UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Máster

INGENIERÍA Y CIENCIA DE DATOS

CUADRO DE MANDOS PARA EL ANÁLISIS VISUAL DE MALWARE

Autor:
Jesús Bintaned Basa

Director:
Antonio Robles Gómez

Curso 2022/2023

Convocatoria de junio

Me gustaría dedicar el esfuerzo realizado en la elaboración de este trabajo a toda mi familia que me ha apoyado desde el minuto uno y que, sin su apoyo incondicional no habría podido conseguir desarrollarlo dentro de los plazos que me había marcado.

Muchísimas gracias a ti, Silvia, por tu paciencia, por tu ayuda, y por todas las veces que me hiciste creer que realmente podía conseguirlo, estoy seguro que sin tu apoyo no habría llegado hasta aquí. Muchísimas gracias por todo amor.

Quisiera dar las gracias también a mi tutor Antonio por su apoyo y ayuda durante todo el desarrollo del proyecto.

Resumen

La tecnología es imprescindible en la vida diaria de las personas, está presente en el trabajo, en los momentos de ocio e incluso en el tiempo de descanso. Esta es una de las razones por la cual el malware también ha aumentado. El malware es un tipo de software que trata de realizar acciones dañinas a un sistema informático de forma intencionada y sin el conocimiento del usuario. Existe también otra práctica que trata de obtener información confidencial a través de la manipulación de usuarios legítimos denominada ingeniería social.

Por todo ello, es muy relevante la detección, prevención y evitación de ataques de ingeniería social en las infraestructuras de redes locales y en la nube. En este sentido, se hace esencial abordar las posibles vulnerabilidades derivadas de ataques a organizaciones y dependiendo del tipo de ataque de ingeniería social que se explote, análisis de malware, etc.

Una de las formas más efectivas de poder analizar un conjunto de datos es mediante un análisis visual, por tanto, con este trabajo se pretende proporcionar a los usuarios un panel de mandos dónde pondrán analizar de forma visual la presencia del malware en la vida diaria. Se pretende que con esta herramienta los usuarios sean capaces de extraer sus propias conclusiones y concienciarse de los peligros que el malware conlleva.

Palabras clave:

Malware, Visualización de datos, Dashboard, Arquitectura hexagonal, Angular.

ABSTRACT

Technology is indispensable in people's daily lives, it is present at work, in leisure time and even in leisure time. This is one of the reasons why malware has also increased. Malware is a type of software that tries to perform harmful actions to a computer system intentionally and without the user's knowledge. There is also another practice that tries to obtain confidential information through the manipulation of legitimate users called social engineering.

For all these reasons, the detection, prevention and avoidance of social engineering attacks on local network infrastructures and in the cloud is very important. In this regard, it is essential to address potential vulnerabilities arising from attacks on organizations and, depending on the type of social engineering attack being exploited, malware analysis, etc.

One of the most effective ways to analyze a set of data is through a visual analysis, therefore, this work aims to provide users with a dashboard where they can visually analyze the presence of malware in daily life. It is intended that with this tool users will be able to draw their own conclusions and become aware of the dangers of malware.

Keywords:

Malware, Data visualization, Dashboard, Hexagonal architecture, Angular.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FRAGMENTOS DE CÓDIGO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE GRÁFICOS

1. INTRODUCCIÓN

La ciberseguridad se ha convertido en un tema crítico en la sociedad actual, ya que la dependencia de la tecnología y la conectividad ha aumentado exponencialmente en los últimos años. Con la creciente cantidad de información personal y confidencial almacenada en línea, la necesidad de protegerla se ha vuelto cada vez más importante. Sin embargo, conforme las medidas de seguridad se vuelven más avanzadas, los delincuentes también buscan nuevas formas de comprometer los sistemas.

Dos de las técnicas más comunes utilizadas por los delincuentes informáticos son el malware y la ingeniería social. El malware es un tipo de software malicioso que se utiliza para dañar o infiltrar sistemas informáticos con el fin de robar información o controlarlos de manera remota. La ingeniería social, por otro lado, es una técnica que los atacantes utilizan para engañar a los usuarios para que proporcionen información confidencial o realicen acciones que faciliten la entrada del atacante a un sistema.

La detección, prevención y evitación de dichos ataques es fundamental para garantizar la seguridad de los sistemas informáticos y la protección de la información de los usuarios. Esta es la idea fundamental sobre la que se nutre este trabajo, el desarrollar un sistema que aporte una serie de herramientas a los usuarios para que se puedan concienciar, mediante análisis visual, del enrome problema que suponen estos ataques hoy en día.

1.1. MOTIVACIÓN

En la actualidad, el malware es una amenaza omnipresente en el mundo digital. Este tipo de programas diseñados con intenciones maliciosas que pueden causar graves daños a sistemas, robar información confidencial o interrumpir el funcionamiento normal de un dispositivo.

La propagación del malware se lleva a cabo de forma sigilosa y suele aprovechar vulnerabilidades en los sistemas y la falta de conciencia de los usuarios.

El estudio y la comprensión del malware es fundamental en el campo de la seguridad informática. Los expertos en ciberseguridad investigan y analizan el malware para descubrir sus técnicas, identificar las vulnerabilidades y para desarrollar contramedidas efectivas para eliminar el malware de los equipos.

Además, el estudio del malware ayuda a comprender mejor las tácticas y motivaciones de los ciberdelincuentes, lo que permite facilitar la protección de los sistemas y de los datos sensibles.

Por todas estas razones surge la motivación sobre el estudio y el desarrollo de este proyecto, porque en un mundo cada vez más conectado, el malware representa una amenaza persistente que requiere atención y medidas preventivas. El comprender cómo funciona el malware y estar al tanto a las últimas tendencias en su evolución es esencial para garantizar la seguridad digital tanto a nivel individual como empresarial.

Al igual que lo anterior, es importantísimo el poder llegar y el sensibilizar a la población para que se conciencie de este gran problema. Es muy importante aquí la visualización de datos, ya que es una de las mejores formas de hacer que la gente vea el auge y el peligro del malware sin tener que recopilar o consultar datos en crudo.

En resumen, la gran motivación de este proyecto es conseguir llevar un gran problema de la actualidad, como es el malware, a la población mediante una de las mejores herramientas disponibles para presentar la información, la visualización de datos.

1.2. OBJETIVOS

El objetivo principal de este proyecto es la de realizar un proyecto de ingeniería y ciencia de datos, orientado a la visualización de los mismos, que tenga como resultado un panel de mandos donde los usuarios podrán ver y analizar ciertas métricas relacionadas con el malware.

Antes de poder comenzar con el proyecto, habrá que realizar un trabajo de investigación que permita conocer a fondo el objetivo de la visualización: el malware y sus familias más importantes. También habrá que repasar los conocimientos adquiridos en el master en torno al procesamiento y la visualización de datos.

Además de todo ello, se quiere que el producto desarrollado utilice lenguajes y arquitecturas modernas, de forma que este proyecto permita al alumno estudiar y poner en práctica algunas herramientas que son utilizadas en el día a día por empresas de desarrollo de software.

En resumen, este proyecto tratará sobre un problema de ingeniería real que se tratará de resolver con las técnicas y herramientas que se esperaría que se resolviera en el mundo real. El dashboard resultante será un producto que ofrecerá las herramientas necesarias para poder tomar decisiones en base a los datos. Además de ello, el producto tendrá la calidad que se esperaría que tuviera un software desarrollado para un cliente del mundo real.

1.2.1. OBJETIVOS ESPECÍFICOS

A continuación, se van a enumerar y explicar todos los objetivos específicos que componen este proyecto y sin los cuales no sería posible el desarrollo del dashboard de malware.

- Realizar un estudio sobre el malware para poder conocer más afondo este tipo de "*software malicioso*", estudiar y comprender cuales son las

principales familias que tiene el malware y como atacan, cada una de ellas, a los dispositivos para así poder establecer las medidas necesarias para proteger los dispositivos.

- Estudio de las diferentes bases de datos existentes de malware y seleccionar la aquella que ofrezca más posibilidades de definición de métricas para las visualizaciones.
- Analizar la base de datos seleccionada para aplicar posteriormente las técnicas de limpieza y procesado de datos aprendidas durante las asignaturas del master para obtener el dataset final.
- Estudiar y comprender el dataset final para determinar las métricas más relevantes que se representarán en el cuadro de mandos de malware.
- Diseñar e implementar la arquitectura de la aplicación a desarrollar, para ello se estudiarán las llamadas "*arquitecturas limpias*", concretamente las "*arquitecturas hexagonales*", para integrarlas en el proyecto. De esta forma se desarrollará una aplicación que estará siguiendo las tendencias de desarrollo web de la actualidad.
- Definir qué tipo de visualizaciones serán las más adecuadas para cada una de las métricas, determinando para cada una los colores, tamaños, disposición, etc. acorde a lo que se quiera visualizar y destacar en cada caso. Lo que se pretende conseguir es que los usuarios puedan extraer las conclusiones de una forma sencilla, que puedan ver las tendencias y patrones de los datos muy rápidamente y hacer que lo más importante les llame la atención.
- Diseñar e implementar el cuadro de mandos. Habrá que determinar cómo ordenar en la pantalla las visualizaciones para que impacten de una mejor forma en los usuarios. Para ello se utilizará técnicas aprendidas en la asignatura de "*visualización de datos*".
- Estudiar como desplegar el sistema en la nube, de forma que pueda estar accesible para todos los usuarios que estén conectados a internet. Habrá

que estudiar cada uno de los proveedores de cloud para ver cuál de ellos proporciona mejores características a la aplicación desarrollada.

1.3. Fases del proyecto

El desarrollo de este proyecto se llevará a cabo en pequeñas iteraciones, de forma que en cada una de ellas se puedan establecer unos objetivos y funcionalidades a desarrollar. De esta forma, el proyecto estará continuamente avanzando para conseguir acabarlo dentro de los plazos establecidos.

El camino para recorrer desde el principio a la finalización de este trabajo de final de máster pasará por una serie de fases:

- **Fase de análisis**: a lo largo de esta fase se tratará de buscar información, leer la documentación que ha proporcionado el equipo docente, analizar cada uno de los datasets que se mencionan en los artículos, definir qué aspectos o características son relevantes. El objetivo de esta fase es conocer el ámbito del problema, conocer el conjunto de datos con el que se trabajará en fases posteriores e intentar conocer las características más relevantes de los datos de forma que se puedan definir métricas en base a ellas en fases posteriores.
- **Fase de transformación de los datos**: esta fase tratará, en primer lugar, de obtener un primer dataset sobre el que se podrá comenzar a trabajar. Este contendrá la información que se haya considerado relevante en la fase de análisis. Habrá que aplicar una serie de técnicas de limpieza y preprocesado de los datos para conseguir el dataset final sobre el que se basará el proyecto. Una vez que se haya conseguido el conjunto de datos final, habrá que albergarlo en alguna base de datos de las estudiadas en el máster.
- **Fase de diseño**: esta será la fase más importante del proyecto, ya que será aquí dónde se definirán las métricas que se mostrarán en el dashboard,

así como se decidirá que técnicas de visualización se utilizarán para mostrar los datos. También se realizarán los diseños arquitecturales del software a desarrollar, dando como resultado el listado con los requisitos necesarios para la implementación del software.

- **Fase de implementación**: una vez definidas las métricas y el listado de requisitos, se puede pasar a la implementación del código fuente que dará lugar al dashboard requerido para mostrar la información. Se puede dividir en dos esta fase: implementación del backend e implementación del frontend.

  En la primera fase, se desarrollará el backend del sistema, es decir, se desarrollará la parte que obtendrá los datos de la base de datos, aplicará las transformaciones que sean necesarias para las métricas que se vayan a mostrar y los devolverá mediante un API REST para que puedan ser consumidos por una aplicación web.

  La segunda fase consistirá en desarrollar la aplicación web que muestre los datos al usuario. Mediante el uso de gráficos y otras herramientas de visualización permitirá mostrarles a los usuarios las métricas propuestas para que sean capaces de comprenderlas sin ser unos expertos en la materia.

  Junto a la implementación de las funcionalidades, se llevará a cabo el desarrollo de los test correspondientes que permitirán indicar que el software desarrollado se comporta como fue diseñado.

- **Fase de documentación**: esta fase se desarrollará a la par que todas las fases anteriores, ya que todo el proceso de desarrollo del proyecto estará documentado dentro este documento técnico.

  Es importante desarrollarlo a la par ya que de esta forma todas y cada una de las decisiones que se tomen, así como el análisis, diseño y problemas serán documentados en su totalidad.

1.4. PLANIFICACIÓN

Para la culminación de este proyecto ha sido necesario dedicar una serie de recursos temporales y materiales.

En la parte temporal, este proyecto ha tenido una duración de 3 meses a tiempo parcial, ya que ha tenido que compaginarse esta elaboración con los compromisos laborales. Se realizó un trabajo aproximado de unas 6h por 5 días a la semana, es aproximado porque hubo semanas que los esfuerzos fueron mayores que otras, pero por lo general se mantuvo esos esfuerzos. El desglose de horas destinadas a cada parte del proyecto se puede ver en la tabla 1.

Tarea	Horas
Análisis	40h
Transformación de los datos	20h
Diseño	30h
Implementación	100h
Documentación	110h
TOTAL	300h

Tabla 1. Tabla con los esfuerzos temporales del proyecto.

Por otra parte, los recursos materiales utilizados están enfocados en la infraestructura utilizada para el desarrollo.

El ordenador personal que se utilizó fue un MacBook pro de 16 pulgadas del 2019, con un procesador 2,6 GHz Intel Core i7 de 6 núcleos, tarjeta gráfica Intel UHD Graphics 630 1536 MB y una memoria RAM 16 GB 2667 MHz DDR4. Actualmente, estaría valorado en unos 1500€.

Para el desarrollo del código se ha utilizado el software de JET BRAINS, concretamente se utilizó el IDE WebStorm (v2023.1). La licencia de todo el paquete que ofrece JET BRAINS cuesta 28,90€ al mes.

Finalmente, se utilizó office 360 para disponer de su suite de ofimática para poder desarrollar el presente documento. La licencia tiene un coste de 7€ al mes.

A todo esto hay que sumar los recursos humanos necesarios, cualquier proyecto de ingeniería necesita una serie de roles para poder desarrollarse:

- Un jefe de proyecto que tiene la labor de organizar el trabajo y la gestión de los recursos. Este tipo de perfil suele tener un salario medio en España de 46.000€ brutos al año, lo que se traduce en unos 3800€ brutos aproximadamente al mes.
- Un programador senior que tendrá la labor de desarrollar el código según las directrices del jefe de proyecto. Este perfil tiene un salario medio en España de 29.800€, lo que se traduce en unos 2500€ brutos aproximadamente al mes.

Hay que destacar que en este proyecto el alumno ejerció los dos roles, ya que es de desarrollo individual. En la siguiente figura se pueden apreciar los costes que hubiera supuesto este proyecto si se hubiera desarrollado en el mundo laboral.

Recurso	Coste mes	Coste total - 3 meses -
MacBook pro de 16 pulgadas del 2019	-	1500€
Licencia de JET BRAINS	28,90€	86,7€
Licencia de Microsoft Office 365	7€	21€
Salario de jefe de proyecto	3800€	11400€
Salario de programador senior	2500€	7500€
	TOTAL	20507,7€

Tabla 2. Detalle de los costes asociados a este proyecto si hubiera sido un proyecto de ingeniería real.

1.5. Estructura de la memoria

A lo largo del presente apartado se va a tratar de explicar brevemente cómo está estructurado este documento y de qué trata cada una de las secciones que lo componen.

Durante el *capítulo* 1 se da una introducción al proyecto, explicando los objetivos y fases del proyecto, así como la motivación de la elección del tema y la explicación de cómo se estructura este documento.

El *capítulo* 2 contiene el estado del arte, aquí se explican los conceptos básicos del proyecto, así como se establece un marco donde se sitúa el proyecto. Se definirá el término de malware y se presentarán a las familias más importantes. También se definirá la visualización de datos, explicando sus usos y beneficios. Se culminará el capítulo concretando el significado de panel de mandos o dashboard, que, será el resultado de la implementación de este proyecto.

En el *capítulo* 3 se mencionan todos los aspectos relacionados con el dataset que se utilizará para este proyecto. Se comenta el procedimiento seguido para la formación el dataset, posteriormente se definirán las métricas necesarias para la visualización de los datos. Posteriormente se detallan los procedimientos realizados para limpiar y preparar los datos para la visualización de las métricas. Finalmente se expondrá el resultado final del dataset.

En el *capítulo* 4 se procederá a explicar el diseño de la aplicación desarrollada, se mencionará la arquitectura elegida para el desarrollo de la aplicación, *la arquitectura hexagonal.* Partiendo de esta base, se detallarán todas las decisiones de diseño tomadas, sobre todo las decisiones a nivel de arquitectura, elección y diseño de la base de datos, elección de las tecnologías, etc.

En el *capítulo* 5 se comentará todo lo referente con la implementación de la aplicación propiamente dicha. Esta sección va ligada a la anterior, ya que será aquí donde se desarrollará todo el código que permitirá crear la aplicación siguiendo las directrices de diseño establecidas.

Finalmente, llegará el *capítulo* 6 donde se explicará todas las metodologías utilizadas para garantizar la calidad y la evaluación del producto desarrollado. En términos de machine learning, existen diferentes métricas para evaluar un algoritmo. En el desarrollo de software se utilizan los test unitarios, integración, etc. para determinar que el código desarrollado se comporta como realmente se diseñó. Además, también existen herramientas que permiten evaluar la calidad de las aplicaciones. Todo esto se abordará en este capítulo.

El *capítulo* 7 está destinado a las conclusiones que se extraen del proyecto. Mencionando cada uno de los conocimientos que se han adquirido con la investigación y elaboración de este proyecto.

En el *capítulo* 8 se mencionarán todas las posibles mejoras que se podrían realizar sobre la aplicación desarrollada en el futuro.

El *anexo* A contiene el formato que siguen los datos devueltos por el API utilizada para la obtención del dataset inicial.

El *anexo* B contine la información referente a la descarga del dataset inicial, indicando el procedimiento a seguir para obtener los datos, así como se realiza un análisis inicial de los datos recién descargados.

El *anexo* C explica como descargar el código fuente de los repositorios de GitHub y como poner en marcha la aplicación en local.

Como bien es sabido, la aplicación desarrollada se desplegará en la nube, por tanto, en el *anexo* D se explicará el proceso seguido para realizar este despliegue.

2. ESTADO DEL ARTE

A lo largo del presente apartado se va a enmarcar el contexto de este proyecto, definiendo inicialmente el concepto de Malware, pasando por sus diferentes clases y características. Posteriormente se pasará a definir el arte de la visualización de datos, mencionando en que consiste y sus características más relevantes. Finalmente se relacionará todo lo anterior para formar el dashboard o panel de mandos, que es el resultado de este proyecto.

2.1. MALWARE

El malware [1] es un tipo de software malicioso que está diseñado para dañar, comprometer o infiltrarse en sistemas informáticos sin el conocimiento o consentimiento del usuario. El nombre proviene de la abreviatura "*malicious software*", en castellano "*software malicioso*".

Por lo general, el malware es descargado e instalado por los usuarios en sus ordenadores o dispositivos móviles, ya que en muchas ocasiones estos no toman las precauciones adecuadas antes de instalar un programa descargado de internet. Una vez instalado, el código malicioso entra en el sistema y comienza a trabajar en segundo plano, por lo que es difícil detectarlos a simple vista utilizando el dispositivo.

El malware puede causar daños graves como la eliminación de archivos importantes, el robo de información confidencial, el secuestro del dispositivo para fines de extorsión o el control remoto de la máquina por parte de un atacante. Para protegerse es importante mantener el software actualizado, evitar descargar programas de fuentes no fiables, tener cuidado al acceder a enlaces o archivos adjuntos de correo electrónico desconocidos y usar antivirus y antimalware de buena reputación.

En la actualidad existen incontables tipos de malware diferentes, ya que en muchas ocasiones a partir del código de un software malicioso se pueden

generar otros diferentes. Es por esta razón por la que se clasifican por familias. Adware, Spyware, Virus, Troyanos, Gusanos, etc. son algunas de estas familias [1], [2], a lo largo de la siguiente sección se expondrán las familias más importantes de la actualidad.

2.2. Familias de Malware

Las familias de malware, como se ha comentado anteriormente, son grupos de programas maliciosos que comparten características similares, como el método de propagación, la forma de dañar el sistema y los objetivos específicos de la infección.

Cada familia tiene sus propias características y técnicas únicas de infección, lo que los hace más o menos peligrosos dependiendo de su impacto en el sistema infectado.

Las familias más utilizadas en la actualidad son: adware, spyware, ransomware, troyanos, gusanos, virus, keyloggers, botnets, malware de PUP, malware sin archivos, bombas lógicas y malware hibrido.

A lo largo de los siguientes apartados se presentará cada una de estas familias, describiendo sus características y mencionando algunos ejemplos de programas pertenecientes a dichas familias.

2.2.1. Adware

El Adware [3] es un software que muestra anuncios no deseados, a veces maliciosos, en la pantalla del dispositivo. El objetivo de este malware es generar ingresos para sus desarrolladores a través de la visualización de los anuncios o por la recopilación de información de los usuarios.

Puede venir incluido en programas legítimos descargados desde la web o puede ser instalado sin el conocimiento del usuario a través de descargas de

software no confiable. Una vez instalado, puede mostrar anuncios intrusivos mediante ventanas emergentes, anuncios en línea, etc.

Además de ser una gran molestia para el usuario, el Adware puede ralentizar el rendimiento del dispositivo, consumir datos y poner en riesgo la privacidad y seguridad de los usuarios.

Algunos ejemplos de Adware son: *Fireball*, *Appearch*, *DeskAd*, *etc.*

2.2.2. Spyware

El spyware [4] es un software diseñado para recopilar información personal y confidencial del usuario sin su conocimiento o consentimiento. Su objetivo es obtener información valiosa como contraseñas, información bancaria, datos personales, *etc.*

Puede ser instalado en un dispositivo a través de descarga de programas no fiables, correos electrónicos de phishing o a través de vulnerabilidades del sistema.

Este malware puede ser muy peligroso ya que puede comprometer la privacidad y la seguridad del usuario, permitiendo a los atacantes robar información personal para usarla posteriormente para fines malintencionados.

Algunos ejemplos de Spyware son: *CoolWebSearch*, *Gator*, *Zlob*, *etc.*

2.2.3. Ransomware

El Ransomware [5] es un software malicioso que cifra los archivos del usuario en un dispositivo para posteriormente exigir un rescate para restaurar el acceso a los mismos. Los atacantes utilizan este tipo de software como forma de extorsionar a los usuarios para obtener dinero a cambio de la liberación de los archivos cifrados.

Este tipo de malware, una vez instalado, comienza a cifrar los archivos del usuario y muestra una nota de rescate indicando la cantidad de dinero que debe pagar para desbloquear el equipo. A menudo, el pago se exige mediante criptomonedas, para poder mantener el anonimato y dificultar el rastreo de dinero. Sin embargo, no existe garantía de que los atacantes desbloqueen la información tras la realización del pago.

Además de las medidas comunes para protegerse para cualquier tipo de malware, para esta familia en concreto se recomienda la realización de copias de seguridad regulares de los archivos importantes, de forma que se pudieran restaurar en caso de infección.

Algunos ejemplos de Ransomware son: *Ryuk*, *DarkSide*, *REvil*, *etc.*

2.2.4. Troyanos

Los troyanos [6] son un tipo de malware que se *disfraza* de programas legítimos para engañar al usuario y permitir el acceso no autorizado a su dispositivo. A diferencia de otros tipos de malware, los troyanos no se propagan por sí mismos, sino que necesitan otro software para poder instalarse en el dispositivo.

Una vez que se ha instalado puede permitir a los atacantes el acceso remoto a dicho dispositivo, lo que les permite el robo de información personal y confidencial, o el control del dispositivo, etc. Es por esta razón por la que pueden ser muy peligrosos, porque permiten acceder al dispositivo infectado de forma remota sin ser detectados.

Existen muchos tipos de troyanos en circulación, algunos de ellos son más dañinos que otros. Afortunadamente, la mayoría se pueden detectar de forma rutinaria y eliminar con los mejores softwares antivirus.

Algunos ejemplos de Troyanos son: *Storm Worm*, *Zeus*, *Qbot*, *etc.*

2.2.5. Gusanos

Los gusanos [7]–[9] son un tipo de malware que se propagan a través de redes y sistemas conectados a internet. Pueden tener diferentes objetivos, como recopilar información, dañar el sistema operativo, robar credenciales y propagar otros tipos de malware.

Una vez que un gusano infecta un dispositivo, puede replicarse y propagarse a otros dispositivos a través de la red. A diferencia de los virus, los gusanos no necesitan un archivo huésped para infectar un sistema y se propagan de manera autónoma aprovechando vulnerabilidades en el sistema operativo o aplicaciones.

Pueden ser muy peligrosos porque pueden propagarse rápidamente a través de la red y causar daños graves en los sistemas infectados. Además, al propagarse de forma autónoma, pueden ser muy difíciles de detectar y eliminar.

Algunos ejemplos de gusanos son: *Conficker*, *Nimda*, *Code Red*, *etc.*

2.2.6. Virus

Los virus informáticos [10] son un tipo de malware que se propagan a través de archivos infectados y se ejecutan en un dispositivo sin el conocimiento o el consentimiento del usuario. Pueden estar diseñados para dañar, modificar o destruir archivos y programas en el dispositivo infectado, o para robar información personal y confidencial.

Este tipo de malware, a diferencia de otros, requieren la interacción del usuario para propagarse. Una vez que se instala en un dispositivo, puede replicarse y propagarse a otros dispositivos a través de redes informáticas y dispositivos conectados a Internet.

Algunos virus también pueden permanecer latentes en un dispositivo hasta que se activan mediante un desencadenante específico, como una fecha determinada o un evento del sistema.

Algunos ejemplos de virus son: *ILOVEYOU*, *Melissa*, *WannaCry*, *etc.*

2.2.7. KEYLOGGERS

Los keyloggers [11], [12] son un tipo de malware diseñados para registrar las pulsaciones de teclas de un dispositivo y enviar la información a un tercero. Pueden ser tanto software como hardware pero su objetivo es común: registrar las contraseñas, nombres de usuario, números de tarjetas de crédito... cualquier tipo de información sensible que se introduce en el dispositivo infectado.

La forma más común de que este tipo de malware sea instalado en un dispositivo es mediante un ataque de phishing, pero también se puede mediante la descarga desde la web o incluso mediante el acceso físico al dispositivo.

2.2.8. BOTNETS

Los botnets [13], [14] son una red de dispositivos infectados por malware que son controlados de forma remota por una atacante. Estos dispositivos pueden ser ordenadores, servidores, móviles, routers, etc.

Son creados con el objetivo de realizar ataques DDoS (ataque de denegación de servicio distribuido), enviar SPAM, propagar malware, etc. Los dispositivos infectados se llaman bots y pueden ser controlados de forma remota por el atacante.

Los botnets pueden ser difíciles de detectar y desmantelar debido a su naturaleza descentralizada y la distribución geográfica de los dispositivos infectados. Además los atacantes pueden utilizar técnicas de cifrado y ofuscación para ocultar el tráfico malicioso y evitar así ser detectados.

2.2.9. MALWARE DE PUP

El malware de PUP[1] [15]–[17] es un tipo de software malicioso que se presenta como un programa útil, pero en realidad es dañino y puede causar problemas en el dispositivo infectado.

Puede incluir programas que instalan software adicional no deseado sin el consentimiento del usuario, herramientas que realizan cambios no autorizados en la configuración del navegador, o programas que realizan un seguimiento de la actividad del usuario y recopilan información personal sin su conocimiento.

A menudo, el malware PUP se distribuye mediante la descarga de software gratuito de fuentes no confiables, y puede ser difícil de eliminar completamente debido a su naturaleza persistente y la capacidad de reinstalarse después de la eliminación.

Para protegerse de este tipo de malware concretamente es importante descargar software sólo de fuentes confiables, leer cuidadosamente los términos y condiciones antes de instalar cualquier programa, y utilizar software antivirus y antimalware para detectar y eliminar cualquier malware potencial en el dispositivo.

Algunos ejemplos de malware de PUP son: *Crossrider*, *OpenCandy*, *Wajam*, *etc.*

2.2.10. MALWARE SIN ARCHIVOS

El malware sin archivos [18]–[20] es un tipo de programa malicioso que no utiliza archivos para infectar un sistema informático. Sino que aprovecha las herramientas y aplicaciones ya existentes en el dispositivo, como los scripts

[1] El término PUP significa "*Programas Potencialmente no Deseados*", y se refiere al software que puede ser considerado no deseado por los usuarios debido a su comportamiento o función.

del sistema operativo o aplicaciones legítimas, para llevar a cabo sus actividades maliciosas.

Funciona mediante la explotación de vulnerabilidades en los procesos en ejecución, y puede ser difícil de detectar y eliminar debido a que no deja rastros en el disco duro del dispositivo. En lugar de eso, este malware se carga en la memoria del sistema y se ejecuta en segundo plano, lo que lo hace difícil de detectar con herramientas de seguridad tradicionales.

El malware sin archivos puede ser utilizado para robar información confidencial, instalar más malware en el dispositivo, o para realizar ataques DDoS (ataques de denegación de servicio). A menudo se distribuye mediante el phishing o la ingeniería social.

Algunos ejemplos de malware sin archivos son: *Poweliks*, *Kovter*, *DNSMessenger*, *etc.*

2.2.11. Malware hibrido

El malware híbrido [21], [22] es una combinación de diferentes tipos de malware que se utilizan juntos para llevar a cabo un ataque mucho más sofisticado que el que podría realizarse con un solo tipo de malware. Por ejemplo, puede ser una combinación de un troyano y un gusano, o un virus y un keylogger, etc.

Su objetivo es aprovechar las debilidades de diferentes tipos de sistemas de seguridad para evitar su detección y propagación en el sistema infectado. Además, el malware híbrido puede evadir las herramientas de seguridad tradicionales y explotar múltiples vectores de ataque al mismo tiempo.

Este tipo de malware es muy peligroso porque puede cambiar su comportamiento y estructura en tiempo real, lo que dificulta su detección y eliminación. Además, puede personalizarse para atacar objetivos específicos o redes de computadoras en particular.

Algunos ejemplos de malware hibrido son: *Stuxnet*, *Shamoon*, *Emotet*, *etc.*

2.3. Distribución de Malware

En las secciones anteriores se han comentado algunos de los tipos de Malware que existen en la actualidad. En sí, para que un dispositivo sea infectado por un software malicioso, tiene que ocurrir un factor muy importante: **el software tiene que entrar al dispositivo para poder infectarlo.** Para cumplir su objetivo de infiltrarse en sistemas y dispositivos, el malware utiliza diversos métodos de distribución.

A continuación se exponen algunos de los métodos de distribución [23] de malware más utilizados por los atacantes para infectar a los dispositivos de los usuarios.

2.3.1. Descarga de ficheros de la web

El método de distribución de malware a través de la descarga de archivos desde la web consiste en ofrecer archivos maliciosos para su descarga en sitios web, lo que lleva a que los usuarios los descarguen inadvertidamente y se infecten con malware.

Existen varias formas en las que los atacantes utilizan este método:

- Sitios web maliciosos: Los delincuentes crean sitios web aparentemente legítimos que ofrecen contenido atractivo y que, cuando los usuarios intentan descargar estos archivos, se les proporciona un enlace que descarga un archivo infectado en lugar del archivo deseado.
- Redes de intercambio de archivos: Los delincuentes suelen ocultar malware dentro de archivos legítimos o los renombran con nombres engañosos para que los usuarios descarguen el archivo infectado sin darse cuenta.

- Anuncios falsos: Algunos sitios web pueden mostrar anuncios falsos que simulan descargas legítimas. Estos anuncios suelen ser engañosos y pueden llevar a los usuarios a descargar archivos maliciosos en lugar del contenido esperado.
- Descargas no autorizadas: Los atacantes pueden aprovechar las vulnerabilidades de los sitios web legítimos para insertar enlaces o scripts maliciosos en páginas específicas.

2.3.2. WEB DRIVE-BY

El método de distribución de malware conocido como "*drive-by*" se refiere a una técnica en la cual los usuarios son infectados con malware simplemente al visitar un sitio web comprometido, sin necesidad de que realicen ninguna acción adicional, como descargar un archivo o hacer clic en un enlace.

Aquí hay algunos aspectos clave relacionados con el método de distribución de malware "*drive-by*":

- Vulnerabilidades de software: Los atacantes buscan y explotan las vulnerabilidades en el software utilizado en los servidores web o en los navegadores de los usuarios. Estas pueden permitir que el malware se inserte en el código del sitio web o se descargue en el dispositivo del usuario sin su conocimiento.
- Inyección de código malicioso: Los atacantes pueden insertar código malicioso, como JavaScript o HTML, en el sitio web comprometido. Este código aprovecha las vulnerabilidades en los navegadores o complementos del usuario para descargar e instalar automáticamente el malware en su dispositivo.
- Redirecciones no deseadas: Los atacantes pueden utilizar técnicas de redirección para enviar a los usuarios de un sitio web legítimo a otro sitio web malicioso sin su conocimiento.

- *Exploit Kits*: Los *exploit kits* son herramientas utilizadas por los atacantes para automatizar y simplificar el proceso de distribución de malware *drive-by*. Estos kits aprovechan varias vulnerabilidades conocidas en el software comúnmente utilizado y, cuando un usuario visita un sitio web comprometido, el *exploit kit* identifica las vulnerabilidades en su sistema y las explota para entregar el malware.

2.3.3. EMAILS

El método de distribución de malware a través de correos electrónicos es una táctica comúnmente utilizada por los ciberdelincuentes para infectar los dispositivos de los usuarios. Consiste en enviar emails que contienen enlaces maliciosos o archivos adjuntos infectados que, al ser abiertos o descargados, pueden desencadenar la instalación de malware en el dispositivo del usuario.

Existen varias formas de descargar el malware de un correo electrónico malicioso:

- Enlaces maliciosos: Los ciberdelincuentes incluyen enlaces maliciosos en los correos electrónicos, que pueden llevar a sitios web comprometidos o a páginas diseñadas específicamente para distribuir malware.
- Descarga de archivos adjuntos infectados: Los archivos adjuntos en los correos electrónicos pueden contener malware, que se suelen disfrazar como documentos legítimos, al abrir o ejecutarlos se desencadena la instalación del malware en el dispositivo del usuario.
- Ingeniería social: Los ciberdelincuentes a menudo utilizan técnicas de ingeniería social para engañar a los usuarios y hacer que interactúen con los enlaces o archivos adjuntos maliciosos.
- Phishing: Los ciberdelincuentes envían correos electrónicos que parecen provenir de empresas o servicios conocidos, solicitando al usuario que haga clic en un enlace malicioso o descargue un archivo de malware adjunto para realizar una acción urgente.

2.4. VISUALIZACIÓN DE DATOS

La visualización de datos fue explorada en detalle dentro de una asignatura del presente máster. En ella se definía la visualización como "*la representación y presentación de datos que explota las capacidades de percepción visual para amplificar la cognición orientada hacia un objetivo*"[24]. Dicho de otra forma, la visualización de datos es el proceso de utilizar elementos visuales, como gráficos o mapas, para representar los datos. De esta forma, se trasladan datos complejos, de alto volumen o numéricos a una representación visual más fácil de procesar.

La visualización de datos es muy importante dentro de las empresas modernas, ya que estas suelen procesar grandes volúmenes de datos procedentes de diversos orígenes. Los datos sin procesar pueden ser muy difíciles de comprender y de usar. Es por esta razón que los científicos de datos preparan y presentan los datos según el contexto más adecuado. Les dan forma visual para que los responsables de la toma de decisiones puedan identificar relaciones, detectar patrones, etc.

2.4.1. BENEFICIOS DE LA VISUALIZACIÓN DE DATOS

Gracias a una correcta visualización de los datos se pueden conseguir una serie de beneficios que ayudarán a entender los datos a los usuarios a los que vaya dirigida. Hay una gran cantidad de beneficios, a continuación se exponen los más importantes [25]:

- **Más atención a los detalles.** Cuando los datos se encuentran representados se pueden descubrir de forma sencilla todo tipo patrones, tendencias, formas, etc. Además, en muchas ocasiones también se puede hacer zoom en ciertas partes de las representaciones para poder llegar a la granularidad que se desee.
- **Respuesta emocional.** Es evidente que todos los estímulos que entran por los ojos de una persona le generan una serie de emociones. Es por esta

razón, que representando los datos mediante un gráfico, una imagen, etc. provocarán una reacción más grande en un usuario que un conjunto de datos en crudo[2].

- **Comparaciones sencillas.** Gracias a la representación visual de la información, se puede comparar de una forma mucho más sencilla dos, o más, tendencias o patrones. Incluso se pueden comparar en un mismo gráfico y ver rápidamente en que se diferencian.

  Realizar comparaciones sin representación visual puede ser una tarea ardua y costosa, ya que habría que comparar toda la información de un resultado con los otros. Además del aumento del tiempo también implicaría un aumento del margen de error.

- **Rápidas predicciones.** Gracias a las tendencias y patrones que se presentan en los datos es muy sencillo poder hacer una predicción sobre qué ocurrirá en el futuro.

  Basta con imaginar la repetición de esa tendencia o patrón específico para poder sacar las conclusiones. No hay que realizar ningún tipo de cálculo, ni procesamiento, tan solo hay que analizar la forma en la que el gráfico cambia para predecir lo que vendrá.

- **Más impacto.** Para un humano, es mucho más sencillo recordar un gráfico con muchos patrones que una serie de datos en crudo. Los gráficos son una forma simple de expresión que no requieren mucho esfuerzo mental para que permanezcan en la memoria.

  Es por esta razón por la que se genera más impacto en el usuario y, por consiguiente, se puede concluir que la visualización de datos es una forma muy eficaz para transmitir información.

[2] Los datos crudos es un término para los datos que no han sido sujetos al procesamiento u otra manipulación.

- **Ayuda en la toma de decisiones.** Gracias a todos los beneficios anteriores, si se sitúa la visualización de datos en un contexto empresarial, se puede conseguir ayudar al personal directivo a tomar las decisiones de una forma más sencilla y respaldada con datos.

  Es posible utilizar la visualización para analizar una serie de métricas o KPIs[3] permitiendo comprender de una forma rápida y sencilla lo que se debe realizar para mejorar o reforzar el rendimiento.

Una vez mencionados los beneficios que aporta la visualización de datos, es el momento para explicar cuál es el procedimiento para crear una visualización correctamente.

2.4.2. Fases de la visualización de datos

Para lograr una visualización de datos eficaz, partiendo del objetivo de la representación hasta llegar al elemento visual que la muestre, hay que seguir una serie de pasos [26]:

- **Definición de los objetivos.** Esta primera fase es quizás una de las más importantes, ya que hay que definir el objetivo de la visualización. Para ello es preciso identificar qué métrica se desea representar, qué tipo de datos se van a utilizar, qué tipo de análisis y qué elementos visuales se van a utilizar. Teniendo claro todos estos aspectos, ya se puede comenzar a preparar la visualización.
- **Recopilación de los datos.** Es muy importante seleccionar correctamente la fuente sobre la que se realizará la visualización. Hay que determinar si habría que enriquecer el dataset mediante la unión con otros datasets, ver si existen datos históricos relacionados para poder mejorar los datos

[3] Un KPI, Indicador Clave de Desempeño o Medidor de Desempeño, hace referencia a una serie de métricas que se utilizan para sintetizar la información sobre la eficacia y productividad de las acciones que se lleven a cabo en un negocio con el fin de poder tomar decisiones y determinar aquellas que han sido más efectivas a la hora de cumplir con los objetivos marcados en un proceso o proyecto concreto.

que se utilizarán para la visualización actual, etc. Es decir, en esta fase hay que preparar un conjunto de datos rico y útil para conseguir el objetivo planteado en la fase anterior.

- **Limpieza y preprocesamiento de los datos.** Una vez definido el dataset, hay que realizar un procesamiento y limpieza del mismo para eliminar datos redundantes, eliminar aquellos que no sean útiles para la visualización, realizar operaciones matemáticas o de agrupación para que estén preparados para los posteriores análisis. Filtrar y convertir los datos para que se ajusten a los criterios del objetivo de visualización, etc.
- **Selección de los elementos visuales.** Llegado a este punto, los datos que se tienen están perfectamente preparados para poder conseguir el objetivo propuesto en la primera fase. Ahora hay que determinar qué elemento visual es el más adecuado para representar la información. Existen diferentes tipos de elementos visuales como gráficos de evolución, de barras, circular, mapas, etc.

  También existen diferentes tipos de visualización: una *visualización estática* ofrece únicamente una sola visión de una historia de datos específica, mientras que una *visualización interactiva* permite a los usuarios interactuar con los gráficos y diagramas.

  Es muy importante determinar correctamente que elemento visual es el más adecuado para el objetivo planteado, ya que la selección de uno incorrecto puede llevar una incorrecta interpretación de los datos.
- **Creación de la representación.** Finalmente, el proceso de visualización concluye con la representación, propiamente dicha, del elemento visual seleccionado. En esta fase hay que prestar atención en los detalles: en el color, tipo de letra y tamaño del gráfico, en la elección de las combinaciones de colores adecuadas a lo que se desea representar, a las leyendas y a las etiquetas de los datos.

Es muy importante tener en cuenta cada uno de estos detalles, ya que una elección errónea de uno de ellos puede llevar a que un tipo de gráfico adecuado para los datos y objetivo no sea capaz de mostrar eficazmente los datos y por tanto, no se interpreten correctamente los datos.

Ahora que ya se ha definido el procedimiento para crear una visualización de datos, se va a explicar que es un cuadro de mandos o dashboard, que información contiene y cuál es su fin, ya que el objetivo de este proyecto es el desarrollo de un dashboard sobre los datos de malware.

2.5. ¿Qué es un cuadro de mandos o Dashboard?

Un cuadro de mandos o dashboard es una presentación visual de todos aquellos KPIs de una organización. Su uso es muy común en los negocios, finanzas, salud, educación y otros campos donde la toma de decisiones basada en datos es importante [24], [27].

Están compuestos por una variedad de elementos gráficos, como tablas, medidores, mapas, gráficos, etc. La información presentada en un cuadro de mandos está diseñada para proporcionar una visión rápida de los datos para que los usuarios puedan identificar tendencias y patrones.

Últimamente se han vuelto muy populares debido a la creciente cantidad de datos disponibles y la necesidad de la toma de decisiones informadas. Esta presentación visual permite a los usuarios ver la información de una forma muy rápida y clara, permitiendo extraer los patrones y tendencias de los datos y plantear una estrategia o decisión en base a ellos.

Son muchas las ventajas de los dashboards, entre ellas está en que permiten utilizar los datos en bruto de diversas fuentes para construir un tablero donde los usuarios podrán consultar y comprender los datos de forma inmediata.

Además, como permiten utilizar colores, simbologías, fuentes personalizadas se pueden resaltar aquellas zonas, gráficos o puntos más relevantes. Esto permite a los usuarios escanear rápidamente el panel y obtener la información que necesita sin tener que consultarla sobre los datos en crudo.

Otra de las ventajas que ofrecen es que en la mayoría de las ocasiones pueden exportarse las visualizaciones del panel y generar automáticamente informes con dicha información. Por lo que esto permite ahorrar tiempo y recursos, ya que la generación manual de los informes de análisis de KPIs suele ser bastante costoso.

Ahora, que se han presentado todos los conceptos y temas sobre los que trata este proyecto, se va a explicar el dataset seleccionado, el procesamiento del mismo, la selección de métricas y se culminará con el diseño y la implementación del dashboard que contendrá las visualizaciones de las métricas.

3. DATASET DE MALWARE

Para la elaboración de este trabajo era necesario la obtención de un dataset sobre el que trabajar para poder definir una serie de métricas que se pintarán en el panel de mandos.

Lo primero era definir un dataset, generarlo manualmente a través de diferentes fuentes o buscar alguno por la web que se adaptara a los intereses del proyecto. Lo que se necesitaba era que tuviera un volumen grande de datos y que tuviera diferentes propiedades de valor sobre las que se pudieran definir una serie de métricas.

Tras buscar en diferentes fuentes, se decantó por utilizar el dataset "*malware bazaar*"*[28]*. Este dataset ofrece una gran cantidad de datos desde marzo de 2020, además su uso es totalmente gratuito, tanto para uso comercial como para uso personal.

Desde su web se puede consultar los datos más recientes y ofrece la posibilidad de descargar los datos mediante diferentes formatos, así como ofrece un API a la que se puede hacer peticiones para descargar los datos. Ofrecen tanto los datos característicos de malware como los ficheros ejecutables que contiene el software malicioso, esto podría ser de utilidad si se necesitara entrenar una red para detectar software malicioso, como este trabajo trata de la visualización de datos de malware en un panel de control, se decidió que solamente se utilizarán los datos con las características de cada malware.

Una vez seleccionado el dataset, se procedió a la obtención de los datos mediante la descarga del fichero *csv* que ofrecían en su web. Aquí se pudo comprobar que se encontraban la totalidad de los datos que había en la base de datos, pero no tenían todas las propiedades, estaba limitado para evitar que fuera excesivamente grande el fichero. Para obtener los datos con todas las propiedades habría que hacerlo mediante el uso del API que ofrecen.

3.1. DESCARGA DE LOS DATOS DEL API

Desde la plataforma de "*Malware bazaar*" se ofrecen varios tipos de API a los que se pueden realizar peticiones para obtener información más detallada sobre los datos de Malware. De entre todas ellas, la que se ajustaba a las necesidades del proyecto era el endpoint que devuelve toda la información disponible de un archivo malware a partir de su *hash md5*[4][29].

La información para la ejecución del endpoint es la siguiente:

```
POST https://mb-api.abuse.ch/api/v1

Body: {
  query: "get_info",
  hash: "ca3a02fe9d62fe2e4c1fe87993254163"
}

Headers: {
  "API-KEY": <your api key>
}
```

Código 1. Ejemplo de los parámetros requeridos para la ejecución del API de Malware Bazaar.

El formato de salida de los datos se detalla en el Anexo A, en dicha sección se muestra el listado con cada uno de los campos que devuelve junto a una descripción del significado del cada valor.

Para poder obtener la información del API se tuvo que desarrollar un pequeño software que se conectara a ese endpoint de forma automática y lo llamara tantas veces como hash md5 hay disponibles dentro de dicho dataset.

Para obtener el listado de todos los hashes md5 disponibles se descargó un fichero de texto que se ofrece en la plataforma de "*Malware bazaar*". Una vez que se disponía de esta información, se desarrolló un código en *Typescript* que leía los hashes y para cada uno de ellos se llamaba al API para obtener la información asociada y después se almacenaba en una base de datos en MongoDB [30].

[4] *Hash md5* es un protocolo criptográfico utilizado para autenticar mensajes, así como para verificar contenidos y firmas digitales.

Se decidió almacenar los datos inicialmente en este tipo de base de datos, porque es una base de datos NoSQL, orientada a documentos y que permite almacenar los datos en JSON. Por tanto, utilizando esta base de datos se ganaba agilidad para este proceso, simplemente se almacenan los datos en crudo, tal y como los devuelve el API. Posteriormente, cuando se pase a las fases de diseño del proyecto y haya que transformar y procesar los datos, ya se decidirá qué base de datos será la que utilizará el panel final, si seguirá siendo Mongo o si será otra diferente.

Los datos se almacenarán dentro de mongo con la misma estructura y sin realizarles ningún tipo de modificación, por tanto los datos se almacenarán con la estructura que se puede ver en el anexo A.

