

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN**

TRABAJO FIN DE GRADO

**DISEÑO Y DESARROLLO DE UNA HERRAMIENTA
WEB PARA ANÁLISIS DE MENSAJERÍA
INSTANTÁNEA**

**JAIME CONDE SEGOVIA
ENERO 2023**

TRABAJO DE FIN DE GRADO

Título: DISEÑO Y DESARROLLO DE UNA HERRAMIENTA WEB PARA ANÁLISIS DE MENSAJERÍA INSTANTÁNEA

Título (inglés): DESIGN AND DEVELOPMENT OF A WEB TOOL TO ANALYSE INSTANT MESSAGING

Autor: JAIME CONDE SEGOVIA

Tutor: JUAN FERNANDO SÁNCHEZ RADA

Departamento: Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente: —

Vocal: —

Secretario: —

Suplente: —

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN**

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes



TRABAJO FIN DE GRADO

**DISEÑO Y DESARROLLO DE UNA
HERRAMIENTA WEB PARA ANÁLISIS DE
MENSAJERÍA INSTANTÁNEA**

JAIME CONDE SEGOVIA

ENERO 2023

Resumen

Las aplicaciones de mensajería instantánea son un componente principal de las comunicaciones interpersonales; más aún para personas que no se ven en el día a día. En octubre de 2020, WhatsApp reportaba enviar unos 200 miles de millones de mensajes al día [1] y, a día de hoy, cuenta con más de 2 mil millones de usuarios.[2] Globalmente en segundo lugar en lo referente a descargas, Telegram cuenta con 700 millones de usuarios [3] que envían más de 15 mil millones de mensajes al día. [4]

Según Robin Dunbar, la capacidad del ser humano para mantener relaciones activas es limitada.[5] Conocer el estado objetivo de las relaciones personales con nuestros círculos puede ser de vital importancia para el mantenimiento, conservación y mejora de relaciones fuertes, duraderas y sanas.

El principal objetivo de este trabajo es brindar al usuario una herramienta que ofrezca estos datos objetivos de una forma visual y fácilmente comprensible. Estos datos pueden ayudar a los usuarios a tomar acciones para mejorar sus interacciones. Hemos decidido llamar a la herramienta: *ChatStats*.

Por otro lado, los usuarios cada vez son más conscientes del valor de sus datos y los posibles usos negativos de la recolección de los mismos. La privacidad es una preocupación principal en el ámbito de las conversaciones personales, por lo que ha sido ampliamente considerada durante el desarrollo de este proyecto. Para proteger los datos del usuario, toda la aplicación se envía al cliente, en el que se realizan todas las operaciones necesarias con las conversaciones de manera local.

Además, *ChatStats* es de código libre, permitiendo el acceso al código para su lectura y modificación por parte de usuarios y contribuyentes; bajo la licencia GNU General Public License Version 3 (GPLv3).[6] La aplicación se ha desarrollado haciendo uso de React, *framework* abierto desarrollado principalmente por Facebook.

Con todo, este proyecto presenta el diseño e implementación de una aplicación web que permite al usuario analizar sus conversaciones de WhatsApp y Telegram, obteniendo información sobre sus interacciones sociales y relaciones. Todo esto asegurando la privacidad de los datos, que nunca abandonan el dispositivo, y bajo una licencia de código libre.

Palabras clave: mensajería instantánea, aplicación web, JavaScript, código libre, WhatsApp, privacidad, número de Dunbar.

Abstract

Instant messaging applications are a major component of interpersonal communications; even more so for people who do not see each other on a day-to-day basis. In October 2020, WhatsApp reported sending about 200 billion messages per day [1] and, as of today, it has more than 2 billion users.[2] Globally, in a second place according to application downloads [3], Telegram has 700 million users who send more than 15 billion messages on daily basis. [4]

According to Robin Dunbar, humans capacity to maintain active social relationships is limited.[5] Knowing the objective state of personal relationships with our circles can be of vital importance for the maintenance, preservation and improvement of strong, lasting and healthy relationships.

The main objective of this project is to provide the user with a tool that offers this unbiased data in a visual and easily understandable way. This data can help users take actions to improve their interactions. We decided to name the tool: *ChatStats*.

On the other hand, users are becoming increasingly aware of the value of their data and the possible negative uses of its collection. Privacy is a major concern in the realm of personal conversations, so it has been extensively considered during the development of this project. To protect user data, the entire application is sent to the client, where all necessary operations with conversations are performed locally.

In addition, *ChatStats* is open source, allowing access to the code for reading and modification by users and contributors; under the GNU General Public License Version 3 (GPLv3) license. The application has been developed using React, an open source framework developed mainly by Facebook.

Overall, this project presents the design and implementation of a web application that allows users to analyze their WhatsApp or Telegram chats and gain insights into their social interactions and relationships, all while ensuring data privacy and accessibility under an open-source license.

Keywords: instant messaging, web application, JavaScript, open-source soft-

ware, WhatsApp, privacy, Dunbar's number.

Agradecimientos

Me gustaría expresar un especial agradecimiento a mi tutor, Juan Fernando Sánchez Rada, que en las buenas y en las malas, ha prestado ayuda en todo lo posible, ofreciendo apoyo y soluciones cuando ha sido necesario; en el proyecto y en lo personal.

Agradezco a mis compañeros de la universidad, familia y amigos por acompañarme durante esta etapa de mi vida y arrojar luz allá donde hubo sombra.

A mi compañero y amigo Carlos García-Mauriño Dueñas por alojarme una instancia de ShareLatex en su servidor, por sus consejos, así como por el incondicional apoyo que he recibido por su parte durante todo el proyecto.

A mi compañero y amigo Guillermo García Grao, por apoyar y contribuir en los primeros pasos del desarrollo del código de este proyecto.

También quiero expresar mi agradecimiento al grupo *GSI* en la ETSIT-UPM, que ha hecho posible realizar este trabajo sobre uno de mis primeros proyectos personales.

Por último, agradezco a todos los contribuyentes al código libre del que hace uso este proyecto, dado que sin sus contribuciones este trabajo no sería posible.

“In God we trust. All others must bring data.” - W. Edwards Deming

Índice general

Resumen	I
Abstract	III
Agradecimientos	V
Contenidos	VII
Lista de Figuras	XI
1. Introducción	1
1.1. Contexto y motivación	1
1.2. Objetivos	2
1.3. Estructura del documento	2
2. Tecnologías habilitantes	5
2.1. Tecnologías en el cliente	5
2.1.1. React	5
2.2. Progressive Web App	6
2.2.1. WebAssembly	6
2.2.2. Expresiones regulares	7
2.3. Tecnologías en el servidor	7
2.3.1. GNU/Linux	8
2.3.2. Traefik	8

2.4.	Herramientas de desarrollo	8
2.4.1.	Git	8
2.4.2.	Docker	9
2.4.3.	Software Libre	9
2.4.4.	Tests unitarios	10
3.	Análisis de requisitos	11
3.1.	Introducción	11
3.2.	Casos de uso	12
3.2.1.	Actores del sistema	13
3.2.2.	Instalar aplicación en el cliente	14
3.2.3.	Importar datos en el sistema	14
3.2.4.	Visualización de chat individual	15
3.2.5.	Visualización de chat grupal	16
3.2.6.	Exportar analíticas	17
3.3.	Especificación suplementaria	17
3.3.1.	Reglas de dominio	17
3.3.2.	Requisitos no funcionales	18
3.3.3.	Restricciones	18
4.	Arquitectura	19
4.1.	Introducción	19
4.2.	Arquitectura en el cliente	20
4.3.	Arquitectura en el servidor	22
4.4.	Arquitectura de la aplicación	23
4.4.1.	Ingesta de datos	24
4.4.2.	Agregación de datos	28

4.4.3.	Cálculo de métricas	31
4.4.4.	Visualización	33
4.4.5.	Exportación	34
4.4.5.1.	Descargar visualizaciones	34
4.4.5.2.	Descargar resultados	35
4.4.6.	Tests unitarios	35
5.	Caso de estudio	39
5.1.	Introducción	39
5.2.	Instalar aplicación en cliente	39
5.3.	Importar datos en el sistema	40
5.4.	Visualización de chat grupal e individual	41
5.4.1.	Cálculo de las estadísticas	41
5.4.2.	Visualización de chat grupal e individual	41
5.5.	Exportar analíticas	45
6.	Conclusiones y futuras líneas de trabajo	47
6.1.	Conclusiones	47
6.2.	Objetivos conseguidos	48
6.3.	Futuras líneas de trabajo	49
Apéndice A.	Impacto del proyecto	I
A.1.	Impacto social	I
A.1.1.	Introducción	I
A.1.2.	Descripción de impactos relevantes relacionados con el proyecto . . .	I
A.1.3.	Conclusiones	II
Apéndice B.	Presupuesto económico	III

B.1. Costes	III
Apéndice C. Código	V
C.1. Acceso al código fuente	V
Apéndice D. Expresiones regulares	VII
D.1. Grupos de captura	VIII
Apéndice E. Formato de mensajes exportados por Telegram	XI
Bibliography	XIII

Índice de figuras

3.1. Diagrama UML	12
4.1. Arquitectura general	20
4.2. Arquitectura en el cliente	21
4.3. Arquitectura en el servidor	22
4.4. Arquitectura de la aplicación	24
4.5. Estructura de un componente de visualización	33
4.7. Ejecución de tests unitarios	36
4.6. Ejemplos de los componentes de visualización	37
5.1. Instalación de la aplicación PWA	40
5.2. Importar datos en el sistema	40
5.3. Cálculo de métricas	41
5.4. Visualización de chat grupal	43
5.5. Ejemplo de variación para chat individual	44
5.6. Exportar analíticas	45
D.1. Grupos de captura iOS en color	VII
D.2. Grupos de captura Android en color	VII

Introducción

1.1. Contexto y motivación

Con 14 años compré mi primer teléfono inteligente. Con ello, gran parte de las interacciones con las personas de mi entorno más cercano, tenían lugar a través de aplicaciones de mensajería instantánea: principalmente WhatsApp y Telegram. Por entonces no era consciente de la información que se perdía por estas vías. Con el tiempo me he dado cuenta de la importancia de la información no verbal que se manifiesta en gestos, emociones, entonación y; finalmente, con la acumulación de todos estos factores, puede llegar a deteriorar una relación si no se reacciona correctamente.

Han sido numerosas las ocasiones en las que he sentido la necesidad de comprobar si mi relación con alguna persona cercana estaba decayendo, o se trataba únicamente de un sentimiento sin fundamentos. Es por ello que quise enfocar este trabajo al desarrollo de una herramienta de código libre para el análisis de conversaciones exportadas por aplicaciones de mensajería instantánea. Al tratar información tan personal, la arquitectura estaba clara: todo el procesamiento debe hacerse en el cliente y evitar enviar información a servidores salvo la autorización del usuario final.

1.2. Objetivos

Con este trabajo se pretende diseñar una aplicación web que analice datos de conversaciones de WhatsApp y Telegram; y ofrezca al usuario una visualización de datos relevantes de la misma, tanto generales como de la evolución en el tiempo.

Podemos segregar los objetivos en los siguientes:

- Importar datos conversacionales de aplicaciones de mensajería instantánea.
- Selección y cálculo de estadísticas de conversaciones.
- Diseño y desarrollo de las visualizaciones.
- Integración multiplataforma.
- Aplicación de técnicas de ingeniería de software.

1.3. Estructura del documento

En esta sección describiremos la estructura de este documento, así como una breve descripción de los capítulos. La estructura es la siguiente:

Capítulo 1. Introducción. Se introduce la motivación que ha llevado al desarrollo del proyecto, así como los objetivos que busca alcanzar y resolver. Todo esto son ingredientes para la adecuada resolución de un problema. Asimismo, se busca explicar la estructura del documento.

Capítulo 2. Tecnologías habilitantes. En este capítulo se explicarán las tecnologías utilizadas que permiten la realización del proyecto, fundamentando la elección de las mismas de manera segregada en cliente, servidor y desarrollo.

Capítulo 3. Análisis de requisitos. Se detalla el análisis de requisitos funcionales en casos de uso, así como se recogen los requisitos no funcionales, reglas de dominio y restricciones.

Capítulo 4. Arquitectura. Se introducirá la arquitectura general, entrando en detalle en decisiones de diseño, módulos y unidades lógicas en las que se divide, así como la implementación de las mismas.

Capítulo 5. Caso de estudio. Se expone el caso de estudio completo del sistema, mostrando los casos de uso en detalle. Se explica el funcionamiento completo desde el punto de vista del usuario.

Capítulo 6. Conclusiones y futuras líneas de trabajo. En este capítulo se extraen las conclusiones del proyecto, así como posibles formas de continuar el desarrollo del proyecto tras esta memoria.

