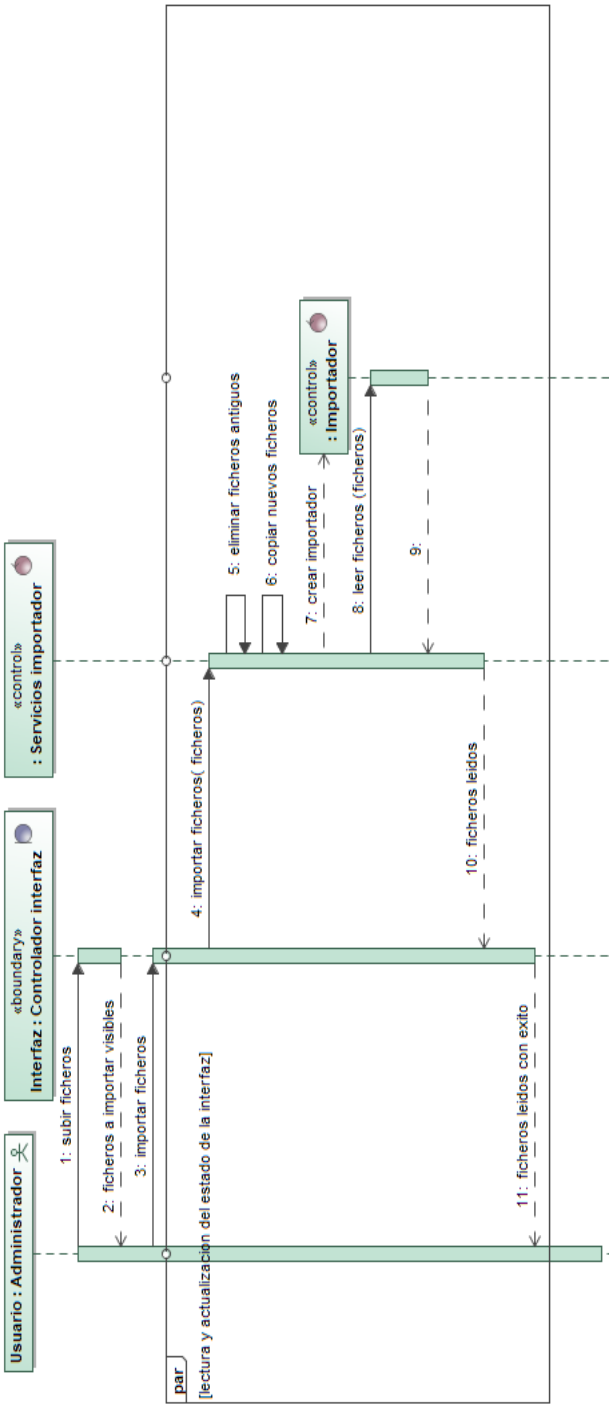
Maquetas de interfaz

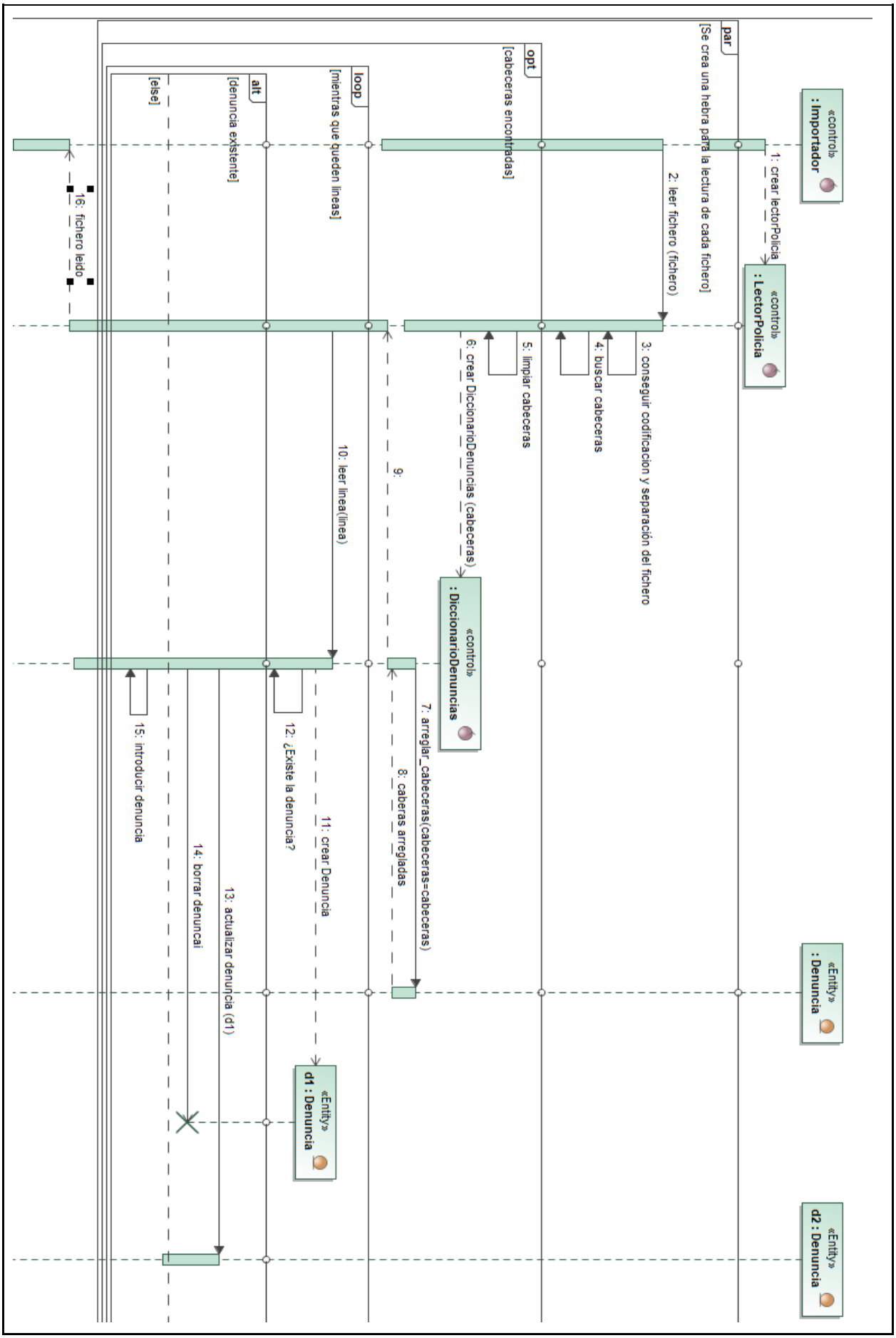


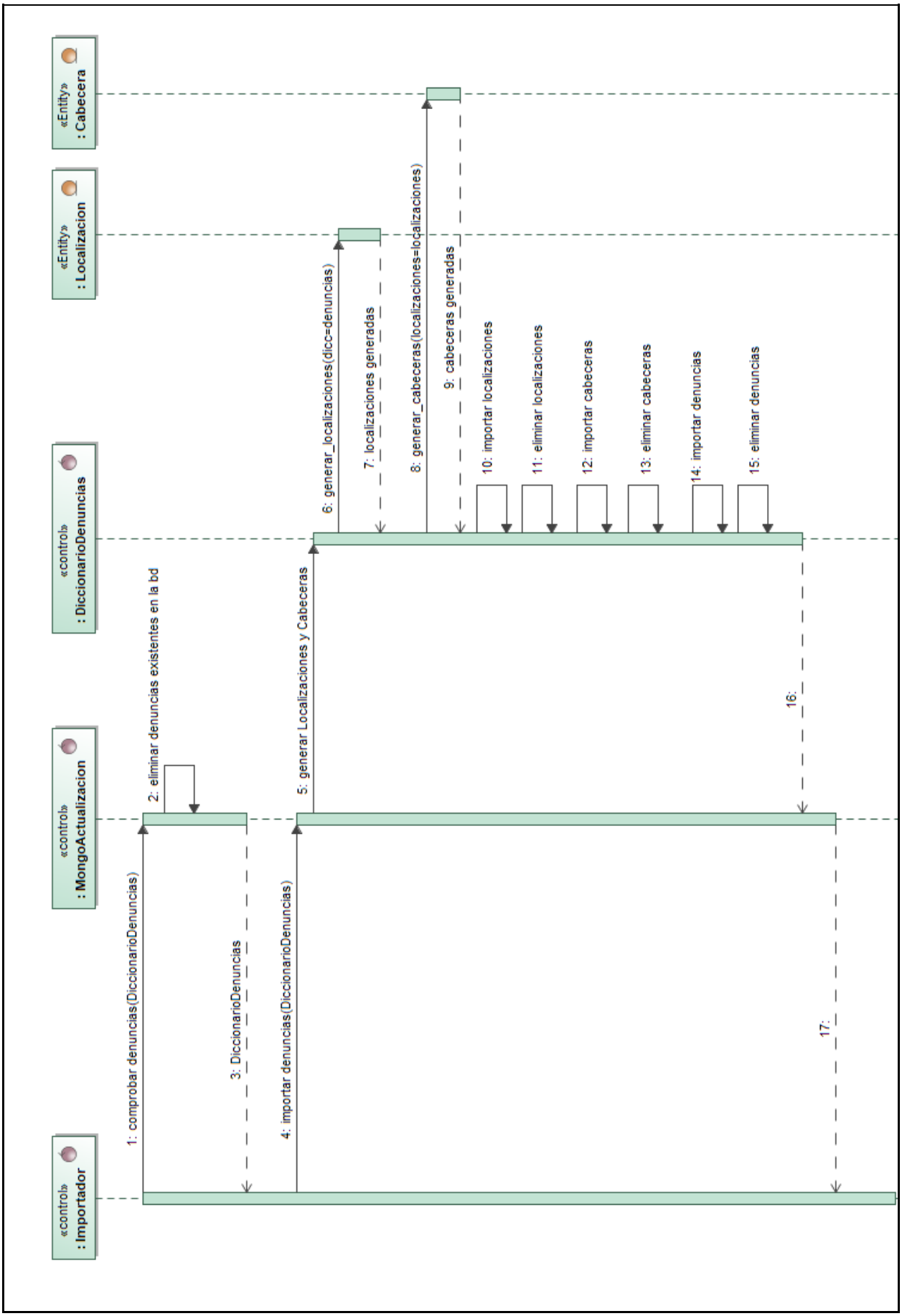


Diagramas de secuencia

Hay tres diagramas: el segundo y el tercero ocurren entre el paso ocho y nueve del primer diagrama, ejecutándose primero el segundo y, a continuación, el tercero.




«control»
: Importador
«control»
: LectorPolicia
«Entity»
: Denuncia
«Entity»
d2 : Denuncia
«control»
: DiccionarioDenuncias
«Entity»
d1 : Denuncia
par
[Se crea una hebra para la lectura de cada fichero]
opt
[cabeceras encontradas]
loop
[mientras que queden lineas]
alt
[denuncia existente]
[else]
1: crear lectorPolicia
2: leer fichero (fichero)
3: conseguir codificacion y separación del fichero
4: buscar cabeceras
5: limpiar cabeceras
6: crear DiccionarioDenuncias (cabeceras)
7: arreglar_cabeceras(cabeceras=cabeceras)
8: caberas arregladas
9:
10: leer linea(linea)
11: crear Denuncia
12: ¿Existe la denuncia?
13: actualizar denuncia (d1)
14: borrar denuncai
15: introducir denuncia
16: fichero leido

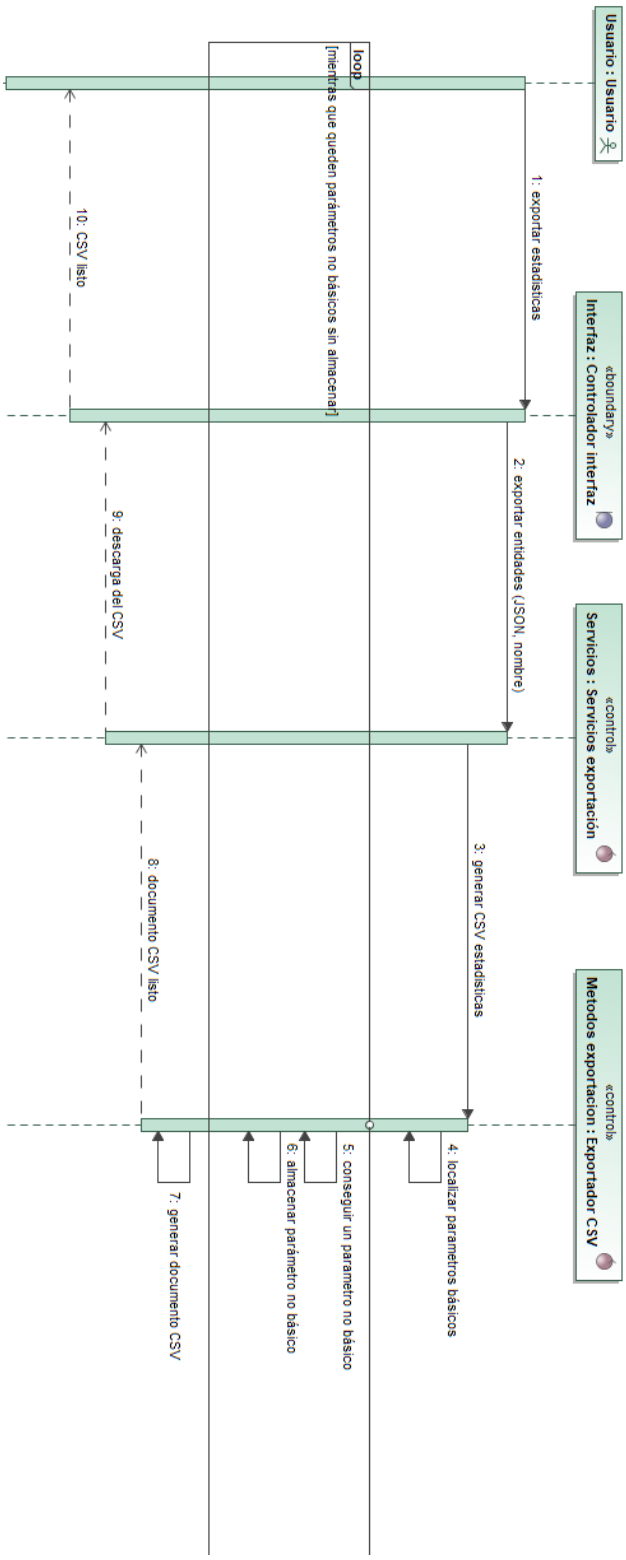

«Entity»
: Cabecera
«Entity»
: Localizacion
«control»
: DiccionarioDenuncias
«control»
: MongoActualizacion
«control»
: Importador
1: comprobar denuncias(DiccionarioDenuncias)
2: eliminar denuncias existentes en la bd
3: DiccionarioDenuncias
4: importar denuncias(DiccionarioDenuncias)
