universidad de león

Escuela de Ingenierías I.I.

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo de Fin de Grado

Sync WhatsApp: Creación y utilización de APIs para la integración de WhatsApp en una plataforma web.

Autor: César Gutiérrez Pérez

Tutor: Héctor Alaiz Moretón

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías I.I.
GRADO EN INGENIERÍA INFORMÁTICA
Trabajo de Fin de Grado

ALUMNO: César Gutiérrez Pérez

TUTOR: Héctor Alaiz Moretón

TÍTULO: Sync WhatsApp: Creación y utilización de APIs para la integración de WhatsApp en una plataforma web.

CONVOCATORIA: Diciembre, 2017

RESUMEN:

El proyecto consiste en la creación y posterior utilización de APIs para WhatsApp, creando una herramienta de sincronización que se integrará con una plataforma creada anteriormente a nivel de SQL.

Este trabajo da solución a la inexistencia de APIs por parte de WhatsApp, ofreciendo un servicio que permite la sincronización total con cuentas de WhatsApp.

Los principales problemas a los que ofrece solución son:

- Autorización de usuarios desde que un usuario permite a la aplicación hacer uso de su cuenta, hasta que decida revocar el permiso o un administrador lo desautorice en el sistema.
- Automatización del envío de mensajes desde una cola de SQL integrada en una plataforma.
- Monitorización de toda la actividad de las cuentas de WhatsApp relacionadas con la plataforma, para poder ser analizada y poder trabajar con ella en el futuro.

Palabras clave: WhatsApp, APIs, asíncrono, concurrente, automatizar, tiempo real.

Firma del alumno:	**VºBº Tutor:**

Índice de contenidos

0. Introducción 10
0.1 Contexto y antecedentes del proyecto 10
0.2 Justificación del proyecto 11
0.3 Motivación personal 11
0.4 Descripción del problema 12
0.5 Que es la concurrencia y la programación asíncrona 13
0.6 Tecnologías utilizadas 14
0.7 Alcance del proyecto 17

1. Planificación 18
1.1 Planificación inicial 18
1.1.1 Planificación de tareas para familiarizarse con el proyecto. 18
1.1.2 Planificación de un sistema de autorización para cuentas de WhatsApp. 21
1.1.3 Planificación de un sistema automático de envío de iteraciones para cuentas de WhatsApp. 22
1.1.4 Planificación de un sistema de monitorización para cuentas de WhatsApp. 24
1.1.5 Planificación de una interfaz Web donde se muestren los resultados obtenidos de la monitorización 25
1.1.6 Evaluación, refactorización y corrección de errores 25
1.1.7 Métricas y sistemas de evaluación futuras 26
1.1.8 Documentación del proyecto 26
1.2 Planificación final 27
1.3 Diagramas Gantt de la planificación 29

2. Objetivos 30
2.1 Objetivos del dueño del producto 30
2.2 Requisitos funcionales 31
2.3 Requisitos no funcionales 32
2.4 Objetivos técnicos 33
2.4.1 Autorización de cuentas de WhatsApp 33
2.4.2 Envío automático de iteraciones sobre WhatsApp 35
2.4.3 Captura y monitorización de los datos 37
2.4.4 Interfaz Web para mostrar los datos capturados 38
2.4.5 Especificaciones comunes para todas las interfaces de tipo Web 39

3. Resultados 40
3.1 Autorización 40
3.1.1 En el lado del cliente 40
3.1.2 En el lado del servidor 42
3.2 Ejecución automática de iteraciones sobre WhatsApp 44
3.2.1 En el lado del cliente 44

3.2.2 En el lado del servidor....46
3.3 Captura y monitorización de datos....48
3.4 Interfaz web para mostrar datos de cuentas monitorizadas....52
3.5 *Logs* de todos los procesos....56
4. Descripción técnica....59
4.1 Descripción detallada de cada uno de los subproyectos....59
4.1.1 Autorización de cuentas....59
4.1.2 Envío automático de iteraciones....61
4.1.3 Monitorización de datos....64
4.1.4 Interfaz web para mostrar los resultados....68
4.1.5 *Logs* del sistema....70
4.2 Web scraping....70
4.3 Comparativa entre Selenium y NW.js....71
4.4 Socket.IO....73
4.5 API REST....74
4.6 Modelo de datos....77
4.6.1 Autorización de cuentas....77
4.6.2 Envío automático de iteraciones....79
4.6.3 Monitorización de datos....81
4.7 Librerías de terceros utilizadas....84
4.7.1 Librerías de Node.js....84
4.7.2 Librerías para las interfaces web....85
5. Manual de usuario....86
5.1 Autorización....86
5.2 Envío automático de iteraciones....87
5.3 Monitorización de datos....89
5.3.1 Interfaz web para mostrar los resultados....90
6. Conclusiones....91
7. Líneas futuras....92
8. Agradecimientos....94
9. Lista de referencias....95

Índice de figuras

Figura 0-1 - Logo de HumedoVALLEY 10
Figura 0-2 - Logo León Literario 10
Figura 0-3 - Arquitectura actual de León Literario 12
Figura 0-4 - Diagrama de ejecución de tareas asíncronas 14
Figura 0-5 - Diagrama de ejecución de tareas en paralelo 14
Figura 0-6 - Logo de Materialize 15
Figura 0-7 - Logo de SQL Server 15
Figura 0-8 - Logo de Selenium 15
Figura 0-9 - Logo de NW.js 16
Figura 0-10 - Funcionamiento de Node.js. (11) 16
Figura 1-1 - Diagrama de Gantt Inicial 29
Figura 1-2 - Diagrama de Gantt Final 29
Figura 2-1 - Diagrama del sistema de autorización 34
Figura 2-2 - Diagrama de desautorización 34
Figura 2-3 - Diagrama de comprobación de autorización 35
Figura 2-4 - Diagrama del envío de iteraciones 36
Figura 2-5 - Diagrama de herramienta para ejecutar consultas 36
Figura 2-6 - Diagrama del sistema de monitorización 37
Figura 2-7 - Diagrama de la interfaz web 38
Figura 3-1 - Interfaz de autorización cargando 40
Figura 3-2 - Interfaz de autorización cargada 41
Figura 3-3 - Autorización completada 41
Figura 3-4 - Interfaz de autorización en dispositivo móvil 42
Figura 3-5 - Terminal Node para la autorización 42
Figura 3-6 - Terminal Node -> Enviando código QR 43
Figura 3-7 - Terminal Node -> Autorización completa 43
Figura 3-8 - Terminal Node -> Desconexión del cliente 43
Figura 3-9 - Herramienta para ejecutar consultas 44
Figura 3-10 - Herramienta para ejecutar consultas rellena 45
Figura 3-11 - Herramienta para generar consultas -> Consulta ejecutada 46
Figura 3-12 - Terminal Node -> Buscando iteraciones 46
Figura 3-13 - Terminal Node -> Ejecutando batch 47
Figura 3-14 - Terminal Node -> Batch finalizado 47

Figura 3-15 - Interfaz con lista de usuarios 48
Figura 3-16 - Lista de usuarios visible 49
Figura 3-17 - Monitorización de cuentas 50
Figura 3-18 - Monitorización de un único usuario 50
Figura 3-19 - Chat auxiliar 51
Figura 3-20 - Mensaje nuevo 51
Figura 3-21 - Monitorización de nuevos mensajes 52
Figura 3-22 - Interfaz con los usuarios monitorizados 52
Figura 3-23 - Resultado de la busqueda por usuario 53
Figura 3-24 - Buscador de usuarios 53
Figura 3-25 - Botón para desautorizar 54
Figura 3-26 - Interfaz con lista de chats 54
Figura 3-27 - Resultado de búsqueda en los chats 55
Figura 3-28 - Interfaz con los mensajes de un chat 55
Figura 3-29 - Interfaz con mensajes de un chat en dispositivo móvil 56
Figura 3-30 - Log de autorización 57
Figura 3-31 - Log de envío automático de iteraciones 57
Figura 3-32 - Log de monitorización de cuentas 58
Figura 4-1 - Cookies de WhatsApp Web 60
Figura 4-2 - Fragmento del script para el envío de iteraciones 63
Figura 4-3 - Botones de WhatsApp Web 65
Figura 4-4 - Fragmento de la función para iniciar la monitorización 65
Figura 4-5 - Header de usuario en WhatsApp Web 66
Figura 4-6 - Información de usuario en WhatsApp Web 66
Figura 4-7 - Mensajes sin leer 67
Figura 4-8 - Mensajes de chat monitorizado 69
Figura 4-9 - Explicación de Web Scraping 70
Figura 4-10 - Uso de Web Scraping 71
Figura 4-11 - Señal de envío correcto de un mensaje 72
Figura 4-12 - Implementación de Socket.IO en Node 73
Figura 4-13 - Implementación de Socket.IO en el cliente 73
Figura 4-14 - Envío y recepción de datos con Socket.IO 73
Figura 4-15 - Tabla autorización 77
Figura 4-16 - Tabla envío automático de iteraciones 79
Figura 4-17 - Tabla de monitorización de datos 82

Figura 5-1 - Terminal para iniciar Selenium 86
Figura 5-2 - Terminal para iniciar Node.js 86
Figura 5-3 - Envío de una petición POST con Postman 87
Figura 5-4 – Programa Java sin datos 87
Figura 5-5 - Programa Java con datos 88
Figura 5-6 - Herramienta Java ejecutando consultas 88
Figura 5-7 - Disposición de directorios 89
Figura 5-8 - Botón salir 89
Figura 5-9 - Botón desautorización..... 90
Figura 7-1 - Iniciar sesión con WhatsApp en León Literario 92

Índice de cuadros y tablas

Tabla 1-1 - Tarea 1: Explicación del proyeto 18
Tabla 1-2 - Tarea 2: Aprendizaje de GIT 19
Tabla 1-3 - Tarea 3: Familiarización con el uso de APIs 19
Tabla 1-4 - Tarea 4: Actualización de Sync Twitter 20
Tabla 1-5 - Tarea 5 - Desarrollo de Sync Facebook 20
Tabla 1-6 Tarea 6: Análisis del problema y búsqueda de soluciones 21
Tabla 1-7 - Tarea 7: Análisis y eleccion de tecnologias para Sync_Whatsapp_Auth 21
Tabla 1-8 - Tarea 8: Implementación del Software de Sync_WhatsApp_Auth 22
Tabla 1-9 - Tarea 9: Análisis y elección de tecnologías para Sync_WhatsApp_Queue 22
Tabla 1-10 - Tarea 10: Implementación del Software de Sync_WhatsApp_Queue23
Tabla 1-11 - Tarea 11: Análisis y elección de tecnologías para Sync_WhatsApp_Anatytics 24
Tabla 1-12 - Tarea 12: Implementación del Software de Sync_WhatsApp_Analytics 24
Tabla 1-13 - Tarea 13: Creación de la interfaz Web 25
Tabla 1-14 - Tarea 14: Evaluación, refactorización y corrección de errores 25
Tabla 1-15 - Tarea 15: Métricas y sistemas de evaluación futura 26
Tabla 1-16 - Tarea 16: Documentación del proyecto 26
Tabla 1-17 - Desfase en la planificación 27
Tabla 2-1 - Requisitos funcionales 31
Tabla 2-2 - Requisitos no funcionales 32
Tabla 4-1 – *Endpoint* Deauthorize 74
Tabla 4-2 – *Endpoint* List_elements 75
Tabla 4-3 – *Endpoint* Delete_element 75
Tabla 4-4 – *Endpoint* Get_chats 76
Tabla 4-5 – *Endpoint* Get_messages 76
Tabla 4-6 - Procedimiento almacenado Authorize 78
Tabla 4-7 - Procedimiento almacenado DeAuthorize 78
Tabla 4-8 - Procedimiento almacenado GetSesions 79
Tabla 4-9 - Procedimiento almacenado addACTION 80
Tabla 4-10 - Procedimiento almacenado updateACTION 80
Tabla 4-11 - Procedimiento almacenado getBATCH 81

Tabla 4-12 - Procedimiento almacenado add..... 83
Tabla 4-13 - Procedimiento almacenado GetChats 83
Tabla 4-14 - Procedimiento almacenado GetMessages..... 84

0. Introducción

0.1 Contexto y antecedentes del proyecto

Este proyecto ha sido realizado para la integración con una plataforma de la empresa *HumedoValley*.

Figura 0-1 - Logo de HumedoVALLEY

HumedoValley es una *startup* cuya actividad se centra en el desarrollo de soluciones de tipo tecnológicas, utilizando herramientas novedosas, para diversos ámbitos, aunque cabe destacar el ámbito cultural, donde su última solución es *León Literario*.

León Literario es una plataforma web orientada al ámbito literario que trata de fomentar la literatura mediante actividades de información y promoción de artes culturales relacionadas con la literatura, así como permitir a los usuarios interactuar de manera activa con los autores, figuras destacadas. El portal también cuenta con una sección de noticias relacionadas con el mundo de la literatura, concursos, ferias, entrevistas, etc.

Figura 0-2 - Logo León Literario

Actualmente, la empresa tiene buenas relaciones con librerías, escritores, editores, promotores de eventos y administraciones locales dentro de la provincia de León.

El objetivo de *HumedoValley* es extender *León Literario* no solo a nivel local, sino a nivel nacional y aquí es donde cobra sentido la necesidad de herramientas de sincronización con las redes sociales más utilizadas, como son: Twitter, Facebook y WhatsApp. El uso de estas herramientas permitiría expandir la actividad de *León Literario* en las redes sociales, así como también obtener el perfil de los usuarios que participan o visitan esta plataforma de una manera automatizada.

0.2 Justificación del proyecto

Actualmente, en *HumedoValley*, la gestión de redes sociales se realiza de forma manual utilizando herramientas conocidas. Ante los planes de expansión a nivel nacional y una previsión de crecimiento del número de usuarios se ha encontrado un gran problema.

Para resolver este problema, surge la idea de crear software independiente que se encargue de realizar de forma automática las interacciones de *León Literario*, y del mismo modo, las de los usuarios que utilicen esta plataforma y tengan la autorización necesaria para publicar su actividad dentro de *León Literario* en las diversas redes sociales de manera automática.

Para solucionar este problema, surge un proyecto para crear un software independiente encargado de automatizar el envío de interacciones propias, así como la actividad de los usuarios asociados con la plataforma *León Literario* y publicarla en las redes sociales de manera automática.

Otra razón para el desarrollo de este proyecto es la necesidad de conocer y aprender de los perfiles del cliente que utiliza esta plataforma. Con esto, me refiero a monitorizar la información que los usuarios de *León Literario* depositan en las redes sociales, para posteriormente analizarla consiguiendo así mejorar los servicios que ofrece la plataforma en función de los intereses de los mismos. Esto se traduce en ofrecer distintos servicios para cada perfil de usuario basándose en sus intereses.

0.3 Motivación personal

Desde un principio me llamó la atención este proyecto, ya que consideraba que era interesante y además presentaba un gran reto, donde seguro que iba a aprender nuevos conocimientos y a familiarizarme con el uso de APIs.

En este proyecto se utilizan muchos lenguajes de programación de distinta naturaleza y herramientas muy diversas. Esto me produjo interés ya que vi en ello una gran oportunidad para aprender nuevos lenguajes y técnicas de programación que no conocía.

Cuando me decanté por este proyecto y Pedro Pérez, director de *HumedoValley*, me explicó en detalle su contenido y lo que pretendía ofrecerme en él, vi una gran ventaja y una gran desventaja. La desventaja es que WhatsApp no ofrece APIs. La ventaja es que eso me llevo a hacer un estudio del funcionamiento interno de WhatsApp Web para crear APIs propias y, de esa forma, aprender muchas cosas que antes no conocía.

Otra de las grandes ventajas de este proyecto es que era un desarrollo desde cero por lo que tendría que seleccionar desde las tecnologías a utilizar hasta el diseño de la arquitectura.

Durante el desarrollo no todo fue tan fácil, hubo etapas muy difíciles. Encontré problemas tanto buscando información, ya que me atrevo a decir que no existe información relacionada con la creación de APIs de WhatsApp, como para realizar procesos de forma secuencial, ya que uno de los principales lenguajes que utilicé tiene naturaleza asíncrona. Aun así, conseguí superar estos retos, motivándome a seguir hasta el final.

0.4 Descripción del problema

Este proyecto, como mencioné anteriormente, resuelve el problema de poder realizar de forma automatizada, y desde una plataforma, diversas tareas de sincronización con distintas redes sociales. Ahora me centraré en explicarlo de manera más detallada.

Este problema surge con la necesidad de expansión de la plataforma *León Literario*. Esta necesidad está asociada también a la funcionalidad de que la actividad del usuario fuera publicada de manera automática en las principales redes sociales, siempre que el usuario autorizara a la plataforma, dio pie al problema de crear un software que se adaptara a la arquitectura existente.