Tecnologías habilitantes

En este capítulo exploraremos las tecnologías habilitantes esenciales para el desarrollo e implementación de nuestra aplicación web. Estas tecnologías incluyen los lenguajes de programación, marcos, librerías y herramientas que han sido utilizadas para construir la aplicación. Las tecnologías habilitadoras juegan un papel crucial en el desarrollo de cualquier aplicación de software, y es esencial elegir el conjunto correcto de tecnologías que se alineen con los objetivos y requisitos del proyecto.

2.1. Tecnologías en el cliente

Para el desarrollo del cliente, se han elegido las siguientes tecnologías:

2.1.1. React

Para el desarrollo del cliente web se ha elegido el framework React. React es una librería para el desarrollo de front-end en JavaScript. Es de código libre y permite construir interfaces de usuario mediante la definición de componentes reutilizables. Es impulsado y mantenido por Meta, así como una gran comunidad de individuos y compañías.

Cuenta con la mayor comunidad de desarrolladores frente a sus competidores: AngularJS, VueJS, NextJS. Esto facilita el desarrollo, con mayor cantidad de librerías disponibles y mayor facilidad para solventar errores debido a la cantidad de usuarios.

React y JSX forman un stack tecnológico que nos permite sustituir completamente el stack conformado por HTML, JavaScript y CSS, ganando modularidad durante el desarrollo.

Para modificar el contenido que se muestra al usuario, cuenta con acceso al DOM, que representa la página web como un árbol de nodos que pueden ser modificados con JavaScript.

2.2. Progressive Web App

El mercado de las aplicaciones está segmentado en función a las distintas tiendas de aplicaciones de cada plataforma. Esto hace que el desarrollo multiplataforma sea complejo. Recientemente han ido surgiendo frameworks como Flutter, aunque la comunidad de desarrollo es todavía pequeña.

Es por ello que para este proyecto nos hemos acogido a las PWA, cuyo soporte ha sido desarrollado para navegadores basados en Chromium, así como Safari; mientras Firefox ha decidido no acoger el estándar de la Web abierta. [7]

Mediante un archivo de manifiesto, se detalla el título, descripción, imágenes y acciones que la aplicación puede ejecutar o intervenir una vez instalada, permitiendo interacción con el sistema operativo.

Con ello, nos permite mantener una sola fuente de código mientras podemos ejecutar nuestra aplicación en la mayoría de los clientes web.

Además, en noviembre de 2016 el tráfico web móvil superó al tráfico web de escritorio. Desde entonces, se ha mantenido la distancia ligeramente, por lo que tendría sentido optimizar la aplicación hacia clientes móviles. [8]

2.2.1. WebAssembly

Web Assembly (WASM) define un formato de código binario portable e instrucciones correspondientes a modo de interfaz para facilitar interacciones entre programas y el entorno del host. Se trata de un estándar abierto que apunta a soportar la ejecución de cualquier lenguaje en cualquier sistema operativo.

Para ello, requiere de la integración del soporte de Web Assembly (WASM) por los

navegadores. Los navegadores principales (Chrome, Firefox, Safari y Edge) ya soportan la versión 1.0 de WebAssembly.

WebAssembly nos permite ejecutar programas desde nuestra página web en el host con alto rendimiento, puesto que las instrucciones son precompiladas. Tarda menos en cargar al ser un binario y permite ejecutar órdenes a bajo nivel, obteniendo un rendimiento más cercano al nativo del sistema en el que se ejecuta (con menor número de capas intermedias).

Hemos elegido Rust como lenguaje de desarrollo para los módulos Web Assembly (WASM), por su tendencia reciente de crecimiento, así como por su rendimiento en todo tipo de entornos.

2.2.2. Expresiones regulares

Las expresiones regulares son una secuencia de caracteres y operadores especiales que definen y controlan una búsqueda de patrones y filtros en un texto de destino.

Haremos uso de las mismas para segmentar los mensajes recibidos en texto plano, obteniendo así diferentes grupos de texto que conformarán la fecha del mensaje, la hora, el nombre del contacto y el cuerpo del mensaje. Pese a ser conocidos por ser un problema más que una solución, con el correcto uso de la tecnología, puede simplificar mucho el desarrollo. Esto se debe a que podríamos obtener resultados similares mediante el uso de funciones que separen la cadena de caracteres en los lugares adecuados, eliminen caracteres innecesarios y formen los grupos anteriormente descritos, cosa que aumentaría la complejidad del desarrollo y añadiría unidades lógicas a la lógica de negocio.

2.3. Tecnologías en el servidor

Debido a la sencillez de nuestra lógica de negocio, no contamos con un *back-end* en el servidor, dado que únicamente servimos la aplicación al cliente. Una vez servida la aplicación, todas las operaciones se ejecutan en el cliente. Con ello, se han seleccionado las siguientes tecnologías para el servidor:

2.3.1. GNU/Linux

Pese a no haber dominado el mercado de los escritorios, Linux se encuentra en el 96,3 % de los servidores del mundo [9]. Se trata de un kernel de código libre, por lo que se alinea con este trabajo perfectamente. Junto con los programas asociados y necesarios para un servidor, como puede ser SSH, nos permite operar y administrar el sistema para ejecutar programas y servicios a ofrecer.

2.3.2. Traefik

Un proxy inverso nos permite interceptar peticiones a nuestro servidor y redirigir el tráfico al servicio back-end adecuado. Además, permite la implementación de certificados SSL, que facilita el tráfico seguro en las conexiones externas.

Hemos elegido Traefik frente a alternativas como Nginx o Apache, que, pese a ser contendientes más establecidos, no ofrecen una integración sencilla con Docker. Además, Traefik ofrece integración nativa con Let's Encrypt, autoridad de certificados sin ánimo de lucro que, mediante una prueba, expide certificados para los dominios bajo nuestra propiedad.

2.4. Herramientas de desarrollo

A lo largo del desarrollo del proyecto, hemos hecho uso de las siguientes tecnologías para facilitar el control y versionamiento del código, propiedad intelectual, despliegue de infraestructura y comprobación de errores.

2.4.1. Git

Git es un sistema de control de versiones distribuido, de código libre y gratuito, diseñado para manejar proyectos de cualquier tamaño, independientemente del número de contribuidores, de forma rápida y eficiente. Nos permite versionar cambios de nuestro código, así como recuperar versiones anteriores y desarrollar nuevas funciones en paralelo en diferentes ramas, entre otros.

Además, se trata de un sistema distribuido, por lo que evitamos la centralización del código; lo que nos permite mantener el control de versiones sin necesidad de estar conectados a Internet, ya que contamos con nuestro propio nodo local. Una vez tengamos acceso a Internet, podemos sincronizar nuestro trabajo con el nodo origen u otros.

2.4.2. Docker

Para servir el cliente desarrollado en React, se ha utilizado Docker como tecnología de contenerización y virtualización ligera. Esto nos permitirá aislar nuestra aplicación en una capa superior al sistema operativo, permitiendo ejecutar la misma en cualquier sistema operativo basado en Linux, independientemente de las dependencias que este tenga instalado; siempre y cuando cuente con Docker.

Hemos optado por virtualización ligera para reducir el impacto en el servidor, permitiendo que este varíe sus recursos ocupados en función a la demanda dentro de las capacidades de nuestro servidor.

2.4.3. Software Libre

Durante las fase de desarrollo, se usará Git como sistema de control de versiones, unido a la publicación total del código en repositorios de acceso libre y gratuito, pudiendo definirlo como “Open Source Software” bajo la licencia GNU General Public License v3.0 [6]. Esta licencia nos permite, en resumen:

1. Cualquiera puede copiar, modificar y distribuir este software.
2. Se debe incluir la licencia con todas y cada distribución de este código.
3. Se permite el uso privado de este software.
4. Se permite el uso de este software con fines comerciales.
5. En caso de construir un negocio únicamente de este código, se arriesga a publicar la fuente del resto del código derivado.
6. En caso de modificación, se deben indicar los cambios realizados al código.
7. Cualquier modificación de este código debe ser distribuida con la misma licencia, GPLv3.
8. Este software se provee sin ningún tipo de garantía.
9. Ni el autor del código ni la licencia pueden ser marcadas como responsables de daños producidos por el software.

El código fuente está disponible en Codeberg para consulta, contribuciones y seguimiento de errores. Se ha elegido Codeberg como alternativa de código abierto a otras plataformas

como GitHub, ya que está basada en Gitea. A su vez, existe un repositorio alternativo o *mirror* en Github. Ambos pueden verse en el Apéndice C.

2.4.4. Tests unitarios

A lo largo del desarrollo se han implementado numerosos tests unitarios para comprobar el funcionamiento de las diferentes unidades lógicas que componen la aplicación. Se ha utilizado Jest como librería para estas pruebas, puesto que es utilizada por Meta, que mantiene el marco de React y Jest, haciendo que ambas sean altamente compatibles y cuenten con gran cantidad de documentación.

Análisis de requisitos

3.1. Introducción

Una de las tareas más difíciles en la construcción de un sistema de software es decidir precisamente qué construir. En este capítulo cubriremos la parte del proyecto relacionada con la Ingeniería de Requisitos, aunque únicamente los procesos de captura y análisis de requisitos. Se dejan fuera del capítulo los procesos que continúan; que son: validación, negociación, especificación y gestión. Para ello, hemos optado por una aproximación basada en el modelado organizacional, específicamente la basada en Unified Modelling Language (UML).

La captura de casos de uso en UML involucra la creación de una representación visual de las interacciones entre los actores y el sistema que se modela. También provee una vista clara y comprensiva de la funcionalidad del sistema, así como ayuda a la identificación de requisitos y limitaciones del sistema en el contexto del dominio. Se expone un diagrama de casos de uso en la Figura 3.1.

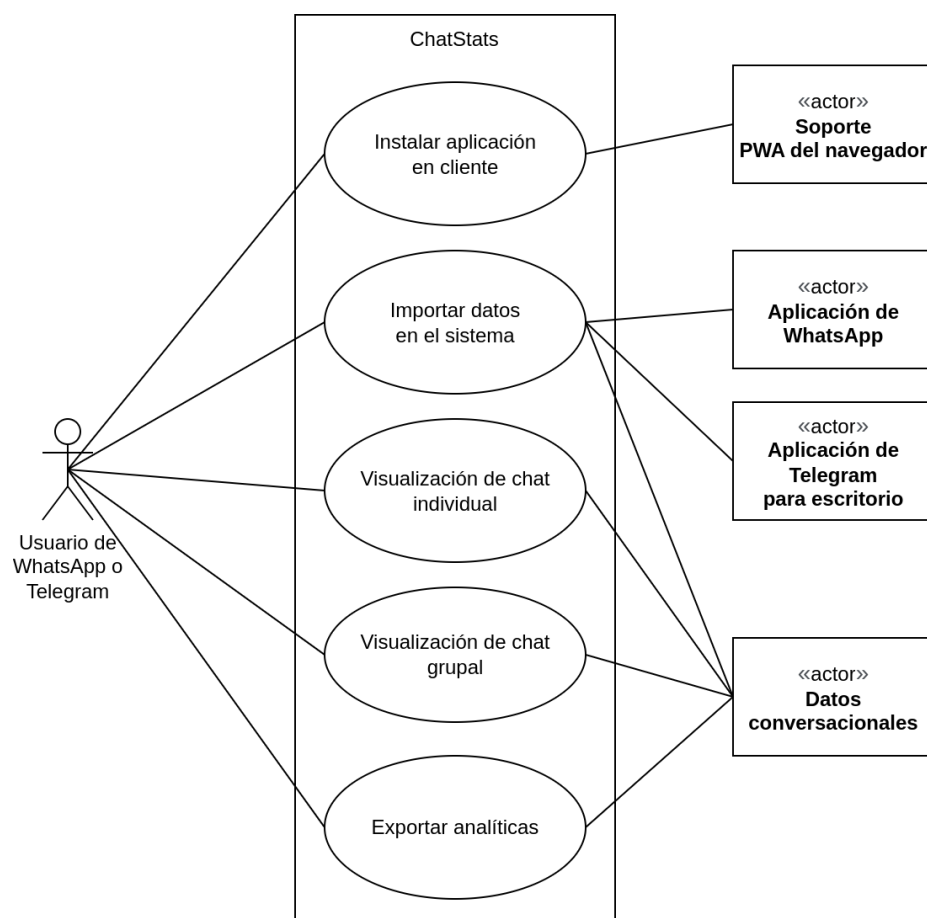


Figura 3.1: Diagrama UML

Finalmente, se expondrá la especificación suplementaria que se ha detectado, tal como las reglas de dominio que se imponen al campo de uso de la herramienta, así como los requisitos no funcionales que se esperan de una aplicación web en el siglo XXI y restricciones impuestas por sistemas externos al nuestro.

3.2. Casos de uso

En UML, los requisitos funcionales se representan como casos de uso. Un caso de uso provee una representación de una funcionalidad específica del sistema y la interacción entre los actores y el sistema. Describen una secuencia de acciones que el usuario o actor deben realizar para conseguir un objetivo o resultado. En esta sección cubriremos los casos de uso expuestos en la figura anterior, así como los actores que aparecen en la misma.

