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 CHATS DE MENSAJERÍA INSTANTÁNEA

JAIME CONDE SEGOVIA

ENERO 2023

TRABAJO DE FIN DE GRADO

Título:	DISEÑO Y DESARROLLO DE UNA HERRAMIENTA WEB PARA ANÁLISIS DE CHATS DE MENSAJERÍA INSTANTÁNEA
Título (inglés):	DESIGN AND DEVELOPMENT OF A WEB TOOL TO ANALYSE INSTANT MESSAGING CHATS
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 CHATS 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 ofrece 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: instante 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

Índice de figuras

CAPÍTULO 1

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 aplicaciónes 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 detallan el análisis de requisitos, casos de uso.

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 ha escogido un caso de uso para analizar 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.

CAPÍTULO 2

Tecnologías habilitantes

Escribir pequeña introducción o resumen del capítulo

2.1. Tecnologías en el cliente

Escribir pequeña introducción

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

Meter algo del lenguaje utilizado en WASM

WebAssembly 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 WebAssembly 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 intermediarias).

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

Pequeña introducción

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

Actualizar y cálcular el %

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, y se ha implementado una cobertura del 85 % de las unidades lógicas de la lógica de negocio. La decisión de utilizar esta librería tiene en cuenta su compatibilidad con React y las facilidades que ofrece durante el desarrollo.

CAPÍTULO 3

Análisis de requisitos

3.1. Introducción

El siguiente diagrama UML representa un resumen visual de los casos de uso que se describirán a continuación:

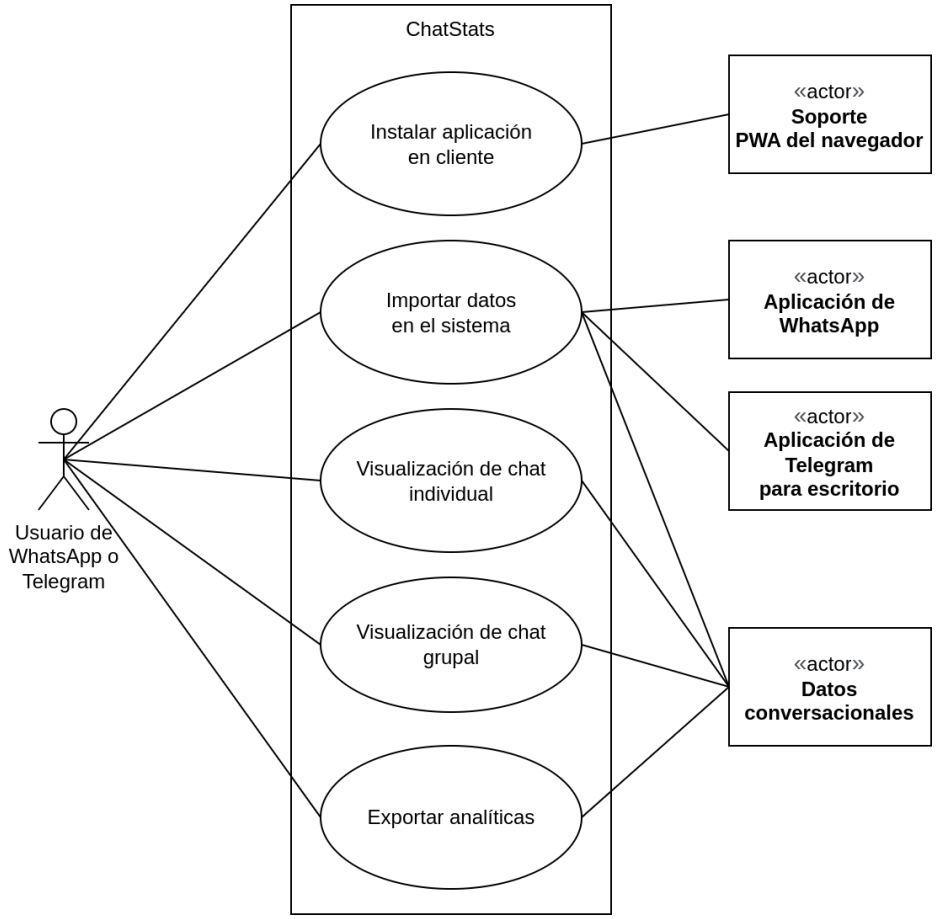


Figura 3.1: Diagrama UML

3.2. Casos de uso

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.

CAPÍTULO 4

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 escenario. A continuación se mostrará 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 el escenario principal de uso de ChatStats:

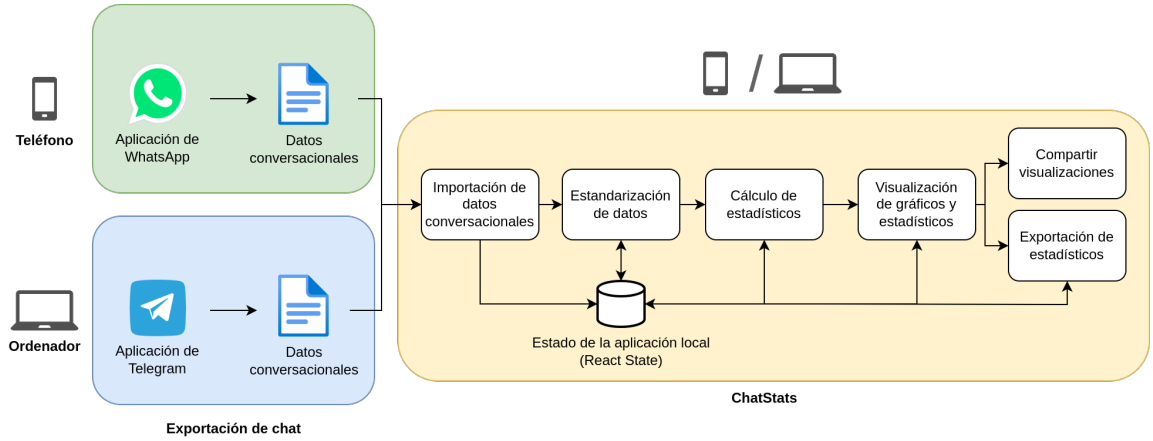


Figura 4.1: Escenario de ChatStats

El servidor únicamente sirve la página... ¿Debería meterlo aquí? ¿Cómo?

