ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DEL SOFTWARE

PLATAFORMA PARA LA PREVENCIÓN DE DELITOS

SIMULACIÓN DE ESCENARIOS Y SISTEMAS MULTIAGENTE

CRIME PREVENTION PLATFORM

STAGE SIMULATION AND MULTIAGENT SYSTEMS

Realizado por
SERGIO GAVILÁN PRIETO

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMRE DE 2019

Fecha defensa: septiembre de 2019

Fdo. El/la Secretario/a del Tribunal

Resumen

Con el fin de analizar la situación de los delitos ocurridos en Málaga y con la idea de disminuir su porcentaje, surge este TFG cuyo objetivo principal es desarrollar una plataforma web para el análisis y la prevención de los delitos.

Para ello, a partir de un conjunto de datos históricos, se podrá ver en la plataforma un mapa de calor con los delitos más comunes en cada zona y sus características. Además, se crearán modelos basados en agentes con los delitos más comunes y sus características, tales como hora, lugar y modus operandi, ayudando así al Cuerpo Nacional de Policía de la localidad de Málaga a distribuir mejor sus patrullas y disminuir el crimen.

Palabras clave: aplicación web, prevención de crímenes, sistemas multiagente, Python.

Abstract

In order to analyze the crime situation in Málaga and decreasing its percentage, this Final Degree Project's main objective is to develop a web platform for their analysis and prevention.

In order to do so, this web platform takes historical criminal data and colors a heat map based on the number of occurrences of crimes in each zone. Furthermore, it creates agent-based models in order to predict future crimes, associating them with a certain hour, place and modus operandi, helping the National Police Corps of Málaga arrange their troupes in a more efficient way, improving, in that way, their field performance.

Keywords: web application, crime prevention, multiagent systems, Python.

Índice

1

Introducción

1.1 Motivación

Unos de los temas que más preocupa a los ciudadanos de a pie hoy día es su seguridad, por tanto, es imprescindible que las actuaciones policiales sean lo más eficaces posible. Para ello es primordial que los cuerpos de seguridad sepan organizar sus tropas y recursos de la mejor manera posible, sabiendo dónde es más potencial cada tipo de delito.

1.2 Objetivos

El principal objetivo de este Trabajo de Fin de Grado (TFG) ha sido proporcionar al Cuerpo Nacional de Policía (CNP) un entorno visual con el fin de poder identificar de forma clara y precisa las zonas más conflictivas en datos empíricos.

Para ello, se creará una plataforma web capaz de proporcionar un entorno visual basado en mapas de calor, coloreados a partir de datos extraídos directamente de la Policía Nacional. Además, permitirá la simulación de posibles futuros eventos y su lugar de ocurrencia, así como la lectura de datos oficiales generados por el CNP.

El proyecto ha sido **desarrollado en grupo**, por lo que existen 3 partes claramente diferenciadas: procesamiento de datos, entorno visual y simulación basada en sistemas multiagente, basándose el presente TFG en esta última parte.

Para ello, es necesario crear una estructura bien conectada que conste de:

- Datos estandarizados capaces de representar un delito y sus características, tales como lugar, hora, responsable y tipo de hecho.
- Mapas sobre los que poder dibujar y visualizar datos de forma sencilla.
- Servicios RESTful que reciban y comuniquen de forma eficaz y segura los datos.
- Y un módulo para la simulación por agentes.

Cabe destacar también que este TFG se ha desarrollado bajo el marco de un convenio entre la Universidad de Málaga y el CNP y, por esta razón, se han mantenido diversas reuniones con responsables de este cuerpo de seguridad para realizar una captura de requisitos del proyecto.

1.3 Antecedentes

Antes de comenzar a desarrollar la aplicación se llevó a cabo un estudio sobre el estado de arte en el campo de la prevención de delitos. En él destacan la aplicación *X-Law*, el Trabajo de Fin de Máster *Investigación y análisis de*

crímenes en las ciudades de Chicago y Los Ángeles y el *opendata* del Ayuntamiento de Málaga.

Por un lado, el sistema *X-Law* es una aplicación diseñada y desarrollada por un miembro del cuerpo de Policía de la ciudad de Mestre, que ya ha ayudado a reducir en más de un 22% el crimen en esta ciudad. Su funcionamiento consiste en, a partir de datos históricos, ver que zonas son más conflictivas y cuándo lo son, de forma que se lleve a cabo una distribución más eficiente de las patrullas, anticipándose así a los criminales.

El Trabajo de Fin de Máster *Investigación y análisis de crímenes en las ciudades de Chicago y Los Ángeles* desarrollado por Adrián García Humanes, antiguo estudiante de la Universidad de Málaga, se basa en el estudio de los datos de crímenes ofrecidos por los *opendata* de las ciudades de Chicago y Los Ángeles, de forma que se pueda realizar un análisis sencillo a nivel visual de los mismos.

Por otro lado, el *opendata* del Ayuntamiento de Málaga ha sido crucial para desarrollar nuestro sistema y la organización de los datos y mapas de calor.

1.4 Metodología

Con esta finalidad en mente, la metodología elegida para conseguir los objetivos se basó en *Scrum*. Veamos qué es exactamente *Scrum*, y, adicionalmente también, qué es una metodología.

Según el Diccionario de la Real Academia de la Lengua Española, una metodología es un conjunto de métodos que se siguen en una investigación

científica o en una exposición doctrinal. En el mundo de la informática tradicionalmente se han seguido una serie de metodologías muy estructuradas y estrictas, tales como la metodología en cascada o la metodología en espiral, que dan mucha importancia a la planificación inicial del proyecto y que dependen en gran medida de que no pueda haber cambios a lo largo del mismo, dependiendo así mucho del análisis de riesgos.

Las metodologías ágiles para el desarrollo software están basadas en iteraciones, cada una de ellas con 6 fases completas: Planificación, Análisis de Requisitos, Diseño, Implementación, Pruebas y Documentación. El objetivo de esto es, partiendo de un subconjunto de los requisitos totales, obtener el mínimo producto viable en cada iteración, de forma que cada iteración complemente a la anterior y el proyecto o sistema avance de forma continua.

Esto requiere a su vez de una mayor implicación del cliente en el proyecto, con el fin de que apruebe cada prototipo y pueda resolver cualquier duda sobre la especificación de los requisitos. Por tanto, se trata de una metodología de desarrollo software enfocada al trabajo en equipo y la comunicación continua tanto entre sus miembros como entre los mismos y el cliente.

Las principales características que hacen que Scrum se diferencie frente al resto de metodologías ágiles, son la existencia de equipos auto-dirigidos y auto-organizados; reuniones diarias sobre las tareas del equipo y sus posibles complicaciones y la existencia de tres roles principales: *Product Owner*, *ScrumMaster* y Equipo de desarrollo, cada uno con su función característica.

Además, *Scrum* cuenta con varias fases propias, como son Sprint (similar a una iteración), Planificación del Sprint, *Scrum* diario (reuniones diarias entre los distintos integrantes), Revisión y Retrospectiva del Sprint, y cuenta a su vez

con distintos tipos de documentación, a nivel global: el *Product Backlog* (descripción de alto nivel de las tareas del producto global) y el Sprint Backlog (tareas que se realizarán en el sprint), entre otros. Estos se suelen estructurar en un tablero o pizarra denominado *Scrum Taskboard* que permite saber en cualquier momento el estado de las tareas definidas en ellos.

La variante realizada para este proyecto no incluye los diferentes tipos de roles, pues el equipo constaba únicamente de tres integrantes que realizaban los roles de equipo de desarrollo y *ScrumMaster*. En cambio, se adoptaron las ideas de *Product Backlog* y *Sprint Backlog* para la organización de las tareas en iteraciones, organizadas en el *Scrum Taskboard*.

En total se llevaron a cabo 4 iteraciones: iteraciones 0 - 3, actuando la iteración 0 como introductoria al resto, donde se llevó a cabo la investigación de las tecnologías a utilizar, la toma inicial de requisitos, la interpretación inicial de los datos y el modelado del sistema. Las iteraciones 1, 2 y 3 sirvieron para la implementación del sistema.

1.5 Tecnologías empleadas

Las principales tecnologías empleadas han sido las siguientes: Python, MongoDB, JWT y SQLite.

Python es utilizado como lenguaje base para desarrollo de todo el proyecto, aunque, en el marco de este TFG, el compañero Alberto Ramírez Mena utiliza JavaScript para la parte del front-end.

Para el almacenamiento masivo de datos se utiliza MongoDB, mientras que para almacenar a los usuarios del sistema se utiliza SQLite.

Por último, para proporcionar seguridad al sistema se ha utilizado JWT (JSON Web Tokens) para el intercambio de tokens únicos entre el cliente y servidor.

1.6 Estructura de la memoria

El cuerpo de la memoria está dividido en 4 iteraciones, una por cada iteración. Además, en las iteraciones 0, 1 y 2 hubo una reunión con responsables de la Policía Nacional en cada una, con el fin de guiar el desarrollo.

A continuación, se explica brevemente el contenido de cada una de ellas:

- **Iteración 0:** investigación de las tecnologías que se utilizarán en el desarrollo y primera toma de requisitos.
- **Iteración 1:** creación de los primeros filtros para las búsquedas de delitos y primeras configuraciones de seguridad y usuarios.
- **Iteración 2:** creación de modelos basados en agentes básicos, basados en la infraestructura del momento, automatización del primer arranque del servidor y establecimiento de la comunicación entre el cliente y el servidor. Así mismo, se llevó a cabo un análisis del coste de los servicios para la aplicación de filtros y su correspondiente reestructuración.
- **Iteración 3:** profundización en el diseño de los agentes y creación de una infraestructura escalable para futuras modificaciones, configuración de seguridad en el cliente y creación de administradores para el uso del sistema.

- **Conclusiones y trabajos futuros:** resumen de los objetivos que se han conseguido en el proyecto, así como los futuros desarrollos que podría ser interesante realizar.

2

Iteración 0

2.1 Introducción

La iteración 0 sirve como base para el estudio de las tecnologías empleadas y como una primera toma de requisitos sobre los que poder diseñar el sistema.

2.2 Desarrollo

El desarrollo de la Iteración 0 estará divido en 5 secciones: investigación, requisitos del sistema, modelado del sistema, formato de datos y primera reunión con la Policía Nacional.

2.2.1 Investigación

Las tecnologías que necesitaron investigación fueron: Python, Flask, MESA, MongoDB, JWT y Heroku.

Por un lado, Python conllevó la creación de varios proyectos básicos, partiendo desde el básico *Hello World* hasta llegar a crear el proyecto actual. La curva de aprendizaje no fue muy costosa, ya que se trata de un lenguaje sencillo y capaz de ejecutar cualquier conjunto de comandos sin necesidad más allá del intérprete Python. Esto sería clave en futuras iteraciones a la hora de realizar pruebas para todos los métodos internos y servicios.

Por otro lado, tenemos Flask, un Framework bajo la licencia BSD para la creación de microservicios orientado a Python. La estructura de sus servicios es sencilla y flexible, permitiendo varios modos de escritura.

```
@cabeceras_bp.route('/inicio', methods={'GET'})
@login_required
def get_cabeceras():
    """..."""
    provincia = request.values.get("provincia")
    res = obtener_cabeceras(provincia=provincia)
    if len(res) > 0:
        for x, y in res.items():
            if type(y) is list:
                y.sort()
    return make_response(jsonify(res), 200)
```

Imagen 2.2.1.1: Estructura de un servicio REST en Flask

Cabe destacar que Flask no cuenta con capas para la conexión con base de datos u otro estilo de características existentes en otros Frameworks para el desarrollo de servicios, como sería Django, por ejemplo. Para ello utilizamos

como respaldo librerías de terceros que serán mencionadas más adelante y que aportaban al conjunto mayor libertad de acción. Es por ello por lo que Flask fue el Framework elegido para trabajar.

Así mismo nos encontramos con MongoDB, una base de datos NoSQL, es decir, una base de datos enfocada a manejar grandes cantidades de datos sin dejar de lado el rendimiento y la escalabilidad de esta. Esto hace de MongoDB un ecosistema perfecto para el almacenamiento y búsqueda de los datos procedentes del CNP, pues los archivos CSV que sirven de entrada para el sistema son de gran escala.

MongoDB está estructurado en bases de datos, que a su vez se estructuran en colecciones las cuales contienen documentos. Una de las ventajas de este tipo de base de datos, es que la estructura de los documentos no tiene por qué ser consistente, es decir, dos documentos dentro de la misma colección no tienen por qué tener los mimos campos. Esto dotó a la aplicación de mayor dinamismo a la hora de hacer los algoritmos de lectura de ficheros.

Para conectar Flask con MongoDB se utilizó PyMongo, la librería de Python recomendada por los creadores de MongoDB para trabajar en este lenguaje. Se basa en el uso de diccionarios y permite la creación de varios tipos de consultas, como pueden ser un *find* o *count*, junto con operadores como *group*, *max* o *min*, que agrupan y devuelven el máximo y el mínimo respectivamente. Su escritura es sencilla y permite la realización de varias consultas simultáneas, abriendo únicamente una conexión a la base de datos para todas ellas.

Por otro lado, MESA es un Framework para Python destinado al modelado de sistemas multiagente. Tiene dos componentes principales: modelos y agentes. Los modelos son las entidades encargadas de manejar a los agentes y su comportamiento, mientras que los agentes son representaciones dentro de un conjunto (modelo). Cada agente y modelos pueden tener tanto atributos como se desee, de forma que el resultado sea un sistema que modele y represente el fin deseado.

Los agentes pueden interactuar entre ellos y moverse en un espacio toroidal, pero para nuestro sistema no fue necesario esto último, pero sí que cada agente tuviera una serie de características que lo hicieran único y capaz de identificar un delito potencial en una zona específica.

En cuanto a la seguridad de la API y sus recursos, se utilizó JWT (JSON Web Token), de acuerdo con el estándar RFC 7519, de forma que compartir información entre el front y back-end fuera seguro.

El JWT contiene un conjunto de información que es generada por el servidor y posteriormente almacenada por el front-end, y debe ser validada por servidor de nuevo en cada petición que reciba. Principalmente, suele contener su fecha de expedición, fecha de caducidad y el sujeto al que pertenece. Toda esta información va encriptada con una clave única del servidor, de forma que sólo él podrá decodificarla.

Por último, utilizamos a Heroku, una plataforma en la nube para el despliegue de aplicaciones web con soporte para varios lenguajes, entre los que se hayan Python, Java y PHP, entre otros. Para probar su funcionamiento, se creó una aplicación básica y se desplegó.

2.2.2 Requisitos del Sistema

En un principio, la toma de requisitos globales del sistema reflejó que éste debía ser una plataforma que permitiese visualizar, mediante el uso de mapas de calor, dónde y cuándo era más común cada tipo de delito y sus características. También quedó claro que la entrada de datos al sistema se haría mediante el formato de datos CSV

Por otro lado, la forma en la que el usuario podría aplicar filtros para visualizar los datos y cuáles serían estos, no quedó claro. Se trataba de algo que se iría cambiando en las siguientes iteraciones, pues al ser una aplicación novedosa para el CNP de Málaga, no tenían del todo claro qué iba a ser necesario y qué no.

Esta toma de requisitos fue suficiente para poder trabajar en el lector de archivos, la interfaz y los servicios REST que darían vida al sistema.

2.2.3 Formato de datos

En un principio, los archivos CSV procesados generarían un documento por cada denuncia, que se almacenarían en una colección de MongoDB llamada **denuncias**. Estos documentos contendrían toda la información perteneciente a una denuncia en específico, tal como el municipio y lugar donde ocurrió, como puede ser una farmacia; el tipo de hecho que fue, por ejemplo, un hurto o una amenaza; los responsables involucrados y las víctimas, cada uno con su edad,

sexo, nacionalidad, estado civil y cualquier dato que se hubiese considerado relevante en el momento de la recogida de datos.

Las denuncias también contenían el modus operandi utilizado, la fecha de ocurrencia, los objetos involucrados (por ejemplo, ballestas, cuchillos o teléfonos móviles) y toda la información sobre la plantilla de actuación.

Pero denuncias no era una buena idea para ver datos de estadística generales, tales como el número de ocurrencias de cada tipo de hecho o el número de responsables de cierta edad en un año y municipios en concreto. Por ello, se creó una segunda colección llamada **localizaciones** que se encargaría de tener una entrada por cada mes de cada año de cada municipio y provincia, recogiendo el total de cada tipo de hecho, lugar, característica de responsables, modus operandi, días de la semana y tramos horarios que habían ocurrido en dicho lugar en dicho tiempo.

Esta colección sería de gran utilidad en el futuro cuando el usuario quisiera ver una estadística de los tipos de hechos en Málaga o Fuengirola en todo el histórico, por ejemplo.

Cada entrada en la tabla localizaciones tendría un identificador único del tipo: **Año-Mes#Provincia_Municipio** para poder acceder a los datos específicos de forma rápida desde Python.

Por último, más adelante surgiría la idea de crear otra colección auxiliar llamada **cabeceras**, que contendría una entrada por cada provincia con todos los nombres de cada hecho cometido y su lugar, modus operandi y municipios de ocurrencia, junto con las nacionalidades implicadas. De esta forma, la

interfaz gráfica podría obtener de forma dinámica esta información según se va actualizando la base de datos. Es decir, no es posible buscar un hecho que no esté registrado en el sistema, pues el front-end no lo permite. Toda la información que puede buscar el usuario está controlada.

Ambas colecciones auxiliares se irían actualizando con cada importación nueva de datos, de forma que todo estuviese en orden.

Por otro lado, para el tratamiento de datos de los agentes y los JWT se usa una base de datos relacional, pequeña y portable, como es SQLite, pues los datos almacenados son de pequeño tamaño y no requieren un acceso tan constante como los de MongoDB.

2.2.4 Modelado del Sistema

En esta fase se llevaron a cabo importantes decisiones de diseño, tales como el uso de una base de datos no relacional y la estructuración del sistema.

Como ya hemos visto, el uso de una base de datos relacional no sería una decisión correcta, ya que los datos de entrada pueden ser variables, es decir, puede que en el futuro intenten introducir en la aplicación un campo que antes no existía, lo que conllevaría a cambiar parte del sistema. Pero al tratarse de una base de datos NoSQL, esta se adapta sin problema a cualquier cambio de este tipo.

Por otro lado, el sistema quedó dividido en seis módulos:

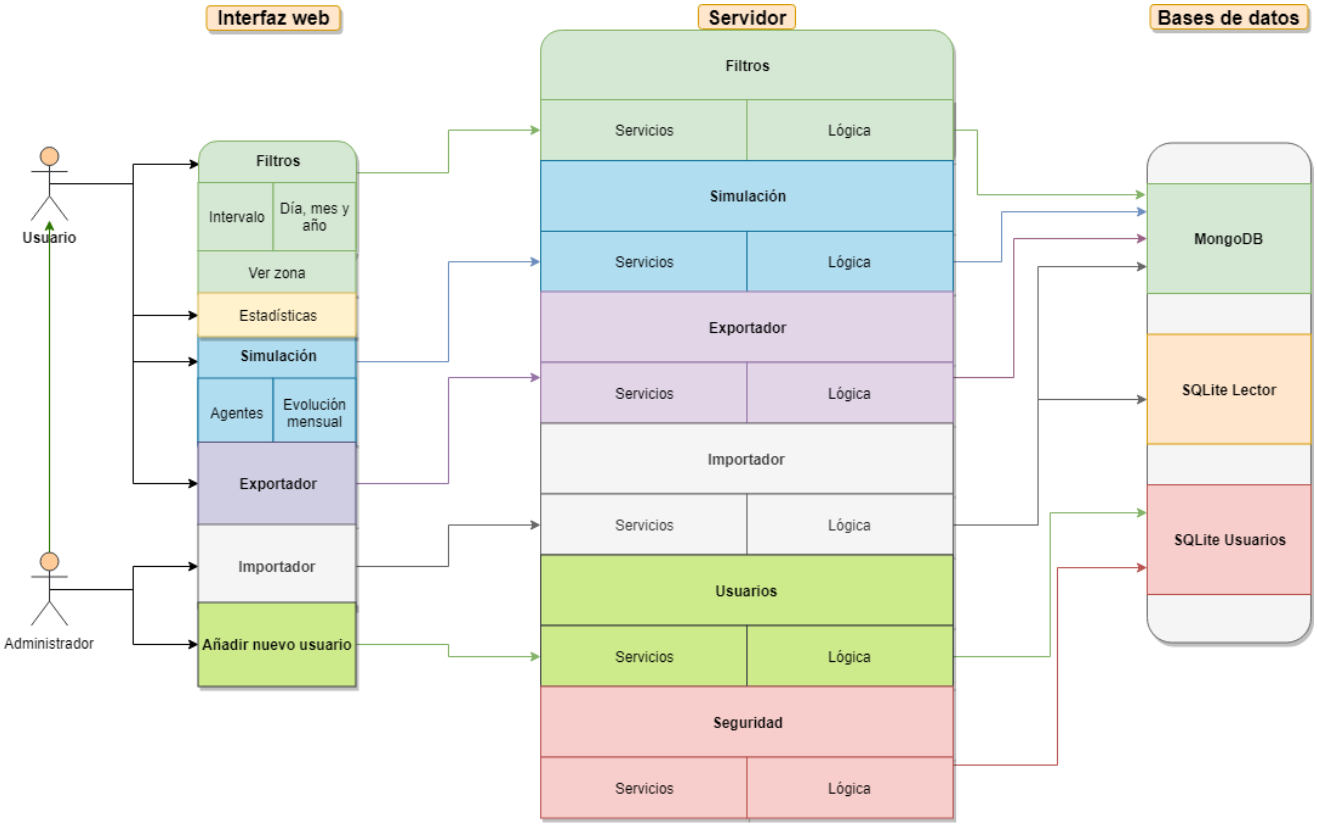


Imagen 2.2.4.1: Estructura del sistema y funcionalidades principales

- Lector: encargado de leer los datos, pre-procesarlos e insertarlos en la base de datos.
- Filtros: encargados de filtrar en la base de datos de acuerdo con los parámetros elegidos por el usuario.
- Interfaz gráfica: encargada de interpretar la información recibida del servidor y hacerla entendible al usuario.
- Agentes: encargados de simular futuros eventos.
- Seguridad: encargado de proporcionar una comunicación segura punto a punto entre *front* y *back-end.*
- Usuarios: encargado de la gestión de usuarios de la base de datos.

Estos 2 últimos módulos son más reducidos que los 4 anteriores, como veremos más adelante.