Este software tendría que encargarse de gestionar las autorizaciones de los usuarios de las redes sociales, así como de los permisos de los mismos, de publicar de manera automática las interacciones y de monitorizar su actividad. (4)

Uno de los retos más importantes para poder crear el software era la adaptación

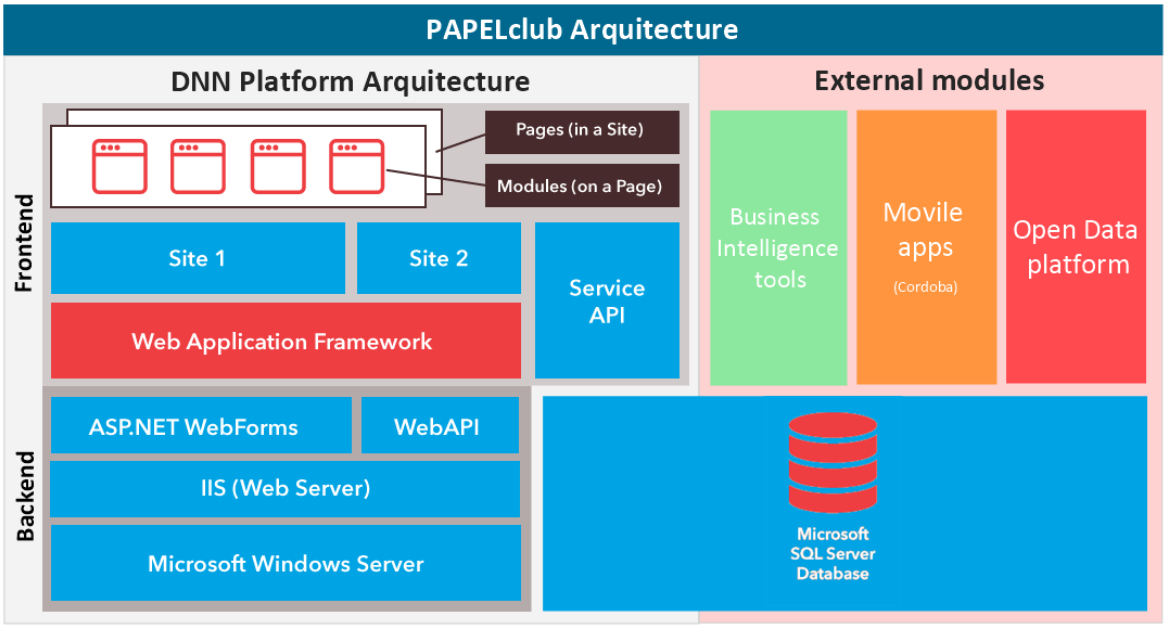

Figura 0-3 - Arquitectura actual de León Literario

con la arquitectura existente. Como se ve en la figura 0.3 la arquitectura se centra principalmente en los datos, teniendo un modelo estable y bien definido y construyendo a partir de él.

El desarrollo de esta arquitectura se basó en DNN (DotNetNuke), el cual, es un gestor de contenidos creado en ASP.NET. Utilizando DNN como núcleo, se

desarrollaron módulos que proveían de servicios web a los que llamaba una interfaz creada en HTML5 con CSS y Javascript.

Así mismo, otros componentes utilizaban los datos de la base de datos de manera independiente a DNN para los diversos usos descritos en la figura 0.3.

Otro problema que surgió fue la necesidad de conocer el perfil de los usuarios de la plataforma. Como el anterior problema, requería tener acceso a los datos del usuario para publicar de manera automática en su nombre, con ese mismo acceso podríamos obtener algún detalle más del usuario, así como monitorizar la actividad relacionada con la plataforma en las redes sociales.

Para intentar solucionar los problemas descritos anteriormente se buscaron alternativas existentes, pero ninguna se acercaba a los requisitos necesarios, o bien porque no podían ser adaptadas a la arquitectura existente, o porque el alto número de peticiones supondría, o bien un coste inasumible o una pérdida de estabilidad en el programa. Ante esto, se optó por un desarrollo propio.

En el caso de la creación de software para la sincronización de la plataforma con Twitter y Facebook, se optó por crear herramientas que utilizasen las APIs oficiales de Twitter y Facebook respectivamente. En cambio, en el caso de WhatsApp, al no existir una API oficial, se optó por la creación de una API propia para su posterior utilización.

0.5 Que es la concurrencia y la programación asíncrona

En el desarrollo de software surgió un nuevo problema técnico: es necesario realizar en un orden secuencial diversas funciones manteniendo ese orden en todo momento para que funcione de manera correcta el software. Por ejemplo, cuando se está monitorizando información de una cuenta de WhatsApp algunas tareas para llevar a cabo esta acción deben realizarse de manera secuencial, de forma que es preciso que finalice la tarea actual completamente para empezar la siguiente. De igual modo hay tareas que se pueden realizar de forma concurrente como, por ejemplo, una vez que tenemos cierta información y la queremos guardar en la base de datos, mientras esta se está guardando, podemos seguir recopilando nueva información. (12)

Para solucionar los problemas de concurrencia se pueden usar diferentes aproximaciones: la programación asíncrona o la programación paralela. (9)

- La programación asíncrona consiste en ejecutar las tareas concurrentes sin esperar a que unas terminen para que otras puedan comenzar. Se puede apreciar en el siguiente diagrama:

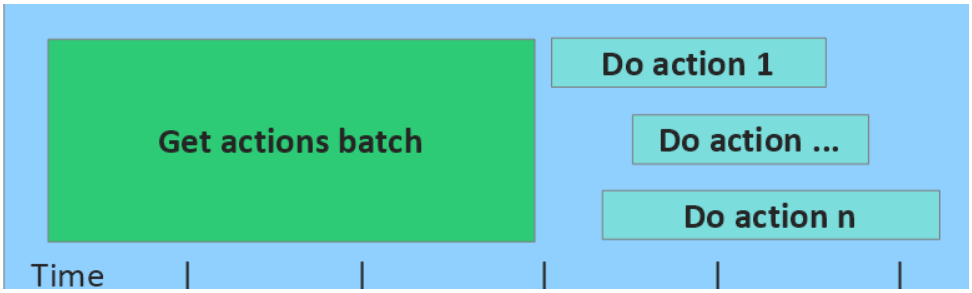

Figura 0-4 - Diagrama de ejecución de tareas asíncronas

- La programación paralela consiste en realizar de manera simultánea todas las acciones necesarias. En la figura siguiente se puede apreciar:

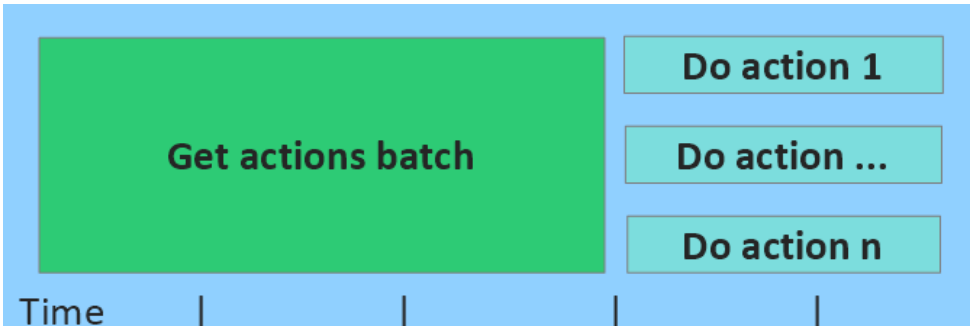

Figura 0-5 - Diagrama de ejecución de tareas en paralelo

Las ventajas de la programación paralela son las grandes mejoras en rendimiento, pero el inconveniente suele ser la gran dificultad de implementación. En la programación asíncrona suele ser muy sencillo de implementar debido a que no hace falta que el usuario cree hilos porque en esta programación se gestionan automáticamente, pero no es recomendable para realizar acciones muy complejas. (5)

0.6 Tecnologías utilizadas

Antes de empezar con este apartado cabe destacar que el proyecto Sync WhatsApp se divide en tres subproyectos que son independientes y que utilizan tecnologías muy diferentes.

- Sync_Whatsapp_Auth: El encargado de realizar la autorización de los usuarios. En este subproyecto se utiliza:

- Una interfaz web en HTML5, desarrollada con el Framework Materialize junto con CSS propio, Javascript y la implementación de Socket.IO.

Figura 0-6 - Logo de Materialize

- Una base de datos Microsoft SQL Server.

Figura 0-7 - Logo de SQL Server

- Un entorno de pruebas software para aplicaciones basadas en web llamado Selenium.
- Un servidor realizado en Node.js que sirve la interfaz web, realiza las consultas a la base de datos y controla Selenium.

- Sync_Whatsapp_Queue: Es utilizado para obtener, mediante una cola, iteraciones de usuarios almacenadas en la base de datos y publicarlas en WhatsApp. Este subproyecto consta de:
  - Una base de datos Microsoft SQL Server.
  - Un pequeño programa realizado en el lenguaje JAVA que se encarga de la creación y ejecución de scripts en la base de datos.
  - Un entorno de pruebas software para aplicaciones basadas en web llamado Selenium.
  - Un servidor realizado en Node.js que se encarga procesar una cola de iteraciones de usuario de la base de datos.

Figura 0-8 - Logo de Selenium

- Sync_Whatsapp_Analytics: Es utilizado para realizar la monitorización de cuentas de WhatsApp. Este subproyecto consta de:
    - Una base de datos Microsoft SQL Server.
    - NW.js: Es un creador de aplicaciones de escritorio con la peculiaridad de que dichas aplicaciones se ejecutan dentro de un servidor Node.js. Se utiliza para simular el uso de WhatsApp Web en un servidor Node.js.

Figura 0-9 - Logo de NW.js

Como se puede observar todos los subproyectos tienen algo en común y es que todos ellos utilizan un servidor Node.js, que es el núcleo donde se gestiona prácticamente la totalidad de los procesos software.

Me decante por el uso de Node.js porque utiliza un modelo asíncrono en el que un solo hilo. Es decir, este hilo (*event loop*) se encarga de registrar las operaciones y cuando estas son completas independientemente de cuando llegan las peticiones o cuando acaban de procesarse. De esta manera es un lenguaje cuya naturaleza es asíncrona y por esto no conlleva bloqueos de I/O, lo que se traduce en un mayor rendimiento y menor tiempo de ejecución de un conjunto de tareas. (15)

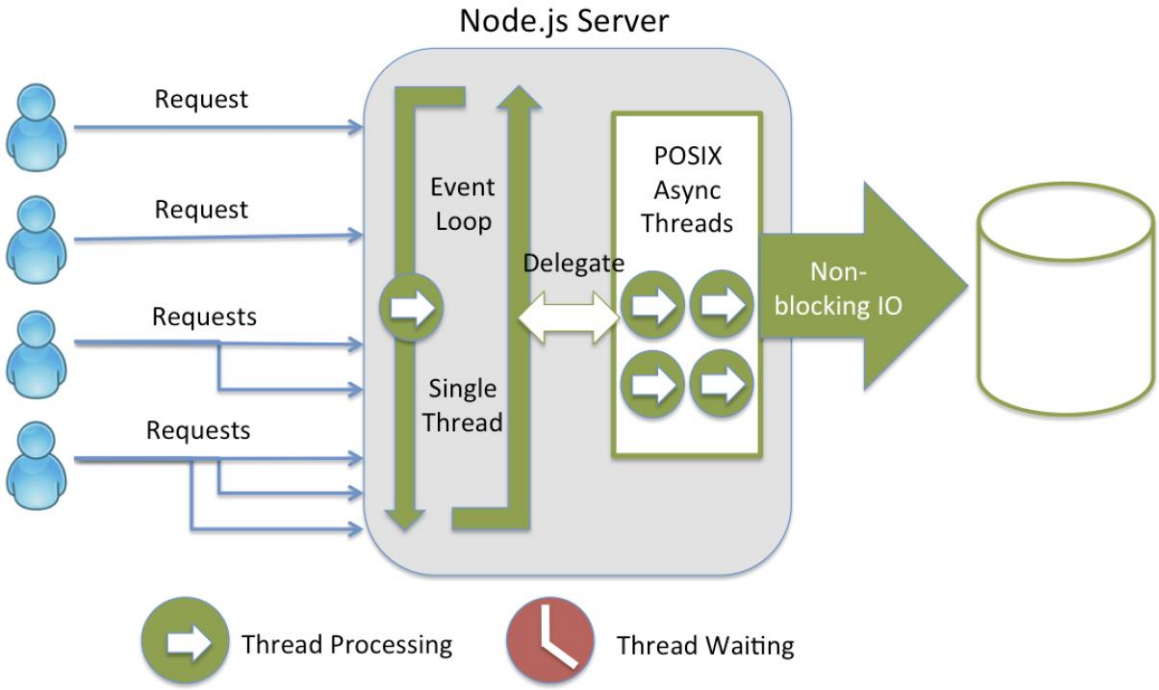

Figura 0-10 - Funcionamiento de Node.js. (11)

Node.js además de tener una naturaleza asíncrona que resulta muy útil para realizar tareas de forma concurrente también nos ofrece diversas ventajas frente a otros lenguajes de programación:

- Permite utilizar un mismo lenguaje (JavaScript) tanto en el cliente como en el servidor.
- Ofrece una gran gestión de paquetes de calidad gracias a NPM, que incluye un grandísimo número de librerías para realizar muchas tareas.
- Node.js permite gestionar una gran variedad de tareas que se pueden realizar en el servidor, como son el acceso a ficheros, a bases de datos, conexiones con cliente, etc.
- Nos ofrece un servicio de API REST bastante sencillo, que puede ser utilizado para realizar comunicaciones con una interfaz web, por ejemplo.

0.7 Alcance del proyecto

El cliente va a contar con una plataforma que ofrece diversos servicios. En primer lugar, contará con una página web donde podrá autorizar a León Literario a utilizar su cuenta de WhatsApp.

Una vez el usuario haya concedido el permiso para que León Literario gestione su cuenta podrá utilizar una herramienta para enviar iteraciones a la base de datos, que posteriormente serán ejecutadas en forma de una cola gestionada por Node.js. De esta misma forma la plataforma de León Literario podrá en todo momento, mientras el usuario o un administrador no revoque la autorización, monitorizar la actividad de la cuenta y almacenar las analíticas en una base de datos.

La plataforma consta también de una interfaz web para mostrar todas las analíticas dividiéndolas por usuario y a su vez por chat, es decir los usuarios tienen chats y estos a su vez contienen mensajes. Para facilitar la búsqueda de datos se podrá utilizar un buscador para filtrar información que está integrado en la página web.

El proyecto no abarca el análisis de dichas analíticas que deberá ser creado por un programa externo y ajeno a este proyecto.

El proyecto menciona en múltiples ocasiones la necesidad de rendimiento que, siendo cuidada durante el desarrollo del mismo, es una parte que también queda fuera, cumpliendo con un funcionamiento en condiciones normales de usuario y dejando como tema a tratar a posteriori la escalabilidad en entornos de alto tráfico. Aun así, se han realizado pruebas con varios usuarios y se ha comprobado el normal funcionamiento de la totalidad del software.

1. Planificación

En este apartado trataré de explicar la planificación que se ha llevado en el proyecto, mostrando la planificación inicial y la final con los cambios llevados a cabo durante el desarrollo.

Inicialmente para realizar la planificación se dividió el proyecto en pequeñas tareas, con un tiempo y unos recursos estimados para llevarlas a cabo y se establecieron las circunstancias para considerar las tareas terminadas.

La asignación de recursos humanos no se tuvo en cuenta ya que el trabajo se ha realizado por una sola persona así que todas las tareas son asignadas al autor de este proyecto.

La asignación del tiempo se realizó teniendo en cuenta que la persona que se encarga de la realización del proyecto no dispone de tanto tiempo como un trabajador normal.

En cuanto a los recursos materiales que se tuvieron en cuenta para la realización del proyecto son:

- Un PC con Windows y un Mac utilizados para desarrollar y probar el software.
- Un Smartphone Android para realizar pruebas de autorización con cuentas de WhatsApp, así como para probar la adaptación de las interfaces web y el correcto funcionamiento de estas en dispositivos móviles.
- Una suscripción a Microsoft Azure para alojar un servidor de SQL Server que permite realizar consultas de manera rápida y eficiente.

1.1 Planificación inicial

1.1.1 Planificación de tareas para familiarizarse con el proyecto.

Tabla 1-1 - Tarea 1: Explicación del proyeto

T1. Explicación detallada del proyecto	1 días
Descripción:	
Se explica el proyecto con más profundidad, estableciendo los objetivos y los resultados que se desean obtener. Se planifica el resto de las tareas.	
Recursos:	
Ninguno.	
Criterios de aceptación:	

El desarrollador del proyecto debe entender con precisión las ideas explicadas y hacer una valoración personal del proyecto. También debe participar en la asignación de los tiempos de cada tarea.