3.2.1. Actores del sistema

Usuario de WhatsApp o Telegram, datos conversacionales, aplicación de WhatsApp, aplicación de Telegram para escritorio y soporte PWA del navegador.

Usuario de WhatsApp o Telegram

Se trata del usuario único en el que se centran nuestros casos de uso. Este es un usuario de WhatsApp o Telegram, ya que ChatStats es compatible con los archivos de chat exportados por estas aplicaciones.

Datos conversacionales

Es el fichero de datos que nuestro sistema espera como entrada. En este caso de uso, se trata de un fichero de texto plano con todos los datos conversacionales exportados por la aplicación WhatsApp o Telegram para un chat individual o grupal, con o sin contenido multimedia.

Aplicación de WhatsApp

Este actor constituye un sistema externo necesario para producir el actor anterior. Sin la aplicación de WhatsApp, ChatStats no tiene ninguna finalidad por sí solo.

Aplicación de Telegram para escritorio

Este actor constituye un sistema externo necesario para producir los datos conversacionales. Sin la aplicación de Telegram para escritorio, ChatStats no tiene ninguna finalidad por sí solo.

Soporte PWA del navegador

Los navegadores con soporte para Progressive Web App (PWA) permiten instalar aplicaciones web, mejorando la integración con el sistema operativo.

3.2.2. Instalar aplicación en el cliente

Nombre del caso de uso Instalar aplicación en el cliente.

Actores Usuario de WhatsApp o Telegram, soporte PWA del navegador.

Resumen El usuario de WhatsApp o Telegram podrá, si su navegador lo soporta, instalar la aplicación de ChatStats para facilitar el acceso a la misma desde su sistema operativo y ofrecer mayor integración con el mismo.

Secuencia de acciones

1. El usuario accede a la página principal de ChatStats.
2. El navegador con soporte PWA le ofrece la opción de instalar la aplicación.
3. El usuario instala la aplicación obteniendo un acceso directo en el escritorio o en el cajón de aplicaciones del navegador.

3.2.3. Importar datos en el sistema

Nombre del caso de uso Importar datos en el sistema.

Actores Usuario de WhatsApp o Telegram, aplicación de WhatsApp, aplicación de Telegram para escritorio, datos conversacionales.

Resumen El usuario de WhatsApp o Telegram exporta un chat de WhatsApp o Telegram, individual o grupal. Posteriormente, el usuario accede a la aplicación, selecciona e importa el archivo de datos conversacionales de WhatsApp. Finalmente, confirma su selección.

Secuencia de acciones

1. El usuario de WhatsApp o Telegram exporta un chat individual o grupal.
2. El usuario accede a la aplicación.

3. El usuario selecciona e importa el archivo de datos conversacionales exportado anteriormente.
4. El usuario observa y confirma la selección.

3.2.4. Visualización de chat individual

Nombre del caso de uso Visualización de chat individual.

Actores Usuario de WhatsApp o Telegram, datos conversacionales.

Resumen El usuario puede observar, analizar e interactuar con las gráficas y estadísticas calculadas para chats individuales.

Secuencia de acciones

1. El usuario de WhatsApp o Telegram inicia el cálculo de estadísticas desde la aplicación.
2. El usuario puede ver el número de mensajes enviado por ambas partes, así como el número de caracteres.
3. El usuario puede ver la media de palabras y caracteres por mensaje.
4. El usuario puede ver quién contesta más rápido los mensajes, así como el tiempo medio de respuesta.
5. El usuario puede ver quién comienza normalmente las conversaciones, así como cuántas conversaciones ha comenzado cada uno.
6. En caso de haber exportado el contenido multimedia, el usuario puede ver el número de fotos, vídeos, audios y pegatinas enviadas por ambas partes.
7. El usuario puede ver tres distribuciones de mensajes: distribución de mensajes por mes, distribución de la media por día de la semana y distribución de la media de mensajes por hora del día.
8. El usuario puede ver las palabras más utilizadas, así como los emoticonos.

Condiciones previas El usuario debe haber realizado anteriormente la exportación de un chat individual con o sin contenido multimedia desde la aplicación de WhatsApp o Telegram. También debe haber realizado la importación de datos en el sistema.

3.2.5. Visualización de chat grupal

Nombre del caso de uso Visualización de chat grupal.

Actores Usuario de WhatsApp o Telegram, datos conversacionales.

Resumen El usuario puede observar, analizar e interactuar con las gráficas y estadísticas calculadas para chats grupales.

Secuencia de acciones

1. El usuario de WhatsApp o Telegram inicia el cálculo de estadísticas desde la aplicación.
2. El usuario puede ver el número de mensajes enviado por cada persona, así como el número de caracteres.
3. El usuario puede ver la media de palabras y caracteres por mensaje.
4. El usuario puede ver quién contesta más rápido los mensajes, así como el tiempo medio de respuesta.
5. El usuario puede ver quién comienza normalmente las conversaciones, así como cuántas conversaciones ha comenzado cada uno.
6. En caso de haber exportado el contenido multimedia, el usuario puede ver el número de fotos, vídeos, audios y pegatinas enviadas por cada usuario.
7. El usuario puede ver tres distribuciones de mensajes: distribución de mensajes por mes, distribución de la media por día de la semana y distribución de la media de mensajes por hora del día.
8. El usuario puede ver las palabras más utilizadas, así como los emoticonos.

Condiciones previas El usuario debe haber realizado anteriormente la exportación de un chat grupal con o sin contenido multimedia desde la aplicación de WhatsApp o Telegram. También debe haber realizado la importación de datos en el sistema.

3.2.6. Exportar analíticas

Nombre del caso de uso Exportar analíticas.

Actores Usuario de WhatsApp o Telegram, datos conversacionales.

Resumen El usuario puede exportar sus analíticas en un fichero, así como exportar una imagen para compartir las visualizaciones obtenidas.

Secuencia de acciones

1. El usuario de WhatsApp o Telegram desciende en la página de visualización.
2. El usuario selecciona la opción de exportar o compartir.
3. El usuario puede compartir la imagen de las visualizaciones obtenidas o guardar el archivo de los estadísticos calculados.

Condiciones previas El usuario debe haber realizado anteriormente la exportación de un chat grupal con o sin contenido multimedia desde la aplicación de WhatsApp o Telegram. También debe haber realizado la importación de datos en el sistema y el cálculo de las estadísticas para la visualización del chat.

3.3. Especificación suplementaria

3.3.1. Reglas de dominio

- **Legales:** Cumplimiento de la Ley General de Protección de Datos Europea.

3.3.2. Requisitos no funcionales

- **Operación:** Interfaz de Usuario para navegador web, smartphone y tablet.
- **Seguridad:** Los datos no deben ser enviados a ningún servidor externo sin la autorización previa del usuario. Estos datos, además, no deben ser almacenados de manera temporal o permanente en ningún servidor. Todas las conexiones entre cliente y servidor deben estar cifradas con SSL.
- **Portabilidad:** La aplicación podrá ejecutarse en los navegadores: Chrome, Firefox, Edge y Safari; siendo compatible con PWA en los navegadores que lo soporten.
- **Rendimiento:** El sistema debe ser capaz de analizar conversaciones de varios años sin aumentar drásticamente el tiempo de espera.
- **De implementación:** El código ha de ser libre bajo la licencia GNU General Public License Version 3 (GPLv3).

3.3.3. Restricciones

- **De interfaz:** uso de archivos *txt* y *zip* para importar los chats exportados desde la aplicación WhatsApp, puesto que únicamente exporta en estos dos formatos. Uso de archivos *JSON* para importar los chats exportados desde la aplicación de Telegram para escritorio.

Arquitectura

4.1. Introducción

En este capítulo trataremos la fase de diseño de este proyecto, así como los detalles de implementación de la arquitectura del mismo. Primero, presentaremos una vista general del proyecto por medio de un diagrama. A continuación se detallará la arquitectura en el cliente, así como la del servidor y los servicios que la componen. Posteriormente, se detallarán los módulos que conforman la aplicación, así como las métricas que se han decidido calcular y sus motivos. Con ello tenemos la intención de facilitar al lector una vista general de la arquitectura del proyecto.

Se presenta a continuación una vista general de la arquitectura de ChatStats:

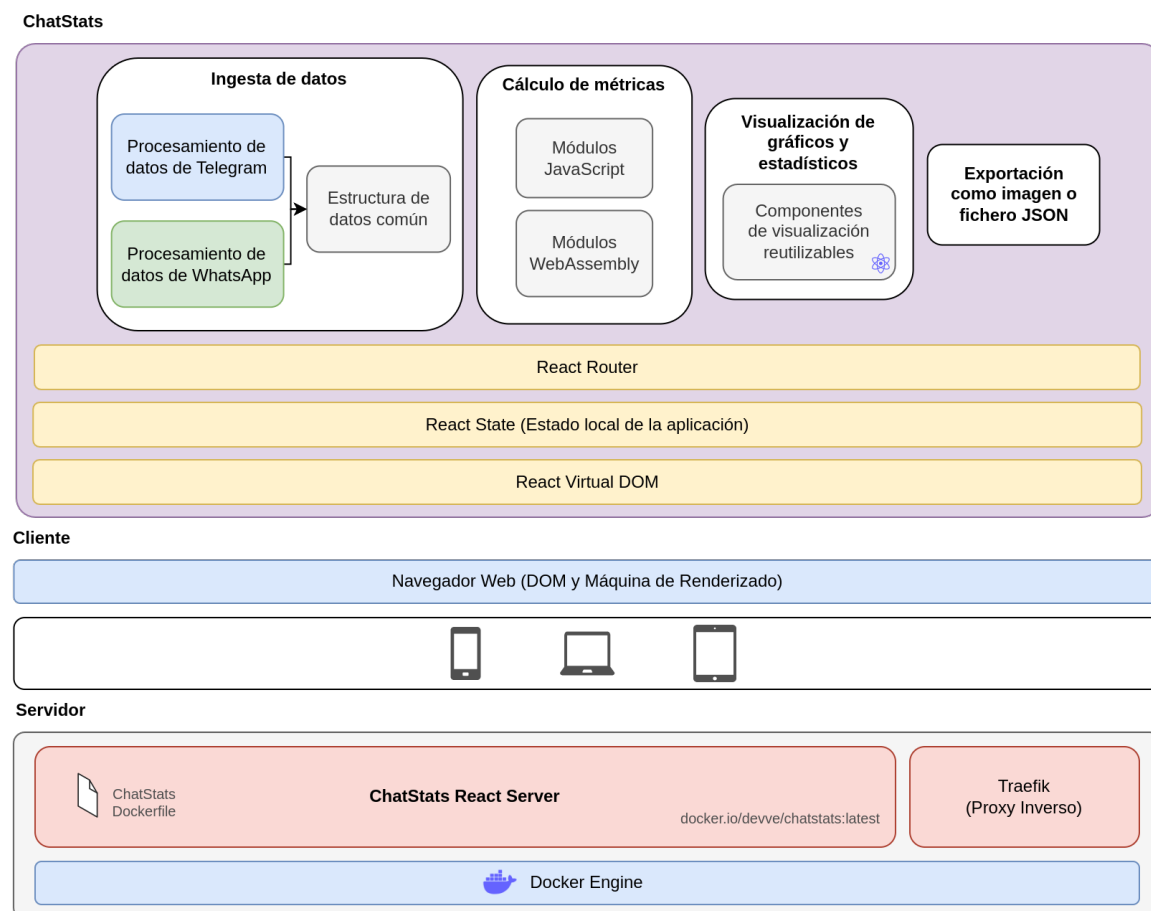


Figura 4.1: Arquitectura general

4.2. Arquitectura en el cliente

A continuación se presenta la arquitectura que podemos encontrar en un cliente cualquiera. Puede tomarse la Figura 4.2 para observar las capas que la componen.

Navegador. El navegador juega un papel fundamental para el acceso y ejecución de cualquier aplicación web. Aunque no se entra en detalle en su estructura interna, sí que vamos a detallar la integración con Progressive Web App (PWA). Esta integración que ofrecen algunos navegadores como los basados en Chromium, permite instalar aplicaciones web, con ventajas como:

- La aplicación saldrá en el escritorio en teléfonos Android e iOS, así como en el cajón de aplicaciones del navegador.

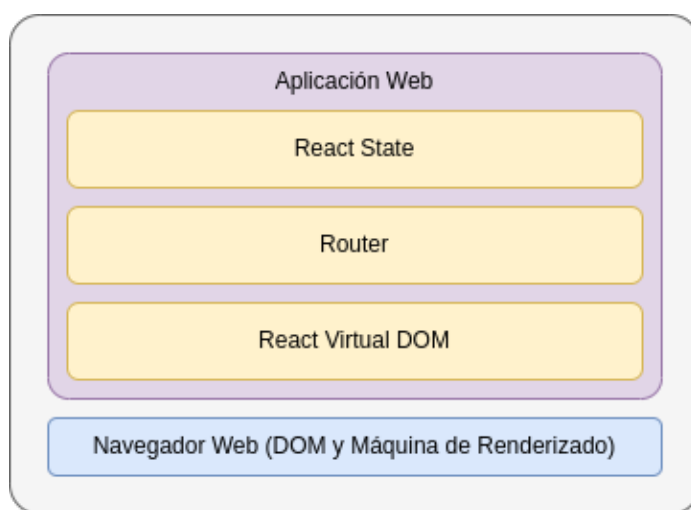


Figura 4.2: Arquitectura en el cliente

- Posibilidad de acceso a notificaciones *push* para el sistema (que no utilizaremos).
- Posibilidad de guardar en *cache* el código del cliente, permitiendo su uso sin conexión a Internet.