Como se puede observar, el usuario debe exportar un chat desde la aplicación de WhatsApp (móvil) o Telegram para escritorio. Son estas las únicas versiones de las aplicaciones que permiten exportar las conversaciones. Con estos datos exportados, el usuario accede a ChatStats desde el navegador, ya sea en un dispositivo móvil o en un ordenador. El servidor le envía el cliente completo, por lo que no intercambia más peticiones en adelante. El usuario puede seleccionar e importar el archivo a la aplicación, que los estandariza, calcula métricas sobre la conversación y ofrece una visualización de los mismos. Finalmente, el usuario puede compartir las visualizaciones, así como exportar los datos calculados para su uso personal.

Se desarrolla toda la aplicación en *JavaScript*, puesto que puede ejecutarse en todos los navegadores salvo que lo tengan deshabilitado. Otros lenguajes como *PHP* o Python requieren de un servidor para realizar operaciones y que estos envíen una plantilla rellena con los resultados.

4.2. Arquitectura en el cliente

A continuación se presenta la arquitectura que podemos encontrar en un cliente cualquiera:

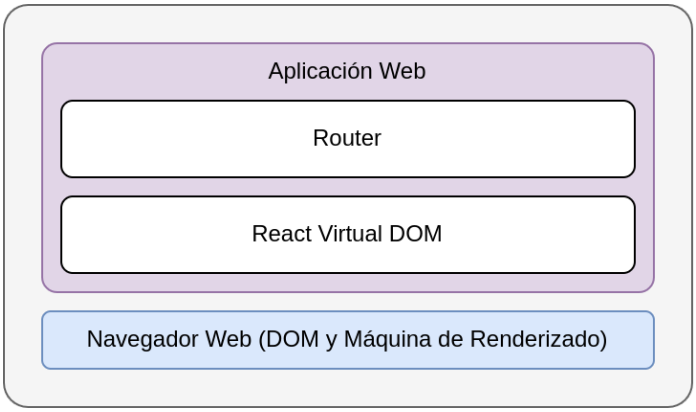


Figura 4.2: Arquitectura en el cliente

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.
- 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.

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. Es la 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:

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.

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.

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.

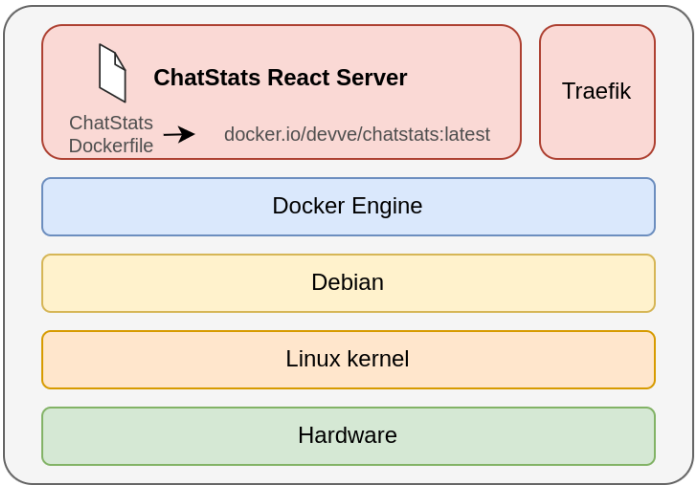


Figura 4.3: Arquitectura en el servidor

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 la versión de producción con el 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:

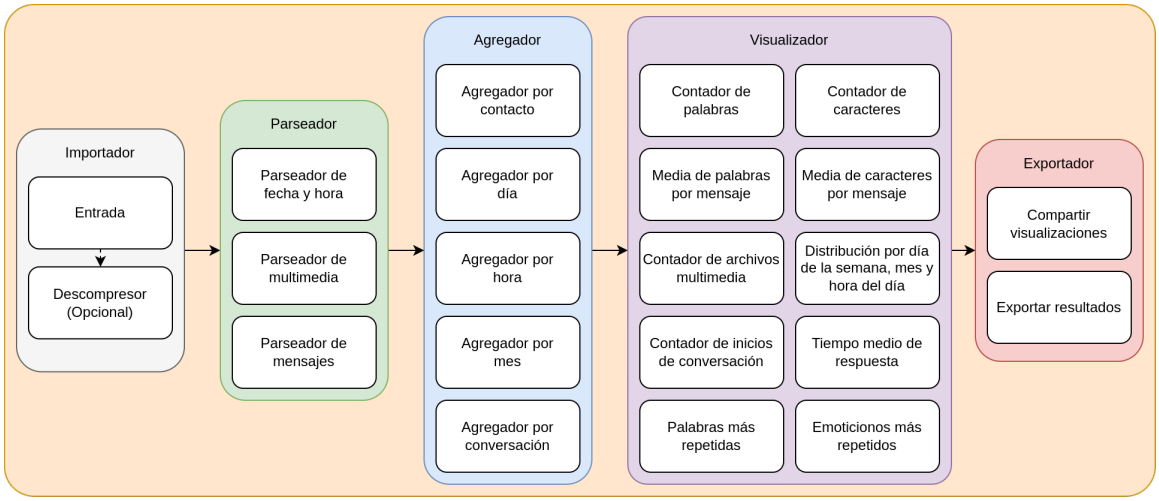


Figura 4.4: Arquitectura de la aplicación

4.4.1. Importador

El módulo importador se encarga de leer el fichero de entrada. 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*, ChatStats no soporta este formato, puesto que está diseñado para la visualización y no para el tratamiento y procesado del mismo.

En caso de exportar el chat con contenido multimedia, ChatStats únicamente espera el fichero de texto plano (ya sea directamente o dentro de un archivo comprimido), puesto que este se encuentran todos los datos necesarios para el análisis del contenido multimedia. Es por ello que todo el contenido multimedia recibido se descarta.

4.4.1.1. Entrada

Este submódulo se encarga de cargar el fichero seleccionado por el usuario, que el navegador ofrece desde una ruta falsa 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.

4.4.1.2. 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 "Entrada".

4.4.2. Parseador

El objetivo de este módulo es estandarizar los datos de entrada con la finalidad de poder utilizar los mismos módulos para cualquier archivo de entrada.

Mientras que Telegram ofrece los datos 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.

Se expone a continuación un breve ejemplo del formato utilizado por Telegram:

```
{
  "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": []
    }, ...
    ]
  }
```

También se muestra un mensaje en el formato de exportación de WhatsApp en los dispositivos 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. Dicha información está disponible también en los mensajes de Telegram.