5: generar Localizaciones y Cabeceras
6: generar_localizaciones(dicc=denuncias)
7: localizaciones generadas
8: generar_cabeceras(localizaciones=localizaciones)
9: cabeceras generadas
10: importar localizaciones
11: eliminar localizaciones
12: importar cabeceras
13: eliminar cabeceras
14: importar denuncias
15: eliminar denuncias
16:
17:

C.3 Exportación de ficheros de estadísticas de una zona

Título	Exportación de fichero con las estadísticas de una zona.
Descripción	Un usuario exporta las estadísticas de una zona en formato *CSV*.
Pre-condición	Se ha tenido que generar unas estadísticas visibles en la pantalla.
Post-condición	Se ha debido exportar con éxito las estadísticas seleccionadas.
Prioridad	Media
Autor(es)	Juan Palma Borda
Escenario principal	
1. Se selecciona la exportación de estadísticas. 2. Se envía el JSON al *Backend*, para no tener que generarlo de nuevo. 3. Se reúnen los datos básicos de las estadísticas dentro del JSON, como la provincia o el municipio. 4. Se consiguen las estadísticas concretas del JSON, tales como número de hurtos. 5. Se redacta en formato *CSV* con la codificación cp1252, para que no haya caracteres raros cuando se abra en *Excel*. 6. Se descarga en el navegador del usuario.	

Clases de análisis	
A. Clases de entidad	-
B. Clases de control	*Servicios exportación y exportador CSV*
C. Clases de interfaz	*Controlador interfaz*
Maquetas de interfaz	