2.2.5 Primera reunión con el Cuerpo Nacional de Policía de Málaga

La primera reunión con la policía tuvo lugar el día 12 de marzo de 2019. En ella se presentó un primer diseño de la interfaz, realizado por Alberto Ramírez Mena, en base a unos requisitos iniciales suministrados por el tutor del TFG.

La primera impresión de la interfaz resultó positiva y se habló de cómo se podrían aplicar los filtros y cuáles serían estos. La policía destacó los siguientes atributos de entre todos los contenidos en lo archivos CSV suministrados de manera previa a la reunión:

- Tipo de hecho
- Lugar de ocurrencia
- Modus operandi
- Día de la semana
- Nacionalidad del responsable
- Sexo del responsable

También se habló, por tanto, de los datos y cómo se tratarían estos. Se llegó al acuerdo de un identificador para los datos a partir de ciertas columnas de los CSV generados por ellos.

Así mismo, se habló de la posibilidad de distinguir varios roles en el sistema: un usuario básico con acceso a las utilidades de filtros de la aplicación

y un usuario administrador encargado de añadir nuevos datos al sistema, idea ante la cual la policía se mostraba reacia en un principio, pero llegamos a la conclusión de que tenía más sentido tener controlado quién añadía datos al sistema y quién no.

3

Iteración 1

3.1 Introducción

La Iteración 1 sirvió como base para los servicios de filtros y usuarios. Así mismo para la seguridad.

3.2 Desarrollo

El desarrollo de la Iteración 1 estará divido, por ende, en 4 bloques: filtros, usuarios, seguridad y segunda reunión el cuerpo de policía.

3.2.1 Filtros

3.2.1.1 Primera versión de los filtros

La primera versión de los filtros contaba con 4 servicios principales:

- Servicio para la carga inicial del mapa de Málaga.
- Filtro por intervalo: destinado al uso de filtros junto con un rango de fechas, como, por ejemplo: 12/02/2018 - 23/05/2018.
- Filtro por día, mes y año: destinado a la búsqueda en varios años, varios meses o un mes dentro de uno o varios años. Su propósito inicial también incluía la búsqueda de uno o varios días de la semana junto con los otros parámetros.
- Filtro para funciones estadísticas: su finalidad era poder ver varios días, meses o años (sin combinaciones entre ellos) dentro de todo el historial de datos, pero en un futuro sería desechado y los filtros DMA se ajustarían a estas características.

La existencia del filtro para funciones estadística estaba justificada en este punto, pues la policía tenía claro que quería funciones estadísticas en el sistema, pero no cómo, por lo que se pensó habilitar un único servicio para ello y que se expandiese con el paso del tiempo.

Para la conexión con MongoDB se optó por crear una capa intermedia, de forma que los servicios únicamente reciban los datos y se encarguen de manejar errores generales, como errores en los parámetros recibidos. Cualquier otra acción será delegada al archivo **filtros_middle.py**.

3.2.1.2 Capa de persistencia

La característica más importante de estas capas de persistencia es la existencia de 2 conexiones con MongoDB:

- **denuncias**: para la colección denuncias
- **localizaciones**: para la colección localizaciones

Ambas se encuentran alojadas en el archivo **mongo_manager.py**.

Esta capa intermedia (filtros_middle) y sus archivos asociados (filtros_util) hacen uso de esta clara distinción entre colecciones para distinguir sus métodos.

Existe uno o más métodos intermedios para cada filtro mencionado. Para el caso de filtros por día, mes y año existen 2 métodos intermedios: uno para buscar en denuncias y otro para buscar en localizaciones, dependiendo del tipo de consulta. Si los parámetros incluyen únicamente la fecha, la consulta se hace a localizaciones; en cualquier otro caso, a denuncias.

En cambio, para los filtros por intervalo, las consultas siempre van a la colección denuncias, pues permiten buscar por un día, mes y año específicos en rango.

En esta primera versión de los filtros, todas las consultas se realizaban bajo la función **find** de PyMongo, y las consultas cruzadas, es decir, con varios

tipos de hecho u otras características, se realizaban bajo el operador **$or** de MongoDB.

Por ejemplo, supongamos que deseamos buscar los hurtos y amenazas cometidas en la ciudad de Málaga en el mes de agosto. La consulta sería la siguiente:

db.denuncias.find({"Hecho.Lugar.Provincia" : "MALAGA", "Hecho.Lugar.Municipio" : "MALAGA", "Hecho.Fecha.Mes" : 8, "$or" : [{"Hecho.Tipos" : "HURTO"}, {"Hecho.Tipos" : "AMENAZAS"}]})

Esto nos devuelve todos los documentos, íntegros, encontrados que cumplan esas características. Como ya habíamos dicho, esta consulta se realiza en denuncias, aunque sea de un mes únicamente, ya que, aparte de la fecha (agosto), contiene un filtro para hecho.

Imagen 3.2.1.2.1: Estructura del objeto Hecho en un documento de la colección denuncia

Por otro lado, la carga de mapa inicial de la provincia de Málaga se hace sobre la colección localizaciones, ya que existe un único documento que recoge toda la información histórica de la provincia. Esto resulta en una gran eficiencia en el inicio de la aplicación, pues estamos hablando de algo instantáneo, puesto que la búsqueda se realiza bajo el identificador único, en este caso, **#MALAGA.**

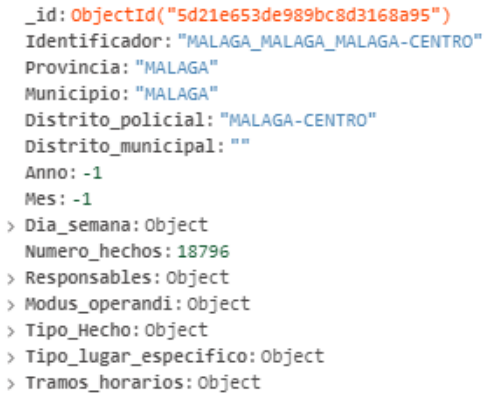


Imagen 3.2.1.2.2: Estructura de un documento de la colección localizaciones

3.2.2 Usuarios

Un usuario viene representado por las siguientes características:

- Identificador único
- Nombre de usuario único
- Correo electrónico (añadido en la 3ª iteración)
- Contraseña
- Rol

Existen dos tipos de roles:

- ***ROLE_USER***: para usuarios normales, sin privilegios.

- ***ROLE_ADMIN***: con privilegios de administrador, cuyas ventajas son: añadir datos nuevos al sistema, crear nuevos usuarios y borrar la base de datos.

Tanto el nombre de usuario como su contraseña se encuentran almacenados bajo un hash **SHA-3** de 256 bits para mayor seguridad. El identificador y rol se almacenan en texto plano, mientras que el email estará codificado en base 64.

En este punto existían únicamente dos servicios para los usuarios: Login y Logout.

El Login se encargaba de comprobar que los datos recibidos del front-end correspondían con un usuario registrado en la base de datos, y en caso correcto, le asignaba un token de autenticación y uno de refresco que el front-end debería almacenar para futuras comunicaciones.

El Logout se encarga, en cambio, de almacenar los tokens de autenticación y refresco en una lista negra para que evitar accesos no deseados.

En su capa de persistencia, los usuarios se buscan mediante nombre de usuario, y si éste coincide, se compararán las contraseñas. Para ello, a los datos recibidos se les aplica un hash SHA-3 DE 256 bits y se comparan las cadenas generadas.

Por otro lado, también existen métodos internos para la creación de usuarios y su correspondiente asignación correcta de identificadores.

3.2.2 Seguridad

3.2.2.1 JWT

Para la seguridad se ha utilizado **JWT**: JSON Web Token, un estándar de seguridad en la web, más concretamente, el RFC 7519.

Se trata de tokens que pueden contener distinta información, pero en reglas generales suelen contener siempre: la fecha de creación, la fecha de expiración y el usuario al que pertenece, en este caso, su identificador.

Son generados por el servidor, bajo la librería **jwt** de Python, y va encriptado bajo una clave única del mismo, generada aleatoriamente en cada inicio del sistema.

Solo se genera uno por usuario en cada sesión, cuya duración establecida es de 6 horas. Pare evitar accesos no deseados durante la sesión, se utilizan tokens de refresco, es decir, tokens con menos tiempo de expiración que van cambiando cada cierto tiempo para asegurar la confidencialidad de la información.

Estos tokens de refresco tienen una duración de 1 minuto y en cada petición al servidor se comprueba su validez:

- Si el token de autenticación es válido pero el de refresco no, se genera un nuevo token de refresco que sustituirá al anterior y se le entrega al usuario.
- Si ambos tokens son válidos, los tokens no se modifican.
- Si ambos son no válidos, el acceso queda denegado y es necesario realizar de nuevo el Login.

Para el manejo de los JWT existe el módulo de seguridad. En él encontramos la clave alfanumérica propia del servidor y los métodos para creación y decodificación de tokens de autenticación y refresco.

Una parte importante de la implementación de los JWT en el sistema es la existencia de una **lista negra** para ellos, almacenada en SQLite, a la que van aquellos tokens que hayan expirado o cuyos usuarios hayan decido no usar más (tras el Logout). Con esto, al recibir un token del cliente, siempre comprobamos que no esté en la lista negra previamente, para de nuevo evitar accesos indeseados.

Para estos casos especiales, existen las clases BlacklistedAuthException y BlacklistedRefreshException.

3.2.2.2 Middleware de seguridad

Por otro lado, tenemos los **middleware** para el control de accesos a los recursos. Se han implementado de forma que sean referenciables con una etiqueta del tipo ***@login_required***, haciendo que su utilización en cualquier módulo sea sencilla.

Se han desarrollado 3 middleware:

- Login required
- Admin required
- Middleware de salida

Login required, etiquetado como @login_required, comprueba que el usuario ha iniciado sesión previamente para poder acceder al recurso. Para ello, toma el token de autentificación y comprueba que existe. Si existe y es válido, continúa la ejecución al recurso deseado; en caso contrario, lanza una excepción devolviendo un estado HTTP 403 de acceso denegado.

Para comprobar que los tokens son válidos, se decodifican con la clave del servidor. Si esto falla, quiere decir que no se firmó con la misma clave y por tanto el token no es válido, y se inserta en la lista negra.

Todos los servicios contienen esta etiqueta excepto el Login y recuperar contraseña olvidada.

Admin required, etiquetado como @admin_required, toma el id del usuario y comprueba en la base de datos su rol. Si es administrador permite continuar la ejecución, y si no, lanza un código 403 de acceso denegado.

Solo llevarán esta etiqueta los servicios que sean exclusivos de administrador: crear usuario, cargar datos y borrar base de datos.

Middleware salida, etiquetado como @middleware_salida, se ejecuta tras cada petición y se encarga de obtener el token de refresco de las cabeceras

HTTP y comprobar su validez. Si es válido, continúa hacia el cliente. Si ha expirado, lo inserta en la lista negra y genera otro para el usuario.

El middleware de salida se ejecuta siempre que el token de autenticación sea válido.

3.2.3 Segunda reunión con el Cuerpo Nacional de Policía de Málaga

La segunda reunión tuvo lugar el día 17 de mayo de 2019. En ella se presentó la primera versión implementada de la interfaz, filtros y lector de datos, y se pidieron nuevos datos para probar más a fondo las funcionalidades ya implementadas.

En esta primera versión sólo se podían aplicar filtros básicos, es decir, seleccionar solo un tipo de hecho, lugar o modus, y solo una única nacionalidad para los responsables. Es decir, no permitía seleccionar varios valores del mismo campo, aunque sí consultas cruzadas.

La policía sugirió que les gustaría poder filtrar por ciertas festividades, como Navidad, Semana Santa o la feria de Málaga. También pidieron añadir los tramos horarios, que serían: Mañana, Tarde y Noche.

Por último, se comentó la idea de los agentes, de cómo plantearlos y qué considerarían ellos más relevante a la hora de predecir un delito. Gracias a esto, se pudo comenzar a construir la base para el sistema multiagente de la siguiente iteración.

4

Iteración 2

4.1 Introducción

La Iteración 2 sirvió para crear el puente de comunicación cliente-servidor, crear el primer diseño de los agentes, analizar el coste en tiempo de los servicios y crear la inicialización automática del servidor.

4.2 Desarrollo

El desarrollo de la Iteración 2 estará divido, por ende, en 5 bloques: comunicación cliente-servidor, análisis del coste de los servicios, inicialización automática del servidor, primer diseño de los agentes y tercera y última reunión con la policía.

4.2.1 Comunicación cliente-servidor

La comunicación cliente-servidor se llevó a cabo mediante el servicio **$http** de AngularJS. Este servicio se basa en los conceptos de promesas para devolver la respuesta obtenida, y utiliza el objeto XMLHttpRequest o el patrón JSONP para enviar peticiones al servidor.

En este caso, la comunicación entre el cliente y el servidor se hace íntegramente con JSON.

Las promesas en JavaScript son objetos de los cuales en un principio no conocemos su valor, pero se aceptan como válidos para no bloquear el programa y continuar su ejecución normal. Eventualmente, conoceremos su valor y podremos continuar por esa rama de ejecución en la que dependíamos de su valor.

Es una funcionalidad bastante útil y en la iteración 3 veremos más en profundidad su uso dentro de este sistema.

El cliente cuenta con varios usos de $http, uno por cada servicio alojado en el servidor. Estas peticiones se configuran añadiéndoles la siguiente información:

- URL: dirección URL a la que queremos enviar la petición (*endpoint*).
- Método: método HTTP que requiere el endpoint, por ejemplo, GET, POST o PUT.
- Cabeceras: cabeceras HTTP, como el tipo de datos que se transmite en la petición o los tokens de autentificación.

- Datos: datos enviados con sus valores.

Por otro lado, para enviar la respuesta, el servidor utiliza el método **make_response** de Flask, que toma como argumentos:

- Datos JSON de respuesta.
- Cabeceras de respuesta.
- Código de la respuesta: 200, 400 o 403 normalmente en nuestro sistema.

Por último, para los filtros, el cliente recoge los datos recibidos y aplica un color a cada zona recibida según su número de delitos registrados y en base a un total. Las zonas más conflictivas estarán representadas en color rojo fuerte y las menos conflictivas en azul cian, representando el frío y el calor simultáneamente.

A partir de aquí, se pudo determinar de una forma más correcta el coste del tiempo de respuesta de los servicios, sobre todo de los filtros.

4.2.2 Análisis del coste de los servicios

Esta parte de la iteración fue desarrollada junto al compañero Juan Palma Borda, ya que se trataba de una parte común.

Tras completar la comunicación entre el cliente y el servidor, se hizo evidente que servicios eran más costosos y cuáles no tanto, aunque la mayoría se sospechaban.

Los servicios para el Login y Logout resultaron ser de una mayor eficiencia comparados con los de filtros, pues funcionan en una base de datos pequeña comparada con la de MongoDB.

En cuanto a los filtros, los servicios que buscaban en la colección localizaciones tenían un tiempo de respuesta de 1 o 2 segundos, mientras que los que buscaban en denuncias podían escalar hasta el minuto. Por ello, en denuncias, se crearon 3 índices compuestos:

- Provincia, Municipio y Distrito municipal.
- Año, Mes y Día.
- Día de la semana y Tramo Horario.

Las búsquedas sobre denuncias se forzaban sobre esos índices, ordenándolas mediante el operador **$order**, lo que hizo aumentar la eficiencia en aproximadamente un 50%.

Pero esto, a su vez, tenía una limitación: la memoria interna de Python, ya que el método *order* de PyMongo ordenaba los resultados ya en memoria y

no previamente, por lo que, para búsquedas levemente acotadas, esto resultaba en un gran cuello de botella.

Entonces, el uso tanto del operador *order* como del *find* quedaron descartados y se pasó a usar el operador ***aggregate***, que permite realizar varias consultas simultáneamente si se combina con su operador interno ***$facet.***

Descubrimos entonces que *aggregate* resultaba en una gran eficiencia, pues hacía uso nativamente de los índices y no era necesario siquiera ordenar los resultados.

Además, ya no era necesario devolver cada documento completo encontrado en la base de datos, sino que se podía especificar el campo deseado a devolver o incluso simplemente sumar a una variable acumulativa el valor deseado por cada documento con coincidencias.

Se pasó, por tanto, a trabajar con *aggregate* en ambas colecciones. Denuncias utilizaba un acumulador que sumaba 1 por cada resultado encontrado, devolviendo el número de documentos encontrados. En cambio, las consultas a localizaciones están configuradas de forma que puedan devolver el campo deseado, siendo este normalmente o **Numero_hechos**, aunque puede devolver cualquier otro según el filtro aplicado, como, por ejemplo, el número específicos de hurtos cometidos.

Cabe destacar que la mayor parte de esta refactorización fue llevada a cabo por Juan Palma Borda.

4.2.3 Inicialización automática del servidor

El sistema cuenta con 2 bases de datos SQLite: una para los usuarios y otra para almacenar información sobre las calles y sobre palabras claves para lectura de los ficheros.

Ambas bases de datos se crean con el inicio de la aplicación, siempre que no existan previamente, de forma que, la primera vez que se ejecute el servidor en una máquina, se creen desde cero.

Esto se diseñó así pensando en una futura modificación de los scripts que contienen sus datos. Cada vez que se desee cambiar uno de estos archivos y que se apliquen los cambios, bastará con borrar la base de datos afectada y relanzar el servidor.

Junto con el inicio de la aplicación, también se generan las claves para los tokens de autenticación y refresco.

Después de todo esto, seguidamente, se ejecuta la aplicación Flask, haciendo operativo al sistema desde fuera.

4.2.4 Primer diseño de los agentes

4.2.4.1 Agentes

El primer diseño de los agentes contaba con un único tipo de modelo y un único tipo de agente. Estos eran DelitoModel y HechoAgent respectivamente.

La idea era que cada modelo instanciado tuviera un gran número de agentes, cada uno representando un delito con las siguientes cualidades:

- Tipo de hecho
- Lugar (tipo de establecimiento o vía)
- Modus operandi
- Nacionalidad del responsable
- Sexo del responsable
- Dia de la semana
- Tramo horario
- Municipio o distrito

Cada DelitoModel contaba con 3 variables principales:

- Número de agentes que va a generar
- Una lista para almacenar dichos agentes
- Un diccionario de diccionarios para almacenar porcentajes de asignación.

Además, consta de 2 métodos: asignar_característica y _calcular_porcentajes, para la asignación de un tipo de característica a cada agente (como puede ser un tipo de hecho) y el cálculo de porcentajes de estas asignaciones, respectivamente.

Asignar característica consta de dos parámetros: una lista con valores y el nombre de la característica a la que representan. Esta lista de valores contiene tripletas del tipo (característica, cantidad, porcentaje empírico de ocurrencia), representando el nombre específico de la característica (por

ejemplo: HURTO), la cantidad de ocurrencia registrada en el año que se realizó esta consulta y el porcentaje que representó esta característica dentro de todos los delitos registrados ese año. Un ejemplo podría ser:

(HURTO, 2356, 47.63)

La asignación se lleva a cabo lanzando un número aleatorio por cada agente y elemento de la lista. Si este número supera al porcentaje de la tripleta iterada, esa característica se le asigna al agente que lanzó el número. En caso contrario, se pasa a la siguiente tripleta. Así sucesivamente hasta haber asignado una característica de este tipo a cada agente.

A su vez, al terminar de asignar cada grupo de características, se realiza un cálculo sobre el porcentaje de agentes que posee cada tipo específico de dicha característica. De ello se encarga **__calcular__porcentajes.**

Esto nos sirve para poder visualizar posteriormente qué características son más comunes en un delito que otras, por ejemplo, qué hecho o modus operandi son más comunes que otros de su categoría.

4.2.4.2 Infraestructura para los agentes

Para que la información llegue de esta forma a los agentes, fue necesario crear una infraestructura superior que lo permitiera.

El primer diseño de esta infraestructura consistía en un método principal, **agentes__simulación**, que tomaba un municipio o un distrito y una lista de

parámetros ordenados según orden de relevancia para la búsqueda, pues se buscaba crear un **árbol de decisiones**.

El parámetro con las selecciones tiene la siguiente estructura:

[{"hecho" : ["HURTO"]}, {"lugar" : []}, {"modus" : []}, {"nacionalidad" : []}, {"sexo" : []}, {"dia" : []}, {"tramo" : []}]

Se trata de una lista de diccionarios, con strings como claves y listas como valores. El orden en que los diccionarios se encuentran ordenados en la lista marca su prioridad. Tendrá más prioridad, en este ejemplo, tipo de hecho, y menor prioridad tramo horario.

El árbol de decisión se implementó encadenando llamadas entre distintos métodos que contenían las condiciones necesarias para avanzar.

Cabe destacar la implementación de la opción **autocompletar**, que, si es seleccionada, busca los tipos más relevantes de aquellas características que tengan sus listas vacías, en orden de preferencia. En el ejemplo anterior, se buscarían los lugares más relevantes (dentro del municipio o distrito seleccionado) en los que se cometen hurtos; posteriormente, los modus más utilizados en esos lugares para los hurtos; y así sucesivamente, hasta obtener los más relevantes para cada uno vacío.

Estas consultas son realizadas para cada año que contenga datos completos en la base de datos. En nuestro sistema, cada modelo representará un año.

Tras realizar todas estas búsquedas para dicho año en el distrito o municipio seleccionado, se le asignan las características al modelo como hemos visto previamente.

Por último, cabe destacar también la existencia de varios métodos intermedios encargados de calcular los elementos más relevantes en cada característica y asignarle un porcentaje respecto al total de dicha característica. Si la característica contiene métodos en su lista, estos se ordenan de mayor a menor relevancia. En cambio, si la búsqueda es automática para una característica, se realiza una criba: solo se mantienen los tipos que compongan el 90% de la búsqueda realizada, y posteriormente se ordenan según su relevancia.

4.2.5 Última reunión con el Cuerpo Nacional de Policía de Málaga

La tercera y última reunión con la policía se realizó el día 8 de julio de 2019.

En ella le presentamos la penúltima versión del sistema que tuvo aceptación por su parte.

Se comentó más a fondo la idea de los agentes y cómo se estaba implementando, y ellos sugirieron la idea de, además, poder ver la evolución de los delitos a lo largo de un año o en fechas específicas dentro de los meses. Ambas cosas requerían de un mayor número de datos para poder llevar a cabo un buen acercamiento a ambos problemas.

También les gustó la idea de autocompletar en los agentes, obteniendo los campos más relevantes que no fuesen rellenos, caso en el cual ellos establecerían la preferencia.