Es por ello que ChatStats puede ser instalada en los dispositivos con un navegador con soporte PWA, permitiendo su instalación y ejecución sin acceso a Internet. Para ello, ChatStats registra un *service worker* o trabajador de servicio que guarda el código en el cliente, así como un archivo de manifiesto que recoge un título, descripción e iconos de la aplicación.

Hay que mencionar también que Mozilla no ofrece soporte para PWA en su navegador Firefox[7], aunque este puede ser habilitado mediante una extensión[10].

Aplicación Web. En esta última capa se encuentra nuestra aplicación, cuya arquitectura de procesamiento se explica en la Sección 4.4. Explicamos a continuación los componentes lógicos que se encuentran en el cliente:

React State. Se trata de la capa que maneja el estado de nuestra aplicación, en el que se almacenan los resultados de los cálculos, importación de archivos y otros, como se verá en la Sección 4.4. Permite además que nuestra aplicación reaccione a los cambios sucedidos en esta capa de estado, pudiendo refrescar componentes que hagan uso de la información almacenada si fuese necesario.

Router. ChatStats es una aplicación multipágina. Esto quiere decir que cuenta con diferentes rutas web en las que se muestran diferentes páginas. La página principal consolida la ruta '/', mientras que '/graphs' enruta la página para la visualización de los gráficos. Usamos el enrutador proporcionado por React para permitir la navegación por la aplicación, mapeando las rutas a los componentes de React que se deben visualizar en cada una. Estos se explicarán en la Sección 4.4.

React Virtual DOM. El DOM virtual es un concepto de programación en el que una representación virtual de la interfaz de usuario (UI) es guardada en memoria y sincronizada con el DOM del navegador. Esto nos permite definir qué queremos en la interfaz de usuario y React conseguirá que el DOM virtual y el DOM del navegador se sincronicen.

Por último, cabe añadir que se han implementado reglas de estilo que tienen en cuenta el tamaño de la pantalla y se adaptan al mismo, permitiendo mantener la usabilidad del cliente en teléfonos, tabletas y ordenadores.

4.3. Arquitectura en el servidor

Se muestra a continuación una grafo de la arquitectura en el servidor, donde se muestran las capas que lo componen.

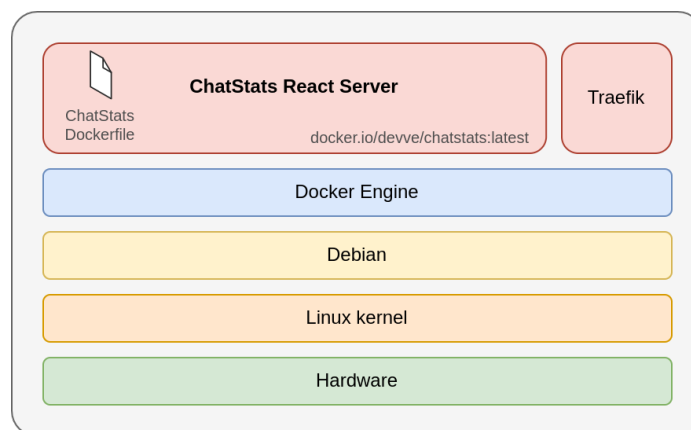


Figura 4.3: Arquitectura en el servidor

Se ha decidido no instalar el software directamente sobre el sistema operativo, evitando problemas de dependencias y distintas versiones de las mismas para los componentes del sistema operativo. Asimismo, se evitan problemas de seguridad que puedan venir por vulnerabilidades en el código fuente y sus dependencias.

Hemos elegido virtualización ligera para ejecutar nuestro código en contenedores, por las razones que se exponen:

- Se contienen las dependencias de terceros en una imagen.
- En caso de vulnerabilidad, solo se expone el contenedor y no el sistema completo.
- Los recursos se ocupan dinámicamente en función a las necesidades, al contrario que con la virtualización completa.
- Permite el despliegue en cualquier sistema operativo compatible con Linux, salvo arquitecturas ARM (que no es frecuente en servidores).

ChatStats React Server. Se trata del servidor de React que sirve el contenido. Tras construir una imagen con la versión de producción a partir del código fuente, este contenedor sirve el contenido estático final, que enviará al cliente completamente cuando este solicite la aplicación web. Se ha implementado por medio de un fichero Dockerfile, que parte de una imagen de *NodeJS*, instala las dependencias y sirve el contenido. Con esta secuencia de instrucciones, se ha creado una imagen para uso en contenedores Docker, que puede encontrarse públicamente en el repositorio de imágenes DockerHub.

Traefik Se ha decidido usar Traefik como *proxy* inverso, que se sitúa frente al servidor de ChatStats para redirigir las peticiones realizadas a su contenedor correspondiente en el puerto adecuado.

Además, Traefik gestiona los certificados SSL haciendo uso de *Let's Encrypt*: autoridad sin ánimo de lucro que provee certificados para la capa TLS sin coste alguno.

4.4. Arquitectura de la aplicación

Con el objetivo de mantener la privacidad de los datos del usuario, se ha planteado una arquitectura centrada en el cliente, donde el servidor únicamente envía la totalidad de la aplicación al cliente en la primera petición. Esto supone un mayor coste computacional en el cliente para realizar todas las operaciones necesarias, por lo que la eficiencia del código es necesaria. Además, esta arquitectura se puede extender, en un futuro, ofreciendo analíticas adicionales si el usuario opta por enviar información al servidor para su procesamiento. Este caso de extensión se detallará más adelante.

A continuación se muestra la arquitectura de la lógica de negocio en la aplicación:

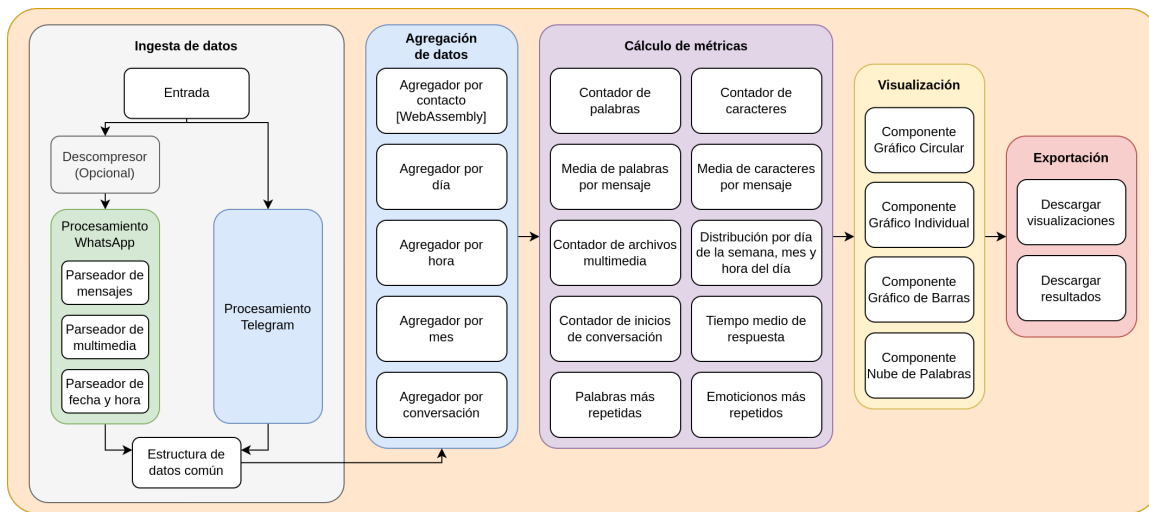


Figura 4.4: Arquitectura de la aplicación

4.4.1. Ingesta de datos

El módulo importador se encarga de leer el fichero de entrada exportado desde la aplicación WhatsApp o Telegram y convertirlos a una estructura de datos normalizada para el procesamiento de los módulos posteriores. Dicho fichero puede estar en tres formatos diferentes:

- *txt* si se trata de un chat exportado de WhatsApp en un dispositivo Android.
- *zip* si se trata de un chat exportado de WhatsApp desde un dispositivo iOS.
- *JSON* si se trata de un chat exportado de Telegram desde la aplicación para escritorio.

Aunque Telegram también soporta la exportación de conversaciones en *HTML*, ChatS-tats no soporta este formato, puesto que está diseñado para la visualización y no para el tratamiento y procesamiento del mismo.

Mientras que Telegram ofrece los datos de exportación en formato JSON, facilitando así la operabilidad de los mismos; WhatsApp no ofrece una estructura de datos ni consistencia entre distintas versiones de su aplicación. Es por ello que, en busca de la reutilización del código y escalabilidad del mismo, este módulo tiene como salida la siguiente estructura de datos:

```
{
```

```
date: new Date("2022-10-17T10:37:00"),
from: "Juan Pedro",
text: "IMG-20221020-WA0013.jpg",
type: "message",
media_type: "image"
}
```

Esta estructura de datos conforma el máximo común divisor de la información que podemos obtener de Telegram y WhatsApp, por lo que es un buen punto de partida.

Además, en una primera instancia, cabe destacar que ChatStats únicamente hace uso de los metadatos del contenido multimedia, tales como formato de archivo, extensión y fecha. Estos datos se incluyen en el fichero que la aplicación espera a la entrada. Esto se detallará más adelante en este mismo módulo.

Entrada

Este submódulo se encarga de cargar el fichero seleccionado por el usuario, que el navegador ofrece desde una ruta virtual y protegida, para que la aplicación no pueda acceder a todos los archivos del dispositivo. En caso de tratarse de un fichero de texto plano, este módulo abre el archivo como una cadena de caracteres. En caso de tratarse de un fichero comprimido, se lo envía al módulo “Descompresor” esperando un archivo de texto plano como salida. Por último, en caso de tratarse de cualquier otro tipo de fichero, el módulo alerta al usuario de que el archivo de entrada no es válido.

Este submódulo hace uso de una instancia de *FileReader*; clase que ofrecen los navegadores, permitiendo abrir el fichero como secuencia de caracteres.

Descompresor (Opcional)

Este submódulo se encarga de la descompresión de ficheros *zip*, que descomprime bajo petición del módulo anterior. Finalmente, el módulo devuelve un fichero llamado *_chat.txt*, que, como se ha comentado anteriormente, contiene toda la información necesaria para el análisis (con o sin multimedia).

El descompresor hace uso de la librería *jszip* para la descompresión del fichero. La lógica añadida a esta aplicación nos permite encontrar el fichero de texto plano dentro del fichero comprimido y devolver el mismo al módulo de “Procesamiento WhatsApp”.

Procesamiento Telegram

El objetivo de este submódulo es transformar el modelo de datos que exporta Telegram a el modelo de datos que utiliza ChatStats para el resto de sus módulos. Un ejemplo del objeto JSON exportado por Telegram puede observarse en el Apéndice E.

En este caso, el submódulo de “Entrada” nos envía una cadena de caracteres en formato JSON. Haciendo uso de la librería *JSON* nativa para *JavaScript*, podemos parsear el contenido de la cadena para obtener el objeto JSON.

Debido a que Telegram ofrece una estructura que puede utilizarse para chats individuales o grupales, además de ofrecer soporte para mensajes multimedia, ChatStats hereda ciertas claves de dicha estructura. Es por ello que este submódulo elimina todas las claves no necesarias del objeto mensaje original de Telegram, quedándonos únicamente con las siguientes claves:

- **“from”**: almacena el nombre del contacto. Admite cualquier secuencia de caracteres.
- **“type”**: almacena el tipo de mensaje. Actualmente su valor es *“message”*, pero se ha considerado útil para futura escalabilidad del proyecto, dando soporte a videollamadas y cambio de nombre de grupo (anuncios), entre otros.
- **“date”**: almacena la fecha y hora del mensaje.
- **“media_type”**: almacena el tipo de contenido multimedia. Puede tomar como valor: *“voice_message”*, *“video_file”*, *“sticker”* o *“image”*

Procesamiento WhatsApp

Este submódulo se encarga, al igual que el submódulo anterior, de obtener mensajes con la estructura descrita al principio del módulo, a partir del archivo exportado de WhatsApp.