Diagramas de secuencia

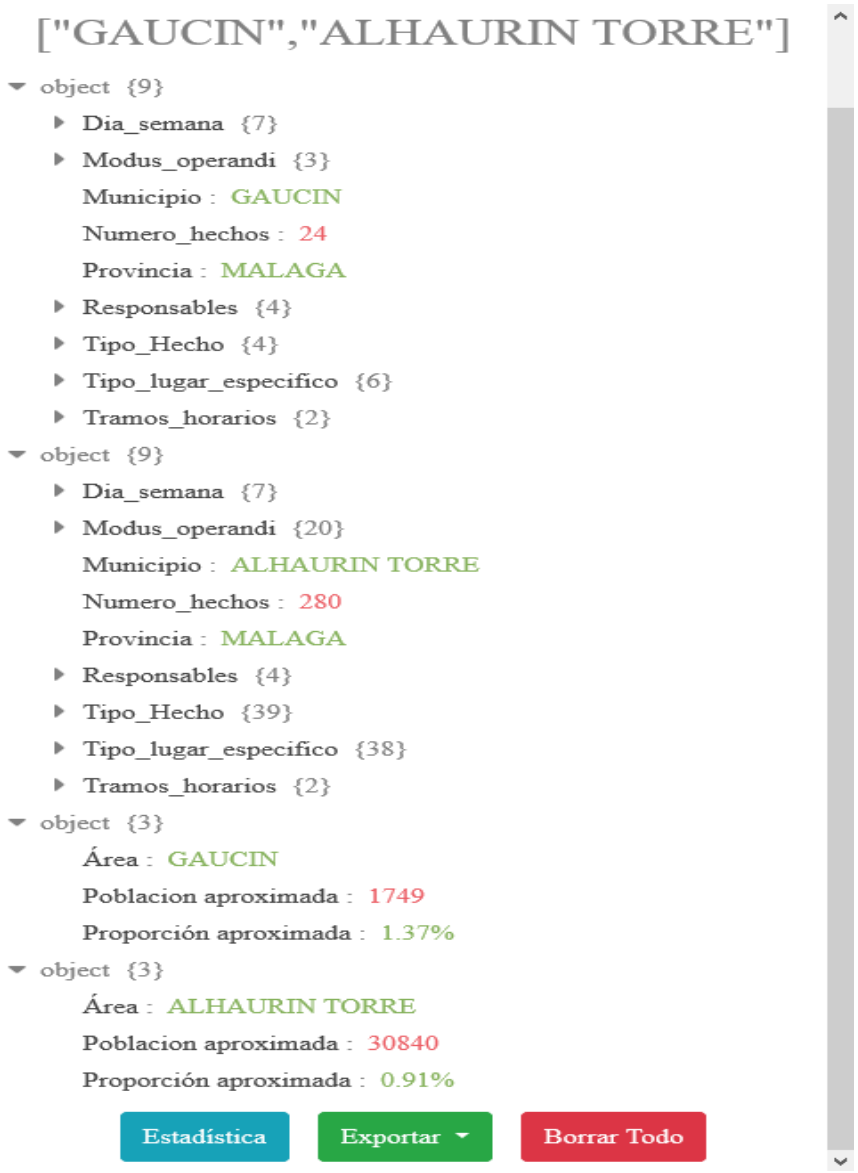


C.4 Exportación de ficheros de entidades de una búsqueda

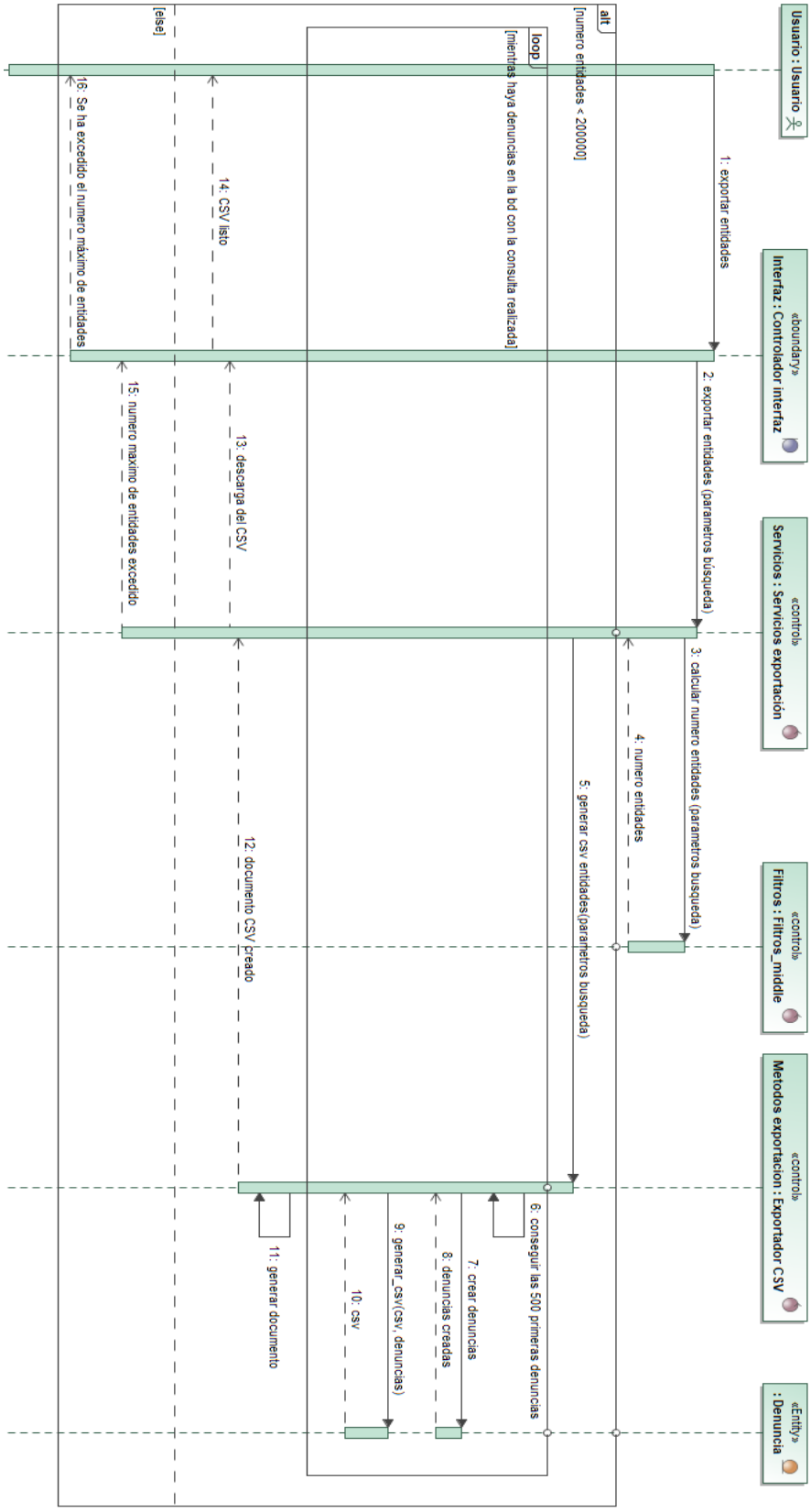
Título	Exportación de fichero con las entidades de una búsqueda.
Descripción	Un usuario exporta las entidades de una búsqueda concreta.
Pre-condición	Se ha tenido que generar unas estadísticas visibles en la pantalla.
Post-condición	Se ha debido exportar con éxito las entidades de la búsqueda.
Prioridad	Alta
Autor(es)	Juan Palma Borda
Escenario principal	
1. Se selecciona la exportación de entidades. 2. Se envía la búsqueda al *Backend*. 3. Se comprueba que no supera las 200.000 entidades. 4. Se van juntando las denuncias de quinientos en quinientos y se van almacenando en un diccionario especial para generar el *CSV*. 5. Una vez se tienen todas las denuncias, se procede a generar el *CSV*. 6. Se redacta el documento en formato *CSV* con la codificación cp1252, para que no haya caracteres raros cuando se abra en *Excel*. 7. Se descarga en el navegador del usuario.	
Escenario alternativo	
3.a Supera las 200.000 entidades. 4.a Se notifica que la exportación ha fallado por exceso de entidades.	

Clases de análisis	
A. Clases de entidad	*Denuncia*
B. Clases de control	*Servicios exportación, filtros_middle y exportador CSV*
C. Clases de interfaz	*Controlador interfaz*

Maquetas de interfaz



Diagramas de secuencia



C.5 Búsqueda por mes y año en la provincia de Málaga

Título	Búsqueda por mes y año (DMA) en la provincia de Málaga
Descripción	Un usuario realiza una búsqueda filtrando por mes, año y/u otros parámetros para visualizar el resultado en el mapa de la provincia de Málaga.
Pre-condición	El usuario está viendo el mapa de la provincia de Málaga.
Post-condición	Mapa cargado correctamente
Prioridad	Alta
Autor(es)	Juan Palma Borda Sergio Gavilán Prieto