Tabla 1-2 - Tarea 2: Aprendizaje de GIT

T2. Aprendizaje de GIT	**2 días**
Descripción:	
Se explica el funcionamiento de un repositorio y que servicios ofrece. Se probarán los conocimientos de manera práctica realizando pequeñas tareas en GitLab.	
Recursos:	
Repositorio en GitLab GitKraken: Cliente de GIT	
Criterios de aceptación:	
Se revisarán las tareas realizadas en GitLab verificando que se han realizado correctamente y que el usuario entiende el uso de repositorios.	

Tabla 1-3 - Tarea 3: Familiarización con el uso de APIs

T3. Familiarización con el uso de APIs	**15 días**
Descripción:	
Se estudiará la documentación oficial de las APIs de Twitter y Facebook. Posteriormente se realizarán pequeños ejemplos de uso de estas APIs, como obtener y recibir de una cuenta de Facebook y Twitter.	
Recursos:	
APIs de Twitter y Facebook. Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	
Se dará por finalizada la tarea cuando tras evaluar los ejemplos de uso de APIs se compruebe que el desarrollador dispone de los conocimientos necesarios sobre el uso de estas APIs.	

Tabla 1-4 - Tarea 4: Actualización de Sync Twitter

T4. Adaptación de Sync Twitter a una nueva versión de la API	30 días
Descripción:	
Una vez que el usuario comprende y sabe utilizar APIs, se dispondrá a realizar la adaptación de un proyecto existente llamado Sync Twitter, remplazando la versión de la API de Twitter usada por la versión actual y adaptando la arquitectura de la plataforma a la nueva API.	
Recursos:	
API de Twitter. Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	
Se terminará esta etapa cuando tras realizar las pruebas pertinentes se compruebe el normal funcionamiento de Sync Twitter y su integración en la plataforma.	

Tabla 1-5 - Tarea 5 - Desarrollo de Sync Facebook

T5. Desarrollo de Sync Facebook	60 días
Descripción:	
Se creará un software de sincronización entre Facebook y la plataforma *León Literario* basándose en el modelo de Twitter. El proyecto constará de 3 subproyectos: uno para que los usuarios puedan autorizar el uso de su cuenta en la plataforma, otro que consistirá en realizar una estructura de cola para publicar en Facebook de forma automática las iteraciones de los usuarios y un tercero para realizar una monitorización de las cuentas de los usuarios y mostrarlas posteriormente en una interfaz web.	
Recursos:	
API de Facebook. Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	
Se dará por finalizada esta etapa cuando se compruebe el normal funcionamiento de la totalidad del software implementado. No se realizará la integración con la plataforma León Literario. Las pruebas se realizan fuera de la plataforma.	

Tabla 1-6 Tarea 6: Análisis del problema y búsqueda de soluciones

T6. Análisis del principal problema y búsqueda de soluciones	**2 días**
Descripción:	
Se analizará el principal problema que se nos presenta, WhatsApp no ofrece APIs y hay que buscar una solución.	
Recursos:	
Ninguno	
Criterios de aceptación:	
Se ofrecerán diversas soluciones y se estudiarán los pros y contras de cada una de ellas, llegando finalmente a una elegir solo una de las soluciones, la que se considere más viable.	

1.1.2 Planificación de un sistema de autorización para cuentas de WhatsApp.

Tabla 1-7 - Tarea 7: Análisis y eleccion de tecnologias para Sync_Whatsapp_Auth

T7. Análisis y elección de tecnologías para Sync_WhatsApp_Auth	**4 días**
Descripción:	
Después de tomar la decisión de crear APIs propias de WhatsApp, se empezará por elegir el software y las tecnologías que se utilizarán para poder implementar una forma de autorizar a la plataforma el uso de cuentas de WhatsApp.	
Recursos:	
Internet Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	
Se finalizará esta etapa cuando, después de haber analizado las diferentes herramientas que se presentan y haber hecho una valoración, se haya seleccionado el método más viable para la realización de este subproyecto.	

Tabla 1-8 - Tarea 8: Implementación del Software de Sync_WhatsApp_Auth

T8. Implementación del Software de Sync_WhatsApp_Auth	20 días
Descripción:	
Una vez seleccionado el entorno y las herramientas para la realización del subproyecto se procederá a su implementación y desarrollo hasta darlo por finalizado.	
Recursos:	
Selenium SQL Server Node.js HTML5 con Framework Materialize, CSS propio y JavaScript Socket.IO	
Criterios de aceptación:	
Se considera terminada una vez que se compruebe que se ha finalizado y la totalidad del software funciona correctamente, permitiendo a los usuarios dar autorización a *León Literario* para utilizar sus cuentas de WhatsApp.	

1.1.3 Planificación de un sistema automático de envío de iteraciones para cuentas de WhatsApp.

Tabla 1-9 - Tarea 9: Análisis y elección de tecnologías para Sync_WhatsApp_Queue

T9. Análisis y elección de tecnologías para Sync_WhatsApp_Queue	4 días
Descripción:	
Una vez los usuarios puedan dar su autorización, se analizará cómo crear un software que permita el envío automático de iteraciones de usuarios programas en el tiempo o de la plataforma, para ser ejecutadas posteriormente en cuentas de WhatsApp. Se valorarán las distintas herramientas que pueden ser utilizadas para este fin.	
Recursos:	
Internet Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	

Se finalizará esta etapa cuando después de haber analizado las diferentes herramientas que se presentan y haber hecho una valoración, se haya seleccionado el método más viable para la realización de este subproyecto.

Tabla 1-10 - Tarea 10: Implementación del Software de Sync_WhatsApp_Queue

T10. Implementación del Software de Sync_WhatsApp_Queue	**20 días**
Descripción:	
Una vez seleccionado el entorno y las herramientas para la creación del subproyecto se procederá a su implementación y desarrollo hasta darlo por finalizado.	
Recursos:	
Selenium SQL Server Node.js Java	
Criterios de aceptación:	
Se considera terminada una vez que se compruebe que se ha finalizado y la totalidad del software funciona correctamente, permitiendo que tanto los usuarios como *León literario* pueden ejecutar iteraciones programadas en el tiempo en cuentas de WhatsApp previamente autorizadas en la plataforma.	

1.1.4 Planificación de un sistema de monitorización para cuentas de WhatsApp.

Tabla 1-11 - Tarea 11: Análisis y elección de tecnologías para Sync_WhatsApp_Anatytics

T11. Análisis y elección de tecnologías para Sync_WhatsApp_Anatytics	**4 días**
Descripción:	
Una vez los usuarios de *León Literario* y la misma plataforma puede realizar iteraciones sobre cuentas de WhatsApp, se estudiará cómo crear un software que permita la monitorización de las cuentas de WhatsApp asociadas a la plataforma. Se valorarán las distintas herramientas que pueden ser utilizadas para este fin.	
Recursos:	
Internet Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	
Se finalizará esta etapa cuando después de haber analizado las diferentes herramientas que se presentan y haber hecho una valoración, se haya seleccionado el método más viable para la realización de este subproyecto.	

Tabla 1-12 - Tarea 12: Implementación del Software de Sync_WhatsApp_Analytics

T12. Implementación del Software de Sync_WhatsApp_Analytics	**20 días**
Descripción:	
Una vez seleccionado el entorno y las herramientas para la realización del subproyecto se procederá a su implementación y desarrollo hasta darlo por finalizado.	
Recursos:	
NW.js SQL Server	
Criterios de aceptación:	
Se considera terminada una vez que se compruebe que se ha finalizado y la totalidad del software funciona correctamente, permitiendo que a plataforma *León Literario* pueda monitorizar en todo momento las cuentas de WhatsApp asociadas.	

1.1.5 Planificación de una interfaz Web donde se muestren los resultados obtenidos de la monitorización

Tabla 1-13 - Tarea 13: Creación de la interfaz Web

T13. Creación de la interfaz Web	8 días
Descripción:	
Se procederá a diseñar una interfaz web que deberá mostrar los datos obtenidos en la monitorización, mostrándolos de forma clara, y con la opción de realizar búsquedas simples.	
Recursos:	
HTML5 con Framework Materialize, CSS propio y JavaScript SQL Server Entorno de desarrollo para realizar las pruebas	
Criterios de aceptación:	
Se finalizará esta etapa cuando se haya terminado la interfaz y se realicen las pruebas oportunas que demuestren el normal funcionamiento de todos los elementos que se presentan.	

1.1.6 Evaluación, refactorización y corrección de errores

Tabla 1-14 - Tarea 14: Evaluación, refactorización y corrección de errores

T14. Evaluación, refactorización y corrección de errores	20 días
Descripción:	
En esta etapa se llevará a cabo una evaluación constante de todo el software del proyecto, así como una refactorización continua.	
Recursos:	
La totalidad del proyecto. Entorno de desarrollo para realizar las pruebas.	
Criterios de aceptación:	
Se considera terminada una vez que se dé por finalizado este proyecto. Se realizarán las pruebas constantemente para verificar el normal funcionamiento en todo momento.	

1.1.7 Métricas y sistemas de evaluación futuras

Tabla 1-15 - Tarea 15: Métricas y sistemas de evaluación futura

T15. Métricas y sistemas de evaluación futura	5 días
Descripción:	
Definición de posibles métricas de parámetros para mejorar la mantenibilidad del software, estableciendo en que puntos se podría mejorar.	
Recursos:	
La totalidad del proyecto.	
Criterios de aceptación:	
Valorar las posibles métricas y comprobar que efectivamente mejorarían el rendimiento o la mantenibilidad del software.	

1.1.8 Documentación del proyecto

Tabla 1-16 - Tarea 16: Documentación del proyecto

T16. Documentación del proyecto	4 días
Descripción:	
Durante esta etapa se llevará a cabo toda la documentación del proyecto. Debe realizarse durante el transcurso de la realización del proyecto.	
Recursos:	
Internet Proyecto	
Criterios de aceptación:	
Se finalizará cuando se haya terminado la documentación y el dueño del producto verifiqué que se ha realizado de forma correcta.	

La estimación inicial es que el proyecto requeriría de cerca de unos 210 días laborables para ser realizado completamente. A continuación, en la figura 1.1 se muestra el diagrama de Gantt de la planificación inicial del proyecto.

1.2 Planificación final

Una vez se ha terminado el proyecto, se ha comparado la planificación que se estableció inicialmente con la planificación final. Se concluye con que la planificación inicial es diferente a la planificación final, esto se debe principalmente a que en algunas tareas se ha reducido el tiempo y en otras se ha aumentado.

Como conclusión, la planificación inicial en conjunto se ha realizado correctamente porque la planificación final solamente ha aumentado 5 días el tiempo estimado para la finalización del proyecto.

A continuación, una tabla con los desfases en la planificación.

Tabla 1-17 - Desfase en la planificación

Tarea	Desfase	Observaciones
T1	0 días	El responsable del producto explicó el proyecto de manera detallada y fue entendido perfectamente.
T2	0 días	Se comprendió el uso de repositorios, y se realizaron tareas básicas con ellos.
T3	+2 días	Debido a que no tenía ningún conocimiento sobre APIs, esta etapa se alargó más.
T4	-5 días	Debido a la buena documentación existente esta etapa se desarrolló antes de lo previsto.
T5	+3 días	Debido a la cantidad de iteraciones que se pueden publicar en Facebook esta etapa se alargó un poco.
T6	0 días	Se comprendió el principal problema y se analizaron las posibles soluciones en el plazo estimado.
T7	+2 días	Debido a que en un principio no se encontraron las tecnologías necesarias esta etapa se alargó.
T8	-4 días	Esta etapa finalizó antes de lo previsto debido a que se invirtió bastante tiempo.
T9	-1 días	Se decidió utilizar muchas de las tecnologías de la etapa de planificación T8 lo que redujo el tiempo.
T10	-2 días	Se realizó antes del tiempo estimado porque ya se habían utilizado anteriormente esas tecnologías.
T11	+2 días	Esta etapa finalizó después de lo previsto debido a que se decidió usar herramientas nuevas.

T12	+8 días	Debido a un cambio total respecto a las tecnologías utilizadas anteriormente aumento bastante el tiempo de desarrollo respecto a la planificación inicial.
T13	-2 días	El tiempo de desarrollo disminuyó debido a que ya se había creado una interfaz de este tipo anteriormente.
T14	+5 días	El tiempo aumento debido a que el proyecto se alargó un poco.
T15	0 días	Se estimaron nuevas métricas y se desarrollaron nuevas pruebas en el plazo estimado.
T16	+5 días	Esta etapa se alargó como consecuencia de que también se alargó el proyecto.

1.3 Diagramas Gantt de la planificación

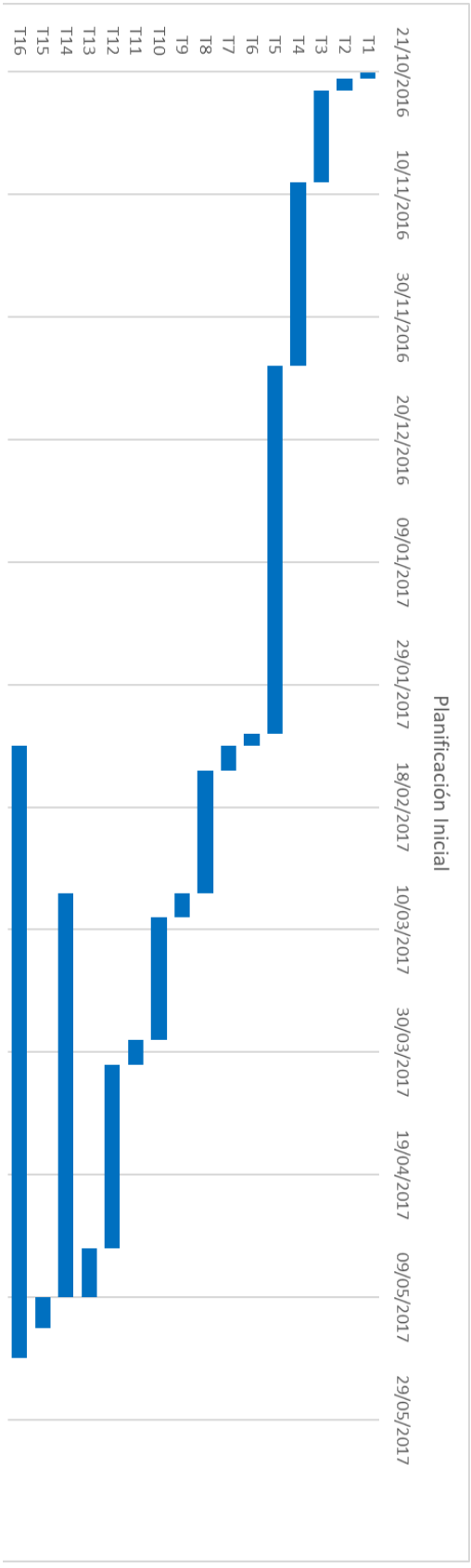

Figura 1-1 - Diagrama de Gantt Inicial

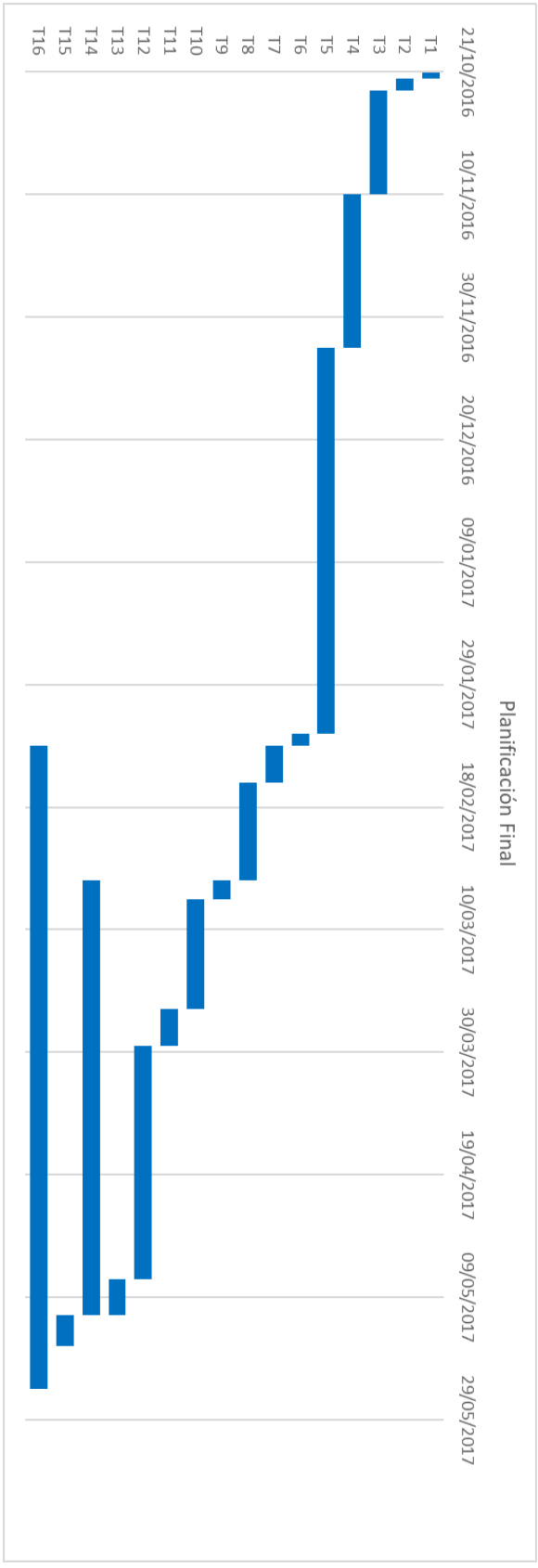

Figura 1-2 - Diagrama de Gantt Final

2. Objetivos

2.1 Objetivos del dueño del producto

En la primera reunión con el dueño del producto, se trató el tema por encima junto con las necesidades que tenía. Me enseño la plataforma *León Literario* y la integración de una herramienta llamada *Sync Twitter* para la sincronización de Twitter con dicha plataforma mediante APIs. Principalmente necesitaba implementar una solución que realizase la misma función que *Sync Twitter*, pero usando APIs de WhatsApp. El dueño del producto me asignó un proyecto para la creación de software que pudiese realizar una sincronización real entre *León Literario* y WhatsApp.