4.4.2.1. Parseador de mensajes

El objetivo del submódulo es convertir la cadena de caracteres de entrada en mensajes con el siguiente formato JSON:

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

Hemos decidido esta estructura ya que se trata de toda la información que podemos obtener de mensajes de WhatsApp. Además, es compatible con la estructura ya propuesta por Telegram. Podemos decir entonces que es un máximo común denominador de la información disponible en ambos casos. Serán estos los objetos que se utilizarán más adelante para calcular las estadísticas y las estructuras de datos de visualización.

Es por ello que analizamos ambos casos por separado:

Telegram En caso de tratar un fichero en formato JSON, utiliza la librería *JSON* nativa para *JavaScript*, parseando todo el contenido del módulo "Importador" y eliminando los datos del objeto original que no se van a utilizar. Posteriormente, llama al módulo "Parseador de fecha y hora" que se describe más adelante.

WhatsApp En caso de tratar con un fichero de WhatsApp, se llega a la siguiente expresión regular que nos permite separar las partes del mensaje, tanto para Android como para iOS.

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

Los grupos de captura que lo componen se describen en detalle en el Apéndice D.

4.4.2.2. Parseador de fecha y hora

Este submódulo tiene como entrada la fecha y la hora. 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*.

La salida de este submódulo se utiliza en la clave *"date"* del objeto mensaje del submódulo anterior.

4.4.2.3. Parser de archivos multimedia

Este submódulo se ejecuta únicamente para chats exportados por WhatsApp. El objetivo de este submódulo es escribir el tipo de contenido en la clave *"media_type"* del objeto mensaje expuesto anteriormente.

Existen dos opciones para exportar un chat: con contenido multimedia o sin él.

Para contenido multimedia

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.