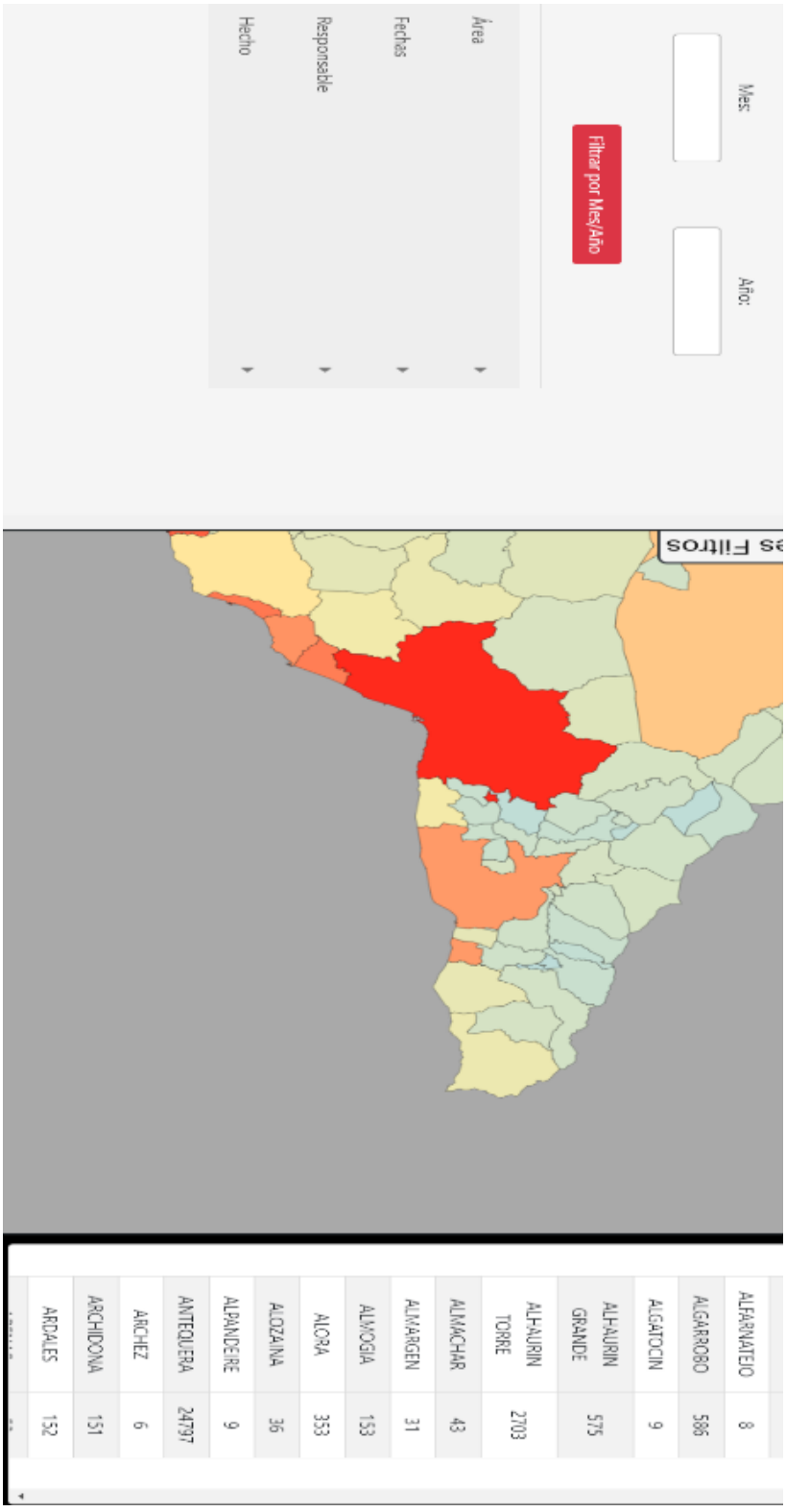
Escenario principal

1. El usuario selecciona los parámetros de la búsqueda.
2. El usuario pulsa el botón de filtrar por DMA.
3. El sistema envía los parámetros de búsqueda al servicio *busqueda_dia_mes_anno*.
4. Se limpian los parámetros.
5. Se comprueba si es una búsqueda simple (un único parámetro que no sea ni de fecha, ni de lugar.
6. Es una búsqueda simple.
7. Se prepara la consulta para la colección Localizaciones.

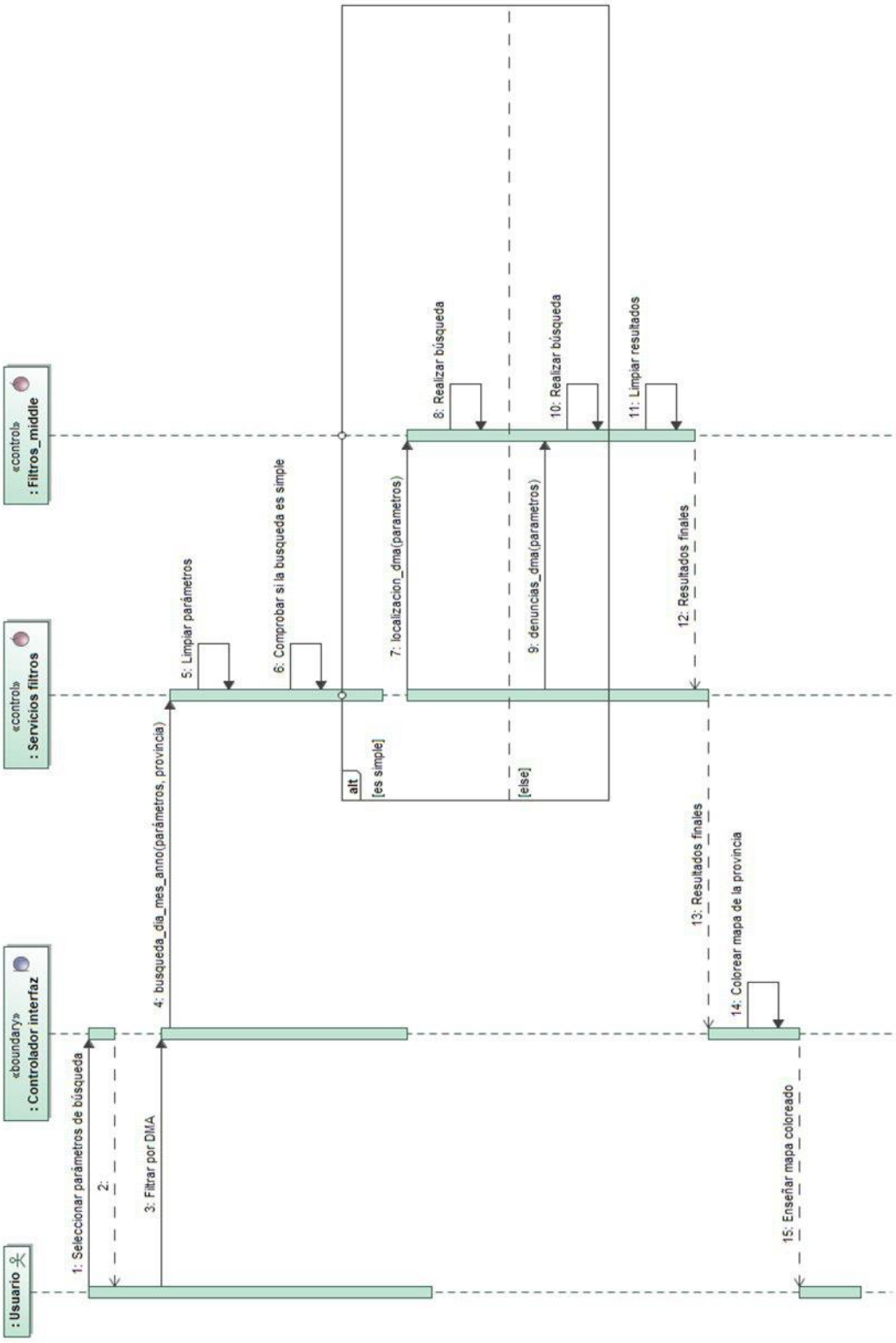
8. Se realiza la búsqueda en MongoDB. 9. Se limpian los resultados y se preparan para el envío. 10. Se envían los resultados a la interfaz. 11. Se colorea el mapa con los datos recibidos.
Escenario alternativo
6.a No es una búsqueda simple. 7.a Se prepara la consulta para la colección Denuncias. 8.a Se realiza la búsqueda en MongoDB. 9.a Se limpian los resultados y se preparan para el envío. 10.a Se envían los resultados a la interfaz. 11.a Se colorea el mapa con los datos recibidos.
Clases de análisis

A. Clases de entidad	-
B. Clases de control	*Servicios filtros y filtros_middle*
C. Clases de interfaz	*Controlador interfaz*

Maquetas de interfaz



Diagramas de secuencia



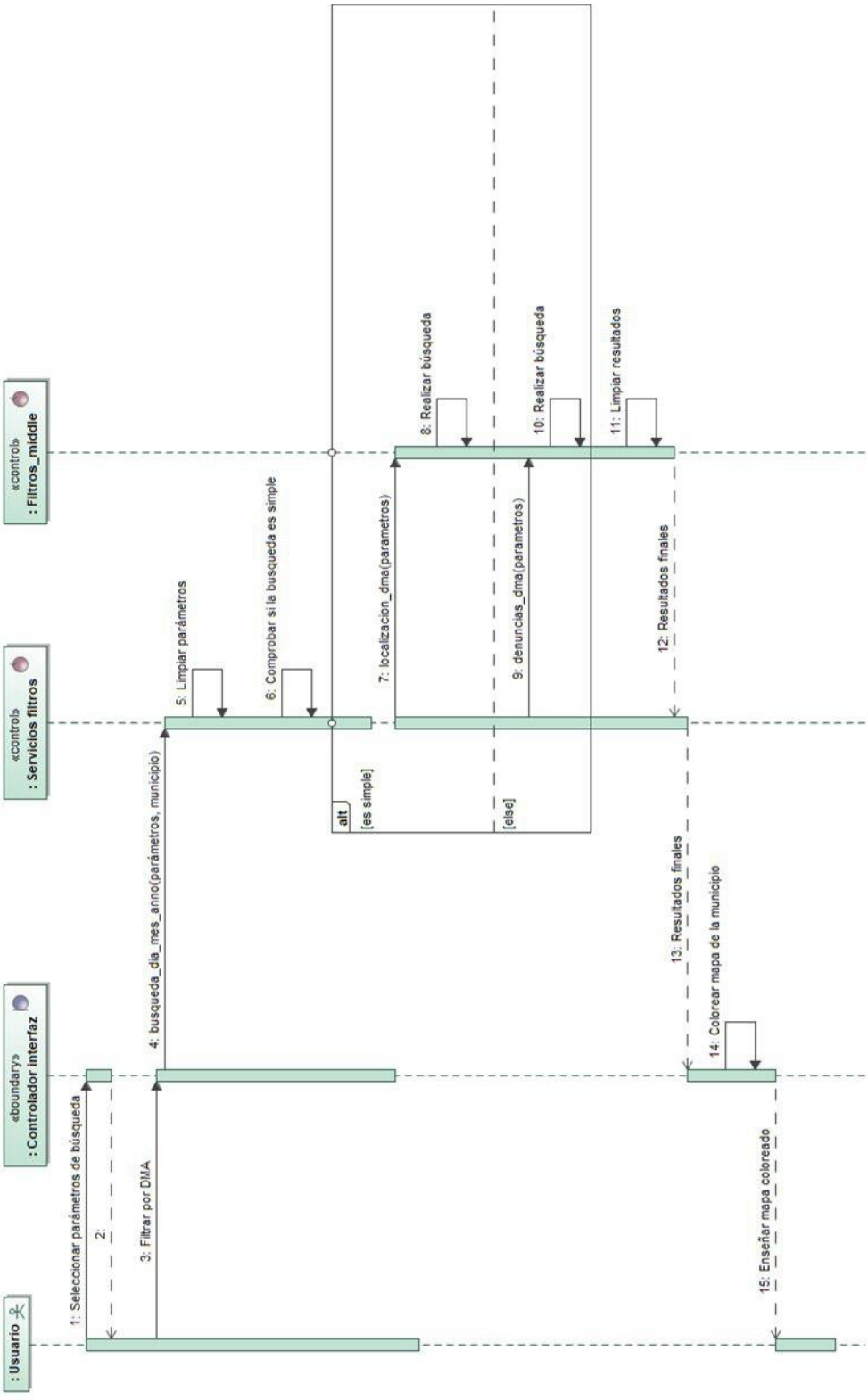
C.5 Búsqueda por mes y año en el municipio de Málaga