El proyecto, al igual que *Sync Twitter* debía estar dividido en tres partes diferenciadas e independientes:

1. Por un lado, se requería la implementación de software que permitiera a los usuarios de León Literario dar su autorización para que esta plataforma pudiese gestionar sus cuentas de WhatsApp. Esto consistía, de alguna manera, en guardar algún tipo de credencial de la cuenta de WhatsApp que posteriormente permitiera el acceso a dicha cuenta.
2. Por otro lado, se requería de la creación de una herramienta sencilla que permitiera realizar consultas sobre una base de datos para simplificar el almacenamiento de iteraciones. Posteriormente se requería un software que recogiese las iteraciones de la base de datos, creando una cola de iteraciones, y que utilizando una API pudieran ser ejecutadas sobre WhatsApp.
3. Por último, se requería un software capaz de realizar una monitorización constante de las cuentas de WhatsApp asociadas a la plataforma. Esto consistiría en obtener todas las iteraciones que existiesen en la cuenta, y posteriormente crear un hilo que recogiera las nuevas iteraciones que se produjesen mientras se continuara monitorizando dicha cuenta.

La gran cantidad de datos de la monitorización que se preveía que iba a ser almacenada en la base de datos supuso un problema. Ver todo ese volumen de datos en forma de texto plano era muy costoso y una tarea que llevaría bastante tiempo.

La solución a este problema fue la creación de una interfaz que mostraría los datos obtenidos en la monitorización de forma ordenada. Es decir, la interfaz debía mostrar en primer lugar una lista con los usuarios asociados a la plataforma, cada usuario debería tener una lista de chats, y cada chat contendría los mensajes. Aun así, se decidió añadir un buscador para facilitar aún más la tarea y poder filtrar usuarios, chats y mensajes.

2.2 Requisitos funcionales

A continuación, se procederá a describir algunos de los requisitos funcionales que fueron previamente acordados en la reunión llevada a cabo con el dueño de la empresa.

Tabla 2-1 - Requisitos funcionales

Numero	Requerimiento	Prioridad
RF1	Obtener la autorización del usuario	5
RF2	Desautorizar al usuario.	5
RF3	Crear una cola de iteraciones que se realizarán sobre WhatsApp.	5
RF4	Obligar a procesar la cola cuando sea necesario.	3
RF5	Ejecutar de forma simultánea varios *batch* de iteraciones. Mientras se ejecuta un conjunto de iteraciones sobre una cuenta, puede ejecutarse otro conjunto sobre otra cuenta diferente al mismo tiempo.	4
RF6	Posibilidad de programar en el tiempo los mensajes que se van a enviar.	4
RF7	Obtener mensajes de texto de los chats de WhatsApp.	5
RF8	Obtener todos los tipos de iteraciones que se pueden enviar en WhatsApp como son imágenes, vídeos, ubicación, documentos, audio y contactos.	3
RF9	Obtener información del destinatario del chat que se está analizando.	5
RF10	Opción de monitorizar solamente una única conversación indicando el número de teléfono o el nombre grupo.	5
RF11	Implementación de un modo en el que, una vez terminada la monitorización, el software está a la espera de que se produzca una nueva iteración en la cuenta para capturarla.	5
RF12	Uso de un chat auxiliar para solucionar un problema en la monitorización de nuevos mensajes. El problema consiste en que si el chat que recibe el nuevo mensaje se encuentra abierto este no será capturado.	3

RF13	Botón que permita detener la monitorización y mostrar la interfaz donde se eligen los usuarios para ser monitorizados.	2
RF14	Interfaz web para mostrar los resultados de la monitorización de forma ordenada.	5
RF15	Botón para revocar la autorización del usuario y eliminarlo de la interfaz.	4
RF16	Buscador para realizar un filtrado del contenido en función del texto introducido.	3
RF17	Registrar en todo momento la actividad tanto por consola como en un fichero de *log*.	5

2.3 Requisitos no funcionales

Los requisitos no funcionales especifican criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos.

A continuación, se muestran en la tabla los más relevantes para el proyecto:

Tabla 2-2 - Requisitos no funcionales

Número	Requerimiento	Prioridad
RNF1	Desempeño	4
RNF2	Confiabilidad	4
RNF3	Usabilidad	5
RNF4	Adaptabilidad	5
RNF5	Rendimiento	4
RNF6	Seguridad	5
RNF7	Escalabilidad	4
RNF8	Usabilidad	4
RNF9	Eficiencia	4
RNF10	Facilidad de mantenimiento	3

RNF11	Portabilidad	4
RNF12	Estabilidad	4
RNF13	Disponibilidad	5
RNF14	Interfaz	3
RNF15	Coste	3
RNF16	Accesibilidad	3
RNF17	Facilidad de uso	4

2.4 Objetivos técnicos

Una vez concluida la definición de los objetivos del dueño del producto y los requisitos funcionales y no funcionales es el momento de definir y describir los objetivos desde un punto de vista técnico.

Se han encontrado cuatro objetivos principales que se explicarán más detalladamente a continuación.

2.4.1 Autorización de cuentas de WhatsApp

La autorización es el primer paso necesario en el desarrollo del proyecto. Esta autorización consistirá principalmente en habilitar una interfaz web visible para los usuarios de la plataforma, que permitirá realizar la autorización de sus cuentas para permitir a la plataforma tener el control total de estas. Al usuario se le avisa de este hecho, con mensaje donde se le indica que tanto su cuenta como sus datos asociados a esta serán totalmente accesibles por parte de la plataforma. Esto quiere decir que se podrá tanto enviar mensajes como monitorizar la actividad cuenta.

La autorización contará con tres métodos imprescindibles:

- **Autorización**: Consistirá en un método donde el usuario al acceder a una interfaz se le mostrará un código QR proveniente de WhatsApp Web que deberá escanear para poder iniciar una sesión. Después de ser escaneado, al usuario se le mostrará a través de la interfaz un mensaje, acompañado de su número de teléfono y su foto de perfil, donde se indicará que fue autorizado con éxito.

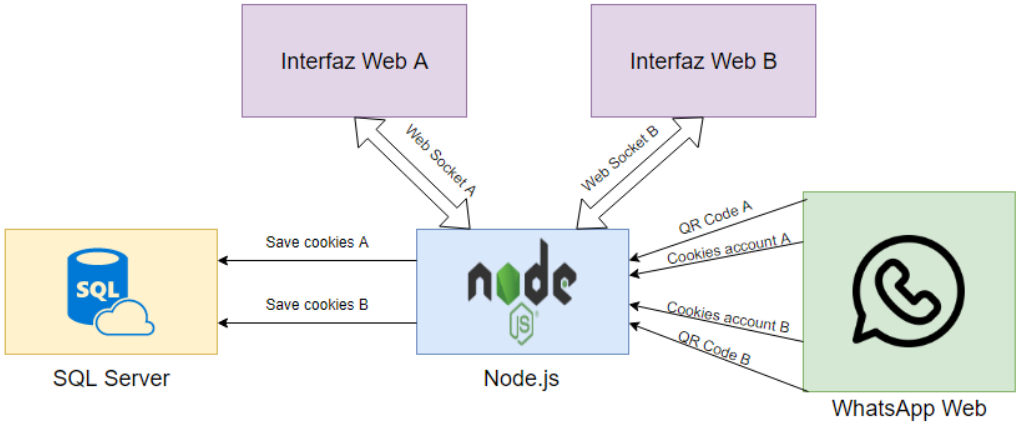

Figura 2-1 - Diagrama del sistema de autorización

- **Desautorización:** Consistirá en desvincular la cuenta de WhatsApp del usuario con la plataforma, de esta manera *León Literario* ya no podrá acceder a dicha cuenta. Cuando un usuario se desautorice se borrarán sus credenciales del servidor. Cabe decir que existe otra forma de desautorización ajena a la plataforma que solo podrá llevar a cabo el usuario. Esta otra forma consiste en acceder a la cuenta de WhatsApp desde la aplicación móvil y eliminar la sesión correspondiente a la plataforma.

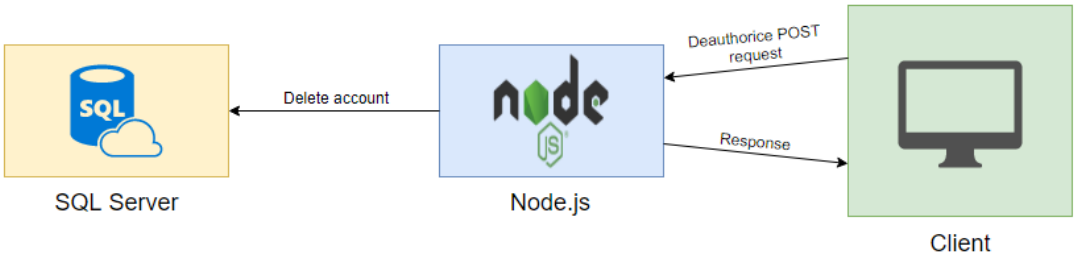

Figura 2-2 - Diagrama de desautorización

- **Comprobar autorización:** Consistirá en una función que comprobará si están almacenados en la base de datos las credenciales para un usuario determinado. Si están se emitirá un mensaje que indicará que la cuenta del usuario está autorizada, pero si por el contrario no existen credenciales válidas para ese usuario se indicará que su cuenta no está autorizada en la plataforma.

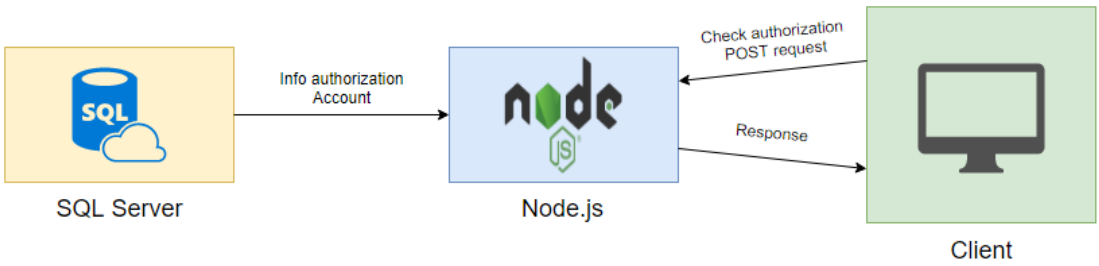

Figura 2-3 - Diagrama de comprobación de autorización

2.4.2 Envío automático de iteraciones sobre WhatsApp

Para poder crear un software que permitiese la ejecución de iteraciones en cuentas de WhatsApp, se ideó una estructura de cola basándose en una solución anterior para lograr el mismo objetivo con cuentas de Twitter. Esta estructura de cola irá seleccionando iteraciones almacenadas en la base de datos y las ejecutará en conjuntos de iteraciones denominados *batchs*.

Un *batch* consiste en un conjunto de iteraciones de una misma cuenta que se corresponden todas ellas con el mismo destinatario. Esto se complica si añadimos también iteraciones programadas, debido a que a la hora de crear un *batch* se comprobará que la iteración tiene como fecha de ejecución una fecha anterior o igual a la actual, y en caso negativo no se incluirá en el *batch* dicha interacción. Una vez creado el *batch*, Node.js ejecutará sobre la cuenta de WhatsApp correspondiente todas sus iteraciones.

Cada vez que una iteración sea realizada se obtendrá un código que informará de si se ejecutó con éxito o si por el contrario hubo algún error en el proceso. Se actualizará el estado de la iteración en la base de datos, indicando el resultado de la ejecución.

La estructura de cola debe funcionar de forma concurrente, esto es fácil debido a la naturaleza asíncrona de Node.js. Cada cierto tiempo se comprobará si existen nuevas iteraciones, en caso afirmativo se creará el *batch* correspondiente y se ejecutará sobre WhatsApp. Mientras se realiza la ejecución de un *batch* se deberá continuar con la comprobación de nuevas iteraciones y si las hay se ejecutara un nuevo *batch*, aunque el anterior no haya finalizado su ejecución.

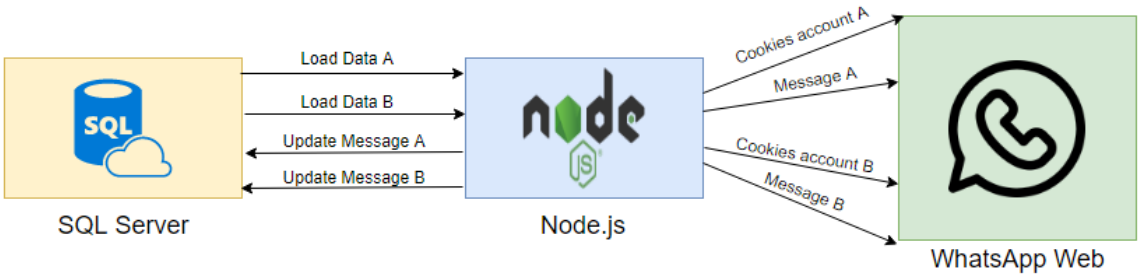

Figura 2-4 - Diagrama del envío de iteraciones

Posteriormente, debido a la pesadez que suponía introducir las iteraciones en la base de datos manualmente, se diseñaría una herramienta simple que permitirá, a través de una interfaz sencilla, automatizar la tarea de ejecutar consultas sobre la base de datos. La herramienta deberá permitir al usuario seleccionar, por un lado:

- El número asociado a una cuenta de WhatsApp autorizada en la plataforma. Se mostrará una lista con todos los números.
- Un campo de texto donde se introducirá el número de teléfono destinatario.
- Un campo de texto para introducir la fecha en la cual se desea enviar el mensaje.
- Por último, un campo de texto donde se deberá escribir el mensaje que será enviado posteriormente.

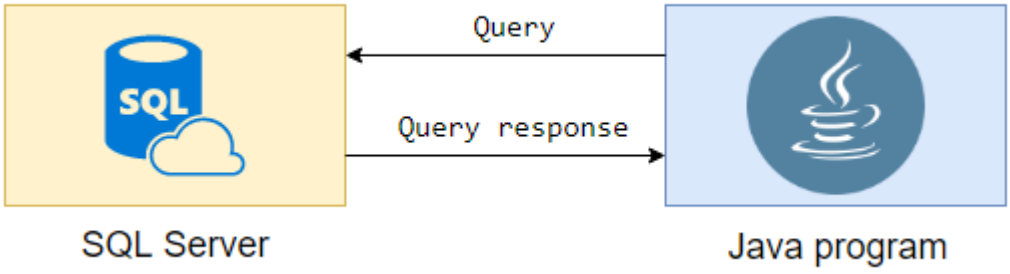

Figura 2-5 - Diagrama de herramienta para ejecutar consultas

2.4.3 Captura y monitorización de los datos

La captura y monitorización de datos es una tarea que surge como una necesidad para poder ofrecer a los usuarios contenido basado en sus intereses y mejorar su experiencia. Es decir, a partir de los datos recopilados, se crearán diferentes perfiles de usuario basándose en los informes realizados con los datos almacenados.

Para poder realizar un software que realizase una captura y monitorización constante de cuentas de WhatsApp asociadas a León Literario se ideó un sistema que primeramente mostrase una interfaz sencilla en la que se pudiese escoger una cuenta de WhatsApp para ser monitorizada. Se deberá mostrar una lista con las cuentas asociadas, y cada cuenta deberá contener su número de teléfono asociado y su foto de perfil. También se dispondrá de un botón, que al ser pulsado y si se ha seleccionado una cuenta, abrirá la interfaz encargada de la monitorización.

La interfaz de monitorización consistirá en una aplicación, que de manera automática, irá abriendo cada uno de los chats y obteniendo todas sus iteraciones independientemente del tipo que sean.

Cuando haya terminado de recoger todas las iteraciones de todos los chats, la aplicación permanecerá a la espera de nuevas iteraciones para capturarlas.

Todos los datos que se recojan de las iteraciones capturadas deberán ser almacenados en la base de datos.