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)
```

Sin contenido multimedia

Para el segundo caso, cada vez que un mensaje sea contenido multimedia, 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.

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

4.4.3. Agregador

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:

4.4.3.1. 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 definidos en el *Parser de mensajes*.

4.4.3.2. 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.

4.4.3.3. 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.

4.4.3.4. 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.

4.4.3.5. 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.

4.4.4. Visualizador

Este módulo prepara los datos para ser representados por la librería de visualización elegida: *ChartJS*. Esta librería ha sido elegida por su alta actividad de contribuciones al proyecto, que es libre y cuenta con 12 mil estrellas en GitHub. Además, permite alta exten-

sibilidad mediante plugins, de los que hacemos uso, por ejemplo, para mostrar las etiquetas de los datos.

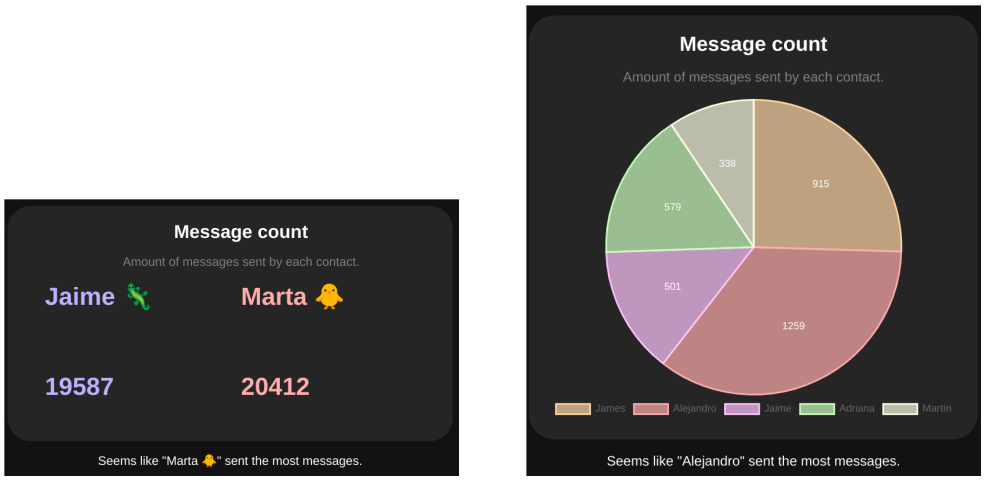
Cada módulo aquí desarrollado realiza el cálculo de una métrica, cuyo resultado pasa por una función que adapta el contenido a la estructura solicitada por *ChartJS*. *ChartJS* nos permite un único formato de entrada para los datos, permitiéndonos elegir el tipo de gráfico de forma independiente a dichos datos. Esta arquitectura nos permite añadir más módulos posteriormente; realizando el cálculo de la métrica a mostrar y pasándolo por la función que estandariza los datos para *ChartJS*.

Todos los submódulos que utilizan *ChartJS* 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.

Además, también se procesan datos para otras librerías de visualización, como *react-wordcloud*, para las nubes de palabras o *word clouds* y nubes de emoticonos.

4.4.4.1. Contador de mensajes

Este submódulo cuenta el número de mensajes enviado por cada contacto. Para chats grupales, usa la librería *ChartJS*, mientras que para chat individuales, únicamente se exponen los números de ambas partes directamente.



(a) Chat individual (b) Chat grupal

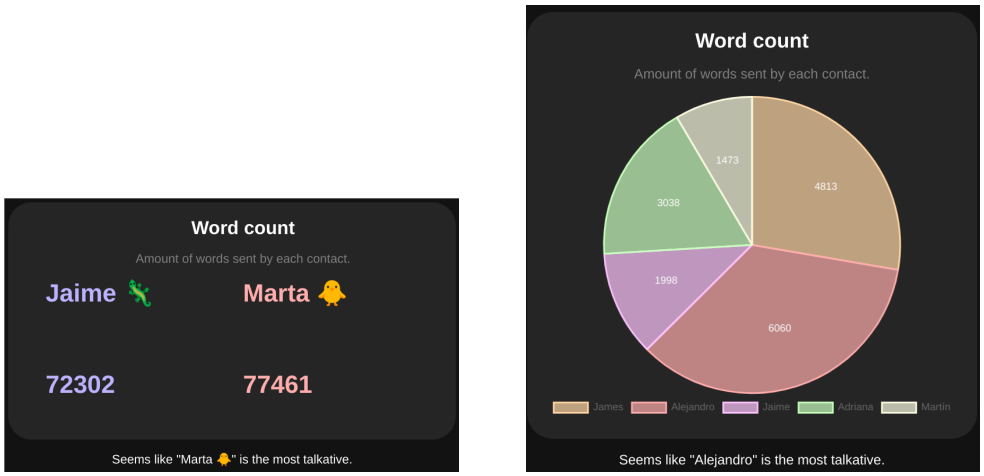
Figura 4.5: Contador de mensajes

Se ha optado por calcular esta métrica puesto que puede ayudar a observar diferencias

drásticas en la cantidad de texto que aporta cada contacto.

4.4.4.2. 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. Para chats grupales, usa la librería *ChartJS*, mientras que para chat individuales, únicamente se exponen los números de ambas partes directamente.

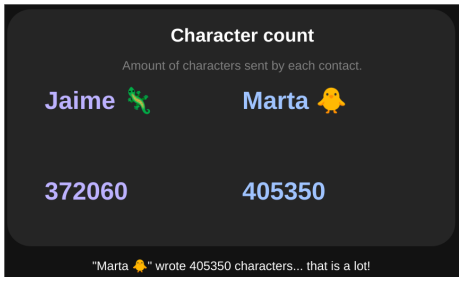