Título	Búsqueda por mes y año (DMA) en el municipio de Málaga
Descripción	Un usuario realiza una búsqueda filtrando por mes, año y/u otros parámetros para visualizar el resultado en el mapa del municipio de Málaga.
Pre-condición	El usuario está viendo el mapa del municipio de Málaga.
Post-condición	Mapa cargado correctamente
Prioridad	Alta
Autor(es)	Juan Palma Borda, Sergio Gavilán Prieto
Escenario principal	
1. El usuario selecciona los parámetros de la búsqueda. 2. El usuario pulsa el botón de filtrar por DMA. 3. El sistema envía los parámetros de la búsqueda al servicio *busqueda_dia_mes_anno.* 4. Se limpian los parámetros. 5. Se comprueba si es una búsqueda simple (un único parámetro que no sea ni de fecha, ni de lugar. 6. Es una búsqueda simple. 7. Se prepara la consulta para la colección Localizaciones. 8. Se realiza la búsqueda en MongoDB. 9. Se limpian los resultados y se preparan para el envío.	
10. Se envían los resultados a la interfaz. 11. Se colorea el mapa con los datos recibidos.	

Escenario alternativo	
6.a No es una búsqueda simple. 7.a Se prepara la consulta para la colección Denuncias. 8.a Se realiza la búsqueda en MongoDB. 9.a Se limpian los resultados y se preparan para el envío. 10.a Se envían los resultados a la interfaz. 11.a Se colorea el mapa con los datos recibidos.	
Clases de análisis	
A. Clases de entidad	-
B. Clases de control	*Servicios filtros y filtros_middle*
C. Clases de interfaz	*Controlador interfaz*

Maquetas de interfaz

Diagramas de secuencia

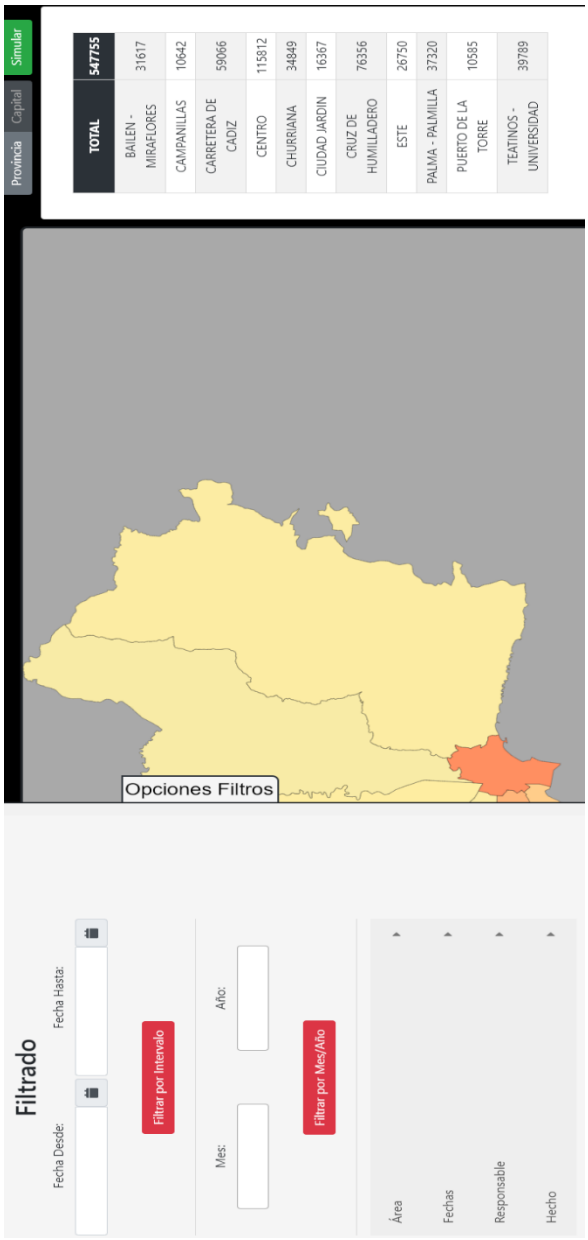


C.6 Búsqueda por intervalo de tiempo en el municipio de Málaga

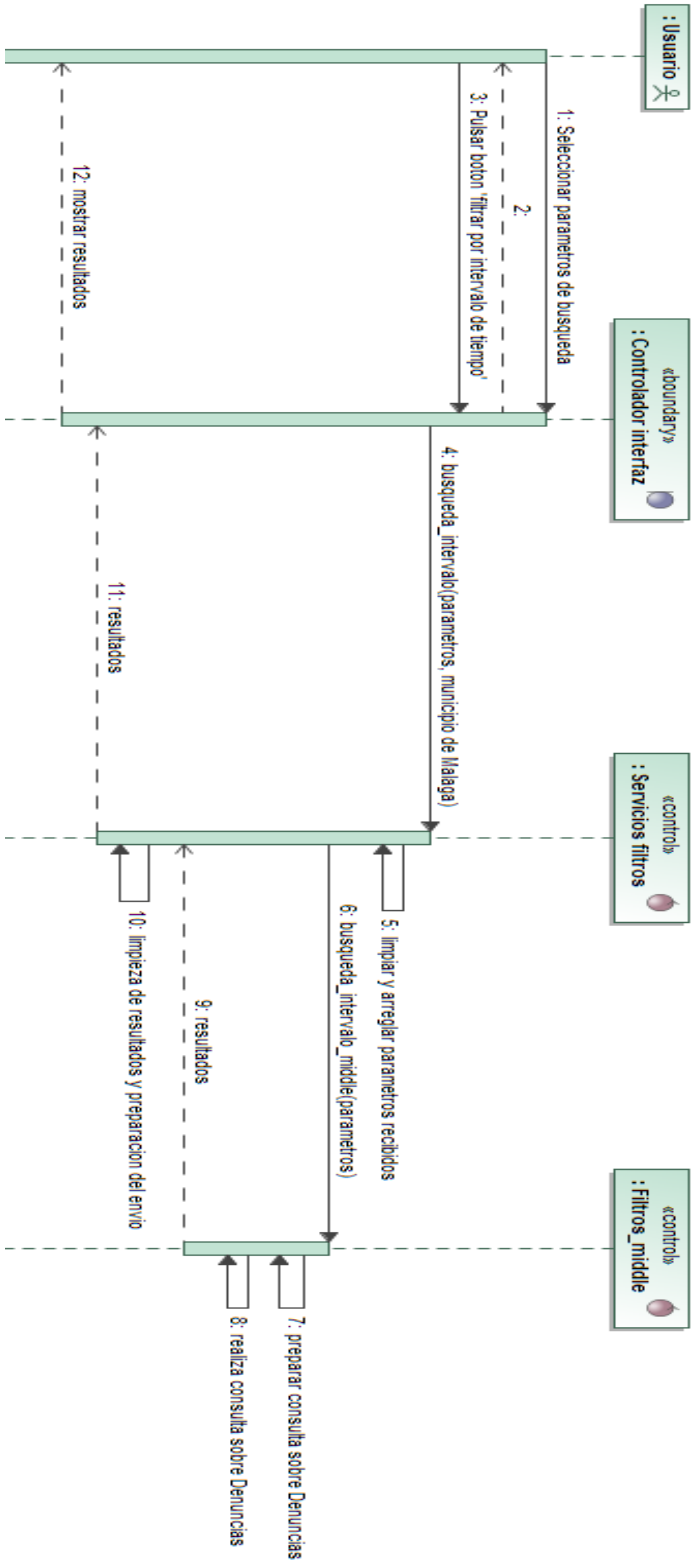
Título	Búsqueda por un intervalo de tiempo en el municipio de Málaga
Descripción	Un usuario realiza una búsqueda filtrando por un intervalo temporal y/u otros parámetros para visualizar el resultado en el mapa del municipio de Málaga.
Pre-condición	El usuario está viendo el mapa del municipio de Málaga.
Post-condición	Mapa cargado correctamente
Prioridad	Alta
Autor(es)	Juan Palma Borda Sergio Gavilán Prieto
Escenario principal	
1. El usuario selecciona los parámetros de la búsqueda. 2. El usuario pulsa el botón de filtrar por intervalo de tiempo 3. El sistema envía los parámetros de la búsqueda al servicio *busqueda_intervalo.* 4. Se limpian los parámetros. 5. Se prepara la consulta para la colección Denuncias. 6. Se realiza la búsqueda en MongoDB. 7. Se limpian los resultados y se preparan para el envío. 8. Se envían los resultados a la interfaz. 9. Se colorea el mapa con los datos recibidos.	
Clases de análisis	