A continuación se muestra un mensaje exportado desde WhatsApp por un dispositivo Android:

```
17/07/2022, 01:28 - Alice: Este es un mensaje de prueba.
```

Así como un mensaje exportado desde WhatsApp por un dispositivo iOS:

```
[29/12/22, 0:14:55] Bob: Te escribo desde mi iPhone.
```

Podemos observar que, en el caso de WhatsApp, ambos mensajes se componen de la fecha, la hora, el nombre del contacto y el propio cuerpo del mensaje, aunque con distinto formato.

Parseador de mensajes Se implementa, como puede verse en el Apéndice D un parseador mediante una expresión regular. Dicho parseador encuentra los mensajes en el texto recibido del submódulo de “Entrada” y los divide en: fecha y hora, contacto y cuerpo del mensaje. Podemos observar que no encontramos ninguna referencia al contenido multimedia aún, puesto que para ello hay que parsear el cuerpo del mensaje en busca de metadatos. Además, la fecha y la hora son una cadena de caracteres, que transformaremos a continuación.

Parseador de fecha y hora Este componente tiene como entrada la fecha y la hora del mensaje obtenido en el paso anterior. Devuelve un objeto de tipo *Date*, nativo de JavaScript. Esto nos permite realizar operaciones de tiempo con facilidad en otros módulos.

Este submódulo utiliza la librería *momentjs* para parsear los posibles distintos formatos de fecha que utilizan Telegram y WhatsApp en sus cadenas de caracteres. Además, considera el *locale* del cliente, invirtiendo mes y día para el caso *en_US*.

Parseador de archivos multimedia El objetivo de este componente es escribir el tipo de contenido en la clave “*media_type*” del objeto mensaje expuesto anteriormente, buscando la extensión del archivo adjunto.

Se indican a continuación mensajes de ejemplo para cada tipo de archivo adjunto, únicamente para Android:

```
17/10/2022, 21:11 - Juan Pedro: PTT-20221017-WA0078.opus (file attached)
20/10/2022, 10:37 - Juan Pedro: IMG-20221020-WA0013.jpg (file attached)
14/11/2022, 18:58 - Juan Pedro: VID-20221114-WA0039.mp4 (file attached)
24/11/2022, 19:13 - Jaime Conde: STK-20220717-WA0090.webp (file attached)
```

En el fichero de texto plano con contenido multimedia, observaremos que el cuerpo incluye el nombre del fichero que se ha exportado con la extensión del formato del mismo. Por ello, categorizamos como *voice_message*, *video_file*, *sticker* o *image* en función a la extensión; *.opus*, *.mp4*, *.webp* o *.jpg*, respectivamente. Usamos estos valores puesto que son los que Telegram utiliza para su formato de mensajes.

Si el fichero no incluye estos metadatos, cada vez que un mensaje sea contenido multi-

media, aparecerá *<Media omitted>* (multimedia omitido). Estos pueden ser fotos, vídeos, música, notas de voz o documentos. Se definirán como *undefined* o indefinidos, ignorándose en las visualizaciones y módulos posteriores. Se muestra un ejemplo a continuación:

```
17/07/2022, 01:33 - Juan Pedro: <Media omitted>
```

4.4.2. Agregación de datos

Los mensajes se encuentran segregados en una lista, por lo que a continuación, el módulo de agregador se encargará de agregar los mensajes en diferentes grupos. En el código los hemos llamado polarizadores. Se describen los distintos submódulos a continuación:

Agregador por contacto

ChatStats se encarga de calcular las estadísticas de cada contacto para visualizarlas y mostrarlas en comparación con el resto de contactos. Hablamos de numerosos contactos, puesto que es compatible con chats individuales y grupales.

El resultado de este submódulo será un objeto JSON con una clave por cada contacto (su nombre), que contendrá un array de los mensajes enviados por este. Se indica un ejemplo:

```
{
  "Jaime": [...messagesByJaime],
  "Juan Pedro": [...messagesByJuanPedro],
  ...
}
```

donde los array de mensajes contienen objetos con la estructura de datos común mencionada en el módulo anterior, Subsección 4.4.1.

Cabe anotar que este submódulo está desarrollado en Rust, haciendo uso de la tecnología Web Assembly (WASM); puesto tras medir los tiempos de ejecución de las distintas funciones, se ha comprobado que este era el mayor. Además, sirve como parámetro de entrada para la mayoría de los agregadores que explicaremos a continuación. Se utiliza la librería “*wasm-pack*” de Rust, que genera automáticamente el paquete JavaScript del que hará uso nuestra aplicación en React. Este paquete, una vez construido, exporta las funciones públicas que estén decoradas para su uso en WASM. Estas funciones enlazan a un binario que “*wasm-pack*” construye también por nosotros.

Agregador por día

Este agregador toma como entrada la salida del submódulo anterior: los mensajes agregados por contacto. Con ello se procede a agregarlos, además, por día de la semana: de lunes a domingo. Se usará el nombre del día de la semana como clave anidada.

El resultado son objetos con la siguiente estructura:

```
{
  "Jaime": {
    "monday": [...messagesByJaimeOnMonday],
    "tuesday": [...messagesByJaimeOnTuesday],
    ...,
    "sunday": [...messagesByJaimeOnSunday]
  },
  "Juan Pedro": {
    "monday": [...messagesByJuanPedroOnMonday],
    "tuesday": [...messagesByJuanPedroOnTuesday],
    ...,
    "sunday": [...messagesByJuanPedroOnSunday]
  },
  ...
}
```

El objetivo de esta estructura de datos es visualizar la distribución de los mensajes a lo largo de la semana, en media.

Agregador por hora

Este agregador toma también como entrada los mensajes agregados por contacto. Con ello se procede a agregarlos, además, por hora del día, usando la hora en formato 24 horas como clave anidada de agregación: de 00 a 23 horas.

El resultado son objetos con la siguiente estructura:

```
{
  "Jaime": {
    "00": [...messagesByJaimeAt00],
    "01": [...messagesByJaimeAt01],
    ...,
    "23": [...messagesByJaimeAt23]
  },
  ...
}
```

```
"Juan Pedro": {  
  "00": [...messagesByJuanPedroAt00],  
  "01": [...messagesByJuanPedroAt01],  
  ...,  
  "23": [...messagesByJuanPedroAt23]  
},  
...
```

El objetivo de esta estructura de datos es visualizar la distribución de los mensajes a lo largo del día, en media.

Agregador por mes

Este agregador toma también como entrada los mensajes agregados por contacto. Con ello se procede a agregarlos, además, por MM/YYYY, por lo que deja de tratarse de un agregador acotado: pueden haber tantas claves anidadas como meses se haya hablado.

El resultado son objetos con la siguiente estructura:

```
{  
  "Jaime": {  
    "10/2022": [...messagesByJaimeOnOctober2022],  
    "11/2022": [...messagesByJaimeOnNovember2022],  
    ...  
  },  
  "Juan Pedro": {  
    "10/2022": [...messagesByJuanPedroOnOctober2022],  
    "11/2022": [...messagesByJuanPedroOnNovember2022],  
    ...  
  },  
  ...  
}
```

El objetivo de esta estructura de datos es visualizar la distribución de los mensajes a lo largo del tiempo, con una agregación mensual.

Agregador por conversación

Este submódulo analiza las diferencias de tiempos entre los mensajes, categorizándolos

como inicio de conversación si son el primer mensaje en las últimas 5 horas, o como respuesta si se trata de un intervalo de tiempo menor. Esto nos permitirá calcular cuántas conversaciones ha iniciado cada contacto, así como el tiempo de respuesta medio de cada uno; módulos que se verán más adelante.

Aunque un sesgo temporal no es una solución perfecta, acierta gran parte de las veces sin necesitar gran capacidad de cómputo.

4.4.3. Cálculo de métricas

Este módulo prepara analiza y extrae los datos para ser representados por los componentes del módulo de visualización, que sigue a este. No se realizarán mayores detalles en la implementación, puesto que se han utilizado funciones nativas de JavaScript para su implementación, que puede verse en el código del Apéndice C.

Cabe destacar que, una vez calculadas las métricas, estas pasan al estado local de la aplicación, permitiendo desencadenar acciones en la interfaz de usuario, tales como la representación visual de las métricas por medio del módulo que sigue a este.

Se describen a continuación las métricas elegidas, así como el razonamiento de su elección:

Contador de mensajes Este submódulo cuenta el número de mensajes enviado por cada contacto. Se ha optado por calcular esta métrica puesto que puede ayudar a observar diferencias drásticas en la cantidad de mensajes que aporta cada contacto.

Contador de palabras Este submódulo cuenta las palabras que hay en los mensajes de cada contacto y calcula la suma total de las mismas, obteniendo el número de palabras totales enviadas por cada contacto.

Contador de caracteres Este submódulo cuenta los caracteres que hay en los mensajes de cada contacto y calcula la suma total de los mismos, obteniendo el número de caracteres totales enviados por cada contacto. Aunque suele indicar resultados similares al módulo anterior, en algunas ocasiones es distinto, por lo que se ha decidido incluir también.

Media de palabras por mensaje Este submódulo calcula y muestra el número medio de palabras por mensaje para cada contacto. Esta métrica puede ayudar, junto con el número

de mensajes totales, a saber si un contacto tiende a mandar más mensajes con menor número de palabras, o menos mensajes con más palabras.

Media de caracteres por mensaje Este submódulo calcula y muestra el número medio de caracteres por mensaje para cada contacto. Aunque suele indicar resultados similares al módulo anterior, en algunas ocasiones puede resaltar personas que tienden a usar palabras más largas.

Número de conversaciones iniciadas Este submódulo calcula el número de veces que cada contacto ha iniciado la conversación, con los criterios y datos obtenidos del módulo “Agregador por conversación”. Se ha decidido calcular esta métrica puesto que es una buena forma de medir la iniciativa de una persona, así como su interés en el grupo o persona individual.

Velocidad media de respuesta Este submódulo calcula la velocidad media de respuesta de cada contacto, con los criterios y datos obtenidos del submódulo “Agregador por conversación”. Se ha decidido calcular esta métrica puesto que es una buena forma de medir la atención a la aplicación de mensajería instantánea, así como la importancia que le da al grupo.

Se eliminan todos los mensajes desde que un usuario comienza a responder hasta que termina, para no tenerlos en cuenta en el cálculo del tiempo de respuesta (puesto que se responde a sí mismo a partir del primer mensaje).

Contador de multimedia En caso de que existan objetos JSON con el campo “*media.type*” distinto de *undefined*, este submódulo cuenta cuántos archivos multimedia de cada tipo ha mandado cada contacto.

Generador de estructuras de datos por día, hora y mes Se elige calcular y mostrar distribuciones en los meses puesto que ayudan a observar la evolución en el tiempo de la cantidad de mensajes intercambiados, así como la distribución en los días de la semana ayuda a ver los días más activos en un periodo de una semana. Por último, la distribución en las 24 horas del día ayuda a analizar las horas pico de conversación, así como las horas de inicio y final del día para cada contacto.

Contador de palabras más repetidas Este submódulo procesa todas las palabras de los mensajes, elimina las palabras más comunes del español y el inglés, así como otros mensajes que WhatsApp añade, como *Media ommited* o *This message has been deleted*.

Las listas de palabras más comunes del español e inglés se han recopilado de distintas fuentes, combinado y eliminado repeticiones.

Esta información resulta útil para resaltar los temas principales tratados en el chat.

Contador de emoticonos más usados Este módulo aplica una expresión regular unicode a todos los mensajes, seleccionando los emoticonos y contando el número de veces que aparecen.

Se elige calcular y mostrar esta visualización puesto que los emoticonos constituyen en un chat la forma más similar a la expresión no verbal.

4.4.4. Visualización

El módulo visualizador está formado por un conjunto de componentes visuales reutilizables para distintos tipos de estadísticas, independientemente del formato de las mismas. Para ello, se desarrolla la siguiente estructura, común a todos los componentes:

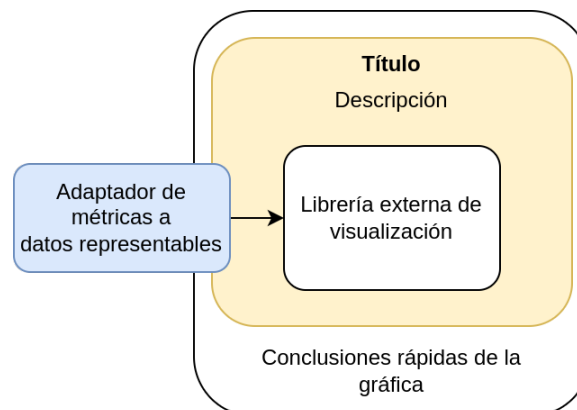


Figura 4.5: Estructura de un componente de visualización

donde el adaptador de métricas a datos representables, asegura que tanto los gráficos circulares como los de barras e individuales, tengan consistencia en los colores utilizados. Además, este adaptador permite eliminar la lógica del cálculo de métricas y del componente de visualización, atomizando, en la medida de lo posible, el código y la arquitectura del proyecto. La estructura de datos que ofrece el módulo a la salida es la siguiente:

```
{
  "labels": ["etiqueta 1", "etiqueta 2", ["valores anidados"]]{
  "datasets": ["valor 1", "valor 2", ["valores anidados"]],
  "backgroundColors": ["color 1", "color 2", ["valores anidados"]],
  "fillColors": ["color 1", "color 2", ["valores anidados"]]
}
```

donde los valores anidados permiten introducir múltiples datos para una única etiqueta.