(a) Chat individual (b) Chat grupal

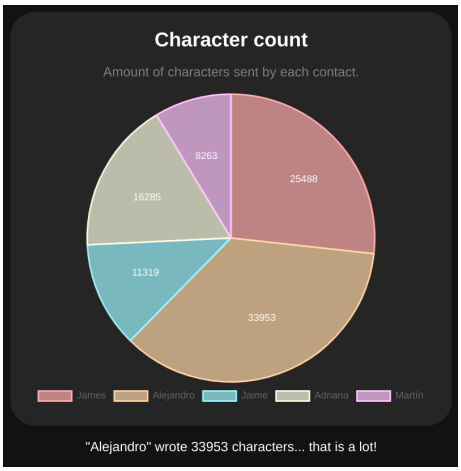
Figura 4.6: Contador de palabras

4.4.4.3. 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.



(a) Chat individual



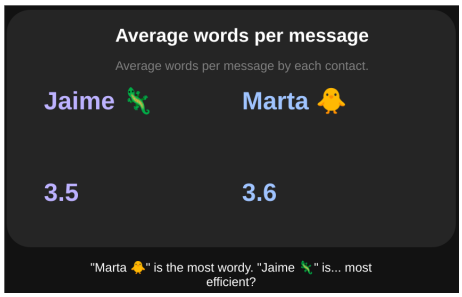
(b) Chat grupal

Figura 4.7: Contador de caracteres

De nuevo, para chats grupales, usa la librería *ChartJS*, mientras que para chat individuales, únicamente se exponen los números de ambas partes directamente.

4.4.4.4. 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.



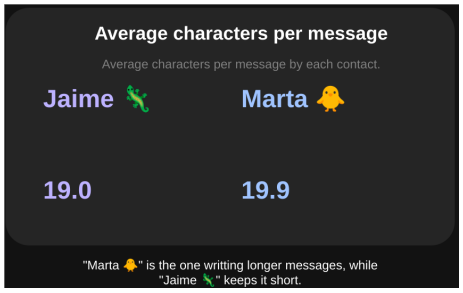
(a) Chat individual

(b) Chat grupal

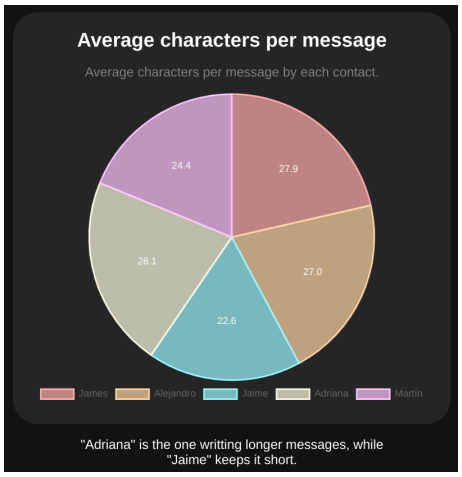
Figura 4.8: Media de palabras por mensaje

4.4.4.5. 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.



(a) Chat individual

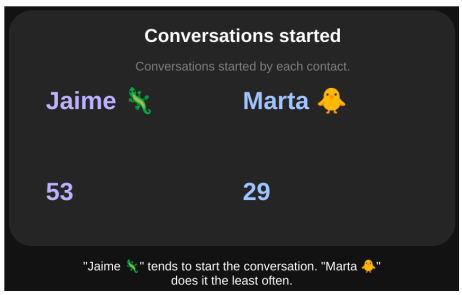


(b) Chat grupal

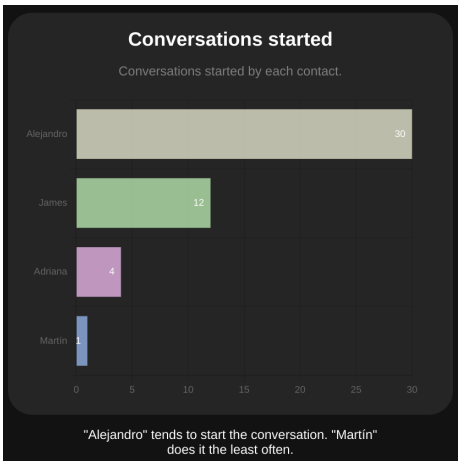
Figura 4.9: Media de caracteres por mensaje

4.4.4.6. 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.



(a) Chat individual



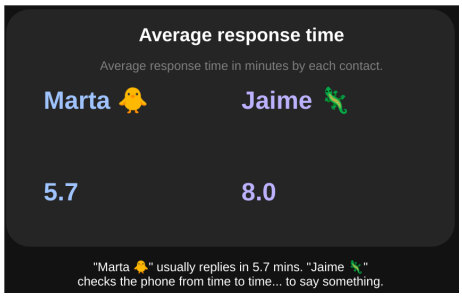
(b) Chat grupal

Figura 4.10: Conversaciones iniciadas por cada contacto

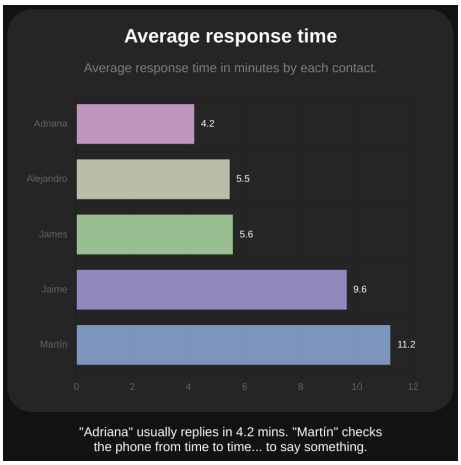
En este ejemplo, podemos ver cómo Alejandro, el presidente de la asociación, suele tomar la iniciativa y comenzar las conversaciones.

4.4.4.7. Velocidad media de respuesta

Este submódulo calcula la velocidad media de respuesta de cada contacto, 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 atención a la aplicación de mensajería instantánea, así como la importancia que le da al grupo.



(a) Chat individual



(b) Chat grupal

Figura 4.11: Tiempo medio de respuesta

Añadiendo esta figura a la anterior, podemos ver cómo Martín inicia pocas conversaciones y, además, suele tardar bastante en responder. Esto puede sugerir que está menos involucrado en el grupo.

4.4.4.8. 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.

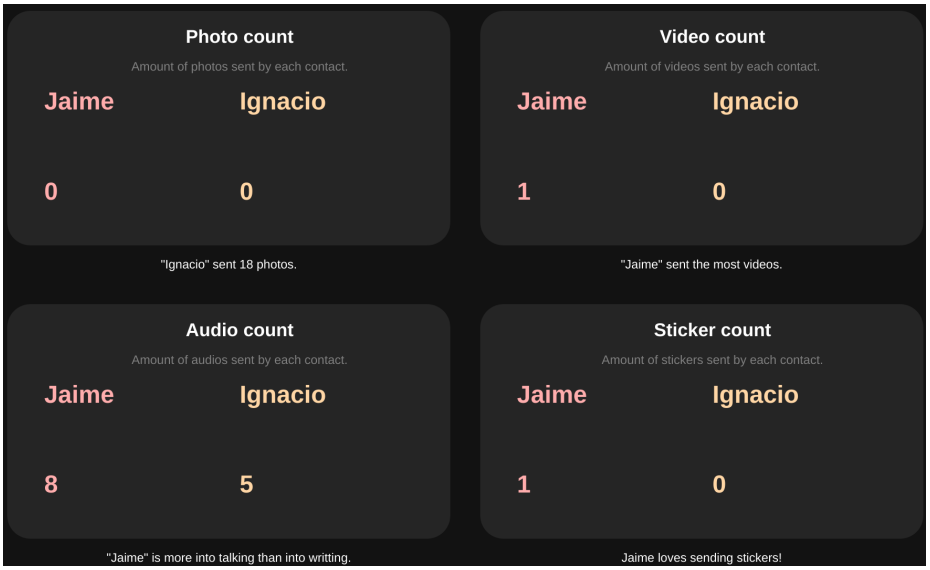


Figura 4.12: Contador de multimedia para chats individuales

4.4.4.9. Generador de estructuras de datos por día, hora y mes

Para el gráfico de barras con el número de mensajes en el tiempo, *ChartJS* necesita una estructura de datos para mensajes por día, siendo los días la variable independiente y el número de mensajes la variable dependiente.

Figura 4.13: Contador de multimedia para chats grupales

Figura 4.14: Distribuciones en el tiempo

Se elige calcular y mostrar estas distribuciones puesto que ayudan a observar la evolución en el tiempo de la cantidad de mensajes intercambiados, así como los días de la semana más activos. Por ejemplo una pareja que pasa los fines de semana juntos tendrá menos mensajes los fines de semana. Por último, la distribución en las 24 horas del día ayuda a analizar las horas pico de conversación, así como las horas de final e inicio del día para cada contacto.