Por último, sugirieron un par de cambios en la exportación a CSV generada por la aplicación tras la aplicación de un filtro, y se sugirió la idea de dar permiso al administrador de borrar la base de datos en caso de haber introducido datos erróneos.

5

Iteración 3

5.1 Introducción

La tercera y última iteración sirvió para profundizar en el diseño de los agentes, cerrar la implementación de la seguridad con el cliente, creación del rol administrador y sus privilegios, y la creación de una nueva función para la evolución de un hecho a lo largo del tiempo.

5.2 Desarrollo

El desarrollo de la tercera iteración está divido en 4 secciones: implementación definitiva de JWT, diseño final de los usuarios, modificaciones de los agentes y evolución temporal.

5.2.1 Implementación definitiva de JWT

En la iteración 1 se llevó a cabo la implementación de los JWT en el servidor, pero el cliente aun no tenía constancia ellos. Esta parte fue desarrollada junto a Alberto Ramírez Mena, el encargado del front-end.

Ambos JWT, el de autenticación y el de refresco, se almacenan en las *cookies* del navegador de forma temporal, únicamente mientras el usuario mantenga su sesión activa. Para ello se usó la directiva **$cookies** de AngularJS. Se almacenan una vez el usuario ha hecho un Login correcto, y se eliminan una vez el usuario cierra conscientemente la sesión. Esto fue diseñado así para modelar el caso en el que el usuario cierre la ventana por accidente y así no pierda todo su progreso.

Existen 3 métodos en el front-end destinados al manejo de los tokens: isLoggedIn, isAdmin y parseJWT.

parseJWT se encarga de decodificar el *payload* (contenido útil) del token JWT de autenticación. Esto permite obtener su fecha de expiración, fecha de expedición e id interna del usuario al que pertenece.

isLoggedIn se encarga de comprobar que existe un token JWT de autenticación en las cookies para el usuario actual, y que éste es válido, es decir, no ha expirado aún. Para ello se compara la fecha y hora actual con la del token. En caso de que sea válido, devolverá True, si no, False.

La finalidad de este método es poder prevenir el acceso no deseado mediante URL a zonas de la aplicación, configurando la directiva **$routeProvider**, encargada de obtener las plantillas del servidor, de forma que solo las obtuviera si existía un usuario en la sesión. En caso contrario, toda acción es redirigida a la página de Login.

Por último, **isAdmin** se encarga de hacer una petición al servidor con la id del usuario obtenida del token con el fin de comprobar si es administrador o

no. Es utilizado para comprobar que solo los administradores pueden acceder a sus funciones y plantillas alojadas en el servidor.

5.2.2 Diseño final de los usuarios

De nuevo, en la iteración 1, se llevó a cabo un diseño inicial de los usuarios en el que solo constaban de 4 atributos: id, nombre de usuario, contraseña y rol.

En esta última iteración, se valoró la idea de añadir una función para poder recuperar contraseñas, en caso de que el usuario la olvidase. Para ello, se le añadió un nuevo atributo a cada usuario que sería el correo electrónico. También se creó, por tanto, un servicio para poder recuperar contraseñas, que cambia la contraseña del usuario por una temporal y se la envía a su correo electrónico. El usuario deberá cambiarla posteriormente.

Así mismo, se añadió una funcionalidad para poder cambiar la contraseña a deseo del usuario desde la sección **Ayuda** de la interfaz.

Se añadieron también, junto al compañero Alberto Ramírez Mena, las funciones de administrador a la interfaz y el servidor, las cuales permanecen ocultas siempre que el usuario de la sesión no sea administrador. Esta nueva funcionalidad engloba otras tres: crear usuario, borrar base de datos y añadir datos nuevos.

Crear un nuevo usuario requiere su nombre (debe ser único), su contraseña y su correo electrónico, y si se desea, la concesión del rol administrador.

Por otro lado, las funciones de añadir nuevo usuario y borrar la base de datos, fueron desarrolladas por Juan Palma Borda y Alberto Ramírez Mena, pues correspondían a la lectura y tratamiento de datos.

5.2.3 Modificaciones del módulo de agentes

5.2.3.1 Modelo final de los agentes

La versión final de los agentes pasó de tener un solo tipo de modelo a contener una jerarquía. Los nuevos modelos son: HistoricoModel y SimulacionModel, y ambos heredan de DelitoModel.

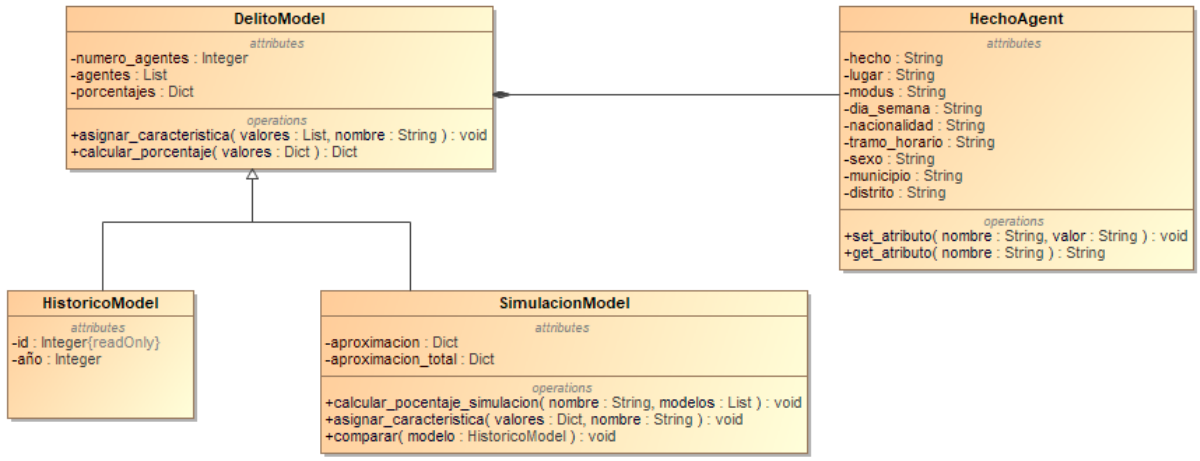


Imagen 5.2.3.1.1: Diagrama de herencia entre los agentes

DelitoModel se mantiene intacta respecto a su diseño original, aunque su antiguo funcionamiento fue delegado en HistoricoModel, que cuenta con dos atributos nuevos: **id**, para diferenciarlo, y **año**, para poder reconocer a que año pertenecen los datos del modelo. Por el resto, reutiliza los métodos de su superclase sin ningún tipo de reescritura.

Por otro lado, SimulacionModel, será el modelo encargado de representar los datos simulados a partir de un conjunto de HistoricoModels. Consta de dos atributos nuevos: aproximación y aproximación total, encargados de representar los resultados de la simulación y su aproximación respecto al último año empírico, respectivamente. Para ello, se obtienen los datos generados en

HistoricoModels del antepenúltimo y penúltimo año al actual, se asignan a la instancia de HistoricoModel y se comparan con los datos del último año.

Los datos del antepenúltimo año no tendrán tampoco la misma relevancia que los del penúltimo, por lo que, actualmente, se les asignan a los datos más recientes del mismo tipo el 70% de su valor, y a los más antiguos el 30%.

Posteriormente, estos datos calculados se le asignan al modelo dentro de su variable **aproximación**, constituyendo así, la simulación de los tipos de delitos seleccionados.

Por último, cada porcentaje asignado a cada característica dentro de la simulación es comparado con los datos del HistoricoModel del último año registrado, obteniendo así una **aproximación orientativa** sobre la calidad de la simulación.

5.2.3.2 Infraestructura final para la creación de modelos

A petición de la policía, se añadieron los siguientes parámetros y funcionalidades para la simulación:

- Simular en varios municipios
- Simular en varios distritos municipales
- Buscar dentro de un rango específico de meses
- Buscar dentro de un rango específico de días
- Buscar dentro de un rango de horas
- Buscar en festividades

Las búsquedas en festividades anulan la posibilidad de buscar simultáneamente dentro de unos rangos de días o meses, por razones obvias.

Le ejecución fue modificada también, de forma que tras rellenar todos los modelos HistoricoModel con sus agentes, se crea un SimulacionModel con todos estos modelos y se simula.

También se llevó a cabo una refactorización de toda esta infraestructura, dando lugar a un código más limpio y eficiente.

5.2.4 Evolución temporal

Esta nueva función consiste en poder ver, a través de los meses del año o los meses preseleccionados, la evolución de un filtro específico, tal como podría ser "Los hurtos en farmacias producidos en Málaga durante la noche".

Su funcionamiento está basado en los mismos filtros que se aplican de forma normal al mapa. La búsqueda en la base de datos se realiza desde el primer año registrado en ella hasta el más reciente, obteniendo así una vista más completa de la evolución criminal. Además, permite buscar no sólo en los meses completos, sino en ciertos días o incluso solo en ciertos meses.

La interfaz fue desarrollada de nuevo junto a Alberto Ramírez Mena, y permite una fácil navegación mediante el clic de flechas hacia izquierda y derecha.

6

Conclusiones y Líneas Futuras

6.1 Introducción

Este apartado está destinado al desarrollo de las conclusiones obtenidas del proyecto, las dificultades encontradas en el camino y sus posibles mejoras.

6.2 Conclusiones

En este TFG realizado en grupo se ha desarrollado con éxito una plataforma web para el procesamiento, visualización y simulación de datos sobre delitos en la provincia de Málaga. Este entorno utiliza como información de entrada datos de delitos proporcionados por el CNP de Málaga. Los miembros del CNP que han colaborado durante el desarrollo de este TFG han valorado muy positivamente tanto la plataforma como las funcionalidades que esta ofrece.

Bajo mi punto de vista, las iteraciones 0 y 1 fueron las más sencillas de realizar, pero también de las más extensas, ya que componían la mayor parte del trabajo de investigación y la creación de una base lo suficientemente sólida como para construir una aplicación dividida en 3 de forma paralela.

Durante estas primeras fases, el servidor sufrió diversas reformas hasta llegar a ser suficiente como para continuar, ya que continuamente se encontraban mejoras que llevar a cabo y, en ocasiones, caminos sin salida.

Por otro lado, el desarrollo de las iteraciones 2 y 3 fue más intenso, ya que se trataba del desarrollo de los agentes y su diseño, junto con la seguridad y las consecuentes optimizaciones del cliente y servidor para su correcto funcionamiento.

En general, trabajar conjuntamente con el CNP de la ciudad Málaga, ha sido una gran experiencia puesto que hemos podido conocer de primera mano cómo es trabajar con un cliente real, entendiendo sus problemas del día a día y poniendo solución a ellos.

6.3 Dificultades encontradas

Entre las diferentes dificultades encontradas, caben destacar dos principales: las consultas cruzadas y su eficiencia, y la predicción basada en agentes.

Por un lado, realizar consultas de filtros compuestos fue un requisito desde el primer día de desarrollo, pero éste no era el problema. El problema era hacer que estas consultas fueran eficientes ya que se trataban de conjuntos de datos de gran tamaño. Para ellos, Juan Palma Borda y yo creamos una segunda

colección de datos, más ligera, y optimizamos los servicios REST de Flask, disminuyendo el tiempo de carga en aproximadamente un 80%.

Por otro lado, la predicción basada en agentes también debía ser eficiente, ya que el proceso de predicción constaba de varias fases: recoger los datos históricos relevantes seleccionados por el usuario, crear un modelo para los últimos años registrados y crear un último modelo para simular posibles comportamientos futuros.

Para ello, se llevaron a cabo varias refactorizaciones de código y se cambiaron algunos atributos por defecto de los agentes de la librería MESA, debido al que el uso de ellos resultaba en una salida bastante ineficiente.

6.4 Trabajos futuros

Algunas posibles mejoras o nuevas funcionalidades se consideran a continuación.

6.4.1 Implementación de HTTPS

En un futuro resultará imprescindible que la comunicación entre el cliente y el servidor tenga lugar mediante HTTPS, ya que se trata de datos confidenciales a los que nadie debería tener acceso salvo autorizados.

Esta acción no se puedo llevar a cabo por falta de tiempo.

6.4.2 Despliegue de la aplicación en la nube

La idea principal del proyecto era desplegarlo en algún *host* online, como podía ser Heroku. De esta forma, la plataforma sería accesible a través de la

web para ellos y no se quedaría en nuestras máquinas locales. Esto tampoco fue posible realizarlo por falta de tiempo.

6.4.3 Obtención de perfiles de personas en la simulación por agentes

Una función interesante para un futuro podría ser la obtención de perfiles de personas con mayor potencial a cometer un tipo de delito. Ello se podría conseguir modificando el módulo de agentes y buscando la relevancia de cada característica de los responsables frente a un determinado delito.

Esto fue una idea que se tenía en un principio, pero se consideró más importante el crear la estructura de los agentes y poder ver en primer lugar los delitos más propensos a ocurrir en un futuro y dónde serían.

Bibliografía

MongoDB. "The most popular database for modern apps | MongoDB"

https://www.mongodb.com/

PyMongo. "PyMongo 3.9.0 Documentation"

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

MESA. "MESA: Agent-based modeling in Python 3+"

https://mesa.readthedocs.io/en/master/

SQLite. "SQLite Documentation"

https://www.sqlite.org/docs.html

sqlite3. "DB-API 2.0 interface for SQLite databases"

https://docs.python.org/3/library/sqlite3.html

Flask. "Flask Documentation"

https://flask.palletsprojects.com/en/1.1.x/

JWT. "JSON Web Tokens"

https://jwt.io/

PyJWT. "Welcome to PyJWT"

https://pyjwt.readthedocs.io/en/latest/

Ayuntamiento de Málaga. "Datos abiertos Ayto. Málaga"

https://datosabiertos.malaga.eu/

X-Law. "La aplicación que permite prever dónde se van a producir robos y atracos"

https://www.diariosur.es/tecnologia/aplicacion-permite-prever-20181123105411-nt.html

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

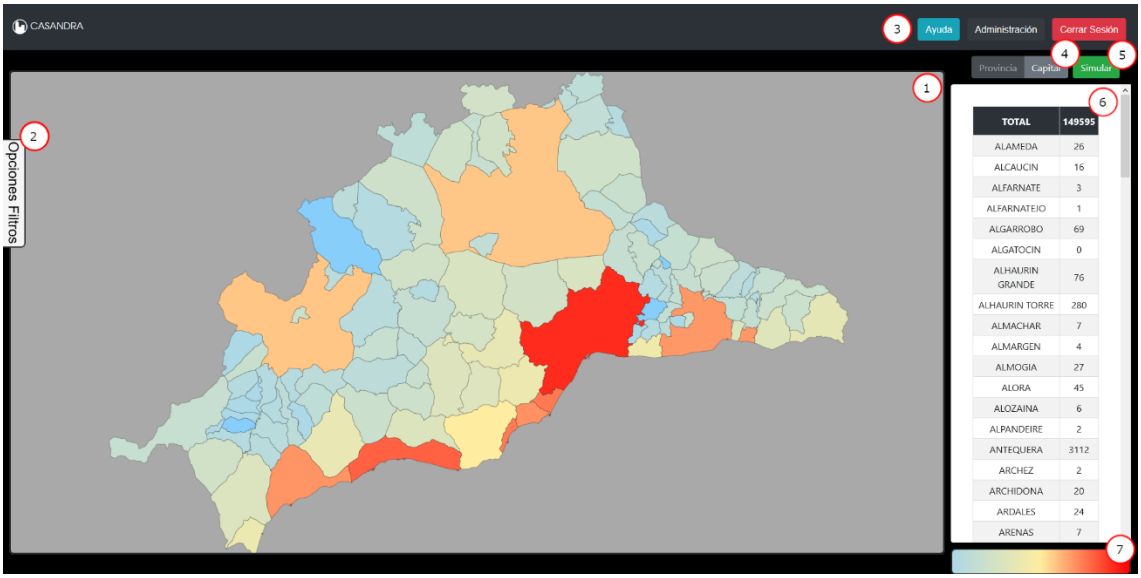


Figura 2 *Interfaz Principal*

1.- Mapa de calor 2.- Pestaña de filtro 3.- 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]).

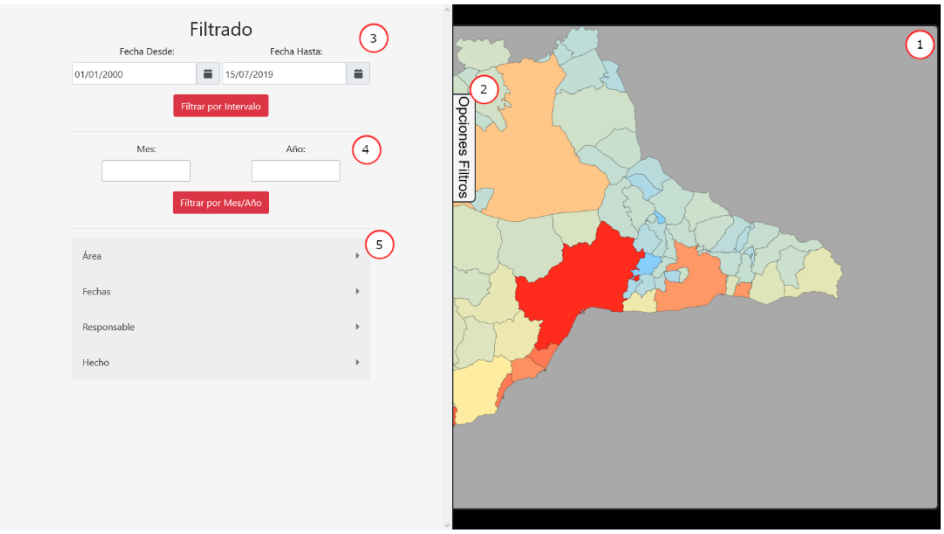


Figura 2.1 *Pestaña de Filtros*

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.

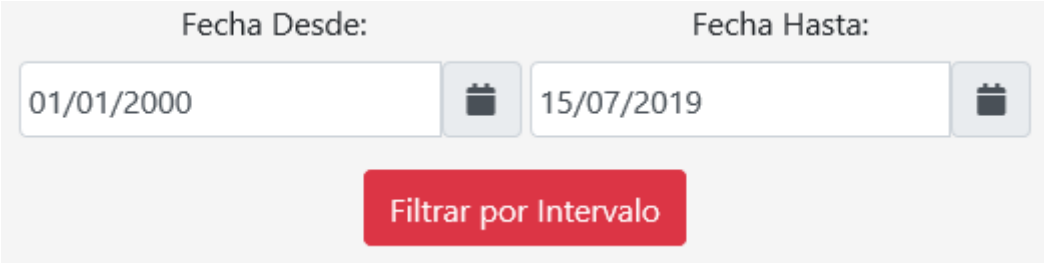


Figura 2.1.1 *Filtrar por Intervalo*

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]).

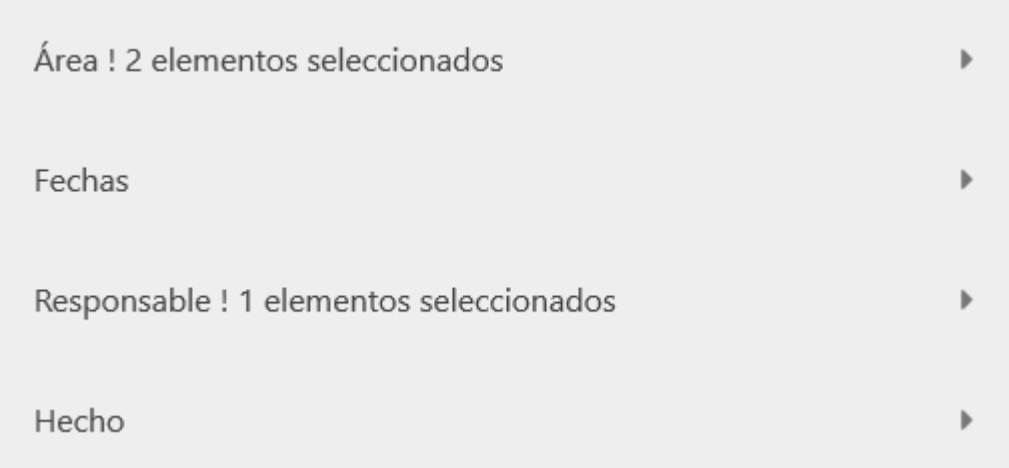


Figura 2.1.3 *Opciones de filtrado*

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]).

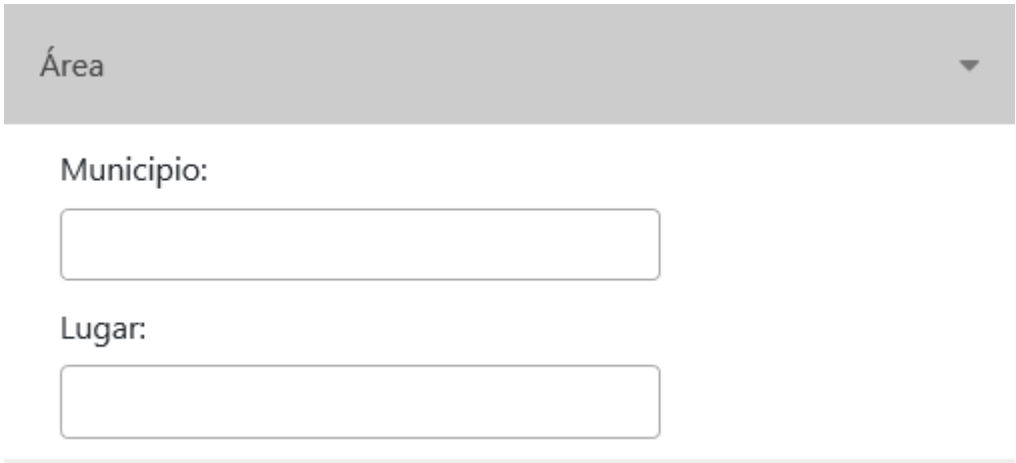


Figura 2.1.3.1.1 *Opciones de área (Provincia)*

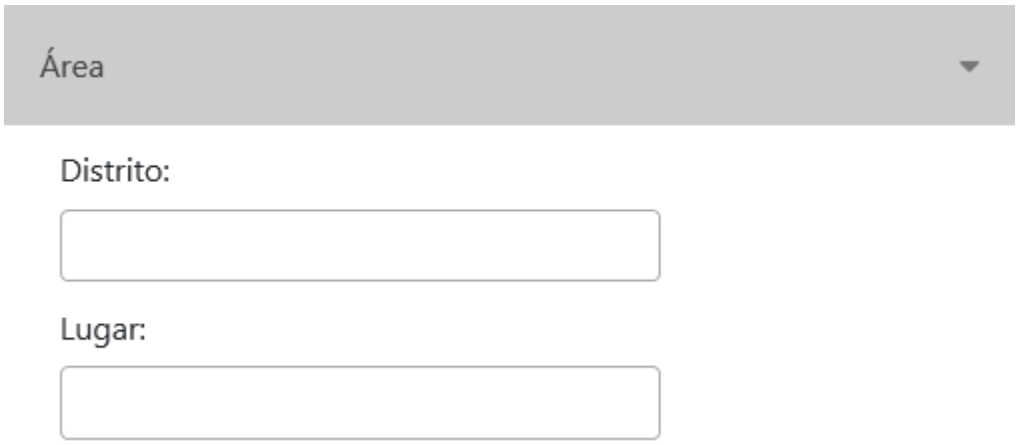


Figura 2.1.3.1.2 *Opciones de área (Capital)*

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]).