A continuación, se muestra el pseudocódigo que se diseñó para obtener los datos del API:

```
// Se obtiene los hashes md5, en un array de strings
var md5_ids = readFile("full_md5.txt");

var counter = 1;
var total = md5_ids.length;

for md5 of md5_ids
do
  // La function post es la encargada de hacer la llamada al API según la especificación
  var api_result = post(md5);

  if api_result.status_code == 200
  then
    // Si entra a este bucle, la llamada al API ha sido correcta (HTTP STATUS = 200)
    insertDocumentInMongo(api_result.data);
  fi
done
```

Código 2. Pseudocódigo diseñado para descargar los datos del API y almacenarlos dentro de mongo.

Este código sufrió algunas modificaciones hasta que se llegó a la versión final. Algunas de estas modificaciones fue el ir almacenando los resultados del API en pequeños *baches* o *lotes*, de forma que no se salvarán los datos en mongo de uno en uno, sino que se almacenarán de 50 en 50, esto es más óptimo que el diseñó del boceto inicial.

Hay que destacar que este trabajo no descarga los datos en tiempo real para tener la última versión de los datos, es decir, se tiene una versión offline, por tanto si se volviera a descargar los datos ahora, los datos serían diferentes, ya que el dataset que se utilizó para este proyecto tiene datos desde marzo de 2020 hasta abril de 2023.

Tras descargar y almacenar los datos en mongo, se pudieron visualizar las dimensiones del dataset, así como algunas estadísticas de los datos, como el peso, número de propiedades diferentes, número de columnas, etc. Toda la información referente a como descargar los datos, así como todas estas estadísticas se puede visualizar en el Anexo B, aunque el código desarrollado también se encuentra dentro del repositorio de GitHub asociado a este trabajo final de master [31].

Una vez que se tenían los datos descargados y almacenados en una colección de mongo, se pudo pasar a la siguiente fase de los datos, la fase del preprocesado y limpieza.

3.2. Definición de Métricas

Una vez que se ha seleccionado el conjunto de datos y que se han podido descargar a través del API de "*Malware bazaar*"*[29]*, se podía proceder a la realización de un análisis para poder seleccionar una serie de métricas que se mostrarán en el panel de control a desarrollar.

Tras analizar detenidamente estos datos (ver anexo A y B), se pudo ver que tenían una serie de propiedades clave sobre las que realizar las métricas. Algunas de estas propiedades son: Tamaño del fichero malicioso (en bytes), país donde se detectó, extensión o tipo del fichero, tipo de familia de malware, fecha de detección, método de distribución, etc.

Tras encontrar estas propiedades se comenzó con el diseño de las métricas propiamente dichas. Dado que los campos que se han mencionado

son de diversa índole, se decidió crear una serie de grupos, de forma que posteriormente también fuera más sencillo el diseño e implementación del dashboard.

Los grupos que se crearon fueron los siguientes: “métricas asociadas a la evolución del malware en el tiempo”, “métricas asociadas a las familias de malware”, “métricas asociadas a los métodos de distribución del malware” y “métricas asociadas a las características de los ficheros de malware”.

En las próximas secciones se explicará cada uno de estos grupos y se definirán cada una de las métricas asociadas a ellos.

3.2.1. Métricas asociadas a la evolución del malware en el tiempo

En este grupo se tratará de visualizar diferentes métricas, pero todas con el factor común del tiempo. Se posee la fecha de detección de cada fichero malware que se encuentra registrado en la base de datos, por tanto es sencillo saber: el día, semana, mes o año que un programa malicioso fue detectado.

Esto permite una gran cantidad de opciones para visualizar los datos, ya que el tiempo se puede combinar con el país o con el número de registros asociados a una fecha, entre otras posibles opciones.

Durante las siguientes líneas se van a presentar todas las métricas que se agrupan en este conjunto, mostrando para cada una de ellas que es lo que se desea mostrar y cuál será el objetivo que cumplir con su visualización.

3.2.1.1. Evolución de la detección del Malware en el mundo

Esta métrica trata de analizar el aumento semanal del número de casos detectados de malware en cada país del mundo. Es evidente que el malware no se distribuye de forma equitativa a lo largo del mundo, hay unos países que sufren más ataques a dispositivos que otros. Esto se debe a diversos

motivos, pero los fundamentales sería el nivel de digitalización y el número de dispositivos que posee cada país.

Es por tanto, una métrica muy importante para tener en cuenta, ya que se podrá ver que países han registrado un mayor número de malware y por ende, se podrá saber en qué lugar del mundo habría menor probabilidad de infección de malware.

Se comenta "menor probabilidad de infección", pero esta frase es muy relativa, porque desde cualquier lugar del mundo se tiene la misma probabilidad de infectar un dispositivo de malware, pero lo que si influirá para que unos países tengan mayores tasas de infección que otros son los hábitos de navegación por internet, la concienciación de la población sobre el uso seguro de internet, el tipo de dispositivos que utilicen, entre otros factores ajenos al estudio y elaboración de este trabajo. Pero cuando aquí se utilice el concepto menor probabilidad, se refiere a la aplicación de la siguiente operación matemática:

$$P = \frac{n_{country}}{N}$$

Donde P es la probabilidad de infección, $n_{country}$ es el número de casos registrados en un país determinado, N es el número total de casos registrados.

Esta métrica seguramente irá representada mediante un mapa y se coloreará cada país con una progresión de colores que destaque aquellos que más malware ha detectado, pero se decidirá en la fase de diseño.

3.2.1.2. Comparativa del Malware detectado en un año

En esta ocasión, el objetivo es realizar la comparación del número de casos de malware detectados el año actual respecto a los casos detectados el año anterior. Es una métrica muy sencilla de mostrar y de analizar, sin embargo

aporta un gran valor, ya que permite ver a simple vista si este año ha sido un peor año, en cuanto al número de dispositivos infectados.

En esta ocasión, cuando se menciona la palabra detectar, se habla de detectar softwares maliciosos diferentes, pero cada uno de ellos puede infectar a millones de equipos en todo el mundo. Pero, suele ser proporcional, a mayor número de softwares diferentes, mayor número de infecciones.

Esta métrica probablemente irá representada con un gráfico de barras, con una barra para cada año, pero no será hasta la fase de diseño cuando ser determine el tipo de gráfico a utilizar.

3.2.1.3. Comparativa del Malware detectado en un mes

Esta métrica es idéntica a la anterior, solamente varía el periodo de comparación, en este caso se compararán los casos de malware detectados el mes en curso respecto a los detectados el mes anterior.

El tipo de representación de esta métrica será igual que la anterior, con un gráfico de barras.

3.2.1.4. Evolución mensual de Malware

El objetivo de esta métrica es analizar la progresión del número de casos detectados de Malware por meses. Además, como se poseen datos de más de un año, se intentará utilizar un tipo de representación que permita mostrar más de una serie, es decir, una serie por cada año de los que se dispongan datos.

De esta forma, el usuario podrá ver la evolución de casos de malware para cada año individualmente, pero también podrá visualizar todas las series juntas y ver como difiere el número de casos de un mes en los años de los que se disponen datos.

No será hasta la fase de diseño cuando se decida el tipo concreto de representación, pero es probable que se utilice un gráfico de líneas con una línea por cada año de los que se disponga información.

3.2.1.5. Evolución semanal de Malware

Esta métrica es similar a la anterior, salvo que aquí no se desea mostrar la información con varias series, sino que se desea mostrar la evolución en una única sucesión.

El objetivo es similar al de la métrica anterior, es poder mostrarle al usuario como ha sido la progresión de la detección de malware desde que se tienen registros hasta el día actual.

Esta métrica y la anterior son muy similares, pero no son iguales, ya que esta métrica al agrupar el número de casos por semana los resultados serán mucho más precisos, ya que se podrán ver las alteraciones con mucho más detalle que en el caso de las evoluciones mensuales.

Es probable que se utilice una gráfica de líneas, al igual que en la métrica anterior, pero en este caso, habrá que utilizar alguna técnica para poder ampliar la información del gráfico, ya que al disponer de tantos datos, si no se ofrece alguna forma para hacer zoom a los datos, es probable que pudiera quedarse oculta a la vista alguna alteración de los datos. Pero, como se ha mencionado en las métricas anteriores, todo esto se determinará en la fase de diseño.

3.2.2. Métricas asociadas a las familias de malware

El segundo grupo de métricas estará destinado a analizar los datos en torno al tipo o familia de malware. Llegados a este punto hay que aclarar una cuestión, en el capítulo de *estado del arte* se enumeran una serie de familias o tipos de malware, pero hay que saber que cada uno de estos tipos puede tener millones de software maliciosos, cada uno con un nombre propio. Por

tanto, cuando se vean los tipos de malware que aparecen en las visualizaciones no hay que sorprenderse si son nombres diferentes a los descritos aquí, lo que sí que es seguro que pertenecen a una de las categorías estudiadas y descritas en este documento.

A continuación se van a exponer cada una de las métricas que se encontrarán agrupadas bajo este conjunto, detallando para cada una de ellas en que consiste y como podrá ser su representación.

3.2.2.1. Tipos de Malware más comunes

Esta primera métrica tratará de determinar el número de malware que existe para cada una de las familias existentes dentro del conjunto de datos, es decir, como cada registro del dataset se corresponde con un software malicioso diferente, se va a mostrar el número de variantes de malware existentes para cada familia.

Dado que existen una gran cantidad de familias de malware en el dataset, unas 400, habrá que proporcionarle al usuario algún tipo de herramienta para que pueda hacer zoom en los datos, ya que si no será imposible que se pueda ver con claridad cada uno de los registros.

El tipo de gráfico que representará los datos será uno de barras, ya que es el tipo de gráfico idóneo para mostrar la cantidad de datos que posee cada categoría. No obstante, el gráfico a utilizar se decidirá en la fase de diseño, ahora son todo sugerencias de representación, todavía no está decidido como representar la información en este punto del documento.

3.2.2.2. Variantes de una familia de Malware vs. Infecciones detectadas

Esta métrica tratará de comparar, para las 10 familias que más variantes tienen, el número de infecciones que provocó esa familia. Pueden parecer dos términos equivalentes, pero no lo son. Cuando se habla de número de

variantes es del número concreto de programas maliciosos que posee una determinada familia. Mientras que el número de infecciones que provocó una familia es el número de dispositivos que registro un ataque con algún programa malicioso de dicha familia. El objetivo es analizar si el número de variantes influye, o no, en el número de infecciones.

El gráfico de barras es probablemente el gráfico más adecuado para representar este tipo de información, con una barra para el número de variantes y una barra para el número de infecciones. Habrá un par de barras por cada familia de malware analizada, como se ha determinado que sea el TOP 10, el gráfico estará compuesto por 20 barras diferentes.

3.2.2.3. Tipos de Malware más comunes por país

Finalmente, la última métrica que compone a este grupo es el número de variantes de malware agrupadas por cada país, es decir, se quiere mostrar en un mapa donde se han registrado más variantes de malware de una familia determinada.

Para esta métrica habrá que tener en cuenta dos aspectos muy importantes. El primero es que habrá que determinar algún tipo de herramienta para que el usuario sea capaz de seleccionar que familia de malware desea visualizar la distribución, ya que son cerca de 400 familias de malware diferentes.

El otro aspecto a tener en cuenta es la selección de colores a utilizar en el mapa para que se pueda ver con cierta claridad que países han registrado más variantes y cuales han registrado menor cantidad. El mapa de colores es una decisión de diseño muy importante para esta métrica, quizás la más importante, ya que una mala decisión de colores puede llevar a sacar conclusiones erróneas de los datos.

3.2.3. Métricas asociadas a los métodos de distribución del malware

El tercer grupo de métricas está destinado a agrupar las progresiones relacionadas con los métodos de distribución de malware. Existen multitud de canales mediante los cuales un dispositivo puede infectarse, por esta razón es por la que se determinó que era interesante ofrecerles a los usuarios las herramientas para poder analizar los datos en base a esta casuística.

Ahora se van a exponer los detalles de cada una de las métricas que componen este grupo.

3.2.3.1. Métodos de distribución más comunes por tipo de Malware

Esta métrica tratará de analizar cómo influye el método de distribución en las cinco familias de malware con más variantes. Es decir, para las cinco familias de malware que más tipos diferentes de software malicioso poseen, se tratará de analizar qué métodos de distribución son más comúnmente utilizados.

De esta forma, se podrá determinar qué tipo de método de distribución es más usado por estas familias, de forma que, los usuarios que interpretasen esta visualización podrían tomar medidas para intentar mitigar la infección de dispositivos para cada una de estas familias.

La gráfica más óptima para reflejar esta métrica sería una de barras, agrupando por cada una de las cinco familias de malware, de forma que haya una barra por cada método de distribución, si existen, por ejemplo, cuatro métodos de distribución diferentes, habría cuatro barras por cada una de las familias de malware.

3.2.3.2. Métodos de distribución más comunes por tipo de fichero

Esta segunda métrica es muy similar a la anterior, salvo que en lugar de mostrar las estadísticas sobre las 5 familias de malware que más variantes tienen, ahora se agrupan los métodos de distribución con los tipos de fichero.

Por consiguiente, se mostrarán los datos asociados a los cinco tipos de fichero que más han sido utilizados para infectar dispositivos.

El tipo de gráfico a utilizar será del mismo tipo que en la métrica anterior, una gráfica de barras agrupando los datos por métodos de distribución de cada tipo de fichero.

3.2.3.3. Métodos de distribución de Malware más comunes por país

La última de las métricas de este grupo consiste en mostrar la distribución de software malicioso por cada método de distribución a lo largo del mundo. Habrá que utilizar un código de colores óptimo para representar el acumulado de datos por cada país, de forma que los colores más claros indiquen menos malware y los más oscuros indiquen mayor concentración de ficheros malware distribuidos por el canal seleccionado.

Habrá que determinar cómo seleccionar el tipo de medio por el que se distribuye el malware, ya que el mapa se deberá actualizar con los datos asociados al método seleccionado por el usuario. Este método de seleccionado del canal de distribución tiene que ser dinámico para los usuarios que utilicen el panel, de forma que sean ellos mismos los que indiquen que tipo de información desean mostrar en el mapa.

3.2.4. Métricas asociadas a las características de los ficheros de malware

Finalmente, el cuarto grupo de métricas está compuesto por varias distribuciones que mostrarán al usuario información de los casos de malware registrados según las características de los ficheros.

En los datos se tienen varias características de los ficheros disponibles para utilizarse, pero las más importantes son el tipo y el tamaño de los ficheros.

Esta información puede ser muy importante, ya que seguro que existen tipos de ficheros que es más probable que contengan software malicioso. En cuanto al tamaño, o peso, de los ficheros, se creyó interesante ofrecerles a los usuarios las herramientas para ver cómo influye el tamaño de los ficheros en el malware.

Por todo ello, a continuación se define cada una de las métricas que contiene este grupo, mencionando los objetivos de visualización, así como se indicará que tipo de gráfico puede ser el más adecuado para representar la información.

3.2.4.1. Distribución del tipo de ficheros maliciosos

Esta métrica tratará de mostrar la cantidad de ficheros maliciosos que posee cada tipo de fichero. Dado que existen una gran cantidad de tipos de ficheros diferentes, habrá que determinar algún tipo de límite para no mostrar todos, ya que sino el grafico podría ser ilegible.

La idea de esta métrica es presentarle a los usuarios los tipos de ficheros que son más vulnerables de contener código malicioso en su interior, de forma que se pueda concienciar a la población de la peligrosidad de algunos tipos de ficheros.

En la fase de diseño habrá que establecer el límite comentado anteriormente, por ejemplo un límite se podría poner cuando el número de ficheros maliciosos asociados a un tipo de fichero sea inferior a 10.000-9.000 y todos los registros que se encuentren por debajo de estar horquilla se etiqueten bajo la denominación de “otros tipos de ficheros”.

El gráfico que se podría utilizar para presentar esta métrica podría ser un gráfico de sector o de anillo, ya que además de mostrar el número exacto de ficheros maliciosos que tiene ese tipo de archivo, se podrá ver de forma clara y directa el porcentaje estimado que ocupa cada tipo de fichero en la totalidad de los datos.

3.2.4.2. Tamaño medio de los ficheros maliciosos según el tipo de fichero

Esta métrica permitirá comprobar el tamaño o peso medio de los ficheros maliciosos para cada tipo de fichero existente en los datos. Para esta métrica sí que habrá que mostrar la totalidad de los datos, no se puede mostrar un subconjunto de ellos, ya que se desea mostrarles a los usuarios el tamaño medio para todos tipos de ficheros.

Además, se añadirá a la gráfica la media absoluta del tamaño de los ficheros, de forma que los usuarios podrán comparar para cada una de las tipologías de los ficheros si están por encima o por debajo de la media absoluta.

Como habrá que mostrar los datos para la totalidad de los tipos de fichero, habrá que determinar algún tipo de fórmula para poder hacer zoom en el gráfico, para que los usuarios puedan ver sin ningún tipo de problemas los datos que posean unos registros medios de tamaño realmente pequeños.

Esta métrica podría representarse mediante un gráfico de barras, para que se muestre el tamaño medio de cada tipo de fichero mediante una barra para cada uno. A este gráfico se le añadiría una línea recta horizontal

representando la media absoluta de tamaño, así todas las barras que estén por encima tendrán registros por encima de la media y si están por debajo sería justamente lo contrario.

3.2.4.3. Tamaño medio de los ficheros maliciosos según el tipo de Malware

Esta métrica es idéntica a la descrita en la sección anterior, solamente cambia que en lugar de mostrarse los valores de tamaño medio para cada tipo de fichero, ahora se mostrarán para cada tipo de familia de malware.

La gráfica resultante será mucho más grande, ya que hay muchas más familias de malware que tipos de ficheros en los datos, pero como se comentó en la sección anterior, habrá que buscar la forma de permitir hacer zoom en la gráfica para poder visualizar los datos correctamente.

Consecuentemente, al ser la gráfica idéntica a la anterior, solamente cambia la propiedad sobre la que se muestra el tamaño medio, habrá que utilizar el mismo tipo de gráfico, con los mismos elementos: barras para el tamaño medio asociado a cada tipo de malware y una línea horizontal para mostrar el tamaño medio absoluto.

3.2.4.4. Tipos de Malware más comunes para cada tipo de fichero

Finalmente, la métrica que cierra este grupo consiste en mostrar para cada tipo de fichero una lista con las familias de malware más comunes. Dada la gran cantidad de variedades de tipos de ficheros que se tienen en los datos, se decidió realizar esta métrica para los cinco tipos de fichero que más software malicioso tienen.

Paralelamente, dado que cada tipo de fichero tiene asociados una gran cantidad de tipos de malware diferentes, se decidió mostrar para cada uno de los ficheros el top 10 de las familias de malware más comunes de cada uno.

Dado que la información a representar para esta métrica es toda textual, habrá que pensar muy bien qué tipo de gráfico utilizar para mostrar la información. Quizás una buena opción sería utilizar algún gráfico que represente la información en forma de árbol, mostrando en la raíz el tipo de fichero y una hoja para cada uno de sus tipos de malware más comunes.

3.3. Preprocesado y limpieza de los datos

Tras la definición de las diferentes métricas, antes de proceder al diseño e implementación del dashboard, hay que realizar todas las transformaciones y limpieza de los datos para poder construir dichas métricas.

Las tareas principales de limpieza y preprocesado están relacionadas con el renombrado, la transformación o la eliminación de propiedades de los datos. A lo largo de las siguientes secciones se dará detalle y se mostrará el código utilizado para la ejecución de cada tarea.

3.3.1. Transformación de columnas

Una de las principales propiedades que posee el dataset es la fecha en la que se detectó por primera vez un fichero de malware. Por desgracia, las fechas tal y como se descargaron del API de *Malware bazaar [29]* son cadenas de caracteres con el formato: "2023-04-30 19:24:54", este dato no puede ser utilizado si no se realiza una conversión.

Por tanto, antes de poder generar columnas relacionadas con esta fecha, había que hacerle una transformación para que pase de ser una cadena de caracteres a un dato de tipo fecha. Para ello se utilizó el siguiente código:

```
db.malware_data.aggregate([{
    $addFields: {
        datetime: {
            $toDate: "$first_seen"
        }
    }
}]);
```

Código 3. Fragmento de código que transforma el tipo de dato de la fecha para que sea "datetime".

Con la función "*toDate*" de mongo se consigue convertir una columna de texto, en este caso la columna "*first_seen*", a otra de tipo fecha. Tras ejecutar este código, se tiene un dataset con una nueva columna denominada "*datetime*" con el cambio de formato de datos comentado.

3.3.2. CREACIÓN DE COLUMNAS A PARTIR DE LA FECHA

Ahora que se posee la fecha con el formato correcto, se pueden crear una serie de nuevas columnas con datos generados a partir de dicha fecha. Estos campos puede ser el número de semana, de mes o de año.

Es necesario calcular estos campos ahora en la fase de preprocesado, ya que de esta forma será más sencillo diseñar las sentencias que generen los datos para construir las métricas y al tener el valor calculado no será necesario tener que calcular, en tiempo de ejecución, cual es el número de semana para todas las fechas que devuelve una consulta, por ejemplo.

Se consideró necesario generar tres nuevas columnas a partir de la fecha, para almacenar el número de semana, el mes y el año. El código que se utilizó fue el siguiente:

```
db.malware_data.aggregate([
    {
        $addFields: {
            num_week: { $week: "$datetime" }
        }
    },
    {
        $addFields: {
            year: { $year: "$datetime" }
        }
    },
    {
        $addFields: {
            month: { $month: "$datetime" }
        }
    }
]);
```

Código 4. Código que crea nuevas columnas con el número de semana, el mes y el año a partir de la comuna que contiene la fecha.

Tal y como se puede ver, todas las operaciones utilizan la columna "*datetime*" que se generó en el apartado anterior. Se utiliza una sentencia de

agregación de MongoDB [32] para realizar las operaciones una detrás de la otra, de forma que se puede ir utilizando los resultados obtenidos en los pasos anteriores.

Con las operaciones "*$week*", "*$year*" y "*$month*" se obtiene el valor del número de semana, del año y del mes, respectivamente, que se almacenarán en nuevas columnas del dataset resultante.

3.3.3. RENOMBRADO DE COLUMNAS

Otra de las tareas de preprocesado de los datos es el renombrado de algunas columnas, para asignarles un nombre que sea más correcto o que le aporte más valor a la información que almacena.

Concretamente, se realizó el renombrado de tres columnas, dos de ellas para asignarle un nombre que aporte más valor y que permita conocer con claridad que tipo de información esta almacenando. El renombrado restante se realizó para que el nombre fuera un poco más correcto.

```
db.malware_data.aggregate([
    {
        $addFields: { num_detections: "$intelligence.uploads" }
    },
    {
        $addFields: { malware_type: "$signature" }
    },
    {
        $addFields: { country: "$origin_country" }
    }
]);
```

Código 5. Fragmento de código que renombra algunas de las columnas que posee el dataset.

La primera operación almacenará el valor de la columna "*uploads*" (número de veces que se subió el mismo fichero de malware, es decir, el número de veces que fue detectado ese malware) y ahora tendrá el nombre de "*num_detections*".

En segundo lugar se renombra la columna "*signature*" y ahora tendrá el nombre de "*malware_type*", que es un nombre mucho más representativo para almacenar el tipo de malware que corresponde a cada registro.

Finalmente, se renombra la columna "*origin_country*" para que simplemente se denomine "*country*".

Como se puede observar, no se está haciendo un renombrado de columna, lo que técnicamente se está haciendo es la creación de nuevas columnas con los nombres que se desean y en un paso posterior se eliminarán las columnas con el nombre original. Esta tarea del borrado de columnas se detallará en la siguiente sección.

3.3.4. BORRADO DE COLUMNAS

Finalmente, se realizó una tarea para eliminar todas las columnas innecesarias para la consecución de las métricas anteriormente descritas. Esto tiene como resultado un dataset de menores dimensiones, y por ende, de menor peso.

```
db.malware_data.aggregate([
    {
        $project: {
            "_id": 0,
            "sha1_hash": 1,
            "datetime": 1,
            "num_week": 1,
            "year": 1,
            "month": 1,
            "file_size": 1,
            "file_type": 1,
            "delivery_method": 1,
            "malware_type": 1,
            "country": 1,
            "num_detections": 1
        }
    }
]);
```

Código 6. Código que indica que columnas del dataset se conservarán, todas las demás columnas serán eliminadas.

La sentencia anterior lo que hace es realizar una proyección de los datos y devolverá, para cada registro de la base de datos, solamente las columnas cuyos nombres aparecen dentro del objeto "*$project*", siempre y cuando este seguido por un 1, ya que si están seguidos por un 0 significa que esa columna se descarta en la salida.

3.3.5. MIGRACIÓN A POSTGRESQL

Como se expuso anteriormente, cuando se descargaron los datos del API de *Malware Bazaar*, se comentó que se utilizaba la base de datos MongoBD porque en ese momento del proyecto esa base de datos daba flexibilidad, ya que se almacenaban los datos tal y como llegaban del API, ya que existían objetos y listas dentro de los datos, y Mongo ofrece la posibilidad de almacenar los datos de este tipo sin ningún tipo de problemas.

Pero, llegados a este punto había que decidir si se continuaba con MongoDB o si se cambiaba a otra base de datos. Ahora el dataset se había procesado y no poseía ninguna lista ni objeto entre sus propiedades. Por tanto, se podría cambiar de base de datos si así se consideraba.

Se estuvo debatiendo si hacer un cambio de base de datos o si se dejaban los datos en MongoBD. Finalmente se concluyó que se iba a realizar una migración de datos a PostgreSQL. Las decisiones que llevaron a tomar esa decisión fueron las siguientes:

- Los datos no contenían ninguna estructura compleja dentro de sus propiedades, solo se tenían cadenas de texto, números o fechas.
- Se podía considerar que los datos tenían una estructura fija, ya que al no poseer objetos ni listas entre sus propiedades, todos los datos poseían el mismo tipo de información en cada una de las columnas.
- PostgreSQL es una base de datos relacional perfecta para datos estructurados. Mientras que Mongo es una base de datos no relacional orientada a documentos. Por lo que encajaría mejor PostgreSQL con el dataset que se posee en este momento.
- Se tenía mucha más experiencia con PostgreSql que con MongoDB. Lo que es una ventaja a la hora de diseñar e implementar las consultas para obtener los datos de las diferentes métricas.
- PostgreSQL encaja perfectamente dentro de la arquitectura hexagonal.

Por todo ello, se inició el proceso de migración de datos de MongoDB a PostgreSQL. Lo primero que se realizó fue la creación del script que crearía la estructura de la base de datos:

```
CREATE TABLE malware.malware_data_filtered
(
  sha1_hash text primary key,
  country text,
  datetime text,
  delivery_method text,
  file_size bigint,
  file_type text,
  malware_type text,
  month integer,
  num_detections integer,
  num_week integer,
  year integer
);
```

Código 7. Script SQL con el código para crear la tabla que albergará los datos dentro de PostgreSQL.

Una vez creada la estructura, el siguiente paso era la migración de los datos, esto se consiguió mediante el software DataGrip de Jet Brains. Este programa ofrece la opción de exportar los datos de una colección en diferentes formatos, entre ellos se encuentra en formato de "SQL *Inserts*" [33]. Por tanto, se utilizó esta opción y se tuvo como resultado un fichero *sql* con los datos listos para ser insertados en PostgreSql.

```
INSERT INTO malware.malware_data_filtered (
  country, datetime, delivery_method, file_size, file_type,
  imphash, malware_type, month, num_detections, num_week, sha1_hash, year
) VALUES (
  null, '2022-01-30T18:31:04.000Z', 'email_attachment', 87028, 'xlsx', null,
  'SilentBuilder', 1, 1, 5, '2540e5c8150ad77189275458e65bddd10e49b4dc', 2022
);
```

Código 8. Extracto del código SQL obtenido mediante DataGrip, donde se pueden ver las sentencias de inserción de los datos.

Ahora se procedió a insertar los datos en la estructura creada en PostgreSql, para ello se decidió hacerlo mediante línea de comandos, ya que el proceso es más rápido y eficiente que si se ejecutara desde DataGrip. El comando es el siguiente:

```
$> psql -h 127.0.0.1 -p 5432 -U postgres -d tfm -f malware_data_filtered.sql
```

Código 9. Comando necesario para importar los datos dentro de PostgreSQL, se tiene que ejecutar desde línea de comandos.

Una vez que finalizó el proceso, se comprobó que efectivamente se habían migrado los datos correctamente y que se podría continuar con los procesos finales de limpieza.

3.3.6. Limpieza de los datos

La última acción por realizar sobre los datos era la de decidir qué hacer con las columnas que tuvieran valores nulos. Para ello, en primer lugar, se realizó un análisis para ver la cantidad de datos nulos que tenía cada columna.

Se realizó una consulta sobre la tabla *pg_stats* [4] de Postgres para observar las estadísticas que tienen los datos almacenados. La consulta y los resultados obtenidos se pueden ver a continuación:

```
SELECT * FROM pg_stats WHERE tablename = 'malware_data_filtered'

+----------+--------------------+---------------+----------+-----------+
|schemaname|tablename           |attname        |null_frac |n_distinct |
+----------+--------------------+---------------+----------+-----------+
|malware   |malware_data_filtered|country        |0.97326666|44         |
|malware   |malware_data_filtered|datetime       |0         |-0.93178517|
|malware   |malware_data_filtered|delivery_method|0.37346667|6          |
|malware   |malware_data_filtered|file_size      |0         |20478      |
|malware   |malware_data_filtered|file_type      |0         |67         |
|malware   |malware_data_filtered|malware_type   |0.13666667|393        |
|malware   |malware_data_filtered|month          |0         |12         |
|malware   |malware_data_filtered|num_detections |0         |112        |
|malware   |malware_data_filtered|num_week       |0         |53         |
|malware   |malware_data_filtered|sha1_hash      |0         |-1         |
|malware   |malware_data_filtered|year           |0         |4          |
+----------+--------------------+---------------+----------+-----------+
```

Código 10. Consulta que muestra estadísticas de la tabla "malware_data_filtered".

La consulta devuelve más campos, pero los que interesan para esta tarea de limpieza son los que aquí se presentan. La columna más importante es "*null_frac*" que indica el porcentaje de filas que presentan datos nulos.

Existen tres columnas que poseen datos nulos, se va a analizar cada una de ellas de forma independiente para ver qué acciones se toman al respecto con cada una de ellas.

3.3.6.1. COUNTRY (PAÍS)

La columna "*country*" tiene un 97% de datos nulos, concretamente posee 623.684 registros nulos. Dada la gran cantidad de datos se procedió a realizar un análisis para comprobar cómo están distribuidos los valores nulos en los datos. Para ello se creó y ejecuto la siguiente consulta:

```
SELECT p.year, p.num_week, (p.num_nulls/p.total::float)*100 as null_percent
FROM (SELECT l.year,
             l.num_week,
             count(*) as num_nulls,
             (SELECT count(*) as total
              FROM malware.malware_data_filtered p
              WHERE p.year = l.year AND p.num_week = l.num_week) as total
      FROM malware.malware_data_filtered l
      WHERE l.country IS NULL
      GROUP BY l.year, l.num_week
      ORDER BY l.year, l.num_week) p
```

Código 11. Consulta que muestra el porcentaje de datos nulos que posee el país.

Lo que se está calculando es el porcentaje de datos, cuyo país sea nulo, agrupado por año y número de semana, de esta forma se podrá averiguar cómo están distribuidos los valores nulos a lo largo del tiempo.

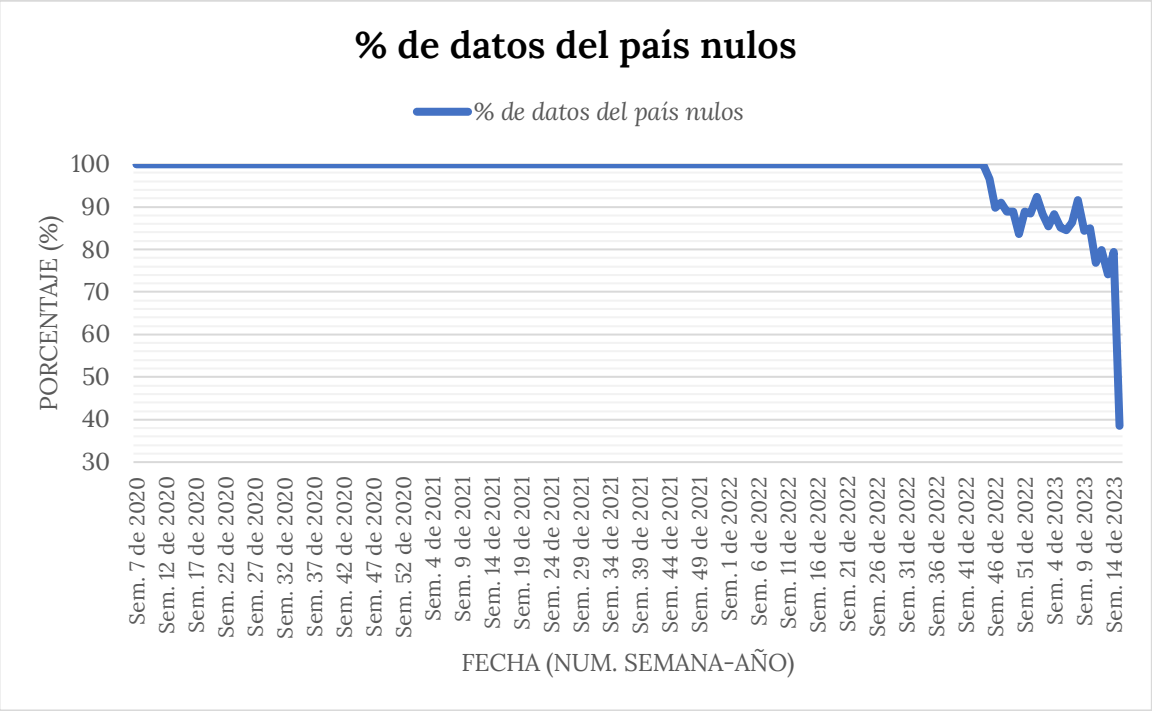


Gráfico 1. Porcentaje semanal de datos con el país con valor nulo.

Como se puede observar, no se poseen datos del país hasta la semana 44 del año 2022, esto puede deberse a que el dataset de Malware Bazaar no almacenaba esta información hasta dicha fecha, posteriormente no se almacena en el 100% de los datos, pero sí que se observa que cada vez va disminuyendo este porcentaje.

Se decidió por tanto no aplicar ninguna técnica de limpieza de datos para esta columna, pero en todas las consultas que se construirán para obtener las métricas relacionadas con el país de origen, se añadirá "*where country is not null*" de esta forma se filtrarán los resultados y solamente se devolverán aquellos que tengan información del país, ya que carece de sentido para este tipo de métricas devolver los datos para los registros que no tienen información del país.

3.3.6.2. Delivery method (Método de distribución)

La columna "*delivery_method*" tiene un 37% de datos nulos, concretamente posee 240.374 registros nulos. Antes de tomar ninguna decisión de que realizar, se procedió a realizar un pequeño análisis.

Se creó una consulta que agrupara el número de registros que posee cada uno de los métodos de distribución:

```
SELECT delivery_method, count(*) FROM malware.malware_data_filtered GROUP BY delivery_method
```

Código 12. Consulta que cuenta el número de registros que tiene cada método de distribución.

Esta consulta mostrará cada uno de los diferentes métodos de distribución junto al número de filas que tienen asignada cada categoría. Los resultados se pueden ver en la siguiente gráfica:

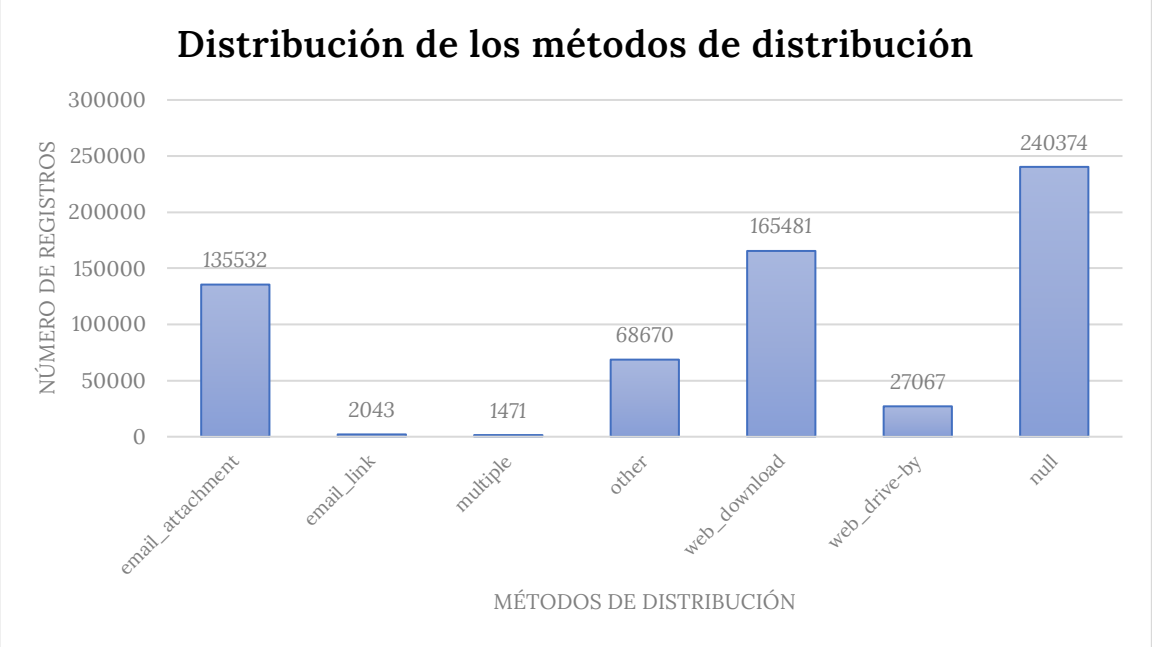


Gráfico 2. Distribución de los registros según el método de distribución.

Como se puede observar, los datos están muy desequilibrados, ya que hay cuatro categorías que tienen más de 100.000 datos, que contrastan con otras categorías que apenas llegan a los 1.000 registros. Pero esto no es un inconveniente para la visualización de los datos. Lo que hay que determinar es que hacer con los datos que no tienen asignada ninguna categoría.

Finalmente, se consideró que lo más correcto sería que a los 240.374 registros que no tienen un método de distribución se les asigne un método nuevo. Este nuevo método tendrá el nombre de "*desconocido*", en inglés "*unknown*". Esta decisión se tomó porque no era necesario eliminar todos esos registros, era una cantidad de datos lo suficientemente grande como para contemplarla dentro del dashboard de visualización de datos y tampoco existía un número elevado de categorías, ya que inicialmente solo había 6 categorías diferentes.

La sentencia que modificaría los datos para asignarles el nuevo método de distribución sería la siguiente:

```
UPDATE malware.malware_data_filtered SET delivery_method='unknown'
WHERE delivery_method IS NULL
```

Código 13. Código que actualiza los datos que no tienen método de distribución asociado.

Tras la ejecución de esta sentencia, todos los datos tendrán asignada un método de distribución.

3.3.6.3. Malware type (Tipo o familia de Malware)

La columna "*malware_type*" tiene un 13% de datos nulos, concretamente posee 86.233 registros nulos. Para estor registros se tomó la decisión de borrar estas filas, debido a que hay una gran cantidad de categorías (casi 500 categorías) y a que es un porcentaje relativamente pequeño que no influirá en las métricas a mostrar en el dashboard.

Para eliminar los registros nulos se ejecutó la siguiente sentencia:

```
DELETE FROM malware.malware_data_filtered WHERE malware_type IS NULL
```

Código 14. Sentencia que elimina todos los registros que no tienen familia de malware asociada.

Llegados a este punto, ya se habría culminado el proceso de limpieza, procesado y transformación de los datos. En la siguiente sección se mostrará algún gráfico que mostrará como es el modelo de datos resultante de todo este proceso.

3.3.7. Modelo de datos final

En esta sección se va a presentar la estructura de datos que presenta el dataset final tras culminar los procesos de limpieza, procesado y transformación de los datos.

La estructura de los datos se puede ver en la imagen adyacente, se tiene un conjunto de datos con 11 columnas, solamente lo imprescindible para poder calcular las métricas anteriormente presentadas.

Ilustración 1. Estructura que posee la tabla que contiene el modelo final de datos.

Con esta consulta se pudo obtener información sobre el tamaño y el peso del dataset:

```
SELECT 'malware_data_filtered'                                                as table,
       pg_size_pretty(pg_total_relation_size('malware.malware_data_filtered')) as size,
       (SELECT count(*) FROM malware.malware_data_filtered)                   as count
```

Código 15. Consulta que permite calcular las estadísticas de tamaño y peso del modelo final de datos.

Los resultados que se obtuvieron fueron los que se muestran a continuación:

```
+--------------------+------+------+
|table               |size  |count |
+--------------------+------+------+
|malware_data_filtered|205 MB|554405|
+--------------------+------+------+
```

Tabla 3. Tabla con los resultados que devuelve la consulta anterior.

Como se puede ver, hay una gran cantidad de información, pero contrasta con el tamaño y peso que tenían los datos que se almacenaban en MongoDB, en la primera colección donde se almacenaron. Si se extrae la información que devuelve mongo y se formatea del mismo modo que la tabla anterior, se obtiene los siguientes datos:

```
+-------------+-------+------+
|table        |size   |count |
+-------------+-------+------+
|malware_data |3.34 GB|640638|
+-------------+-------+------+
```

Tabla 4. Tabla con la información estadística del dataset inicial que se almacenó en MongoDB.

Aparecen 86.233 datos menos que se corresponden con las filas que no tenían familia de malware asociada y que se eliminaron, pero donde se aprecia el gran trabajo de limpieza realizado es en el peso. Inicialmente los datos ocupaban 3,34GB te espacio en disco, mientras que el dataset resultante ocupa 205MB, es decir, tras los procesos de limpieza se ha reducido unas 16 veces el tamaño inicial de los datos.

Para que se pueda apreciar mejor la comparativa entre el dataset inicial [34] y este dataset, se van a presentar una serie de gráficas que mostrarán la gran optimización llevada a cabo.

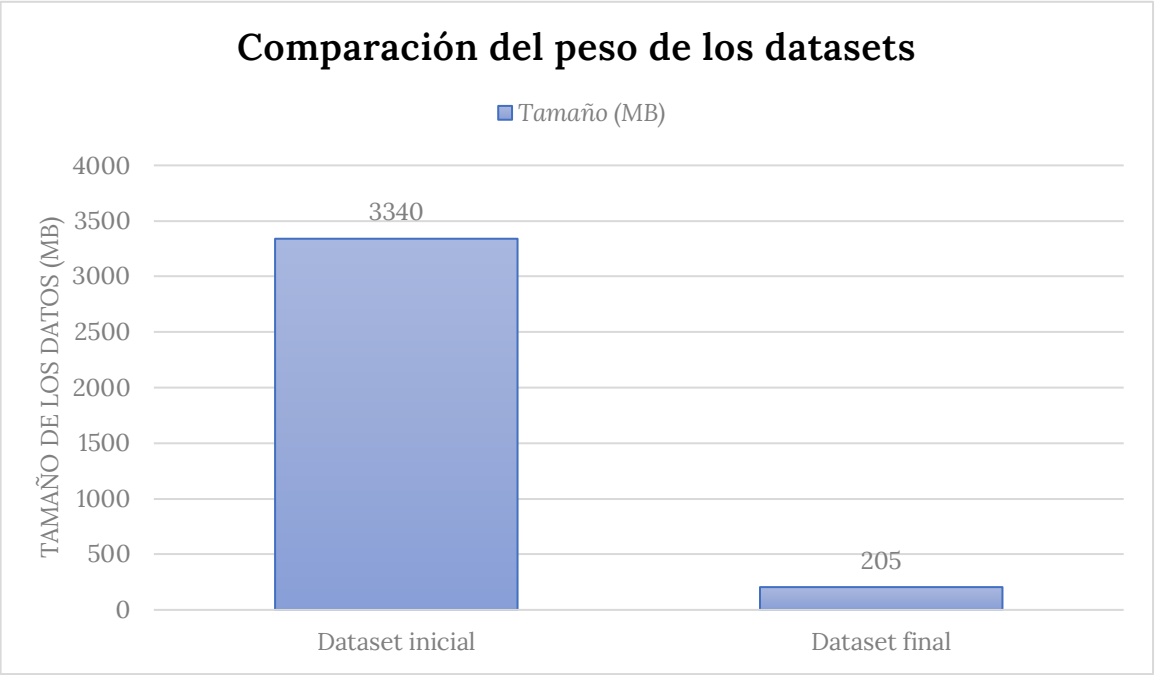


Gráfico 3. Comparativa entre el espacio que ocupa en memoria el dataset final respecto al inicial.

Ahora se puede comprender mejor el nivel de optimización tan grande que se ha realizado. Se puede ver rápidamente que el espacio en memoria que ocupa el dataset resultante es muchísimo menor que el que ocupa el dataset inicial. Toda esta optimización se debe a la modificación de las dimensiones del dataset final, ya que con los procesos de limpieza se eliminaron registros y columnas del dataset inicial, haciendo así más ligero el conjunto final.

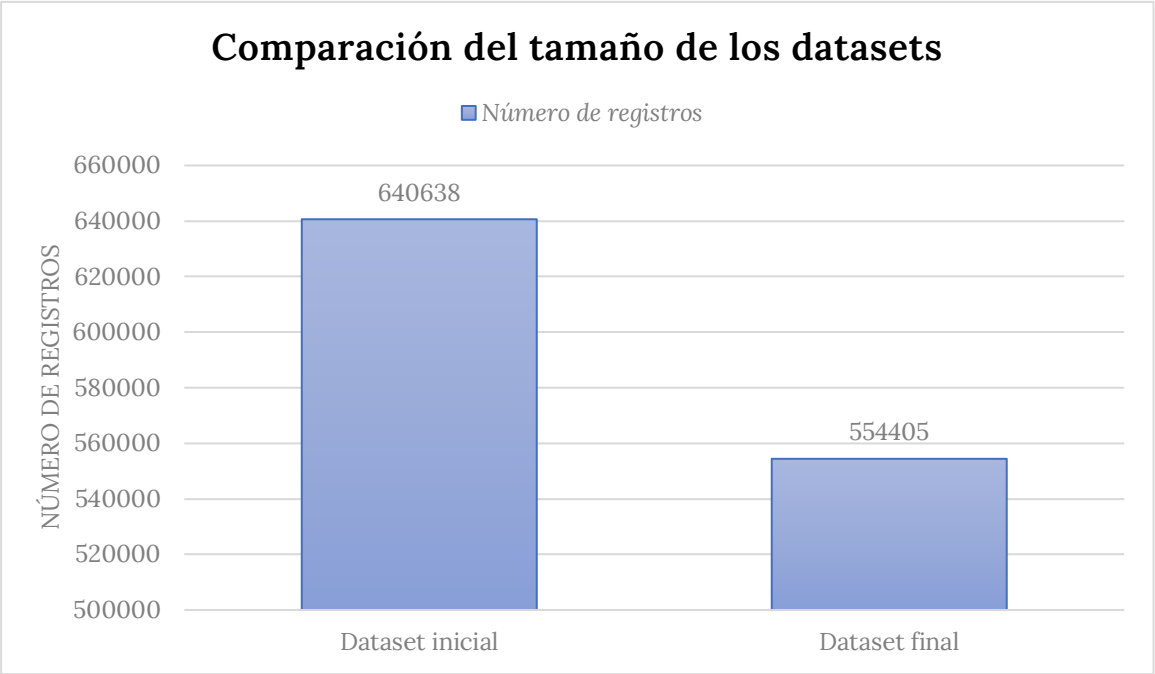


Gráfico 4. Comparativa del número de registros que posee el dataset final respecto al inicial.

Se puede ver que hay casi 90.000 registros menos en el dataset final, esto se debe a los procesos de limpieza que se realizaron en secciones anteriores.