A. Clases de entidad	-
B. Clases de control	*Servicios filtros y filtros_middle*
C. Clases de interfaz	*Controlador interfaz*

Maquetas de interfaz



Diagramas de secuencia

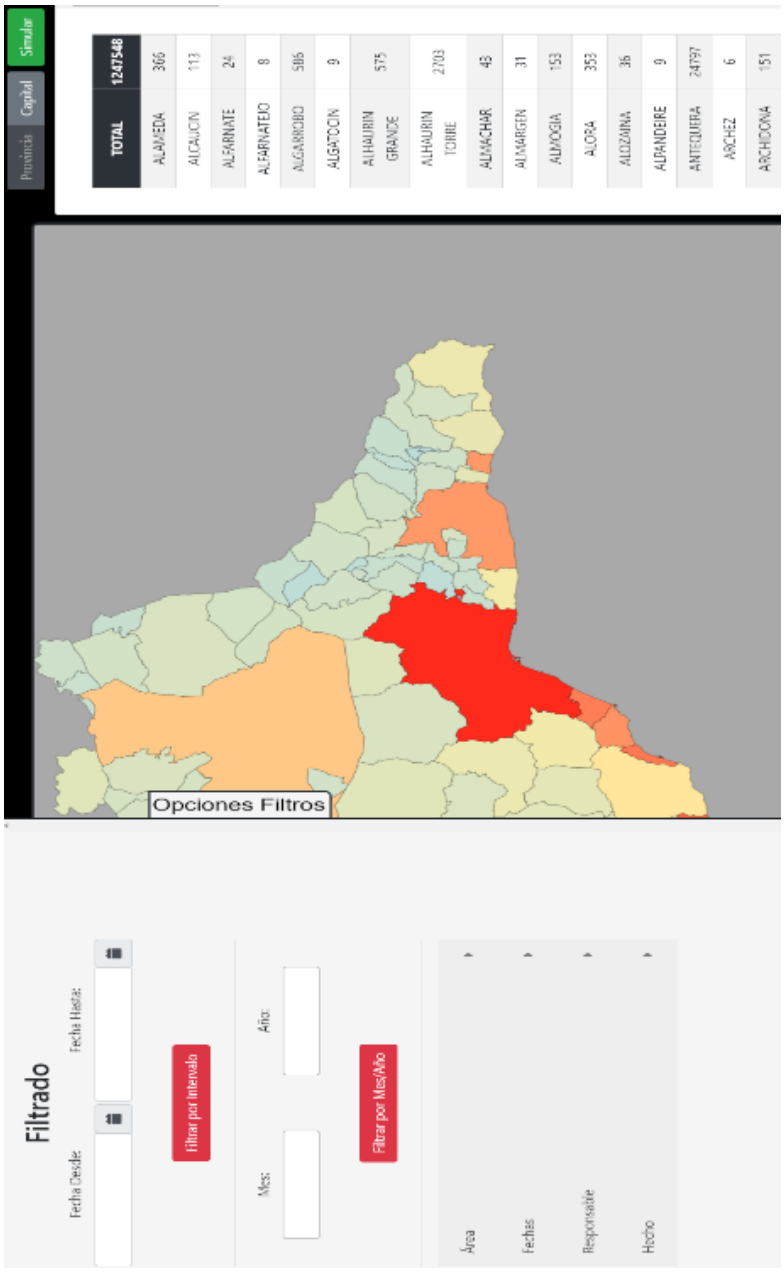


C.7 Búsqueda por intervalo de tiempo en el municipio de Málaga

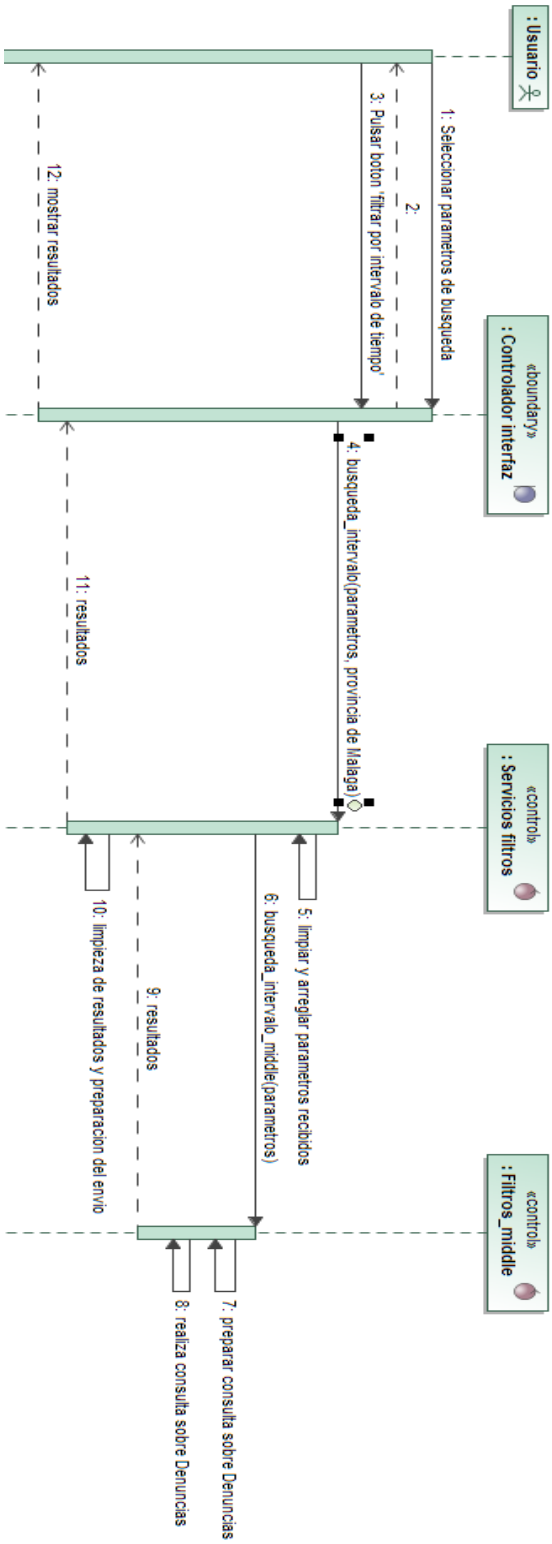
Título	Búsqueda por un intervalo de tiempo en la provincia de Málaga.
Descripción	Un usuario realiza una búsqueda filtrando por mes, año y/u otros parámetros para visualizar el resultado en el mapa de la provincia de Málaga.
Pre-condición	El usuario está viendo el mapa del municipio de Málaga.
Post-condición	Mapa cargado correctamente
Prioridad	Alta
Autor(es)	Juan Palma Borda Sergio Gavilán Prieto
Escenario principal	
1. El usuario selecciona los parámetros de la búsqueda. 2. El usuario pulsa el botón de filtrar por intervalo de tiempo. 3. El sistema envía los parámetros de la búsqueda al servicio *búsqueda_intervalo.* 4. Se limpian los parámetros. 5. Se prepara la consulta para la colección Denuncias. 6. Se realiza la búsqueda en MongoDB. 7. Se limpian los resultados y se preparan para el envío. 8. Se envían los resultados a la interfaz. 9. Se colorea el mapa con los datos recibidos.	
Clases de análisis	

A. Clases de entidad	-
B. Clases de control	*Servicios filtros y filtros_middle*
C. Clases de interfaz	*Controlador interfaz*

Maquetas de interfaz



Diagramas de secuencia



C.8 Ver estadísticas de un municipio de Málaga

Título	Ver estadísticas de un municipio de Málaga.
Descripción	Un usuario pulsa un municipio de la provincia de Málaga para ver las estadísticas de dichos distritos después de haber o no aplicado una búsqueda en el mapa.
Pre-condición	El usuario está viendo el mapa de la provincia de Málaga.
Post-condición	Estadísticas visibles en la parte derecha del mapa.
Prioridad	Alta
Autor(es)	Juan Palma Borda, Sergio Gavilán Prieto
Escenario principal	
1. El usuario pulsa en una provincia con más de un hecho. 2. El sistema envía los parámetros de la búsqueda al servicio *verZona.* 4. Se comprueba si es una búsqueda de DMA, de intervalo de tiempo o sin parámetros temporales. 5. Es una búsqueda de DMA 6. Se comprueba si es una búsqueda simple (un único parámetro que no sea ni de fecha, ni de lugar. 7. Es una búsqueda simple. 8. Se prepara la consulta para la colección Localizaciones. 9. Se realiza la búsqueda en MongoDB.	