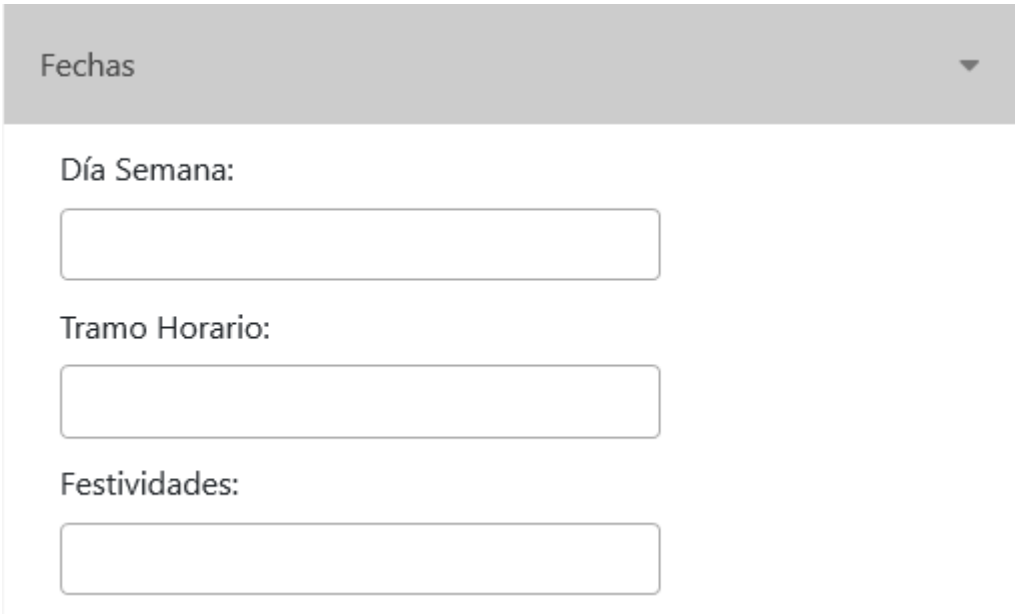


Figura 2.1.3.2 *Opciones de Fecha*

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]).

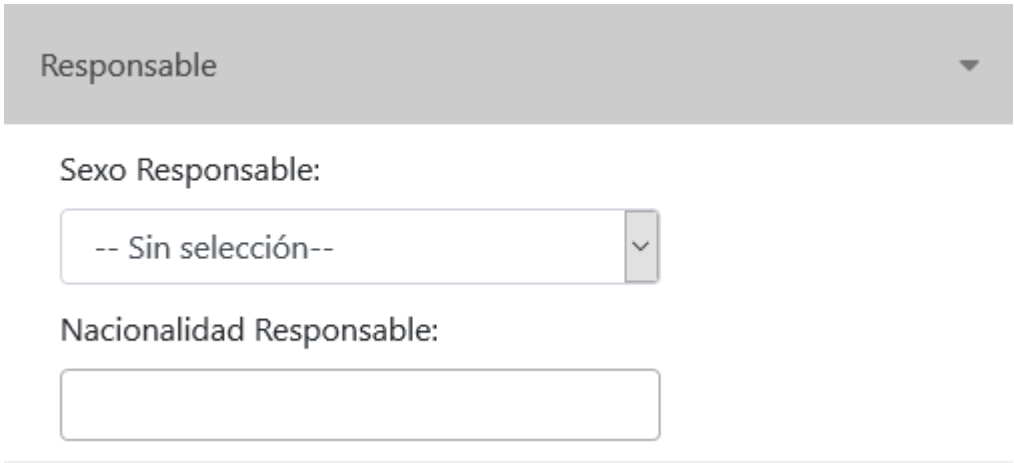


Figura 2.1.3.3 *Opciones de Responsable*

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*

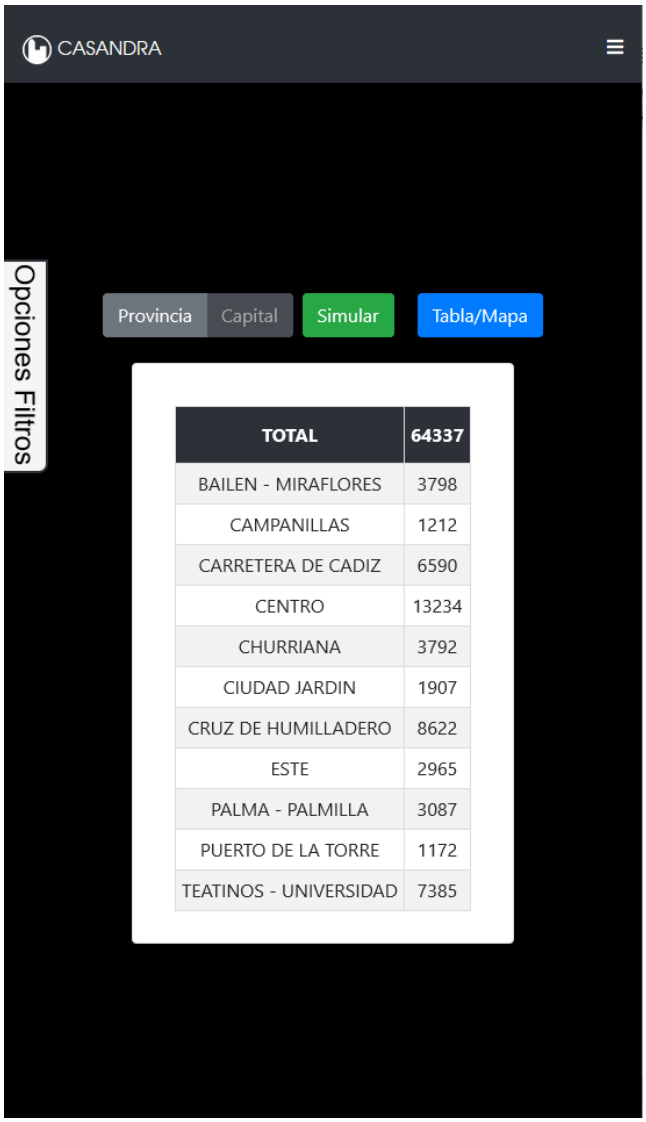
A.2.2. Consultar datos y estadísticas

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].



TOTAL	64337
BAILEN - MIRAFLORES	3798
CAMPANILLAS	1212
CARRETERA DE CADIZ	6590
CENTRO	13234
CHURRIANA	3792
CIUDAD JARDIN	1907
CRUZ DE HUMILLADERO	8622
ESTE	2965
PALMA - PALMILLA	3087
PUERTO DE LA TORRE	1172
TEATINOS - UNIVERSIDAD	7385

Figura 2.2.1 *Tabla (Pantallas pequeñas)*

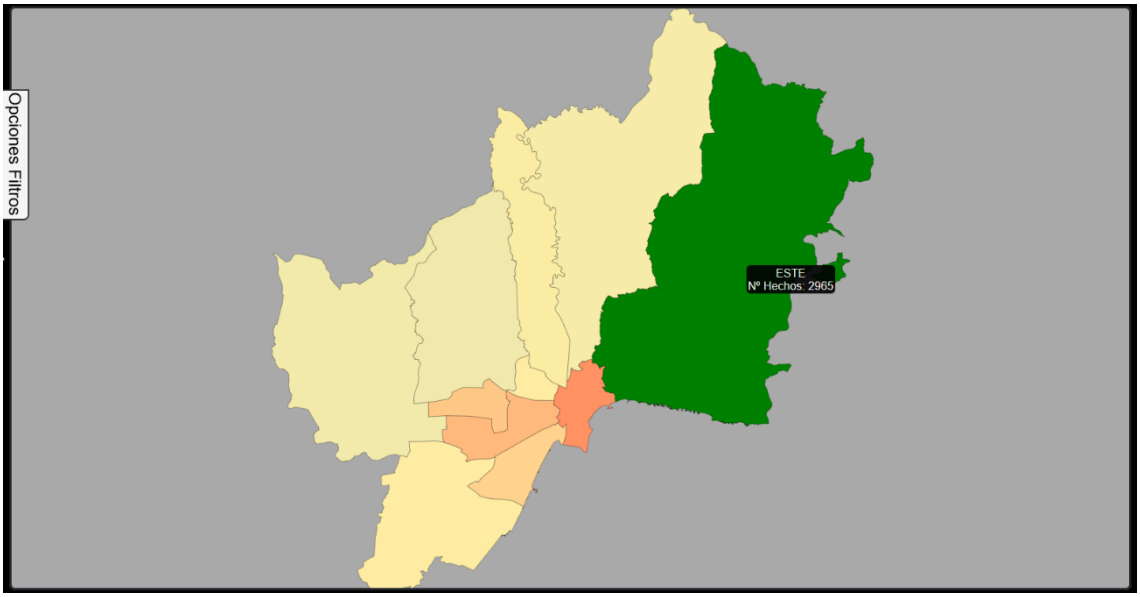


Figura 2.2.2 *Datos de un área determinada*

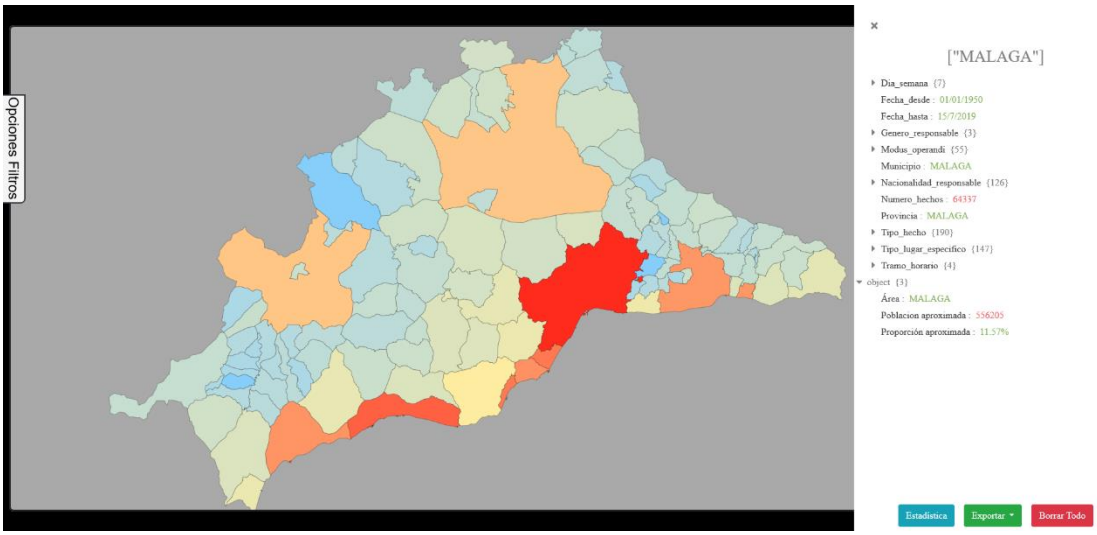


Figura 2.2.3 *Representación en árbol*

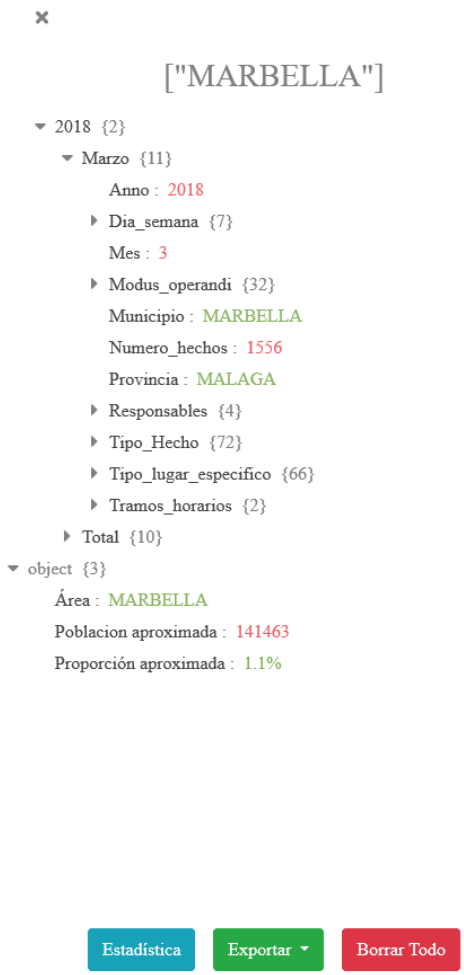


Figura 2.2.4 *Representación alternativa*

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].

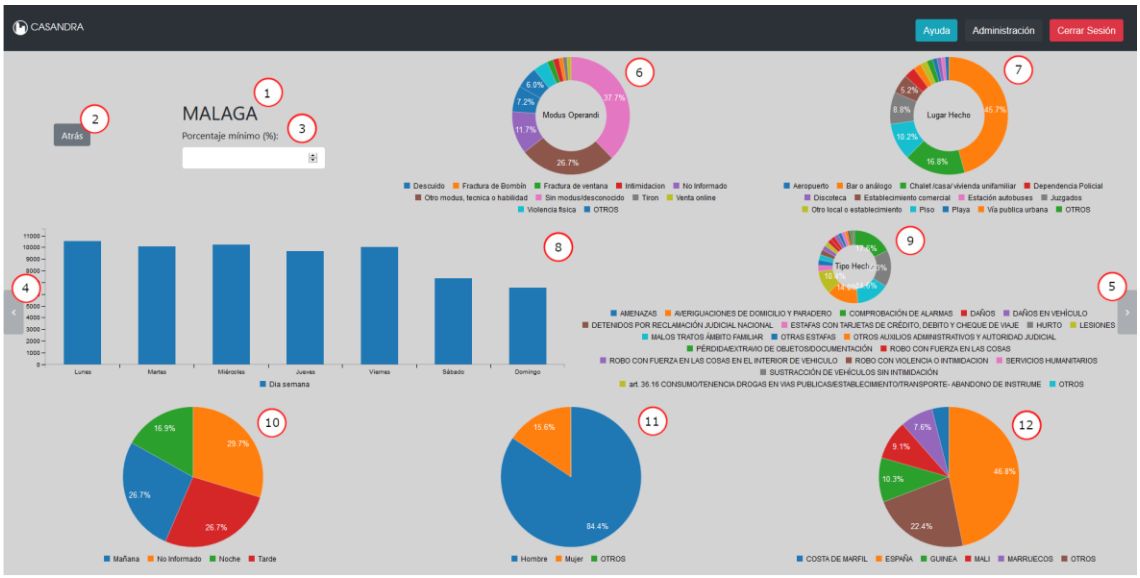


Figura 2.2.1.1 *Estadísticas de una zona*

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].

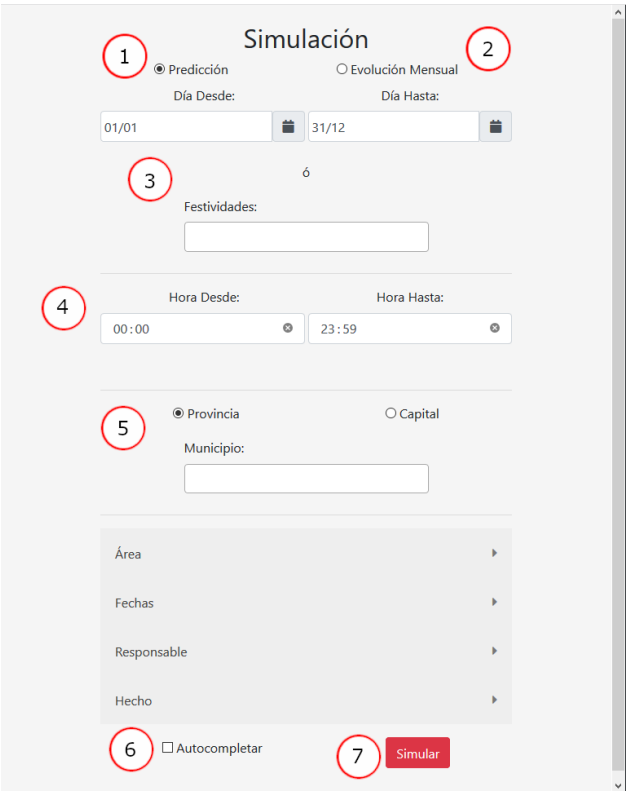


Figura 2.3.1.1 *Menú de Predicción*

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

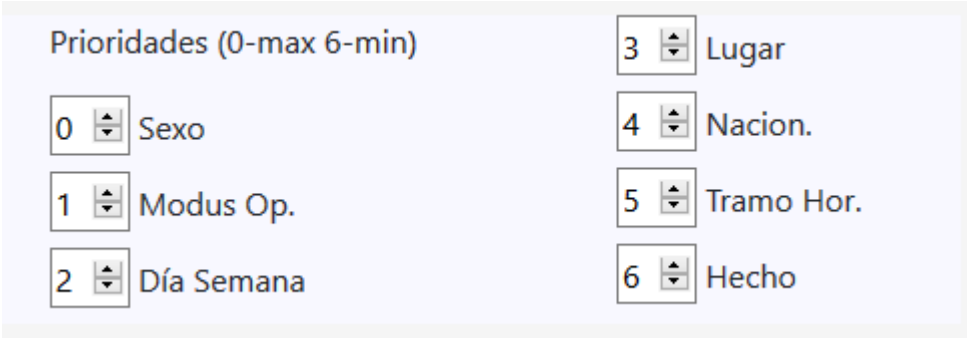


Figura 2.3.1.2 *Prioridades autocompletar*

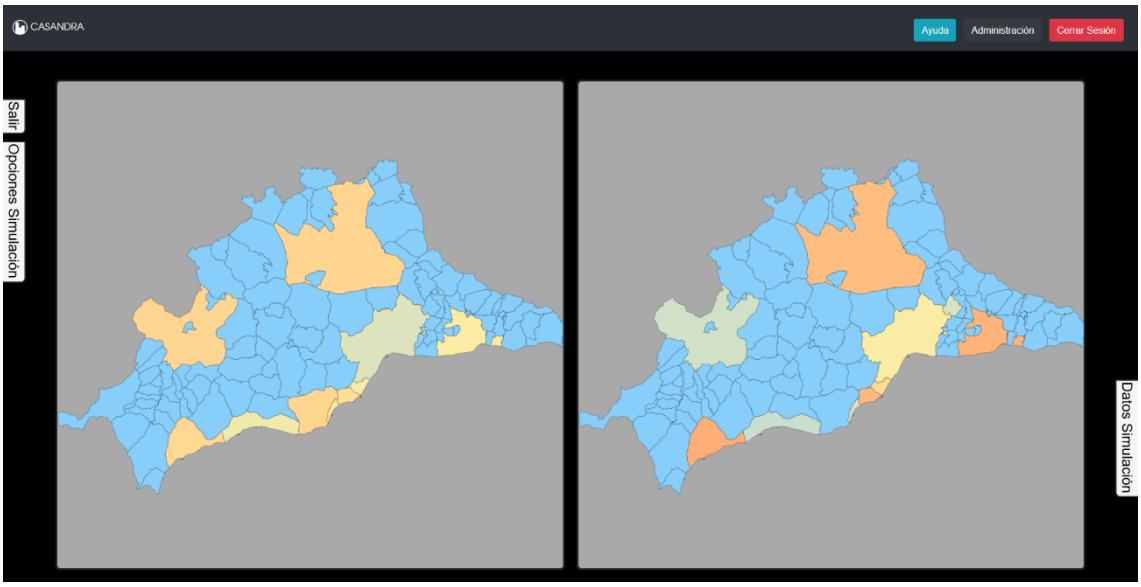


Figura 2.3.1.3 *Datos Empíricos y Simulados (Mapas)*

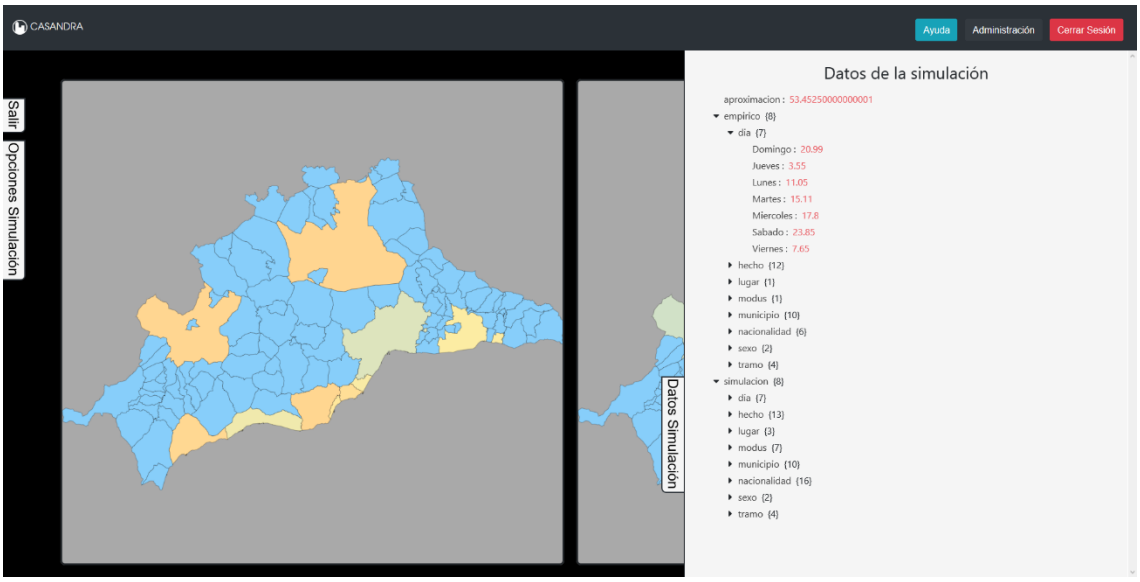


Figura 2.3.1.4 *Datos Empíricos y Simulados (Árbol)*

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".