En todo momento se podrá detener la monitorización de la cuenta, para realizar esta acción se introducirá un botón que al ser pulsado cerrará la interfaz de monitorización y abrirá la interfaz en la que se podrá escoger de nuevo una cuenta para comenzar la monitorización.

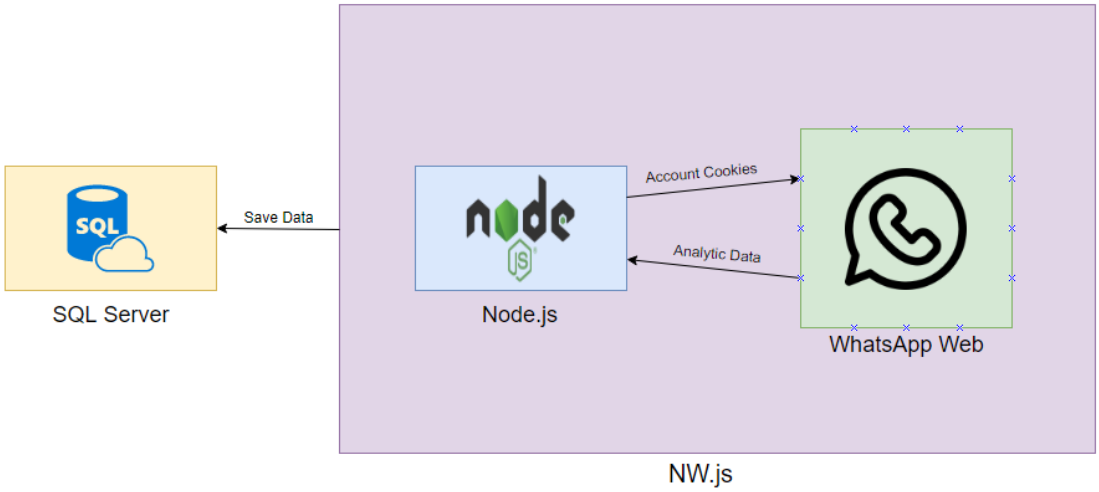

Figura 2-6 - Diagrama del sistema de monitorización

2.4.4 Interfaz Web para mostrar los datos capturados

En un principio, no se estableció este objetivo, pero después de realizar varias monitorizaciones de cuentas se observó que la cantidad de datos almacenados hacía muy difícil la tarea de ordenar y comprender esos datos. Esto suponía un nuevo problema, cuya solución sería la creación de un nuevo software que permitiera mostrar ese volumen de datos de una forma clara y ordenada.

Se ideó una interfaz web cuya función seria recoger todo ese volumen de datos de la base de datos y mostrarlo de la forma más ordenada y limpia posible.

En primer lugar, se debería mostrar una lista con los usuarios que hubieran sido autorizados en la plataforma. Junto a cada usuario se mostraría la fecha en la que fue autorizado y un botón que permitiese desautorizarlo. También era útil la implementación de un buscador para poder filtrar estos usuarios por su número de teléfono.

Después de elegir un usuario, debería mostrarse una lista con los chats de dicho usuario, tanto si es una persona como si es un grupo. En cada elemento de la lista de chats se mostraría el nombre asignado a ese chat por el usuario en WhatsApp y la foto de perfil de este. Era necesaria la implementación de un buscador para poder filtrar los chats por su nombre.

Cada chat deberá contener las iteraciones tanto enviadas por el usuario, como también las recibidas por este. Se deberán diferenciar de forma clara las iteraciones que han sido enviadas de las que han sido recibidas, para esta tarea se puede tomar una solución similar a la que utiliza WhatsApp. También se implementará un buscador para poder filtrar los mensajes de texto por su contenido.

Por último, la ventana de chats y la de mensajes deben tener un botón que redirija al usuario a la página anterior.

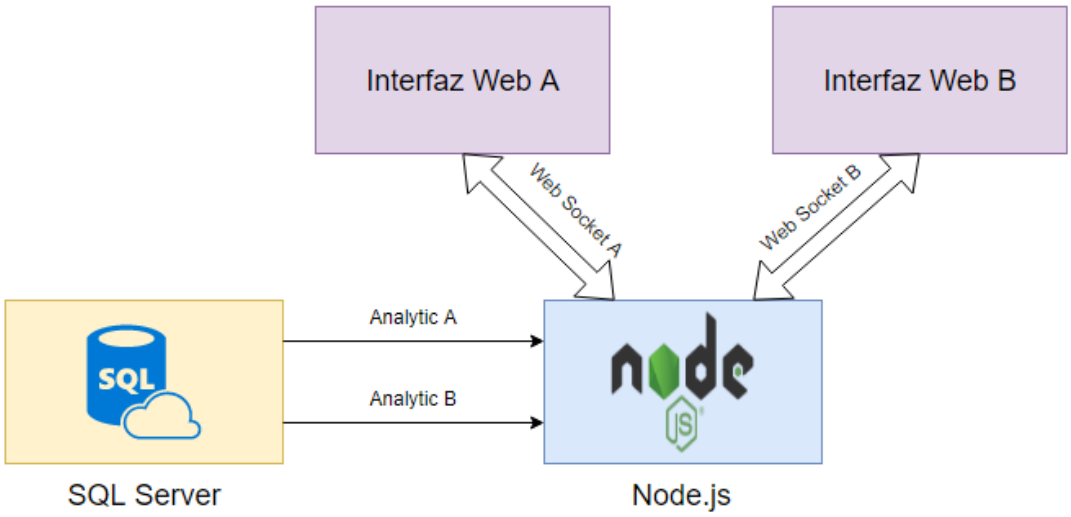

Figura 2-7 - Diagrama de la interfaz web

2.4.5 Especificaciones comunes para todas las interfaces de tipo Web

Todas las interfaces de tipo Web que se presentan en el proyecto estarán desarrolladas con el *framework* Materialice e implementarán el CSS propio que se considere necesario, al igual que los scripts necesarios para conseguir el correcto funcionamiento de todo lo relacionado con el diseño de estas.

Las interfaces serán responsivas, es decir deberán adaptarse perfectamente a cualquier tamaño, pudiendo ofrecer la misma funcionalidad en todo tipo de pantallas, ya sean de tipo escritorio, de tablets o de dispositivos móviles más pequeños.

Para conseguir esto se utilizarán elementos diseñados en forma de tarjetas que son reconocibles en cualquier formato de pantalla o resolución y se deberán de adaptar siguiendo la siguiente especificación: en pantallas de tipo escritorio deberán mostrarse un máximo de 3 columnas con las tarjetas que contengan los elementos correspondientes; en pantallas de tablets se mostrarán un máximo de dos columnas únicamente; y finalmente en pantallas de dispositivos móviles más pequeños se mostrará únicamente una columna que deberá ocupar todo el ancho de la pantalla.

3. Resultados

Para tratar este apartado es preciso contar con varias capturas de pantalla donde se podrán observar de forma clara los resultados obtenidos.

3.1 Autorización

3.1.1 En el lado del cliente

En primer lugar, supongamos que un usuario desea autorizar a *León Literario* el uso de su cuenta de WhatsApp. Para realizar esta acción deberá acceder a esta interfaz:

Figura 3-1 - Interfaz de autorización cargando

Figura 3-2 - Interfaz de autorización cargada

Como se puede observar en figura 3-2, al usuario se le muestra un código QR, que una vez sea escaneado por la aplicación de WhatsApp de su Smartphone se procederá a realizar la autorización.

Cuando se haya completado la autorización, el usuario podrá observar una ventana como esta:

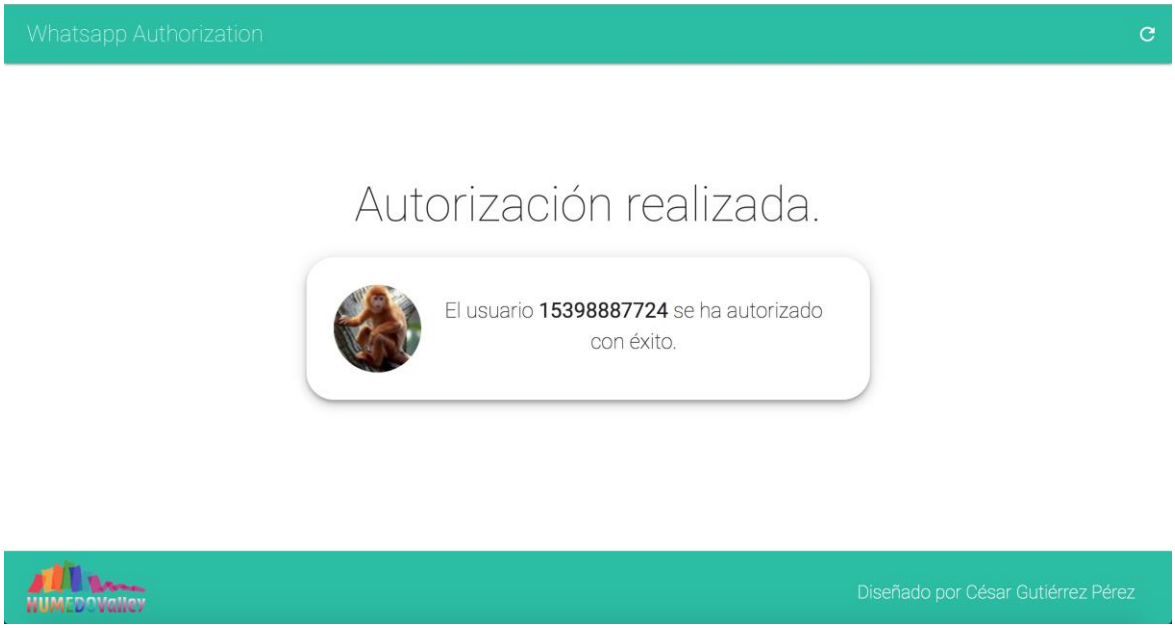

Figura 3-3 - Autorización completada

Como se puede observar se indica al usuario que se ha sido autorizado y por lo tanto se procedido con éxito a su integración en la plataforma y ya puede cerrar la página web.

Cabe recordar que se cumplen perfectamente las especificaciones comunes para todas las interfaces web del proyecto como se observa en algunas imágenes.

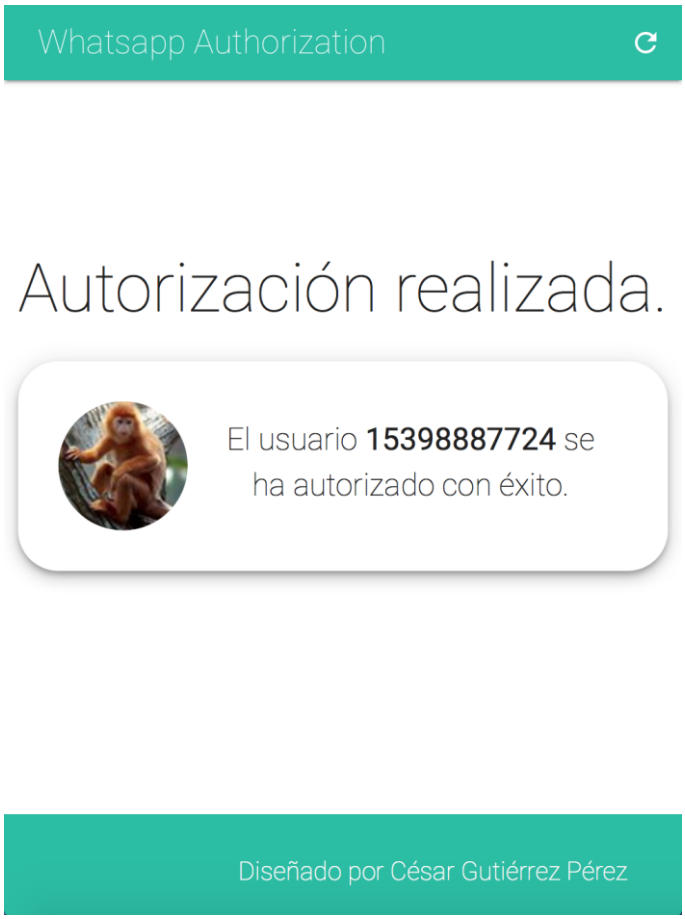

Figura 3-4 - Interfaz de autorización en dispositivo móvil

3.1.2 En el lado del servidor

En un primer lugar contamos con una terminal en la que se mostrarán las acciones que realiza el cliente en la interfaz de autorización.

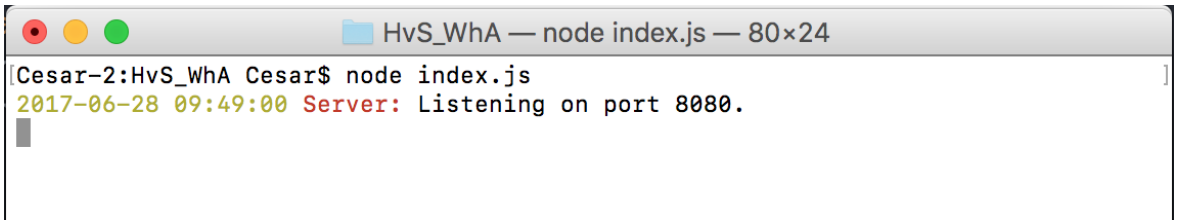

Figura 3-5 - Terminal Node para la autorización

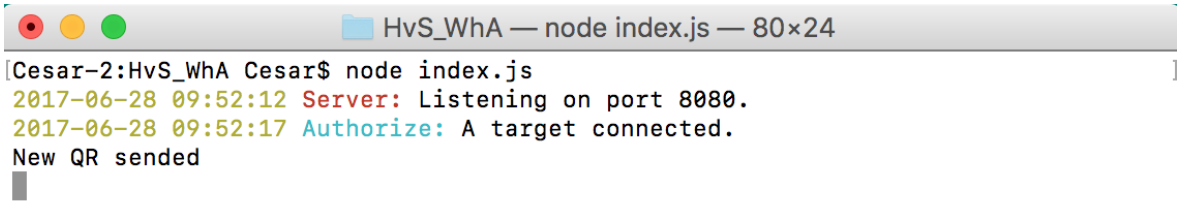

Figura 3-6 - Terminal Node -> Enviando código QR

Como se puede observar en la figura 3-6 un cliente ha accedido a la interfaz web, y se le ha enviado un código QR para que pueda autorizarse.

En la siguiente imagen vemos que el cliente ya ha sido autorizado, por lo tanto, sabemos que el cliente ha procedido a escanear con WhatsApp el código QR que le enviamos anteriormente.

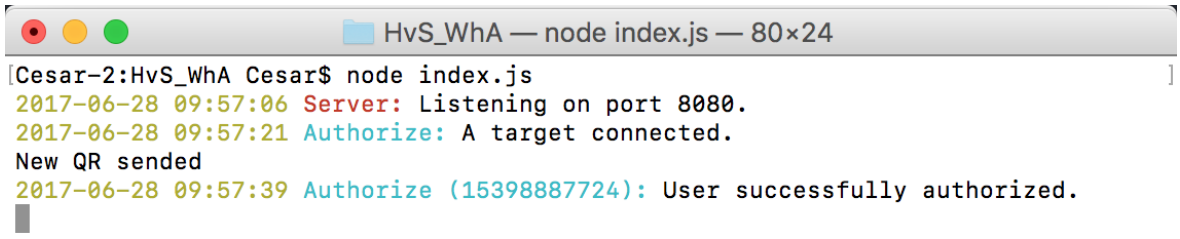

Figura 3-7 - Terminal Node -> Autorización completa

La figura 3.8 muestra que el cliente ha visto que se ha realizado la autorización con éxito y ha cerrado la ventana de la interfaz de autorización.