Ahora si se realiza este mismo análisis para el número de propiedades, o columnas, que posee el dataset de inicio y el de fin, se tiene el siguiente resultado:

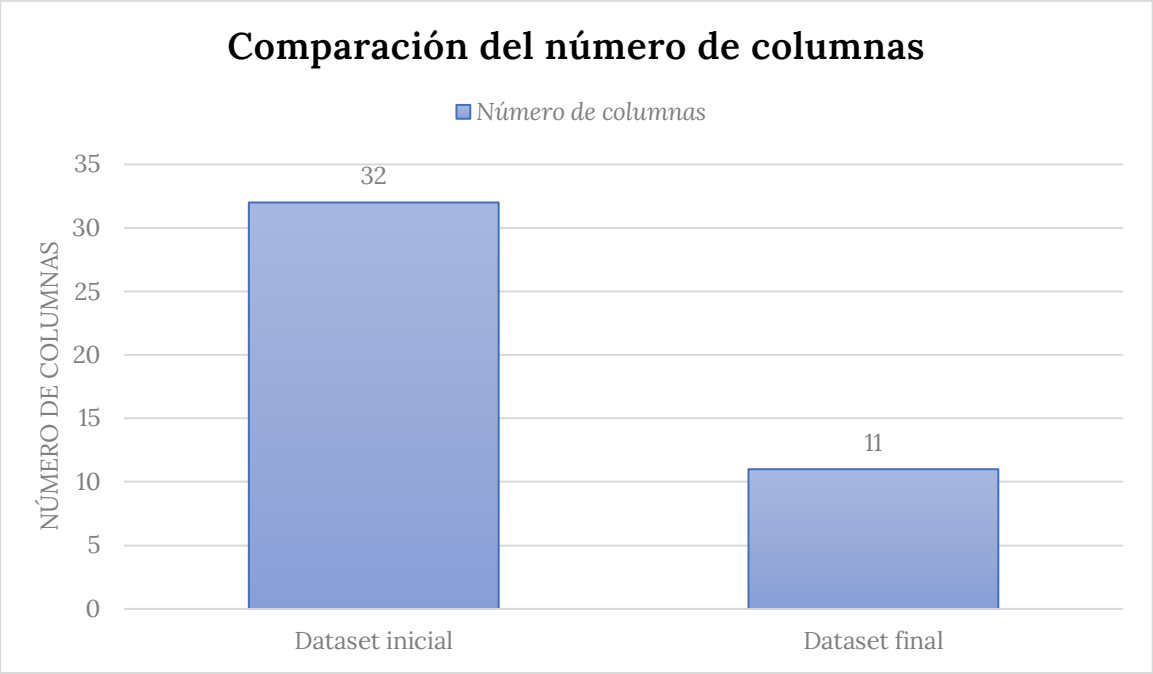


Gráfico 5. Comparativa del número de columnas o propiedades que posee el dataset final respecto al dataset inicial.

Tal y como se puede apreciar, el número de columnas se ha reducido a un tercio de las iniciales. Por lo que el dataset final [35], que será con el que se trabajará a partir de este momento, tiene unas dimensiones y un peso muy manejables para poder montar una aplicación web, ya que ninguna de las consultas devolverá la totalidad de los datos, por lo que ninguna las consultas que se realice tendrán un peso superior a 205MB, todas tendrán un tamaño inferior.

3.4. Sentencias SQL

Una vez que se ha conseguido el dataset final, ya se pueden diseñar y crear las sentencias SQL que devolverán los datos para poder calcular las métricas comentadas anteriormente.

A continuación, se va a presentar para cada una de las métricas la sentencia SQL correspondiente y la salida que ofrece. La gran mayoría de estas sentencias devolverá datos procesados, listos para llegar y construir el gráfico que corresponda.

3.4.1. Métricas asociadas a la evolución del malware en el tiempo

En la siguiente sección se van a presentar las consultas SQL para las métricas asociadas a la evolución del malware a lo largo del tiempo. A continuación, se va a presentar una subsección por cada una de las métricas de este grupo dónde se explicará la sentencia creada, destacando los puntos más importantes y se mostrará un fragmento de la salida que ofrece cada sentencia.

3.4.1.1. Evolución de la detección del malware en el mundo

La sentencia necesaria para obtener los datos para esta métrica necesitará realizar la agrupación de los datos por el número de semana del año y por el país, para poder realizar el conteo de todos los registros que tiene asociados.

```
SELECT num_week, year, country, count(*)
FROM malware.malware_data_filtered
GROUP BY num_week, year, country
ORDER BY year ASC, num_week ASC, country ASC
```

Código 16. Sentencia SQL para la métrica "evolución de la detección del malware en el mundo".

Viendo el código SQL se puede ver que gracias al "*group by*" se realiza la agrupación de los registros por el número de semana, el año y el país. Posteriormente, gracias a la función "*count(*)*" se puede obtener el número de registros asociados a dicha agrupación.

La salida de los datos se ordena de forma ascendente en el tiempo, de forma que sea más cómoda la representación de los datos, ya que se tendrá que mostrar la evolución temporal, es decir, como cambian los datos conforme aumenta el tiempo.

Los resultados que arroja esta sentencia se pueden ver a continuación, solamente se muestra un fragmento de unos pocos resultados:

```
+--------+----+-------+-----+
|num_week|year|country|count|
+--------+----+-------+-----+
|50      |2022|ID     |1    |
|50      |2022|IE     |12   |
|50      |2022|IL     |18   |
|50      |2022|IT     |32   |
|50      |2022|JP     |4    |
|50      |2022|LA     |4    |
|50      |2022|PL     |2    |
|50      |2022|RO     |1    |
+--------+----+-------+-----+
```

Tabla 5. Fragmento de los resultados de la consulta SQL de la métrica "evolución de la detección del malware en el mundo".

3.4.1.2. COMPARATIVA DEL MALWARE DETECTADO EN UN AÑO

La siguiente métrica necesita una sentencia que agrupe por cada uno de los años el conteo del número de registros que tiene asociados.

```
SELECT year, count(*)
FROM malware.malware_data_filtered
WHERE year = <year>
GROUP BY year
```

Código 17. . Sentencia SQL para la métrica "comparativa del malware detectado en un año".

Como se puede ver en el código anterior, esta sentencia es muy simple, ya que agrupa los datos por año y devuelve el conteo mediante el uso de la función "*count(*)*" .

Lo único relevante de esta sentencia es el filtrado de los datos por el año. Como esta métrica requiere ver el número de registros para un año determinado, esto se tiene que permitir mediante la cláusula "*where*", de forma que devolverá solamente los datos para el año que se le pase en el parámetro "*<year>*".

Los resultados de esta sentencia, para el parámetro "*year*" = 2022, se pueden ver a continuación:

```
+----+------+
|year|count |
+----+------+
|2022|153269|
+----+------+
```

Tabla 6. Fragmento de los resultados de la consulta SQL de la métrica "comparativa del malware detectado en un año".

3.4.1.3. COMPARATIVA DEL MALWARE DETECTADO EN UN MES

La consulta asociada a esta métrica es muy similar a la presentada en la sección anterior, ya que ahora se calcularán los datos asociados a un mes concreto, en lugar de a un año.

```
SELECT month, count(*)
FROM malware.malware_data_filtered
WHERE year = <year> AND month = <month>
GROUP BY month
```

Código 18. Sentencia SQL para la métrica "comparativa del malware detectado en un mes".

Como se puede ver, la sentencia es la misma, se modifica los datos que devuelve, cambiando el año por el mes. Otro cambio es en los criterios de filtrado, ahora se filtra por año y por mes para que solamente se devuelvan los datos asociados a un mes concreto. Si solo se añadiera el mes a los criterios de filtrado, se devolvería el acumulado de todos los meses de marzo, por ejemplo, en lugar de devolver los datos de marzo de 2022.

Los resultados que devuelve esta sentencia, para los parámetros "*year*" = 2022 y "*month*" = 3, se pueden ver a continuación:

```
+-----+-----+
|month|count|
+-----+-----+
|3    |15158|
+-----+-----+
```

Tabla 7. Fragmento de los resultados de la consulta SQL de la métrica "comparativa del malware detectado en un mes

3.4.1.4. Evolución mensual de malware

La sentencia SQL necesaria para obtener los datos para la visualización de esta métrica debe de devolver el número de registros que existen agrupados por mes y por año. Por lo tanto, el código sería el siguiente:

```
SELECT month, year, count(*)
FROM malware.malware_data_filtered
GROUP BY month, year
ORDER BY year ASC, month ASC
```

Código 19. Sentencia SQL para la métrica "evolución mensual de malware".

Como se puede observar, gracias al uso de "*group by*" la sentencia está agrupando los datos mediante los campos "*month*" y "*year*" para obtener el número de registros que posee la agrupación.

Otro aspecto por destacar es que se está realizando una ordenación de la salida para que los datos estén listos para su posterior utilización, ya que los gráficos requerirán que los datos con una fecha más antigua estén al inicio y los más recientes al final.

A continuación se puede ver un fragmento de los datos de salida:

```
+-----+----+-----+
|month|year|count|
+-----+----+-----+
|2    |2020|299  |
|3    |2020|2114 |
|4    |2020|4641 |
|5    |2020|11391|
|6    |2020|14751|
|7    |2020|12292|
|8    |2020|25217|
+-----+----+-----+
```

Tabla 8. Fragmento de los resultados de la consulta SQL de la métrica "evolución mensual de malware".

3.4.1.5. Evolución semanal de malware

La sentencia de esta métrica es muy similar a la ofrecida en la sección anterior, lo único que cambia es el nivel de agrupación que se realizará, agrupando por el número de semana del año, en lugar de agrupar por el mes.

```
SELECT num_week, year, count(*)
FROM malware.malware_data_filtered
GROUP BY num_week, year
ORDER BY year ASC, num_week ASC
```

Código 20. Sentencia SQL para la métrica "evolución semanal de malware".

Esto se ve reflejado en el código, como se puede ver es idéntico al ofrecido en la sección anterior, solamente se sustituye el campo "*month*" por el campo "*num_week*", el resto de la sentencia es igual.

Los resultados que arroja este código SQL se pueden ver a continuación:

```
+--------+----+-----+
|num_week|year|count|
+--------+----+-----+
|1       |2021|1279 |
|2       |2021|1711 |
|3       |2021|2092 |
|4       |2021|988  |
|5       |2021|1495 |
|6       |2021|1410 |
|7       |2021|1339 |
|8       |2021|2894 |
+--------+----+-----+
```

Tabla 9. Fragmento de los resultados de la consulta SQL de la métrica "evolución semanal de malware".

3.4.2. Métricas asociadas a las familias de malware

En la siguiente sección se van a presentar las consultas SQL para las métricas asociadas a las familias de malware. Al igual que en la sección anterior, se van a mostrar las métricas de este grupo y se presentará, para cada una de ellas, la sentencia SQL y los resultados que ofrece.

3.4.2.1. Tipos de malware más comunes

Para esta métrica se tiene que desarrollar una sentencia muy similar a las vistas anteriormente, habrá que agrupar los datos por el tipo de malware para poder obtener el conteo de registros asociados a cada familia.

Posteriormente, habrá que ordenar los datos de salida para facilitar la generación del gráfico asociado a la visualización de la métrica.

```
SELECT malware_type, count(*) as result
FROM malware.malware_data_filtered
GROUP BY malware_type ORDER BY count(*) DESC
```

Código 21. Sentencia SQL para la métrica "Tipos de malware más comunes".

Tal y como se puede ver, se agrupan los datos por la familia de malware y mediante la función "*count*(*)" se obtiene el total de registros asociados a cada agrupación. Finalmente, los datos se ordenan por este conteo de forma descendente, es decir, en los primeros puestos aparecerán aquellos valores que tengan un coteo más elevado e ira bajando hasta llegar a la familia con menos registros asociados.

Los resultados de esta sentencia se pueden ver a continuación:

```
+--------------+------+
|malware_type  |result|
+--------------+------+
|Heodo         |95303 |
|Quakbot       |62065 |
|AgentTesla    |61961 |
|Dridex        |36008 |
|Mirai         |31878 |
|Formbook      |24483 |
|RedLineStealer|24234 |
+--------------+------+
```

Tabla 10. Fragmento de los resultados de la consulta SQL de la métrica "Tipos de malware más comunes".

3.4.2.2. Variantes de una familia de malware vs. infecciones detectadas

Esta métrica es un poco especial, porque hace una comparación entre dos métricas diferentes: la primera es la métrica que se ha comentado en la sección anterior (número de registros agrupado por cada tipo de malware) y la segunda métrica es el acumulado número de detecciones para cada tipo de malware.

Por tanto, para hacer la comparación se utilizará por una parte la sentencia SQL comentada en la sección anterior y otra diferente. Esta otra sentencia

será similar a la anterior, salvo que en lugar de calcular el resultado mediante "*count(*)*" habrá que calcular el acumulado del número de detecciones, es decir, habrá que utilizar la función "*sum(num_detections)*". Además el resultado habrá que ordenarlo utilizando este acumulado, y de la misma forma que en la otra sentencia, habrá que ordenar los datos de forma descendente.

```
SELECT malware_type, sum(num_detections) as result
FROM malware.malware_data_filtered
GROUP BY malware_type
ORDER BY sum(num_detections) DESC
```

Código 22. Sentencia SQL para la métrica "Variantes de una familia de malware vs. infecciones detectadas".

Poco hay que añadir sobre esta sentencia, ya que se ha explicado todo lo relevante. Los resultados que arroja son los siguientes:

```
+--------------+------+
|malware_type  |result|
+--------------+------+
|RedLineStealer|309286|
|Mirai         |222669|
|Heodo         |206784|
|ArkeiStealer  |173777|
|AgentTesla    |111793|
|Quakbot       |85909 |
|Smoke Loader  |85222 |
|GCleaner      |75542 |
+--------------+------+
```

Tabla 11. Fragmento de los resultados de la consulta SQL de la métrica "Variantes de una familia de malware vs. infecciones detectadas".

3.4.2.3. TIPOS DE MALWARE MÁS COMUNES POR PAÍS

Finalmente para esta agrupación de métricas, la sentencia necesaria tendrá que agrupar los datos utilizando el país y la familia de malware. Al igual que en otras consultas, habrá que ordenar los datos para facilitar la representación de los mismos.

```
SELECT country, malware_type, count(*) as result
FROM malware.malware_data_filtered
GROUP BY country, malware_type
ORDER BY country ASC, count(*) DESC
```

Código 23. Sentencia SQL para la métrica "Tipos de malware más comunes por país".

Como se puede apreciar, se agrupan los datos mediante el país y la familia de malware y se obtiene el número de registros de la agrupación. Finalmente, hay que destacar que para realizar la ordenación de los datos se utiliza un doble criterio. En primer lugar se ordenan los datos por país, para que estén todos los datos de un mismo país seguidos, después se reordenan para que aparezcan con el conteo de forma descendente.

Finalmente, se presenta un extracto del resultado que arroja la sentencia SQL anterior:

country	malware_type	result
AE	IcedID	4
AF	Quakbot	1
AR	Mekotio	50
AR	Casbaneiro	5
AR	IcedID	4
AR	BumbleBee	4
AR	BrasDex	2
AR	AZORult	2

Tabla 12. Fragmento de los resultados de la consulta SQL de la métrica "Tipos de malware más comunes por país".

3.4.3. MÉTRICAS ASOCIADAS A LOS MÉTODOS DE DISTRIBUCIÓN DEL MALWARE

En la siguiente sección se van a presentar las consultas SQL para las métricas asociadas a los métodos de distribución de malware. Al igual que en las secciones anteriores, se van a mostrar las métricas de este grupo presentando para cada una el código SQL y un fragmento de los resultados que arroja.

3.4.3.1. MÉTODOS DE DISTRIBUCIÓN MÁS COMUNES POR TIPO DE MALWARE

Para calcular la información asociada a esta métrica habrá que agrupar los datos por la familia de malware y por el método de distribución.

Posteriormente habrá que realizar un conteo del número de registros que tiene cada agrupación.

```
SELECT malware_type, delivery_method, count(*)
FROM malware.malware_data_filtered
GROUP BY malware_type, delivery_method
ORDER BY malware_type ASC, count(*) DESC
```

Código 24. Sentencia SQL para la métrica "Métodos de distribución más comunes por tipo de malware".

Como se puede ver, se hace la agrupación de los datos por las propiedades comentadas anteriormente y se realiza el cálculo de los registros que posee cada una de estas agrupaciones gracias a la función "*count*(*)".

Finalmente se realiza una ordenación similar a las explicadas anteriormente, es una ordenación de doble criterio que ordena los datos por familia de malware en primer lugar y posteriormente los reordena para que el coteo de los registros de familia de malware se dispongan de mayor a menor conteo.

A continuación aparece un extracto del resultado de esta sentencia:

```
+------------+---------------+-----+
|malware_type|delivery_method|count|
+------------+---------------+-----+
|000Stealer  |web_download   |4    |
|000Stealer  |web_drive-by   |2    |
|000Stealer  |unknown        |1    |
|1ms0rryMiner|web_download   |6    |
|1ms0rryMiner|unknown        |4    |
+------------+---------------+-----+
```

Tabla 13. Fragmento de los resultados de la consulta SQL de la métrica "Métodos de distribución más comunes por tipo de malware".

3.4.3.2. MÉTODOS DE DISTRIBUCIÓN MÁS COMUNES POR TIPO DE FICHERO

Esta métrica necesitará una sentencia que sea idéntica a la anterior, solamente cambiará el criterio de agrupación. En lugar de utilizar la familia de malware, habrá que utilizar el tipo de fichero.

Por tanto, la sentencia será prácticamente idéntica, solo cambiará la columna de familia de malware por la de tipo de fichero:

```
SELECT file_type, delivery_method, count(*)
FROM malware.malware_data_filtered
GROUP BY file_type, delivery_method
ORDER BY file_type ASC, count(*) DESC
```

Código 25. Sentencia SQL para la métrica "Métodos de distribución más comunes por tipo de fichero".

Poco hay que destacar de este fragmento de código, ya que tal y como se ha comentado, es idéntico al que se explicó en la sección anterior.

El fragmento con los resultados que arroja esta sentencia se pueden ver a continuación:

```
+---------+----------------+-----+
|file_type|delivery_method |count|
+---------+----------------+-----+
|001      |email_attachment|7    |
|001      |unknown         |1    |
|7z       |email_attachment|894  |
|7z       |multiple        |311  |
|7z       |unknown         |148  |
+---------+----------------+-----+
```

Tabla 14. Fragmento de los resultados de la consulta SQL de la métrica "Métodos de distribución más comunes por tipo de fichero".

3.4.3.3. Métodos de distribución de malware más comunes por país

Al igual que ocurría en la métrica anterior, aquí se va a utilizar prácticamente el mismo código SQL que el utilizado en la sentencia anterior. Ahora se utiliza como criterio agrupador el país.

```
SELECT country, delivery_method, count(*)
FROM malware.malware_data_filtered
GROUP BY country, delivery_method
ORDER BY country ASC, count(*) DESC
```

Código 26. Sentencia SQL para la métrica "Métodos de distribución de malware más comunes por país".

A continuación se puede ver un fragmento de los resultados que devuelve la sentencia SQL anterior:

```
+-------+---------------+-----+
|country|delivery_method |count|
+-------+---------------+-----+
|AE     |unknown         |4    |
|AF     |unknown         |1    |
|AR     |unknown         |61   |
|AR     |email_link      |7    |
|AR     |other           |1    |
|AR     |web_download    |1    |
|AT     |email_attachment|4    |
|AU     |unknown         |26   |
+-------+---------------+-----+
```

Tabla 15. Fragmento de los resultados de la consulta SQL de la métrica "Métodos de distribución de malware más comunes por país".

3.4.4. MÉTRICAS ASOCIADAS A LAS CARACTERÍSTICAS DE LOS FICHEROS DE MALWARE

En la siguiente sección se van a presentar las consultas SQL para las métricas asociadas a las características de los ficheros de malware. Al igual que en las secciones anteriores, se va a mostrar para cada una de las métricas de este grupo el fragmento SQL de cada consulta y un extracto de los resultados.

3.4.4.1. TIPOS DE MALWARE MÁS COMUNES PARA CADA TIPO DE FICHERO

Esta métrica sigue la misma estructura de código SQL que las sentencias anteriores. Esta vez los criterios de agrupación será la familia de malware y el tipo de fichero. Poco más se puede comentar, ya que en las secciones anteriores se describió correctamente los aspectos a destacar de la sentencia.

```
SELECT malware_type, file_type, count(*) as result
FROM malware.malware_data_filtered
GROUP BY malware_type, file_type
ORDER BY malware_type ASC, count(*) DESC
```

Código 27. Sentencia SQL para la métrica "Tipos de malware más comunes para cada tipo de fichero".

A continuación se muestra un fragmento de los resultados que arroja la consulta anterior.

malware_type	file_type	result
000Stealer	exe	7
1ms0rryMiner	exe	10
1xxbot	exe	13
404keylogger	exe	1
404Keylogger	exe	92
404Keylogger	arj	30
404Keylogger	zip	14
404Keylogger	rar	8

Tabla 16. Fragmento de los resultados de la consulta SQL de la métrica "Tipos de malware más comunes para cada tipo de fichero".

3.4.4.2. DISTRIBUCIÓN DEL TIPO DE FICHEROS MALICIOSOS

Esta consulta tiene el formato un poco diferente a las anteriores, pero se sigue el mismo concepto. Se agruparán los datos mediante el tipo de fichero y se calculará el número de elementos que compone a cada agrupación.

```
SELECT file_type, count(*) as result
FROM malware.malware_data_filtered
GROUP BY file_type
ORDER BY count(*) DESC
```

Código 28. Sentencia SQL para la métrica "Distribución del tipo de ficheros maliciosos".

Hay que destacar que la salida de los datos será ordenada por el conteo de resultados de forma descendente.

A continuación se puede ver un extracto de la salida de dicha sentencia SQL:

file_type	result
exe	271427
xlsx	57117
dll	53586
elf	39438
docx	27469
zip	19712

Tabla 17. Fragmento de los resultados de la consulta SQL de la métrica "Distribución del tipo de ficheros maliciosos".

3.4.4.3. TAMAÑO MEDIO DE LOS FICHEROS MALICIOSOS SEGÚN EL TIPO DE MALWARE

Esta consulta tendrá que agrupar los datos por la familia de malware y después calcular la media del tamaño de los ficheros para cada agrupación.

```
SELECT malware_type, avg(file_size) as result
FROM malware.malware_data_filtered
GROUP BY malware_type
ORDER BY malware_type ASC, avg(file_size) DESC
```

Código 29. Sentencia SQL para la métrica "Tamaño medio de los ficheros maliciosos según el tipo de malware".

Tal y como se puede ver en el código SQL los datos se agrupan mediante la familia de malware y después, para calcular el promedio del tamaño del fichero se utiliza la función "*avg(file_size)*".

Finalmente, se ordenan los datos para que aparezcan ordenados por el tipo de malware y luego por el promedio de forma descendente, de esta forma estarán preparados para la representación en los gráficos.

A continuación, se muestra un fragmento de los resultados que devuelve la sentencia anterior:

```
+----------------+--------------------+
|malware_type    |result              |
+----------------+--------------------+
|000Stealer      |4354378.285714285714|
|1ms0rryMiner    |6818626             |
|1xxbot          |696411.461538461538 |
|404keylogger    |130560              |
|404Keylogger    |536107.181818181818 |
|4444            |803925.333333333333 |
|44CaliberStealer|758357.333333333333 |
|5050            |6532                |
+----------------+--------------------+
```

Tabla 18. Fragmento de los resultados de la consulta SQL de la métrica "Tamaño medio de los ficheros maliciosos según el tipo de malware".

3.4.4.4. Tamaño medio de los ficheros maliciosos según el tipo de fichero

Esta consulta es similar a la presentada en la sección anterior, solamente hay que modificar el criterio de agrupación para que se calcule el promedio del tamaño de ficheros para cada tipo de fichero.

```
SELECT file_type, avg(file_size) as result
FROM malware.malware_data_filtered
GROUP BY file_type
ORDER BY file_type ASC, avg(file_size) DESC
```

Código 30. Sentencia SQL para la métrica "Tamaño medio de los ficheros maliciosos según el tipo de fichero".

Como se puede comprobar, el código de la sentencia SQL es idéntico al de la métrica anterior, por tanto, poco se puede añadir a lo que se comentó anteriormente.

Finalmente, se presenta un fragmento con los resultados que arroja la sentencia anterior.

```
+---------+--------------------+
|file_type|result              |
+---------+--------------------+
|001      |1226630.375         |
|7z       |912377.77505407354  |
|ace      |496576.827822120867 |
|apk      |6186581.044731610338|
|arj      |421662.466115702479 |
|b1       |622615              |
|bat      |998907.951048951049 |
|bz       |501041.5            |
+---------+--------------------+
```

Tabla 19. Fragmento de los resultados de la consulta SQL de la métrica "Tamaño medio de los ficheros maliciosos según el tipo de fichero".

Llegado a este punto, ya se tienen los datos preparados, las métricas definidas y se han preparado las sentencias SQL que obtienen los datos necesarios para poder construir las visualizaciones de las métricas.

En el siguiente apartado se va a abordar el diseño de la aplicación a desarrollar, tanto de la parte del backend que expondrá un API REST para poder hacer peticiones para obtener los datos para construir las métricas, haciendo uso de las sentencias SQL aquí presentadas.

Como la parte del frontend, donde se diseñará el dashboard que plasmará los gráficos y otros elementos de representación para mostrar las diferentes visualizaciones de las métricas presentadas.

Todo ello se detallará en la siguiente sección, la sección de diseño de la aplicación desarrollada.

4. Diseño

Una vez que se tenía el dataset preparado, se procedió a diseñar el sistema que consumiría dichos datos, los transformaría y serviría para mostrar a los usuarios las métricas que se definieron en secciones anteriores.

Se partía del hecho de que se quería utilizar este proyecto para investigar y aprender lenguajes y técnicas que se están utilizando en la actualidad para montar aplicaciones web. Se planteó como reto utilizar Angular y Typescript para la implementación del sistema.

Además, se quería hacer uso de *arquitecturas limpias* para tener una solución más simple, limpia, flexible y robusta. Ya que las arquitecturas limpias son unos enfoques de diseño de software que buscan promover la creación de aplicaciones mantenibles y escalables, manteniendo un alto nivel de cohesión y bajo acoplamiento entre las capas, promoviendo la reutilización de código y facilitando el mantenimiento y la evolución del software a lo largo del tiempo.

Existen muchos tipos de arquitecturas limpias: hexagonal, onion, MVP, etc. para este trabajo se decidió utilizar la arquitectura hexagonal, ya que se había leído artículos sobre ella y pareció que era una arquitectura muy útil, robusta y versátil. Por ello, se decidió utilizarla en este proyecto, para poder estudiarla en profundidad y aplicarla dentro de un proyecto de ingeniería real.

4.1. Arquitectura hexagonal

La arquitectura hexagonal busca separar la lógica de negocio de los detalles de la infraestructura tecnológica y la tecnología utilizada. Esta separación se realiza a través de puertos y adaptadores.

El núcleo de la aplicación debe ser independiente de la tecnología y de los detalles de la infraestructura. De esta forma, se logra una mayor flexibilidad

y escalabilidad, ya que se pueden cambiar los detalles de la infraestructura sin tener que cambiar la lógica de negocio de la aplicación.

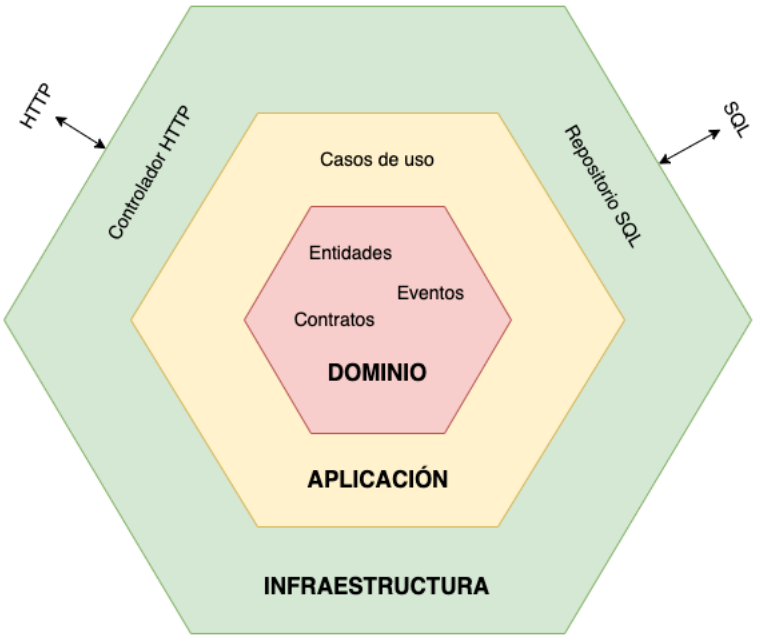


Ilustración 2. Diagrama que representa la disposición de las capas de la arquitectura hexagonal.

La arquitectura hexagonal propone que el dominio de la aplicación sea el núcleo de las capas y que este no se acople a nada externo, por tanto, las capas interiores de esta arquitectura no pueden conocer nada de las capas exteriores. En lugar de hacer uso explícito, y mediante el *principio de inversión de dependencias*, habrá que acoplarse mediante contratos (interfaces o puertos) y no a implementaciones concretas.

Gracias a estos contratos se consigue que las capas de la aplicación estén desacopladas. Los *contratos o puertos* se definen mediante interfaces públicas que exportan el listado de operaciones que ofrecen a la capa que las utilizará.

Posteriormente, se definirán *las especificaciones de los puertos o adaptadores*, que son las implementaciones de las interfaces que definen a los puertos, para que otros módulos puedan implementarlas y comunicarse con la capa de negocio sin que esta sepa el origen de la conexión.

Gracias a este desacoplamiento se obtiene también la ventaja de poder testear las capas sin que intervengan otras externas.

Esta arquitectura se suele representar con forma de hexágono, pero el número de lados no es lo que importa, sino lo que estos representan. Cada lado representa un puerto, hacia dentro o fuera, de la aplicación.

4.2. Domain-Driven Design

Domain-Driven Design o *DDD* es una técnica de diseño de software que se centra en el análisis y diseño del dominio del problema *[36]*, *[37]*. Se utiliza para crear software de calidad que se ajuste a los requisitos del negocio. Permite un uso eficiente del código y que el software sea fácil de mantener y modificar.

Esta técnica de diseño ayuda a los desarrolladores a entender mejor el dominio del problema, por lo que, a su vez, conlleva a crear soluciones más efectivas y específicas.

También permite una mejor mantenibilidad y evolución del código, ya que el diseño esta más cercano al modelo mental del dominio del problema. Al ser más fácil de entender y modificar, es muy sencillo implementar cambios y mejoras en el sistema.

Pero, el DDD, puede ser difícil de aplicar en casos en los que el modelo de dominio es complejo o no está bien definido. Así como puede añadir una mayor complejidad en el código si este no se implementa correctamente.

En resumen, el DDD es una técnica que permite mejorar la calidad del código desarrollado, gracias a que se centra en el análisis del dominio del problema.

4.3. Arquitectura Hexagonal y DDD

A lo largo de las secciones anteriores, se presentó la arquitectura hexagonal y DDD. Ambos conceptos, por separado, ofrecen al desarrollador las herramientas necesarias para desarrollar una aplicación de calidad, robusta, flexible, escalable y mantenible. Pero es posible unir ambos conceptos para conseguir una solución muy sólida.

La arquitectura hexagonal fomenta que el dominio sea el núcleo de todas las capas, y que no se acople a nada externo. Esto encaja a la perfección con la idea del DDD.

Gracias al desacoplamiento de las capas, la unión de la arquitectura hexagonal y el DDD hará que la aplicación resultante tenga los siguientes beneficios:

- Alta **testabilidad**, gracias el principio SOLID[5] de *inversión de dependencias* para la interacción del dominio con el resto de elementos.
- Alta **tolerancia al cambio**, gracias al principio SOLID de *abierto/cerrado*.
- Alta **reutilización de código**, gracias al principio SOLID de *responsabilidad* única se consigue la división estricta de responsabilidades a nivel de aplicación y dominio.
- Permite **postponer decisiones de implementación**, gracias al estar basada en contratos establecidos por los puertos en lugar de basarse en implementaciones concretas.

Por todo ello, se decidió utilizar la arquitectura hexagonal junto a DDD para la implementación de la aplicación a desarrollar. Además, se aplicarán otras técnicas de desarrollo de software para que la calidad del código

[5] Los principios SOLID son un conjunto de principios aplicables a la Programación Orientada a Objetos que, si se usan correctamente, permiten desarrollar software de calidad en cualquier lenguaje de programación orientada a objetos. El código resultante de su utilización será más fácil de leer, testear y mantener.

desarrollado sea todavía mejor, pero esto se comentará en la sección de implementación.

Una vez que se han comentado los aspectos relevantes para el desarrollo de la parte *backend* de la aplicación a desarrollar, se va a detallar el diseño de todo lo referente al dashboard. Partiendo de los primeros bocetos en papel, hasta llegar al diseño final del panel y del diseño de cada una de las visualizaciones de métricas.

4.4. Boceto del dashboard

A lo largo de la presente sección se va a comentar como se forjó el diseño del dashboard de Malware, partiendo desde los primeros bocetos en papel hasta llegar al diseño final del mismo. Comentando todas las decisiones tomadas a lo largo del camino.

Es importante señalar que esta es una fase de diseño y que, en ningún momento se va a mezclar con la fase de implementación. Es decir, hasta que no se culmine la fase de diseño no se pasará a la fase de implementación, por consiguiente, no se escribirá ni una sola línea de código hasta que no se tenga claro el diseño final del panel.

4.4.1. ¿Dónde se ejecutará la aplicación?

Lo primero que se tenía que decidir era a que tipo de dispositivos iba a ir orientado el uso de la aplicación, ya que dependiendo del tipo de estos habrá que distribuir la información de una forma u otra. Las opciones disponibles son: móvil/tablet, ordenador o hibrido.

Una aplicación para móvil/tablet tendrá una baja resolución de pantalla, ya que aunque cada vez fabrican dispositivos con pantallas más grandes, los tamaños rondan entre 5-10 pulgadas. Además, dado que la aplicación estará en la web, hay algunas funciones o librerías que no funcionan en los

navegadores de móviles. El punto fuerte de usar móviles/tablet es que puedes visualizar la información cuando quieras y desde donde quieras.

Una aplicación para ordenador no tiene estos problemas, los ordenadores portátiles suelen tener un tamaño de pantalla superior a 13 pulgadas y en la mayoría de los casos se pueden conectar pantallas secundarias con un tamaño mucho mayor. Para el caso de los ordenadores, si es portátil no habría inconveniente en ver la información correcta desde donde se quiera, pero si es un ordenador de sobremesa, solo se podrá ver desde el lugar donde esté instalado el equipo.

La opción hibrida es desarrollar una aplicación que se adapte al tipo de pantalla y que sea universal, es decir que funcione tanto para móvil como para ordenadores. Esta opción parece muy buena, pero el desarrollar una aplicación de este estilo podría tener un coste de desarrollo muy elevado, ya que habría que revisar que se viera todo perfectamente en ambos tipos de dispositivos.

Tras barajar detenidamente todas las opciones, se llegó a la conclusión de que la opción solo para móviles se descartaba totalmente, ya que son unos dispositivos que tienen un tamaño de pantalla muy muy pequeño, donde apenas podría visualizarse algunas de las métricas seleccionadas.

Por consiguiente, entre las dos opciones restantes, se decidió que la aplicación tendría que desarrollarse para ordenadores. Carece de sentido desarrollar una aplicación híbrida cuando se descartó inicialmente la opción de móvil por el tamaño tan pequeño de las pantallas. Además, el desarrollo de una versión 100% responsive haría que el número de horas de desarrollo se incrementara bastante, ya que habría que comprobar que cada pantalla, gráfico, elemento, etc. se ve y se comporta correctamente para cualquier tipo de resolución.

Concluyendo que, como se quiere para grandes resoluciones, se descartó la opción híbrida y se centrarán los esfuerzos en hacer que el panel se vea correctamente en pantallas grandes. Se podría estudiar, como trabajos futuros, el desarrollar una aplicación híbrida que permitiera ver el panel de tablets, ya que estas tienen la pantalla un poco más grande que los móviles. Pero sería trabajos futuros, ya que desarrollarlo dentro de este proyecto haría que se superaran los límites establecidos para un trabajo final de master.

4.4.2. ¿Cómo se distribuirá la información?

Una vez elegido el tipo de dispositivo que ejecutará la aplicación, se podía proceder con el diseño del panel propiamente dicho. Tal y como se comentó en secciones anteriores, concretamente en la sección 3.2, las métricas que compondrán el panel se agrupan en cuatro conjuntos diferentes. Por tanto, lo primero que se debía tener en cuenta es como se podría distribuir la información para se viera de una forma clara y directa. Es decir, había que idear una solución que permita distinguir a simple vista cada una de las agrupaciones.

Dada esta condición, se pensó que la mejor opción sería crear una aplicación que tuviera un menú donde el usuario pudiera seleccionar el tipo de agrupación y que, al seleccionarla, el panel cargase las métricas asociadas. A continuación, se muestran los primeros bocetos que se crearon siguiendo estas premisas para definir la estructura del panel.

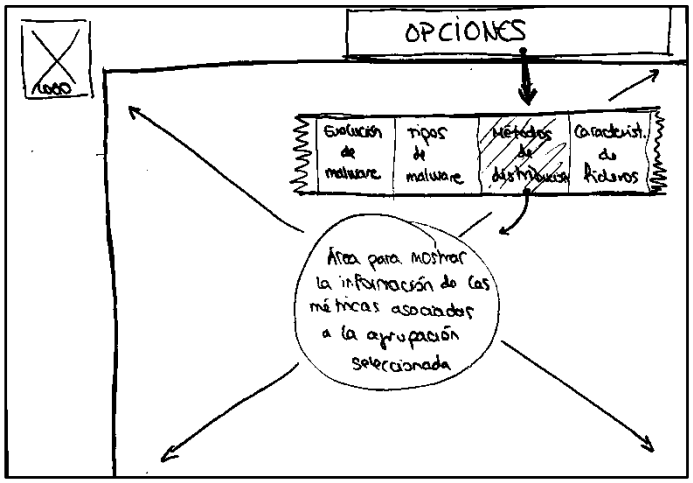


Ilustración 3. Boceto de la primera opción propuesta del diseño de la interfaz básica del dashboard.

La primera opción es la que se puede ver aquí contigua. El menú de operaciones estaría situado en la parte superior de la pantalla, en la cabecera de la aplicación. Cuando el usuario seleccione la agrupación que desee, la información de las métricas se cagaría en la parte principal de la pantalla.

La segunda opción cambiaría la situación del menú de operaciones, ahora se situaría en la barra de la izquierda. La funcionalidad sería completamente igual, solo cambiaría el lugar donde se coloca dentro de la pantalla.

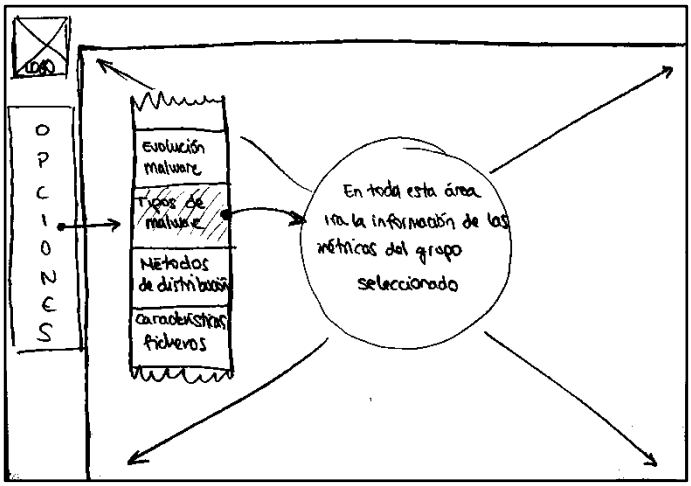


Ilustración 4. Boceto de la segunda opción propuesta del diseño de la interfaz básica del dashboard.

Ambas opciones tendrían un logo situado arriba a la izquierda, esto se decidió ya que, hoy en día todas las aplicaciones poseen una imagen que las identifica. Dado que esta aplicación está destinada al público en general y que estará desplegada en la nube, se consideró necesaria la presencia de un logo cuyos colores permitieran tener como resultado un panel acogedor, vistoso y elegante.

Tras darle muchas vueltas, se decidió utilizar la opción número dos. Ya que se creyó que sería la más fácilmente manejable porque los usuarios están acostumbrados a utilizar aplicaciones para ordenador con el menú de

operaciones situado a la izquierda (algunos clientes de email, páginas de comercio electrónico, redes sociales, etc.).

Ahora, en las siguientes secciones se determinará el diseño de los paneles para cada una de las agrupaciones de métricas.

4.4.3. Panel para las métricas de evolución

El primer panel que se procedió a diseñar era el destinado a la visualización de las métricas asociadas a la evolución del malware en el tiempo.

A continuación se presenta el boceto que se realizó con el diseño de este primer panel.

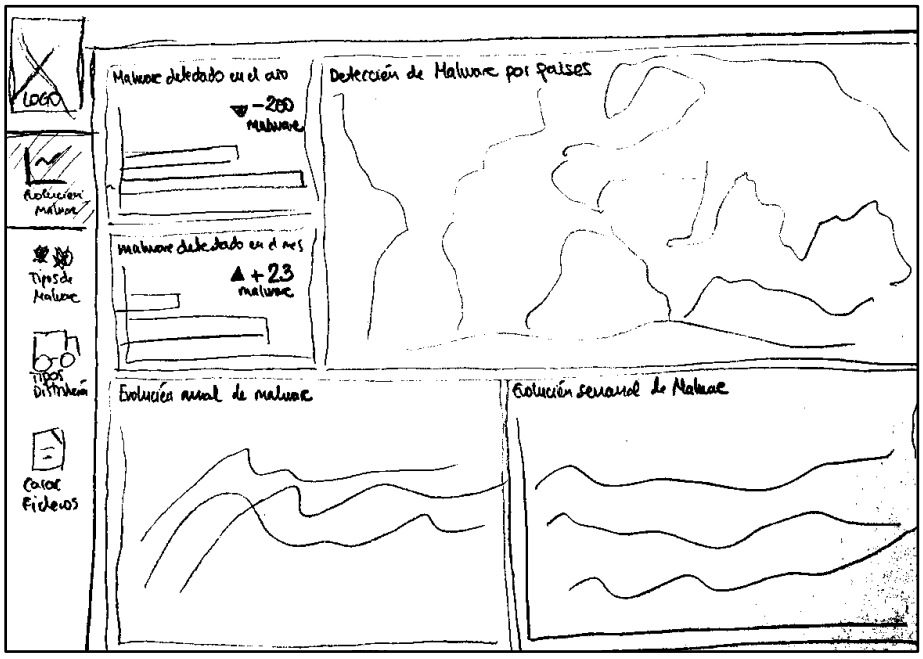


Ilustración 5. Boceto del diseño que tendrá el panel para las métricas de evolución.

Como se puede observar, dentro de este diseño aparecen las cinco métricas asociadas a esta agrupación. Para este diseño se tuvo en cuenta que métricas tenían más peso visual para asignarles un mayor espacio en la pantalla.

Se creyó que la mejor forma para distribuir la información era en dos filas diferentes, que más o menos ocuparan el 50% del espacio disponible, es decir, que ambas filas tuvieran la misma altura o similar.

En la primera fila, se le da mucho más peso visual al mapa que a la visualización de las métricas de comparación anual y mensual de malware. Esto se debe a tres razones: una es que las métricas de comparación no necesitan un gran tamaño para visualizarse correctamente, otra razón es que es mucho más relevante la información que se muestra en el mapa y la última es que se desea que el mapa llame la atención del usuario por lo que hacerlo más grande será la mejor opción para conseguir ese cometido.

En la segunda fila, se mostrarán dos gráficas que tienen la misma importancia, por este motivo en el boceto aparecen con el mismo tamaño. Se decidió ponerlas dentro de la misma fila, porque la información que se representa es la misma pero agrupada por un periodo diferente. A la izquierda los datos aparecen agrupados por meses, mientras que a la derecha están agrupados semanalmente. Al encontrarse en la misma fila, será más sencillo que los usuarios puedan analizarlas al mismo tiempo para poder ver las diferencias y sacar conclusiones de los datos.

Este sería el diseño final para este conjunto de métricas, posteriormente en la fase de implementación habrá que implementarlo para que sea lo más parecido posible a este diseño.

Hay que destacar que los tipos de gráficos que aparecen en el boceto puede ser que no sean los que finalmente se utilicen, ya que el tipo de gráfico que se utilizará para cada métrica se decidirá en secciones posteriores.

4.4.4. PANEL PARA LAS MÉTRICAS DE TIPOS DE MALWARE

El siguiente panel que se diseñó fue el que agruparía las métricas asociadas a la familia de malware. El boceto que se realizó se puede ver a continuación:

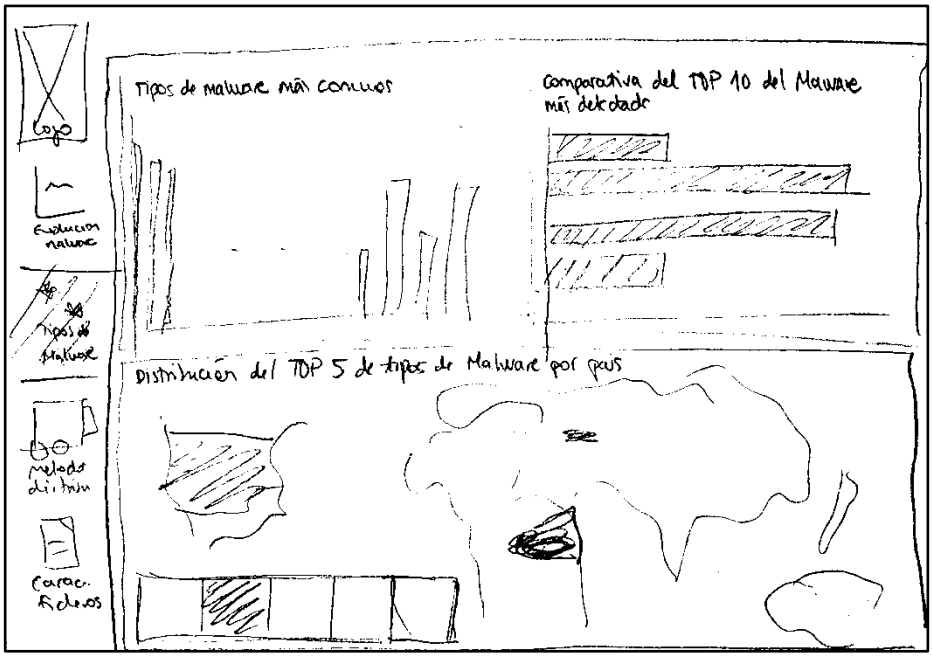


Ilustración 6. Boceto del diseño que tendrá el panel para las métricas de tipos de malware.

Como se puede observar, el diseño es muy similar al que se ofreció en la sección anterior, y es muy probable que todos los paneles tengan esta misma estructura, ya que en dos filas se representa muy bien la información. Dado que al tratarse de pantallas de ordenador el tamaño es grande y se podrá ver cada una de las métricas perfectamente.

Para esta pantalla, se consideró que el elemento que más captase la atención de usuario fuera el mapa. Ya que, con una adecuada elección de los colores, se puede comprender a simple vista cuales son los países que más variantes de malware han detectado. No hace falta ningún conocimiento por parte del usuario, simplemente viendo colores más oscuros el usuario sabe que en esa zona se han detectado más casos. Este comportamiento se debe a que este tipo de infografía es muy común en la vida diaria y por tanto es muy sencillo para los usuarios interpretar la información representada en los mapas.

Las otras dos métricas se encontrarán en la primera fila de la pantalla, ambos elementos tienen la misma importancia, por tanto, el tamaño de estas gráficas será el mismo.