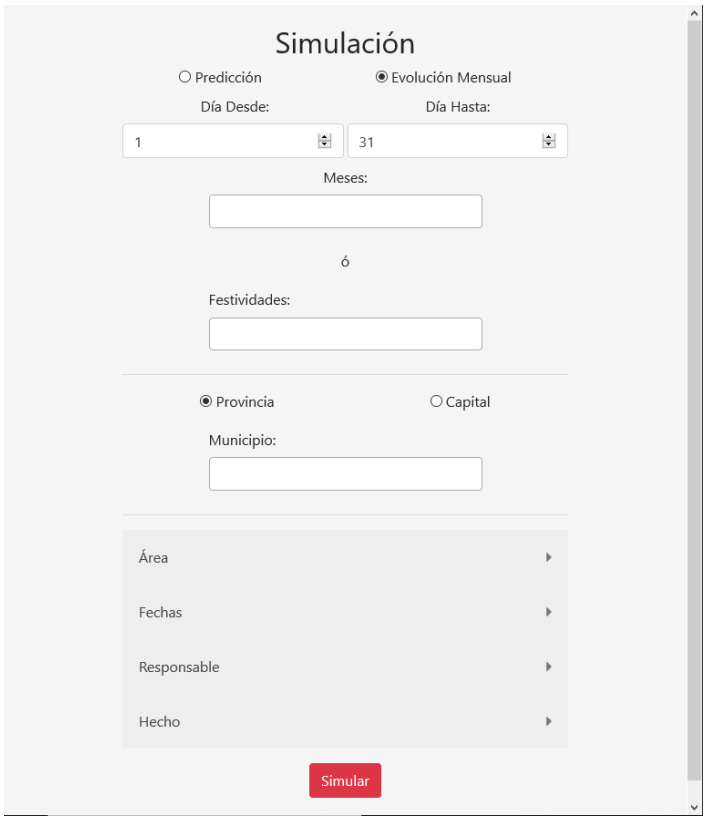


Figura 2.3.2.1 *Menú de Evolución Mensual*

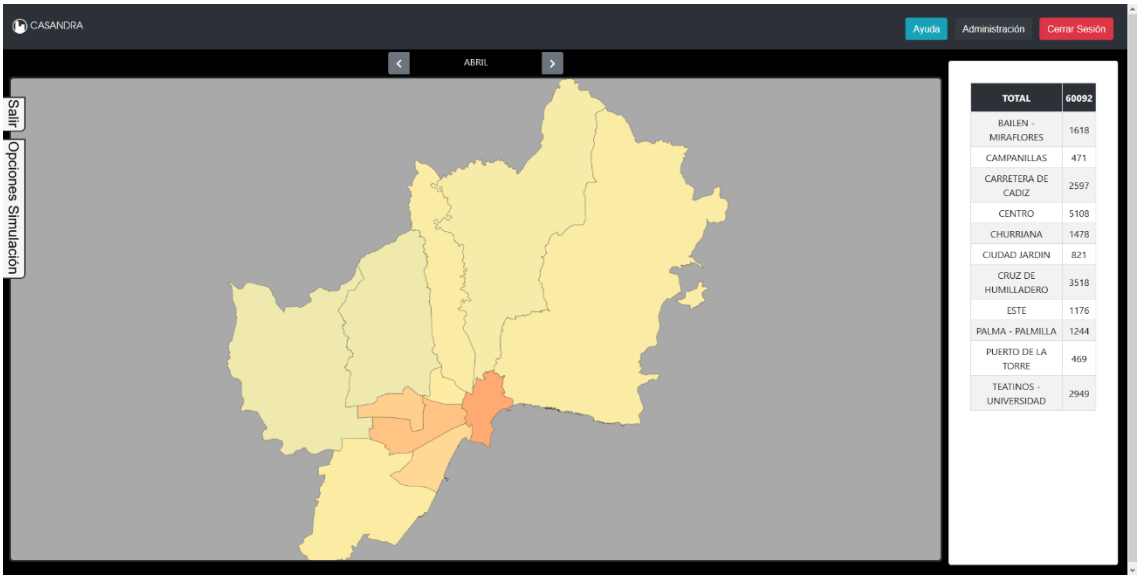


Figura 2.3.2.2 *Mapa de Evolución Mensual*

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.

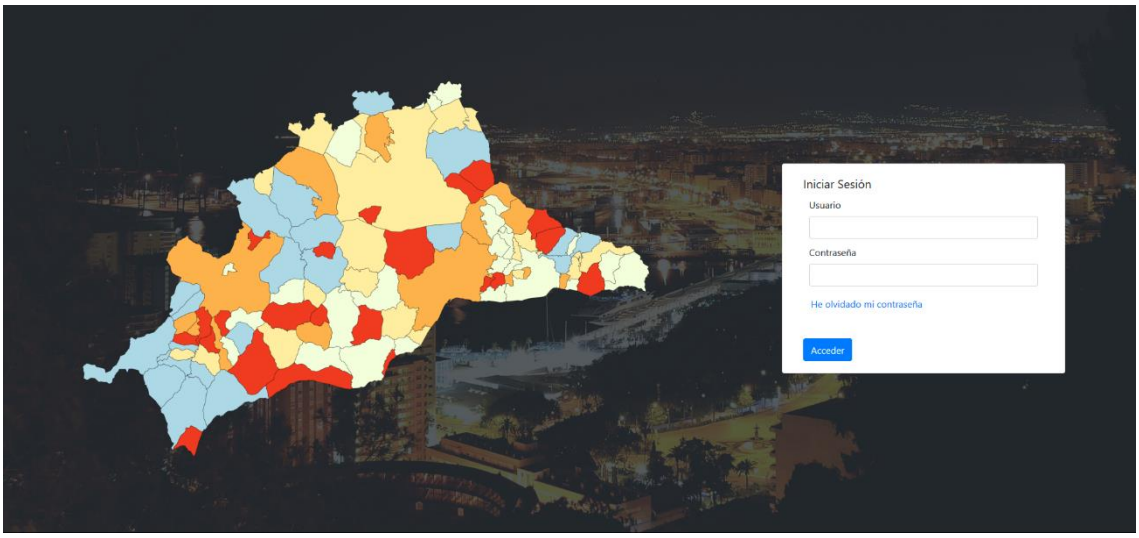


Figura 2.4.1 *Acceso a usuarios*

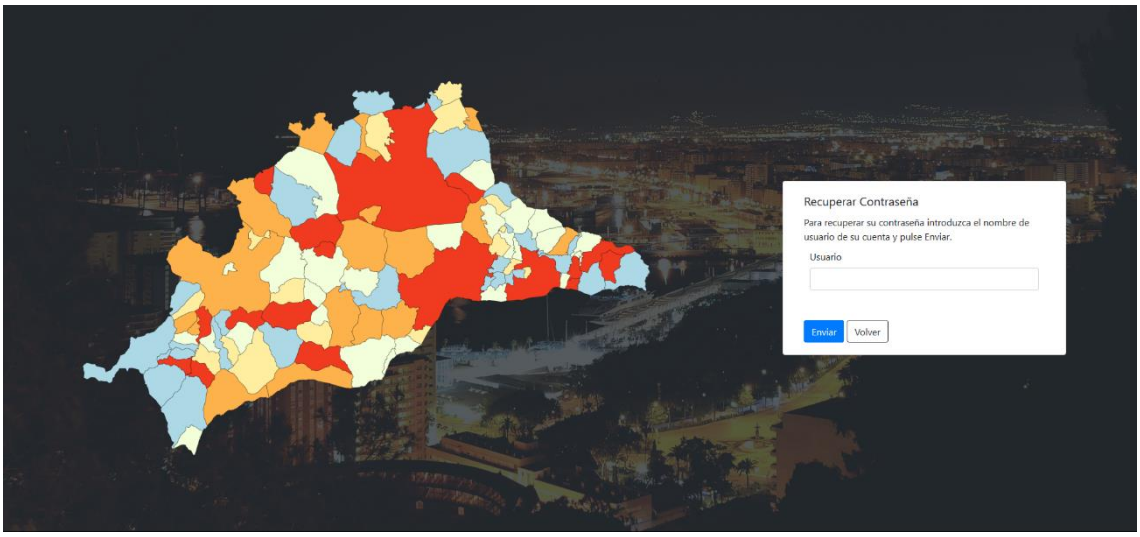


Figura 2.4.1.1 *Recuperar contraseñas*

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].

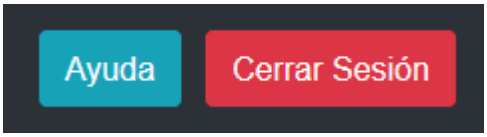


Figura 2.4.2.1 *Botón de Ayuda*

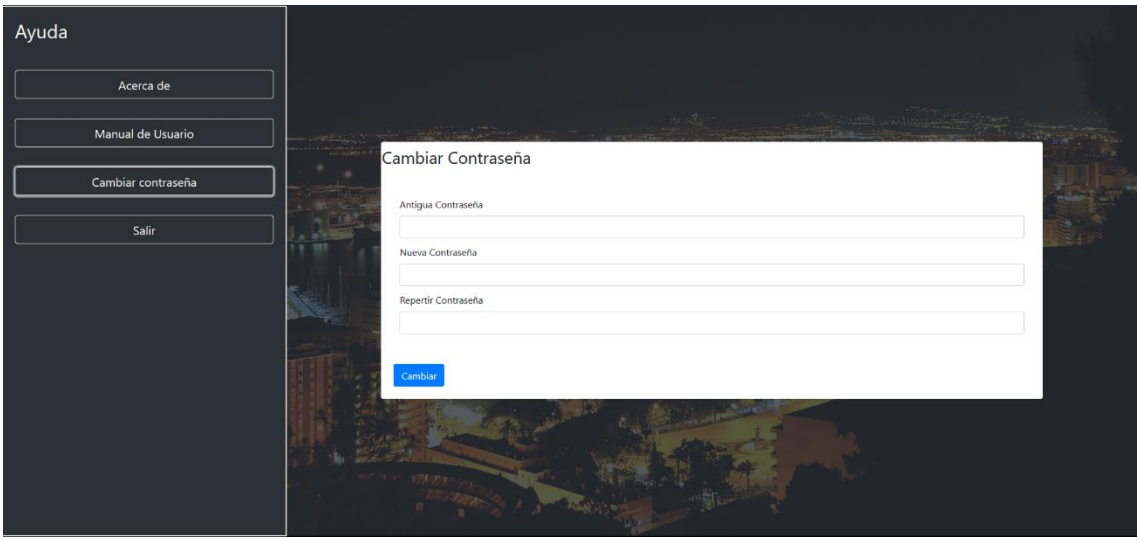


Figura 2.4.2.2 *Ventana de ayuda*

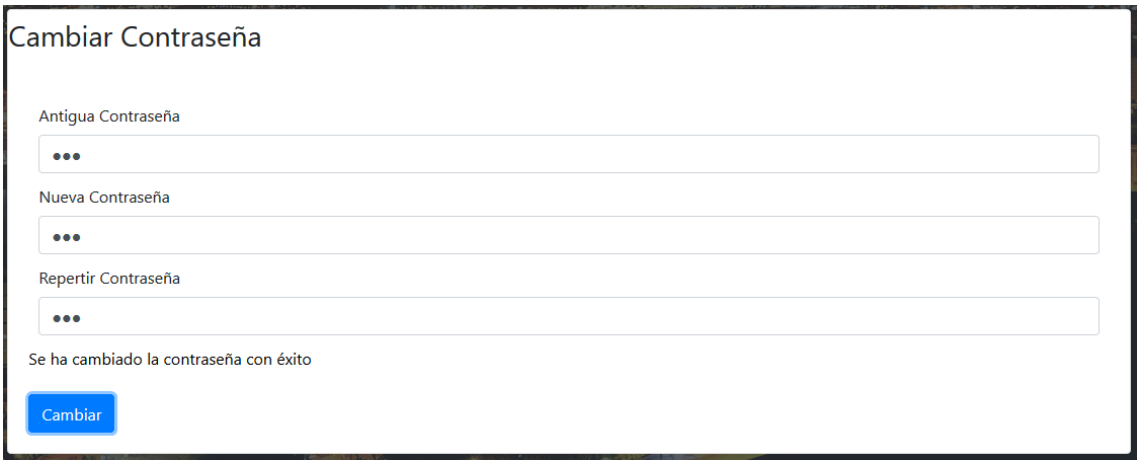


Figura 2.4.2.3 *Confirmación del cambio de contraseña*

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.

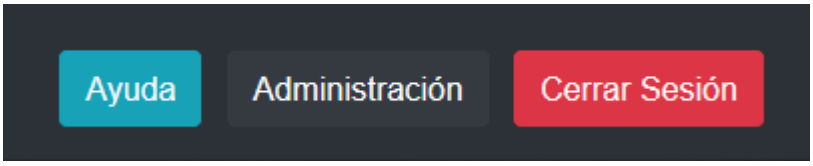


Figura 2.5 *Botón de Administración*

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.*

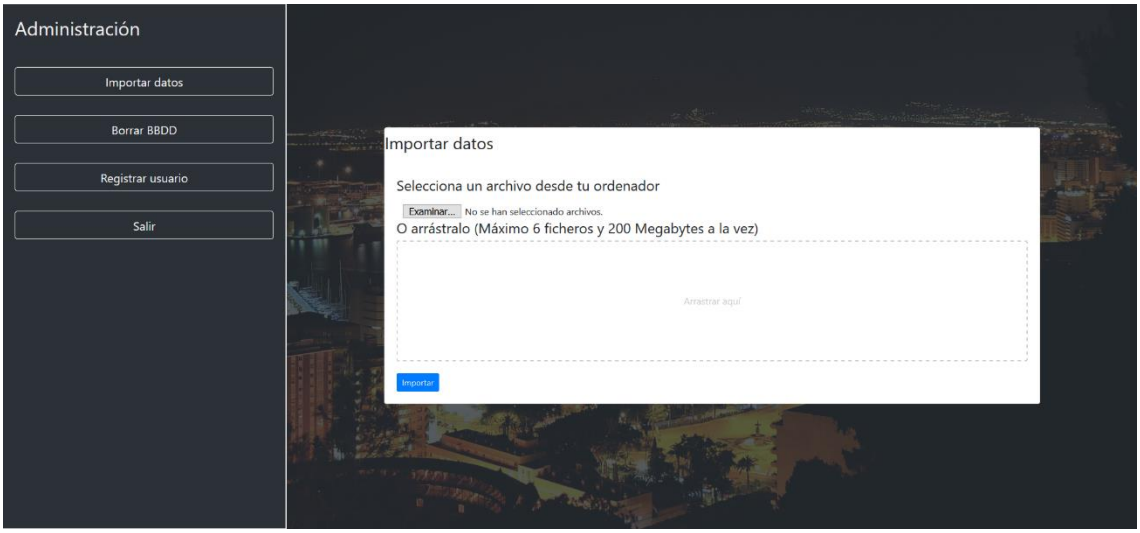


Figura 2.5.1 *Importar datos*

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.

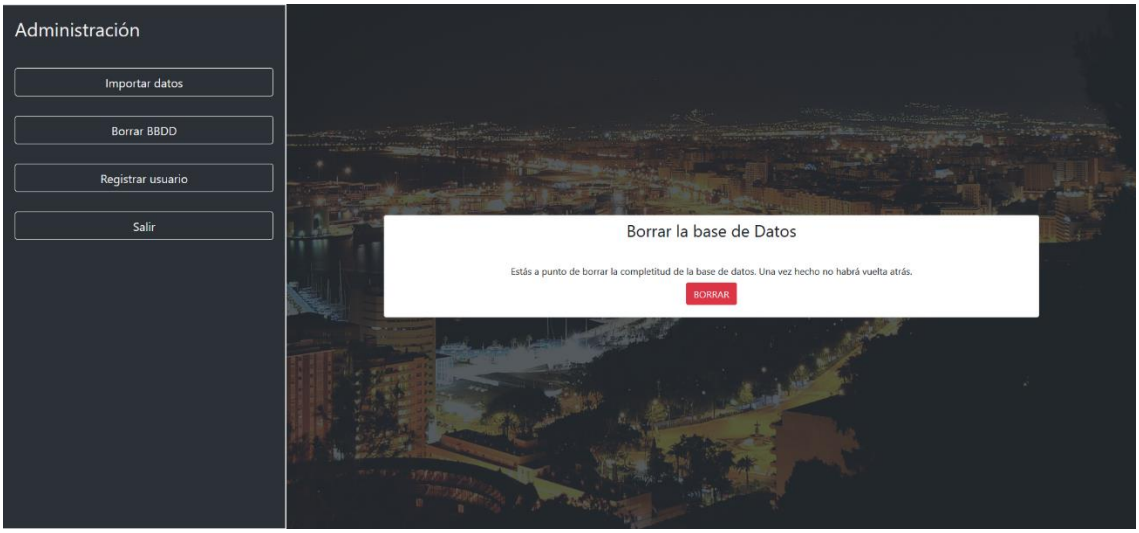


Figura 2.5.2 *Borrar Base de Datos*

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.

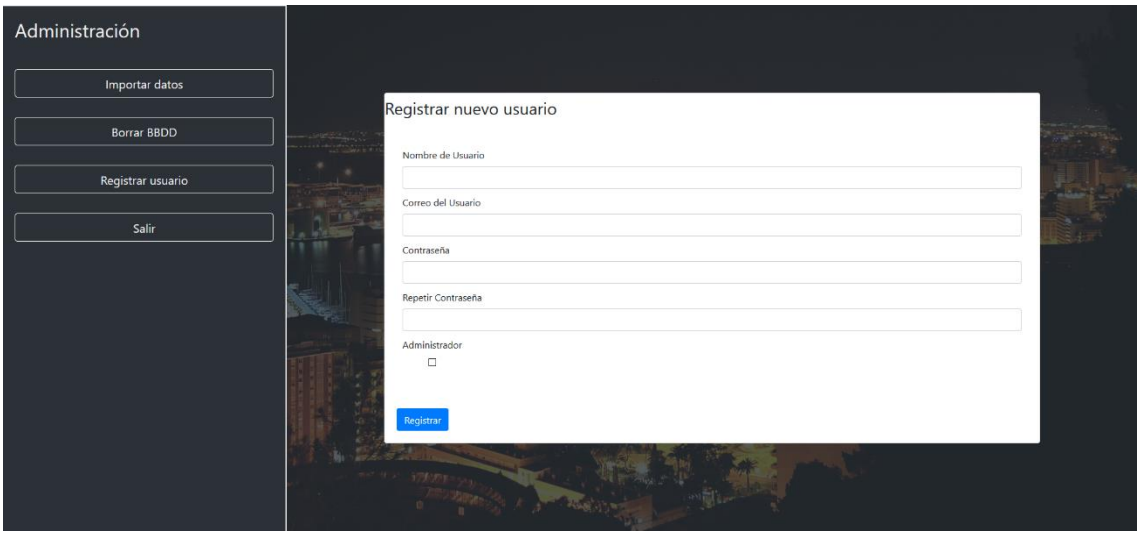


Figura 2.5.3 *Registrar Usuario*

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.

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*****:** servicio 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*****:** servicio 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**.

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.

Este módulo encapsula la importación a bases de datos relacionales en forma de árbol, con unas interfaces realizadas de objetos.

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:*** *g*enera 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 login 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.

Métodos

- ***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_caracteristica***: 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. Introducción

El presente apéndice tiene como objetivo enseñar y detallar los casos de uso más relevantes de los módulos de Filtros y Simulación.