Los componentes esperan un título y una descripción, un set de datos como el mostrado anteriormente y el máximo y mínimo de los datos representados aportados. Con estos últimos datos, en caso de estar presentes, el componente muestra una frase de conclusión sobre el gráfico mostrado.

Con ello, se han desarrollado los siguientes componentes que se muestran en la Figura 4.6.

Para los gráficos circulares y de barras, se elige la librería *ChartJS*, puesto que cuenta con alta actividad de contribuciones al proyecto, es libre y cuenta con 12 mil estrellas en GitHub. Además, permite alta extensibilidad mediante plugins, de los que hacemos uso, por ejemplo, para mostrar las etiquetas de los datos. Los componentes que usan esta librería adaptan el tamaño del gráfico al número de contactos que hay en el grupo de forma *responsive*. También permiten la interacción con el mismo, pudiendo eliminar contactos de la representación haciendo click sobre el nombre de los mismos en la leyenda.

Por último, como alternativa a los gráficos de barras, se ha planteado también usar un gráfico de araña o radar, pero dicha opción se descarta al observar el solapamiento que tenía lugar con varios contactos.

4.4.5. Exportación

Este módulo permite al usuario exportar tanto la visualización a modo de captura de pantalla, así como la exportación de las métricas calculadas para su uso personal.

4.4.5.1. Descargar visualizaciones

Descargar visualizaciones es un submódulo que permite al usuario descargar como imagen, en formato *png*, las visualizaciones que se muestran en su pantalla.

Como el cliente guarda el estado de las visualizaciones en su DOM, haciendo uso de la librería *html2canvas*, podemos acceder al objeto *body* de la página y obtener una imagen

canvas del mismo. Esta imagen no es una captura de pantalla, sino una reconstrucción del objeto *body* del DOM, por lo que puede no ser completamente fiel a la representación que el usuario tiene en su cliente.

Esta imagen se codifica en base64 y se crea un enlace con tipo de contenido (*Content-Type*) *image/png* para su descarga local desde el navegador.

No se puede hacer uso de la funcionalidad de imprimir pantalla del navegador, puesto que los gráficos usan la etiqueta HTML *canvas*, lo que descuadra los gráficos y hace perder formato y colores.

4.4.5.2. Descargar resultados

Del mismo modo que el submódulo anterior, este módulo permite al usuario descargar una versión exportada de los datos y métricas calculadas en formato JSON para su posterior uso personal.

Para ello, el submódulo recoge todo el contenido alojado en el estado local del cliente y crea un enlace local para la descarga de los datos. Dicho enlace está compuesto por el tipo de contenido (*Content-Type*) *text/json* seguido del contenido del estado local (en JSON) volcado a una cadena de caracteres y codificado para URIs.

Pueden verse las métricas que se descargan en la Subsección 4.4.3.

4.4.6. Tests unitarios

Los tests unitarios son una parte esencial del proceso de desarrollo de software, ya que ayudan a garantizar que el sistema funciona correctamente y que se cumplen los requisitos funcionales. En nuestro proyecto, hemos desarrollado tests unitarios para asegurarnos de que cada los módulos y submódulos de la aplicación cumplen con lo que se espera de ellos..

Además, los tests unitarios nos ayudan a detectar errores temprano en el proceso de desarrollo, lo que nos permite corregirlos antes de que el sistema sea puesto en producción. También nos permiten validar que la funcionalidad no se ven afectadas cuando se realizan cambios en el código.

En este proyecto, se ha alcanzado una cobertura del 74 % en los tests unitarios, lo que significa que el 74 % del código ha sido cubierto por pruebas. Aunque no se alcanza el 100 % de cobertura, se han desarrollado para los módulos más importantes, como “Ingesta de datos”, “Agregadores” y cálculo de la mayoría de métricas. No se han realizado tests

para los componentes de visualización ni el módulo “Exportador”, puesto que suponen una interacción con el *front-end* y se dejan para futuras líneas de trabajo.

Para el desarrollo de los mismos, se utiliza la librería *Jest*, que permite alta integración con la estructura del proyecto de React, así como asienta las bases para futuras pruebas de componentes visuales que interactúen con los mismos, así como con las rutas de la aplicación.

Para generar los distintos datos de entrada se proveen pequeños archivos de chat maquetados para tipo de fichero. Al mismo tiempo, para los datos de entrada a los distintos componentes lógicos que se prueban, se ha utilizado la librería *faker*, que permite la generación de datos maquetados como nombres, fechas, frases, párrafos y otros; lo cual nos da la posibilidad de generar mensajes al vuelo con la estructura de datos deseada.

A continuación se muestra la ejecución de los tests unitarios:

```
λ chatstats main X node node_modules/jest/bin/jest.js --verbose
PASS src/__tests__/polarizers.test.js
  ✓ Aggregate by contacts (2 ms)
  ✓ Aggregate by month (1 ms)
  ✓ Aggregate by day (1 ms)
  ✓ Aggregate by hour (1 ms)
  ✓ Aggregate by conversation (1 ms)

PASS src/__tests__/input.test.js
  ✓ Process an Android .txt chat file (7 ms)
  ✓ Process an iOS .txt chat file (1 ms)

PASS src/__tests__/wordCloud.tests.js
  ✓ Word lists (2 ms)
  ✓ Word repetition (5 ms)

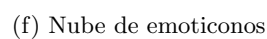
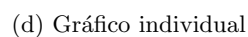
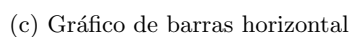
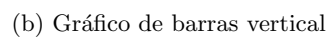
PASS src/__tests__/superString.tests.js
  ✓ Create super string from all messages from each contact (2 ms)

PASS src/__tests__/counters.tests.js
  ✓ Count messages (2 ms)
  ✓ Count words (1 ms)
  ✓ Count characters (1 ms)

PASS src/__tests__/averages.tests.js
  ✓ Word average
  ✓ Character average

Test Suites: 6 passed, 6 total
Tests:      15 passed, 15 total
Snapshots:  0 total
Time:       2.002 s
Ran all test suites.
```

Figura 4.7: Ejecución de tests unitarios



Caso de estudio

5.1. Introducción

En este capítulo vamos a describir los casos de uso principales de ChatStats. Estas descripciones cubrirán todas las funciones principales, así como una descripción detallada de los pasos a seguir y cómo usar la aplicación.

A lo largo de los casos de uso, iremos viendo la aplicación en distintos dispositivos y tamaños, pudiendo comprobar cómo la aplicación se adapta a la pantalla de una forma *responsive*.

5.2. Instalar aplicación en cliente

En esta sección se explicará el proceso de instalación de ChatStats en un dispositivo con navegador con soporte para Progressive Web App (PWA).

En primer lugar, el usuario accede a la web donde se aloja la aplicación. El navegador sugiere la instalación de la aplicación. El usuario pulsa sobre la sugerencia e instala la aplicación, que estará accesible desde el escritorio del dispositivo. La aplicación puede usarse

sin acceso a Internet.

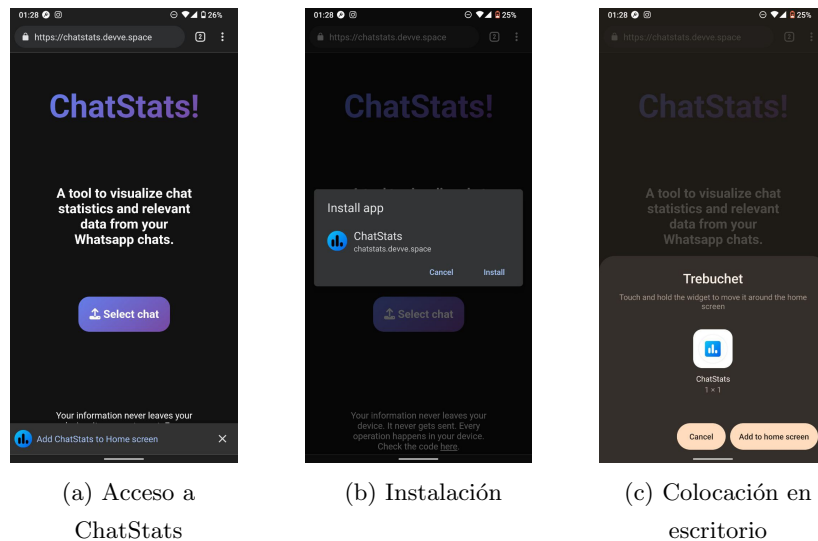


Figura 5.1: Instalación de la aplicación PWA

5.3. Importar datos en el sistema

Durante este caso de uso, el usuario puede seleccionar un archivo para importar a ChatStats. El archivo puede ser de texto plano, *zip* o JSON. En caso de ser otro tipo de archivo, ChatStats alerta al usuario y elimina la selección.

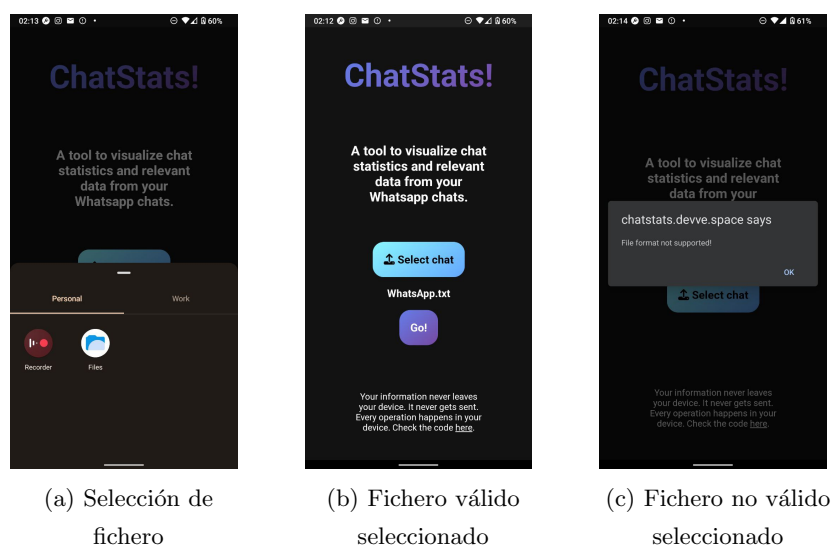
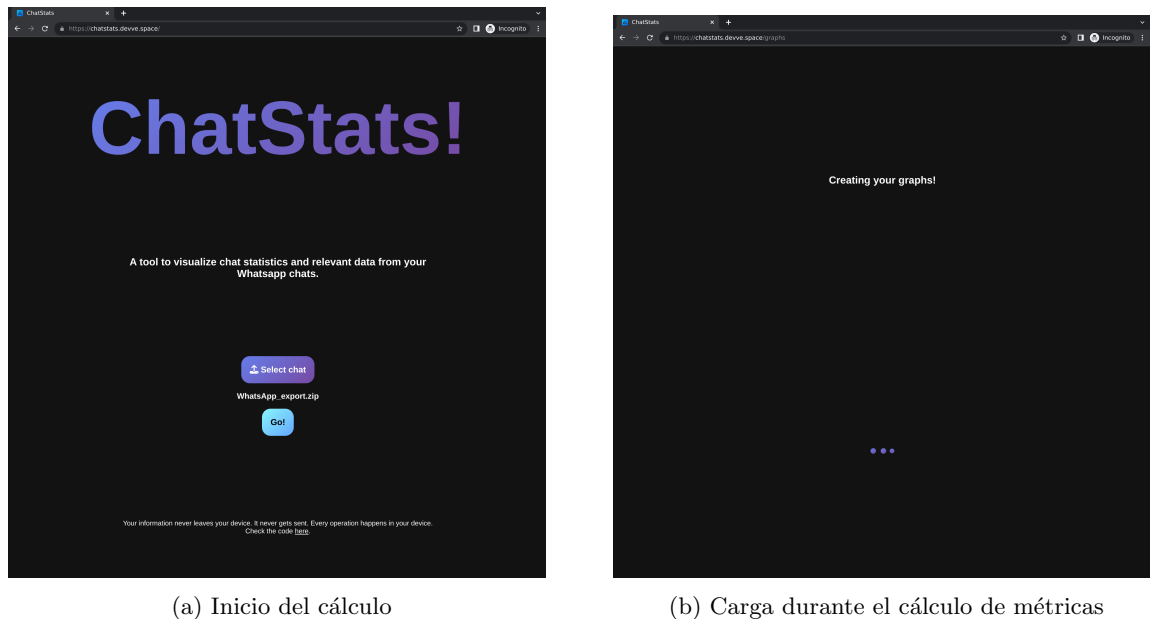


Figura 5.2: Importar datos en el sistema

5.4. Visualización de chat grupal e individual

5.4.1. Cálculo de las estadísticas

Durante esta pantalla, el cliente está ejecutando todos los flujos descritos en el Capítulo 4. Se muestra una animación de carga para hacer saber al usuario que el cliente está realizando operaciones.