Este será el diseño final para esta pantalla, las gráficas de las métricas habrá que terminar de diseñarlas, puesto que hay alguna de las cuales se desconoce el tipo de gráfico que se utilizará, así como la forma en la que se seleccionarán los datos. Pero todas estas decisiones se llevarán a cabo en secciones posteriores.

4.4.5. Panel para las métricas de métodos de distribución

El siguiente panel que se diseñó fue el que albergará las métricas asociadas a los métodos de distribución de malware. Es decir, todas las relacionadas con el medio por el que el malware detectado se propaga para la infección de los dispositivos.

La agrupación está compuesta por tres métricas, dos de ellas tratan de analizar cómo está distribuido el número de elementos (familias de malware y tipo de fichero) según el método de distribución. Por este motivo, estas dos métricas deberían estar en la misma línea.

La otra métrica es mostrar cómo se distribuyen por el mundo los casos de malware por cada método de distribución. Por tanto, esta métrica estará representada por un mapa y deberá estar sola en la línea para captar la atención de los usuarios.

A continuación, se muestra el boceto realizado para el diseño de esta pantalla:

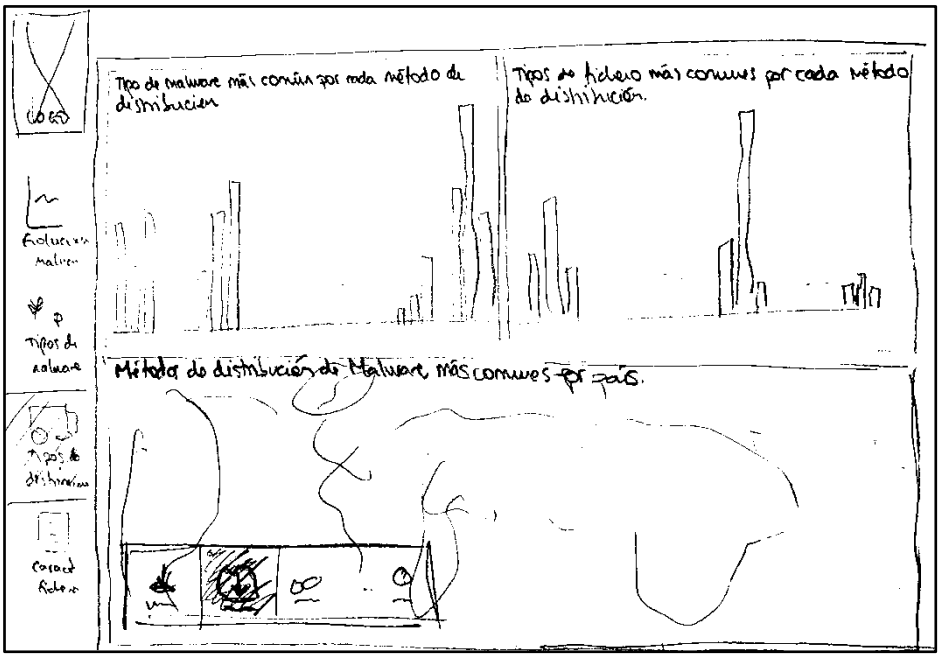


Ilustración 7. Boceto del diseño que tendrá el panel para las métricas de métodos de distribución de malware.

Tal y como se puede apreciar, el diseño de este panel es idéntico al presentado en la sección anterior. El mapa ocuparía la segunda línea completa para captar la atención del usuario. Además, este componente debería de tener algún tipo de selector que permita al usuario seleccionar que tipo de canal de distribución desea mostrar la información en el mapa. Esto se decidirá en secciones posteriores.

En la primera línea aparecerán las otras dos métricas, presumiblemente serán gráficas de barras, con una barra por cada método de distribución y agrupadas por familia de malware o por tipo de fichero, según la métrica que sea.

Al igual que se comentó en otras pantallas, este es el diseño final de la estructura de la pantalla, aunque todavía se ignora cómo será el diseño de las gráficas. Esto se abordará posteriormente en la sección de “Diseño de las visualizaciones”.

4.4.6. PANEL PARA LAS MÉTRICAS DE CARACTERÍSTICAS DE FICHEROS

Finalmente, se procedió a diseñar la última pantalla de la aplicación, la que albergará la información asociada a las características de los ficheros. Esta agrupación posee cuatro métricas diferentes. No existe ninguna métrica que tenga más importancia que otra, y tampoco se desea que ninguna métrica llame más la atención que otra.

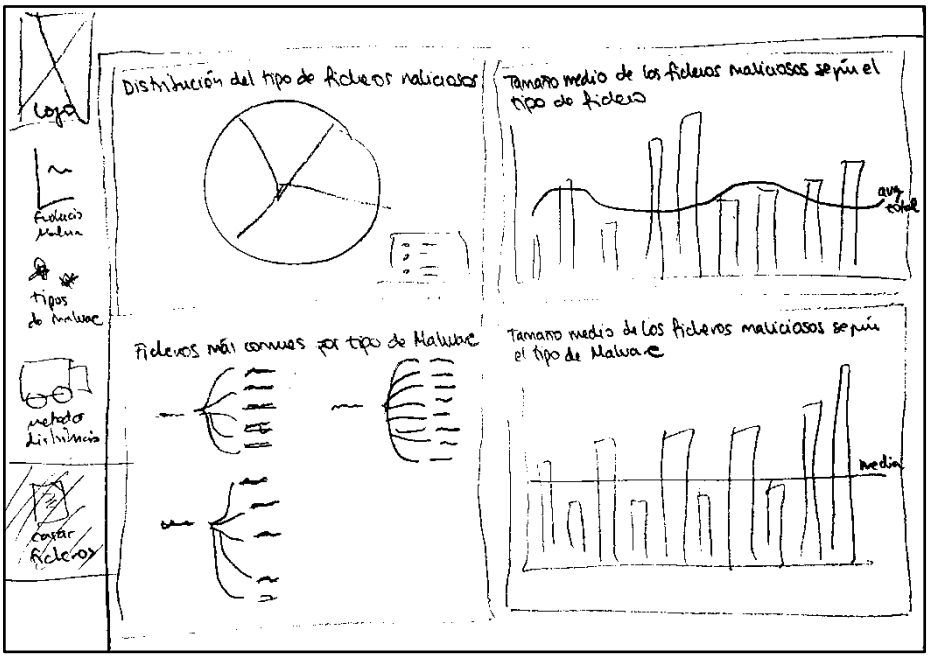


Ilustración 8. Boceto del diseño que tendrá el panel para las métricas de las características de los ficheros.

Por consiguiente, como se puede ver en el boceto correspondiente a esta pantalla, todas las gráficas referentes a las métricas de esta agrupación ocuparán el mismo espacio en la pantalla. Al ocupar el mismo espacio, y estar dividido el espacio en dos filas, el resultado será una cuadrícula de 2x2.

La información se distribuye de la siguiente forma: en la primera fila aparecerán las métricas asociadas al tipo de ficheros y en la segunda fila aparecen las métricas asociadas a la familia o tipo de malware.

Dado que las gráficas que analizan las métricas del tamaño medio de los ficheros son muy similares y quizás el usuario desee comparar los resultados que se muestren en ambas, se decidió colocarlas una encima de la otra, para que se vieran las dos en la misma columna.

Este sería el diseño final de este panel, al igual que se comentó para las otras pantallas, las métricas se diseñarán una a una en la sección “diseño de las visualizaciones” y es posible que las gráficas que se ven en estos bocetos no coincidan con el diseño final de las visualizaciones.

4.4.7. Diseño preliminar del logo

Ilustración 9. Diseño del logo de la aplicación.

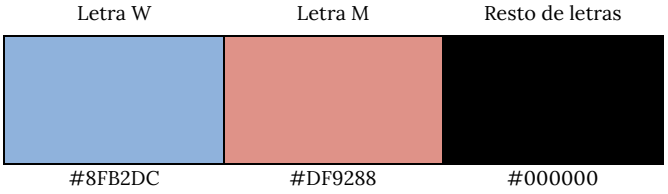
Como se comentó en secciones anteriores, se decidió crear un logo para añadirlo a la aplicación a desarrollar para darle una imagen más elegante al producto desarrollado. El diseño del logo que se realizó es el que se puede ver a continuación.

Dado que se trata de una aplicación cuyo objetivo es mostrar una serie de visualizaciones de datos de malware para analizarlos visualmente, el logo tenía que plasmar en cierta forma este cometido. Como la temática es el malware, está claro que en el logo debía de aparecer la palabra malware. Como se puede ver la M y la W se entre lazan, esto se debe a que el trazo de la unión de ambas letras simula la línea de un gráfico.

Por tanto, este logo muestra la línea de un gráfico y el nombre del malware, es decir, está representando la visualización del malware, que como se comentó es el objetivo de la aplicación.

A pesar de que el boceto del logo está en blanco y negro, en el diseño final aparecerá coloreado de tres colores diferentes. Las letras que se desean destacar son la “M” y la “W”, por este motivo, ambas se encontrarán de colores diferentes. La “M” estará coloreada en un tono rojo, mientras que la “W” la encontraremos en un color azul. La decisión de utilizar estos colores se debe a que la mayoría de los gráficos de líneas que se representan en los

programas usan estos dos colores por defecto. Por lo que, como la unión de ambas líneas simula la línea de un gráfico, se decidió usarlos. El resto de las letras, aparecerán coloreadas en negro ya que no se quiere destacar nada de ellas.



4.5. Diseño de las visualizaciones

A lo largo de este capítulo se va a presentar, para cada una de las métricas, el tipo de gráfico que se utilizará para representar los datos, así como todas las decisiones que se tomen, como colores, orientación de las series, etc.

Para realizar el diseño de las métricas se utilizó el editor online [38] que ofrece la librería "*echarts*" *[39]* que será la librería que se utilizará para la implementación del panel en el frontend, así como diferentes bocetos en papel.

4.5.1. Métricas asociadas a la evolución del malware en el tiempo

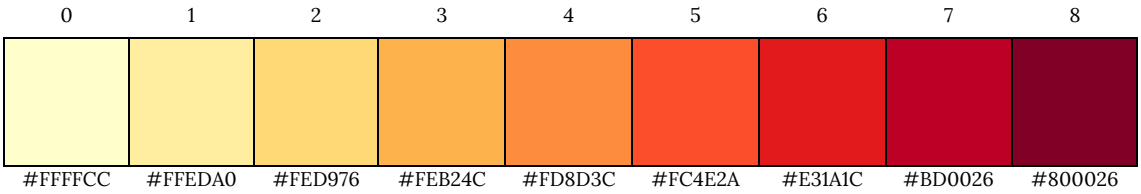
Las primeras métricas que se diseñaron fueron las relacionadas con la evolución del malware a lo largo del tiempo. Se intentó que cada uno de los diseños fuera sencillo, ya que un diseño demasiado complejo podría conllevar a conclusiones incorrectas.

4.5.1.1. Evolución de la detección del malware en el mundo

La primera métrica de este grupo representa la evolución semanal de detección de casos de malware a lo largo del mundo. Dado que esta información esta asociada a países, el gráfico que habrá que utilizar estará formado por un mapa.

Este mapa estará coloreado por una escala de colores que irá desde el amarillo hasta un granate oscuro, dependiendo de la concentración de datos de malware detectados, el color será más intenso conforme más concentración de datos haya.

La escala de colores se determinó que estaría compuesta por 9 tonos diferenciados. A continuación se presenta la escala que se utilizará para la implementación de esta métrica:



Como se puede apreciar, con esta escala de colores los usuarios podrán determinar de una forma precisa los países que han registrado una mayor cantidad de casos.

Para determinar el color que corresponde a cada país según su número de detecciones se utilizaron una serie de operaciones matemáticas. En primer lugar se utilizó una para calcular la el rango de valores que determinarán el uso de cada color. Para ello había que definir el tamaño de cada intervalo, o también denominado frontera, ya que dependiendo de este valor y de un identificador entre 0 y 8, se podrá determinar el color a utilizar.

$$T = Threshold = \frac{Score_{max}}{9}$$

Ecuación 1. Cálculo del tamaño que tendrá asignado cada color, también denominado threshold.

Dónde *threshold* representa la frontera que determinará el color a utilizar y $Score_{max}$ es el mayor número de detecciones registrado, de forma que el intervalo de colores estará determinado por los siguientes valores:

0	T	T · 2	T · 3	T · 4	T · 5	T · 6	T · 7	T · 8	S_{max}

Ahora, para calcular a que zona del intervalo se corresponde un *score* determinado, se utilizará la siguiente operación matemática:

$$id_{color} = \frac{Score_{country}}{T}$$

Ecuación 2. Cálculo del identificador de un color que corresponde al número de detecciones de un país.

Dónde el $Score_{country}$ representa el número de detecciones registrado para el país correspondiente, T es el *threshold* calculado en la ecuación 1 y el id_{color} es el identificador del color que le corresponde a la puntuación del país en cuestión.

Una vez calculado el color, ya se podrá realizar la representación correspondiente en el mapa. El resultado esperado para este tipo de métrica se puede ver a continuación:

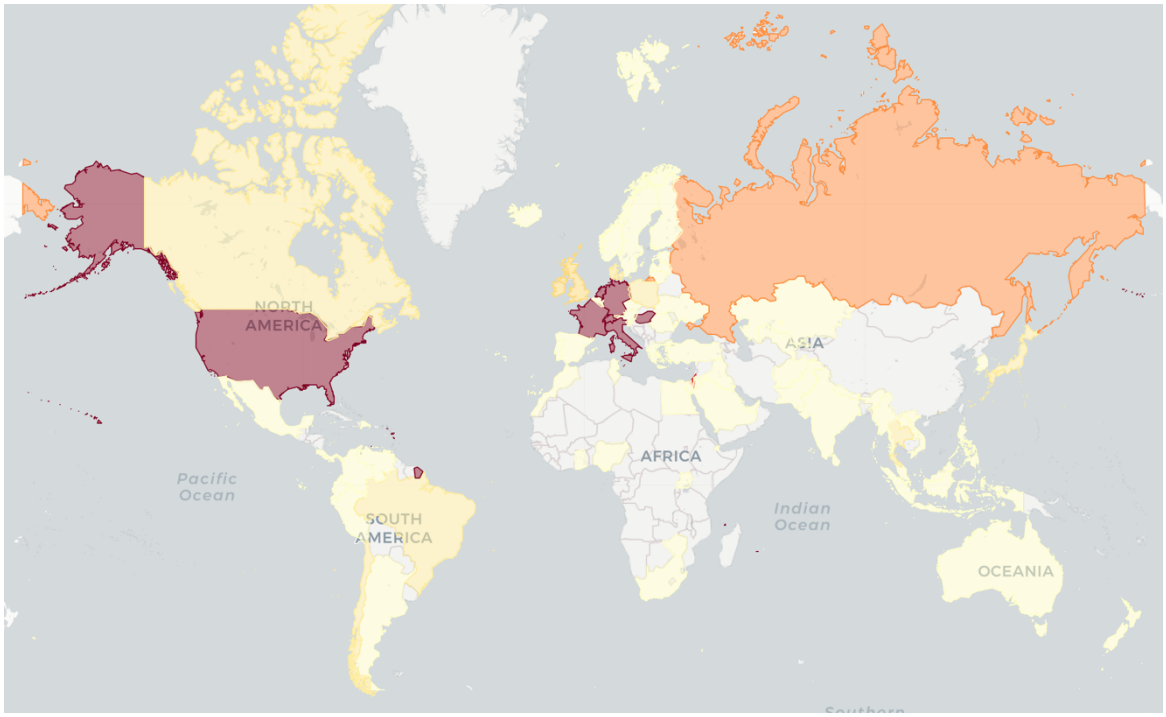


Ilustración 10. Gráfico que representará los datos asociados a la métrica de evolución del malware en el mundo.

Solamente faltaría por destacar, que al ser la evolución semanal, se tratará de mostrar el mapa animado, es decir, que vaya mostrando semana a semana como evoluciono el registro del malware. No habría que tener nada en cuenta, simplemente que el mapa irá añadiendo, cada un número determinado de segundos, los datos correspondientes a la siguiente semana. Por lo que se creará una especie de animación que mostrará la evolución semana.

4.5.1.2. Comparativa del malware detectado en un año

Esta métrica representa la comparativa entre el malware que se registró el año anterior respecto al registrado el año actual. Se determinó que la mejor forma de representar a esta métrica sería mediante un gráfico de barras.

Según los apuntes de la asignatura de visualización de datos [24], este tipo de gráfico es la mejor opción ya que están pensados para la comparación de varios elementos entre sí. Además, relacionándolo a esta métrica, es una muy buena opción para ver, de forma cuantitativa, que año ha registrado más casos.

Además, se pensó que la información estaría mejor representada de forma horizontal, ya que de esta forma se daría a entender que la barra correspondiente al año anterior no puede crecer más y que la barra del año actual, que seguirá creciendo, le faltará para llegar al registro del año anterior la anchura que tenga la gráfica.

Pero, para facilitar las conclusiones al usuario, se determinó que se añadiría un indicador al gráfico que indicará si, por el momento, se ha decretado menos o más casos que el año anterior.

A continuación, se muestra un boceto de la visualización dónde se muestran todos estos elementos.

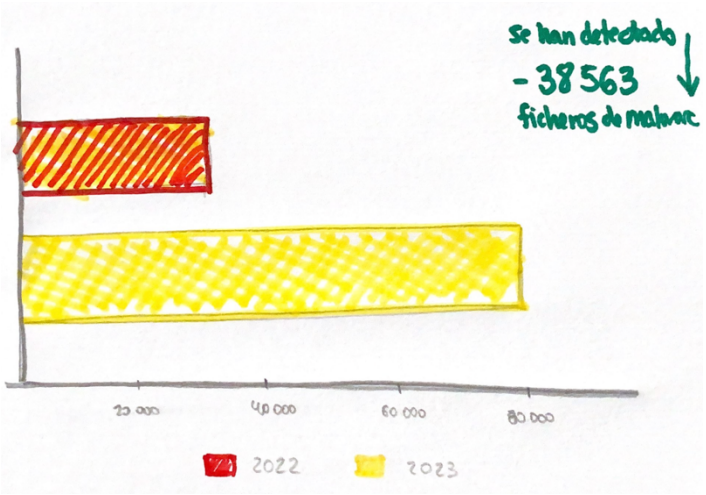
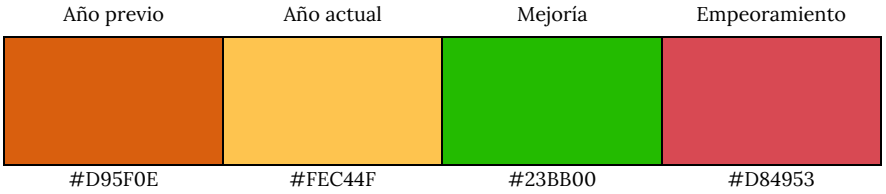


Ilustración 11. Boceto de la métrica de comparación del malware por año.

Los colores que se utilizarán para esta gráfica serán colores consistentes, que estarán relacionados con la temática del panel en general, ya que para este gráfico simplemente se desea identificar un año con

cada color, por ello se tomarán colores que estén relacionados con los colores de la aplicación.

Por tanto, los colores utilizados para el gráfico y para el indicador de la evolución serán:



4.5.1.3. Comparativa del malware detectado en un mes

Esta métrica es 100% idéntica a la comentada en la sección anterior, solamente varía el periodo por el que se seleccionan los datos, en este caso se seleccionarán los datos del mes actual y los del mes anterior.

Por tanto, para esta métrica se utilizará el mismo tipo de gráfico y los mismos colores, ya que la visualización de los datos será exactamente la misma. Esto se puede ver en el boceto contiguo, donde se puede comprobar que se utilizará el comparar.

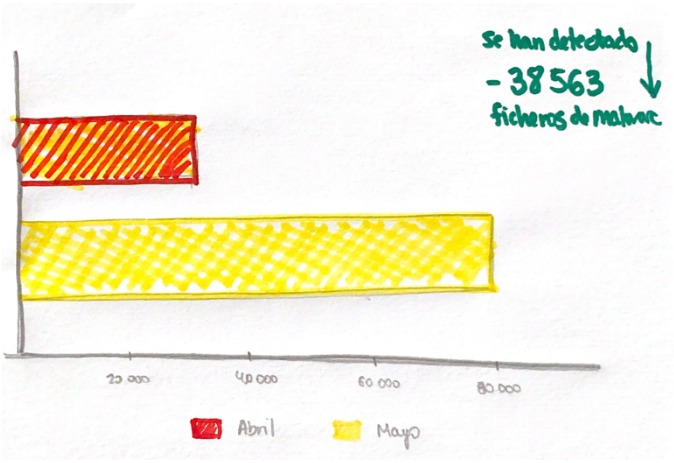


Ilustración 12. Boceto correspondiente a la comparación mensual de los datos de malware.

4.5.1.4. Evolución mensual de malware

La siguiente métrica que diseñar es la que tratará de representar la evolución mensual del malware. Dado que en el dataset se tiene información desde el año 2020, habrá que diferenciar la progresión de los datos para cada uno de los años de los que se tiene datos.

Por consiguiente, habrá que representar en el gráfico una serie por cada uno de los años disponibles. Es esta la razón que determino el tipo de gráfico a utilizar. Antes de comentar el tipo de gráfico que se utilizará se va a comentar las posibilidades que se tuvieron en cuenta y cuáles fueron las decisiones que llevaron a la gráfica final. Dado que había que analizar varias series, se tomaron dos tipos de gráfico como candidatos a representar esta métrica.

El primer candidato era un gráfico de barras apiladas, de forma que para cada mes se viera una barra dividida en tantos segmentos como años diferentes existieran en los datos. Cada uno de los segmentos representaría el valor unitario de casos de malware detectados para cada mes y año. En la siguiente ilustración se puede ver el boceto que representa este tipo de gráfico.

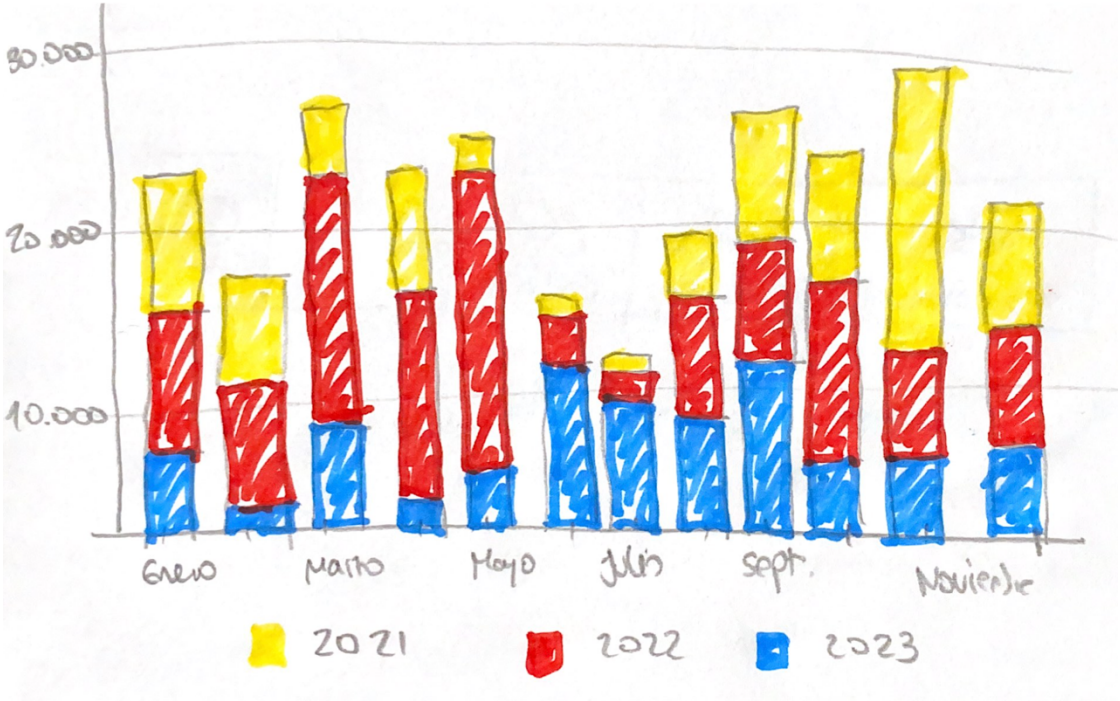


Ilustración 13. Boceto de la primera opción estudiada para representar esta métrica.

El segundo candidato era un gráfico de líneas, teniendo una línea por cada año disponible. Esto permitiría ver para cada uno de los meses los valores que posee cada una de las series. Hasta aquí se estaría representando lo mismo que con el primer candidato, pero este tipo de gráfico permite realzar otra

virtud de los datos: la evolución. Gracias a la unión entre los diferentes valores que tiene cada serie para cada mes se conseguirá ver la evolución de los datos entre los meses. Esto no se puede tan claro en el otro gráfico.

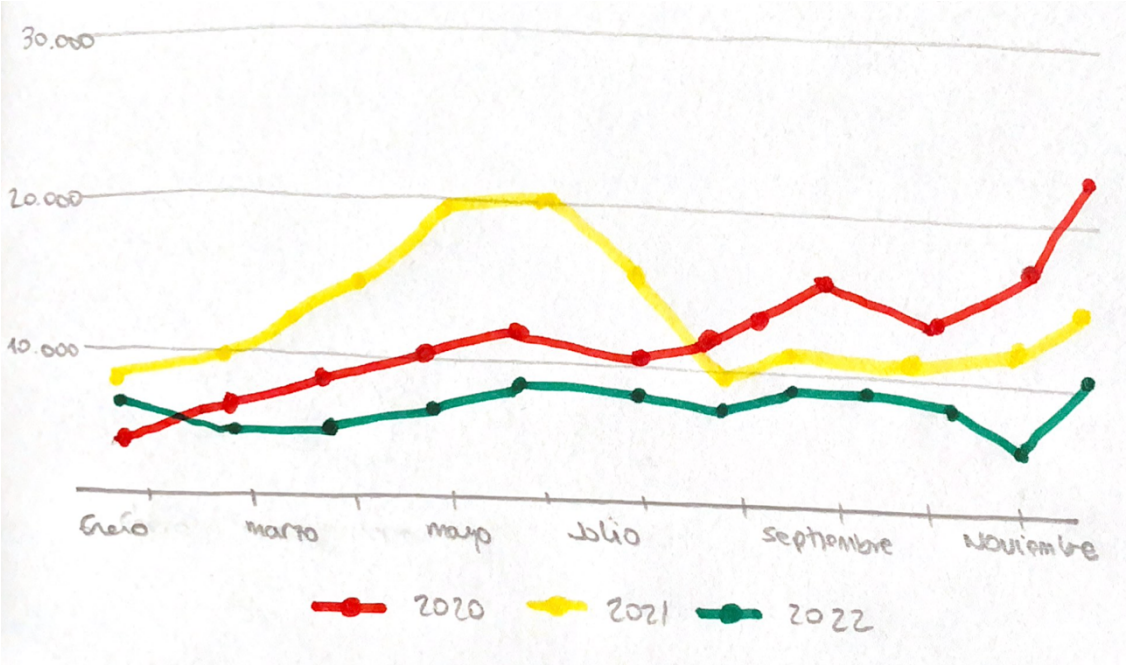
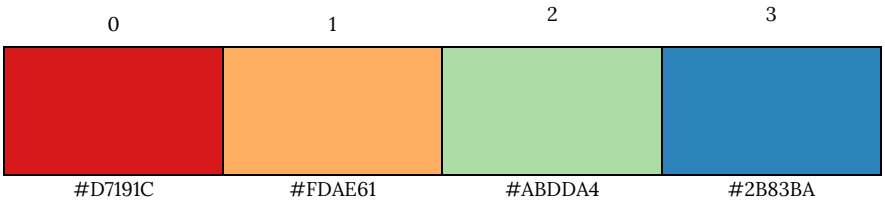


Ilustración 14. Boceto de la segunda opción estudiada para la representación de esta métrica.

Por consiguiente, se optó por utilizar la segunda opción para la implementación de dicha métrica.

La selección de los colores de las series siguió las mismas razones que se comentaron en la sección 4.5.1.2, dado que no se deseaba otra cosa que diferenciar las series, se tomaron colores que fueran en línea con la gama de colores de la aplicación.



4.5.1.5. Evolución semanal de malware

Finalmente, se llegó al diseño de la última métrica asociada a este grupo. Esta métrica consiste en mostrar la evolución semanal de la detección de malware.

Esta métrica, no trata de analizar año a año como ha cambiado, es decir, no se desea mostrar una serie para cada año, se desea mostrar toda la evolución de los datos, es decir, todo en una misma serie.

El gráfico a utilizar será uno de líneas ya que es el mejor gráfico para mostrar la evolución de los datos, además como no se desea comparar por categorías, se descartó el posible uso de un gráfico de barras.

Dado que solo existe una serie en el gráfico, se decidió sombrear del mismo color que la línea el área bajo la curva. No aporta absolutamente nada al análisis, pero se decidió hacerlo de esta forma para ganar impacto visual y para que la gráfica ganara un poco de cuerpo.

A continuación se muestra el boceto con el diseño que se utilizará para la representación de esta métrica:

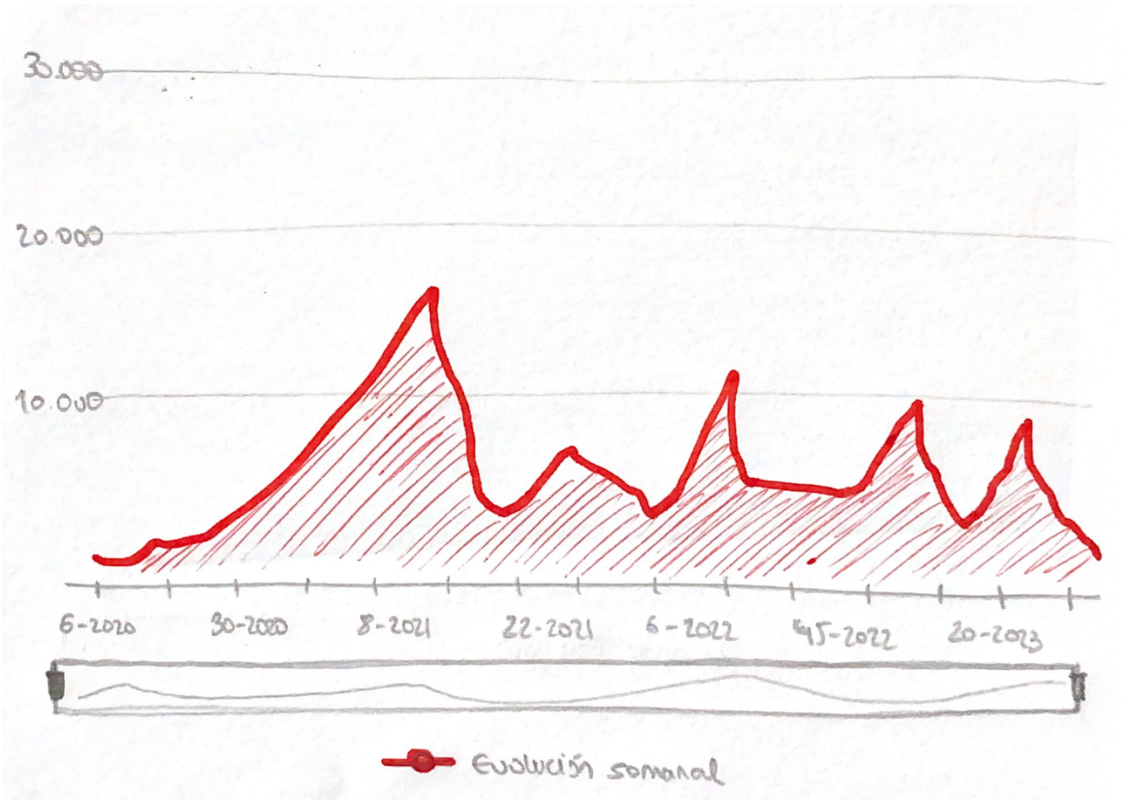


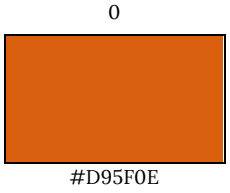
Ilustración 15. Boceto del gráfico que representará la evolución semanal del malware.

Este gráfico presentará una novedad respecto a todos los vistos anteriormente, dado que hay una gran cantidad de datos, es imposible poder ver la evolución con mucho detalle. Es por este motivo por el que se tuvo que

pesar en un mecanismo que permitiera ampliar la información del gráfico y poder ver los datos con mayor nivel de detalle.

Como se puede ver en el boceto, justo encima de la leyenda hay como un rectángulo, este mecanismo es el que ofrece la librería *echarts [39]* para ampliar los datos. Según como se interactúa con este componente, se puede ampliar o reducir el nivel de detalle de los datos del gráfico.

Finalmente, dado que solo existe una serie, solamente se utilizó un color para colorear la serie y su sombra. Al igual que ocurrió en las secciones anteriores, como no se desea destacar nada del gráfico, se utilizó un color que fuera similar a los utilizados en las secciones anteriores. A continuación se muestra un resumen de los colores utilizados.



4.5.2. Métricas asociadas a las familias de malware

El siguiente grupo de métricas que se diseñaron fueron las relacionadas con las familias o tipos de malware. Se intentó que cada uno de los diseños fuera sencillo, ya que un diseño demasiado complejo podría conllevar a conclusiones incorrectas.

4.5.2.1. Tipos de malware más comunes

Esta métrica trata de mostrar el número de detecciones que tiene cada tipo de malware, es decir, se trata de mostrar cuales son las familias más comunes.

Al igual que ocurre con otras métricas, debido a la gran cantidad de datos habrá que añadir un zoom tanto vertical como horizontal para poder ampliar los datos y ver con mayor detalle la distribución de los datos.

Para representar a esta métrica, se decidió utilizar un gráfico de barras, debido a que es un tipo de gráfico que permiten comparar varios elementos entre sí. Además en este contexto, mostrarán el número de detecciones registradas para cada familia, permitiendo así el comparar las familias para determinar cuál de ellas tiene una mayor tasa de infección.

Hay que decir que, según lo indicado en los apuntes de la asignatura de visualización de datos [24], para evitar el desorden y para mostrar de una mejor forma cuales son las más comunes se ordenarán los datos para que a la derecha de la gráfica aparezcan aquellas familias que más veces han sido detectadas.

A continuación, se muestra un boceto que muestra como quedaría el diseño de este gráfico.

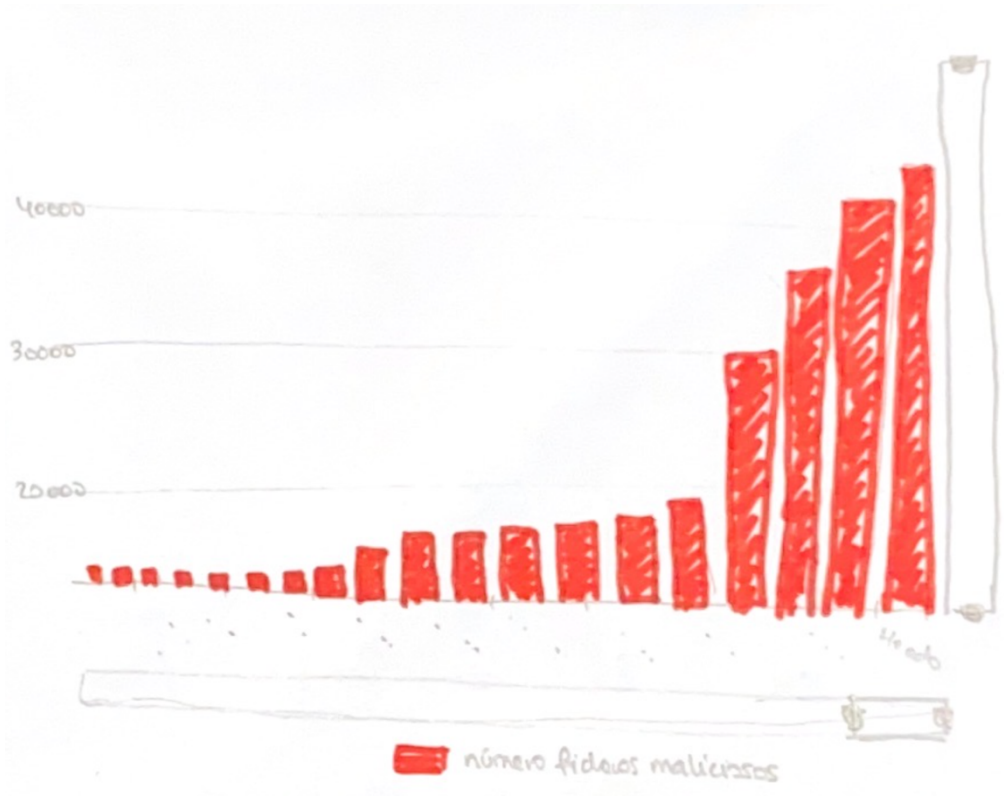
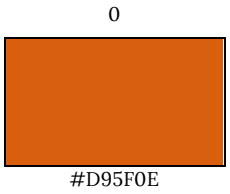


Ilustración 16. Boceto correspondiente al diseño del gráfico de esta métrica.

Los colores que se utilizarán para la representación de los datos serán de la misma gama que los que utiliza la aplicación, al igual que ha ocurrido en la mayoría de gráficos de este panel. Esto se debe a que no es necesario resaltar

ninguna variación en los datos, y como solo se analiza una única característica, no será necesario más que un único color. A continuación se muestra la gama de colores que utilizará este gráfico.



4.5.2.2. Variantes de una familia de malware vs. infecciones detectadas

Esta métrica trata de representar la comparación entre el número de variantes que posee un tipo de malware y el número de infecciones que han sido detectadas para esas familias.

Para la representación de esta comparación, se pensó utilizar un gráfico idéntico al utilizado en la sección 4.5.1.2, esto se debe a que se creyó que la información se comprendería mejor con un gráfico de barras en posición horizontal, así a la hora de comparar se podrá ver si la tasa de detecciones tiene un porcentaje superior o inferior a la de variantes de una forma más clara.

Dado el número tan grande de familias de malware, se decidió analizar solamente las diez familias de malware con mayor número de variantes. La utilidad de esta métrica es ver que la familia que tiene un mayor número de variantes no es necesariamente la que mayor número de dispositivos infecta.

A continuación se presenta un boceto del gráfico que representará a esta métrica, dónde se podrá vislumbrar el diseño final del mismo:

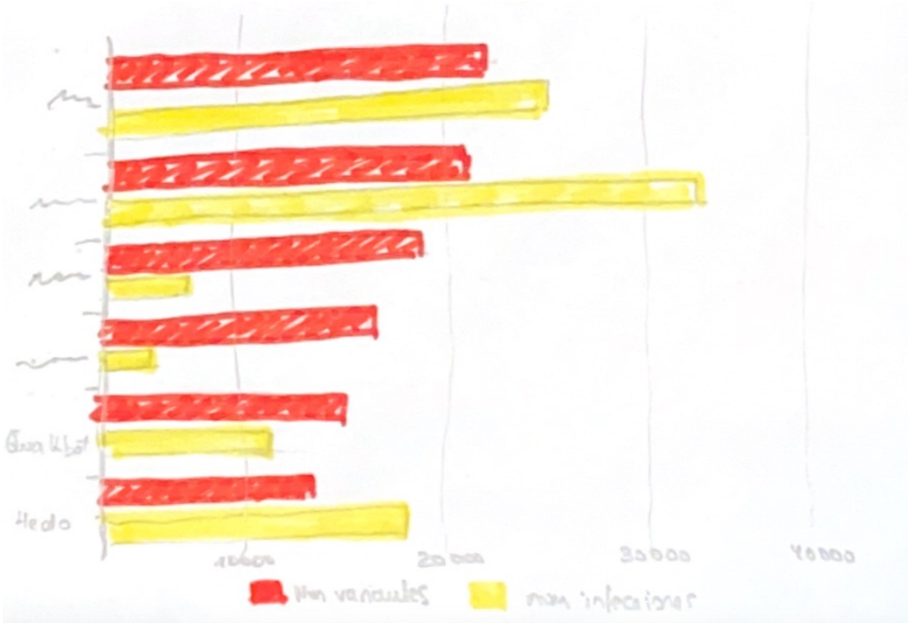
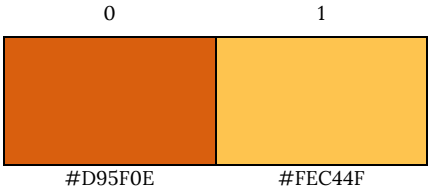


Ilustración 17. Boceto correspondiente al gráfico de la métrica.

Los colores que se utilizarán para las barras de este gráfico serán, al igual que en otras métricas, colores que queden bien con la interfaz del dashboard. Esto se debe a que no es necesario resaltar ninguna variación en los datos, y como solo se analiza un par de características, solo se necesitarán dos colores, uno que identifique a cada característica. A continuación se muestra la gama de colores que utilizará este gráfico.



4.5.2.3. Tipos de malware más comunes por país

Esta métrica tratará de representar los tipos de malware más comunes por cada país. Para representar la información se utilizará un mapa, donde los países que hayan detectado una mayor cantidad de casos de un tipo concreto de malware estarán coloreados por un tono mucho más intenso que aquellos países que hayan registrado un menor número de casos.

La escala de colores y el cálculo para determinar el color de un país determinado seguirán el mismo procedimiento que se documentó en la sección 4.5.1.1, por lo que se recomienda volver a leer dicha sección para repasar estos procedimientos.

Para esta métrica había que determinar la forma para seleccionar el tipo de malware a analizar, ya que hay una gran cantidad de familias disponibles y no existe ningún criterio para destacar a unas respecto a otras, todas son igual de importantes.

Se determinó que la mejor solución para este problema sería añadir un desplegable con el listado de todas las familias de malware disponibles y que, tras seleccionar una de ellas, se actualice el mapa mostrando la información asociada.

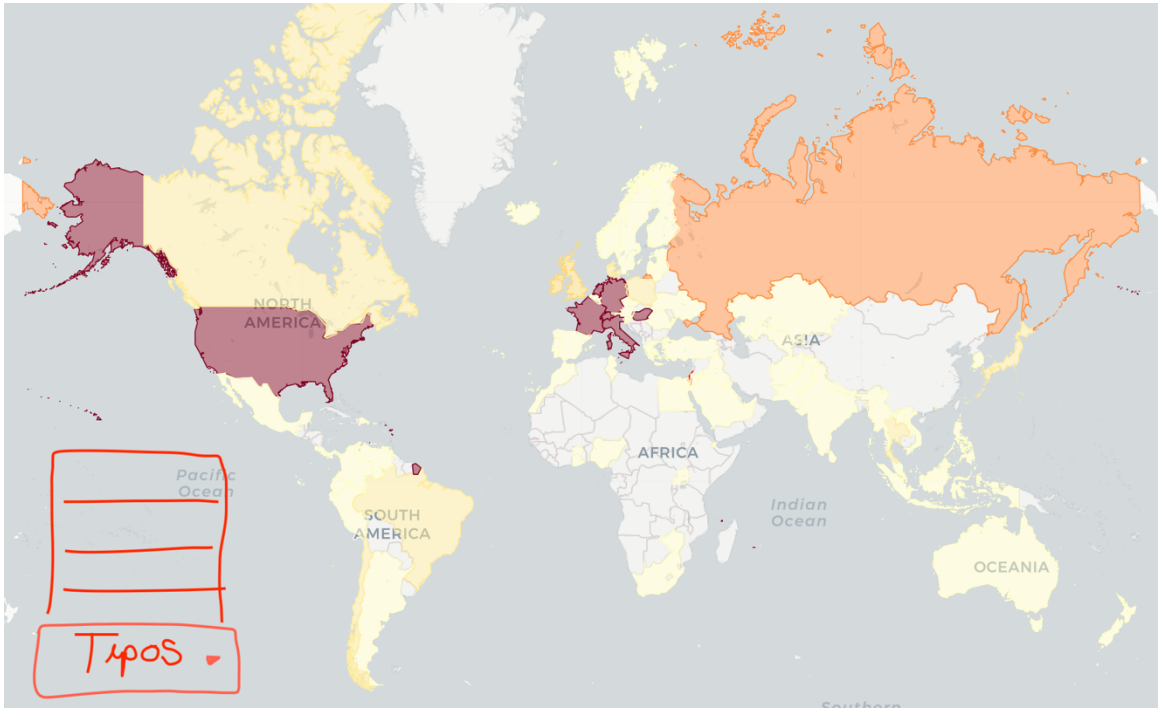


Ilustración 18. Boceto del mapa junto al selector para seleccionar el tipo de malware.

4.5.3. Métricas asociadas a los métodos de distribución del malware

El siguiente grupo de métricas que se diseñaron fueron las relacionadas con los métodos de distribución de malware. Se intentó que cada uno de los diseños fuera sencillo, ya que un diseño demasiado complejo podría conllevar a conclusiones incorrectas.

4.5.3.1. Métodos de distribución más comunes por tipo de malware

Esta métrica tratará de analizar el número de variantes que tiene asociadas cada método de distribución. Dado que existen numerosos tipos de malware, solamente se utilizó para este análisis las cinco familias que más casos tenían reportados.

A la hora de elegir el tipo de gráfico a utilizar, se dudó entre el gráfico de líneas y el gráfico de barras. Finalmente se decantó por utilizar el gráfico de barras debido a que, aunque ambos tipos de gráfico permiten ver la evolución de los datos, el gráfico de barras permite a su vez cuantificar de una mejor forma el número de registros que tiene una cierta categoría. Es por este motivo por el que se decidió utilizar un gráfico de barras para la implementación de dicha funcionalidad.

Hay que destacar que al igual que se hizo en secciones anteriores, habrá que aportar una serie de herramientas para poder ampliar los datos del gráfico y poder analizar la métrica sin tener ningún tipo de problemas. En este caso, el zoom estaría colocado mediante un segmento a la derecha del gráfico. Redimensionando y moviendo este segmento se podrá ampliar o reducir los datos de la gráfica.

A continuación, se muestra un boceto del gráfico en cuestión, donde se puede ver cada una de las funcionalidades aquí descritas.

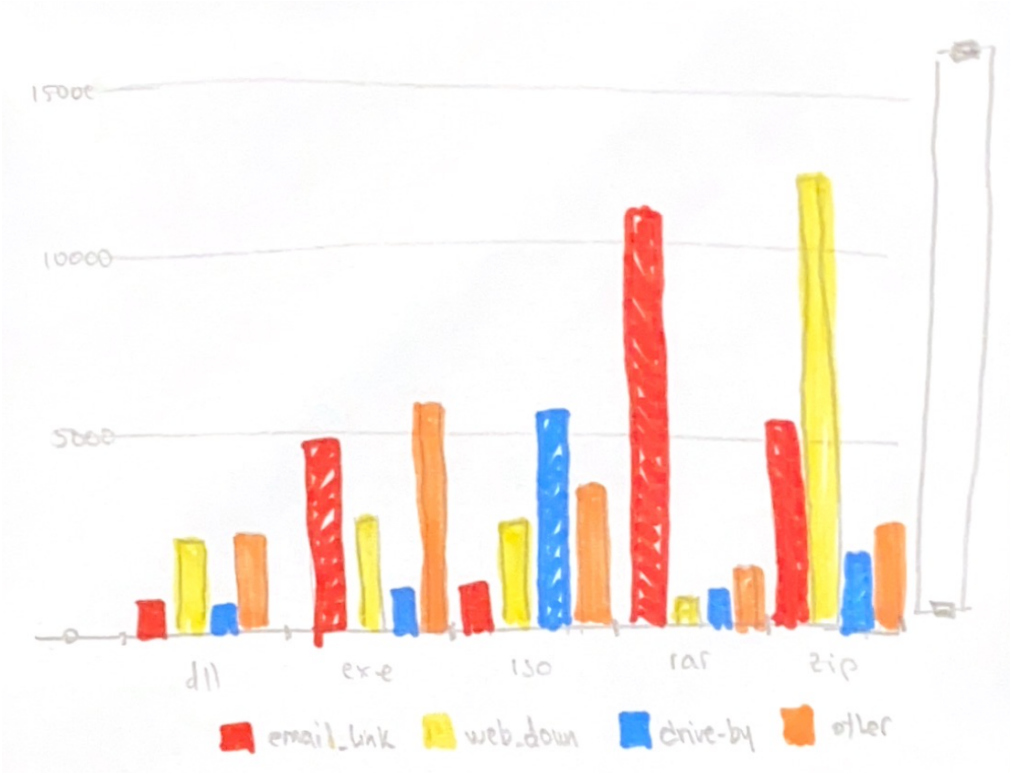


Ilustración 19. Gráfico diseñado para esta métrica.