C.2. Módulo de filtros

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, ni fecha ni 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 filtro, Filtros middle
C. Clases de interfaz	Controlador interfaz

Interfaz de usuario

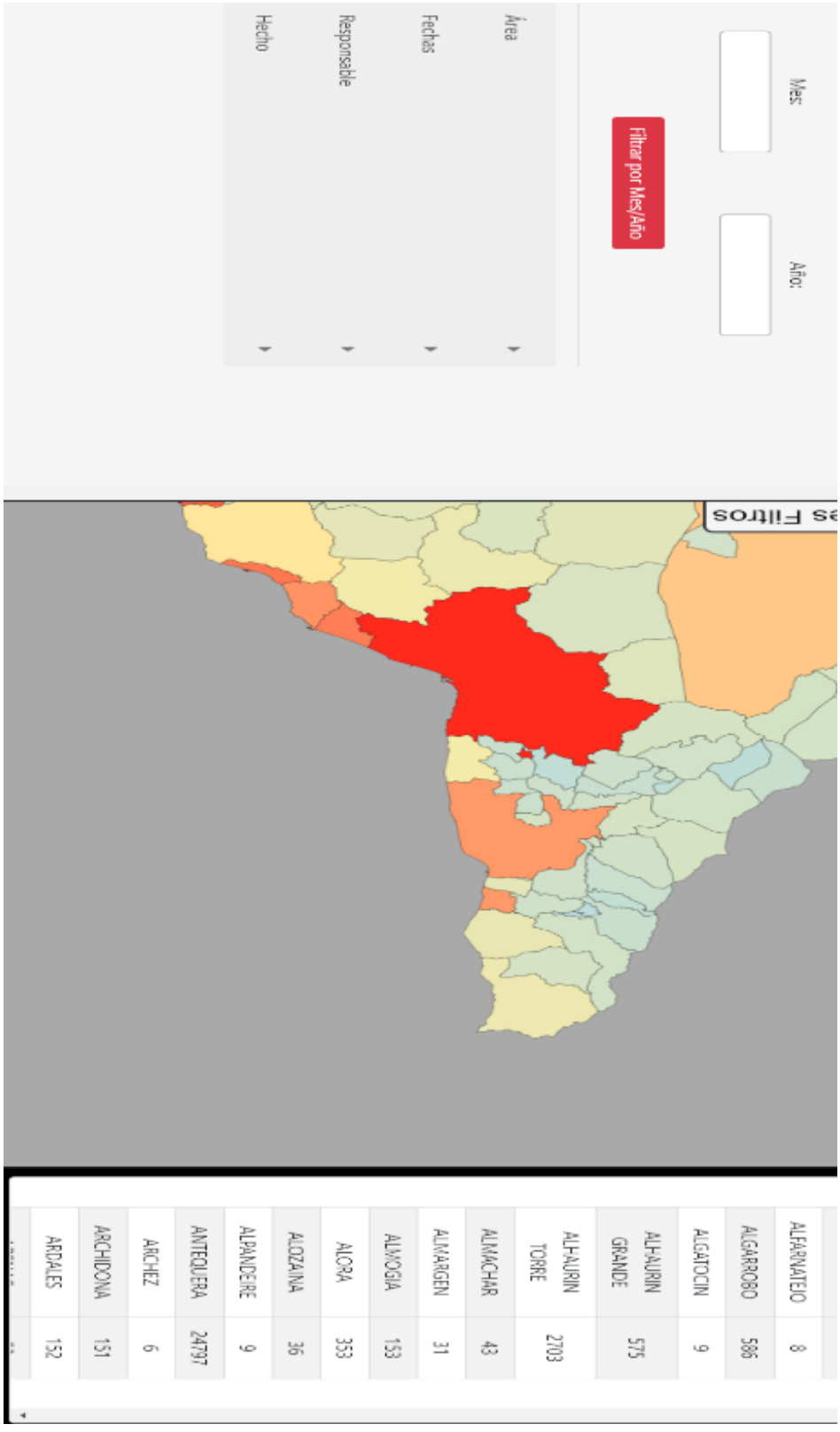
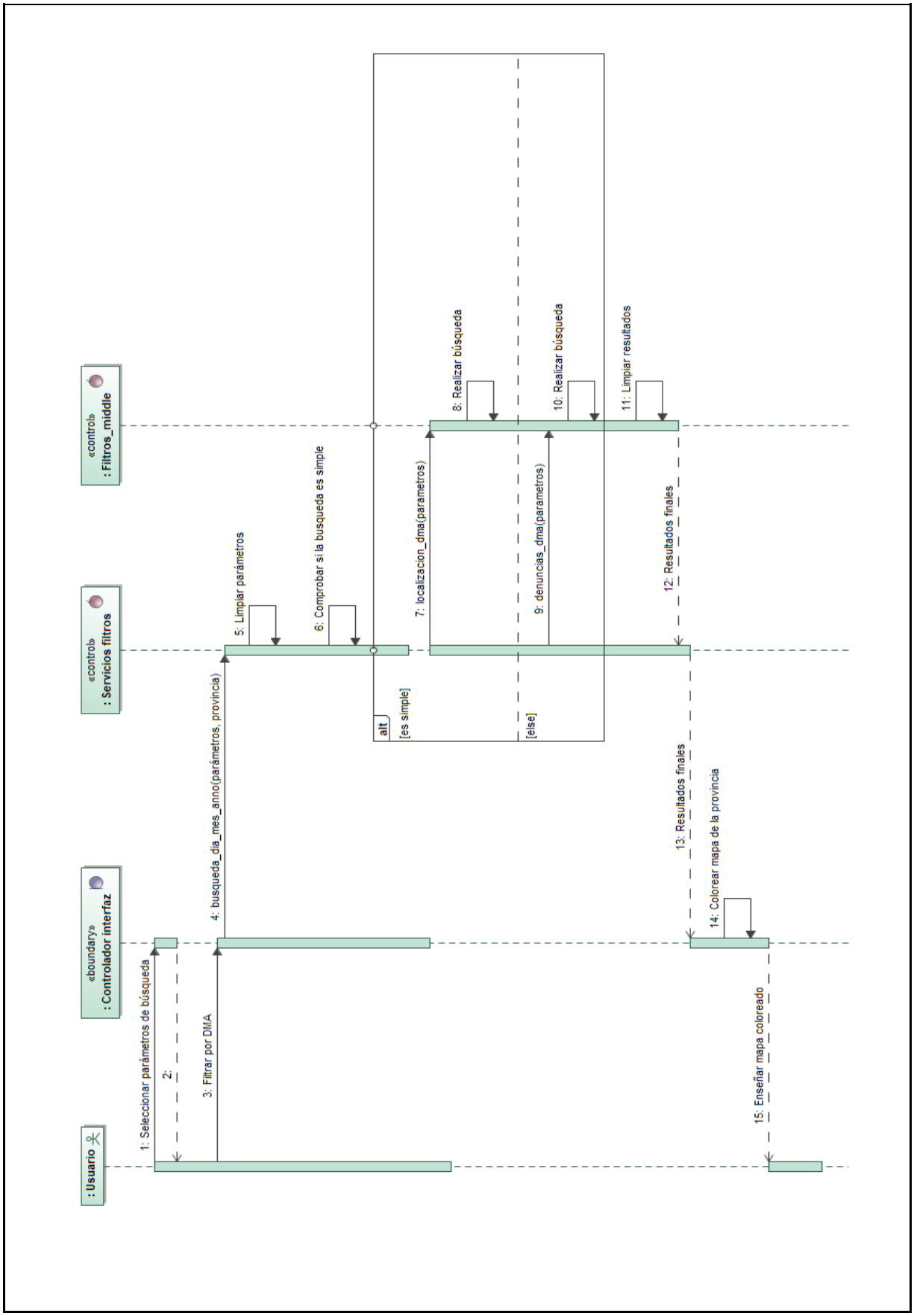


Diagrama de secuencia

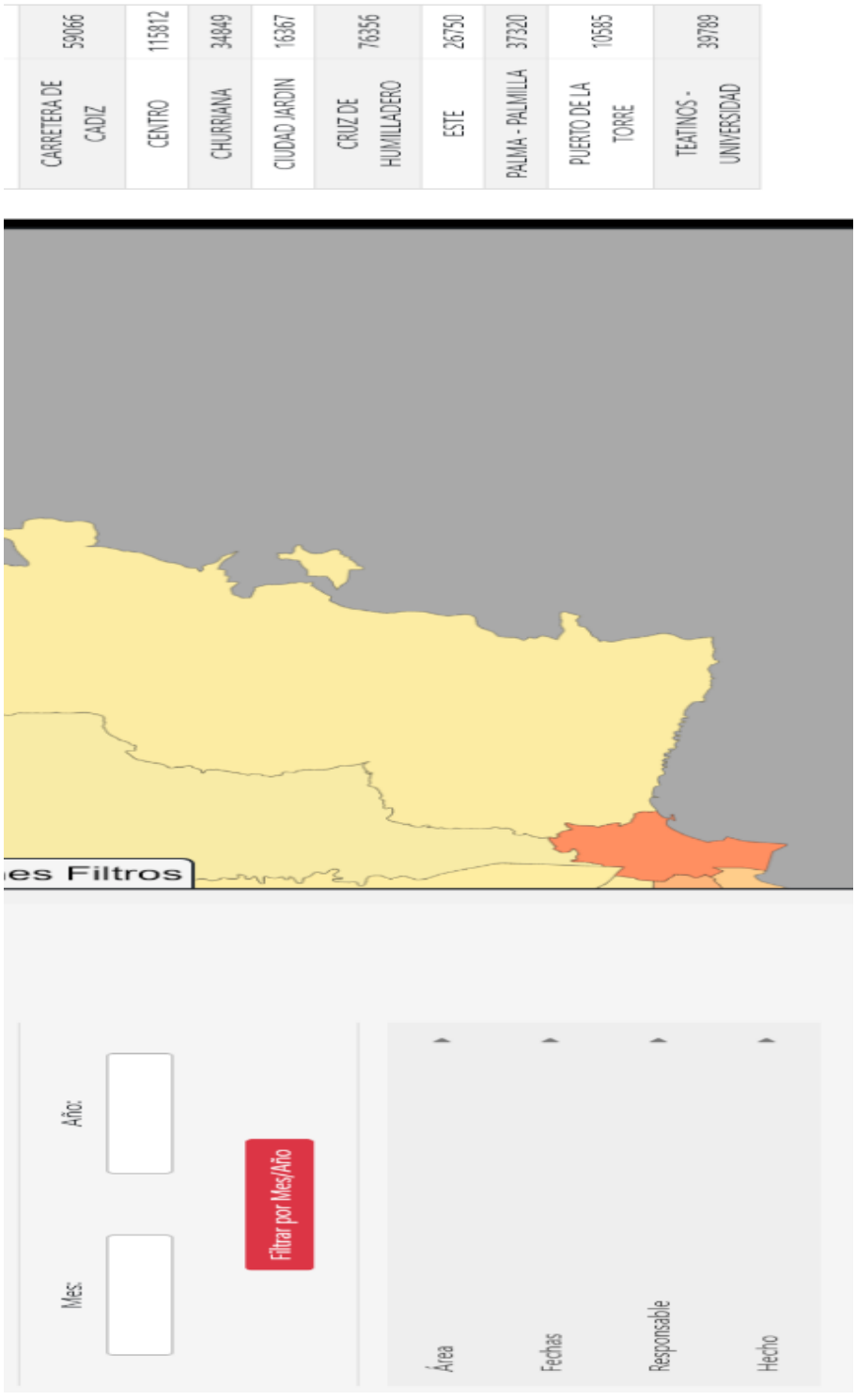

: Usuario
«boundary»
: Controlador interfaz
«control»
: Servicios filtros
«control»
: Filtros_middle
1: Seleccionar parámetros de búsqueda
2:
3: Filtrar por DMA
4: busqueda_dia_mes_anno(parámetros, provincia)
5: Limpiar parámetros
6: Comprobar si la busqueda es simple
7: localizacion_dma(parametros)
8: Realizar búsqueda
9: denuncias_dma(parametros)
10: Realizar búsqueda
11: Limpiar resultados
12: Resultados finales
13: Resultados finales
14: Colorear mapa de la provincia
15: Enseñar mapa coloreado
alt
[es simple]
[else]

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

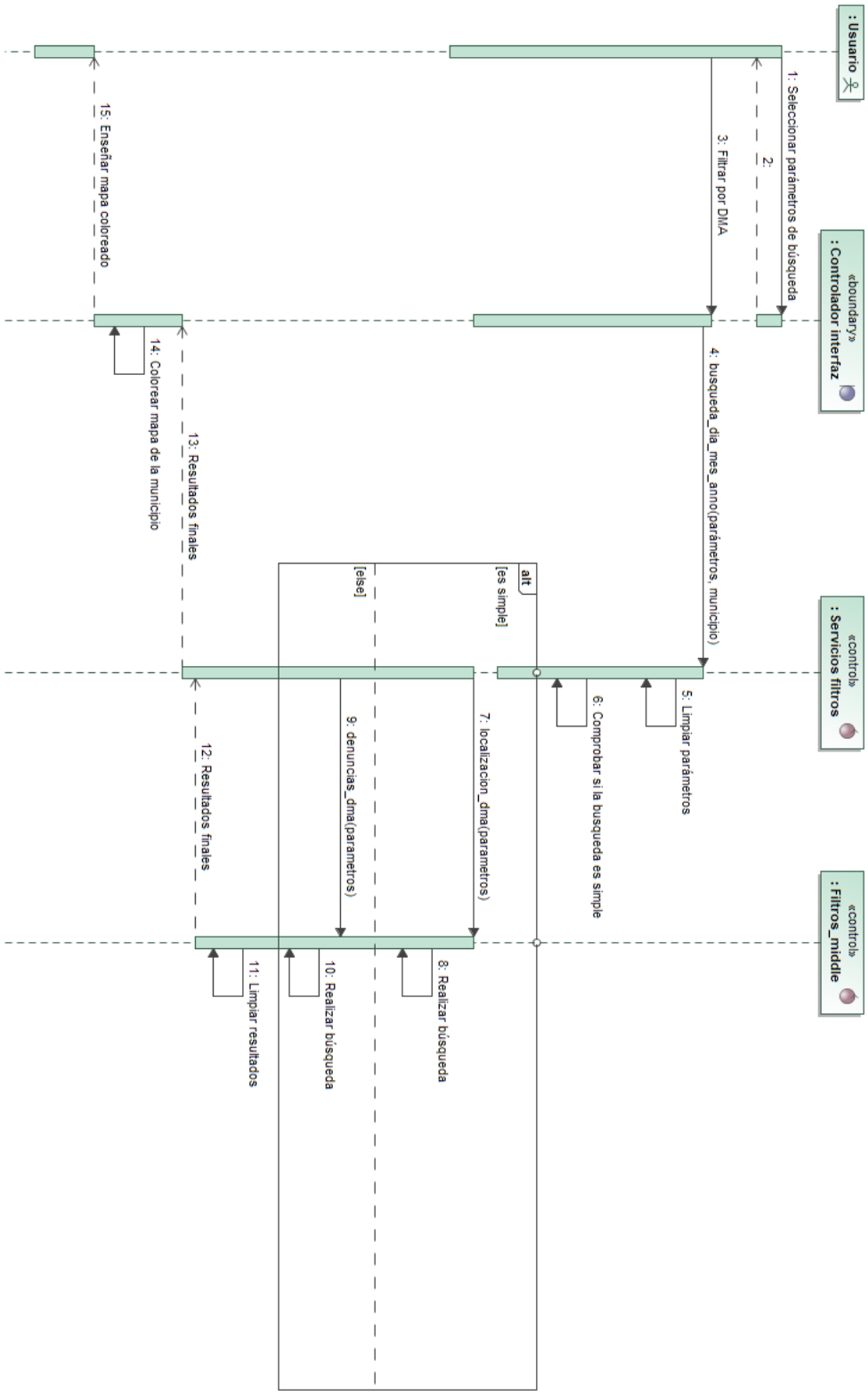
Escenario principal	
1. El usuario selecciona los parámetros de la búsqueda. 2. El usuario pulsa el botón de filtrar por DMA. 3. El sistema envía los parámetros de la búsqueda al servicio *busqueda_dia_mes_anno.* *4. Se limpian los parámetros.* *5. Se comprueba si es una búsqueda simple (un único parámetro 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 filtro, Filtros middle

C. Clases de interfaz	Controlador interfaz

Interfaz de usuario



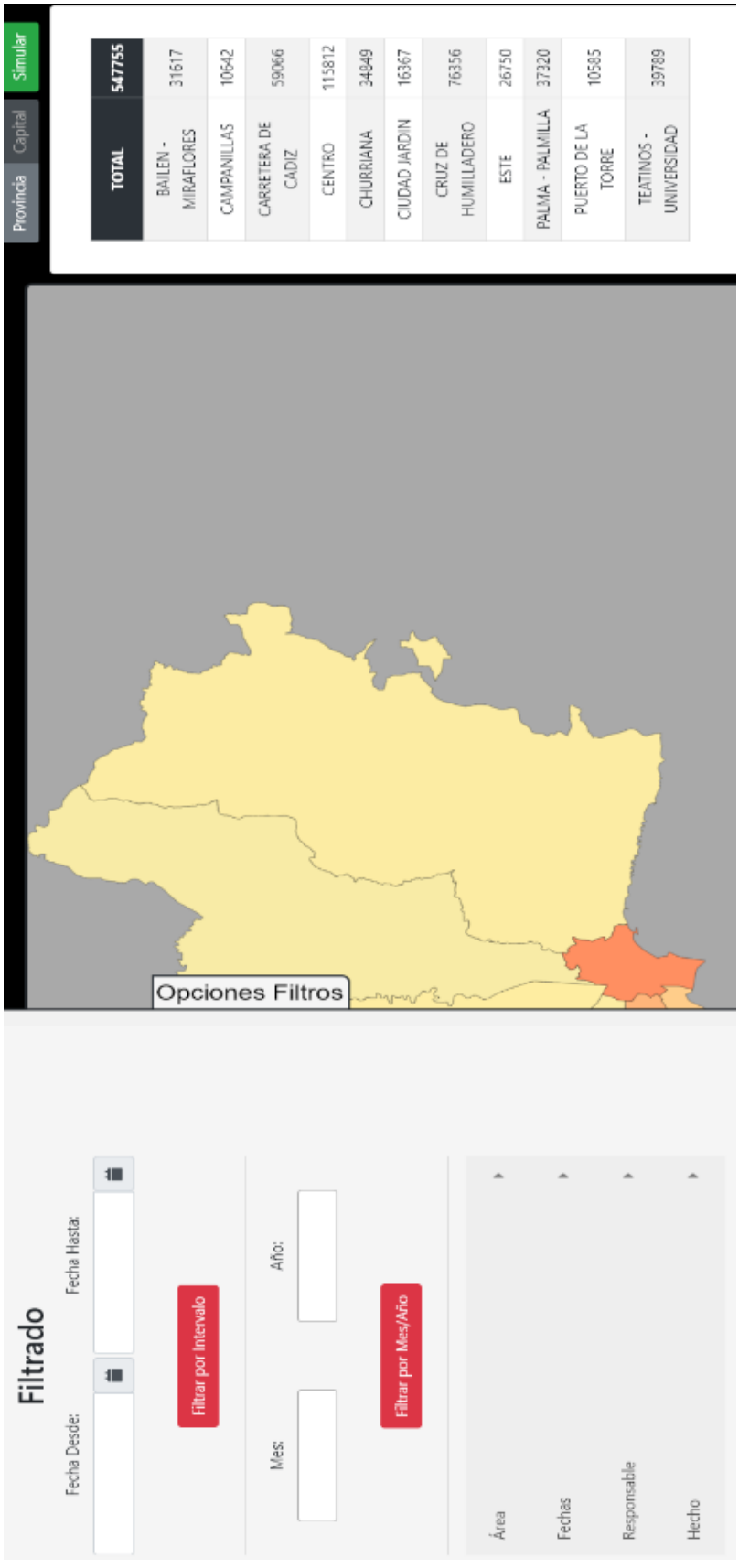
Diagramas de secuencia



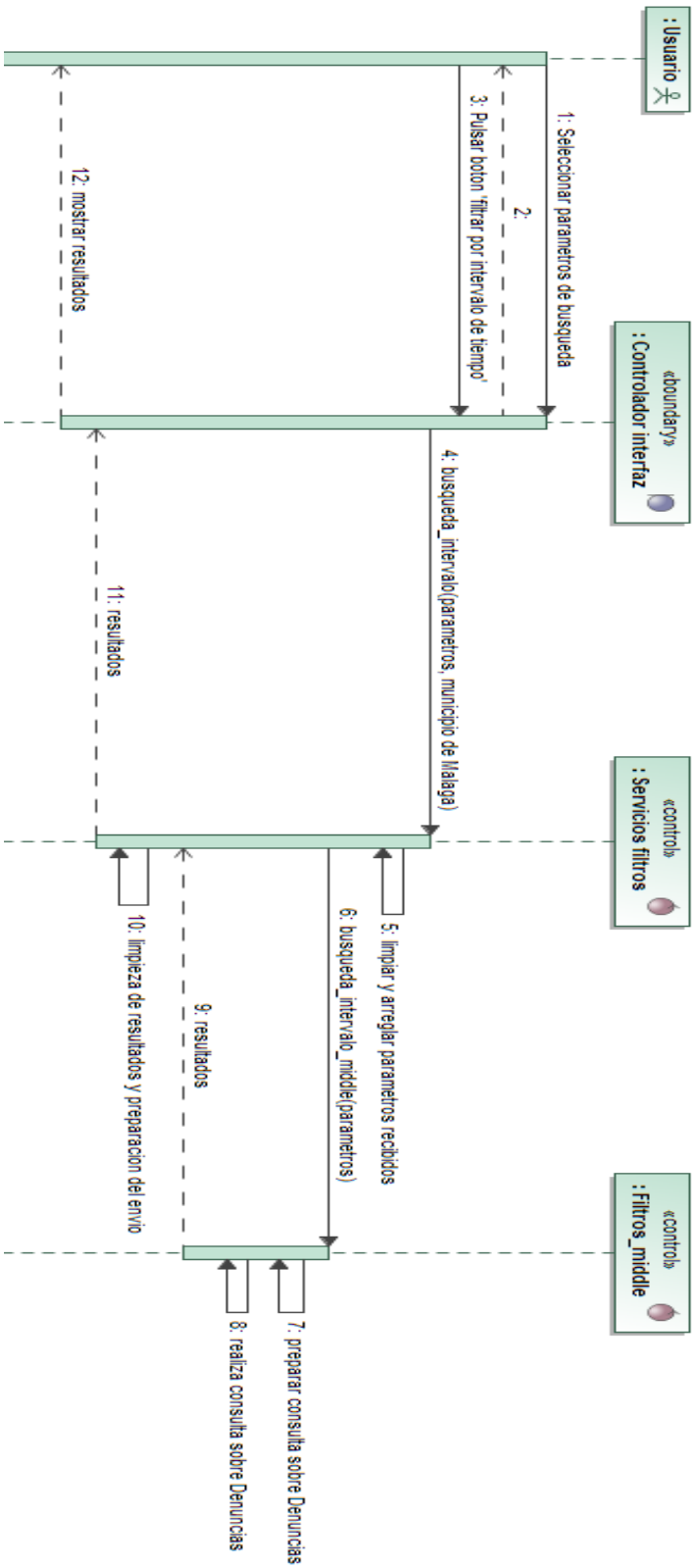
Título	Búsqueda por un intervalo de tiempo en el municipio de Málaga
Descripción	Un usuario realiza una búsqueda filtrando por un intervalo temporal y/u otros parámetros para visualizar el resultado en el mapa del municipio de Málaga.
Pre-condición	El usuario está viendo el mapa del municipio de Málaga.
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	Servicio filtros, Filtros middle
C. Clases de interfaz	Controlador interfaz