Para la distribución por horas en el día, se ha planteado también usar un gráfico de araña o radar, pero dicha opción se descartó al observar el solapamiento que tenía lugar con varios contactos.

En la figura podemos observar cómo el contacto Jaime comienza a mandar mensajes a las 9 de la mañana, mientras que James suele comenzar 2 horas antes: a las 7 de la mañana. Esto sugiere que James comienza antes el día, o Jaime no utiliza el móvil hasta las 9 de la mañana.

4.4.4.10. 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*.

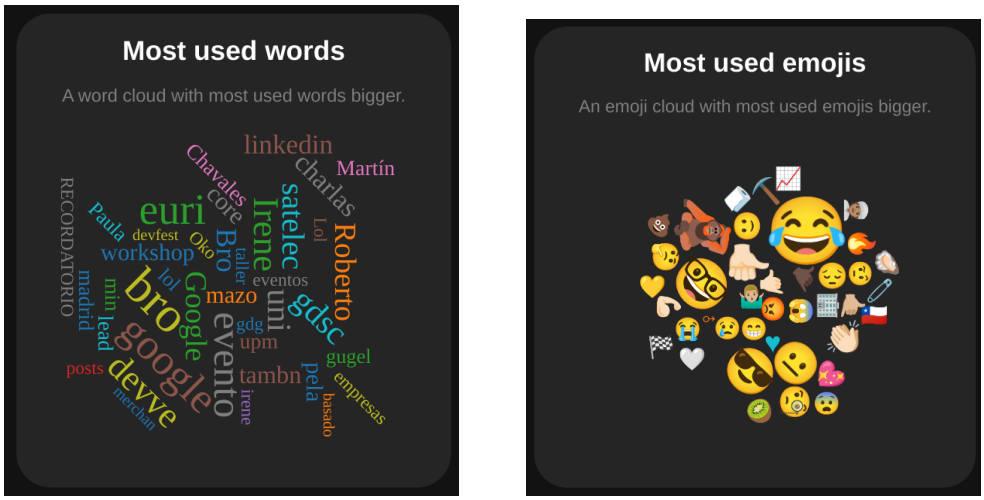
Las listas de palabras más comunes del español e inglés se han recopilado de distintas fuentes, combinado y eliminado repeticiones.

A la salida de este módulo, se muestra un diccionario con las palabras más repetidas y el número de veces que aparece cada una; información de la que hará uso la librería *react-wordcloud*. Esta información resulta útil para resaltar los temas principales tratados en el chat.

Puede observarse en la Figura 4.15

4.4.4.11. 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. Posteriormente otra nube de palabras hace uso de esta información por medio de la librería *react-wordcloud*.



(a) Nube de palabras

(b) Nube de emoticonos

Figura 4.15: Nube de palabras y emoticonos

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

4.5. Conclusión

Con todo, si en un futuro se quiere extender la arquitectura propuesta a un modelo de cliente-servidor, siempre se puede añadir un tercer contenedor en la última capa de la Sección 4.3 que implemente lógica adicional. Sería también necesaria cierta modificación en el código del cliente para implementar peticiones a este nuevo servicio.

CAPÍTULO 5

Caso de estudio

5.1. Introducción

Entiendo que aquí debería centrarme únicamente en un caso de uso, o varios si caben

En este capítulo vamos a describir un caso de uso seleccionado. Esta descripción cubrirá todas las funciones principales del caso de uso, así como una descripción detallada de los pasos a seguir y cómo usar la aplicación.

5.2. Consultar estadísticas y visualizaciones de chat grupal con contenido multimedia

Meter fotos de chat grupal con última versión

Actualizar capturas a última versión

Meter descripciones

Actualizar nombres de figuras

5.2.1. Paso 1. Acceso a la aplicación

El usuario accede a la aplicación web y encuentra la siguiente interfaz de usuario.

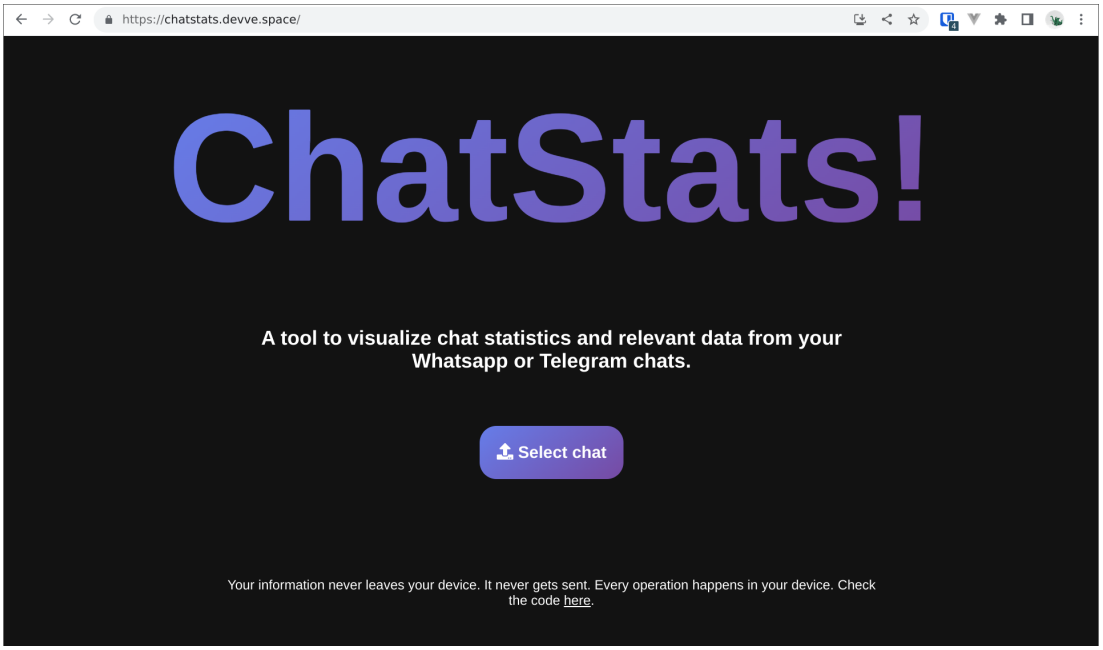


Figura 5.1: Página principal de la aplicación