Finalmente, se presenta la gama de colores utilizada para la implementación de este gráfico. Del mismo modo que ha ocurrido en otras secciones, la elección de los colores no está basada en las labores de destacar ni de comparar los datos. Dado que solamente se necesita identificar un color con una serie para conocerla a simple vista, se decidió utilizar la gama cromática de la aplicación.

A continuación se muestra un resumen de los colores utilizados.

0	1	2	3	4	5	6
#D73027	#F46D43	#FDAE61	#FEE090	#74ADD1	#4575B4	#244167

4.5.3.2. Métodos de distribución más comunes por tipo de fichero

Está métrica es idéntica a la presentada en la sección anterior, ya que se tiene el mismo tipo de información de los datos, simplemente cambia la agrupación de los datos, en lugar de agruparse por el tipo o familia de malware, ahora se agrupa por el tipo de fichero malicioso.

Dado que los datos, estructuralmente hablando, son iguales se mantendrá el tipo de gráfico y la gama de colores utilizada en la sección 4.5.3.1. Solamente cambiará el eje x de la gráfica, ya que en este eje se seguirá manteniendo el nombre de la categoría, pero ahora la categoría ha cambiado al tipo de fichero malicioso.

Un aspecto importante a mencionar, al igual que en la sección anterior solamente se realizaba el análisis sobre las 5 familias que más fue detectadas, ahora se realizará el análisis sobre los 5 tipos de ficheros que más malware fue detectado.

A continuación, se muestra el boceto del gráfico que representará esta métrica.

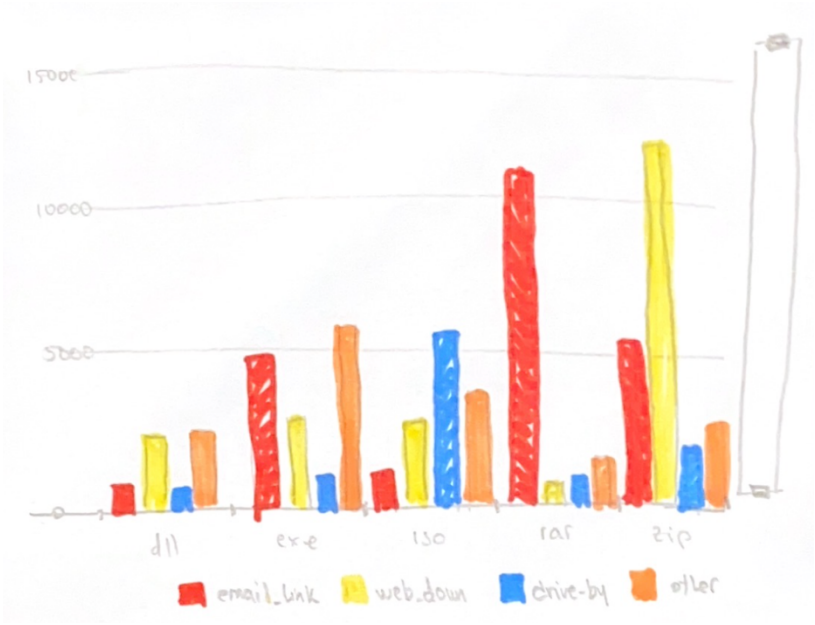


Ilustración 20. Gráfico diseñado para esta métrica.

4.5.3.3. Métodos de distribución de malware más comunes por país

Esta métrica tratará de representar los métodos de distribución más comunes por cada país. Para representar la información se utilizará un mapa, donde los países que hayan detectado una mayor cantidad de casos de un medio de distribución concreto estarán coloreados por un tono mucho más intenso que aquellos países que hayan registrado un menor número de casos.

La escala de colores y el cálculo para determinar el color de un país determinado seguirán el mismo procedimiento que se documentó en la sección 4.5.1.1, por lo que se recomienda volver a leer dicha sección para repasar estos procedimientos.

Para esta métrica había que determinar la forma para seleccionar el medio de distribución a analizar, ya que en este caso hay poca cantidad de criterios de selección, por lo que se podrá diseñar una mejor forma de selección que un desplegable (como se comentó en la sección 4.5.2.3).

Por consiguiente, se determinó que la mejor forma para seleccionar los datos podría ser una especie de selector que mostrara en todo momento todas las posibles opciones que están disponibles.

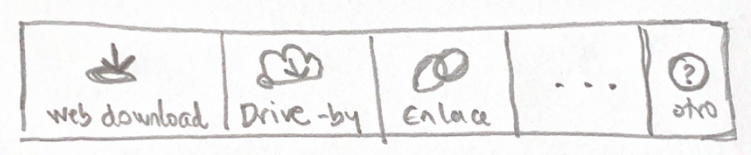


Ilustración 21. Selector para indicar el método de distribución.

Este selector iría colocado abajo a la izquierda del mapa, ya que esta zona es la ideal para que sea útil para el usuario y para que no moleste y dificulte la visualización de los datos. A continuación se puede ver un boceto de como quedaría la visualización de esta métrica:

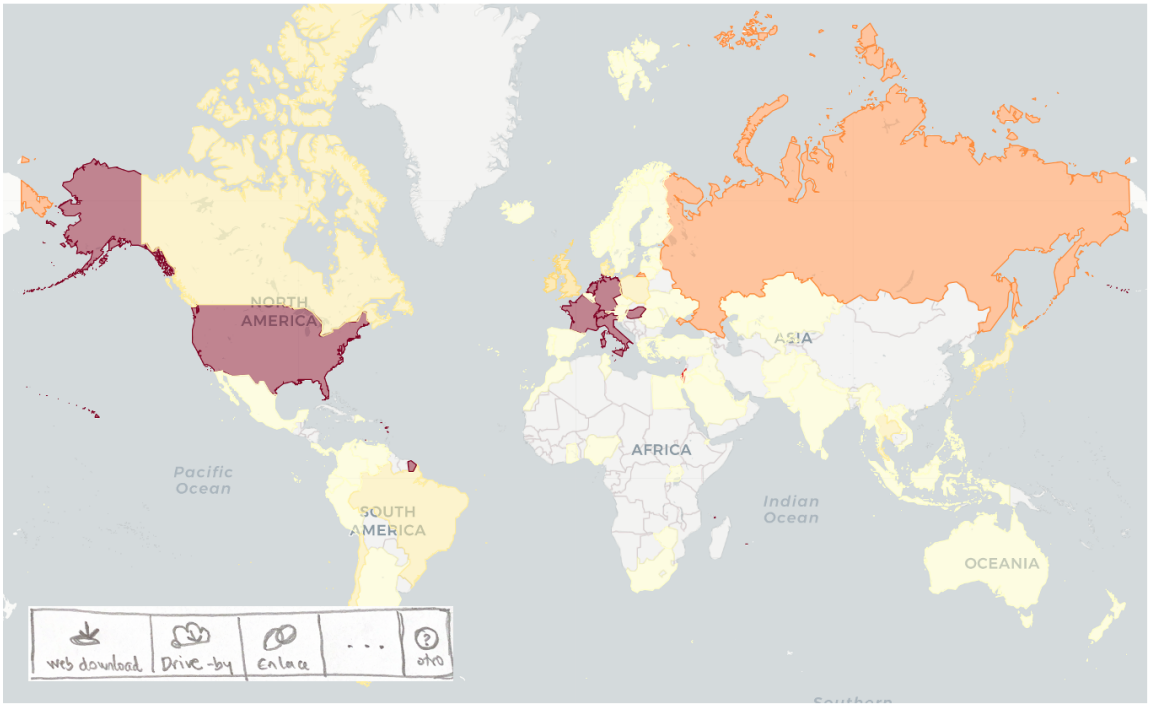


Ilustración 22. Boceto para la métrica de los métodos de distribución más comunes por país.

4.5.4. MÉTRICAS ASOCIADAS A LAS CARACTERÍSTICAS DE LOS FICHEROS DE MALWARE

Finalmente, el último grupo de métricas que se diseñaron fueron las relacionadas con las características de los ficheros maliciosos. Se intentó que cada uno de los diseños fuera sencillo, ya que un diseño demasiado complejo podría conllevar a conclusiones incorrectas.

4.5.4.1. DISTRIBUCIÓN DEL TIPO DE FICHEROS MALICIOSOS

Esta métrica trata de mostrar cómo están distribuidos los datos según el tipo de fichero malicioso, es decir, se trata de mostrar la cantidad de registros que tiene asociado cada tipo de fichero malicioso.

Se decidió utilizar para representar esta métrica un gráfico circular, ya que existe un número determinado de categorías y la suma de los porcentajes de registros, que tiene asociado cada tipo de fichero malicioso, suman el 100%.

Pero para mejorar la representación se decidió mostrar los nueve tipos de fichero que mayor cantidad de registros poseen y el resto de categorías se agruparon en una sola bajo el pseudónimo de "*otros*". De esta forma se simplificaría el gráfico y se mejorarán los resultados.

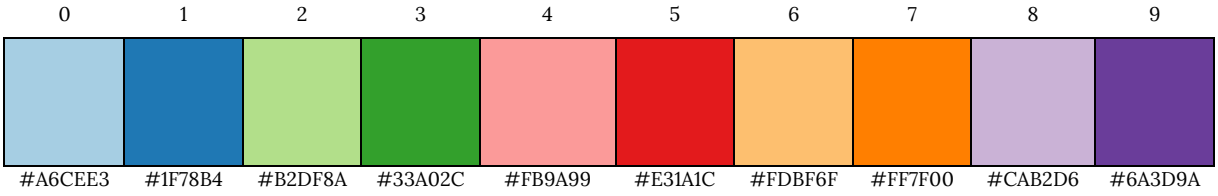
Otra casuística que se tuvo en cuenta fue la de ordenar los registros, de forma que se mejorara la visualización en el gráfico, de esta forma los sectores estarán ordenados acorde a su tamaño y los usuarios tendrán una mejor experiencia para poder conocer el orden de tipos de fichero que mayor cantidad de malware han registrado.

A continuación, se va a mostrar un boceto del gráfico resultante para esta métrica, donde se podrán visualizar todas las decisiones comentadas.

Ilustración 23. Gráfico resultante de la fase de diseño para esta métrica.

Finalmente hay que destacar la selección de colores que se utilizarán para este gráfico. Son nueve categorías diferentes, por tanto habrá que seleccionar tantos colores como categorías hubiera. Se utilizarán los mismos

colores que se utilizaron en otras gráficas ya diseñadas, pero habrá que seleccionar otros colores diferentes hasta llegar a los diez necesarios. A continuación se muestran los colores seleccionados.



4.5.4.2. Tamaño medio de los ficheros maliciosos según el tipo de fichero

Esta métrica trata de representar el tamaño medio de los ficheros maliciosos según el tipo de fichero utilizado. Es decir, tratará de calcular la media del tamaño de los ficheros agrupándolos por el tipo del mismo.

El gráfico más adecuado para representar esta información será un gráfico de barras, ya que no se desea representar una evolución entre los datos, simplemente se desea cuantificar el tamaño medio para cada tipo de fichero. Por consiguiente, el tamaño de las barras representará el tamaño de los ficheros.

Dado la gran cantidad de datos que contendrá el gráfico, habrá que añadir la posibilidad de hacer zoom sobre la gráfica para poder ver mejor los datos representados. El zoom tendrá que estar habilitado en ambos ejes, para poder ver más detalle en el eje x como en el y.

Además del tamaño medio, hay que representar la media absoluta del tamaño de fichero, de forma que se debe permitir el comparar para cada tipo de fichero si el tamaño medio está por encima o por debajo del tamaño medio absoluto. Para representar esta información en el gráfico, habrá que añadir una línea horizontal para poder realizar la comparación comentada.

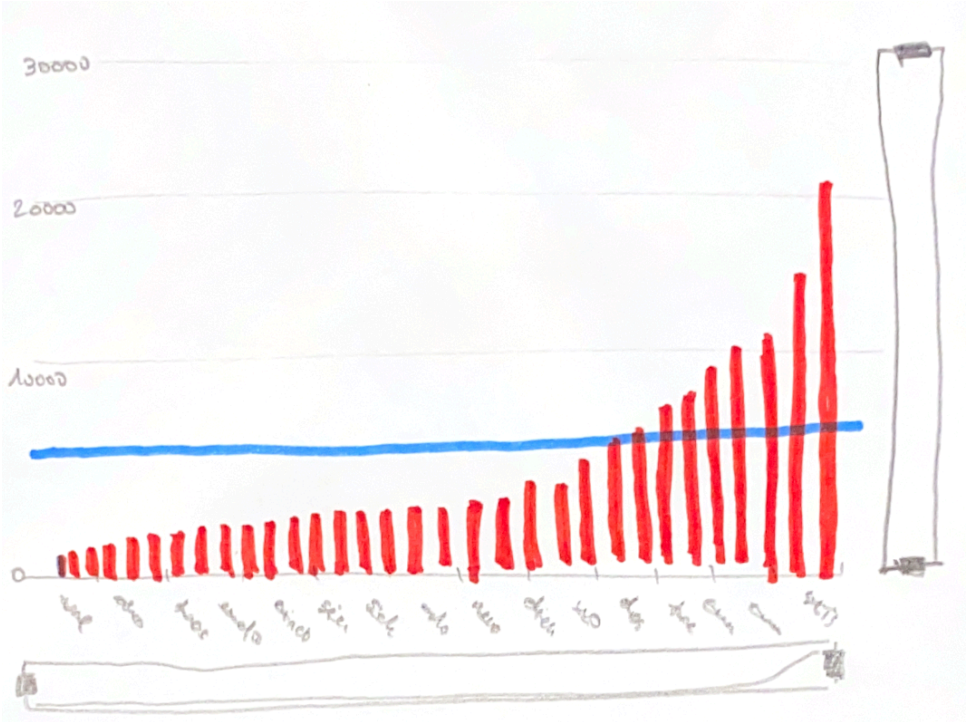
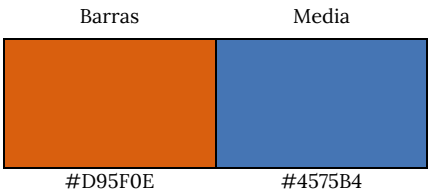


Ilustración 24. Gráfico diseñado para la representación visual de esta métrica.

A continuación se muestra un boceto con el diseño final del gráfico, dónde se podrá visualizar todas las decisiones mencionadas anteriormente.

En cuanto a los colores que se utilizarán para pintar a los diferentes elementos, se seleccionará un color común para todas las barras del gráfico, ya que no hay que destacar nada entre los datos, por tanto solo deberá de tener un solo color. También será necesario un color para la línea que representará la media absoluta, este color tendrá que contrastar bien con el que se seleccione para las barras, ya que si tienen un tono similar no se podría vislumbrar bien ambas series y no se podrá realizar la comparación descrita anteriormente de forma correcta. A continuación se muestran los colores seleccionados para colorear las series.



4.5.4.3. Tamaño medio de los ficheros maliciosos según el tipo de malware

Esta métrica es idéntica a la representada en la sección anterior, lo único que cambia es el criterio de agrupación para calcular el tamaño medio de los ficheros. Pero tanto el tipo de gráfico como los colores serán los mismos que los explicados en la sección anterior.

El tipo de gráfico a utilizar será un gráfico de barras, utilizando las barras para mostrar el tamaño medio de los ficheros y se añadirá una línea horizontal que representará la media global.

A continuación se muestra el boceto que representa el diseño final del gráfico de esta métrica.

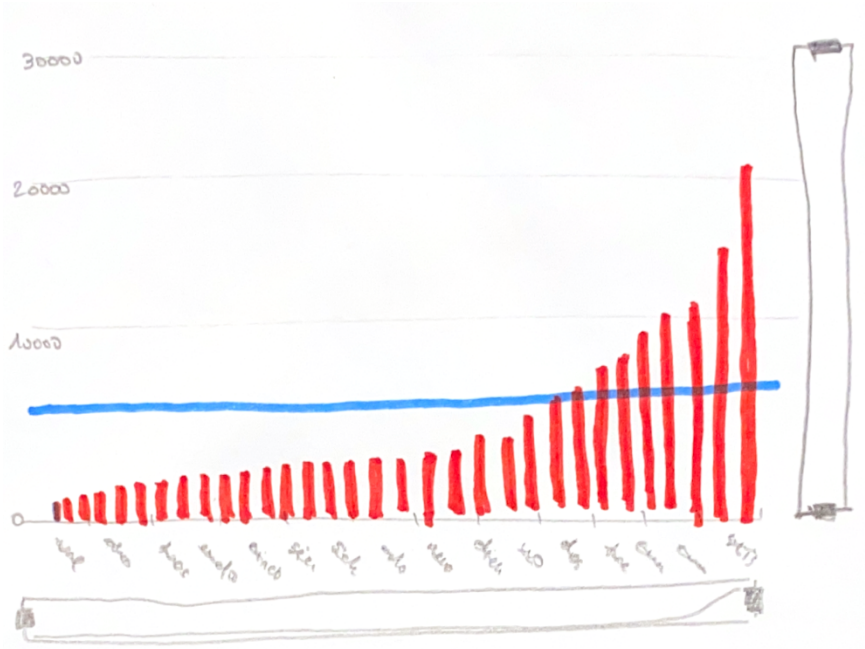
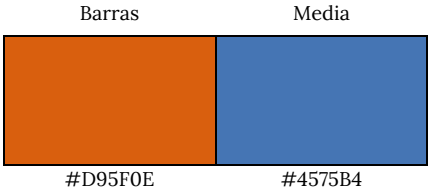


Ilustración 25. Gráfica diseñada para la representación visual de esta métrica.

Finalmente, por las mismas razones que se expresaron en la sección 4.5.4.2, los colores que se utilizarán para la representación de este gráfico se pueden ver a continuación.



4.5.4.4. Tipos de malware más comunes para cada tipo de fichero

Esta métrica es totalmente diferente a las mostradas hasta ahora, ya que no trata de representar para una, o varias categorías un resultado numérico (como puede ser una media, un conteo, etc.). Aquí se trata de representar una relación entre los datos, esta relación es mostrar los tipos de ficheros más comunes para los cinco tipos de malware que más registros tienen asociados.

Dada la gran cantidad de tipos de ficheros que se tienen, se decidió mostrar solamente los diez tipos más comunes para cada familia de malware.

Fue una difícil decisión la de seleccionar el elemento visual que representará esta métrica. Ya que al ser tan diferente a las métricas anteriores, no se tenía claro que método utilizar. Al final, revisando los apuntes de la asignatura de visualización de datos [24], se determinó que la mejor opción sería utilizar un grafo, ya que lo que se trata de mostrar son las relaciones entre las instancias, por tanto este sería el mejor método de representación.

Se utilizará un grafo diferente para cada tipo de fichero malicioso a analizar, de esta forma quedará más limpia y mejor organizada la información.

A continuación se muestra un boceto de cómo quedará el diseño final de la representación de esta métrica.

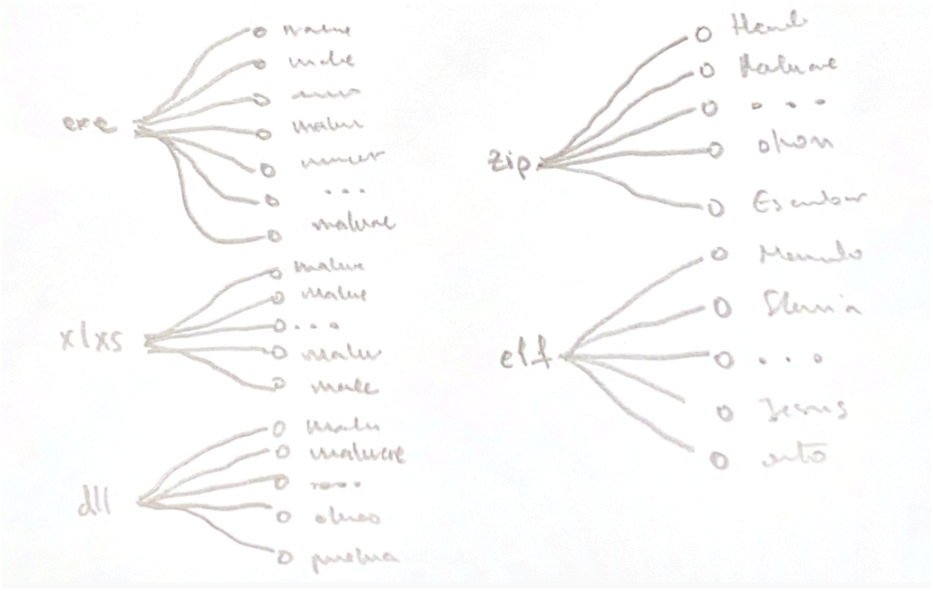


Ilustración 26. Grafo que representará la información asociada a esta métrica.

Dado que el gráfico utilizado es un grafo, no será necesario establecer ningún color para utilizar, ya que en este tipo de representaciones no se tienen que destacar elementos, o colorear líneas, barras, sectores, etc.

5. IMPLEMENTACIÓN

La implementación de software es un proceso esencial en el desarrollo de aplicaciones y sistemas informáticos. Consiste en llevar a cabo todas las actividades necesarias para que un software diseñado pueda ejecutarse y funcionar de manera efectiva en un entorno específico. Esta etapa es crucial para convertir la idea y el diseño conceptual de un programa en una realidad funcional.

A lo largo de la presente sección se van a detallar las decisiones llevadas a cabo durante la fase de implementación, así como se explicará la metodología de desarrollo utilizada tanto para la parte backend como para la parte frontend.

Como se comentó en otras secciones, este proyecto no es simplemente una visualización de datos, no consiste en utilizar un dataset existente y pintar los gráficos asociados a las métricas. Este proyecto es un proyecto de ingeniería y ciencia de datos y como consecuencia de ello, se ha desarrollado toda una infraestructura software, poniendo en práctica todos los conocimientos adquiridos durante el estudio del máster.

Esta infraestructura consta de una parte de base de datos, que se explicó en la sección 3, una parte backend y una parte frontend.

La parte backend [40] que es la parte responsable de la lógica y la gestión de datos que ocurren detrás de escena. Es la columna vertebral del sistema y se encarga de procesar las solicitudes del usuario, interactuar con las bases de datos y proporcionar los datos necesarios al frontend. Su principal función es garantizar que todas las operaciones y funcionalidades del software sean ejecutadas de manera eficiente y segura.

El backend también se encarga de la validación y procesamiento de los datos enviados desde el frontend, asegurándose de que cumplan con los requisitos y restricciones establecidos. Además, se encarga de gestionar las solicitudes de los usuarios, redirigiéndolas a los servicios y recursos apropiados.

Otra función clave del backend es proporcionar interfaces de programación de aplicaciones (API) para permitir la comunicación entre el frontend y el backend, así como con otros servicios o sistemas externos. Estas API definen los puntos de acceso y los formatos de intercambio de datos, permitiendo que las aplicaciones se integren entre sí de manera efectiva.

La parte frontend [41], [42] es la interfaz con la que los usuarios interactúan directamente. Es la parte visible y tangible del sistema, y su función principal es presentar la información de manera visualmente atractiva y facilitar la interacción del usuario con el software.

La función principal del frontend es proporcionar una experiencia de usuario agradable y efectiva. Esto implica diseñar interfaces intuitivas, fáciles de usar y visualmente atractivas. Además, se encarga de garantizar la capacidad de respuesta y la compatibilidad con diferentes dispositivos y navegadores.

El frontend también se encarga de la validación de los datos ingresados por el usuario y de proporcionar retroalimentación visual inmediata, como mensajes de error o confirmaciones, para mejorar la usabilidad y la experiencia general.

Otra función importante del frontend es consumir los servicios y datos proporcionados por el backend. Utilizando las interfaces de programación de aplicaciones (API) expuestas por el backend, el frontend solicita y muestra la información relevante al usuario, actualizando y sincronizando los datos en tiempo real, cuando sea necesario.

A todo ello se suma el trabajo de análisis e investigación realizado para poder llevar a cabo el desarrollo de la aplicación, ya que para poder desarrollar la lógica de negocio de una forma correcta y coherente hace falta comprender a la perfección el problema a resolver.

A continuación, se va a comenzar por explicar la metodología de desarrollo, ya que antes de comenzar a desarrollar código se establecieron una serie de

directrices sobre cómo proceder a desarrollar el código de la aplicación para que el resultado tuviera la mayor calidad posible.

5.1. METODOLOGÍA DE TRABAJO MEDIANTE TDD

A lo largo de esta sección se tratará de describir la metodología utilizada para la elaboración de los diferentes casos de uso [43] dentro de la arquitectura hexagonal que compondrá este proyecto. El desarrollo de código se realizará mediante *Test-Driven Development (TDD)*.

TDD [44], [45] es una práctica de programación que consiste en desarrollar el código siguiendo el siguiente orden:

I. Desarrollo de los test unitarios.
II. Desarrollo del mínimo código fuente que permita que el test pase correctamente.
III. Refactorización del código desarrollado.

Con esta práctica se consigue que el código resultante sea **más robusto, seguro y mantenible.** Además, el código final es **un código muy limpio**, ya que se ha escrito el mínimo código necesario para el funcionamiento de la funcionalidad requerida.

Pero antes de pensar en el desarrollo y de cómo aplicar TDD, hay que realizar el análisis de la funcionalidad a desarrollar.

5.1.1. ANÁLISIS DE LOS CASOS DE USO

El análisis es, probablemente, el paso más importante de todos, ya que un mal análisis puede dar como resultado un código inmantenible.

En esta fase hay que tener en cuenta lo siguiente:

1. **Inputs**: entradas que tendrá ese caso de uso y el tipo de datos.
2. **Outputs**: salidas que tendrá ese caso de uso y el tipo de datos.

3. **Validación de los inputs:** habrá que analizar que restricciones tendrán los datos de entrada:
   - ¿Campo obligatorio?
   - ¿String no vacío?
   - ¿Array?
4. **Validación de la integridad de los datos:** habrá que analizar si se cumplen ciertas premisas que cumplan la integridad de los datos:
   - ¿Existe la entidad con el identificador X?
   - ¿La entidad Y pertenece a la entidad U?
   - ¿La entidad A puede tener más de 8 entidades T asociadas?

Una vez que se ha analizado todos los puntos anteriores, ya se estaría preparado para la elaboración de una lista dónde se definirán los **requisitos del caso de uso.**

5.1.2. REQUISITOS DE UN CASO DE USO

La lista de requisitos de un caso de uso es un documento que materializa los aspectos que se han analizado en la fase anterior y **será el punto de partida** a partir del cual **se definirán los test unitarios del caso de uso.**

Un ejemplo de formato de este documento será el siguiente:

```
Create job by userId
====================
1. Inputs:
  - userId: string
  - job: integer
2. Outputs:
  - Job
3. Validaciones del input:
  - userId: Texto, Obligatorio
    - Si userId = null ==> Error
    - Si userId.length <= 0 ==> Error
  - job: Integer, No obligatorio, mayor de 0
    - Si job != Integer ==> Error
    - Si job <= 0 ==> Error
4. Integridad de los datos:
  - Debe existir un usuario con id = userId ==> Sino existe ==> Error
```

Código 31. Ejemplo del formato que tendría un documento de requisitos de un caso de uso.

Como se puede apreciar, el documento de requisitos es un documento muy simple pero que permite ver de una forma clara y concisa todas restricciones y errores que puede devolver el caso de uso en cuestión.

Tras revisar que el caso de uso esté correctamente plasmado en el documento de requisitos, hay que **diseñar los test unitarios** que se tienen que desarrollar para validar que el caso de uso funciona perfectamente.

5.1.3. Diseño de los test unitarios

Es imprescindible realizar un diseño de los test unitarios y no lanzarse a implementarlos directamente. Es una muy mala práctica implementar los test sin haberlos diseñado previamente, ya que podría tener como consecuencia:

- Test ilegibles.
- Test con falsos resultados.
- Testear sentencias que no tienen sentido.

Es importante tener un código con unos buenos test asociados, ya que esto permitirá conocer si la aplicación sigue funcionando correctamente, a pesar de que se realicen algunos cambios sobre el código.

Pero… ¿Cómo se hace un correcto diseño de los test unitarios? Esta pregunta tiene una sencilla respuesta si se han seguido los pasos anteriores. Lo correcto es **transformar el documento de requisitos en una checklist**, donde se plasme cada una de las sentencias a testear.

Esta checklist permitirá ir marcando cada uno de los test conforme se vayan desarrollando y validando, de esta forma se conseguirá que el caso de uso este completamente testeado. El formato de la checklist deberá ser el siguiente:

```
[TDD] Checklist: Create job by userId
====================================
- [ ] Validación del Input:
  - userId:
    - [ ] Comprobar que es un campo obligatorio
      => FieldValidationError
    - [ ] Comprobar que es string no vacío
      => FieldValidationError
  - job:
    - [ ] Comprobar que es un int mayor o igual que 0
      => FieldValidationError
- [ ] Comprobar que existe un usuario con dicho id
  => UserDoesNotExistError
- [ ] Crear (persistir) el Job si existe el usuario
- [ ] Devolver el Job tras haber sido creado
```

Código 32. Ejemplo de formato de la checklist con los test que habría que desarrollar para este ejemplo.

Como se puede ver, en esta checklist se concretan los errores que lanzará si se incumple alguna de las premisas. **Es importante el crear errores que sean representativos**, ya que si tienen nombres ambiguos podría conllevar a otros problemas (confusión, no identificar correctamente dónde hay un problema...).

Desde este punto, pasar la checklist a un fichero de test unitarios es muy sencillo, ya que se tiene muy claro que test hay que desarrollar y que hay que validar en cada test.

Un aspecto muy importante a tener en cuenta es, que si en un mismo test se está comprobando **más de una responsabilidad** es un claro indicador de que ese test está **mal diseñado.** Cada test tiene que comprobar una única tarea y tener una descripción clara, concreta y concisa.

A continuación, se puede ver cómo quedaría el código del fichero de test de la checklist anterior:

```
describe('Create job by userId', () => {
  describe('Input validation', () => {
    describe('user id', () => {
      it.skip('should throw FieldValidationError when not provided');
      it.skip('should throw FieldValidationError when is not a nonempty string');
    });

    describe('job', () => {
      it.skip('should throw FieldValidationError when is not an int greater than 0');
    });
  });
```

```
  describe('when the user does not exist', () => {
    it.skip('should throw a UserDoesNotExistError');
  });

  describe('when the user exists', () => {
    it.skip('should create the job');
    it.skip('should return the job created');
  });
});
```

Código 33. Ejemplo de los test resultantes de la checklist del ejemplo anterior.

Como se puede ver, el código resultante es muy similar a lo que se puede leer en la checklist. Además, siguiendo esa estructura jerárquica se tiene como resultado unos test muy simples e intuitivos.

Es importante destacar:

I. Hay que escribir todos los test de la checklist vacíos, poniendo el flag "*skip*" se consigue que ese test unitario no se ejecute, de esta forma se irán implementando de uno en uno sin que den errores los test que no se han implementado todavía.

II. No hay que desarrollar todos a la vez. Es imprescindible desarrollarlos de uno en uno. Esto se comentará más adelante.

Ahora, una vez que se tienen los test unitarios diseñados, se puede proceder a la implementación del caso de uso mediante TDD.

5.1.4. Implementación del caso de uso

Finalmente, hay que realizar la implementación del código del caso de uso mediante TDD. Como se mencionó al principio del documento, el fin de utilizar TDD es conseguir que el código resultante sea **más robusto, seguro, limpio y mantenible**.

Realizar la implementación de un caso de uso es muy sencillo siempre que se sigan estos pasos:

I. Se **selecciona un único test unitario** de los desarrollados en el paso anterior.

II. Se **implementa el test**, es decir, se añaden las validaciones necesarias dentro del test unitario que permitan comprobar que el caso de uso se comportará correctamente si ocurre la casuística que comprueba el test.

III. **Se comprueba que el test FALLA.** Como no se ha añadido el código necesario, al caso de uso para que el test pase, tiene que fallar.

IV. **Se escribe el mínimo código del caso de uso que permita que pase el test.** Es imprescindible ser conciso, hay que desarrollar solamente el código necesario, nada de bucles o bloques condicionales innecesarios.

V. Se **lanzan todos los test unitarios** para comprobar que todo funciona correctamente.

VI. Si se puede hacer alguna **refactorización en el código del caso de uso** se hace en este momento. El objetivo de este punto es simplificar y limpiar el código.

VII. Se vuelven a lanzar los test unitarios para comprobar que tras la refactorización todo sigue funcionando.

VIII. Si todo está correcto, se marca la tarea de la checklist como completada y se vuelve al punto I con la siguiente tarea de la checklist y se repite todo el proceso hasta que se completen todas las tareas.

Otro tema importante es que hay que desarrollar el código del caso de uso en el mismo fichero donde están los test, ya que no tiene sentido dividir el código en las diferentes capas de la arquitectura hexagonal en este momento.

También es importante destacar, que aunque el proyecto este montado con *value-objects*[6] **el desarrollo del caso de uso se realizará con primitivos**. Se pasará a utilizar *value-objects* en la refactorización final del caso de uso.

Finalmente, una vez que se ha culminado las tareas de la checklist, es el momento de la refactorización final del código del caso de uso. Será en este punto cuando se modifiquen las entidades y se pasen a sustituir los primitivos por los *value-objects* correspondientes y se simplifique el caso de uso desarrollado.

Después de las refactorizaciones, habrá que dividir el código desarrollado y distribuirlo por las capas del proyecto. Habrá que identificar que partes del código corresponden a la capa de aplicación, cuales a la de dominio y cuales a la de infraestructura. Como se comentó en secciones anteriores, la arquitectura hexagonal divide el código en tres capas diferentes:

- **Aplicación**: esta capa contiene los casos de uso de la aplicación, las suscripciones a los eventos de dominio...
- **Dominio**: esta capa contiene las entidades, las definiciones de los repositorios, los servicios de dominio, value-objects...
- **Infraestructura**: dentro de esta capa se definen las implementaciones específicas de los repositorios definidos en el dominio.

Tras dividir el código y distribuirlo, habrá que lanzar de nuevo los test desarrollados para comprobar que no se ha roto nada y que el caso de uso sigue funcionando correctamente.

Una vez llegado a este momento, ya se habrá concluido el desarrollo del caso de uso implementando el mínimo código necesario. Esto tiene como resultado un código muy limpio y con una calidad muy alta.

---

[6] Un *value-object* es un concepto utilizado en la programación orientada a objetos para representar un objeto inmutable cuyo valor se basa únicamente en sus propiedades o atributos, y no en su identidad

En resumen, siguiendo esta metodología se consigue que haya un muy buen análisis del caso de uso, un buen catálogo de test que cubre todos los requisitos y un código robusto, limpio, seguro y mantenible.

A continuación, se pasa a explicar algunas de las decisiones y aspectos más relevantes de la parte backend de este proyecto.

5.2. Implementación del backend

A lo largo de esta sección se van a documentar algunos de los aspectos más relevantes de la implementación de la parte backend de la aplicación.

Una vez presentada la metodología de desarrollo que se va a utilizar, no será necesario explicar cómo se desarrolló cada uno de los casos de uso, simplemente se mostrará el listado de los casos de uso existentes y se podrá omitir como fue su desarrollo, ya que se comentó en la sección anterior.

Otro de los aspectos que se comentará será como es la estructura del código diseñada para encajar dentro de la arquitectura hexagonal, esto se explicará en esta sección, apoyando la literatura con un diagrama de clases.

Finalmente, se concluirá la sección explicando el diseño del API desarrollada, mostrando los endpoints disponibles y explicando los formatos de los datos que devolverá cada uno de ellos.

5.2.1. Listado de casos de uso

El núcleo de la arquitectura hexagonal está formado por las entidades, eventos de dominio y, entre otros elementos, están los casos de uso. Los casos de uso [43] son una secuencia de acciones realizadas por el sistema que producen un resultado para una petición en particular, es decir, representan el comportamiento del sistema con el fin de dar respuestas a los usuarios.

Esta aplicación consta con tantos casos de uso como métricas diferentes definidas. De este modo, un caso de uso hará todas las acciones necesarias

para obtener los datos que permitan, posteriormente a la parte frontend, visualizar la métrica.

Como se comentó en secciones anteriores, las métricas fueron agrupadas en cuatro grupos diferentes, según la propiedad por la que se realiza el análisis. Por esta razón, cada agrupación representará un recurso diferente dentro de la arquitectura. En este contexto, se denomina recurso a un modelo de datos concreto que tiene una serie de propiedades definidas y que es totalmente independiente con respecto a otros modelos de datos.

Todo el diseño arquitectural de este proyecto gira en torno a los recursos, cada uno de ellos tendrá su capa de aplicación, dominio e infraestructura. Y a este nivel se agruparán todos los casos de uso referentes a la agrupación correspondiente a este recurso.

A continuación, se muestra un diagrama con los modelos de datos resultantes para cada uno de los recursos del sistema:

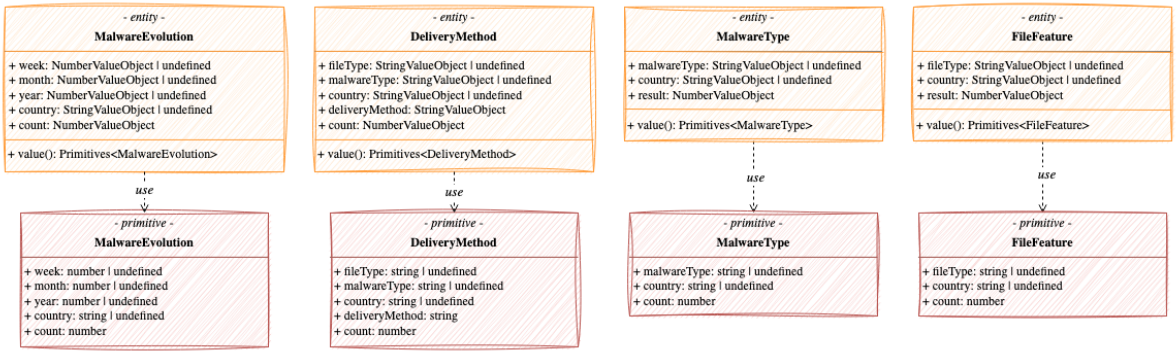


Ilustración 27. Diagrama de clases que muestra las entidades con los value-objects que las componen, y también aparecen definidos los primitivos correspondientes a cada una de las entidades.

Se puede apreciar lo comentado anteriormente, existe un recurso diferente por cada una de las agrupaciones de métricas presentadas anteriormente. Dado que no todas las métricas de un recurso tienen 100% la misma estructura, se pusieron propiedades opcionales, de forma que la entidad fuera válida para todos los casos de uso de una misma agrupación

Finalmente, se presenta un listado de los casos de uso que compondrá cada uno de los recursos:

Métodos de distribución	
findByCountry()	Caso de uso asociado a la métrica de métodos de distribución de malware más comunes por país.
findByFileType()	Caso de uso asociado a la métrica de métodos de distribución más comunes por tipo de fichero.
findByMalwareType()	Caso de uso asociado a la métrica de métodos de distribución más comunes por tipo de malware.
Características de fichero	
findAvgByMalwareType()	Caso de uso asociado a la métrica de tamaño medio de los ficheros según el tipo de malware.
findAvgByFileType()	Caso de uso asociado a la métrica de tamaño medio de los ficheros según el tipo de fichero.
findByFileType()	Caso de uso asociado a la métrica de distribución del tipo de ficheros maliciosos.
findByMalwareAndFileType()	Caso de uso asociado a la métrica de tipos de malware más comunes para cada tipo de fichero.
Evolución del malware	
findByCountry()	Caso de uso asociado a la métrica de evolución de la detección de malware en el mundo.
findByMonth()	Caso de uso asociado a la métrica de evolución mensual de malware.
findByWeek()	Caso de uso asociado a la métrica de evolución semanal de malware.
findByPeriod()	Caso de uso asociado a las métricas de comparativa anual de malware y comparativa mensual de malware.
Tipos de malware	
findByCountry()	Caso de uso asociado a la métrica de tipos de malware más comunes por país.
findMoreNumDetections()	Caso de uso asociado a la métrica de variantes de una familia de malware vs. Infecciones detectadas.
findByMalwareType()	Caso de uso asociado a la métricas tipos de malware más comunes y variantes de una familia de malware vs. Infecciones detectadas.

Tabla 20. Tabla con el listado de casos de uso asociados a cada recurso del sistema.

5.2.2. Estructura del código dentro de la arquitectura hexagonal

Se ha comentado en varias secciones de este documento que se iba a implementar el código utilizando la arquitectura hexagonal. Pero ¿cómo se implementa dicha arquitectura dentro de este proyecto? En esta sección se va a tratar de dar respuesta a esta pregunta.

Se conocen los conceptos teóricos de la arquitectura hexagonal, ya que se explicaron en las secciones 4.1 y 4.3, pero llevar los aspectos teóricos al código no fue una tarea sencilla.

Este tipo de arquitectura parecía muy interesante, y se había realizado alguna pequeña prueba antes de la elaboración de este trabajo. Pero no fue hasta la llegada de este proyecto cuando se intentó realizar un desarrollo de estas dimensiones integró con esta arquitectura.

Finalmente, se consiguió una arquitectura muy limpia y un código muy bien estructurado. Todo ello fruto de un gran trabajo base de investigación sobre este tipo de arquitectura. Una investigación que se inició con anterioridad al inicio de este proyecto y que al finalizarlo, se puede concluir que se han interiorizado y asentado las bases de este tipo de arquitectura muy firmemente.

Dado que este proyecto es medianamente grande para representarlo de forma completa en este documento, se va a proceder a mostrar un diagrama de clases referente a un único recurso del sistema. El diagrama de clases completo sería replicar la misma información pero para el resto de recursos.

El diagrama de clases para un único recurso mostrará una imagen perfecta de cómo se implementa la arquitectura hexagonal dentro de este proyecto. Se puede ver un organización muy cuidada y pensada para que el software desarrollado tenga una calidad óptima.

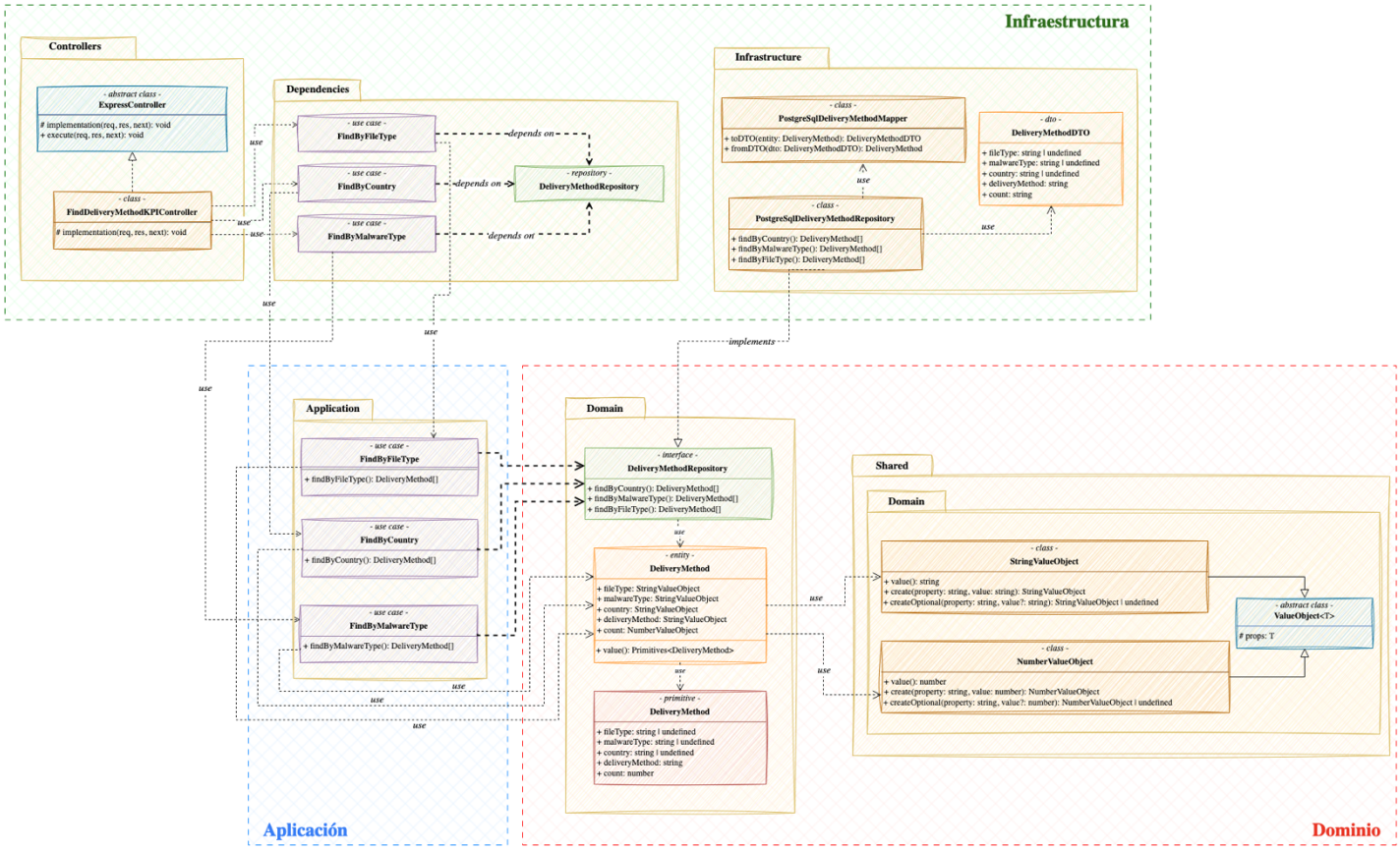


Ilustración 28. Diagrama de clases correspondiente al recurso delivery-method.

En las próximas líneas se va a comentar minuciosamente cada una de las partes de este diagrama de clases y se mostraran imágenes de este diagrama ampliadas para ver mejor los detalles.

Como se comentó en secciones anteriores, la arquitectura hexagonal está compuesta por tres capas: la capa de dominio, la de aplicación y la de infraestructura. En este diagrama aparece marcado por colores que clases corresponden a cada una de estas capas.

Cada una de estas capas se encuentra a un nivel de profundidad dentro de la arquitectura y, tal y como se puede comprobar, las comunicación entre capas es unidireccional, desde las capas exteriores hacia las interiores.

Esta todo perfectamente organizado y siguiendo la arquitectura de forma minuciosa para que no haya nada que incumpla algunos de los aspectos de esta arquitectura.