(a) Inicio del cálculo

(b) Carga durante el cálculo de métricas

Figura 5.3: Cálculo de métricas

5.4.2. Visualización de chat grupal e individual

En esta página se muestran numerosos gráficos descritos durante la arquitectura, por los que el usuario puede desplazarse mediante la navegación vertical.

En el primer grupo de gráficos de la Figura 5.4 (a) observamos el recuento de mensajes, palabras y caracteres para cada contacto. En este caso, podemos apreciar como los usuarios “Alejandro” y “James”, en azul y morado respectivamente, constituyen casi un tercio de la conversación.

En el grupo de gráficos (b), en la Figura 5.4 podemos observar la media de palabras por mensaje, así como la media de caracteres. De ellos podemos concluir, en este ejemplo, que todos los contactos escriben en media mensajes del mismo tamaño. En este grupo se incluye también un gráfico que indica el número de veces que cada contacto ha iniciado la conver-

sación; así como el tiempo medio de respuesta de cada uno. Estos datos pueden ser muy útiles para medir el interés de los contactos por el chat en cuestión, así como qué contacto tiene mayor iniciativa al diálogo. Para nuestro caso de ejemplo, se observa que “Alejandro” suele comenzar la conversación, así como que Adriana es muy rápida respondiendo.

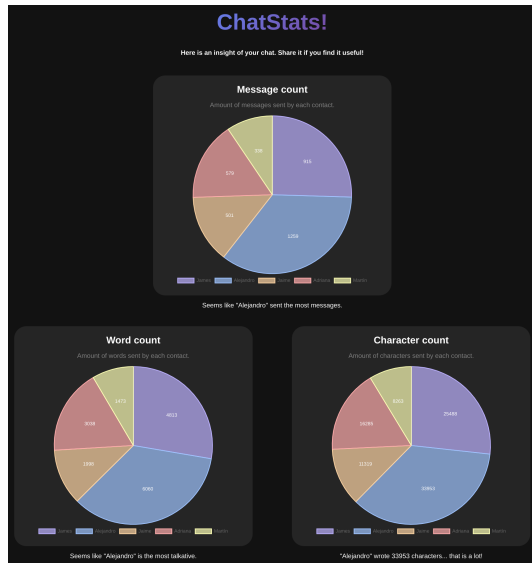
Continuando con el grupo de gráficos (c), podemos observar el número de archivos multimedia de cada tipo que ha enviado cada contacto. Aquí vemos que “Adriana” manda muchas pegatinas o *stickers*, mientras que “Jaime” acostumbra a mandar más audios.

En el grupo de gráficos (d), podemos observar la distribución de los mensajes en los meses, en los días de la semana y en las horas del día. Anotamos que diciembre de 2022 fue el mes más activo del grupo y que los lunes suelen tener mayor actividad (que va decayendo a lo largo de la semana). Cabe comentar que en todos los gráficos anteriores se pueden descartar contactos en la representación. Es por eso que, quedándonos con “James” y “Jaime”, podemos observar que “James” suele comenzar a enviar mensajes una hora antes que “Jaime”.

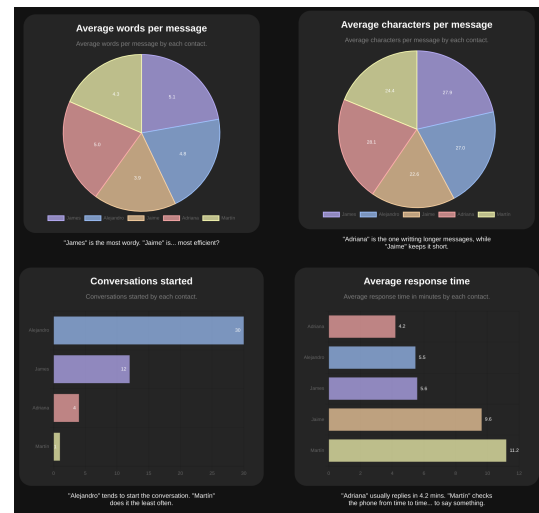
Finalmente, en el grupo de gráficos (e) podemos observar las palabras clave más frecuentes del chat, así como los emoticonos. Estos nos permiten conocer rápidamente los temas principales de conversación, que son, en este caso: *Google*, *GDSC* (*Google Developers Student Club*) y *evento*, entre otros. Asimismo, los emoticonos nos permiten conocer las principales reacciones no verbales que tienen lugar en la conversación.

Para el caso de chats individuales, se elimina la necesidad de la utilización de un gráfico circular y se muestran los números calculados directamente. Los gráficos de barras para las distribuciones en el tiempo no varían. Se muestran algunas diferencias en la Figura 5.5.

5.4. VISUALIZACIÓN DE CHAT GRUPAL E INDIVIDUAL



(a) Número de mensajes, número de palabras y número de caracteres



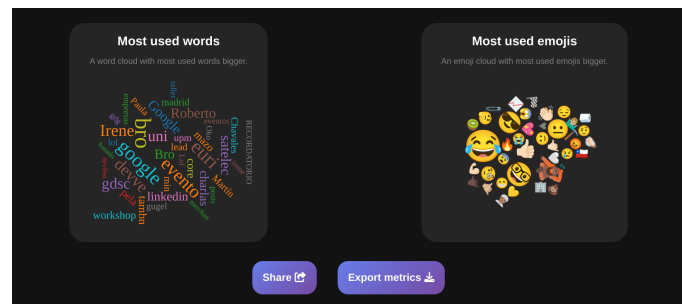
(b) Media de palabras, media de caracteres, conversaciones iniciadas y tiempo medio de respuesta



(c) Número de fotos, vídeos, audios y pegatinas



(d) Distribución de mensajes en meses, días de la semana y horas del día



(e) Nubes de palabras y emoticonos

Figura 5.4: Visualización de chat grupal

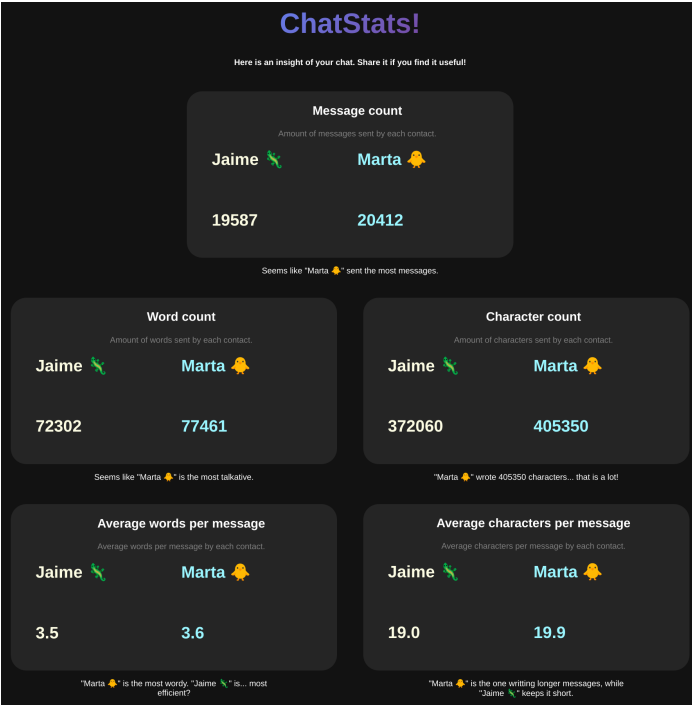
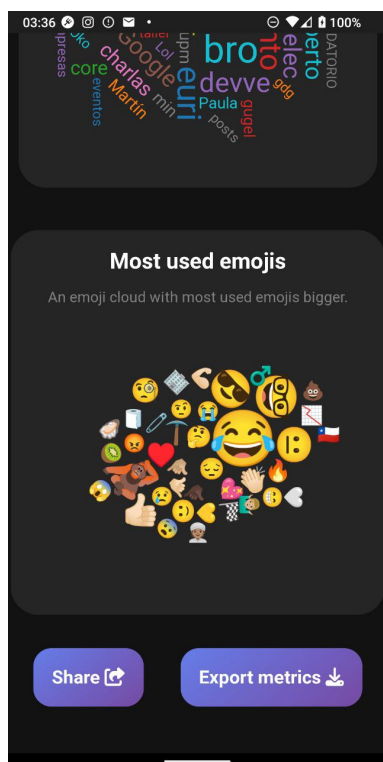


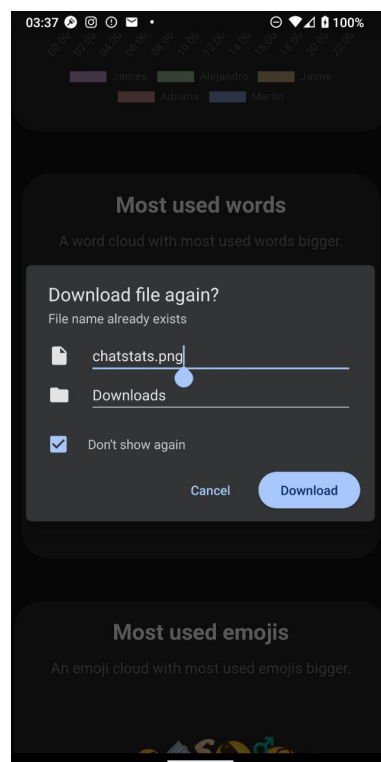
Figura 5.5: Ejemplo de variación para chat individual

5.5. Exportar analíticas

Como puede observarse en la Figura 5.6, se ofrece la opción, al final de la visualización, de exportar una imagen para compartir; así como de exportar las métricas calculadas para su posterior análisis personal.



(a) Compartir imagen



(b) Descarga del archivo exportado

Figura 5.6: Exportar analíticas

Conclusiones y futuras líneas de trabajo

En este capítulo se exponen las conclusiones extraídas del proyecto, así como recomendaciones y posibles futuras líneas de trabajo.

6.1. Conclusiones

Este proyecto ha desarrollado una doble función: personalmente, el desarrollo de un primer proyecto para el mundo real, con casos de uso reales; académicamente me ha permitido profundizar en las técnicas y metodologías de desarrollo de software, concluyendo mi trabajo de fin de grado.

La motivación principal del proyecto era desarrollar una aplicación web para ayudar a analizar conversaciones de WhatsApp y Telegram, permitiendo detectar problemas y tomar acciones para mejorar.

En un primer paso, definimos varios casos de uso para poder definir correctamente nuestro sistema, así como recogimos los requisitos no funcionales del mismo. Con todo lo anterior definido, comenzamos el diseño de la arquitectura del sistema, donde presentamos a grandes rasgos los módulos que la aplicación debía tener. Salvando detalles de implementación,

que pueden observarse en el Apéndice C, hemos encontrado dificultades técnicas como el aprendizaje de Rust para el desarrollo de módulos en Web Assembly (WASM), así como la integración de WASM con el código en JavaScript. Durante esta fase de desarrollo, hemos implementado buenas prácticas de ingeniería de software, como el control de versiones, la escritura de software modular y reutilizable, y tests. Finalmente, se ha expuesto el caso de uso del sistema completo, comprobando el cumplimiento de los requisitos funcionales y no funcionales.

Personalmente, como usuario de la aplicación, este proyecto ha servido para mejorar amistades y relaciones de pareja, así como ayudado a comprender mejor el comportamiento de mis grupos de amigos y seres queridos.

6.2. Objetivos conseguidos

Finalmente, hemos logrado alcanzar todos los objetivos propuestos en el Capítulo 1. Se exponen a continuación en mayor detalle:

Importar datos conversacionales de aplicaciones de mensajería instantánea. Se ha desarrollado un módulo que permite importar datos conversacionales de distintas aplicaciones de mensajería instantánea; particularmente WhatsApp y Telegram, asentando las bases para la inclusión de más aplicaciones dentro del mismo módulo. Además, se pueden importar datos conversacionales de chats grupales e individuales en ambas aplicaciones.

Selección y cálculo de estadísticas de conversaciones. Se han seleccionado e implementado el cálculo de métricas tales como el número de mensajes enviados por cada contacto, número de palabras y caracteres, así como media de palabras y caracteres por mensaje. También se han incluido métricas del tiempo medio de respuesta de cada contacto y número de conversaciones que ha iniciado; así como distribuciones de los mensajes por meses, días de la semana y horas del día, que permiten ver al usuario la evolución de la conversación en el tiempo y el comportamiento semanal y diario de sus chats. Finalmente se incluyen métricas de las palabras y los emoticonos más utilizados.

Diseño y desarrollo de las visualizaciones. Se han diseñado y desarrollado visualizaciones interactivas para mostrar las métricas mencionadas en el objetivo anterior mediante un diseño reutilizable. Tales visualizaciones incluyen gráficos circulares, gráficos de barras, nubes de palabras y frases con las estadísticas resaltadas.