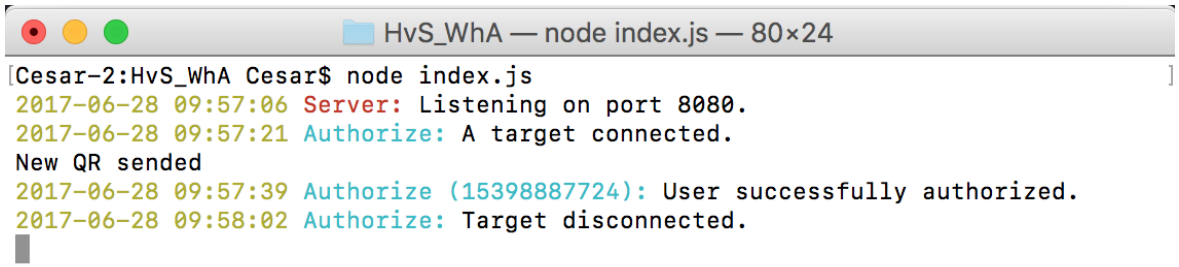

Figura 3-8 - Terminal Node -> Desconexión del cliente

3.2 Ejecución automática de iteraciones sobre WhatsApp

3.2.1 En el lado del cliente

En primer lugar, supongamos que un usuario quiere enviar un mensaje de forma automatizada a través de la plataforma. Para realizar esta acción simplemente deberá utilizar la herramienta que se muestra a continuación en la imagen:

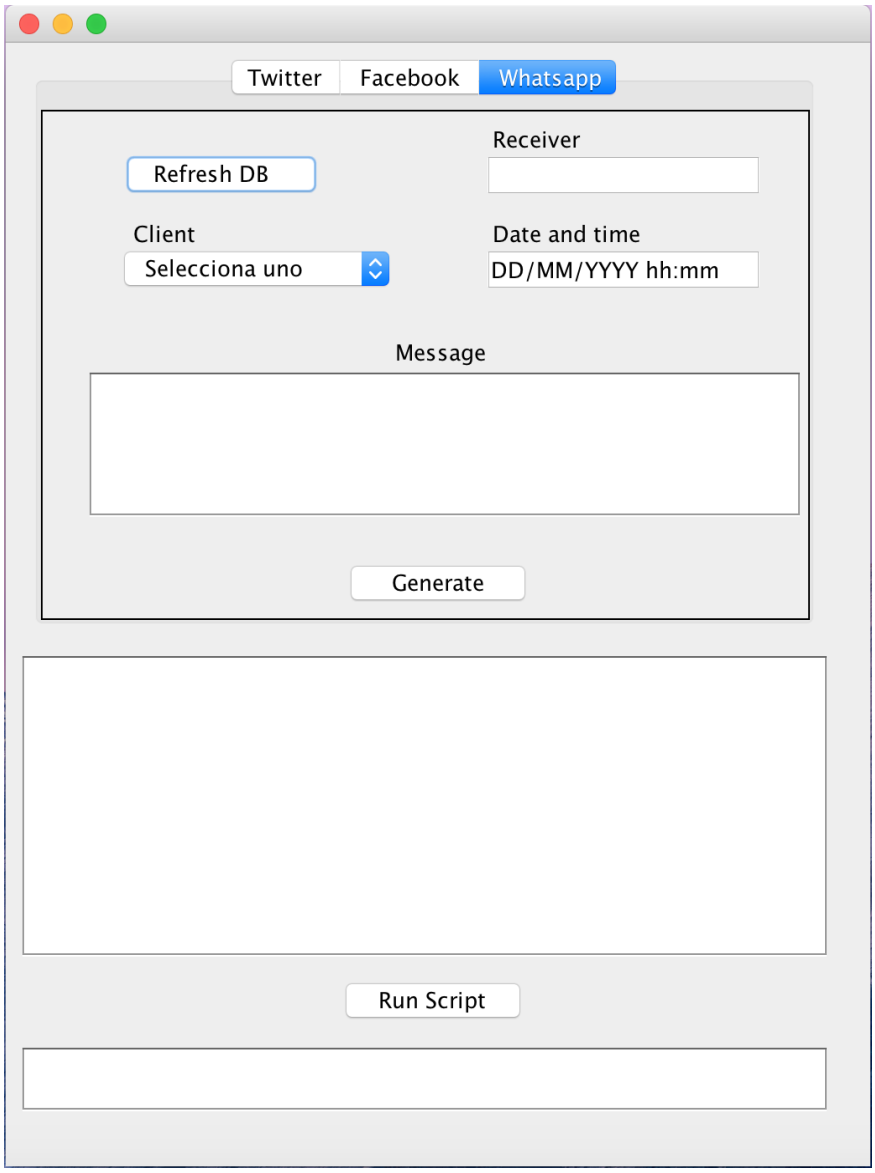

Figura 3-9 - Herramienta para ejecutar consultas

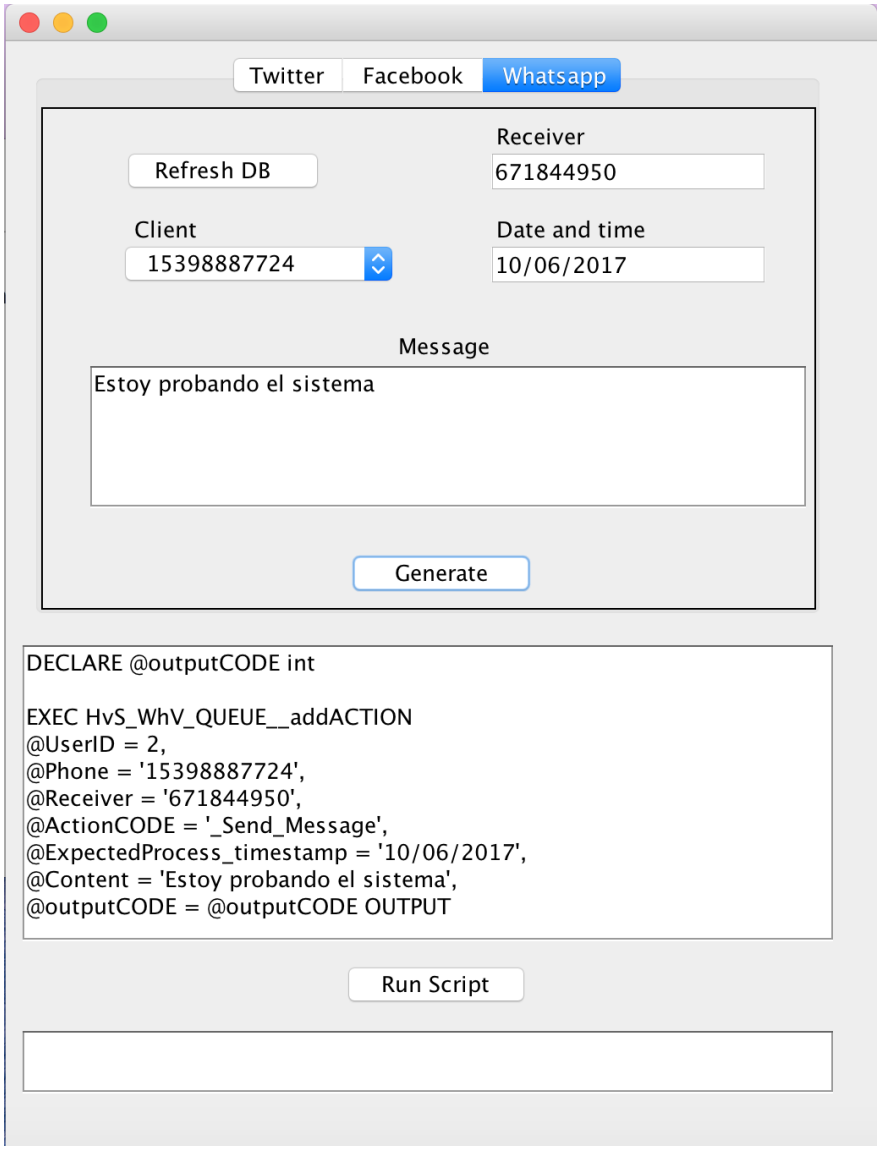

Figura 3-10 - Herramienta para ejecutar consultas rellena

El usuario deberá rellenar todos los campos de la herramienta de forma correcta con la información que él desee. Después pulsara el botón *generate* para crear el script correspondiente para sus datos introducidos.

Posteriormente el usuario deberá pulsar el botón *run script* para ejecutar el script sobre la base de datos. El usuario verá un mensaje que le indicará si el script fue ejecutado con éxito.

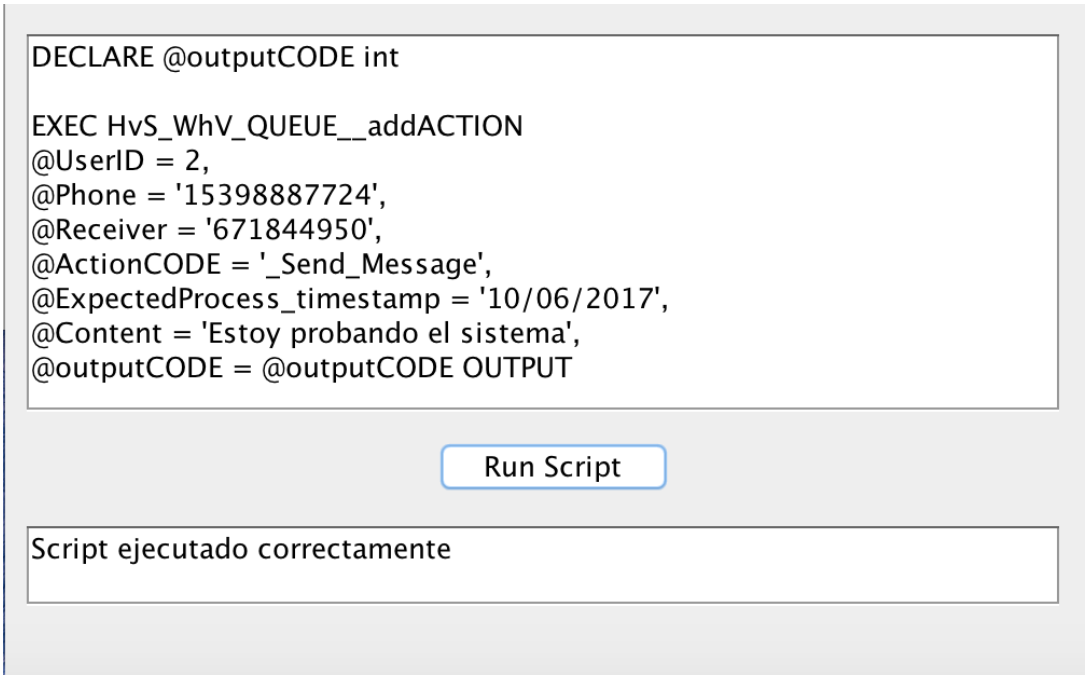

Figura 3-11 - Herramienta para generar consultas -> Consulta ejecutada

3.2.2 En el lado del servidor

Contamos con una terminal en la que se mostrarán las iteraciones en forma de *batch* que se van ejecutando sobre WhatsApp. Cada cierto tiempo se comprobará si hay iteraciones nuevas en la base de datos.

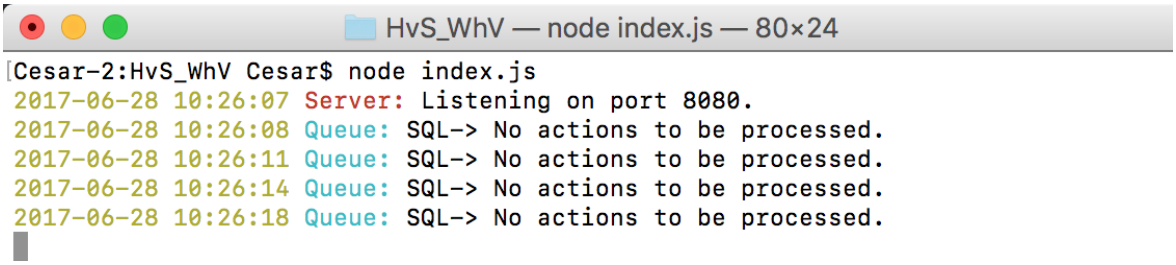

Figura 3-12 - Terminal Node -> Buscando iteraciones

Cuando se haya creado un *batch* a partir de iteraciones se indicará en la terminal, y se procederá a su ejecución de forma automática sobre un navegador que implemente WhatsApp Web.

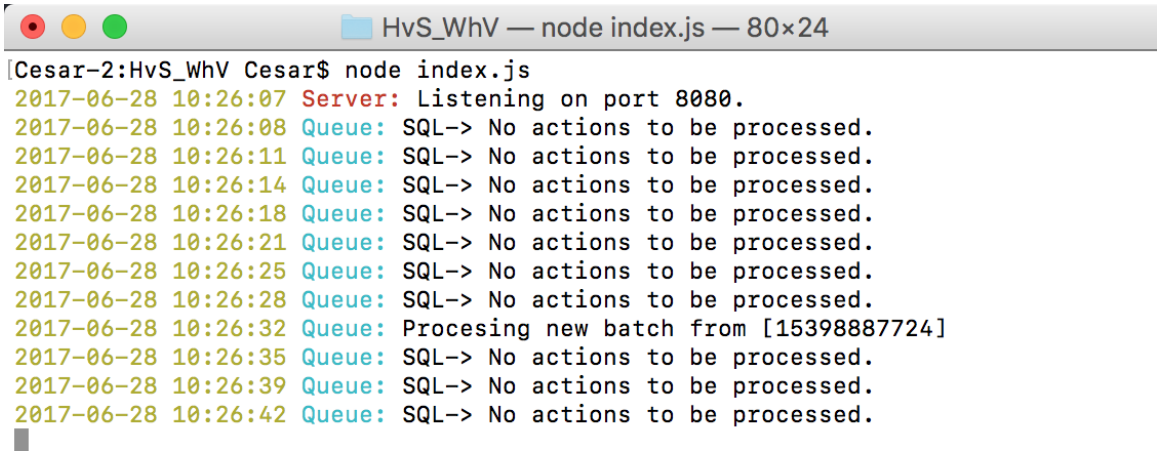

Figura 3-13 - Terminal Node -> Ejecutando batch

Cuando el *batch* finalice se cerrará el navegador y se indicará que ha finalizado en la terminal.

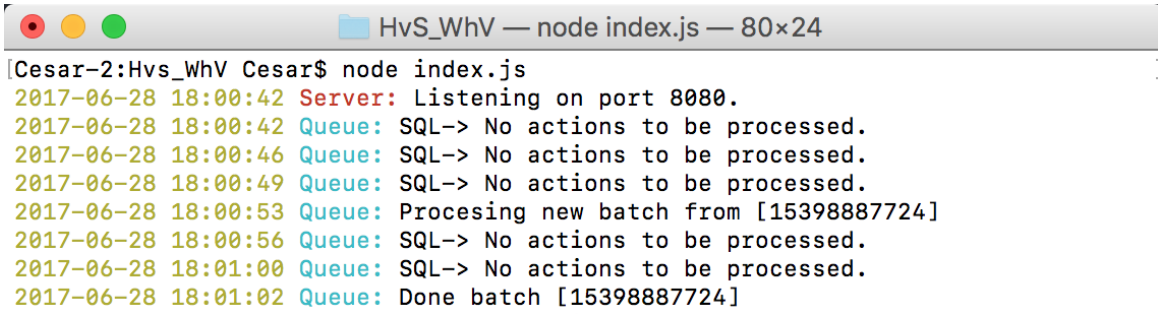

Figura 3-14 - Terminal Node -> Batch finalizado

3.3 Captura y monitorización de datos

Para realizar una monitorización de datos se deberá ejecutar la herramienta para este fin. La herramienta consiste en una aplicación de escritorio.

En primer lugar, se muestra una lista desplegable con las cuentas autorizadas en la plataforma, es preciso seleccionar una para iniciar la monitorización de esta.