10. Se limpian los resultados y se preparan para el envío.

11. Se envían los resultados a la interfaz.

Escenarios alternativos

1.a El usuario pulsa en un municipio vacío.

2.a Notificación de municipio vacío.

5.b Es una búsqueda de intervalo de tiempo.

6.b Se prepara la consulta para la colección Denuncias.

7.b Se realiza la búsqueda en MongoDB.

8.b Se limpian los resultados y se preparan para el envío.

9.b Se envían los resultados a la interfaz.

5.c Es una búsqueda sin parámetros temporales.

6.c Se comprueba si es una búsqueda simple (un único parámetro que no sea ni de fecha, ni de lugar.

7.c Es una búsqueda simple.

8.c Se prepara una búsqueda sobre Localizaciones.

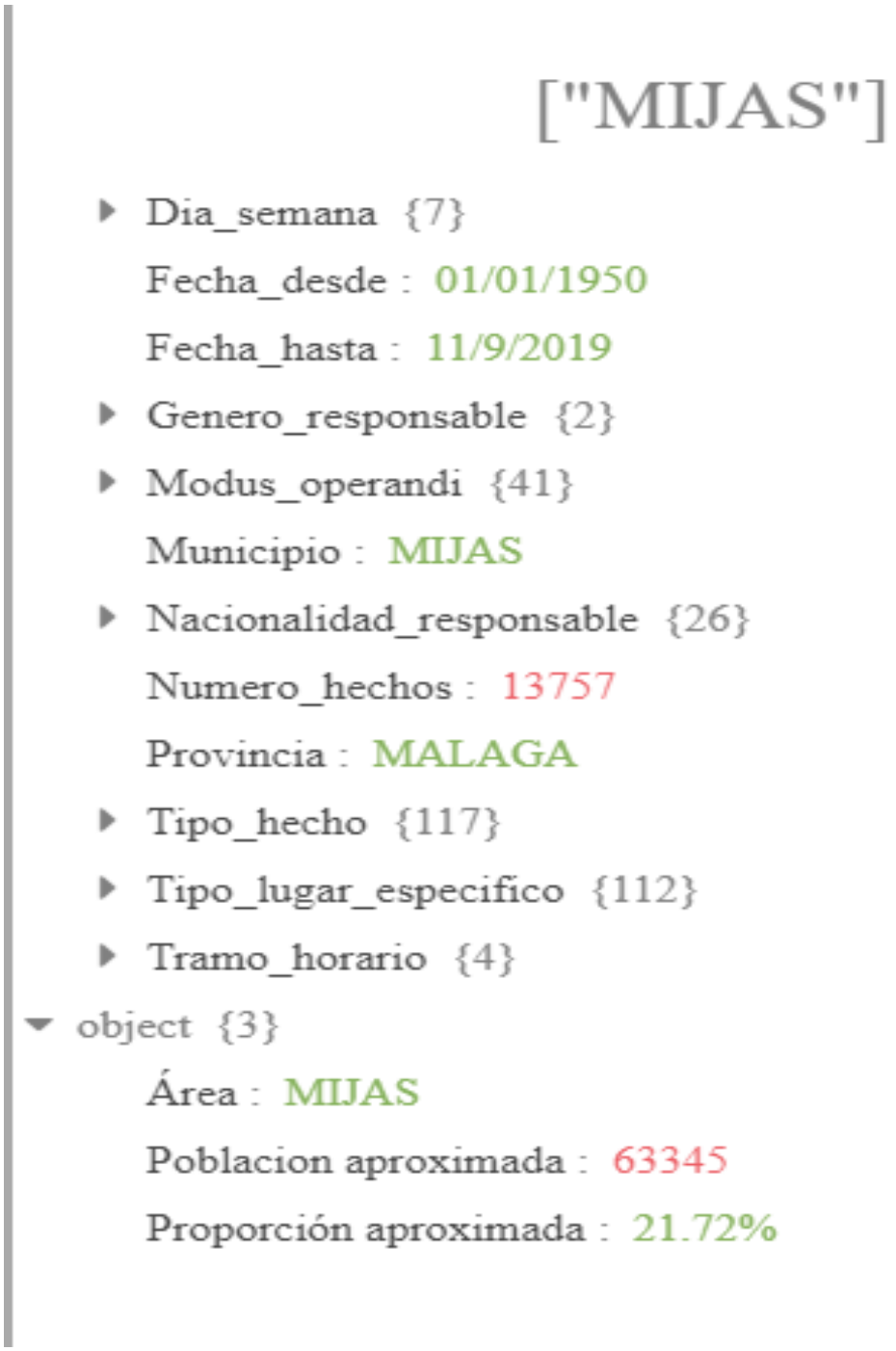
9.c Se realiza la búsqueda en MongoDB.

10.c Se limpian los resultados y se preparan para el envío.

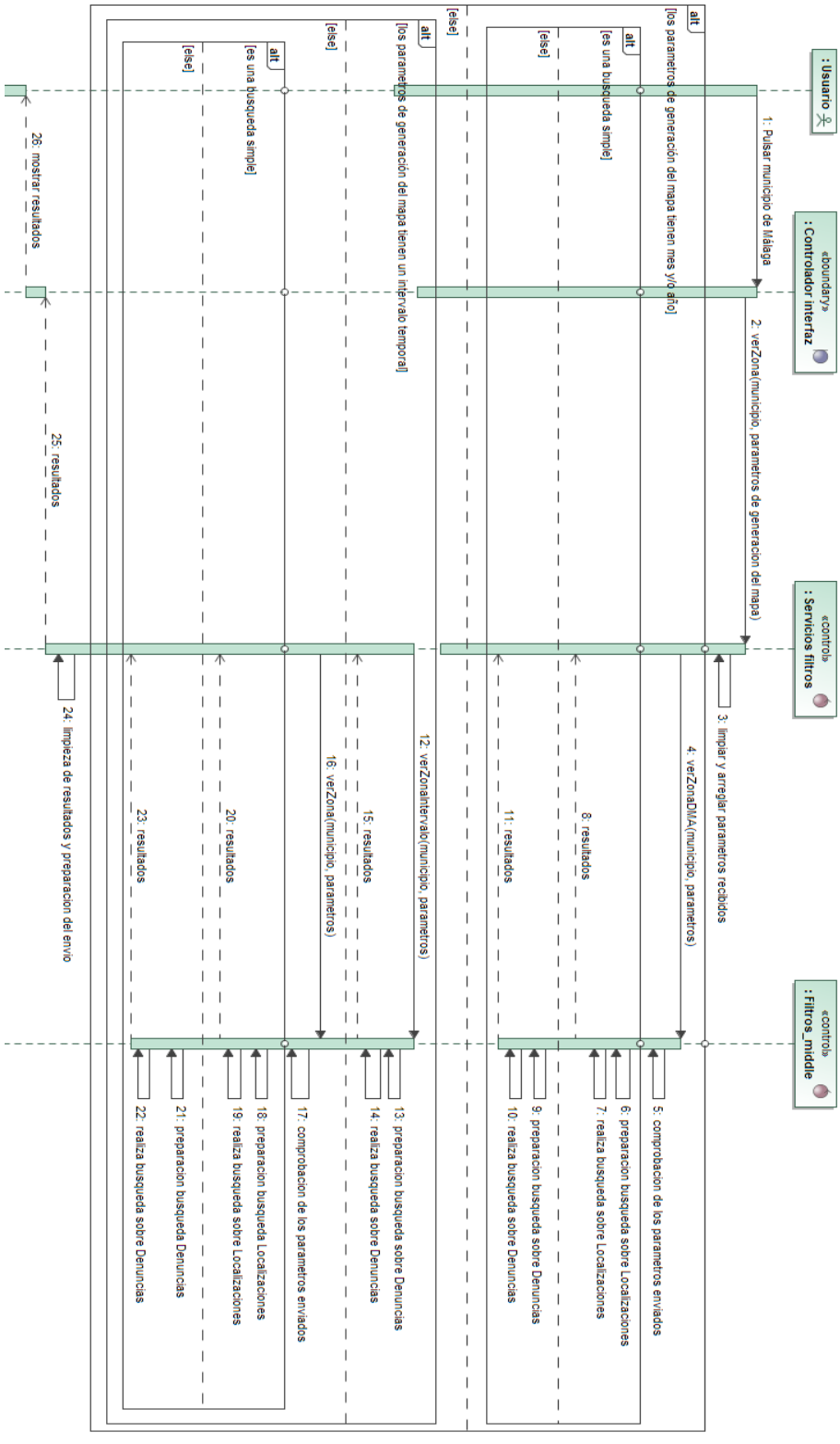
11.c Se envían los resultados a la interfaz.