Integración multiplataforma. Hemos considerado una integración absoluta con todos los dispositivos con navegador web, permitiendo el uso en dispositivos móviles, tabletas y ordenadores; permitiendo la instalación como PWA en los navegadores compatibles. Con esto ofrecemos una sensación de mayor integración con el dispositivo, así como la capacidad de funcionamiento sin conexión a Internet. Todo esto es posible gracias a un diseño responsivo.

Aplicación de técnicas de ingeniería de software. Durante todo el proyecto se han escrito numerosos tests para comprobar el correcto funcionamiento del código, así como se ha usado un sistema de control de versiones durante la fase de desarrollo. También ha constituido la consulta exhaustiva de la documentación de numerosas librerías de JavaScript, tales como React.

6.3. Futuras líneas de trabajo

Con vistas a futuro, hemos asentado una base sólida sobre la que construir y escalar este proyecto. Es por ello que, para futuras contribuciones, se plantean las siguientes mejoras que han quedado fuera de cota para este trabajo:

- Migración de los módulos con mayor carga de trabajo a Web Assembly (WASM), permitiendo obtener un rendimiento mayor, cercano al nativo del cliente que ejecuta la aplicación.
- Mayor facilidad para añadir módulos, mediante un sistema de carpetas y plugins.
- Inclusión de visualizaciones para el análisis de sentimientos y su evolución en el tiempo.
- Detección de grupos de interacción: qué contactos suelen responder entre ellos.
- Nuevas métricas: contador de mensajes eliminados por cada contacto.
- Mejora de la documentación para futuros contribuyentes al proyecto.
- Aumento de la cobertura de tests unitarios, así como implementación de tests de extremo a extremo (*end-to-end*) para cada caso de uso.

Impacto del proyecto

A.1. Impacto social

A.1.1. Introducción

Este proyecto aporta herramientas para el estudio y análisis de las relaciones personales llevadas a cabo mediante las aplicaciones WhatsApp y Telegram, permitiendo evaluar y mejorar y tomar decisiones las mismas mediante la observación de los resultados.

A.1.2. Descripción de impactos relevantes relacionados con el proyecto

El cuidado de las comunicaciones que tienen lugar a través de WhatsApp puede ayudar a las personas a dedicar el tiempo necesario a mantener relaciones saludables y darse cuenta de posibles fallos que están teniendo lugar en la comunicación a largo plazo, o analizar su comportamiento.

Otro impacto relevante a la ética del proyecto es las consideraciones de privacidad en un tema tan importante como son las conversaciones privadas del usuario, que se ha considerado prioritario durante las decisiones de diseño y arquitectura.

A.1.3. Conclusiones

Las consideraciones sociales de este proyecto son el núcleo del desarrollo del mismo, así como la causa por la que el proyecto comenzó en un primer lugar.

Presupuesto económico

B.1. Costes

COSTE DE MANO DE OBRA (coste directo)		
Horas	Precio/hora	Total
360	12 €	4,320.00 €

COSTE DE RECURSOS MATERIALES (coste directo)				
	Precio de compra	Uso en meses	Amortización (en años)	Total
Portátil personal	1,500.00 €	6	5	150.00 €
Tablet	1,099.00 €	6	5	109.90 €
COSTE TOTAL DE RECURSOS MATERIALES				259.90 €

APÉNDICE B. PRESUPUESTO ECONÓMICO

GASTOS GENERALES (costes indirectos)	15 %	de CD	686.98 €
BENEFICIO INDUSTRIAL	6 %	of CD+CI	316.01 €

MATERIAL FUNGIBLE	
Impresión y encuadernación	50.00 €

PRESUPUESTO SUBTOTAL		5,632.89 €
IVA APLICABLE	21 %	1,182.91 €

PRESUPUESTO TOTAL	6,815.80 €
--------------------------	------------

C.1. Acceso al código fuente

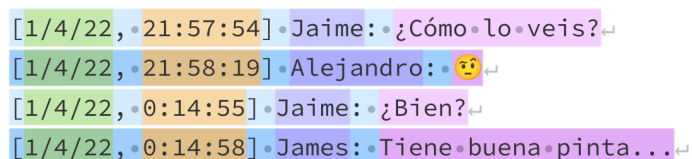
Se enlaza a continuación el repositorio donde reside el código desarrollado durante el proyecto. Se entregan dos enlaces puesto que, con el uso de Git, se ha descentralizado en dos nodos, añadiendo redundancia y disponibilidad a la fuente.

Acceso principal: <https://codeberg.org/devve/chatstats>

Copia (mirror): <https://github.com/d3vv3/chatstats>

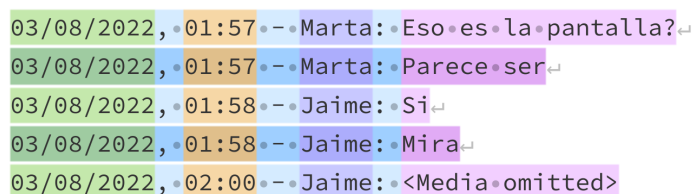
Expresiones regulares

En este anexo se exponen los distintos grupos de captura que definen la expresión regular para el parseo de chats exportados por WhatsApp, tanto desde Android como iOS.



```
[1/4/22, 21:57:54] • Jaime: • ¿Cómo lo veis? •
[1/4/22, 21:58:19] • Alejandro: • 😊 •
[1/4/22, 0:14:55] • Jaime: • ¿Bien? •
[1/4/22, 0:14:58] • James: • Tiene buena pinta... •
```

Figura D.1: Grupos de captura iOS en color



```
03/08/2022, 01:57 -- Marta: • Eso es la pantalla? •
03/08/2022, 01:57 -- Marta: • Parece ser •
03/08/2022, 01:58 -- Jaime: • Si •
03/08/2022, 01:58 -- Jaime: • Mira •
03/08/2022, 02:00 -- Jaime: • <Media omitted> •
```

Figura D.2: Grupos de captura Android en color

D.1. Grupos de captura

Como se describe en el Capítulo 4, poder separar cada mensaje se ha llegado a la siguiente expresión regular, cuyos grupos se explicaran a continuación:

```
/\[*(\d{1,2}\/\d{1,2}\/\d{2,4}),\s(\d{1,2}:\d{2}:\d*)\]*\s(?:-\s)*(.*)
:\{1}\s(.*) (?:=\s\[*(\d{1,2}\/\d{1,2}\/\d{2,4})|$\)/gum
```

Se denotan los distintos grupos de captura por las agrupaciones realizadas con los paréntesis. Se exponen:

Grupo de captura 1: fecha

```
(\d{1,2}\/\d{1,2}\/\d{2,4})
```

Se encarga de la fecha en formato *dd/mm/YYYY* para Android y *d/m/YY* para iOS; denotado con “ $\d\{X,Y\}$ ” que indica que se buscan entre *X* e *Y* dígitos de 0 a 9 seguidos, separados por un “/”. En las expresiones regulares hay que escapar los “/” o *slash* con un “\” o *backslash*.

Durante numerosas pruebas, se ha observado que no hay consistencia entre los ajustes del parámetro *locale* de *en-US* y *es-ES*, siendo *mm/dd/YYYY* y *dd/mm/YYYY* respectivamente. Para ello, ChatStats accederá al *locale* para actuar en consecuencia más adelante. No se ha probado para otras configuraciones.

Puede observarse en verde en la Figura D.2 y Figura D.1.

Grupo de captura 2: hora

```
(\d{1,2}:\d{2}:\d*)
```

Se encarga de la hora en formato *hh:MM* para Android y *h:M:ss* para iOS, donde en caso de comenzar por 0, este no se muestra. Es por ello que se esperan entre 1 y 2 dígitos para la hora y los minutos, así como el parámetro de los segundos es opcional. Este último parámetro, si existe, siempre está conformado por dos números.

Puede observarse en naranja en la Figura D.2 y Figura D.1.

Grupo de captura 3: contacto

```
(.*?)
```

Se encarga del nombre del contacto. Busca la repetición de caracteres ilimitados a excepción del carácter “:”, ya que éste es un separador.

Puede observarse en azul oscuro en la Figura D.2 y Figura D.1.

Grupo de captura 4: mensaje

```
(.*?)
```

Se encarga del cuerpo del mensaje. Busca la repetición de caracteres ilimitados, incluyendo caracteres unicode para tener los emoticonos en cuenta. Esta búsqueda de caracteres se realiza de manera perezosa, expandiendo las coincidencias en caso posible, siempre que no coincida con el siguiente grupo de captura (*lookahead*).

Puede observarse en rosa en la Figura D.2 y Figura D.1.

Look ahead o mirada hacia delante

```
(?=\s\[*\d{1,2}\\/\d{1,2}\\/\d{2,4}|$)
```

Si únicamente contáramos con el grupo de captura 4, solo se reconocería el primer mensaje, puesto que se reconocería el resto del texto como cuerpo del primer mensaje. Para solucionarlo, en el grupo de captura 4 se intentan reconocer el menor número posible de coincidencias, hasta el siguiente patrón reconocido. Este patrón es una mirada hacia delante conformada por la misma expresión regular que en el grupo de captura 1.

Opciones

```
\gum
```

Finalmente se aplican tres opciones a la expresión regular, que son:

- **g:** Nos permite no terminar la búsqueda de patrones tras la primera coincidencia.
- **u:** Nos permite la búsqueda de caracteres unicode, así como emojis (que pertenecen a la especificación unicode).
- **m:** Nos permite realizar la búsqueda en numerosas líneas, pudiendo encontrar coincidencias con mensajes de varias líneas.

Formato de mensajes exportados por Telegram

Se expone a continuación un breve ejemplo del formato utilizado por Telegram para la exportación de sus chats, que es el mismo para chats individuales y grupales.

```
{
  "name": "group_or_contact_name",
  "type": "private_group_or_private_chat",
  "id": 00000000,
  "messages": [
    ...
    {
      "id": 11111,
      "type": "message_service",
      "date": "2019-04-02T20:53:14",
      "date_unixtime": "1554231194",
      "from": "Alice",
      "from_id": "contact_user_id",
      "text": "actual_body_message",
      "text_entities": [
        {
          "type": "plain_link_or_document",
```

```
        "text": "actual_body_message"
    }
]
},
{
    "id": 22222,
    "type": "message",
    "date": "2019-05-29T23:46:00",
    "date_unixtime": "1559166360",
    "from": "Bob",
    "from_id": "contact_user_id",
    "reply_to_message_id": 59386,
    "file": "stickers/sticker.webp",
    "thumbnail": "stickers/sticker.webp_thumb.jpg",
    "media_type": "sticker",
    "sticker_emoji": "actual_unicode_emoji",
    "width": 512,
    "height": 341,
    "text": "",
    "text_entities": []
}, ...
]
```


Bibliografía

- [1] Manish Singh. “WhatsApp is now delivering roughly 100 billion messages a day”. <https://techcrunch.com/2020/10/29/whatsapp-is-now-delivering-roughly-100-billion-messages-a-day/>. Online; accedido a 28 de diciembre de 2022.
- [2] Daniel Ruby. “Whatsapp Statistics 2023 — How Many People Use Whatsapp”. <https://www.demandsage.com/whatsapp-statistics/>. Online; accedido a 28 de diciembre de 2022.
- [3] Manish Singh. “Telegram tops 700 million users, launches premium tier”. <https://techcrunch.com/2022/06/19/telegram-tops-700-million-users-launches-premium-tier/>. Online; accedido a 16 de enero de 2023.
- [4] The Telegram Team. “Telegram tops 700 million users, launches premium tier”. <https://telegram.org/blog/15-billion>. Online; accedido a 16 de enero de 2023.
- [5] “Número de Dunbar”. https://es.wikipedia.org/wiki/N%C3%BAmero_de_Dunbar. Online; accedido a 11 de enero de 2023.
- [6] Inc. Free Software Foundation. “GNU General Public License Version 3”. <https://www.gnu.org/licenses/gpl-3.0.en.html>. Online; accedido a 28 de diciembre de 2022.
- [7] Jared Newman. “Firefox just walked away from a key piece of the open web”. <https://www.fastcompany.com/90597411/mozilla-firefox-no-ssb-pwa-support>. Online; accedido a 11 de octubre de 2022.
- [8] Pablo G. Bejerano. “El tráfico web procedente de smartphones ya supera al de ordenadores”. <https://blogthinkbig.com/el-trafico-web-procedente-de-smartphones-ya-supera-al-de-ordenadores>, 2014. Online; accedido a 14 de octubre de 2022.
- [9] Barry Elad. “Linux Statistics 2022 – Market Share, Usage Data and Facts”. <https://www.enterpriseappstoday.com/stats/linux-statistics.html>. Online; accedido a 10 de octubre de 2022.
- [10] Filip Š. “Progressive Web Apps for Firefox”. <https://addons.mozilla.org/en-US/firefox/addon/pwas-for-firefox/>. Online; accedido a 11 de enero de 2023.