En esta pantalla se describe la aplicación, así como una pequeña nota sobre la privacidad del servicio y un enlace al código fuente del mismo.

5.2.2. Paso 2. Selección del archivo de datos conversacionales

El usuario pulsa el botón para seleccionar un chat, lo que le abre el explorador de archivos de su sistema para seleccionar el fichero a utilizar.

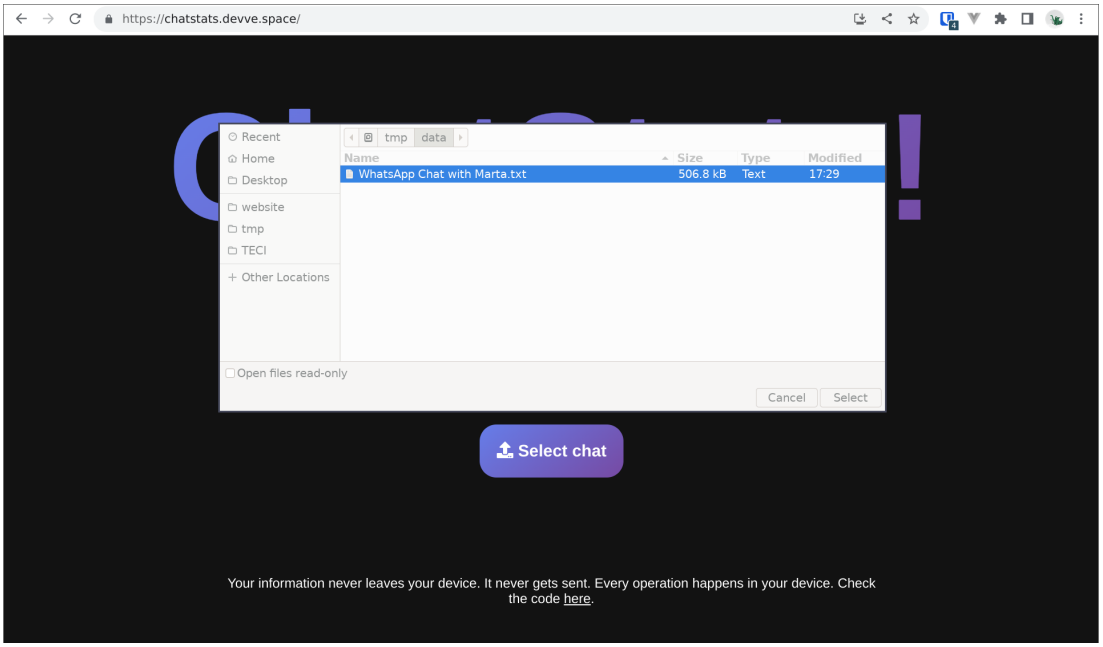


Figura 5.2: Selección del fichero de datos conversacionales

5.2.3. Paso 3. Confirmación del archivo seleccionado

Se muestra al usuario el nombre del archivo que ha seleccionado y se añade un botón para comenzar el análisis de los datos.

Asimismo, el botón para seleccionar un fichero sigue habilitado, permitiendo al usuario cambiar el archivo que ha seleccionado.

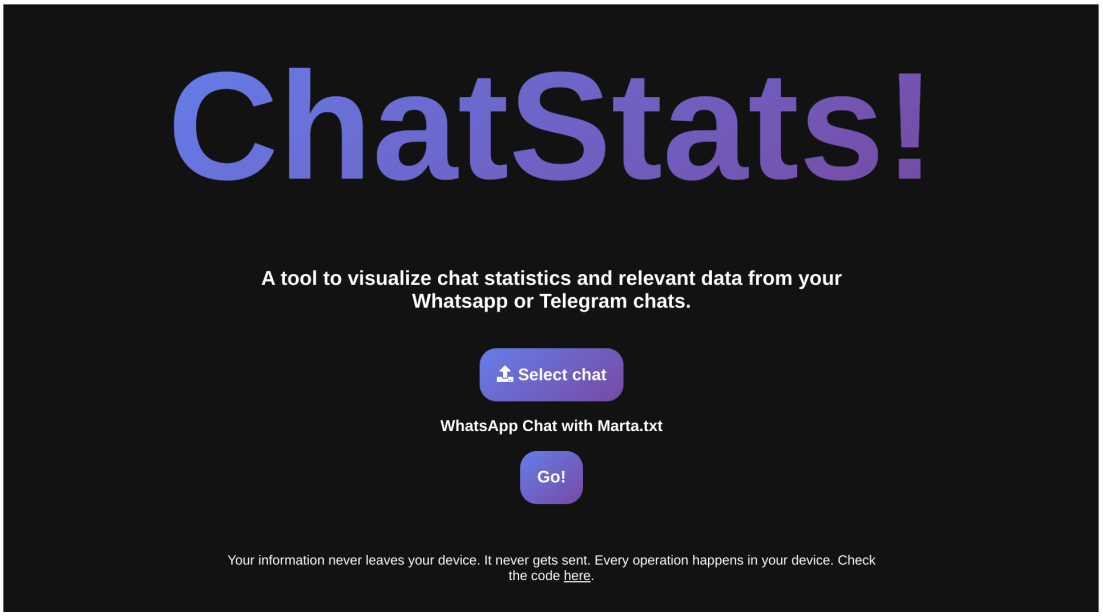


Figura 5.3: Confirmación de la selección

5.2.4. Paso 4. 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.

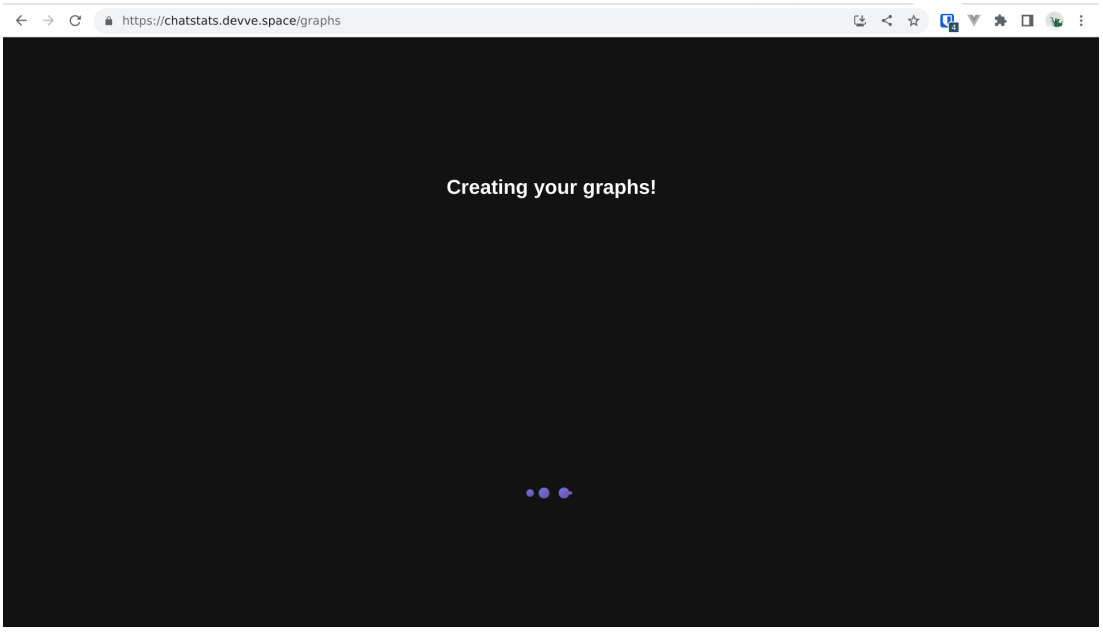


Figura 5.4: Cálculo de las estadísticas

5.2.5. Paso 5. Visualización de estadísticas y gráficos

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

Los primeros gráficos tratan del recuento de mensajes y caracteres totales, así como la media de los mismos por cada mensaje.

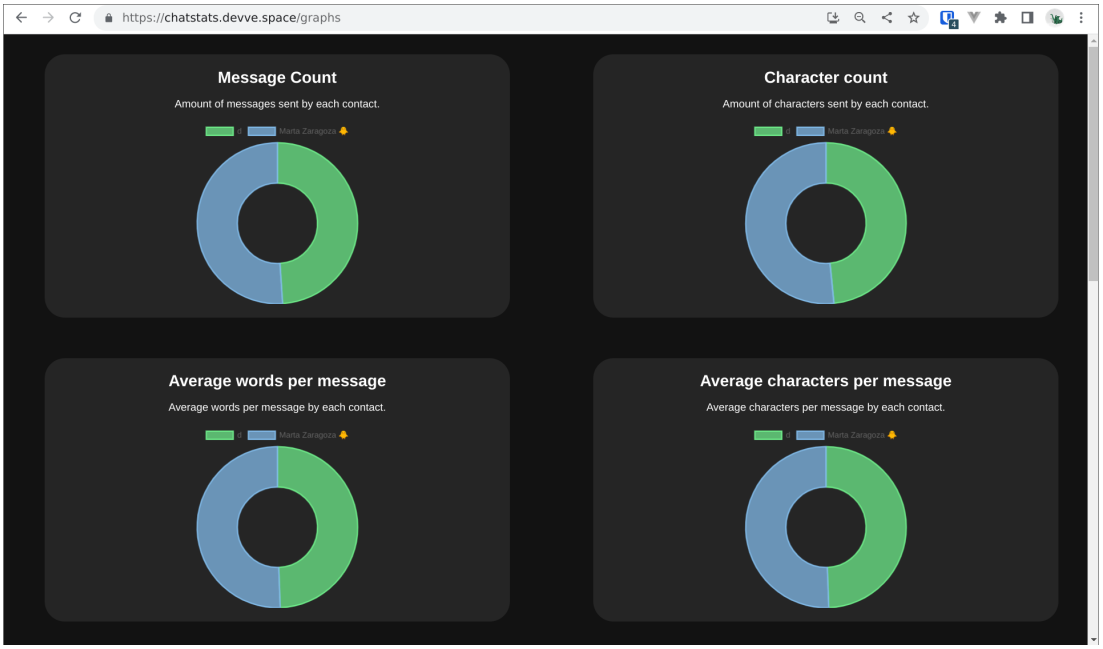


Figura 5.5: Cálculo de las estadísticas

Continuando más abajo, se muestra la distribución de los mensajes por contacto para cada mes, así como la distribución de los mensajes en media a lo largo de la semana. Finalmente, también se muestra la distribución de los mensajes en las horas en las que se envían.

Figura 5.6: Cálculo de las estadísticas

Al final de la página se introducen dos gráficos más.

La nube de palabras muestra las palabras más utilizadas en el chat, teniendo mayor tamaño las palabras más utilizadas.

La nube de emoticonos o, *emojis*, realiza la misma acción con los emoticonos.

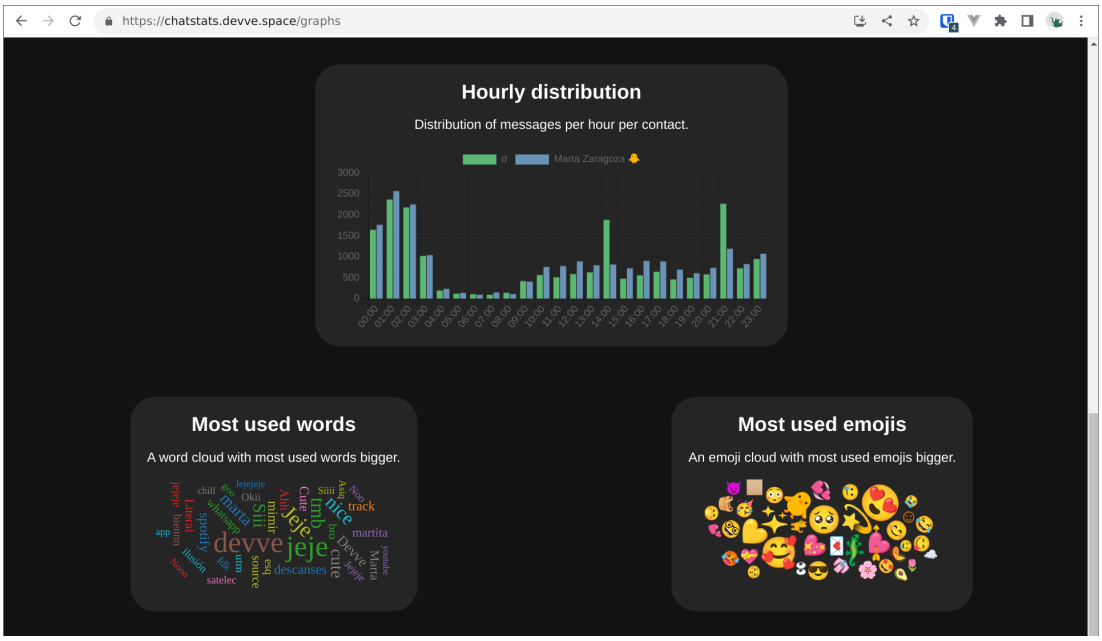


Figura 5.7: Cálculo de las estadísticas

CAPÍTULO 6

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

Académicamente, este proyecto ha desarrollado una doble función: la de proyecto personal y, la de 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. Personalmente, 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 asociaciones de estudiantes.

6.2. Objetivos conseguidos

Finalmente, hemos logrado alcanzar todos los objetivos propuestos en el Capítulo 1:

- Importar datos conversacionales de aplicaciónes 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.

6.3. Futuras líneas de trabajo

- Migración de los módulos con mayor carga de trabajo a Web Assembly (WASM), permitiendo obtener un rendimiento mayor y 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.