7.c.a No es una búsqueda simple 8.c.a Se prepara una búsqueda sobre Denuncias. 9.c.a Se realiza la búsqueda en MongoDB. 10.c.a Se limpian los resultados y se preparan para el envío. 11.c.a Se envían los resultados a la interfaz. 7.d No es una búsqueda simple. 8.d Se prepara la consulta para la colección Denuncias. 9.d Se realiza la búsqueda en MongoDB. 10.d Se limpian los resultados y se preparan para el envío. 11.d Se envían los resultados a la interfaz.

Clases de análisis	
A. Clases de entidad	-
B. Clases de control	*Servicios filtros y filtros_middle*
C. Clases de interfaz	*Controlador interfaz*


["MIJAS"]
Dia_semana {7}
Fecha_desde : 01/01/1950
Fecha_hasta : 11/9/2019
Genero_responsable {2}
Modus_operandi {41}
Municipio : MIJAS
Nacionalidad_responsable {26}
Numero_hechos : 13757
Provincia : MALAGA
Tipo_hecho {117}
Tipo_lugar_especifico {112}
Tramo_horario {4}
object {3}
Área : MIJAS
Poblacion aproximada : 63345
Proporción aproximada : 21.72%

Diagramas de secuencia



C.9 Ver estadísticas de un distrito del municipio de Málaga

Título	Ver estadísticas de un distrito del municipio de Málaga.
Descripción	Un usuario pulsa un distrito del municipio de Málaga para ver las estadísticas de dichos distritos después de haber o no aplicado una búsqueda en el mapa.
Pre-condición	El usuario está viendo el mapa del municipio de Málaga.
Post-condición	Estadísticas visibles en la parte derecha del mapa.
Prioridad	Alta
Autor(es)	Juan Palma Borda, Sergio Gavilán Prieto
Escenario principal	
1. El usuario pulsa en un distrito con más de un hecho. 2. El sistema envía los parámetros de la búsqueda al servicio *verZona*. 4. Se comprueba si es una búsqueda de DMA, de intervalo de tiempo o sin parámetros temporales. 5. Es una búsqueda de DMA 6. Se comprueba si es una búsqueda simple (un único parámetro que no sea ni de fecha, ni de lugar. 7. Es una búsqueda simple. 8. Se prepara la consulta para la colección Localizaciones. 9. Se realiza la búsqueda en MongoDB.	
10. Se limpian los resultados y se preparan para el envío. 11. Se envían los resultados a la interfaz.	

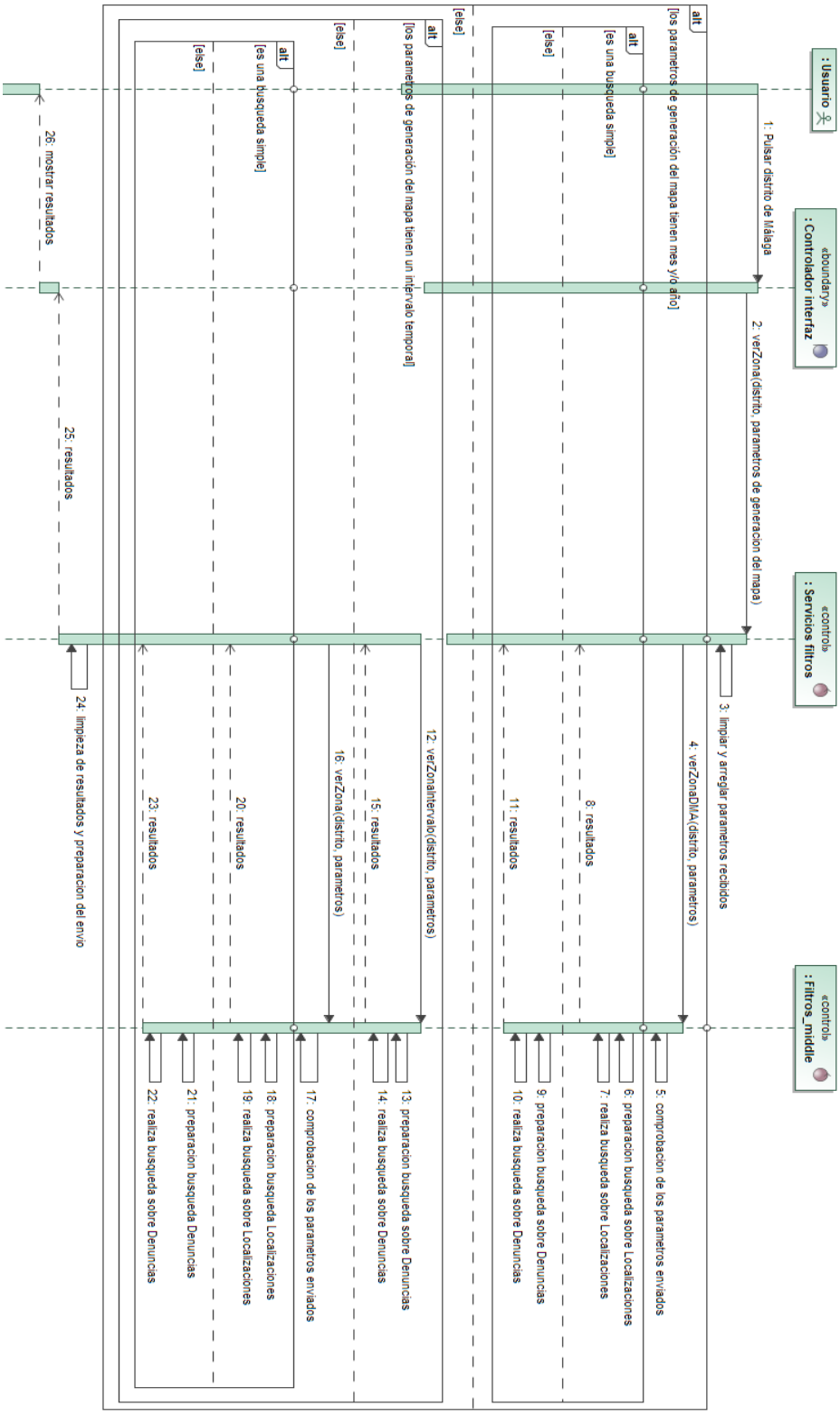
Escenario alternativo
1.a El usuario pulsa en un distrito vacío. 2.a Notificación de distrito vacío. 5.b Es una búsqueda de intervalo de tiempo. 6.b Se prepara la consulta para la colección Denuncias. 7.b Se realiza la búsqueda en MongoDB. 8.b Se limpian los resultados y se preparan para el envío. 9.b Se envían los resultados a la interfaz. 5.c Es una búsqueda sin parámetros temporales. 6.c Se comprueba si es una búsqueda simple (un único parámetro que no sea ni de fecha, ni de lugar. 7.c Es una búsqueda simple. 8.c Se prepara una búsqueda sobre Localizaciones. 9.c Se realiza la búsqueda en MongoDB. 10.c Se limpian los resultados y se preparan para el envío. 11.c Se envían los resultados a la interfaz.

7.c.a No es una búsqueda simple 8.c.a Se prepara una búsqueda sobre Denuncias. 9.c.a Se realiza la búsqueda en MongoDB. 10.c.a Se limpian los resultados y se preparan para el envío. 11.c.a Se envían los resultados a la interfaz. 7.d No es una búsqueda simple. 8.d Se prepara la consulta para la colección Denuncias. 9.d Se realiza la búsqueda en MongoDB. 10.d Se limpian los resultados y se preparan para el envío. 11.d Se envían los resultados a la interfaz.

Clases de análisis	
A. Clases de entidad	-
B. Clases de control	*Servicios filtros y filtros_middle*
C. Clases de interfaz	*Controlador interfaz*


["CIUDAD JARDIN"]
Dia_semana {7}
Distrito_municipal : Ciudad Jardin
Fecha_desde : 01/01/1950
Fecha_hasta : 11/9/2019
Genero_responsable {2}
Modus_operandi {41}
Municipio : MALAGA
Nacionalidad_responsable {17}
Numero_hechos : 16367
Provincia : MALAGA
Tipo_hecho {142}
Tipo_lugar_especifico {109}
Tramo_horario {4}

Diagramas de secuencia



Apéndice D

Acrónimos

HTML	HyperText Makup Language
CSS	Cascading Style Sheets
JS	JavaScript
D3	Data-Driven Documents
JSON	JavaScript Object Notation
BSON	Binary JSON
CSV	Comma-separated values
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SQL	Structured Query Language
NoSQL	Not only SQL
BD/DB	Base de Datos
IDE	Integrated Development Environment
REST	Representational State Transfer
ASCII	American Standard Code for Information Interchange

BOM	Byte Order Mark
UTF	Unicode Transformation Format
UTF-8-SIG	UTF de 8 bytes con BOM
ANTLR4	ANother Tool for Language Recognition 4
ISO	International Organization for Standardization
UML	Unified Modeling Language
MIT	Massachusetts Institute of Technology
BSD	Berkeley Software Distribution
DOM	Modelo de Objetos del Documento (Document Object Model)
URL	Identificador de recursos uniforme (Uniform Resource Locator)
SSL	Secure Socket Layer
SVG	Scalable Vector Graphics
JWT	JSON Web Token