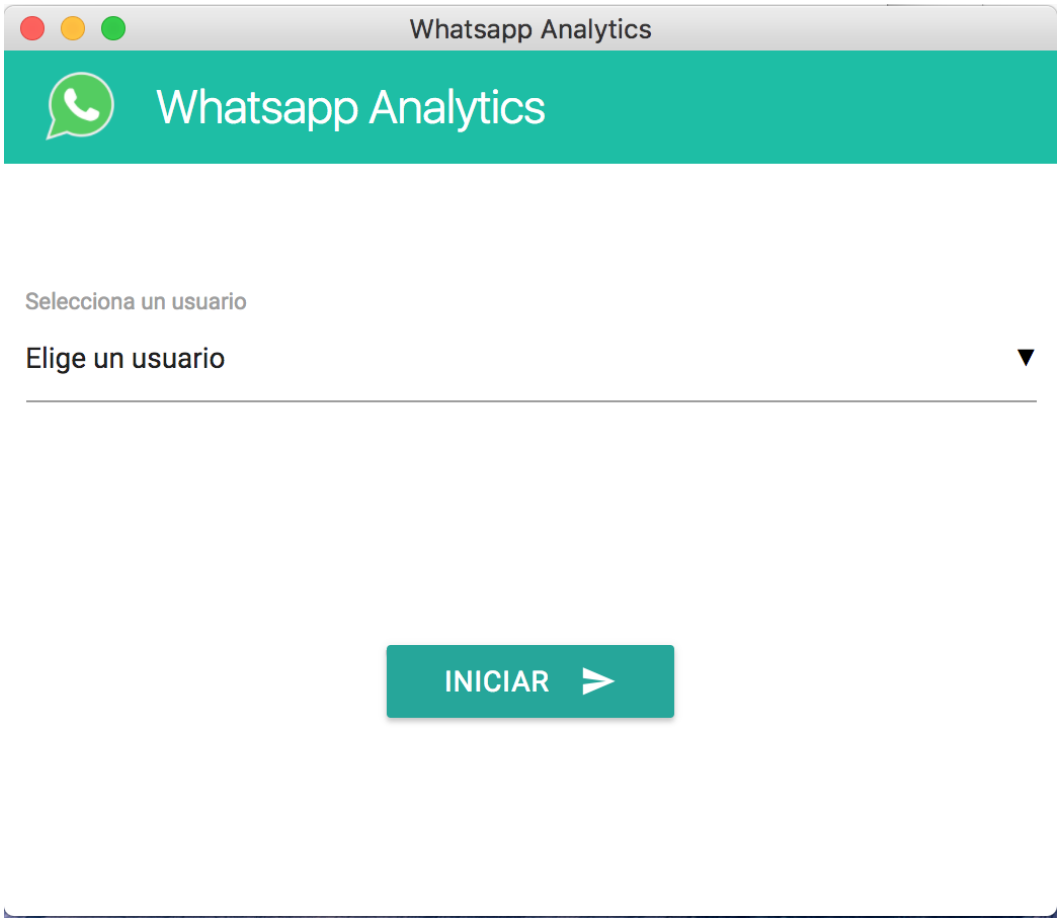

Figura 3-15 - Interfaz con lista de usuarios

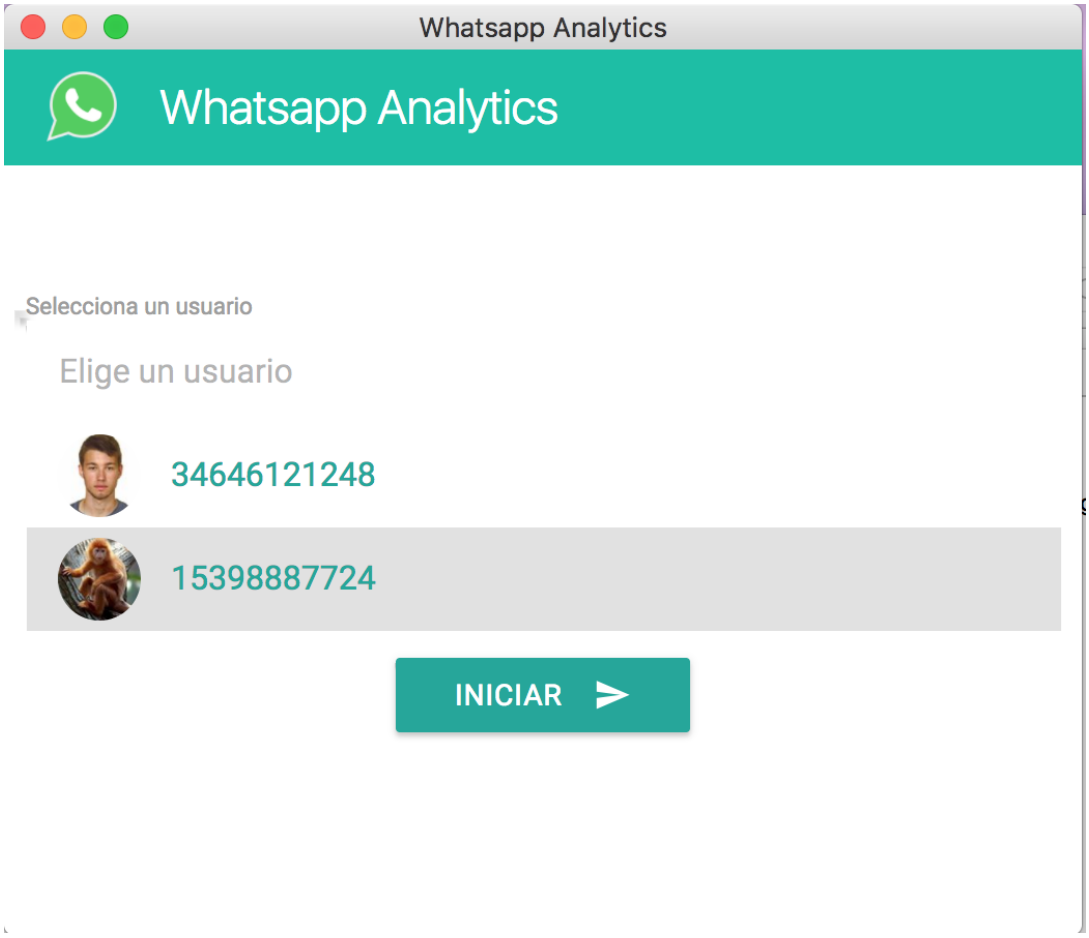

Figura 3-16 - Lista de usuarios visible

Una vez seleccionada una cuenta se mostrará la interfaz de monitorización, donde el usuario en todo momento puede observar el proceso de la captura de datos.

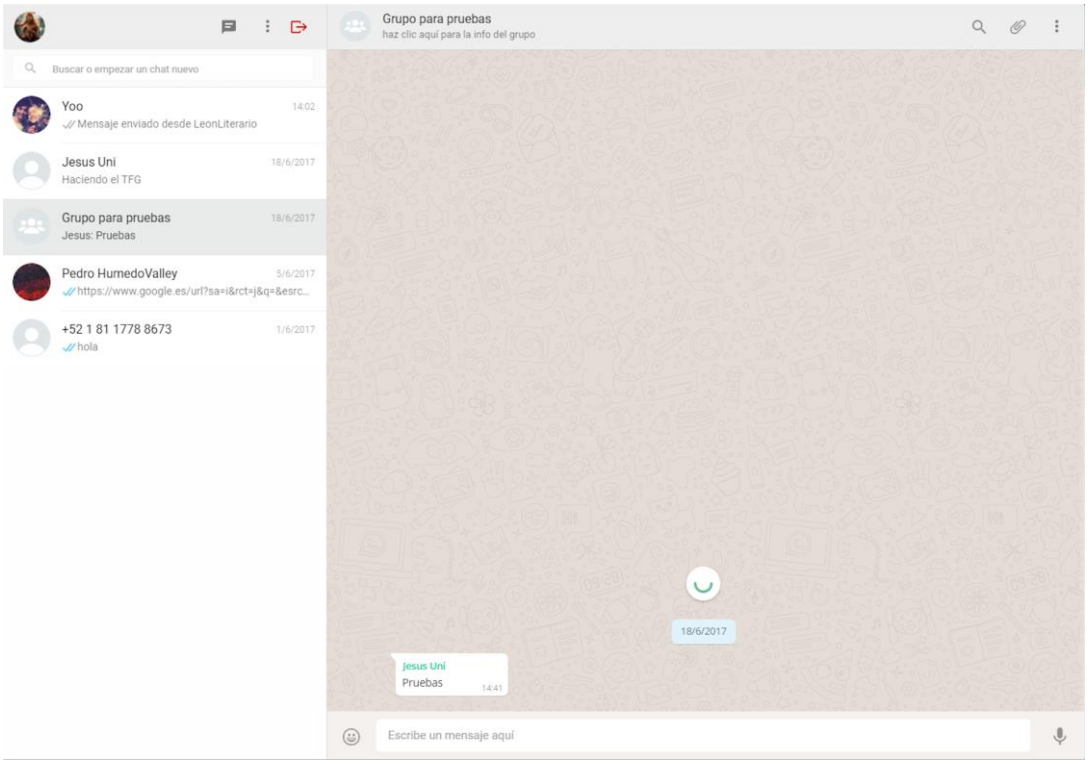

Figura 3-17 - Monitorización de cuentas

Una opción que se contempla es monitorizar únicamente a un grupo o un usuario en concreto, para ello solo bastaría con indicar su número de teléfono o su nombre asociado en WhatsApp y la aplicación de forma automática solo monitorizara dicho chat.

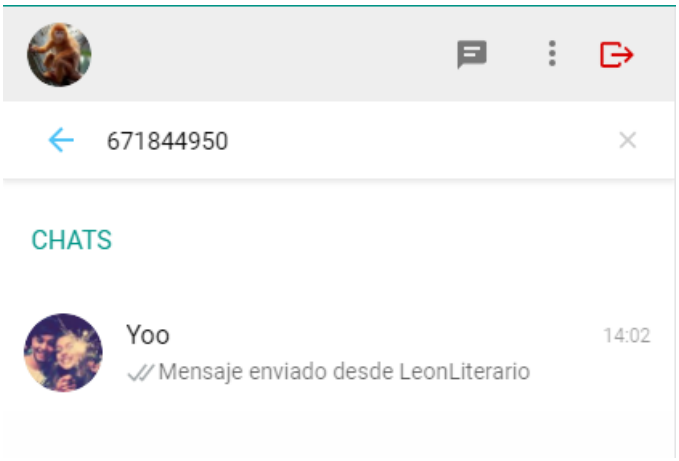

Figura 3-18 - Monitorización de un único usuario

Cuando la captura de datos llegue a su fin, se abrirá un chat auxiliar que se usa para solucionar un problema que explicaré más adelante.

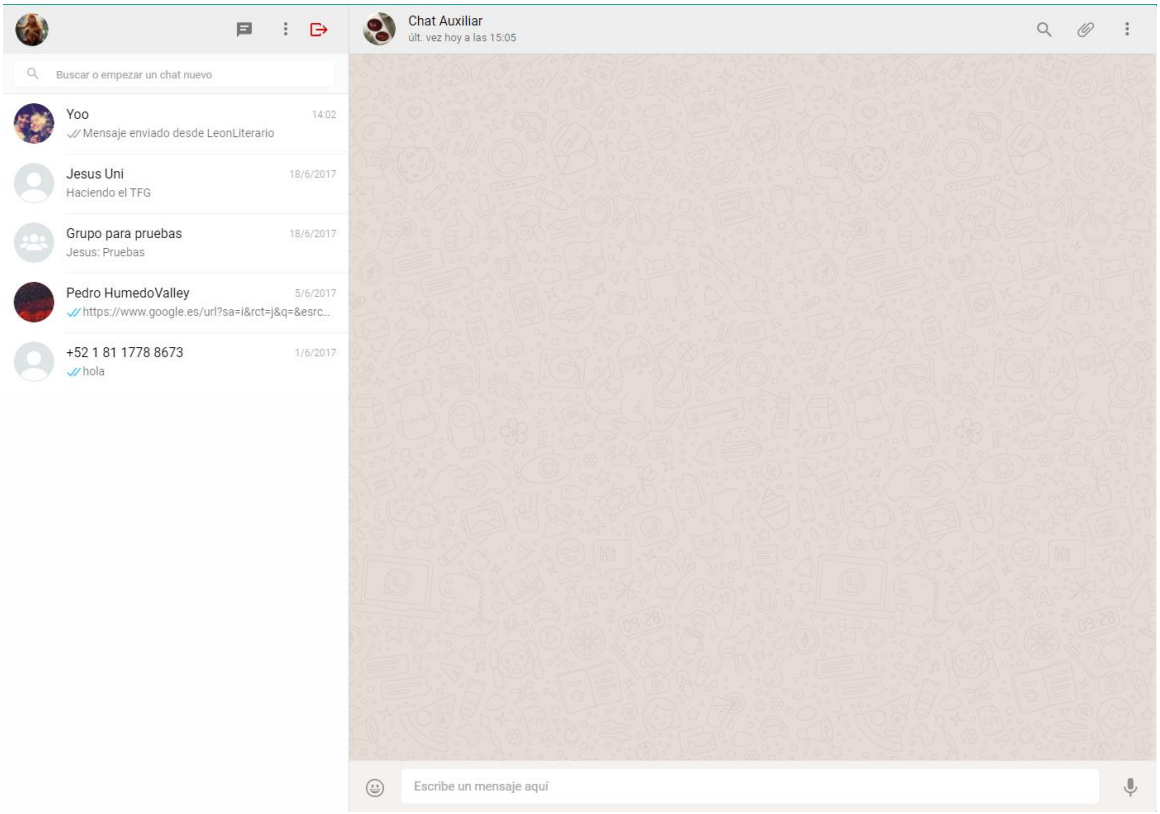

Figura 3-19 - Chat auxiliar

La aplicación estará a la espera de nuevas iteraciones y en el momento que se reciba una nueva iteración se procederá inmediatamente a capturarla. Después se volverá a abrir el chat auxiliar como se puede observar en las figuras 3-20 y 3-21.

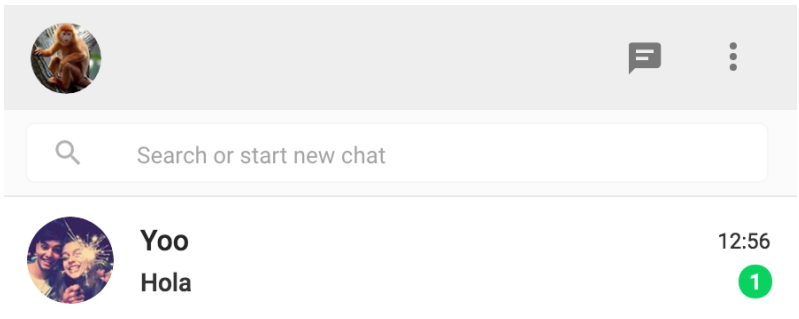

Figura 3-20 - Mensaje nuevo

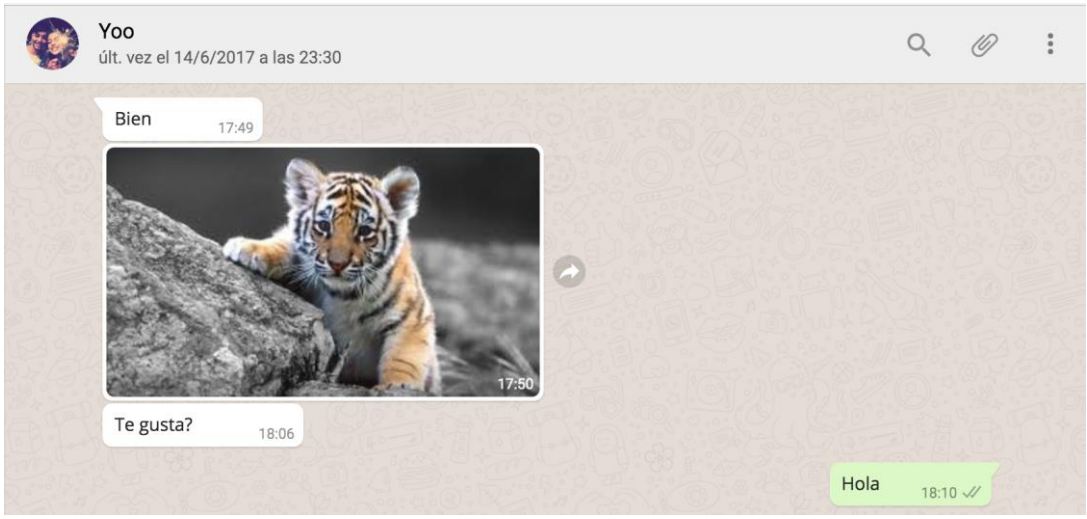

Figura 3-21 - Monitorización de nuevos mensajes

3.4 Interfaz web para mostrar datos de cuentas monitorizadas

Como expliqué con anterioridad esta interfaz simplifica la búsqueda de datos capturados en la monitorización de las distintas cuentas de WhatsApp.

En la siguiente imagen se muestran las cuentas de usuario asociadas a la plataforma de *León Literario*.

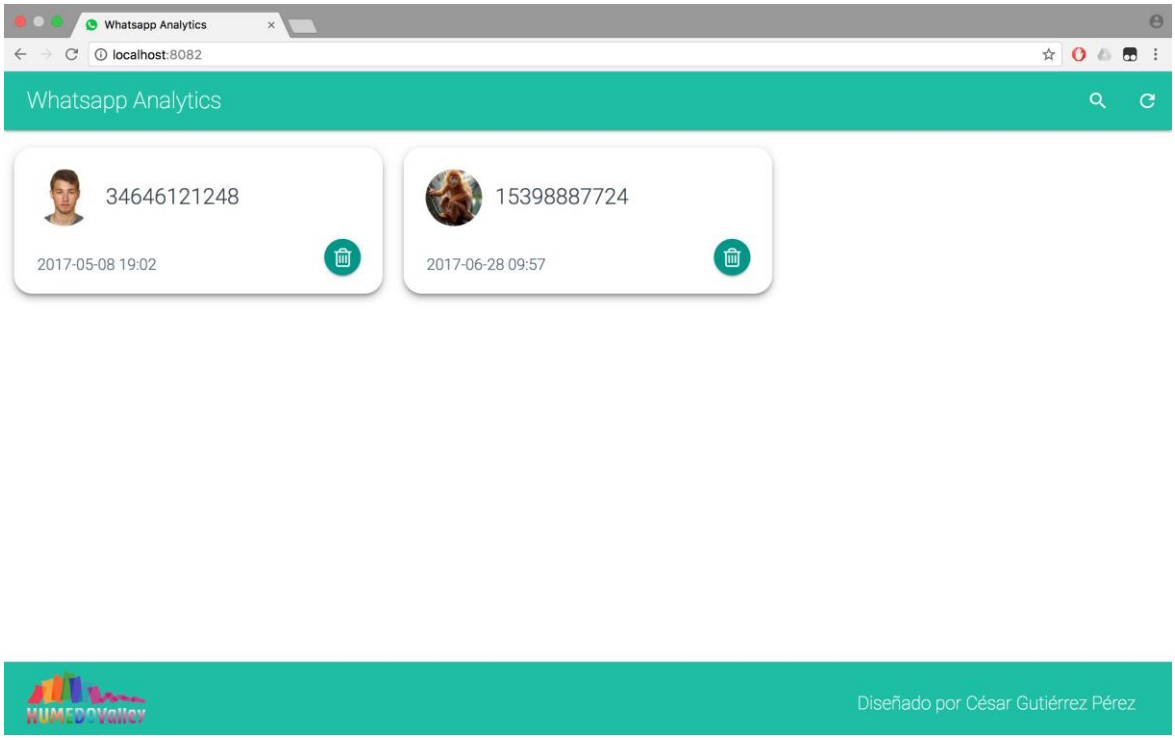

Figura 3-22 - Interfaz con los usuarios monitorizados

Si puede utilizar el buscador para simplificar la búsqueda eliminando los resultados que no me interesan, como se observa en las figuras 3-23 y 3-34.