Interfaz de usuario

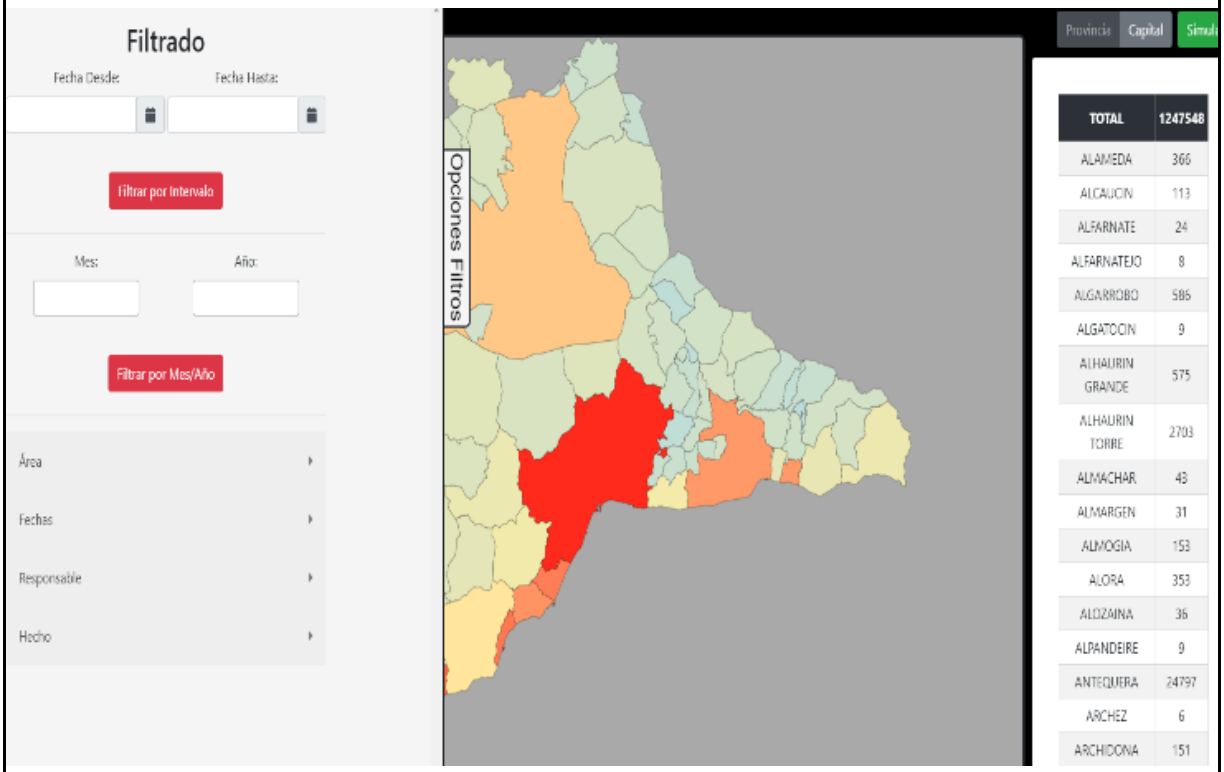


Diagramas de secuencia

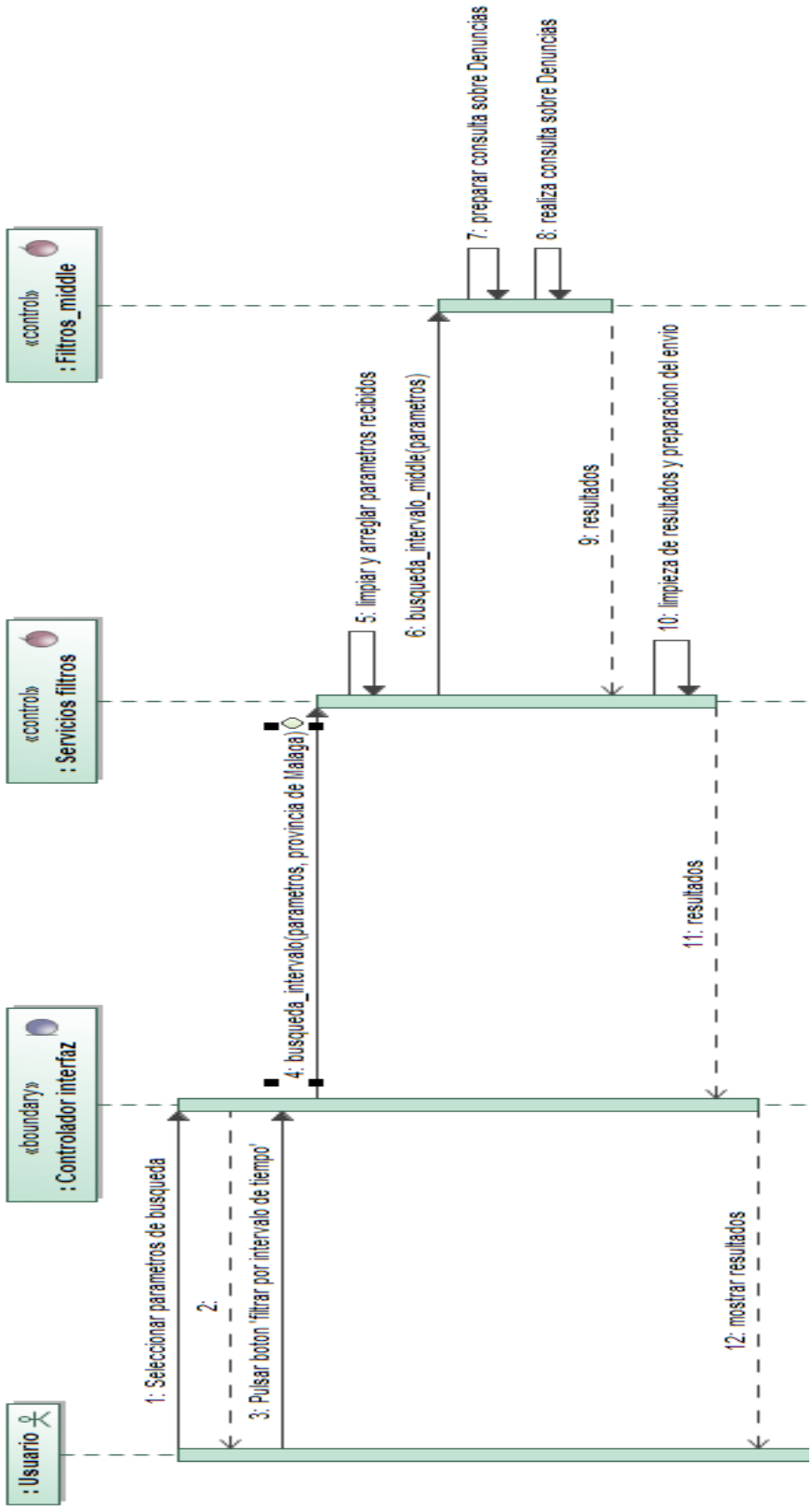


Título	Búsqueda por un intervalo de tiempo en la provincia de Málaga.
Descripción	Un usuario realiza una búsqueda filtrando por mes, año y/u otros parámetros para visualizar el resultado en el mapa de la provincia de Málaga.
Pre-condición	El usuario está viendo el mapa del municipio de Málaga.
Post-condición	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	Servicio filtros, Filtros middle
C. Clases de interfaz	Controlador interfaz
Maquetas de interfaz	



Diagramas de secuencia



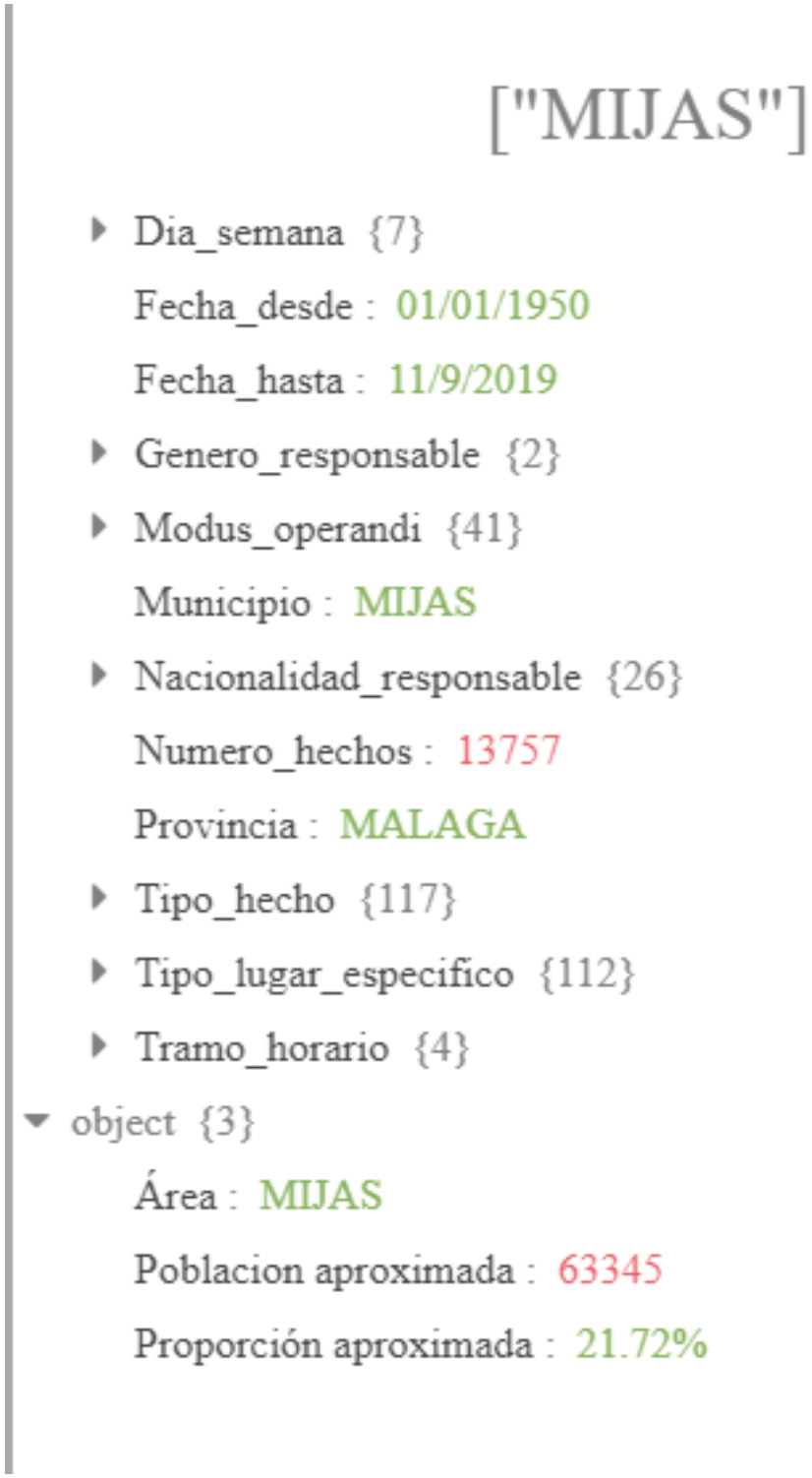
Título	Ver estadísticas de un municipio de Málaga.
Descripción	Un usuario pulsa un municipio de la provincia de Málaga para ver las estadísticas de dichos distritos después de haber o no aplicado una búsqueda en el mapa.
Pre-condición	El usuario está viendo el mapa de la provincia de Málaga.
Post-condición	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.*

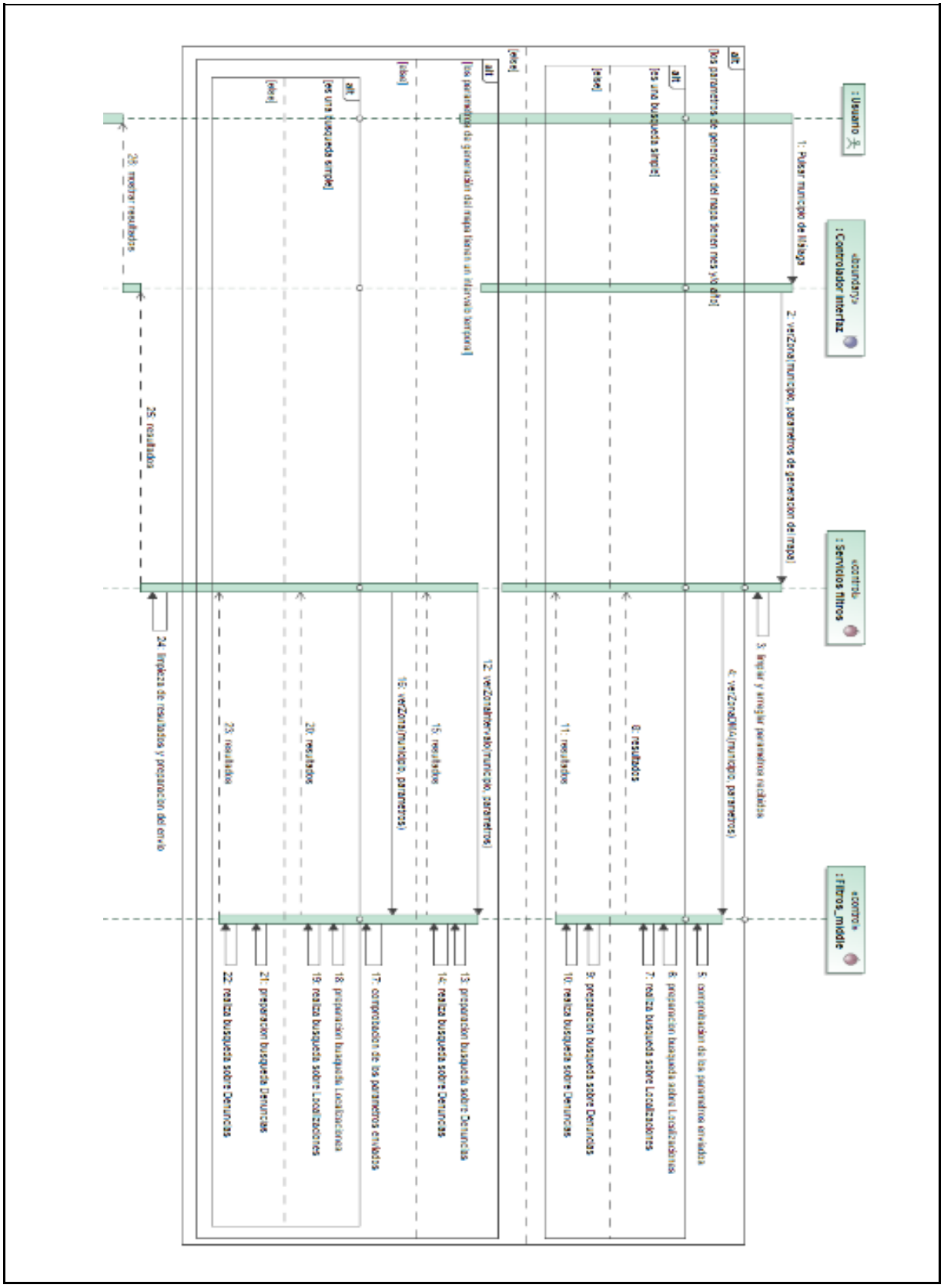
7.c.a No es una búsqueda simple 8.c.a Se prepara una búsqueda sobre Denuncias. *9.c.a Se realiza la búsqueda en MongoDB.* *10.c.a Se limpian los resultados y se preparan para el envío.* *11.c.a Se envían los resultados a la interfaz.* 7.d No es una búsqueda simple. *8.d Se prepara la consulta para la 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	Servicio filtros, Filtros middle
C. Clases de interfaz	Controlador interfaz

Maquetas de interfaz



Diagramas de secuencia


: Usuario
«boundary»
: Controlador interfaz
«control»
: Servicios filtros
«control»
: Filtros_middle
1: Pulsar municipio de listaga
2: verZonaMunicipio(municipio, parametros de generacion del mapa)
3: limpiar y almacenar parametros recibidos
4: verZonaDMA(municipio, parametros)
5: comprobacion de los parametros enviados
6: preparacion busqueda sobre Localizaciones
7: realiza busqueda sobre Localizaciones
8: resultados
9: preparacion busqueda sobre Denuncias
10: realiza busqueda sobre Denuncias
11: resultados
12: verZonaIntervalo(municipio, parametros)
13: preparacion busqueda sobre Denuncias
14: realiza busqueda sobre Denuncias
15: resultados
16: verZonaMunicipio(municipio, parametros)
17: comprobacion de los parametros enviados
18: preparacion busqueda LocalizacionArea
19: realiza busqueda sobre Localizaciones
20: resultados
21: preparacion busqueda Denuncias
22: realiza busqueda sobre Denuncias
23: resultados
24: limpieza de resultados y preparacion del envio
25: resultados
26: mostrar resultados
alt
[los parametros de generacion del mapa tienen un intervalo temporal]
[else]
alt
[es una busqueda simple]
[else]
alt
[es una busqueda simple]
[else]

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.

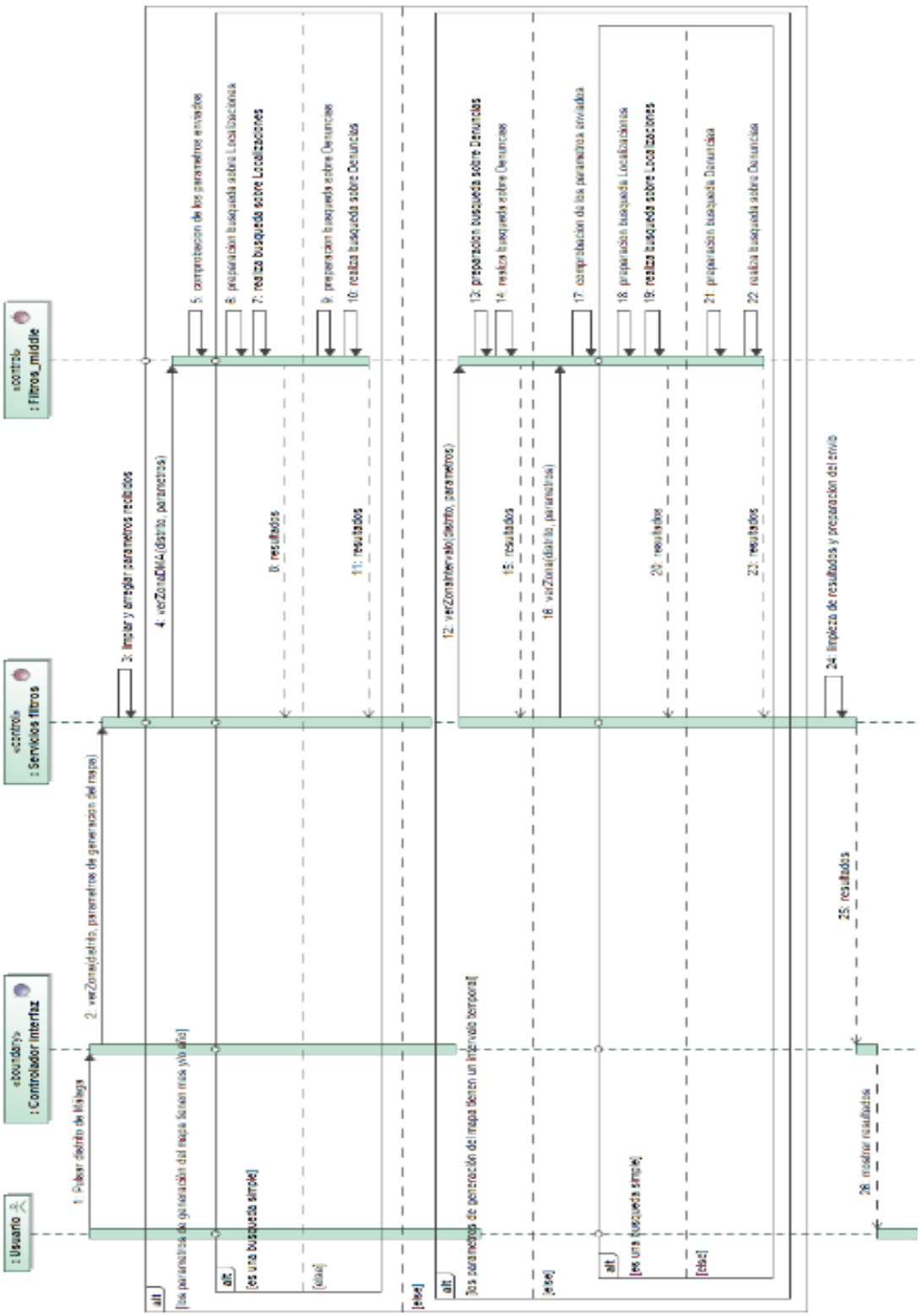
10.d Se limpian los resultados y se preparan para el envío.

11.d Se envían los resultados a la interfaz.