- Inclusión de mensajes eliminados por cada contacto.
- Detección de grupos de interacción: qué contactos suelen responder entre ellos.
- Implementación de una *cache* en la Progressive Web App (PWA) para poder ejecutar la aplicación sin necesidad de acceso a Internet (una vez instalada). Actualmente, la aplicación puede instalarse, pero cada vez que se abre solicita el código al servidor.
- Mejora de la documentación para futuros contribuyentes al proyecto.

APÉNDICE A

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 WhatsAp y Telegramp, 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.

APÉNDICE B

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 €

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 €

APÉNDICE C

Código

C.1. Acceso al código fuente

Acceso principal: `https://codeberg.org/devve/chatstats`

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

APÉNDICE D

Expresiones regulares

D.1. Grupos de captura

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

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

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

Grupo de captura 1: fecha

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

Se encarga de la fecha en formato *dd/mm/YYYY*, denotado con "$\backslash d\{X\}$" que indica que se buscan X 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 la fecha siempre sigue este formato, independientemente del sistema operativo. No hay consistencia entre los ajustes del parámetro *locale* de *en_US* y *es_ES*, siendo *mm/dd/YYY* y *dd/mm/YYYY* respectivamente. Para ello, el cliente accederá al *locale* para actuar en consecuencia más adelante. No se ha probado para otras configuraciones.

Grupo de captura 2: hora

```
(\d(?:\d)?:\d{2})
```

Se encarga de la la hora en formato *hh:MM*, aunque en alguna ocasión se ha observado que no hay consistencia si la hora es de un solo dígito, pudiendo aparecer 01:00 o 1:00 en función del dispositivo y la versión de WhatsApp que ejecuta. Es por ello que la expresión regular es algo más compleja y se buscan dígitos a la izquierda del carácter "*:*" independientemente del número de repeticiones del mismo. Los minutos si han mostrado consistencia, por lo que se utiliza "$\backslash d\{2\}$".

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.

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*).

Look ahead o mirada hacia delante

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

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, acompañada seguida de una coma “,” y un espacio.

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.