5.2.2.1. Capa de dominio

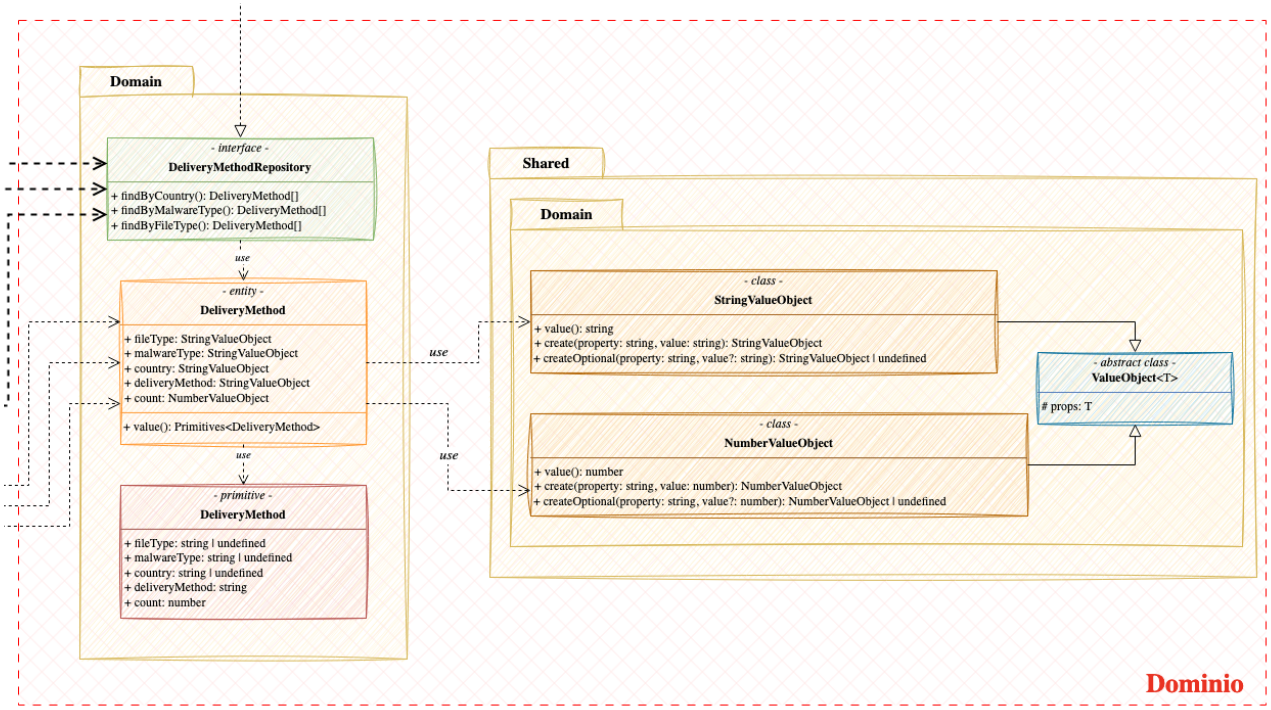


Ilustración 29. Diagrama de clases de la capa de dominio.

En primer lugar, se va a comentar todo lo referente a la capa de dominio. Esta es la capa más interna de la arquitectura y donde se encuentran todos los componentes que forman el núcleo solido de la arquitectura hexagonal.

Como se puede comprobar, aquí se definen los modelos de datos y los contratos, donde se definirán las operaciones de la interfaz del repositorio.

También se puede ver que hay un paquete denominado *shared* donde se encuentran definidos los *value-objects*. Esto se debe porque todos los recursos utilizarán los mismas clases de *value-objects*, por tanto, se decidió añadir un paquete *shared* que albergará los *value-objects* y todas las entidades de todos los recursos los utilizarán.

Antes de finalizar esta sección y proceder a detallar otra capa de las representadas en el diagrama de clases, se va a proceder a mostrar unos fragmentos de código correspondientes a las implementaciones de un *value-object*, la interfaz del *repositorio* y la *entidad*. No se entrará en detalle de que

realiza el código presentado, simplemente se presenta para ver cómo se traduce en código una parte del diagrama presentado.

```
export class StringValueObject extends ValueObject<{ value: string }> {
  get value(): string {
    return this.props.value;
  }

  protected constructor(value: string) {
    super({ value });
  }

  static create(property: string, value: string): StringValueObject {
    if (isNil(value)) {
      throw new FieldValidationError(`Property ${property} must be provided`);
    }

    if (!isString(value)) {
      throw new FieldValidationError(`Property ${property} must be a string`);
    }

    if (isEmpty(value)) {
      throw new FieldValidationError(`Property ${property} must be a non-empty string`);
    }

    return new StringValueObject(value);
  }

  static createOptional(property: string, value?: string): StringValueObject | undefined {
    if (isNil(value) || isEmpty(value)) {
      return undefined;
    }

    if (!isString(value)) {
      throw new FieldValidationError(`Property ${property} must be a string`);
    }

    return new StringValueObject(value);
  }
}
```

Código 34. Fragmento de código de la clase del value-object que encapsula y valida a las cadenas de caracteres.

Como se puede apreciar, un *value-object* no es más que un objeto que encapsula un valor y que para crearse correctamente, este valor tiene que superar una serie de validaciones que determinarán que, la cadena de caracteres en este caso, es válida.

```
export type DeliveryMethodProperties = {
  fileType?: StringValueObject;
  malwareType?: StringValueObject;
  country?: StringValueObject;
  deliveryMethod?: StringValueObject;
  count: NumberValueObject;
};

export class DeliveryMethod extends Entity<DeliveryMethodProperties> {
  get value(): Primitives<DeliveryMethodProperties> {
    return {
      fileType: this.props.fileType?.value,
      malwareType: this.props.malwareType?.value,
      deliveryMethod: this.props.deliveryMethod?.value,
      country: this.props.country?.value,
      count: this.props.count.value
    };
  }

  static create(props: Primitives<DeliveryMethodProperties>): DeliveryMethod {
    return new DeliveryMethod({
      fileType: StringValueObject.createOptional('fileType', props.fileType),
      malwareType: StringValueObject.createOptional('malwareType', props.malwareType),
      country: StringValueObject.createOptional('country', props.country),
      deliveryMethod: StringValueObject.createOptional('deliveryMethod', props.deliveryMethod),
      count: NumberValueObject.create('count', props.count, { minValue: 0 })
    });
  }
}
```

Código 35. Fragmento de código correspondiente a la entidad del recurso.

Viendo este código se puede ver como se implementa una entidad. Está compuesta por una serie de propiedades representadas mediante *value-objects*, de esta forma si todos los *value-objects* se han podido crear de forma correcta, los datos que componen dicha entidad serán válidos.

```
export interface DeliveryMethodRepository {
  findByCountry(): Promise<DeliveryMethod[]>;
  findByMalwareType(): Promise<DeliveryMethod[]>;
  findByFileType(): Promise<DeliveryMethod[]>;
}
```

Código 36. Fragmento de código correspondiente a la definición del contrato del repositorio.

Finalmente, el código resultante de la definición de un contrato tiene poco que destacar, ya que se trata de una simple interfaz que define las operaciones que tendrá disponible el repositorio.

5.2.2.2. CAPA DE APLICACIÓN

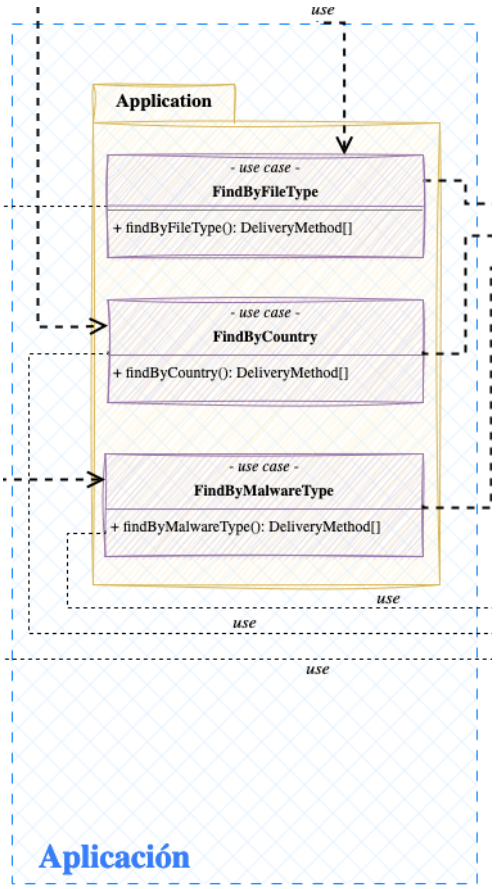


Ilustración 30. Diagrama de clases de la capa de aplicación.

La siguiente capa que posee la arquitectura hexagonal es la capa de aplicación. Es la capa que contiene la lógica de negocio de la aplicación, compuesta por los diferentes casos de uso disponibles para el recurso correspondiente.

Esta capa dependerá de abstracciones, no de implementaciones, por tanto, a la hora de importar el repositorio, no importará la implementación concreta, importará el contrato que define las operaciones disponibles. La asignación de que repositorio concreto utilizará se hará a la hora de exportar las dependencias.

Estos son todos los aspectos más importantes a destacar de esta capa, a continuación se va a presentar el código de uno de los casos de uso que se muestran en el diagrama de clases. De esta forma se podrá ver como se traduce la información del diagrama de clases al código fuente.

```
export function findByCountryBuilder({
  deliveryMethodRepository
}: {
  deliveryMethodRepository: DeliveryMethodRepository;
}): UseCase<void, DeliveryMethod[]> {
  return async function findByCountry(): Promise<DeliveryMethod[]> {
    return deliveryMethodRepository.findByCountry();
  };
}
```

Código 37. Fragmento de código correspondiente al caso de uso de findByCountry().

Este código tiene algunos aspectos interesantes a comentar, se decidió utilizar una función constructora de esta forma para poder realizar la inyección de dependencias. Cuando se construya el caso de uso en las

dependencias, se le pasará por parámetro a la función constructora la implementación concreta del repositorio y el caso de uso trabajará con la concreción determinada sin modificar su código.

Otra razón por la que se realizó de esta forma es porque no se quiso utilizar una clase para modelar los casos de uso, ya que se quería que fueran lo más simples, limpios y livianos posible. Por todas estas razones se modelo con una función constructora, que tras construirse devolverá la función correspondiente al caso de uso con el repositorio concreto inyectado.

5.2.2.3. Capa de infraestructura

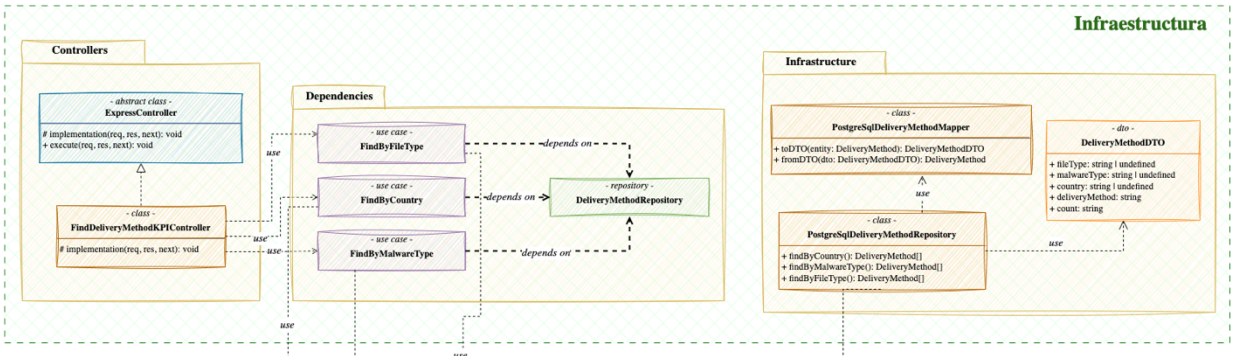


Ilustración 31. Diagrama de clases correspondiente a la capa de infraestructura.

Finalmente, se llegó a la capa de infraestructura, la más externa de la arquitectura hexagonal. Aquí se definen todas las concreciones de las implementaciones: repositorios concretos para el tipo de base de datos a utilizar, controladores concretos del tipo de servidor a utilizar, etc.

Esta aplicación concretamente tiene dos concreciones a detallar, la primera es la implementación del repositorio de base de datos y la otra es el controlador que será el que invocarán las llamadas al API.

Las clases asociadas a la implementación de base de datos merece la pena detallarlas un poco. Como se puede ver, no se dispone solamente de la implementación del repositorio, sino que aparece un *mapper* y un *dto*[7].

Es probable que el modelo desarrollado en la entidad del dominio no este perfectamente mapeado con los nombres y/o estructura de la información almacenada en la base de datos. Es por esta razón por la que se tienen que definir *mappers* y *DTOs*.

Mediante el *DTO* se define la estructura de los datos que tienen en base de datos, con los nombres, tipos de datos, etc. Este objeto será el que devolverá siempre la base de datos, y para poder utilizarlo dentro de las capas internas de la arquitectura hexagonal habrá que transformarlo para utilizar la entidad de dominio.

Esta transformación se realiza mediante el *mapper*, es una clase que tiene dos funciones, una para transformar *DTOs* a entidades de dominio y viceversa. A continuación se muestra el código de un *DTO* y de un *mapper*:

```
export type DeliveryMethodPostgresDTO = {
  malware_type?: string;
  delivery_method: string;
  file_type?: string;
  country?: string;
  count: string;
};

export class PostgresqlDeliveryMethodMapper extends PostgresMapper<
  Primitives<DeliveryMethodProperties>,
  DeliveryMethodPostgresDTO
> {
  private static DICTIONARY: Record<
    keyof Primitives<DeliveryMethodProperties>,
    keyof DeliveryMethodPostgresDTO
  > = {
    malwareType: 'malware_type',
    fileType: 'file_type',
    deliveryMethod: 'delivery_method',
    country: 'country',
    count: 'count'
```

[7] Un data transfer object (DTO) es un objeto utilizado para transferir datos entre diferentes componentes de un sistema de software, actuando como un contenedor de datos sin lógica de negocio adicional.

```
  };

  constructor() {
    super(PostgresqlDeliveryMethodMapper.DICTIONARY);
  }

  toDto(entity: Primitives<DeliveryMethodProperties>): DeliveryMethodPostgresDTO {
    return super.toDto(entity);
  }

  fromDTO(dto: DeliveryMethodPostgresDTO): DeliveryMethod {
    return DeliveryMethod.create({
      ...super.fromDto(dto),
      count: parseInt(dto.count)
    });
  }
}
```

Código 38. Fragmento de código que representa a un DTO y a un mapper.

Como se puede ver, un *DTO* define cual es la estructura y los tipos de datos que posee el recurso dentro de la base de datos. Mientras que el *mapper* se encarga de hacer de traductor entre la entidad de dominio y el data transfer object.

A continuación, se muestra el código referente a la implementación del repositorio de métodos de distribución para la base de datos de PostgreSQL:

```
export function postgresqlDeliveryMethodRepositoryBuilder({
  databaseConnector
}: {
  databaseConnector: DatabaseConnector<Pool>;
}): DeliveryMethodRepository {
  const mapper = new PostgresqlDeliveryMethodMapper();

  async function find(sql: string, params: number[] = []): Promise<DeliveryMethod[]> {
    const pool = await databaseConnector.getConnection();
    const result = await pool.query<DeliveryMethodPostgresDTO>(sql, params);

    if (isNil(result.rows) || isEmpty(result.rows)) {
      return [];
    }

    return result.rows.map((deliveryMethod: DeliveryMethodPostgresDTO) =>
      mapper.fromDTO(deliveryMethod)
    );
  }

  return {
    async findByCountry(): Promise<DeliveryMethod[]> {
      return find(
        'select country, delivery_method, count(*) from malware.malware_data_filtered GROUP BY
country, delivery_method ORDER BY country asc, count(*) desc'
      );
    },
```

```
    async findByMalwareType(): Promise<DeliveryMethod[]> {
      return find(
        'select malware_type, delivery_method, count(*) from malware.malware_data_filtered GROUP
BY malware_type, delivery_method ORDER BY malware_type asc, count(*) desc'
      );
    },
    async findByFileType(): Promise<DeliveryMethod[]> {
      return find(
        'select file_type, delivery_method, count(*) from malware.malware_data_filtered GROUP BY
file_type, delivery_method ORDER BY file_type asc, count(*) desc'
      );
    }
  };
}
```

Código 39. Código resultante a la implementación de PostgreSQL del repositorio de deliveryMethod.

Como se puede ver, se hizo mucho hincapié en tener un código limpio y ordenado, en esta implementación se hizo una refactorización para que todas las llamadas al repositorio invocaran a la misma función, ya que la lógica es 100% igual, solamente cambia la sentencia SQL a ejecutar.

Finalmente, para terminar de comentar como se ha implementado la arquitectura hexagonal, falta mostrar cómo es el código de los controladores que ejecutarán directamente las llamadas al API. A continuación se expone el código de uno:

```
export class FindDeliveryMethodKPIController extends ExpressController {
  protected async implementation(req: Request, res: Response, next: NextFunction): Promise<void> {
    if (isNil(req.query) || isNil(req.query.operation)) {
      return next(BadRequest('The query params are not valid'));
    }

    switch (req.query.operation) {
      case 'count-by-country':
        return this.processCountByCountryOperation(req, res, next);
      case 'count-by-file-type':
        return this.processCountByFileTypeeOperation(req, res, next);
      case 'count-by-malware-type':
        return this.processCountByMalwareTypeOperation(req, res, next);
      default:
        return next(BadRequest('The operation is not valid'));
    }
  }

  private async processCountByCountryOperation(
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> {
    const result = await findDeliveryMethodByCountry();
    const data = result.map(DeliveryMethodByCountryMapper.toDTO);
    res.status(OK).send(data);
  }
```

```
  private async processCountByFileTypeeOperation(
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> {
    const result = await findDeliveryMethodByFileType();
    const data = result.map(DeliveryMethodByFileTypeMapper.toDTO);
    res.status(OK).send(data);
  }

  private async processCountByMalwareTypeOperation(
    req: Request,
    res: Response,
    next: NextFunction
  ): Promise<void> {
    const result = await findDeliveryMethodByMalwareType();
    const data = result.map(DeliveryMethodByMalwareTypeMapper.toDTO);
    res.status(OK).send(data);
  }
}
```

Código 40. Fragmento de código correspondiente a un controlador.

Como se puede ver, hay una función que implementa el código que tiene que ejecutar este controlador. En este caso, se define la métrica a utilizar mediante un *query param*[8] que dependiendo de su valor, llamara a un caso de uso u otro.

En conclusión, durante toda esta sección se ha podido ver el minucioso trabajo realizado para conseguir que el código sea limpio y que tenga la mayor calidad posible. Además se ha realizado un gran esfuerzo en conseguir que el código implemente correctamente la arquitectura hexagonal, intentando que este muy bien estructurado y que la organización sea intuitiva.

5.2.3. Diseño del API

Finalmente, resta comentar el diseño del API para terminar la sección de la implementación del *backend* de la aplicación.

[8] Un query param es un componente de una URL que se utiliza para enviar datos o parámetros en una solicitud HTTP. Estos parámetros se agregan al final de la URL después de un signo de interrogación y se utilizan para personalizar y filtrar la respuesta del servidor.

Al igual que ha ocurrido en otras secciones, hubo que realizar el diseño de cuatro endpoints diferentes, uno por cada agrupación de métricas. Se decidió hacer esto y no crear un endpoint diferente por cada métrica porque se incumplirían algunas directrices del diseño de APIs.

Una de las directrices del diseño de APIs [46] es la organización del API en torno a los recursos. Por consiguiente, solamente se podría tener un endpoint diferente por cada recurso, y luego mediante los verbos HTTP habrá que definir las operaciones para las diferentes métricas de cada grupo.

Dado que todas las métricas estarán asociadas a operaciones GET, se llegó a la conclusión de que habría que crear un único endpoint que mediante un *query param* se le indicase la métrica que se desea utilizar.

Por tanto, a continuación se muestran los endpoints diseñados para este proyecto:

```
GET /malware/api/kpis/malware-evolution
  QUERY PARAMS:
    operation='month-evolution' -> métrica de evolución mensual de malware
    operation='week-evolution' -> métrica de evolución semanal de malware
    operation='country-evolution' -> métrica de evolución de la detección de malware en el mundo
    operation='month-period-evolution' -> métrica de comparación de malware en un mes
    operation='year-period-evolution' -> métrica de comparación de malware en un año


GET /malware/api/kpis/file-features
  QUERY PARAMS:
    operation='avg-by-malware' -> Métrica tamaño medio de los ficheros según el tipo de malware
    operation='avg-by-file-type' -> Métrica tamaño medio de los ficheros según el tipo de fichero
    operation='count-by-file-type' -> métrica distribución de tipo de ficheros maliciosos
    operation='file-types-by-malware' -> métrica tipos de malware mas comunes para cada tipo de fichero


GET /malware/api/kpis/delivery-methods
  QUERY PARAMS:
    operation='count-by-country' -> métrica delivery-methods más comunes por país
    operation='count-by-file-type' -> métrica delivery-methods más comunes por file type
    operation='count-by-malware-type' -> métrica delivery-methods más comunes por tipo de malware


GET /malware/api/kpis/malware-types
  QUERY PARAMS:
    operation='count-by-country' -> métrica tipos de malware más comunes por país.
    operation='malware-more-detections' -> métrica tipos de malware con más infecciones
    operation='count-by-malware-type' -> métrica tipos de malware más comunes
```

Código 41. Listado de los endpoints que posee el API desarrollada.

Como se puede comprobar, solamente existe un endpoint por recurso, pero mediante los *query params* se puede obtener la información para cada una de las métricas diseñadas.

Con esta parte se cierra la implementación de la parte backend del sistema, tal y como se puede ver, el resultado es un código que ha sido muy trabajado pensando en la calidad, ligereza, limpieza y robustez. El resultado presenta una implementación de la arquitectura hexagonal bastante robusta y con una organización del código muy buena.

Para ver el código completo de la parte backend es necesario acudir al repositorio de GitHub [47]. Se recomienda acceder al anexo C para ver como descargar el código fuente y como ejecutarlo en una máquina local.

5.3. Implementación del frontend

Finalmente, la última parte restante de implementar era el frontend de la aplicación. Esta parte será la que contendrá el dashboard con las visualizaciones de las métricas explicadas anteriormente.

Esta parte podría decirse que consta de dos partes a su vez, la parte que se encarga de comunicarse con el API del backend desarrollado y luego estará la parte visual, implementada con *angular [48]* que formará la aplicación web. Son dos partes conceptuales, ya que ambas forman parte del mismo proyecto.

Por este motivo, se va a dividir esta sección en dos subsecciones, la primera detallará toda la arquitectura y la organización de la parte del proyecto frontend que se encarga de gestionar la comunicación con el API y de procesar los datos.

La otra subsección será la encargada de presentar la implementación de la aplicación angular, mostrando como ha quedado cada una de las

agrupaciones de métricas tras la implementación de las métricas diseñadas en el apartado 4.5.

5.3.1. Arquitectura y estructura

Al igual que en el backend, la arquitectura hexagonal también estará presente para la organización y estructuración de este proyecto. No iba a ser una tarea sencilla el utilizar este tipo de arquitectura para la parte frontend.

Por tanto, había que desarrollar un core, formado por las entidades, casos de uso, repositorios, etc. que se encargarían de procesar toda la lógica de negocio. Después, en las capas externas de la arquitectura, en la infraestructura, se implementarán los servicios de comunicación con el API y la aplicación angular.

La aplicación angular se consideró que tenía que estar en la capa externa de la arquitectura, ya que según este tipo de arquitectura, se debería de poder crear otras aplicaciones, con otras tecnologías y el núcleo desarrollado debería permanecer inmutable.

Dado que este proyecto es medianamente grande para representarlo de forma completa en este documento, se va a proceder a mostrar un diagrama de clases referente a un único recurso del sistema. El diagrama de clases completo sería replicar la misma información pero para el resto de recursos.

El diagrama de clases para un único recurso mostrará una imagen perfecta de cómo se implementa la arquitectura hexagonal dentro de este proyecto. Se puede ver un organización muy cuidada y pensada para que el software desarrollado tenga una calidad óptima.

Como para la parte backend se utilizó el recurso *delivery-method* para mostrar el diagrama de clases de la arquitectura desarrollada, se optó a desarrollarlo para este mismo recurso, así se podrá tener una imagen de cómo sería la arquitectura entera del proyecto para un mismo recurso. De

esta forma se podría extrapolar a cualquiera de los otros recursos sin problema, ya que la organización es exactamente la misma para cada uno de los recursos.

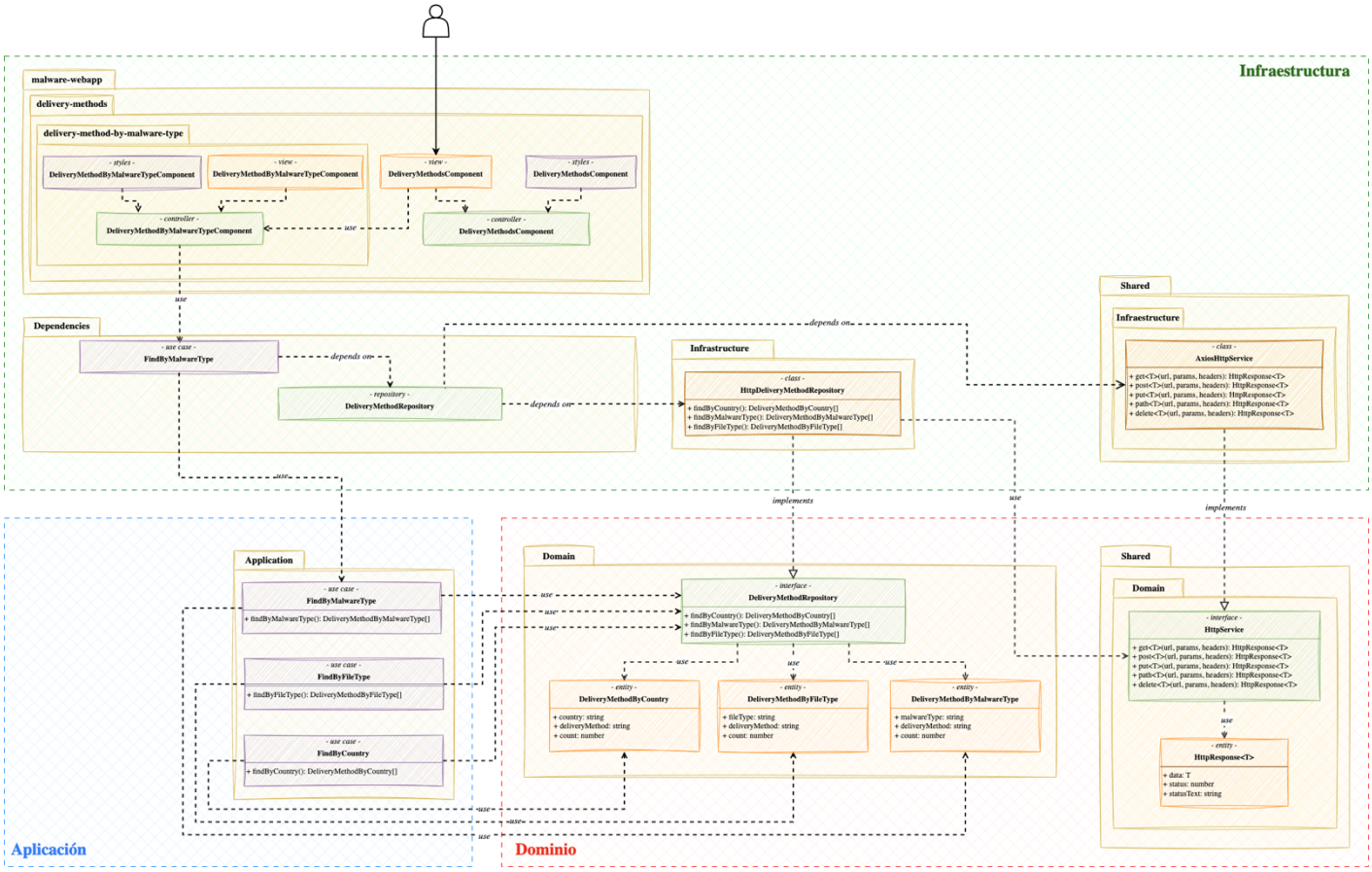


Ilustración 32. Diagrama de clases de la parte frontend de la aplicación.

Dado que es muy extenso el diagrama y que podría ser complicado comprender algunos de los aspectos más relevantes, se va a proceder a explicar los aspectos más importantes para cada una de las capas que componen a la arquitectura hexagonal.

5.3.1.1. Capa de dominio

La primera capa, la más interna de la arquitectura, es la de dominio. Como bien se comentó en secciones anteriores, esta capa contiene las diferentes entidades, interfaces con los contratos de los repositorios, etc.

En este caso se tiene dos paquetes diferentes, uno contiene el contrato del repositorio de *delivery-method* y las entidades necesarias para modelar los datos. El otro paquete contiene la interfaz con el contrato de *http* a través del cual se realizarán las llamadas al API del backend.

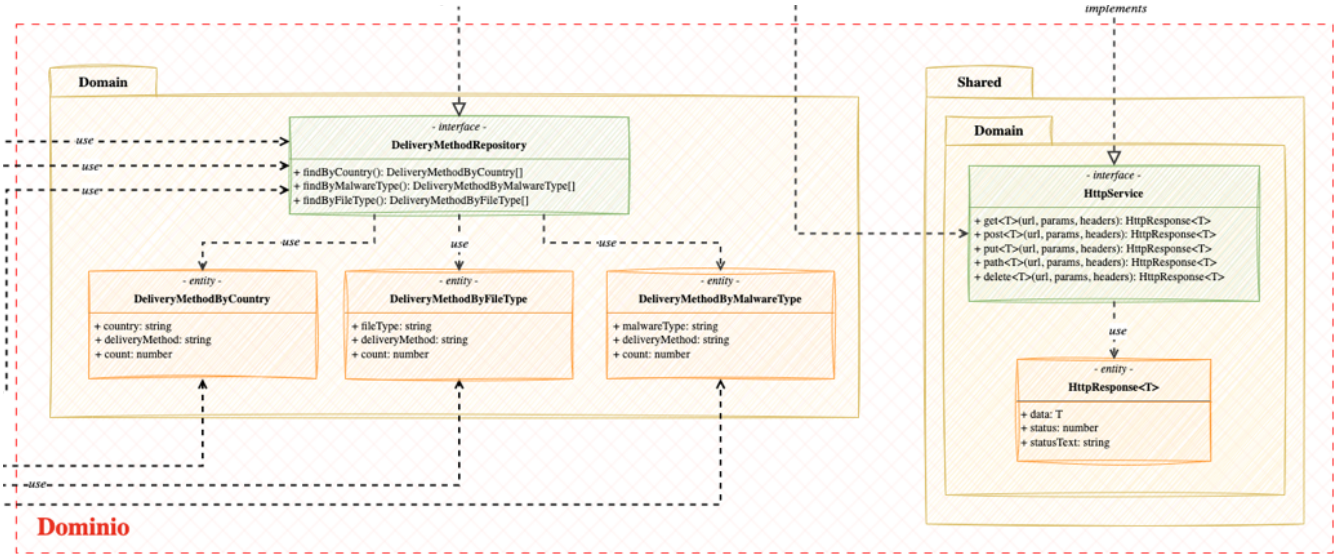


Ilustración 33. Diagrama de clases de la capa de dominio.

Todas estas clases componen el núcleo sólido de esta arquitectura, estos componentes serán utilizados desde las capas superiores. A continuación se mostrará algunos fragmentos de código de las clases que se representan en este diagrama para que se pueda ver como se transforma este diagrama en código.

```
export type DeliveryMethodByMalwareType = {
  malwareType: string;
  deliveryMethod: string;
  count: number;
};

export type DeliveryMethodByFileType = {
  fileType: string;
  deliveryMethod: string;
  count: number;
};

export type DeliveryMethodByCountry = {
  country: string;
  deliveryMethod: string;
  count: number;
};
```

Código 42. Fragmento de Código que muestra las entidades representadas en el diagrama de clases.

Como se puede ver, en el proyecto frontend no se hace uso de *value-objects*, en comparación a lo visto en el backend. Esto se debe a que aquí no había que validar ningún tipo de dato, como todos los datos provienen de base de datos, con validarlos en el backend era suficiente.

Por tanto, se decidió no implementar esa lógica en esta parte, así quedaba el código más limpio y sencillo.

```
export interface DeliveryMethodRepository {
  findByCountry(): Promise<DeliveryMethodByCountry[]>;
  findByMalwareType(): Promise<DeliveryMethodByMalwareType[]>;
  findByFileType(): Promise<DeliveryMethodByFileType[]>;
}
```

Código 43. Fragmento de código correspondiente al contrato del repositorio de delivery-method.

Sin embargo, el repositorio se puede ver cómo es idéntico al que se presentó anteriormente en la parte backend, esto se debe a que en la capa de dominio se utilizan abstracciones para que el código no se acople a ninguna tecnología concreta.

```
export type HttpResponse<T> = {
  data: T;
  status: number;
  statusText: string;
};
export interface HttpService {
  get<T>(
    url: string,
    params?: Record<string, unknown>,
    headers?: Record<string, unknown>
  ): Promise<HttpResponse<T>>;
  post<T>(url: string, data: unknown, headers?: Record<string, unknown>):
Promise<HttpResponse<T>>;
  put<T>(
    url: string,
    data: Record<string, unknown>,
    headers?: Record<string, unknown>
  ): Promise<HttpResponse<T>>;
  patch<T>(
    url: string,
    data?: Record<string, unknown>,
    headers?: Record<string, unknown>
  ): Promise<HttpResponse<T>>;
  delete<T>(
    url: string,
    headers?: Record<string, unknown>,
    data?: unknown
  ): Promise<HttpResponse<T>>;
}
```

Código 44. Fragmento de código del contrato del servicio HTTP que representará las llamadas al API.

Al igual que ocurría en el backend con las llamadas a la base de datos, aquí había que representar una interfaz para el servicio HTTP de forma que en una parte posterior del desarrollo se decidirá con que librería se implementan las llamadas al API, sería un error no implementar esta parte mediante la misma abstracción que se está utilizando para otros repositorios, iría en contra a las directrices de la arquitectura hexagonal, presentadas anteriormente.

Presentados estos fragmentos de código, se podría decir que se ha comentado todos los aspectos más relevantes de esta capa, por tanto se puede pasar a comentar la estructura de la capa de aplicación.

5.3.1.2. Capa de aplicación

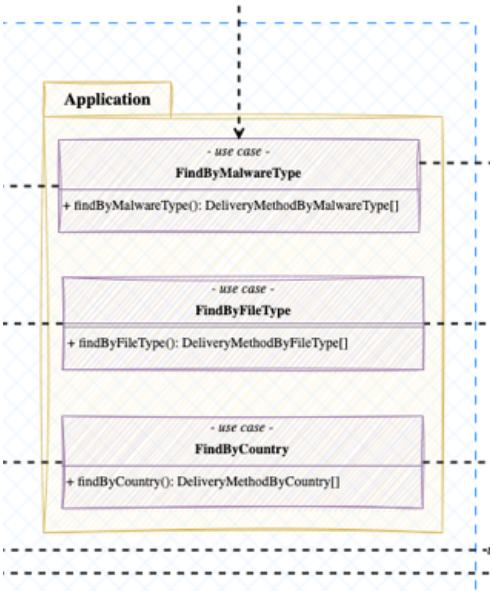


Ilustración 34. Diagrama de clases de la capa de aplicación.

En la capa de aplicación se podrán encontrar los diferentes casos de uso que posee cada recurso. Los casos de uso contienen la lógica de negocio correspondiente a cada una de las operaciones disponibles.

No tienen ninguna diferencia práctica los casos de uso del frontend y los del backend. Básicamente el concepto es el mismo, se trata de ejecutar una serie de operaciones y se devolverán al usuario.

En este caso, en lugar de comunicarse con una base de datos, como era el caso del backend, se comunicará con un API REST.

A continuación, al igual que se hizo con la capa de diseño, se va a presentar algún fragmento de código para que se pueda entender cómo se traduce el diagrama de clases anterior a la implementación.

```
export function findByMalwareTypeBuilder({
  deliveryMethodRepository
}: {
  deliveryMethodRepository: DeliveryMethodRepository;
}): UseCase<void, DeliveryMethodByMalwareType[]> {
  return async function findByMalwareType(): Promise<DeliveryMethodByMalwareType[]> {
    return deliveryMethodRepository.findByMalwareType();
  };
}
```

Código 45. Fragmento de código correspondiente a un caso de uso.

Cómo se puede observar, para el caso de este proyecto, el caso de uso del backend y el del frontend es idéntico, solamente cambia la entidad que modela los datos, ya que aquí se mapea exactamente la información que se necesita para cada una de las métricas.

5.3.1.3. Capa de infraestructura

Finalmente, se llega a la capa más externa de la arquitectura, donde se implementan las interfaces definidas anteriormente, especificando la tecnología que se utiliza para cada una de ellas.

Como se puede ver en el diagrama de clases de la capa de infraestructura, se realiza la implementación del repositorio de *Http* mediante la librería *axios [49]*. También se puede ver la implementación del repositorio de *delivery-method*. Más adelante se podría vislumbrar el código correspondiente a estas clases.

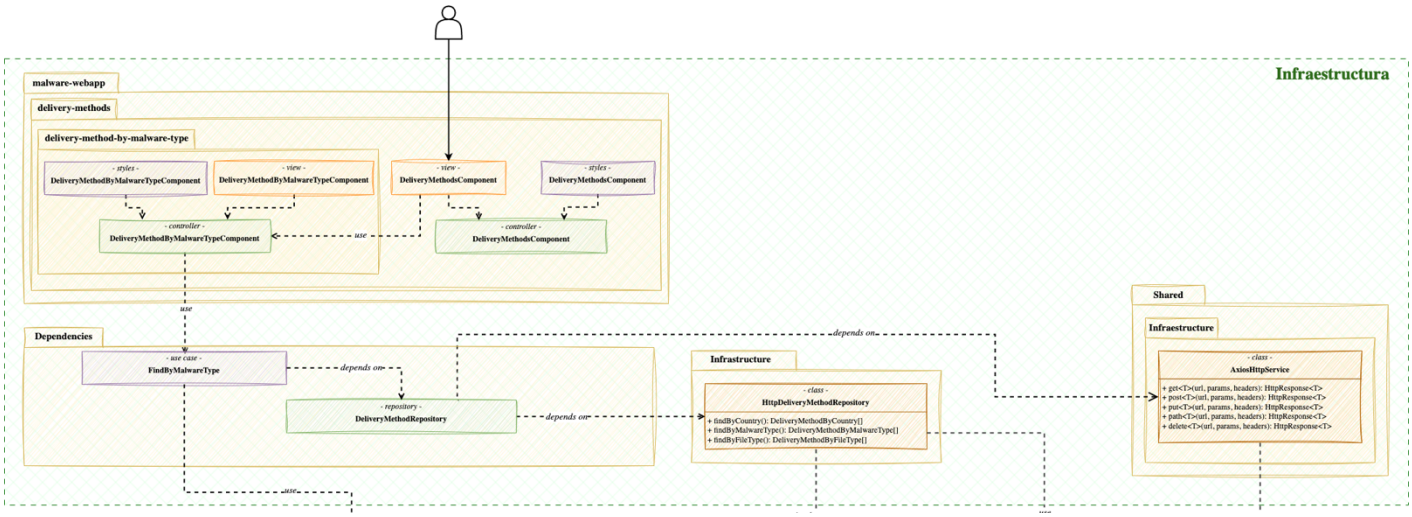


Ilustración 35. Diagrama de clases de la capa de infraestructura.

Un aspecto muy interesante es la generación de las dependencias del proyecto. Es una parte de infraestructura, ya que es propio para cada una de las aplicaciones y/o tecnologías que se deseen utilizar.

En las dependencias lo que se trata es construir mediante las concreciones implementadas los casos de uso que se tendrán que ejecutar, en este caso, desde la aplicación angular.

Finalmente, en el diagrama de clases se puede ver un muy pequeño fragmento de la aplicación angular (paquete *malware-webapp*), desde ahí se puede ver como se realiza el uso de los casos de uso construidos dentro de las dependencias del proyecto.

```
export function axiosHttpServiceBuilder(): HttpService {
  return {
    async get<ItemType>(
      url: string,
      params?: Record<string, string>,
      headers?: Record<string, string>
    ): Promise<HttpResponse<ItemType>> {
      return axios.get<ItemType>(url, { headers, params });
    },
    async post<ItemType>(
      url: string,
      data: unknown,
      headers?: Record<string, string>
    ): Promise<HttpResponse<ItemType>> {
      return axios.post(url, data, { headers });
    },
    async put<ItemType>(
      url: string,
      data: Record<string, unknown>,
      headers?: Record<string, string>
    ): Promise<HttpResponse<ItemType>> {
      return axios.put(url, data, { headers });
    },
    async patch<ItemType>(
      url: string,
      data?: Record<string, unknown>,
      headers?: Record<string, string>
    ): Promise<HttpResponse<ItemType>> {
      return axios.patch(url, data, { headers });
    },
    async delete<ItemType>(
      url: string,
      headers?: Record<string, string>,
      data?: unknown
    ): Promise<HttpResponse<ItemType>> {
      return axios.delete(url, { headers, data });
    }
  };
}
```

Código 46. Fragmento de código que representa la implementación del repositorio de http.

Como se puede ver, esta implementación utiliza *axios* para la comunicación con el API. Habrá que pasarle la URL del endpoint correspondiente para obtener los datos del API del backend.

```
export function deliveryMethodRepositoryBuilder({
  httpService
}: {
  httpService: HttpService;
}): DeliveryMethodRepository {
  return {
    async findByCountry(): Promise<DeliveryMethodByCountry[]> {
      const response = await httpService.get<DeliveryMethodByCountry[]>(
        `${env.API_URL}/kpis/delivery-methods?operation=count-by-country`
      );
      return response.data;
    },
    async findByMalwareType(): Promise<DeliveryMethodByMalwareType[]> {
      const response = await httpService.get<DeliveryMethodByMalwareType[]>(
        `${env.API_URL}/kpis/delivery-methods?operation=count-by-malware-type`
      );
      return response.data;
    },
    async findByFileType(): Promise<DeliveryMethodByFileType[]> {
      const response = await httpService.get<DeliveryMethodByFileType[]>(
        `${env.API_URL}/kpis/delivery-methods?operation=count-by-file-type`
      );
      return response.data;
    }
  };
}
```

Código 47. Fragmento de código que representa la implementación del repositorio de delivery-method.

En la implementación de este repositorio se puede apreciar cómo se define a que endpoint debe de realizar la petición. Simplemente lo que se realiza es la petición al API y se devuelven los datos para que se puedan mostrar en el dashboard.

Finalmente, se muestra un pequeño fragmento de código para que se pueda ver como se construyen las dependencias de la aplicación.

```
# Dependencia del repositorio http service
export const httpService = axiosHttpServiceBuilder();

# Dependencia del repositoro de delivery method
export const deliveryMethodRepository = deliveryMethodRepositoryBuilder({ httpService });

# Dependencia del caso de uso de find delivery method by file type
export const findDeliveryMethodByFileType = findByFileTypeBuilder({ deliveryMethodRepository });
```

Código 48. Fragmento de código que muestra cómo se construyen las dependencias.

Llegados a este punto, se puede ver que la arquitectura del frontend está también muy estudiada y meditada para que la solución sea muy limpia y estructurada primando en la calidad que tendrá el producto resultante.

Ahora se puede proceder a explicar cómo ha quedado el dashboard tras la implementación con angular.

5.3.2. VISUALIZACIONES DE LA APLICACIÓN WEB

A lo largo de esta sección se va a mostrar el resultado de la implementación del dashboard. Como se ha modelado cada grupo de métricas para pasar de los bocetos mostrados en la sección de diseño a los gráficos reales.

Esta sección se va a mostrar los resultados obtenidos tras la implementación en angular, si se desea ver el código concreto de implementarlo podrá hacerlo accediendo al repositorio de GitHub [50].

5.3.2.1. EVOLUCIÓN DE MALWARE

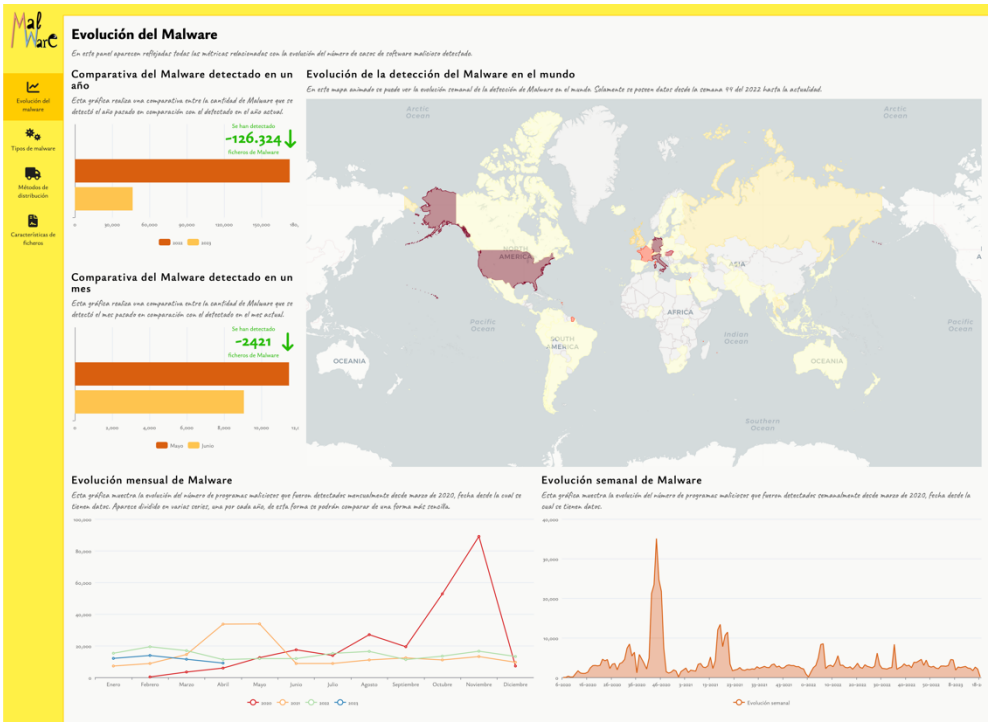


Ilustración 36. Resultado de la implementación del panel correspondiente a las métricas de evolución de malware en el tiempo.

Como se puede ver en la imagen anterior, así queda el resultado de la implementación del panel que contiene las métricas del grupo de evolución del malware en el tiempo.

Se prestó minucioso cuidado en representarlo según lo comentado en la fase de diseño, utilizando los tipos de gráficos comentados para cada una de las métricas, así como se utilizaron los colores que ahí se determinaron.

Con el tema de los colores, ahora que se puede visualizar en su conjunto todas las métricas, se puede ver como la elección de los colores de la misma gama de la aplicación tiene un resultado muy elegante, ya que todos los colores forman una armonía y no hay ninguno que destaque por encima de otro. Salvo en los gráficos que se desee destacar alguna información mediante el color.

Otro aspecto interesante es el referente al mapa, se utiliza la librería *leaflet [51]* para la implementación y la capa base es de *Carto [52]*. Todos los mapas de esta aplicación están construidos con estas tecnologías.

En este panel concretamente, se puede destacar que el mapa realiza una animación que permite ver cómo va aumentando la detección de casos a lo largo de mundo.

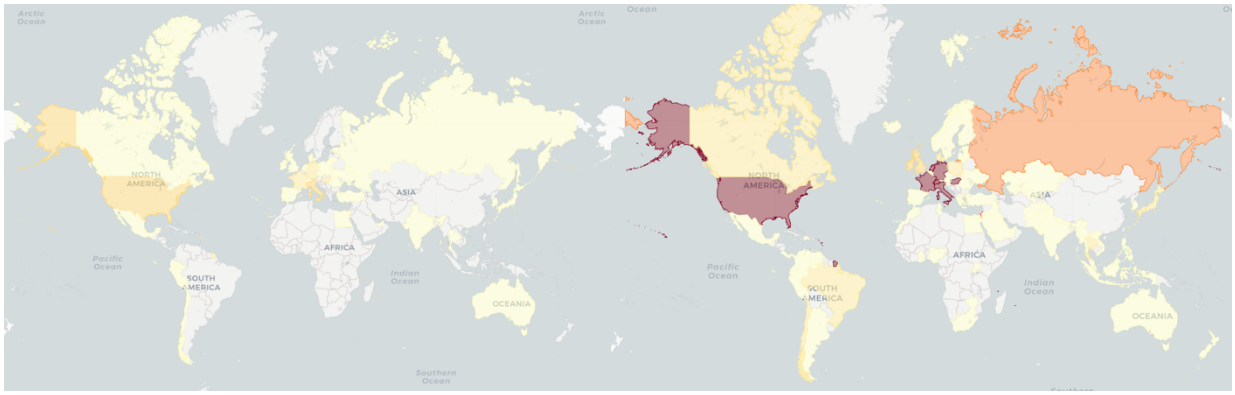


Ilustración 37. Capturas que muestran como el mapa cambia el color de los países conforme aumente el número de casos.