Clases de análisis	
A. Clases de entidad	-
B. Clases de control	-
C. Clases de interfaz	-

Interfaz de usuario

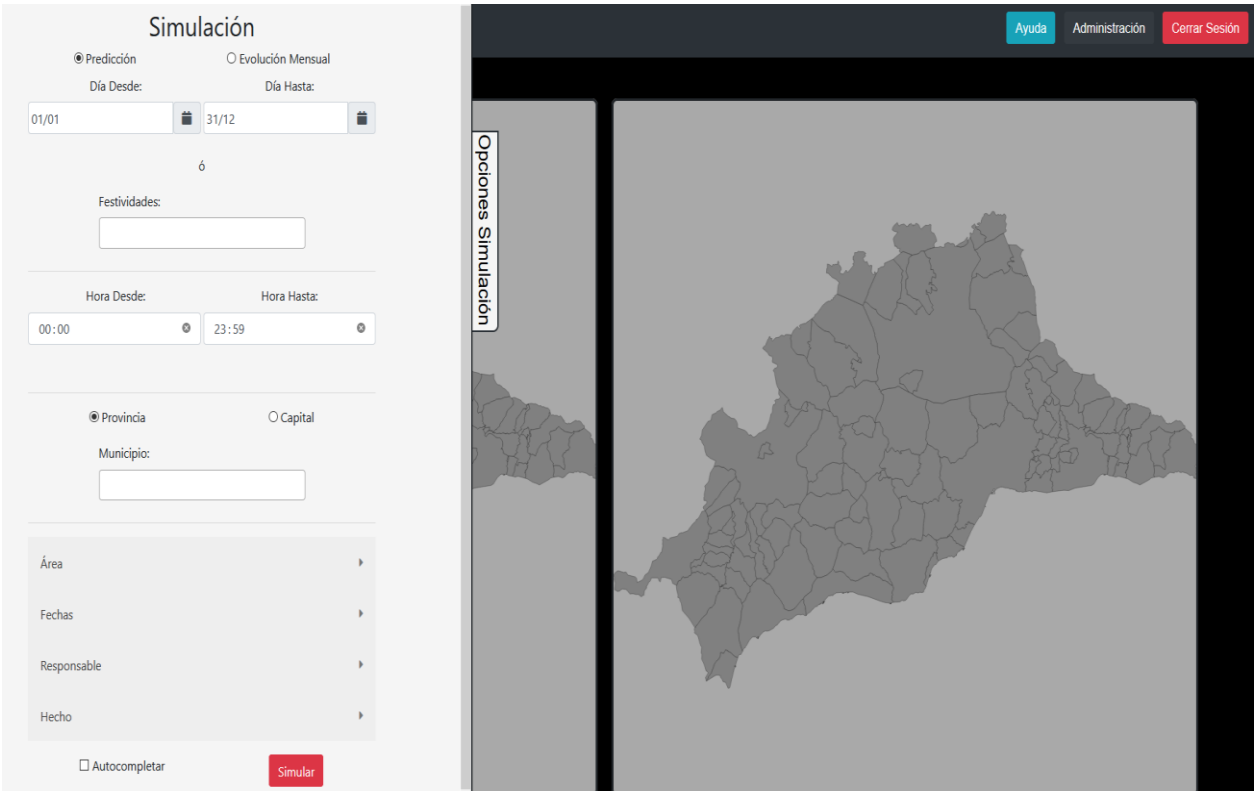
Diagramas de secuencia



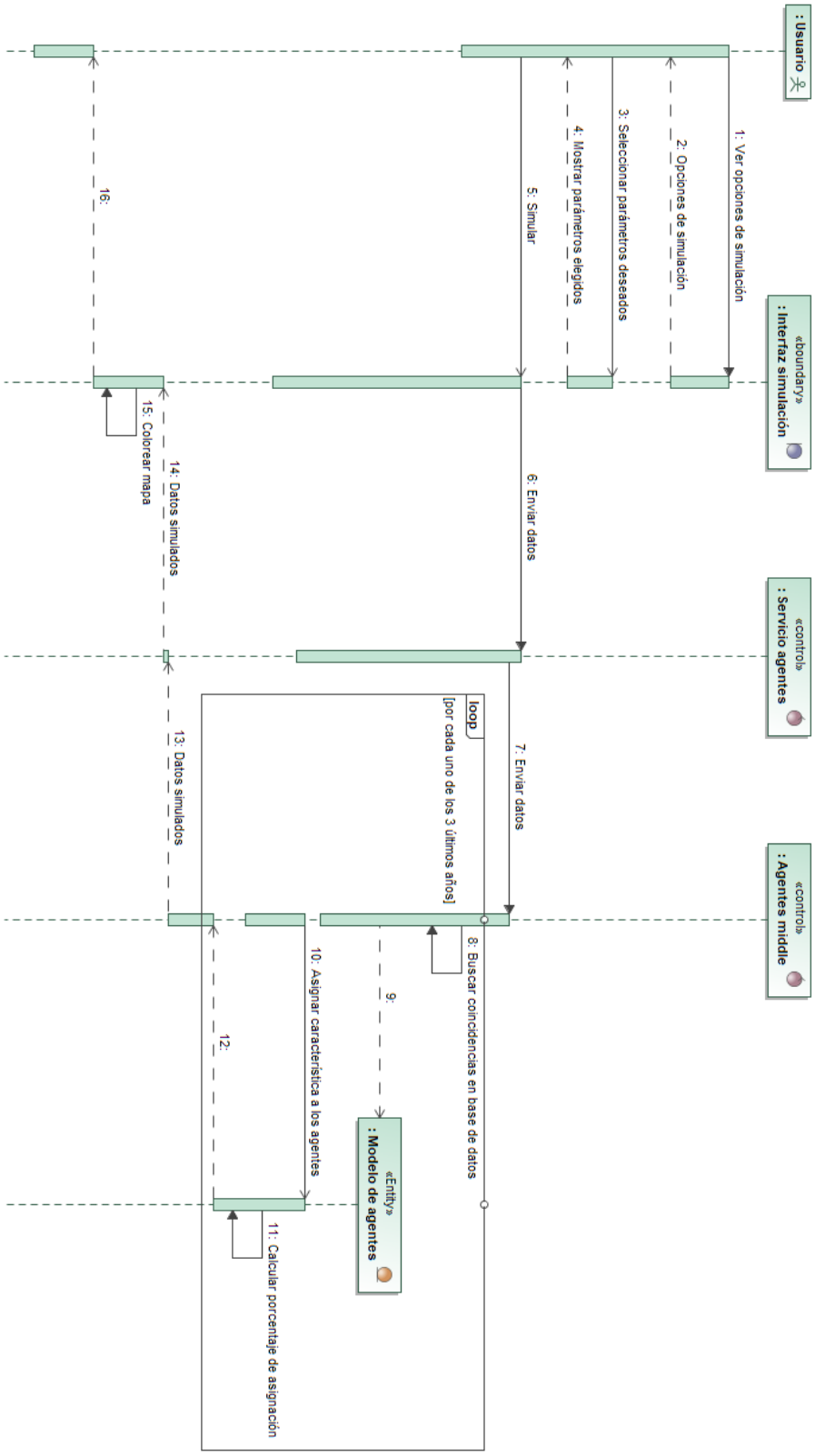
C.3. Módulo de simulación

Título	Simular
Descripción	El objetivo de este caso de uso es visualizar en dos mapas los resultados obtenidos tras la simulación de delitos, de acuerdo con unos parámetros introducidos por el usuario.
Pre-condición	Haber iniciado sesión en el sistema y encontrarse en la pantalla de simulación.
Post-condición	Tener los dos mapas de Málaga coloreados con los datos empíricos y de la simulación.
Prioridad	Media
Autor(es)	Sergio Gavilán Prieto
Escenario principal	
1. El usuario hace clic en la pestaña *Opciones Simulación*. 2. El sistema muestra un deslizable con loas opciones para la simulación. 3. El usuario selecciona los parámetros con los que desea simular. 4. El usuario hace clic en el botón *Simular*. 5. El sistema busca coincidencias en la base de datos para los datos recibidos para cada uno de los últimos 3 años. 6. El sistema crea modelos con agentes a partir de los datos obtenidos. 7. El sistema calcula el porcentaje de aproximación para la simulación realizada. 8. El sistema muestra los dos mapas coloreados, con los datos empíricos (derecha) y simulados (izquierda).	
Escenario alternativo	
5.b. El sistema muestra un mensaje de error porque el usuario ha dejado vacío un campo que se desea enviar. 5.b.1. El usuario rellena el campo. 5.b.2. El usuario hace clic en *Simular*. 5.b.3. El sistema muestra los dos mapas coloreados.	
Clases de análisis	

A. Clases de entidad	Modelo de agentes
B. Clases de control	Servicio agentes, Agentes middle
C. Clases de interfaz	Interfaz simulación
Interfaz de usuario	



Diagramas de secuencia



Título	Evolución mensual
Descripción	El objetivo de este caso de uso es visualizar en un mapa la evolución numérica de los delitos en los distintos municipios o distritos a lo largo de los meses del año, de acuerdo con unos parámetros especificados por el usuario.
Pre-condición	Haber iniciado sesión en el sistema y encontrarse en la pantalla de simulación.
Post-condición	Tener el mapa de Málaga coloreado con los datos del primer mes que se haya seleccionado.
Prioridad	Media
Autor(es)	Sergio Gavilán Prieto

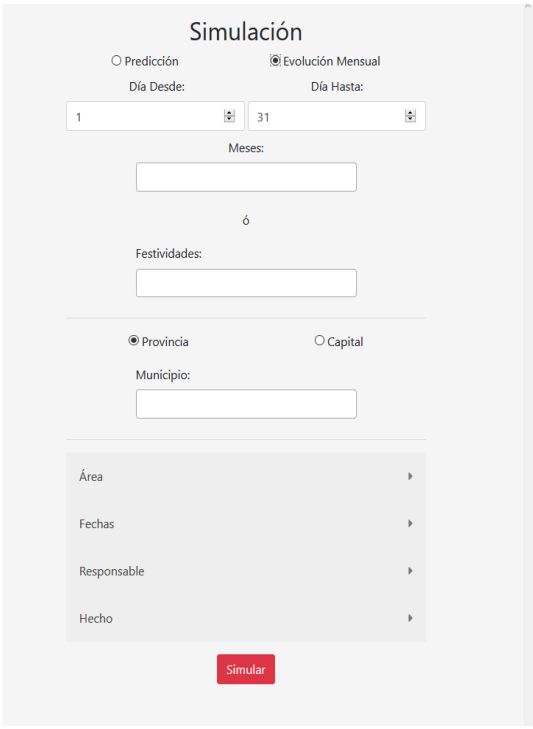
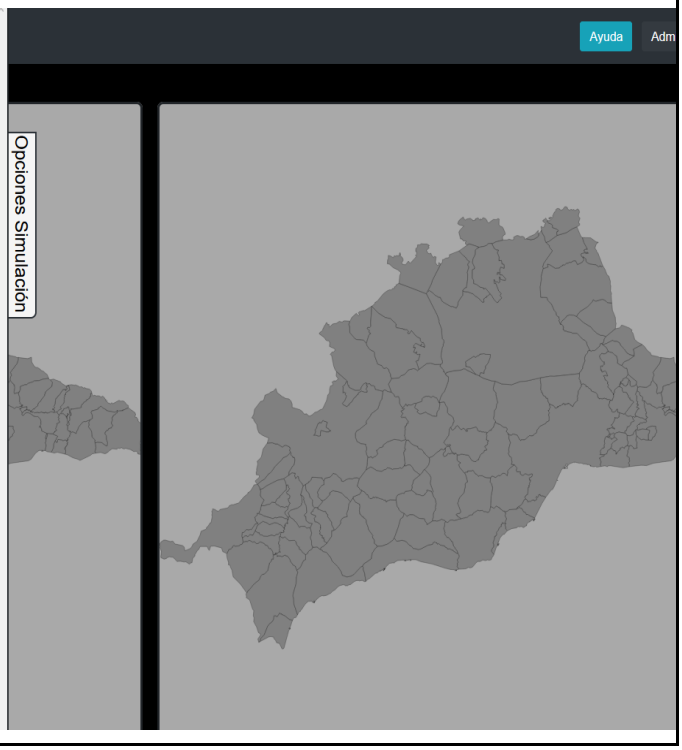
Escenario principal

1. El usuario hace clic en la pestaña *Opciones Simulación*.
2. El sistema muestra un deslizable con las opciones para la simulación.
3. El usuario hace clic en la opción *Evolución mensual*.
4. El usuario selecciona las características de los delitos, los cuales, desea ver su evolución a lo largo del año.
5. El sistema envía los parámetros al módulo *Filtros* (filtros middle) para realizar la búsqueda en la base de datos.
6. El sistema muestra un único mapa, coloreado con los datos de enero.
7. El usuario hace clic en la fecha junto al nombre del mes.
8. El sistema muestra el mapa coloreado con los datos del siguiente mes.

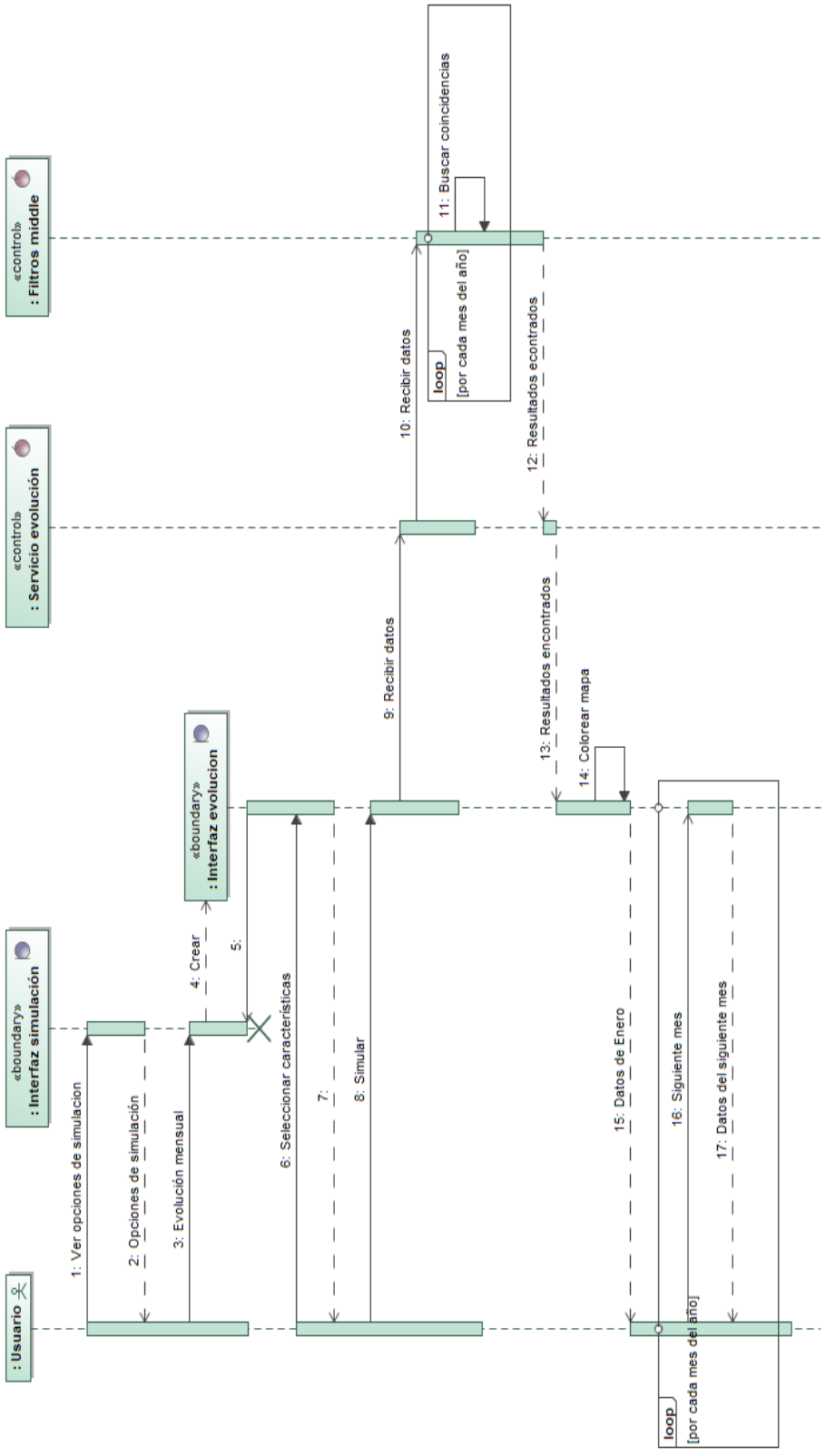
Escenario alternativo

5.b. El usuario selecciona explícitamente los meses del año en los que desea ver la evolución de los delitos.
5.b.1. El usuario hace clic en el botón *Simular*.
5.b.2. El sistema muestra un único mapa, coloreado con los datos del primer mes seleccionado.
5.b.3. El usuario hace clic en la fecha junto al nombre del mes.
5.b.4. El sistema muestra el mapa coloreado con los datos del siguiente mes seleccionado.

Clases de análisis

A. Clases de entidad	-
B. Clases de control	Servicio evolución, Filtros middle
C. Clases de interfaz	Interfaz simulación, Interfaz evolución
Interfaz de usuario	
	

Diagramas de secuencia

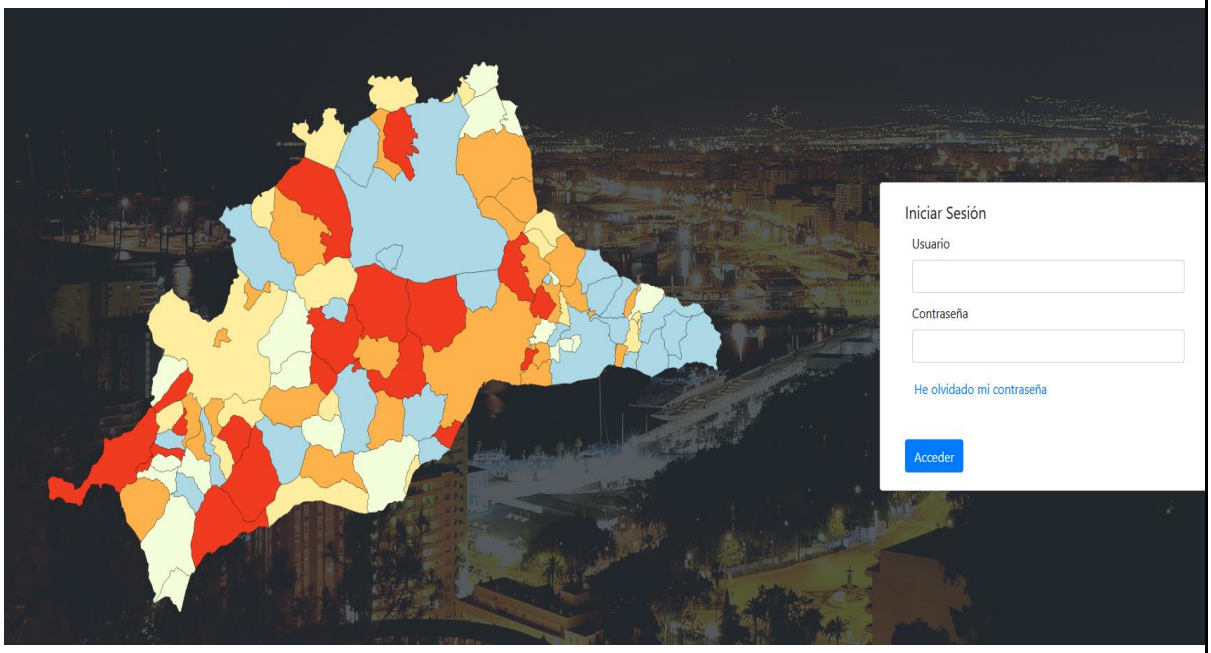


C.4. Módulo de usuarios

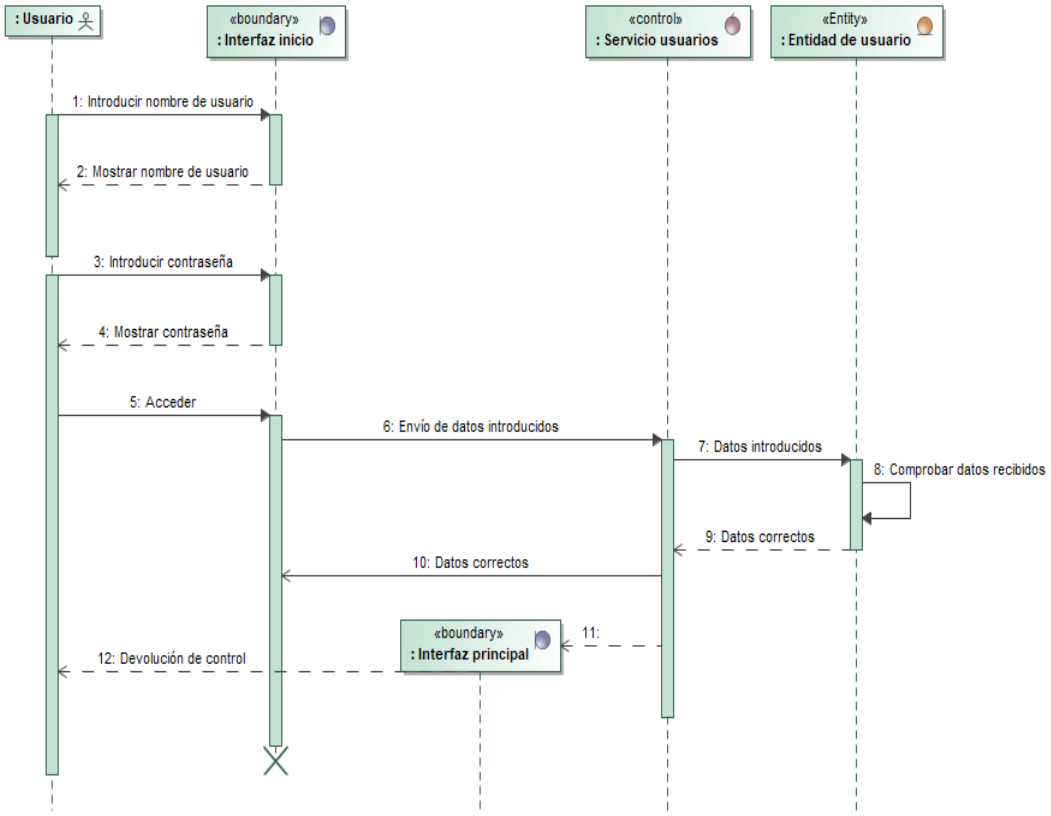
Título	Iniciar sesión
Descripción	El objetivo de este caso de uso es iniciar sesión en el sistema.
Pre-condición	Encontrarse en la página de inicio del sistema.
Post-condición	Encontrarse en la pantalla principal, con el mapa de la provincia de Málaga.
Prioridad	Alta
Autor(es)	Sergio Gavilán Prieto
Escenario principal	
1. El usuario introduce su nombre de usuario. 2. El usuario introduce su contraseña. 3. El usuario hace clic en el botón *Acceder*. 4. El sistema muestra la ventana principal del sistema: un mapa coloreado con todos los delitos en la provincia de Málaga recogidos en la base de datos.	

Escenario alternativo	
2.b. El usuario introduce una contraseña incorrecta. 2.b.1. El usuario hace clic en el botón *Acceder*. 2.b.2. El sistema muestra, en rojo, el mensaje "*Nombre de usuario o contraseña incorrecto*". 2.c El usuario no introduce su contraseña. 2.c.1. El usuario hace clic en el botón *Acceder*. 2.c.2. El sistema muestra, en rojo, el mensaje *"Por favor, rellene todos los campos"*. 2.d. El usuario introduce un nombre de usuario incorrecto. 2.d.1. El usuario hace clic en el botón *Acceder*. 2.d.2. El sistema muestra, en rojo, el mensaje "*Nombre de usuario o contraseña incorrecto*". 2.e El usuario no introduce su nombre de usuario. 2.e.1. El usuario hace clic en el botón *Acceder*. 2.e.2. El sistema muestra, en rojo, el mensaje *"Por favor, rellene todos los campos"*.	
Clases de análisis	
A. Clases de entidad	Entidad de usuario
B. Clases de control	Servicio usuarios
C. Clases de interfaz	Interfaz inicio, Interfaz principal

Interfaz de usuario



Diagramas de secuencia



Apéndice D

Listado de acrónimos

Acrónimos

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
JSON	JavaScript Object Notation
BSON	Binary JSON
CSV	Comma-separated values
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
SSL	Secure Socket Layer
SVG	Scalable Vector Graphics
D3	Data Driven Documents

SQL	Structured Query Language
NoSQL	Not only SQL
BD/DB/BBDD	Base de Datos
IDE	Integrated Development Environment
REST	Representational State Transfer
SSL	Secure Sockets Layer
ASCII	American Standard Code for Information Interchange
UTF	Unicode Transformation Format
ISO	International Organization for Standardization
UML	Unified Modeling Language
MIT	Massachusetts Institute of Technology
BSD	Berkeley Software Distribution
JWT	JSON Web Token
CNP	Cuerpo Nacional de Policía
DOM	Modelo de Objetos del Documento (Document Object Model)
URL	Identificador de recursos uniforme (Uniform Resource Locator)

CRUD	Crear, leer, actualizar y borrar (Create, read, update and delete)