En las imágenes anteriores se puede ver como la coloración del mapa ha cambiado, ya que entre una imagen y la otra han pasado varias semanas y

puede verse como ha variado la detección de casos, aumentando considerablemente la detección de malware en EEUU o Rusia.

Otros de los aspectos relevantes de este panel son que si pasa el ratón por encima de los gráficos, se puede ver una tabla con los valores correspondientes para el punto dónde se encuentra el ratón.

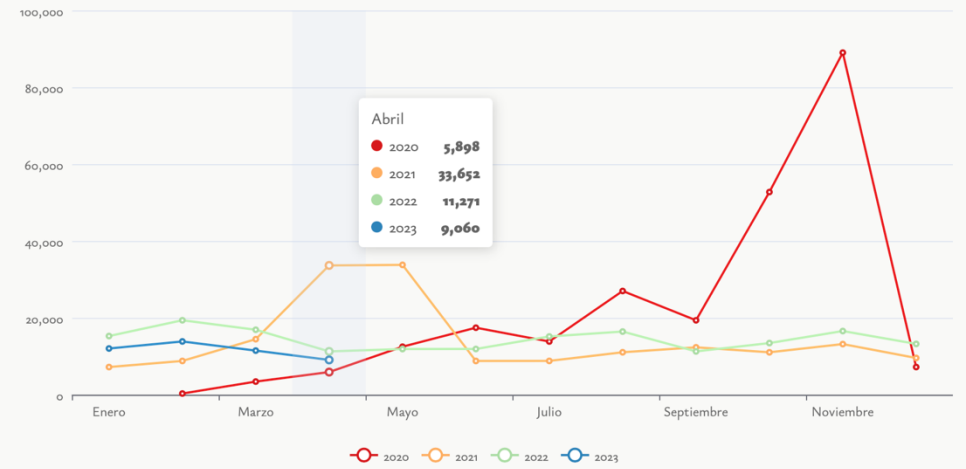


Ilustración 38. Muestra del detalle de las tablas de valores de un punto concreto del gráfico.

Como se puede ver para este gráfico, al posar el ratón sobre el mes de abril, se abre una tabla con los valores correspondientes de cada serie para el mes seleccionado. Esto es muy útil para los usuarios para conocer los valores concretos, ya que si no tendrían un valor aproximado, ya que muchas veces solo con la información de los ejes no es fácil saberlo.

Respecto a la gráfica anterior, se puede comentar lo que se explicó en la fase de diseño. Al presentar una serie por cada año, se puede comparar la evolución mes a mes para cada año. Aquí se puede apreciar que el mes de noviembre del 2020 hubo una escalada de detección de malware enrome, que prácticamente cuatriplica los datos de las series del resto de años para el mismo mes. Habría que estudiar que pudo ocurrir concretamente en ese periodo para justificar esta escalada.

Con estas líneas se puede justificar la decisión de separar los datos en tantas series como años se tenga información, ya que esta anomalía no habría podido verse de una forma tan clara como se ve con esta representación.

5.3.2.2. TIPOS DE MALWARE

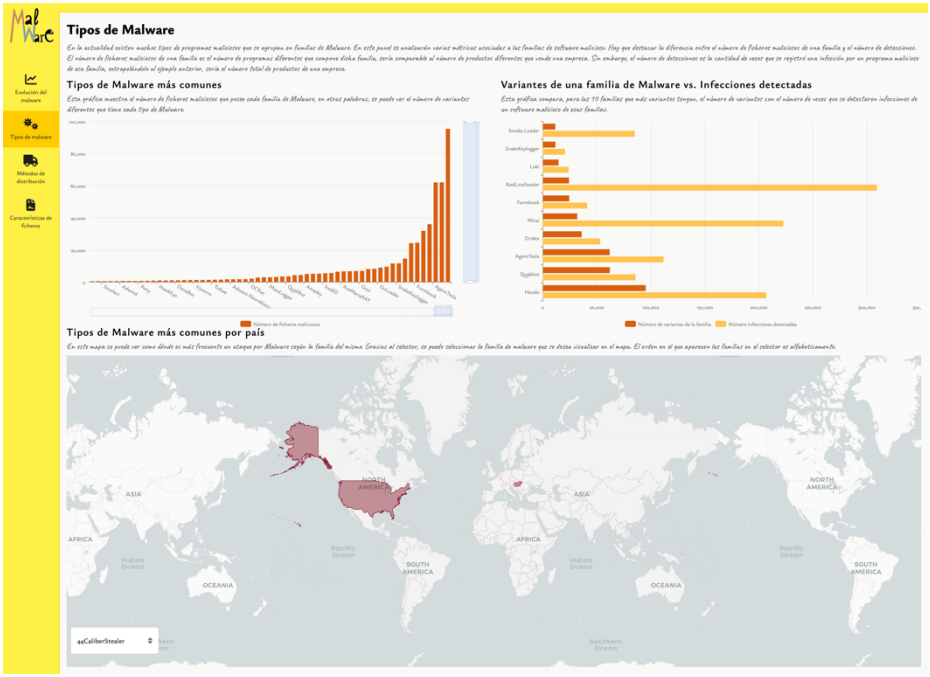


Ilustración 39. Resultado de la implementación del panel correspondiente a las métricas de los tipos de malware.

En la ilustración anterior se puede ver el resultado de la implementación de las métricas relacionadas con los tipos de malware. Como se puede apreciar para este panel también se tuvo mucho cuidado con plasmar exactamente lo que se detalló en el diseño de las métricas.

Como se puede apreciar en la gráfica de tipos de malware más comunes, aparece la posibilidad de hacer zoom en los datos, tanto de forma horizontal como de forma vertical. De esta forma se podrá ampliar para ver mejor los detalles de los datos y así poder sacar mejores conclusiones.

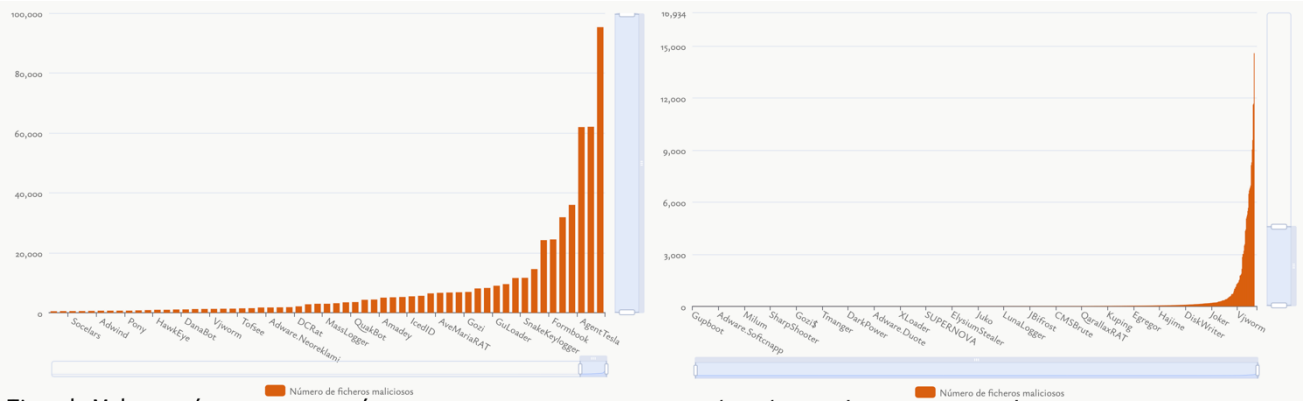


Ilustración 40. Aquí se puede ver la función del zoom en el gráfico.

En la ilustración anterior se puede ver la función del zoom sobre un gráfico. Como se puede apreciar, si no se añadiera esta funcionalidad, analizar los datos sería una tarea imposible, ya que el gráfico que aparece a la derecha aparece sin zoom en el eje x y en el gráfico de la izquierda se ha ampliado y ahora sí que se puede ver con claridad los datos que aparecen en el gráfico.

También se puede visualizar que en el mapa, con el fin de poder seleccionar mejor la familia de malware a analizar, aparece un selector con un desplegable donde se podrá seleccionar la familia de malware deseada.

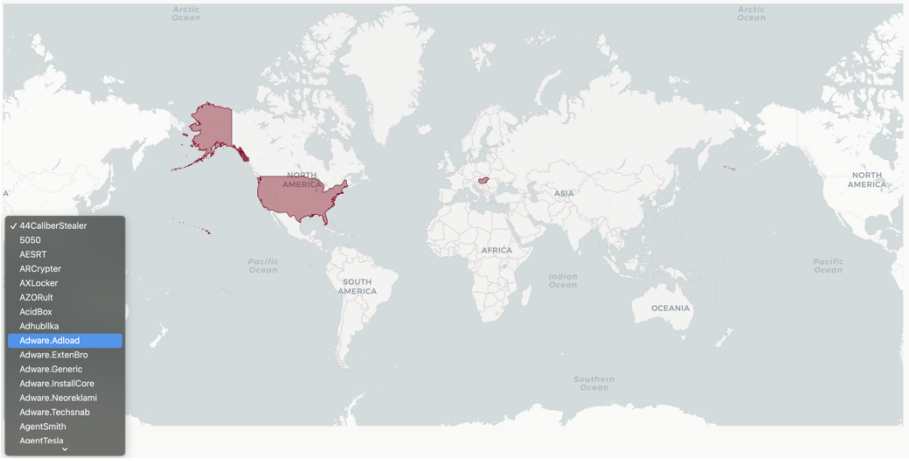


Ilustración 41. Aquí se puede ver el selector desplegado, donde se pueden ver todas las familias de malware que están disponibles para el análisis.

Como se puede apreciar, el selector no es muy invasivo y ofrece al usuario la posibilidad de seleccionar la familia de malware que desee. El listado aparece ordenado alfabéticamente para facilitarle la búsqueda de tipos de malware al usuario, de esta forma el selector será mucho más usable, ya que es más sencillo encontrar un registro si están ordenados por nombre que si intenta buscarlo sobre un listado desordenado.

Pocos aspectos más hay para resaltar de este panel. Como se puede ver, al igual que en el panel anterior, todos los colores forman una única sintonía y el resultado es muy elegante y agradable a la vista.

5.3.2.3. MÉTODOS DE DISTRIBUCIÓN

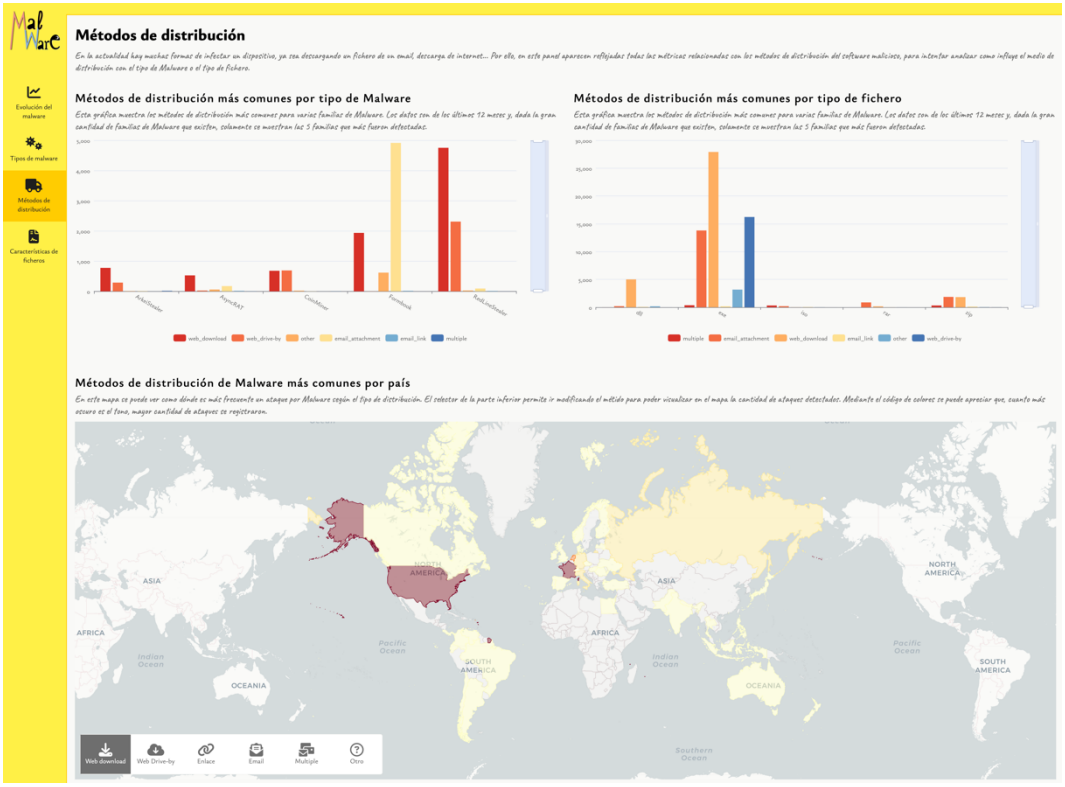


Ilustración 42. Resultado de la implementación del panel correspondiente a las métricas de los métodos de distribución.

En la imagen anterior se puede visualizar el panel que contiene las visualizaciones de las métricas relacionadas con los métodos de distribución del malware.

Como se puede apreciar el diseño de cada una de las gráficas muestra cada una de las propiedades mencionadas en la sección de diseño de métricas. Se tuvo mucho cuidado sobre todo con el selector que aparece en el mapa.

Se diseñó un selector que permite seleccionar el método de distribución y cambiar los datos que aparecen en el mapa, de esta forma el usuario podrá interactuar con él y le permitirá analizar la información mediante el medio de distribución que desee.

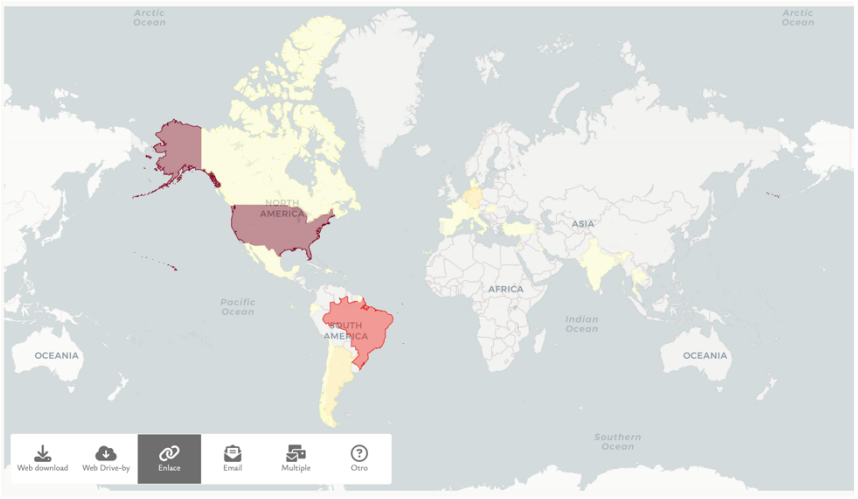


Ilustración 43. Mapa con el selector del medio de distribución.

En la imagen anterior se puede visualizar de una forma más detallada el mapa junto a este selector, como se puede ver es un elemento muy sencillo, que no tiene ningún tipo de complejidad, simplemente muestra los medios de distribución disponibles. Para que fuera un poco más elegante se añadió unos iconos, son puramente estéticos, para llamar la atención del usuario y animarle a interactuar con él.

También se puede ver, que las gráficas de barras que representan a las métricas de familias de malware y tipos de ficheros más comunes por cada medio de distribución, que poseen el mecanismo para poder hacer zoom sobre los datos que se muestran. Aquí solo se permite en el eje vertical, ya que el horizontal se puede ver con claridad todos los métodos de distribución correspondientes. Sin embargo, en el eje vertical hay datos que necesitan ampliar el gráfico para que se puedan ver con claridad.

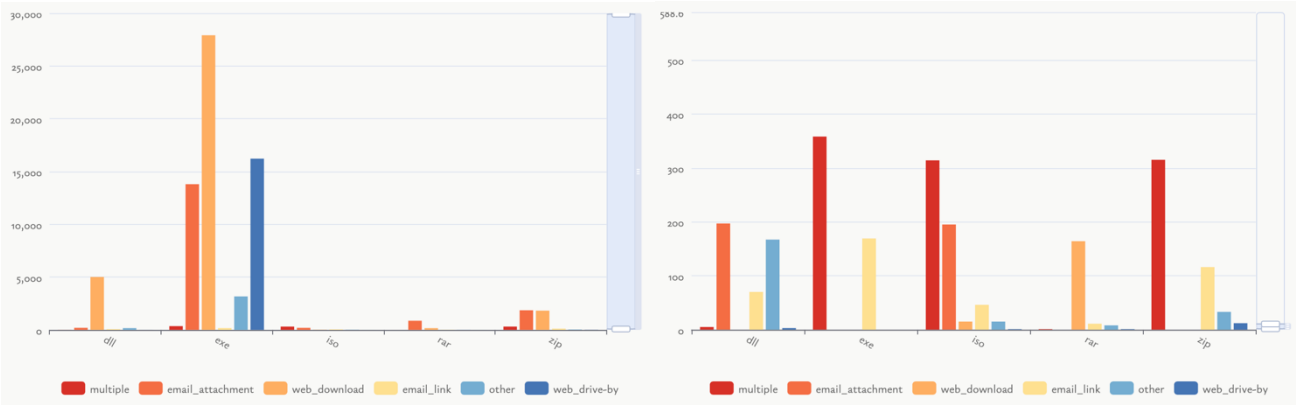


Ilustración 44. Comparativa de los datos tras hacer zoom.

En la imagen anterior se puede ver el mismo gráfico, pero al hacer zoom en el eje “y” se puede ver la información de una mejor manera, ahora se pueden visualizar mucho mejor los datos de las familias de malware que poseen un menor número de registros asociados a los métodos de distribución.

5.3.2.4. CARACTERÍSTICAS DE FICHEROS

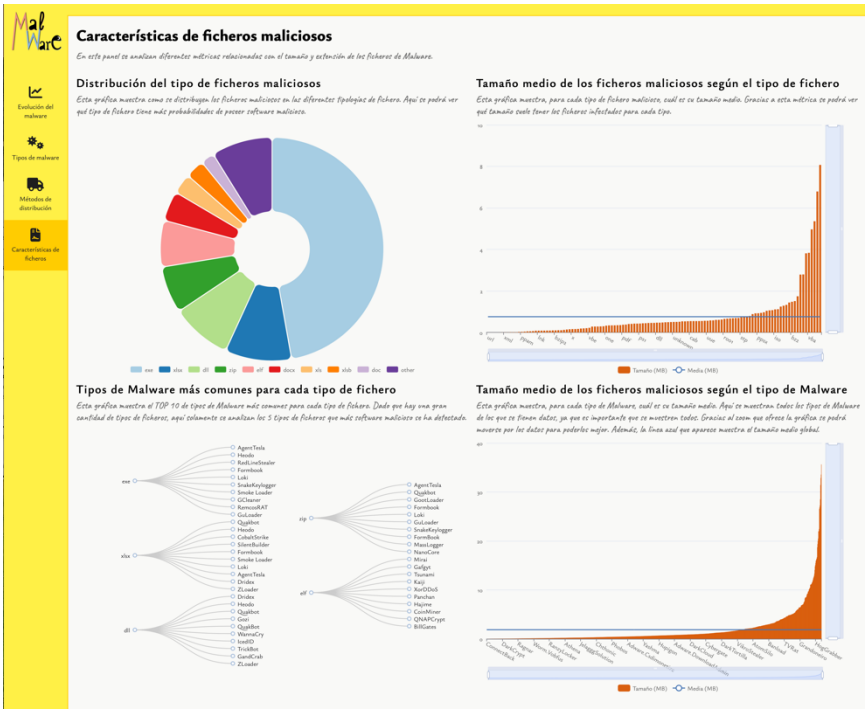


Ilustración 45. Resultado de la implementación del panel correspondiente a las métricas de las características de los ficheros.

Finalmente, se llega al último panel desarrollado, el que muestra las métricas relacionadas con las características de los ficheros maliciosos.

Al igual que se comentó en apartado anteriores, se puede ver como se tuvo especial cuidado en plasmar en la implementación del dashboard lo definido en la fase de diseño, para que no hubiera ningún tipo de diferencia.

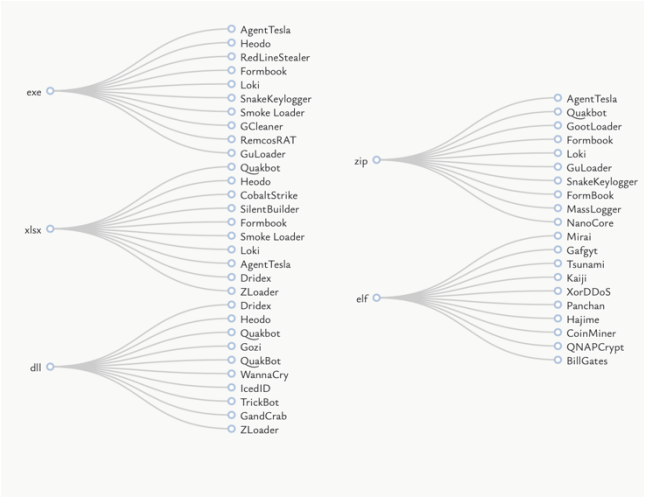


Ilustración 46. Grafo con los tipos de malware más comunes para cada tipo de fichero.

En este panel habría que destacar el grafo que muestra los tipos de malware más comunes para cada tipo de fichero. Como se puede ver, es un grafo que permite visualizar más fácilmente la información que con una tabla de datos, ya que de esta forma es muy visual la relación entre ellos.

Otro aspectos a destacar es como se representa las gráficas del tamaño medio de los ficheros, como se puede apreciar la elección de los colores fue la idónea para poder visualizar correctamente las barras y la línea que presenta la media absoluta de los datos.

Además, estas gráficas también presentan la posibilidad de hacer zoom tanto horizontal como vertical para poder ver de una mejor forma los detalles que muestran los datos, ya que al haber una gran cantidad de registros, hay algunas progresiones o patrones que podrían quedarse ocultos si no se hace zoom en los datos.

Otro gráfico a resaltar de esta sección es el gráfico circular. Como se comentó en la fase de diseño, se tuvo especial cuidado con la elección de los colores y como se distribuían en el gráfico.

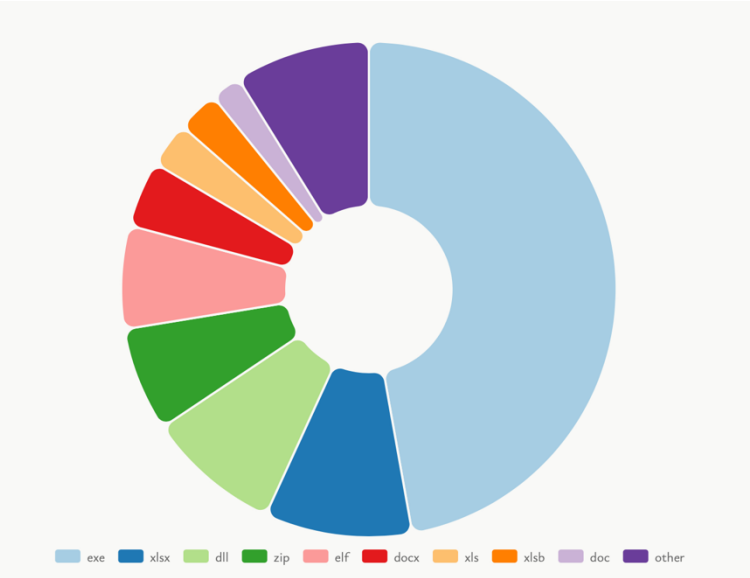


Ilustración 47. Gráfico circular con la distribución de los ficheros maliciosos.

Como se puede ver, los colores se han distribuido por misma tonalidad, de forma que es muy agradable a la vista y da una sensación de orden.

Esto muestra lo importante que son los pequeños detalles en un proyecto de visualización de datos, ya que si se pinta la información de una forma desordenada, los usuarios pueden ser incapaces de poder analizar la información o podrán analizarla pero con unas conclusiones erróneas.

Llegado a este punto, se da por concluida la fase de implementación de la aplicación, ya se posee el producto resultante del trabajo de este proyecto. Todo el código fuente se encuentra en los repositorios de GitHub [47], [50], donde se podrá acceder y descargar el código fuente para poder verlo en detalle y ejecutarlo en local (ver anexo C para más detalles).

Hay que destacar, que el dashboard se instaló en un servidor en la nube y que está disponible para que se pueda entrar y utilizar el panel. La url donde esta desplegado es:

`https://master-uned.jebiba.es`

Los detalles que indican como se desplegó el código en la nube se pueden ver en el anexo D.

6. Evaluación y calidad

Una de las principales tareas dentro de un desarrollo de software es la evaluación y la calidad del producto desarrollado. Durante esta fase se analiza el desarrollo para poder corregir malos comportamientos, errores o vulnerabilidades en el código.

El software defectuoso o inseguro puede tener consecuencias graves, tanto para los usuarios como para las organizaciones que lo utilizan. La evaluación rigurosa del software ayuda a identificar posibles errores y vulnerabilidades, permitiendo a los desarrolladores abordarlos antes de que se conviertan en problemas reales. Esto contribuye a garantizar la integridad y la seguridad del software.

Además, otro de los aspectos clave en el desarrollo software es el poder garantizar la calidad del desarrollo. Esto se puede conseguir mediante diversas formas, pero la más óptima es mediante la elaboración de una serie de pruebas que permitan validar cada una de las funcionalidades, de forma que si se realizan cambios o se añaden otras funcionalidades nuevas se pueda verificar que el software se sigue comportando bien, es decir, que se verifica que los cambios añadidos no han hecho que lo anteriormente desarrollado dejara de funcionar tal y como se esperaba.

Por tanto, en esta sección se van a detallar dos aspectos fundamentales que se han llevado a cabo dentro de este proyecto. El primero es el minucioso trabajo realizado para garantizar la calidad del producto, mediante la elaboración de una serie de pruebas que permiten saber que el código desarrollado se comporta tal y como fue diseñado.

El segundo aspecto es como se evaluó que el desarrollo estaba libre de errores y de vulnerabilidades, así como otras métricas que permiten saber que el producto desarrollado es robusto, seguro y fiable. Esto se consiguió mediante el uso de sonarqube [53].

6.1. Calidad del software mediante tests

Las pruebas unitarias, o test unitarios, desempeñan un papel fundamental en la creación de aplicaciones robustas y seguras. Los tests unitarios son pruebas automatizadas que se centran en evaluar el comportamiento individual de las unidades de código, como funciones, métodos o clases, de forma aislada. Estas pruebas se realizan antes de integrar las unidades en el sistema completo y son esenciales para garantizar la calidad del software.

La principal ventaja de utilizar pruebas unitarias es que permiten detectar errores y problemas de funcionamiento en una etapa temprana del desarrollo. Al evaluar cada unidad de código de forma aislada, los desarrolladores pueden identificar y corregir errores específicos antes de que se propaguen y afecten a otras partes del sistema. Esto no solo agiliza el proceso de desarrollo, sino que también mejora la calidad general del software, evitando la acumulación de errores difíciles de solucionar más adelante.

Otro beneficio clave de los tests es que facilitan la detección de regresiones. Las regresiones son errores que se introducen en el código cuando se realizan cambios. Al volver a ejecutar las pruebas unitarias después de cada modificación, se puede verificar si se han introducido regresiones. De esta manera, las pruebas unitarias actúan como una red de seguridad para garantizar que los cambios realizados no rompan el funcionamiento previamente probado del software. Esto permite una evolución más segura y controlada del código base.

El uso de pruebas unitarias también contribuye a mejorar la modularidad del software. Al desarrollar unidades de código independientes y probables individualmente, se fomenta la creación de componentes bien estructurados y altamente cohesivos. Esto facilita el mantenimiento y la evolución del software, ya que los cambios en una unidad específica no afectarán necesariamente a otras partes del sistema. Además, las pruebas unitarias

actúan como una documentación viva, proporcionando ejemplos claros de cómo se espera que las unidades de código funcionen y se comporten.

Otra ventaja significativa de las pruebas unitarias es que permiten realizar refactorizaciones de manera segura. La refactorización es el proceso de mejorar el código existente sin cambiar su comportamiento externo. Al contar con pruebas unitarias sólidas, los desarrolladores pueden realizar refactorizaciones con confianza, ya que las pruebas actuarán como una protección contra posibles errores introducidos durante el proceso. Esto facilita la mejora continua del código, su legibilidad y mantenibilidad.

Este proyecto está cubierto por casi 100 test unitarios que cubren el 92,6% del código desarrollado. Como se comentó anteriormente, el desarrollo del software se realizó mediante TDD, por tanto, estos tests desarrollados son muy sólidos, ya que se generó el mínimo código para pasar los test diseñados, no al revés.

Gracias al proceso de TDD el código resultante es mucho más limpio, sencillo y robusto. A continuación se va a mostrar el código de unos test unitarios para que se pueda ver como de exhaustivas son estas pruebas y para realmente ver que los test están meditados y que permiten verificar que el código que testean realmente se comporta como debe hacerlo.

```
describe('String Value Object', () => {
  describe('Input validation', () => {
    it('should throw FieldValidationError when value is nil', () => {
      const nilValues: unknown[] = [null, undefined];
      for (const nilValue of nilValues) {
        expect(
          () => StringValueObject.create('test', nilValue as string)
        ).toThrow(FieldValidationError);
      }
    });

    it('should throw FieldValidationError when value is not a string', () => {
      const invalidValues: unknown[] = [1, true];
      for (const invalidValue of invalidValues) {
        expect(
          () => StringValueObject.create('test', invalidValue as string)
        ).toThrow(FieldValidationError);
      }
    });
```

```
    it('should throw FieldValidationError when value is empty', () => {
      expect(() => StringValueObject.create('test', '')).toThrow(FieldValidationError);
    });

    it('should create StringValueObject when value is valid', () => {
      expect(StringValueObject.create('test', 'valid_value')).toBeInstanceOf(StringValueObject);
    });
  });

  describe('value()', () => {
    it('should return the raw value', () => {
      const rawValue = 'valid_value';
      const vo = StringValueObject.create('test', rawValue);
      expect(vo.value).toEqual(rawValue);
    });
  });

  describe('createOpcional()', () => {
    describe('when raw value is undefined', () => {
      it('should return undefined', () => {
        expect(StringValueObject.createOptional('test', undefined)).toBeUndefined();
      });
    });

    describe('when raw value is not undefined', () => {
      it('should throw FieldValidationError when value is not a string', () => {
        const invalidValues: unknown[] = [1, true];
        for (const invalidValue of invalidValues) {
          expect(
            () => StringValueObject.createOptional('test', invalidValue as string)
          ).toThrow(FieldValidationError);
        }
      });

      it('should return StringValueObject', () => {
        expect(
          StringValueObject.createOptional('test', 'valid_value')
        ).toBeInstanceOf(StringValueObject);
      });
    });
  });
});
```

Código 49. Código correspondiente a los tests unitarios de "StringValueObject".

Como se puede apreciar, se comprueba cada una de las posibles casuísticas que se puedan dar en el código, de forma que si todos los test se ejecutan correctamente, se puede estar convencido al 100% de que el código realmente hace lo que se espera que haga.

Para poder ejecutar los test del proyecto, bastaría con ejecutar la sentencia "*npm run test*" y se arrancaría un proceso que ejecutaría todos los tests del sistema y arrojaría por pantalla los resultados que han devuelto.

Ilustración 48. Resultados de la ejecución de los test del sistema.

Con estas herramientas se consigue verificar la calidad del desarrollo realizado, ya que se tiene la seguridad al levantar el código en el servidor que la aplicación desarrollada cumple con los requisitos que fueron determinados para cada una de las funcionalidades del sistema.

En la siguiente sección se puede visualizar la evaluación del software mediante el uso de la herramienta *SonarQube*.

6.2. Evaluación del software con SonarQube

En el mundo del desarrollo de software, es crucial garantizar la calidad y la eficiencia del código para construir aplicaciones sólidas y fiables. Una herramienta ampliamente utilizada para evaluar y mejorar la calidad del software es SonarQube. SonarQube [53] es una plataforma de análisis estático de código abierto que permite a los desarrolladores identificar y corregir problemas de código, mantener estándares de codificación consistentes y mejorar la calidad general del software.

SonarQube proporciona una amplia gama de métricas y análisis estático para evaluar la calidad del código. Utiliza reglas predefinidas y personalizables que abarcan aspectos como complejidad, duplicación de

código, mantenibilidad, seguridad y cumplimiento de estándares de codificación. Al analizar el código fuente, SonarQube identifica automáticamente posibles problemas y genera informes detallados que ayudan a los desarrolladores a comprender las áreas que requieren atención.

Una de las características clave de SonarQube es su capacidad para evaluar la deuda técnica. La deuda técnica es un término que se refiere a los compromisos de calidad que se han adquirido durante el desarrollo del software. Estos compromisos pueden incluir la presencia de código duplicado, complejidad excesiva o violaciones de buenas prácticas. SonarQube calcula la deuda técnica y la presenta en informes claros, lo que permite a los desarrolladores priorizar y abordar los aspectos que requieren más atención. Esto ayuda a reducir la deuda técnica y mejorar la calidad general del software.

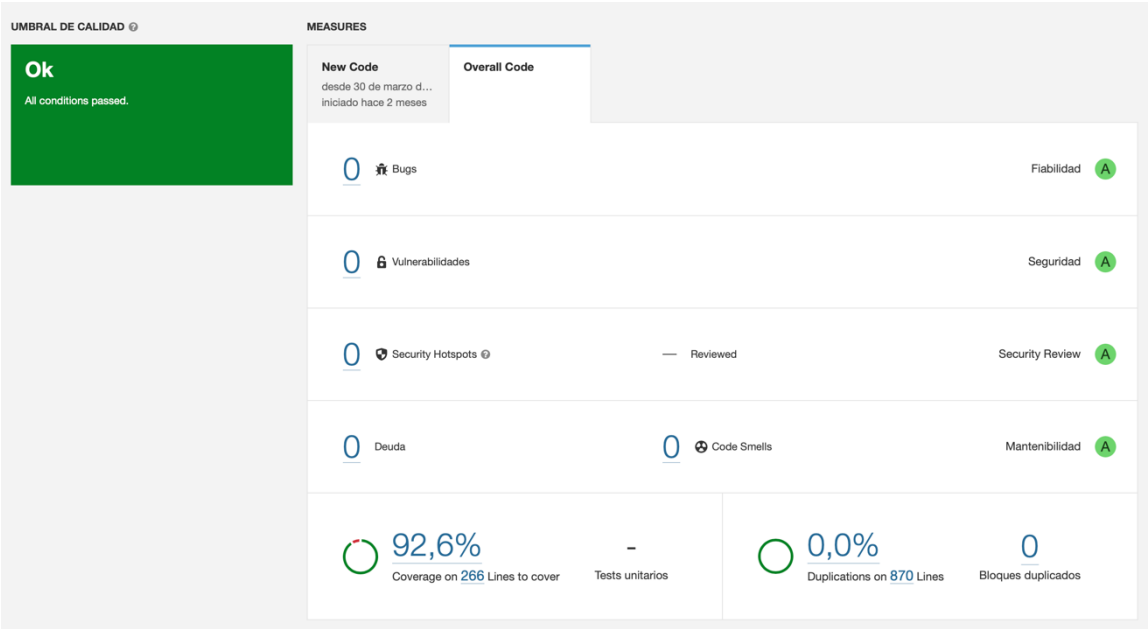


Ilustración 49. Resultados arrojados por SonarQube para el desarrollo realizado.

Tal y como se puede ver en la imagen anterior, el desarrollo final del dashboard no posee ningún tipo de error ni vulnerabilidad, así como la deuda técnica está totalmente corregida, ya que indica que la deuda es 0.

Además indica que no hay ningún bloque de código duplicado en el proyecto, ya que el porcentaje de código duplicado es del 0%. También indica que el porcentaje de código que está cubierto por los tests es del 92,6%.

Por todo ello, la puntuación que devuelve sonar para el desarrollo realizado es de "A" para cada una de las métricas que analiza, así como está marcando que todo está perfecto mediante el color verde y el texto "Ok".

Para poder realizar este análisis del código desarrollado basta con ejecutar un pequeño script que sube el código al servidor de Sonar y ya se inicia el análisis. Este script se puede ver a continuación:

```
npm install
npm run test:coverage
sonar-scanner
  -Dsonar.projectKey=< nombre del proyecto en sonar >
  -Dsonar.sources=src/contexts
  -Dsonar.host.url=http://localhost:9000
  -Dsonar.typescript.lcov.reportPaths=./coverage/lcov.info
  -Dsonar.login=< token de sesion >
```

Código 50. Script utilizado para lanzar el análisis con Sonarqube.

Una vez analizada la calidad y la evaluación del desarrollo realizado, se podía estar convencido de que el desarrollo es seguro, fiable y robusto. Por consiguiente, se puede concluir que la calidad del producto desarrollada es la mejor posible. Esto se considera todo un éxito para ser el primer proyecto que se realiza con esta arquitectura y con este modelo de desarrollo.

7. Conclusiones

Una vez terminado este proyecto final puedo concluir que todos los objetivos que me había pautado los he logrado. Nunca había desarrollado un proyecto de estas dimensiones con el fin de la visualización de los datos y creo que los resultados han sido muy buenos.

En primer lugar, he aprendido muchísimos aspectos que desconocía sobre el malware, he comprendido lo complejo que es y la gran cantidad de familias que existen. Además, investigando algunas de las familias más comunes he logrado conocer como son capaces de infectar cada dispositivo y he conocido algunas de las medidas para intentar mitigar la probabilidad de infección.

También he podido descubrir la cantidad de datos existentes en la web con información sobre malware, en este proyecto utilicé el API de Malware Bazaar, que contiene una gran cantidad de información y es una base de datos viva, muy importante para poder mejorar la aplicación en el futuro para que funcione con datos en tiempo real.

Me siento muy orgulloso del proceso que he realizado para descargar los datos, transformarlos y limpiarlos. Creo que tomé muy bien las decisiones sobre qué base de datos utilizar en cada momento, ya que en cada fase creí necesario, y así lo he justificado en la memoria, utilizar una base de datos diferente.

Quizás una de las cosas más importantes de este trabajo fue la definición de las métricas, ya que si no tienes unas buenas métricas definidas, el panel podría no ser vistoso o puede no tener una buena calidad para que los usuarios puedan realizar un buen análisis visual de los datos. Estoy muy contento, porque creo que supe seleccionar muy bien las métricas y que supe definir el tipo de gráfico más adecuado para cada una de ellas.

Pero si hay algo que más satisfecho y orgulloso me siento es el haber podido integrar la arquitectura hexagonal dentro de los proyectos, tanto en el backend como en el frontend. Tras mucho tiempo investigando y leyendo sobre esta

arquitectura, por fin pude invertir el tiempo necesario en comprender como funciona e integrarla en un proyecto real. El resultado creo que es muy simple, organizado e intuitivo. Me siento muy orgulloso como ha quedado el código desarrollado. Es el punto de partida para proyectos futuros, ya que pienso seguir mejorando esta arquitectura para conseguir elevar la calidad de los desarrollos futuros.

Finalmente, creo que el panel ha quedado muy bonito a la par que útil, creo que los usuarios que utilicen este dashboard serán capaces de analizar la información de una forma muy sencilla y que podrán sacar conclusiones sólidas con los patrones que se reflejen en los datos.

En conclusión, creo que este proyecto ha sido muy beneficioso para aplicar todos los conocimientos adquiridos en el master, sobre todo con las asignaturas relacionadas con el almacenamiento, organización y visualización de los datos. Por todo ello, puedo concluir, que he logrado todos los objetivos planteados y me ha servido como proyecto de investigación donde he podido aprender a utilizar e implementar arquitecturas muy potentes utilizando lenguajes de programación que se usan por millones de personas cada día. En consecuencia de todo ello me siento muy contento y orgulloso de haberlo conseguido.

8. TRABAJOS FUTUROS

El trabajo realizado ha dejado una serie de mejoras que sería apropiado abordar después de la entrega del presente proyecto. Implementando todas ellas se conseguirá unos resultados más profesionales y permitirá ir visualizando las métricas de los datos actualizadas a diario.

Como se ha podido leer en secciones anteriores el dashboard trabaja con datos offline, es decir, muestra las visualizaciones con una serie de datos que fueron descargados y procesados en un momento puntual del pasado. Esto se traduce en que el dashboard es estático, da igual en qué momento del día o en qué día de la semana accedas que siempre te aparecerá la información representada de la misma forma.

Esto podría mejorarse modificando el proceso de importación de datos, de forma que cada día se actualice el dataset con la información más reciente que se posea en *Malware Bazaar*. De este modo, cada vez que los usuarios accedan al panel podrán ver la última información que se disponga y siempre estará al día de los posibles cambios y tendencias que existan dentro de los datos.

Otra posible mejora de la aplicación estaría relacionada con la optimización de la interfaz del dashboard para el uso en multiplataforma. Ahora mismo se desarrolló para el uso en ordenadores que tuvieran una pantalla de unas dimensiones grandes, para que se pudiera analizar la información con claridad.

Realizando esta mejora se permitiría que la aplicación reajustara al tamaño de la pantalla, de forma que se vería perfectamente si se abre en un móvil, una tablet, un ordenador, etc.

Para ello habría que realizar un estudio minucioso de que partes de la aplicación habría que mejorar para conseguir el objetivo y posteriormente llevar a cabo la implementación de estas mejoras.

Finalmente, el mundo relacionado con el software malicioso es muy amplio y cambia muy rápido, ya que constantemente están saliendo nuevas variantes y diferentes métodos de infección. Por tanto, habría que estar constantemente al día para actualizar el dashboard añadiendo otros grupos de métricas o añadiendo otras métricas a grupos ya existentes, de forma que los usuarios puedan siempre analizar la información actual y poder sacar conclusiones sobre el malware más reciente.

BIBLIOGRAFÍA

[1] J. J. S. Chavez, «Malware: Qué Es, Cómo Funciona y Cuáles Son Los Tipos de Malware | Delta Protect», 16 de febrero de 2023. https://www.deltaprotect.com/blog/tipos-de-malware (accedido 7 de junio de 2023).

[2] Kaspersky, «Tipos de malware y ejemplos», *www.kaspersky.es*, 19 de abril de 2023. https://www.kaspersky.es/resource-center/threats/types-of-malware (accedido 7 de junio de 2023).

[3] J. Llamas, «Adware - Qué es, definición y concepto | 2023 | Economipedia». https://economipedia.com/definiciones/adware.html (accedido 7 de junio de 2023).

[4] Tibor Moes, «¿Qué es el spyware? Los 10 ejemplos más terribles (2023)», 1 de mayo de 2023. https://softwarelab.org/es/que-es-spyware/ (accedido 7 de junio de 2023).

[5] Zscaler, «Qué son los ataques de ransomware: ejemplos y prevención», *Zscaler*. https://www.zscaler.es/resources/security-terms-glossary/what-are-ransomware-attacks (accedido 7 de junio de 2023).

[6] Tibor Moes, «¿Qué es un virus troyano? Los 6 ejemplos más terribles», 1 de mayo de 2023. https://softwarelab.org/es/que-es-un-troyano-informatico/ (accedido 7 de junio de 2023).

[7] L. Garcia, «¿Qué es un gusano informático? Características y tipos», *OnRetrieval*, 7 de octubre de 2021. https://onretrieval.com/gusano-informatico/ (accedido 14 de junio de 2023).

[8] https://www.facebook.com/grokkeepcoding, «¿Qué es un gusano informático? | KeepCoding Bootcamps», 20 de junio de 2022. https://keepcoding.io/blog/que-es-un-gusano-informatico/ (accedido 14 de junio de 2023).

[9] «Qué es un gusano informático y cuáles son sus características», *WeLiveSecurity*, 5 de noviembre de 2021. https://www.welivesecurity.com/la-es/2021/11/05/que-es-gusano-informatico-caracteristicas/ (accedido 14 de junio de 2023).

[10] «¿Qué es un virus informático? - Tipos, ejemplos y más | Proofpoint ES», *Proofpoint*, 17 de noviembre de 2021. https://www.proofpoint.com/es/threat-reference/computer-virus (accedido 14 de junio de 2023).

[11] «Qué son los keyloggers: tipos, modus operandi, medidas preventivas y consejos», *LISA Institute*. https://www.lisainstitute.com/blogs/blog/keyloggers-tipos-modus-operandi-medidas-

preventivas-consejos (accedido 14 de junio de 2023).

[12] «Funcionamiento y peligros de los keylogger o registradores de teclas». https://ayudaleyprotecciondatos.es/2021/04/22/keylogger/ (accedido 14 de junio de 2023).

[13] «¿Qué es una botnet (red zombi)? Los 7 ejemplos más terribles». https://softwarelab.org/es/que-es-un-botnet/ (accedido 14 de junio de 2023).

[14] «¿Qué es un botnet? – Kaspersky Daily», 25 de abril de 2013. https://www.kaspersky.es/blog/que-es-un-botnet/755/ (accedido 14 de junio de 2023).

[15] «Qué es un PUP y cómo eliminar programas potencialmente no deseados», *Qué es un PUP y cómo eliminar programas potencialmente no deseados*. https://www.avast.com/es-es/c-what-is-pup (accedido 14 de junio de 2023).

[16] «PUP malware: qué es, por qué es peligroso y cómo protegernos de él», *RedesZone*. https://www.redeszone.net/tutoriales/seguridad/pup-malware-peligros-como-protegernos/ (accedido 14 de junio de 2023).

[17] «¿Qué es un PUP?», *latam.kaspersky.com*, 19 de abril de 2023. https://latam.kaspersky.com/resource-center/definitions/what-is-pup-pua (accedido 14 de junio de 2023).

[18] K. Petrosyan, «¿Qué es el malware sin archivos y cómo puedes protegerte contra estos ataques?», *EasyDMARC*, 24 de junio de 2022. https://easydmarc.com/blog/es/que-es-el-malware-sin-archivos-y-como-puedes-protegerte-contra-estos-ataques/ (accedido 14 de junio de 2023).

[19] «Malware sin archivos ¿Cómo funciona este ciberataque?», *Grupo Atico34*, 18 de mayo de 2021. https://protecciondatos-lopd.com/empresas/malware-sin-archivos/ (accedido 14 de junio de 2023).

[20] https://www.facebook.com/grokkeepcoding, «¿Qué es fileless malware? | KeepCoding Bootcamps», 27 de junio de 2022. https://keepcoding.io/blog/que-es-fileless-malware/ (accedido 14 de junio de 2023).

[21] K. Petrosyan, «¿Qué tan peligroso es el malware híbrido?», *EasyDMARC*, 24 de mayo de 2022. https://easydmarc.com/blog/es/que-tan-peligroso-es-el-malware-hibrido/ (accedido 14 de junio de 2023).

[22] L. Pons, «¿Qué es el malware híbrido? - Tipos de Malwares», *ICM*, 21 de diciembre de 2020. https://www.icm.es/2020/12/21/malware-hibrido/ (accedido 14 de junio de 2023).

[23] J. Jiménez, «5 métodos de distribución de malware y cómo protegernos», *RedesZone*, 18 de junio de 2018. https://www.redeszone.net/2018/06/18/5-metodos-distribucion-malware/ (accedido 7 de junio de 2023).

[24] M. Rincón Zamorano, A. Rodrigo Yuste, Ll. Tobarra Abad, y A. Robles Gómez, «Apuntes de la asignatura "Visualización de los Datos"». Máster en Ingeniería y Ciencia de Datos (UNED), 1 de febrero de 2020.

[25] V. Giraldo, «Visualización de datos: herramientas, técnicas y ejemplos [2020]», *Rock Content - ES*, 11 de septiembre de 2020. https://rockcontent.com/es/blog/visualizacion-de-datos/ (accedido 7 de junio de 2023).

[26] AWS, «¿Qué es la visualización de datos? - Explicación de la visualización de datos - AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/what-is/data-visualization/ (accedido 7 de junio de 2023).

[27] TuDashboard, «Qué es un dashboard y cómo funciona», 29 de mayo de 2021. https://tudashboard.com/que-es-un-dashboard/ (accedido 7 de junio de 2023).

[28] ABUSE.ch, «MalwareBazaar | Malware sample exchange», 1 de marzo de 2020. https://bazaar.abuse.ch/ (accedido 7 de junio de 2023).

[29] ABUSE.ch, «MalwareBazaar API - Query a malware sample (hash)», 1 de marzo de 2020. https://bazaar.abuse.ch/api/#query_hash (accedido 7 de junio de 2023).

[30] «MongoDB Documentation». https://www.mongodb.com/docs/ (accedido 7 de junio de 2023).

[31] J. Bintaned Basa, «Código desarrollado para obtener los datos del API de Malware Bazaar». https://github.com/uned-tfm/uned-tfm-dataset/blob/develop/download-data/scrapper.ts (accedido 7 de junio de 2023).

[32] MongoDB, «Aggregation Operations — MongoDB Manual». https://www.mongodb.com/docs/manual/aggregation/ (accedido 7 de junio de 2023).

[33] J. Bintaned Basa, «Script sql para poblar la base de datos PostgreSQL», *GitHub*. https://github.com/uned-tfm/uned-tfm-dataset (accedido 8 de junio de 2023).

[34] J. Bintaned Basa, «Dataset inicial de datos», *GitHub*. https://github.com/uned-tfm/uned-tfm-dataset (accedido 8 de junio de 2023).

[35] J. Bintaned Basa, «Dataset final de Malware», *GitHub*. https://github.com/uned-tfm/uned-tfm-dataset (accedido 8 de junio de 2023).

[36] M. Coppola, «Qué es Domain Driven Design (DDD), ventajas y proceso básico», 13 de abril de 2023. https://blog.hubspot.es/website/que-es-ddd (accedido 7 de junio de 2023).

[37] Vaughn Vernon, *Domain-Driven Design distilled*. Addison-Wesley.

[38] «Examples - Apache ECharts». https://echarts.apache.org/examples/en/editor.html?c=area-stack-gradient (accedido 9 de junio de 2023).

[39] «Get Started - Handbook - Apache ECharts». https://echarts.apache.org/handbook/en/get-started/ (accedido 9 de junio de 2023).

[40] J. Silk, «Los fundamentos del desarrollo del backend», *Startechup Inc.*, 8 de julio de 2022. https://www.startechup.com/es/blog/back-end-development/ (accedido 12 de junio de 2023).

[41] https://www.facebook.com/grokkeepcoding, «¿Qué es el desarrollo front end? | KeepCoding Bootcamps», 29 de junio de 2021. https://keepcoding.io/blog/que-es-el-desarrollo-front-end/ (accedido 12 de junio de 2023).

[42] M. Coppola, «Frontend y backend: qué son, en qué se diferencian y ejemplos». https://blog.hubspot.es/website/frontend-y-backend (accedido 12 de junio de 2023).

[43] Cillero, Manuel, «Casos de Uso», *manuel.cillero.es*. https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/casos-de-uso/ (accedido 12 de junio de 2023).

[44] «TDD como metodología de diseño de software». https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/ (accedido 12 de junio de 2023).

[45] «¿Qué es el TDD o Test Driven Development?», *Inesdi*. https://www.inesdi.com/blog/que-es-TDD-test-driven-development/ (accedido 12 de junio de 2023).

[46] martinekuan, «Procedimientos recomendados para el diseño de API web - Azure Architecture Center», 29 de marzo de 2023. https://learn.microsoft.com/es-es/azure/architecture/best-practices/api-design (accedido 13 de junio de 2023).

[47] J. Bintaned Basa, «Repositorio con el código fuente del backend de la aplicación.» https://github.com/uned-tfm/uned-tfm-api (accedido 12 de junio de 2023).

[48] «Angular». https://angular.io/ (accedido 13 de junio de 2023).

[49] «Empezando | Axios Docs». https://axios-http.com/es/docs/intro (accedido 8 de junio de 2023).

[50] J. Bintaned Basa, «Repositorio con el código fuente del frontend de la aplicación.», *GitHub*. https://github.com/uned-tfm/uned-tfm-app (accedido 12 de junio de 2023).

[51] «Leaflet — an open-source JavaScript library for interactive maps». https://leafletjs.com/ (accedido 13 de junio de 2023).

[52] «CARTO basemap styles». CARTO, 10 de junio de 2023. Accedido: 13 de junio de 2023. [En línea]. Disponible en: https://github.com/CartoDB/basemap-styles

[53] «SonarQube 10.0». https://docs.sonarqube.org/latest/ (accedido 11 de junio de 2023).

[54] J. Bintaned Basa, «Fichero con los md5 de los ficheros de malware.» https://github.com/uned-tfm/uned-tfm-dataset/blob/develop/download-data/full_md5.txt (accedido 8 de junio de 2023).

[55] ABUSE.ch, «MalwareBazaar | md5 data Export». https://bazaar.abuse.ch/export/txt/md5/full/ (accedido 8 de junio de 2023).

[56] «MongoDB Node Driver — Node.js». https://www.mongodb.com/docs/drivers/node/current/ (accedido 8 de junio de 2023).

[57] «Node Version Manager». nvm.sh, 8 de junio de 2023. Accedido: 9 de junio de 2023. [En línea]. Disponible en: https://github.com/nvm-sh/nvm

[58] «PostgreSQL 15.3 Documentation», *PostgreSQL Documentation*, 11 de mayo de 2023. https://www.postgresql.org/docs/15/index.html (accedido 12 de junio de 2023).

[59] «Docker Docs: How to build, share, and run applications», *Docker Documentation*, 12 de junio de 2023. https://docs.docker.com/ (accedido 13 de junio de 2023).

[60] «Docker compose», *Docker Documentation*, 12 de junio de 2023. https://docs.docker.com/compose/compose-file/ (accedido 13 de junio de 2023).

[61] «nginx documentation». https://nginx.org/en/docs/ (accedido 13 de junio de 2023).

[62] «Cómo instalar y usar Docker en Ubuntu 20.04 | DigitalOcean». https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04-es (accedido 13 de junio de 2023).

[63] «Cómo instalar y usar Docker Compose en Ubuntu 20.04 | DigitalOcean». https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04-es (accedido 13 de junio de 2023).

[64] «Cómo instalar Nginx en Ubuntu 20.04 | DigitalOcean». https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-20-04-es (accedido 13 de junio de 2023).

[65] «PM2 - Quick Start». https://pm2.keymetrics.io/docs/usage/quick-start/ (accedido 13 de junio de 2023).

[66] GPS, «Table statistics in PostgreSQL. Learn to review them - gpsos.es», 12 de marzo de 2021. https://www.gpsos.es/2021/03/table-statistics-in-postgresql-learn-to-review-them/?lang=en (accedido 7 de junio de 2023).

[67] O. C. Wiki, «Datos crudos - OCHA Colombia Wiki». https://wiki.salahumanitaria.co/wiki/Datos_crudos (accedido 7 de junio de 2023).

[68] Colaboración, «KPI's ¿Qué son, para qué sirven y por qué y cómo utilizarlos?», 29 de septiembre de 2017. https://blog.es.logicalis.com/analytics/kpis-qué-son-para-qué-sirven-y-por-qué-y-cómo-utilizarlos (accedido 7 de junio de 2023).

[69] E. Salguero, «Arquitectura Hexagonal», *Medium*, 22 de mayo de 2020. https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f (accedido 7 de junio de 2023).

[70] J. Ferrer, «Introducción Arquitectura Hexagonal - DDD», *CodelyTV*, 12 de mayo de 2016. https://codely.tv/blog/screencasts/arquitectura-hexagonal-ddd/ (accedido 7 de junio de 2023).

[71] A. Leiva, «Principios SOLID: Qué son, cuáles, y qué beneficios aporta usarlos»,

DevExperto, por Antonio Leiva, 4 de mayo de 2021. https://devexperto.com/principios-solid/ (accedido 7 de junio de 2023).

[72] picodotdev, «Introducción a DDD y arquitectura hexagonal con un ejemplo de aplicación en Java», *Blog Bitix*. https://picodotdev.github.io/blog-bitix/2021/02/introduccion-a-ddd-y-arquitectura-hexagonal-con-un-ejemplo-de-aplicacion-en-java/ (accedido 7 de junio de 2023).

[73] «Drive by Downloads: qué son y cómo evitarlas», *Ciberseguridad*. https://ciberseguridad.com/amenzas/drive-by-downloads/ (accedido 7 de junio de 2023).

[74] Alexander Shvets, *Sumergete en los Patrones de Diseño*. Refactoring.Guru, 2022. [En línea]. Disponible en: https://refactoring.guru/es/design-patterns/book

ANEXO A. FORMATO DE LOS DATOS DEL API DE "MALWARE BAZAAR"

Los datos que devuelve el endpoint que consulta toda la información disponible de un archivo malware a partir de su *hash md5* son los siguientes:

Parámetro		Descripción
sha256_hash		Hash SHA256 del fichero Malware
sha3_285_hash		Hash SHA3-384 del fichero Malware
sha1_hash		Hash SHA1 del fichero Malware
md5_hash		Hash MD5 del fichero Malware
first_seen		Timestamp de la primera vez que fue visto el malware por MalwareBazaar
last_seen		Timestamp de la última vez que fue visto el malware por MalwareBazaar
file_name		Nombre del fichero Malware
file_size		Tamaño del fichero en Bytes
file_type_mime		MIME type del fichero
file_type		Tipo del fichero
reporter		Nombre de la cuenta de la persona que detecto el Malware
origin_country		Dos letras del país desde el que se subió el fichero Malware
anonymous		1 = anónimo, 0 = no anónimo
signature		Familia de Malware
imphash		imphash (sólo disponible para ejecutables PE)
tlsh		Trend Micro Locality Sensitive Hash
telfhash		Trend Micro ELF Hash
gimphash		Equivalente imphash para binarios Go
ssdeep		sdeep es un programa para calcular hashes troceados activados por contexto (CTPH)
dhash_icon		En caso de que el archivo sea un ejecutable PE: dhash del icono de muestras
tags		Lista de tags
archive_pw		En caso de que el archivo sea un archivo protegido por contraseña: La contraseña para descifrar el archivo

code_sign	subject_cn	Asunto Nombre común (NC)
	issuer_cn	Emisor Nombre común (NC)
	algorithm	Algoritmo utilizado
	valid_from	Fecha válida desde
	valid_to	Fecha de expiración
	serial_number	Número de serie
	cscb_listed	Estado de la lista de bloqueo de certificados de firma de código (CSCB) (Verdadero o Falso)
	cscb_reason	Motivo de la lista de bloqueo de certificados de firma de código (CSCB)
delivery_method	email_attachement	Distribuido por adjunto en correo electrónico
	email_link	Distribuido por enlace en correo electrónico
	web_download	Distribuido por descarga en web
	web_drive-by	Distribuido por drive-by
	multiple	Múltiples formas de distribución
	other	Otros métodos de distribución
file_information		Información contextual sobre el archivo Malware
yara_rules	rule_name	Nombre de la regla YARA que se activó
	author	Autor de la regla YARA
	description	Descripción de la regla YARA
	reference	Referencia de la regla YARA
ole_information	oleid	Resultados de oleid
	olevba	Resultados de olevba
vendor_intel	ANY.RUN	Análisis dinámico de Malware por ANY.RUN
	CAPE	Análisis dinámico de Malware por CAPE Sandbox
	CERT-PL_MWDB	Información sobre amenazas de la base de datos de malware CERT.PL
	vxCube	Análisis dinámico de Malware por Dr.Web vxCube
	DocGuard	Reputación de documentos ofimáticos de DocGuad
	FileScan-IO	Servicio de análisis de Malware FileScan.IO

	InQuest Labs	Servicio de reputación de ficheros de InQuest Labs
	Intezer	Análisis de código de Intezer
	ReversingLabs	Reputación de ficheros e inteligencia de ReversingLabs TitaniumCloud
	Spamhaus_HBL	Reputación de ficheros Spamhaus Hash Blocklist (HBL)
	Triage	Análisis dinámico de Malware por Hatching triage
	UnpacMe	Servicio de desempaquetado de Malware de UnpacMe
	VMRay	Análisis dinámico de Malware por VMRay
	YOROI_YOMI	Análisis dinámico de Malware por YOROI YOMI
comments	id	Identificador único del comentario
	date_added	Timestamp (UTC) de cuando se realizó el comentario
	twitter_handle	Twitter del autor del comentario
	display_name	Nombre del autor para mostrar de twitter
	comment	Texto del comentario

Tabla 21. Documentación del modelo de datos que proviene del API de Malware Bazaar.

A continuación, se mostrará un ejemplo de un dato extraído directamente del API de *Malware Bazaar*, de forma que se pueda visualizar la estructura comentada anteriormente.


```
{
  "_id": {"$oid": "64521bbaccb3a117ea296f4c"},
  "anonymous": 0,
  "archive_pw": null,
  "code_sign": null,
  "comment": null,
  "comments": [
    {
      "id": "70751",
      "date_added": "2023-04-30 17:15:57",
      "twitter_handle": "zbetcheckin",
      "display_name": "zbet",
      "comment": "url : hxxp://204.44.71.71/x86_64"
    }
  ],
  "delivery_method": "web_download",
  "dhash_icon": null,
  "file_information": null,
  "file_name": "c80917ee10b7c4b88325075535ff1d91",
```

```
    "file_size": 63296,
    "file_type": "elf",
    "file_type_mime": "application/x-executable",
    "first_seen": "2023-04-30 17:15:54",
    "gimphash": null,
    "imphash": null,
    "intelligence": {
      "clamav": ["Sanesecurity.Malware.28877.LC.UNOFFICIAL",
"Sanesecurity.Malware.28878.LC.UNOFFICIAL", "Sanesecurity.Malware.28879.LC.UNOFFICIAL",
"Sanesecurity.Malware.28880.LC.UNOFFICIAL", "SecuriteInfo.com.Linux.Mirai-81.UNOFFICIAL",
"Unix.Dropper.Mirai-9977145-0", "Unix.Trojan.Mirai-9441505-0"],
      "downloads": "142",
      "uploads": "1",
      "mail": null
    },
    "last_seen": null,
    "md5_hash": "c80917ee10b7c4b88325075535ff1d91",
    "ole_information": [],
    "origin_country": "FR",
    "reporter": "zbetcheckin",
    "sha1_hash": "2708001734ad47117b7e886d281102e5df7b6e35",
    "sha256_hash": "341bd14c0a3cf7277b6600598fc7627b35ef41a11c3c5a13f279e2a50d84aa71",
    "sha3_384_hash":
"63237f24ff2e3b29dc1dd7c7c6176aba7d4d42b101e292dc78f2853287240e76e1e420a865300109449f41a275bf7f7d",
    "signature": "Mirai",
    "ssdeep": "1536:dpmbSQ6U3q7cCBT/lZsK/XDiQHLiKimfFoktCe3fYRM9:WShU3q7cEDlCK/XDv9i8Fok06fYRi",
    "tags": ["64", "elf", "Gafgyt", "mirai"],
    "telfhash": "t1992121a2ba6509a0f1fbf561b304d0450d200a1416fa36f2c275b9fadba5b820f78c37",
    "tlsh": "T19D534B17B58280FDC09AC1744B2BBA3AD93775FD0378B2A677D0EB262CA6D211E1DD44",
    "vendor_intel": {
      "CERT-PL_MWDB": {
        "detection": "mirai",
        "link":
"https://mwdb.cert.pl/sample/341bd14c0a3cf7277b6600598fc7627b35ef41a11c3c5a13f279e2a50d84aa71/"
      },
      "YOROI_YOMI": {
        "detection": "Malicious File",
        "score": "1.00"
      },
      "Intezer": {
        "verdict": "malicious",
        "family_name": "Moobot",
        "analysis_url": "https://analyze.intezer.com/analyses/dd96b1b0-3f55-48c2-9e3c-
78f487e7a04d?utm_source=MalwareBazaar"
      },
      "InQuest": {
        "verdict": "MALICIOUS",
        "url": null,
        "details": []
      },
      "Triage": {
        "malware_family": "mirai",
        "score": "10",
        "link": "https://tria.ge/reports/230430-vs4xtsad67/",
        "tags": ["family:mirai", "discovery", "linux"],
        "signatures": [
          {
            "signature": "Contacts a large (37114) amount of remote hosts",
            "score": "9"
          },
          {
```

```
            "signature": "Creates a large amount of network flows",
            "score": "9"
          },
          {
            "signature": "Changes its process name",
            "score": "7"
          },
          {
            "signature": "Reads runtime system information",
            "score": "5"
          },
          {
            "signature": "Writes file to tmp directory",
            "score": "5"
          }
        ],
        "malware_config": [
          {
            "extraction": "c2",
            "family": "mirai",
            "c2": "o.xnyidc.top"
          }
        ]
      },
      "ReversingLabs": {
        "threat_name": "Linux.Trojan.Gafgyt",
        "status": "MALICIOUS",
        "first_seen": "2023-04-30 17:16:07",
        "scanner_count": "24",
        "scanner_match": "19",
        "scanner_percent": "79.17"
      },
      "Spamhaus_HBL": [
        {
          "detection": "suspicious",
          "link": "https://www.spamhaus.org/hbl/"
        }
      ],
      "FileScan-IO": {
        "verdict": "MALICIOUS",
        "threatlevel": "1",
        "confidence": "1",
        "report_link": "https://www.filescan.io/uploads/644ea2551facab193c66d774/reports/f7f8dfed-9a2d-
44be-892a-698ff1578975/overview"
      }
    },
    "yara_rules": [
      {
        "rule_name": "BitcoinAddress",
        "author": "Didier Stevens (@DidierStevens)",
        "description": "Contains a valid Bitcoin address",
        "reference": null
      },
      {
        "rule_name": "linux_generic_ipv6_catcher",
        "author": "@_lubiedo",
        "description": "ELF samples using IPv6 addresses",
        "reference": null
      },
      {
```

```
      "rule_name": "Linux_Trojan_Gafgyt_0cd591cd",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_28a2fe0c",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_33b4111a",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_620087b9",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_807911a2",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_9e9530a7",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_a33a8363",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_d4227dbf",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Gafgyt_d996d335",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
      "rule_name": "Linux_Trojan_Mirai_01e4a728",
      "author": "Elastic Security",
      "description": null,
      "reference": null
    },
    {
```

```
        "rule_name": "Linux_Trojan_Mirai_1e0c5ce0",
        "author": "Elastic Security",
        "description": null,
        "reference": null
      },
      {
        "rule_name": "Linux_Trojan_Mirai_520deeb8",
        "author": "Elastic Security",
        "description": null,
        "reference": null
      },
      {
        "rule_name": "Linux_Trojan_Mirai_6a77af0f",
        "author": "Elastic Security",
        "description": null,
        "reference": null
      },
      {
        "rule_name": "Linux_Trojan_Mirai_e0cf29e2",
        "author": "Elastic Security",
        "description": null,
        "reference": null
      },
      {
        "rule_name": "myMirai",
        "author": null,
        "description": "Mirai",
        "reference": null
      },
      {
        "rule_name": "unixredflags3",
        "author": "Tim Brown @timb_machine",
        "description": "Hunts for UNIX red flags",
        "reference": null
      }
    ]
  }
```


Código 51. Ejemplo de dato de malware obtenido mediante el API de Malware Bazaar.

ANEXO B. DESCARGA DEL DATASET INICIAL Y ESTADÍSTICAS ASOCIADAS

A lo largo de este anexo, se podrá visualizar el procedimiento utilizado para descargar los datos del API de *Malware Bazaar*. Así como se presentarán algunas estadísticas para visualizar las dimensiones que posee este dataset.

Como se comentó anteriormente, los datos que se utilizaron para este proyecto se descargaron en un momento puntual, no utiliza datos en tiempo real, por consiguiente, si ahora se descargarán los datos nuevamente se tendría una versión más actualizada de este dataset.

Para descargar los datos, en primer lugar se utilizó un fichero que contenía los identificadores md5 de los ficheros de malware. Para este proyecto se utilizó el fichero que se puede ver aquí [54], pero se puede descargar la versión más actualizada desde la web de *Malware Bazaar* [55]. El contenido de este fichero se puede ver a continuación:

```
647aa8fc16ce612f6fc6588668a573eb
daf3108790446ea61533ed86e4c501bd
ebb23acd1389a8ed584abb763f740f0a
e9c647fa4a72f64918cb2bac87f697b9
4091b26cdd06b7f6fa568c6a1f210be9
1b0faf6df0915ce8437ce2f76493fea3
ca3a02fe9d62fe2e4c1fe87993254163
d00e1c54ba8238d808564c0a904aced3
```

Código 52. Fragmento del contenido del fichero que contiene los identificadores md5 de los malware.

Como se puede ver, hay una línea por cada identificador, por lo que será relativamente sencillo implementar una función que permita recorrer el fichero para ir leyendo los identificadores.

A continuación, se va a ir comentando el código desarrollado para descargar los datos asociados a dichos identificadores. El código se desarrolló en Typescript, ya que como el backend de la aplicación estará desarrollado en este lenguaje, se decidió implementarlo en este mismo, por si en un futuro se decidiera utilizar datos en tiempo real se pudiera utilizar el mismo script para descargar los datos e ir actualizando el dataset.

En primer lugar se desarrolló la función que permitía ir leyendo los datos del fichero y almacenar los datos en un array en memoria.

```
function readMd5File(limit = 100, inverse = false): string[] {
  const md5File = readFileSync(__dirname + "/full_md5.txt", "utf-8").split(/\r?\n/);
  let dataset = md5File.filter((line) => !line.includes("#") && line.length > 0);

  if (inverse) {
    dataset = dataset.reverse();
  }

  if (limit !== 100) {
    const max = (dataset.length * limit) / 100;
    dataset = dataset.slice(0, max);
  }

  return dataset;
}
```

Código 53. Código que muestra la función que lee el fichero de identificadores md5.

Esta función está diseñada para poder leer el fichero en su totalidad (`limit = 100`) o para leer una parte del mismo. Además, también se puede leer desde el final hacia adelante (`inverse = true`). Esto se debe a que se diseñó para ejecutarlo con dos procesos concurrentes, uno que recorriera el fichero desde el inicio hasta la mitad y el otro desde el final hasta la mitad. De esta forma se obtendrían los datos de una forma más rápida, ya que el proceso de descarga es muy lento.

Lo siguiente que se desarrolló fue el código que se encargaba de conectarse al API de *Malware Bazaar* y descargarse los datos:

```
class MalwareHttpService {
  private readonly API_URL = "https://mb-api.abuse.ch/api/v1/";
  private readonly API_KEY = <- MALWARE BAZAAR API KEY ->;

  async post(md5: string): Promise<HttpResponse<MalwareApiResponse>> {
    const body = {
      query: "get_info",
      hash: md5,
    };
    return axios.postForm<MalwareApiResponse>(this.API_URL, body, {
      headers: { "API-KEY": this.API_KEY },
    });
  }
}
```

Código 54. Clase que gestiona la comunicación con el API de Malware Bazaar.

Como se puede ver, esta clase posee una función que se encarga de realizar la llamada al API realizando una petición POST con una cabecera con el API_KEY que

proporcionó el equipo de *Malware Bazaar*. Se utilizó la librería *axios* [49] que permite hacer llamadas a servidores HTTP y devolver los datos formateados para poder utilizarlos de una forma rápida y sencilla.

El siguiente paso era desarrollar el código que se conectara a Mongo y permitiera insertar los documentos dentro de una colección determinada.

```
class MongoService {
  private readonly URI = "mongodb://<user>:<passwd>@localhost:27017";
  private readonly BD_NAME = "tfm";
  private readonly COLLECTION_NAME = "malware_data";

  private DB_CONN: Db;

  async startConnection(): Promise<void> {
    const client = new MongoClient(this.URI, {
      serverApi: {
        version: ServerApiVersion.v1,
        strict: true,
        deprecationErrors: true,
      },
    });

    await client.connect();
    this.DB_CONN = client.db(this.BD_NAME);
  }

  async insertMany<T>(docs: OptionalUnlessRequiredId<T>[]): Promise<void> {
    const db_collection = this.DB_CONN.collection<T>(this.COLLECTION_NAME);
    await db_collection.insertMany(docs);
  }
}
```

Código 55. Clase que gestiona la conexión y la escritura en MongoDB.

Como se puede apreciar, hay dos funciones, la primera realiza la conexión e indica la base de datos con la que se va a trabajar. La segunda función será la encargada de hacer la inserción de documentos dentro de la colección "*malware_data*", se utiliza la función *insertMany* porque se diseñó el código para hacer inserciones de datos mediante lotes de 50 registros, de esta forma se optimizaría un poco el tiempo de ejecución. Se utiliza la librería *mongodb* [56] que permite realizar las operaciones con Mongo desde el lenguaje de programación utilizado.

Lo siguiente que se implementó fue el proceso que irá acumulando datos dentro de un lote y que cuando se llegue a un tamaño establecido, se realice la llamada

que inserte los datos dentro de Mongo, utilizando la clase que se acaba de presentar.

```
class MalwareScrapper {
  private readonly WRITTEN_BATCH_MAX_SIZE = 50;

  private BATCH_MALWARE_RESULTS: MalwareData[] = [];
  private WRITTEN_BATCHES_COUNTER = 0;

  constructor(private readonly mongoService: MongoService) {}

  async processMalwareResult(
    result: MalwareApiResponse,
    force = false
  ): Promise <void> {
    const OK_RESULT = "ok";
    const { query_status, data } = result;

    if (query_status !== OK_RESULT) {
      return;
    }

    if (this.BATCH_MALWARE_RESULTS.length >= this.WRITTEN_BATCH_MAX_SIZE || force === true) {
      await this.mongoService.insertMany(this.BATCH_MALWARE_RESULTS);
      this.BATCH_MALWARE_RESULTS = [];
      this.WRITTEN_BATCHES_COUNTER++;
    } else {
      const malware = { ...data[0], _id: new ObjectId() };
      this.BATCH_MALWARE_RESULTS = this.BATCH_MALWARE_RESULTS.concat(malware);
    }
  }
}
```

Código 56. Código que gestiona el llenado de los lotes y la llamada para almacenarlos dentro de la base de datos.

Lo más destacable de la función que aquí se implementa es que se puede insertar los datos dentro de Mongo de dos formas diferentes, la primera cuando el lote ha llegado al tamaño máximo, es decir, se ha llenado y se insertan los datos.

O si se fuerza a escribir los datos (`force = true`), entonces da igual el tamaño del lote, tras añadir el registro en cuestión se hará la llamada para insertar los datos que haya en el lote. Esto se hizo así porque en la última ejecución, o si existe algún tipo de problema, es probable que el lote no esté lleno y por tanto si no se fuerza a escribirlos en la base de datos, los registros que estuvieran en el lote se perderían.

Llegados a este punto, solamente faltaba por desarrollar la función principal, la encargada de realizar todo el proceso de para cada identificador md5 hacer la llamada al API, añadir el registro al lote, gestionar los posibles errores, etc.

```
async function main(): Promise<void> {
  const mongoService = new MongoService();
  await mongoService.startConnection();

  const malwareApiService = new MalwareHttpService();
  const malwareScrapper = new MalwareScrapper(mongoService);

  // Del final hacia adelante
  const md5_ids = readMd5File(50, true);

  // Del principio al final
  // const md5_ids = readMd5File(50, false);

  try {
    let counter = 1;
    const total = md5_ids.length;
    console.log('========================== START PROCESS ==========================');

    for (const md5 of md5_ids) {
      let result;
      try {
        result = await malwareApiService.post(md5);

        if (result.status != 200) {
          console.log('<---- ERROR IN Md5 ${counter} ---->');
          await malwareScrapper.processMalwareResult(result.data, true);
          console.log('========================== END PROCESS ==========================');
          return;
        }

        if (counter === total) {
          await malwareScrapper.processMalwareResult(result.data, true);
        } else {
          await malwareScrapper.processMalwareResult(result.data);
        }

        await delay(100);
      } catch (err) {
        console.log('<---- ERROR IN HTTP SERVICE ---->');
        await delay(3000);
      }

      counter++;
    }

    console.log('========================== END PROCESS ==========================');
  } catch (e) {
    console.log(e);
  }
}
```

Código 57. Código principal que realiza el proceso de descarga de los datos.

Lo más destacable del código es que está preparado para ejecutarse para procesar los datos desde el principio hasta la mitad y desde el final a la mitad, de esta forma se podría ejecutar dos procesos concurrentes que vayan procesando

los datos de estas dos formas, así se irá descargando el dataset de una forma más rápida.

Otro aspecto por destacar es que mediante un bloque try/catch se gestionan los posibles errores que pudiera devolver el API de *Malware Bazaar*. Si ocurriera cualquier tipo de problema, se esperarían 30 segundos y se continuaría con la ejecución.

También se deja un tiempo entre peticiones, se hace una pausa de 0,1s. Esto se hace para no sobre cargar el servidor y para que no piense que se trata de un ataque. Según la documentación del API el endpoint [29] que se iba a utilizar no tenía ningún tipo de limitación ni de cuota de uso diario, por lo que no debería haber problemas, pero se añadió ese retardo para prevenir.

El código completo, con todas las importaciones y listo para ser ejecutado está dentro del GitHub junto al código desarrollado para este proyecto [31].

Para ejecutar el código hay que lanzar los siguientes comandos dentro de una terminal.

```
$> cd uned-tfm-dataset
$> nvm use
Found '/Users/jesusbintanedbasa/Desktop/TFM/projects/uned-tfm-dataset/.nvmrc' with version <16>
Now using node v16.17.1 (npm v8.15.0)

$> npm install
up to date, audited 191 packages in 801ms
36 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

$> npm run scrapper
```

Código 58. Comandos necesarios para ejecutar el código desarrollado para descargar los datos.

Es imprescindible instalar *node version manager* [57] para la ejecución de este código, además de haberse descargado el código del repositorio anteriormente citado para poder descargarse las dependencias utilizadas. También es necesario tener una instancia de MongoDB corriendo en el equipo dónde se vaya a ejecutar el código.

Llegados a este punto, ya se tendrá el dataset de *Malware Bazaar* descargado y almacenado en una colección de MongoDB. Antes de proseguir con los procesos de procesado, limpieza y transformación, se procedió a realizar un pequeño análisis para conocer el dataset que se tenía como punto de partida.

El primer análisis que se realizó no fue precisamente sobre el conjunto de datos, sino que comparó el tamaño de los registros del fichero que contiene los identificadores md5 respecto al número de registros descargados y almacenados en mongo. Esto se realizó para calcular el porcentaje de datos que se han perdido por errores en las llamadas al API.

	Número de registros
Fichero con identificadores md5	655.598
Colección "*malware_data*" de MongoDB	640.638
Número de errores	14.960
Porcentaje de errores	2,297%

Tabla 22. Cálculo de la tasa de pérdida de datos en la descarga del API.

Como se puede observar no hubo una gran pérdida de datos, pero hubo cerca de 15.000 registros que no se pudieron descargar por diversos motivos, como errores en las llamadas al API, posibles pérdidas de conexión, fallos de inserción de datos en MongoDB, etc.

Finalmente se concluyó que no era un problema, ya que la tasa de pérdida de datos era muy muy baja, ya que el porcentaje de datos perdidos está en el 2,3%. Por consiguiente, se decidió proseguir con este dataset a pesar de los datos perdidos.

Para culminar este análisis inicial, se procedió a calcular el peso del dataset descargado, el número de elementos y el número de columnas. De esta forma se podrá conocer tanto el coste en memoria de los datos, así como las dimensiones que poseen los datos:

Estadística	
Número de registros del dataset	640.638
Número de propiedades o columnas del dataset	32
Tamaño o peso del dataset	3,42GB

Tabla 23. Estadísticas asociadas al dataset descargado.

Aquí se culminó el análisis del dataset descargado, ya que el estudio de los tipos de datos que contiene cada una de las columnas se detalló en el Anexo A, ya que al no haber realizado todavía ningún proceso de transformación, ni limpieza de datos, los datos almacenados en la colección de mongo son idénticos a los que devuelve el API de *Malware Bazaar* y por consiguiente en dicho anexo se explica que contenido posee cada una de las columnas.

ANEXO C. DESCARGA DEL CÓDIGO FUENTE Y EJECUCIÓN EN LOCAL

A lo largo de este anexo, se detallará como descargar el código fuente desarrollado para este proyecto [47], [50] y que comandos hay que ejecutar para poder lanzar este código en local. También se definirán cuáles son los requisitos previos para poder ejecutar correctamente el código fuente.

Este anexo se va a centrar en la instalación y la puesta en marcha del código asociado al backend y al frontend de la aplicación. En este capítulo no se va a volver a comentar el proceso de limpieza y transformación de los datos, se partirá del hecho de que estos procesos han sido ejecutados con anterioridad o que se importará el fichero *postgresql-creation.sql* que se encuentra dentro del código fuente del backend [47].

Los requisitos para poder instalar la aplicación en local se pueden ver en la siguiente tabla:

Tecnología	Versión
PostgreSQL	15.3 [58]
Node Versión Management	0.39.3 [57]

Tabla 24. Tecnologías y versiones requeridas para poder ejecutar el código en local.

Una vez presentada la lista de requisitos, se procede a explicar el procedimiento para la descarga del código desde los repositorios de GitHub.

Para la descarga del código fuente simplemente hay que abrir la terminal y entrar dentro del directorio dónde se desea que se descarguen los repositorios. En este documento, a modo de ejemplo, se va a descargar el contenido dentro del directorio `~/tfm`

A continuación se va a exponer un pequeño fragmento de código dónde se van a mostrar todos los comandos a ejecutar para descargar tanto el código fuente del backend como del frontend.

```
cd ~/tfm

# Descarga del repositorio del backend
git clone https://github.com/uned-tfm/uned-tfm-api.git

cd uned-tfm-api/

# Se accede a la rama master y se descarga la última versión de los mismos
git checkout master
git pull


cd ~/tfm

# Descarga del repositorio del frontend
git clone https://github.com/uned-tfm/uned-tfm-app.git

cd uned-tfm-app/

# Se accede a la rama master y se descarga la última versión de los mismos
git checkout master
git pull
```

Código 59. Código que descarga el código fuente de los repositorios de GitHub.

Una vez descargado el código fuente, es necesario instalar las dependencias que se necesitan para ejecutar cada proyecto. El proyecto backend solamente requiere de las dependencias anotadas dentro del *package.json*, mientras que el proyecto frontend sí que necesita la instalación de dependencias para poder ejecutarse.

El siguiente fragmento de código muestra la instalación de las dependencias de las aplicaciones.

```
# Dependencias del backend
cd ~/tfm/uned-tfm-api
nvm use
npm install

# Dependencias del frontend
cd ~/tfm/uned-tfm-app
nvm use
npm install -global yarn
yarn install
yarn workspace malware-webapp-workspace install
```

Código 60. Fragmento de código que expone los comandos necesarios para la instalación de las dependencias.

Como se ha comentado anteriormente, en esta sección se supone que el proceso de descarga, limpieza y transformación del conjunto de datos se ha realizado con anterioridad. Por consiguiente, los datos asociados al dataset final estarán ya almacenados en PostgreSQL. Sin embargo, si no se ha realizado ese proceso y no se dispone de los datos, en el repositorio del backend existe un script

de inserción de datos, que creará y poblará la base de datos para que esté lista para utilizarla. El comando para ejecutar este script es el siguiente:

```
psql -h 127.0.0.1 -p 5432 -U postgres -d tfm -f postgresql-creation.sql
```

Código 61. Sentencia que permite la creación y poblado de los datos en PostgreSQL.

Muy importante, antes de poder arrancar el código, será necesario crear un archivo, con el nombre "*.env*", dentro del repositorio del backend donde se definan las variables de entorno. Este fichero no se sube a los repositorios de GitHub porque puede contener información sensible como contraseñas u otros elementos secretos. Por tanto, habrá que crear ese archivo con el siguiente contenido:

```
ENVIRONMENT='development'

API_PORT=8000

POSTGRES_DB_HOST='localhost'
POSTGRES_DB_PORT=5432
POSTGRES_DB_USER='postgres'
POSTGRES_DB_PASSWORD=<- contraseña del usuario de postgres ->
POSTGRES_DB_DATABASE='tfm'
```

Código 62. Contenido del fichero .env

Llegados a este punto, ya se tiene todo el entorno listo para la ejecución de los proyectos. Para arrancar el proyecto del backend hay que ejecutar el siguiente código:

```
$> cd ~/tfm/uned-tfm-api
$> nvm use
Found '~/tfm/uned-tfm-api/.nvmrc' with version <16>
Now using node v16.17.1 (npm v8.15.0)

$> npm run dev

> uned-tfm-api@0.5.1 dev
> nodemon -r dotenv/config

[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node -r dotenv/config src/apps/malware-api/index.ts`
[DATABASE] Database connector: POSTGRESQL
[REPOSITORY] Malware evolution repository: POSTGRESQL
[REPOSITORY] File feature repository: POSTGRESQL
[REPOSITORY] Delivery method repository: POSTGRESQL
[REPOSITORY] Malware type repository: POSTGRESQL
⚡[malware-api]: Server is running on port 8000
```

Código 63. Comandos necesarios para arrancar el proyecto backend.

Ahora que ya está el backend levantado, se puede arrancar el proyecto frontend. Para ello hay que ejecutar los siguientes comandos:

```
$> cd ~/tfm/uned-tfm-app
$> nvm use
Found '~/tfm/uned-tfm-app/.nvmrc' with version <16>
Now using node v16.17.1 (npm v8.15.0)

$> yarn workspace malware-webapp-workspace run start
yarn workspace v1.22.19
yarn run v1.22.19
$ ng serve
✔ Browser application bundle generation complete.

Initial Chunk Files              | Names            |  Raw Size
main.js                          | main             |  20.35 MB |
vendor.js                        | vendor           |   3.86 MB |
polyfills.js                     | polyfills        | 357.72 kB |
styles.css, styles.js            | styles           | 253.49 kB |
runtime.js                       | runtime          |  12.97 kB |

                                 | Initial Total    |  24.82 MB

Lazy Chunk Files                 | Names            |  Raw Size
node_modules_echarts_index_js.js | echarts          |   4.45 MB |
src_app_main_main_module_ts.js   | main-main-module | 932.35 kB |

Build at: 2023-06-12T15:31:20.868Z - Hash: fcd551e8884a567d - Time: 32811ms

** Angular Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200/ **


✔ Compiled successfully.
✔ Browser application bundle generation complete.

Initial Chunk Files | Names   | Raw Size
runtime.js          | runtime | 12.97 kB |

6 unchanged chunks

Build at: 2023-06-12T15:31:23.604Z - Hash: dbddff32c0481d2d - Time: 2375ms

✔ Compiled successfully.
```

Código 64. Comandos necesarios para arrancar el proyecto frontend.

Una vez levantados tanto el backend como el frontend, ya se puede utilizar la aplicación, para ello hay que abrir un navegador e ir a la siguiente dirección: `http://localhost:4200`

ANEXO D. DESPLIEGUE EN LA NUBE

A lo largo de este apartado se va a explicar el procedimiento seguido para desplegar el código de la aplicación en un servidor en la nube.

Para poderlo llevar a cabo hace falta como requisitos las siguientes tecnologías:

Tecnología	Versión
Docker	20.10.6 [59]
Docker-compose	0.39.3 [60]
nginx	1.18.0 [61]

Tabla 25. Tabla con los requisitos y versiones necesarios para poder desplegar la aplicación en la nube.

Hay que destacar que el alumno disponía de un servidor contratado en la nube con anterioridad a este proyecto, con un dominio y unos certificados SSL. Todos estos elementos se podrán reutilizarán para este despliegue, en caso de no disponer de ellos, sería necesario el obtenerlos para poder realizar el despliegue con éxito.

En primer lugar, había que instalar todas las tecnologías mencionadas en la tabla 25, para ello se siguieron los tutoriales de las siguientes referencias: [62], [63], [64]. Una vez instalados, se podía proceder al despliegue de la aplicación.

Se desarrolló el siguiente código para poder desplegar la aplicación utilizando Docker y Docker-compose:

```
FROM node:16.10.0

WORKDIR /tfm-malware-api

RUN npm install -g npm@7.24.0
RUN npm install -g pm2

COPY build/ /tfm-malware-api
COPY package.json /tfm-malware-api

RUN npm install --unsafe-perm

RUN mkdir /tfm-malware-api/pm2-logs
RUN chmod -R 755 /tfm-malware-api/pm2-logs

ENV OPENSSL_CONF=/etc/ssl/
```

```
COPY pm2.config.js pm2.config.js
COPY wait-for-it.sh wait-for-it.sh
RUN chmod 777 wait-for-it.sh

EXPOSE 8000
```

Código 65. Código que forma el fichero Dockerfile

```
version: '3.8'

services:
  postgres:
    container_name: tfm-malware-postgres
    image: postgres:latest
    environment:
      POSTGRES_USER: ${POSTGRES_DB_USER}
      POSTGRES_PASSWORD: ${POSTGRES_DB_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB_DATABASE}
    volumes:
      - ./postgresql-creation.sql:/docker-entrypoint-initdb.d/init.sql
      - "tfm-malware-db:/var/lib/postgresql/data"
    ports:
      - "127.0.0.1:5432:5432"
    restart: unless-stopped
    env_file:
      - .env
  server:
    container_name: tfm-malware-api
    restart: always
    build:
      context: .
      dockerfile: Dockerfile
    command: ["./wait-for-it.sh", "postgres:5432", "--", "pm2-runtime", "start", "pm2.config.js"]
    volumes:
      - ../pm2-logs:/tfm-malware-api/pm2-logs
    ports:
      - "127.0.0.1:8000:8000"
    depends_on:
      - postgres
    links:
      - postgres
    env_file:
      - .env
volumes:
  tfm-malware-db:
    external: true
```

Código 66. Contenido del fichero docker-compose.pro.yml

Además de este código, se utilizará la librería de node *pm2 [65]*. Esta librería ejecuta los procesos de node y en caso de que algún proceso se rompiera y terminase de forma forzada, *pm2* lo vuelve a levantar de nuevo de forma automática.

El fichero de configuración necesario para que *pm2* funcione correctamente se puede ver a continuación:

```
module.exports = {
  apps: [
    {
      name: 'tfm-malware-api',
      script: 'node /tfm-malware-api/src/apps/malware-api/index.js',
      error_file: './pm2-logs/tfm-malware-api-err.log',
      out_file: './pm2-logs/tfm-malware-api-output.log'
    }
  ]
};
```

Código 67. Fichero de configuración de pm2.

Para poder desplegar el código, tanto del backend como del frontend, en la nube hay que compilar primero los proyectos, ya que no se puede ejecutar el código fuente sin compilar en la nube.

Se desarrolló un script en bash para la parte backend. Este script compila el código del backend y después ejecuta el fichero *docker-compose* para levantar los contenedores necesarios y desplegar la aplicación backend. El código del script bash se puede ver a continuación:

```
#!/bin/bash

if [ $# -lt 1 ]; then
  echo "Usage: $0 <docker_up | docker_down>"
  exit 1
fi

if [ $1 == "docker_down" ]; then
  docker-compose -f docker-compose.pro.yml down
  exit 1
fi

if [ $1 == "docker_up" ]; then
  # 1. Creamos el contenedor de node 16.10 y compilamos el proyecto.
  docker run --name tfm_malware_node -v $(pwd):/tfm_malware -it node:16.10 bash -c "sh
/tfm_malware/tfm-malware.build.sh"

  # 2. Eliminamos el contenedor y la imagen.
  docker rm tfm_malware_node
  docker rmi node:16.10

  # 3. Desplegamos el proyecto.
  docker-compose -f docker-compose.pro.yml up --build -d
  exit 1
fi
```

Código 68. Script bash que realiza el compilado y despliegue del código del backend.

Tras la ejecución de este script, el backend estará desplegado correctamente. Si se abre una terminal y se ejecutan los siguientes comandos, se podrá ver que se ha desplegado correctamente:

```
$> docker exec tfm-malware-api pm2 logs
[DATABASE] Database connector: POSTGRESQL
[REPOSITORY] Malware evolution repository: POSTGRESQL
[REPOSITORY] File feature repository: POSTGRESQL
[REPOSITORY] Delivery method repository: POSTGRESQL
[REPOSITORY] Malware type repository: POSTGRESQL
⚡[malware-api]: Server is running on port 8000
```

Código 69. Logs que arroja el contenedor del backend tras levantarse.

Ahora hay que compilar el código del frontend, esta operación es más sencilla, basta con ir al proyecto frontend, abrir una terminal dentro del directorio raíz del proyecto y ejecutar los siguientes comandos:

```
$> cd ~/tfm/uned-tfm-app
$> nvm use
Found '~/tfm/uned-tfm-app/.nvmrc' with version <16>
Now using node v16.17.1 (npm v8.15.0)

$> yarn workspace malware-webapp-workspace run build
```

Código 70. Comandos necesarios para compilar el código fuente del frontend.

Tras ejecutar el comando, se habrá generado una carpeta `dist/malware-webapp` dentro del proyecto, el contenido de esta carpeta habrá que copiarlo en el servidor en la nube dentro de esta ruta: `~/www/malware-webapp`

En este punto, ya se dispondría del código compilado y levantado (en caso del backend) dentro del servidor de la nube. Pero faltaría un último paso para terminar el despliegue. Este último paso es habilitar el *nginx* para que todo el tráfico que entre por el endpoint destinado a albergar esta aplicación se redirija al proceso node levantado y que asigne también a ese endpoint el código html compilado del frontend para que se ejecute al acceder a dicha url.

El código de nginx habrá que copiarlo dentro de esta ruta: `/etc/nginx/sites-available`. Aquí habrá que crear un archivo con nombre `master-uned.jebiba.es`, el código que contendrá este fichero será el siguiente:

```
upstream backend_hosts {
  server 127.0.0.1:8000;
}

client_max_body_size 5M;

server {
  listen 443 ssl;
  server_name master-uned.jebiba.es localhost;
```

```
  root /home/jebiba/www/malware-webapp;
  index index.html;

  access_log /var/log/nginx/app-malware-api.access.log;
  error_log /var/log/nginx/app-malware-api.error.log debug;

  ssl_certificate /etc/letsencrypt/live/master-uned.jebiba.es/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/master-uned.jebiba.es/privkey.pem;

  location / {
    try_files $uri $uri/ /index.html;
  }

  location /malware/api {
    proxy_pass http://backend_hosts;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
  }
}

server {
  listen 80;
  server_name master-uned.jebiba.es localhost;
  return 301 https://master-uned.jebiba.es$request_uri;
}
```

Código 71. Contenido del fichero de nginx master-uned.jebiba.es

Ahora solo faltaría indicar que esta nueva configuración de nginx sea activada para que se dé de alta la aplicación y funcione correctamente el despliegue. Para ello hay que crear un enlace simbólico dentro del directorio de nginx `/etc/nginx/sites-enabled` y posteriormente reiniciar el servicio de nginx. A continuación se muestran los comandos necesarios:

```
sudo ln -s /etc/nginx/sites-available/master-uned.jebiba.es /etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

Código 72. Comandos necesarios para habilitar la configuración nueva de nginx.

Tras la ejecución de estos comandos, se podrá ir a la siguiente dirección web y ver la aplicación desplegada:

https://master-uned.jebiba.es