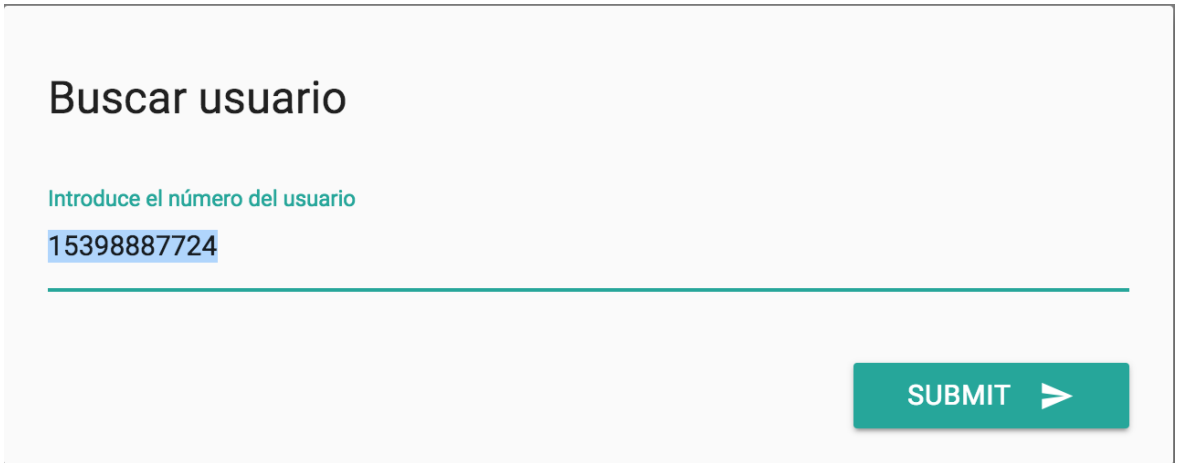

Figura 3-24 - Buscador de usuarios

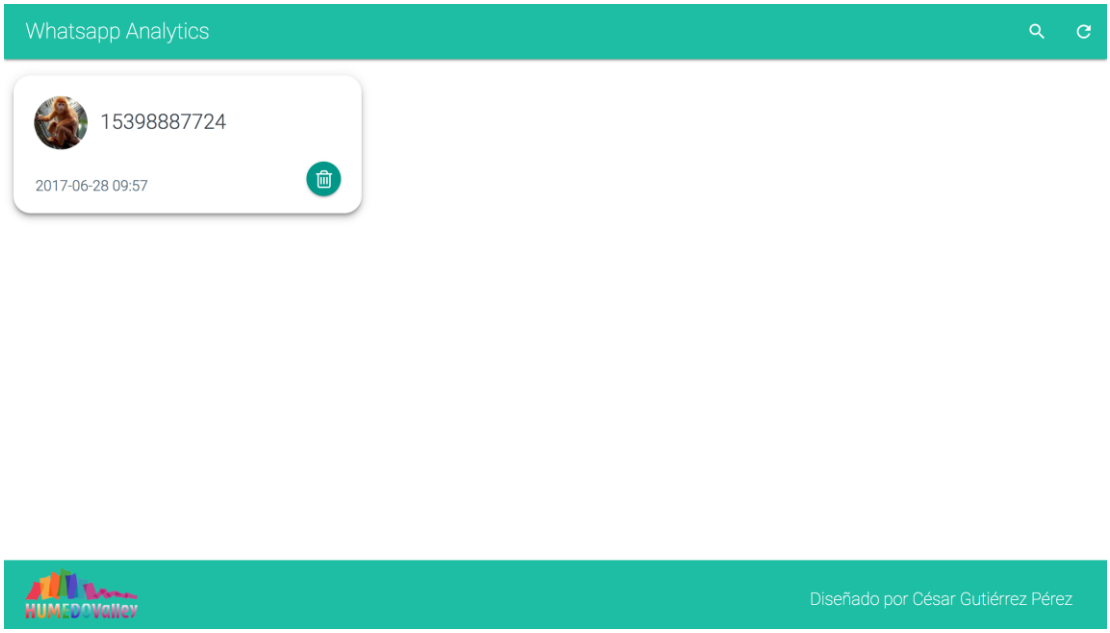

Figura 3-23 - Resultado de la busqueda por usuario

Desde esta misma ventana se permite desautorizar cuentas de WhatsApp asociadas simplemente pinchando un botón que se encargará de realizar esta acción.

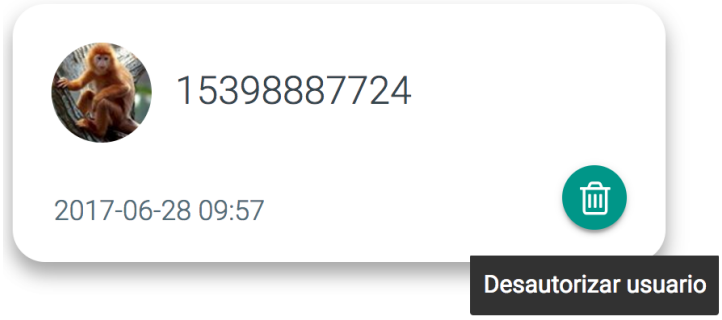

Figura 3-25 - Botón para desautorizar

Una vez elijamos la cuenta que queremos monitorizar veremos la lista de chats asociados a esta cuenta que hay almacenados en la base de datos.

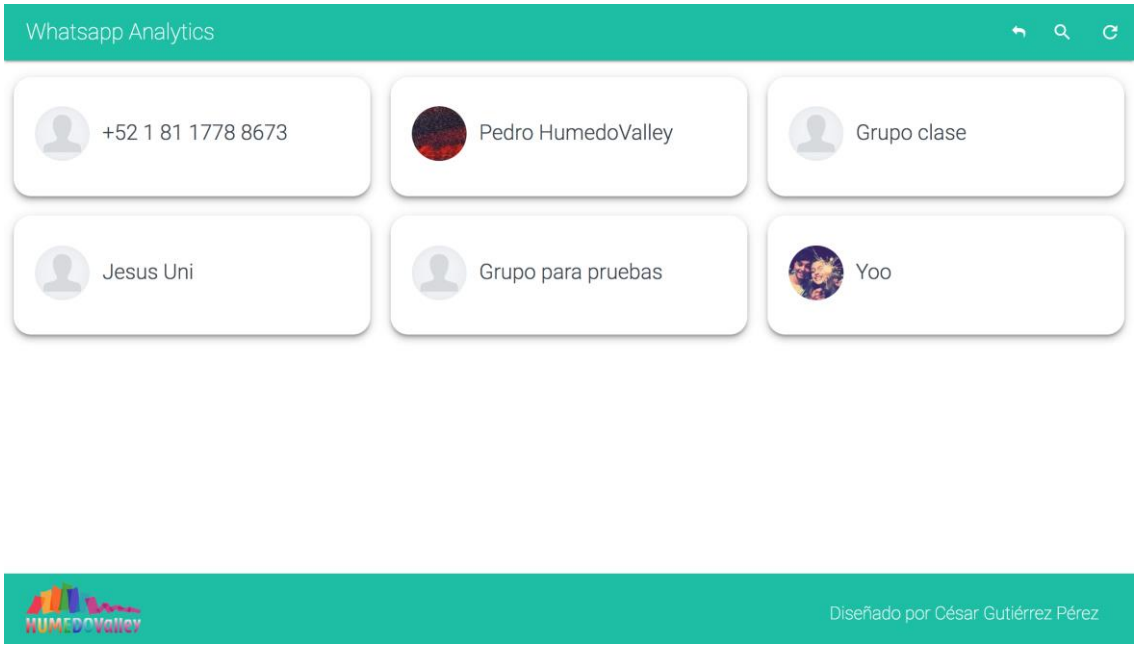

Figura 3-26 - Interfaz con lista de chats

Utilizando el buscador podemos simplificar la búsqueda de manera que solo se mostrarán los datos que contengan un texto que introduzcamos. En la imagen que aparece hemos realizado una búsqueda con el texto “Grupo”.

Figura 3-27 - Resultado de búsqueda en los chats

Si queremos ver los mensajes de un chat bastaría únicamente con hacer click en el chat correspondiente y ya se mostraría la lista de mensajes asociados a ese chat. También existe la posibilidad de utilizar el buscador para filtrar mensajes mediante el texto introducido en el buscador.

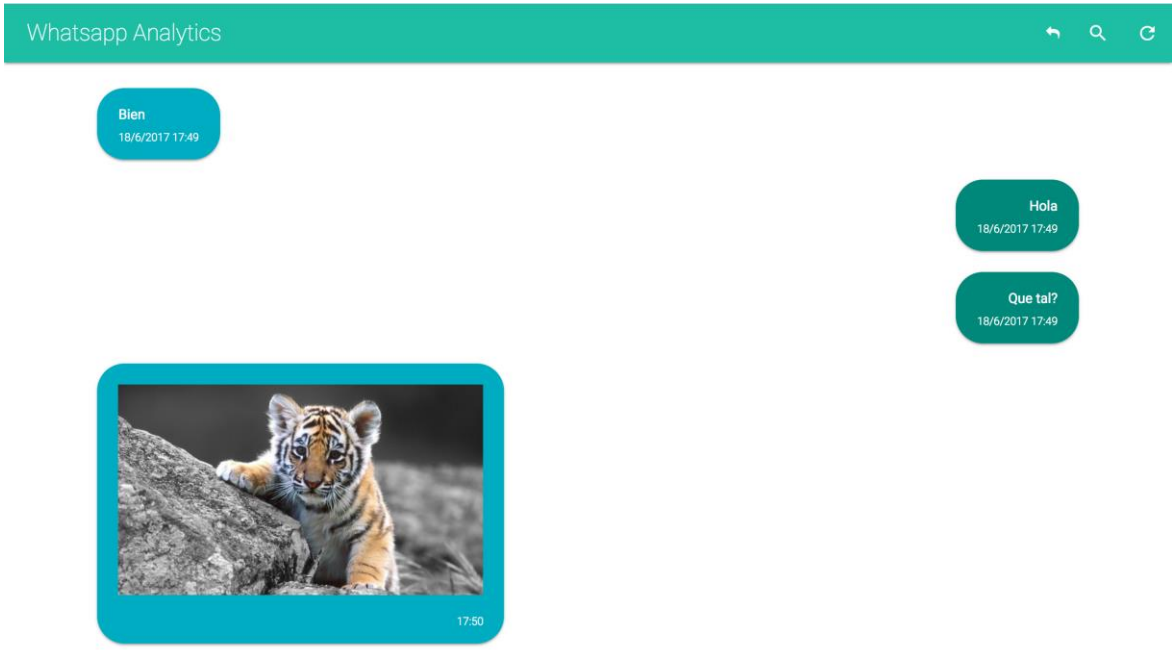

Figura 3-28 - Interfaz con los mensajes de un chat

Como se puede observar en algunas imágenes se cumplen perfectamente las especificaciones comunes para todas las interfaces web del proyecto.

Figura 3-29 - Interfaz con mensajes de un chat en dispositivo móvil

3.5 *Logs* de todos los procesos.

Tanto el software de autorización, como el del envío automático de iteraciones, como el de monitorización, presentan un log propio donde se recogen todos los eventos que se producen durante la ejecución del cada uno de sus procesos. De esta manera queda registrada toda la actividad realizada en el software, indicando en algunos casos el normal funcionamiento y en otros si se producen errores.

Este sería el log para los procesos de autorización:

Hvs_WhA.log

```
2017-05-31 21:42:22 Authorize: A target connected.
2017-05-31 21:43:38 Authorize: Target disconnected.
2017-06-01 10:07:32 Authorize: A target connected.
2017-06-01 10:08:05 Authorize (15398887724): User successfully authorized.
2017-06-01 10:13:52 Authorize: A target connected.
2017-06-01 10:17:17 Authorize: A target connected.
2017-06-01 10:17:34 Authorize (d): User successfully authorized.
2017-06-01 10:20:32 Authorize: Target disconnected.
2017-06-01 10:20:48 Authorize: A target connected.
2017-06-01 10:21:03 Authorize: SQL-> Cannot insert the value NULL into column 'Phone'
2017-06-01 10:22:04 Authorize: A target connected.
2017-06-01 10:22:06 Authorize: Target disconnected.
2017-06-01 10:22:06 Authorize: A target connected.
2017-06-01 10:22:11 Authorize: Target disconnected.
2017-06-01 10:22:18 Authorize: A target connected.
2017-06-01 10:22:35 Authorize (Desconocido): User successfully authorized.
2017-06-01 10:26:53 Authorize: A target connected.
2017-06-01 10:27:11 Authorize (Desconocido): User successfully authorized.
2017-06-02 17:28:21 Authorize: A target connected.
2017-06-02 17:29:06 Authorize (15398887724): User successfully authorized.
```

Figura 3-30 - Log de autorización

Este es el log para los procesos del envío automático de iteraciones:

Hvs_WhV.log

```
2017-06-11 18:25:56 Queue: SQL-> No actions to be processed.
2017-06-11 18:27:22 Server: Listening on port 8080.
2017-06-11 18:27:23 Queue: SQL-> No actions to be processed.
2017-06-11 18:27:38 Queue: Procesing new batch from [15398887724]
2017-06-11 18:27:54 Queue: SQL-> No actions to be processed.
2017-06-11 18:28:09 Queue: SQL-> No actions to be processed.
2017-06-11 18:29:20 Server: Listening on port 8080.
2017-06-11 18:29:21 Queue: SQL-> No actions to be processed.
2017-06-11 18:29:31 Server: Listening on port 8080.
2017-06-11 18:29:31 Queue: Procesing new batch from [15398887724]
2017-06-11 18:29:47 Queue: SQL-> No actions to be processed.
2017-06-11 18:30:02 Queue: SQL-> No actions to be processed.
2017-06-11 18:30:05 Queue: Done batch [15398887724]
2017-06-11 18:30:18 Queue: SQL-> No actions to be processed.
2017-06-11 18:30:34 Queue: Procesing new batch from [15398887724]
2017-06-11 18:30:49 Queue: Procesing new batch from [15398887724]
2017-06-11 18:31:05 Queue: SQL-> No actions to be processed.
2017-06-11 18:31:08 Queue: Done batch [15398887724]
2017-06-11 18:31:20 Queue: SQL-> No actions to be processed.
2017-06-11 18:31:23 Queue: Done batch [15398887724]
2017-06-11 18:33:14 Server: Listening on port 8080.
2017-06-11 18:33:15 Queue: SQL-> No actions to be processed.
```

Figura 3-31 - Log de envío automático de iteraciones

Por último, este es el de los procesos asociados a la monitorización de las cuentas:

HvS_WhY.log

```
1998  2017-06-28 10:53:07 [Server]: Starting server.
1999  2017-06-28 10:53:10 [Server]: Load session (15398887724) successful.
2000  2017-06-28 10:53:15 [Server]: Connected with DB.
2001  2017-06-28 10:54:32 [Server]: Starting server.
2002  2017-06-28 10:54:48 [Server]: Starting server.
2003  2017-06-28 10:54:52 [Server]: Load session (15398887724) successful.
2004  2017-06-28 10:54:59 [15398887724]: Monitoring chat to get messages --> 5218117788673
2005  2017-06-28 10:54:59 [Server]: Connected with DB.
2006  2017-06-28 10:55:00 [15398887724]: Messages successfully obtained --> 5218117788673
2007  2017-06-28 10:55:00 [15398887724]: Monitoring chat to get messages --> 34650640840
2008  2017-06-28 10:55:02 [15398887724]: Messages successfully obtained --> 34650640840
2009  2017-06-28 10:55:02 [15398887724]: Monitoring chat to get messages --> Grupo para pruebas
2010  2017-06-28 10:55:03 [15398887724]: Messages successfully obtained --> Grupo para pruebas
2011  2017-06-28 10:55:03 [15398887724]: Monitoring chat to get messages --> 34663535521
2012  2017-06-28 10:55:05 [15398887724]: Messages successfully obtained --> 34663535521
2013  2017-06-28 10:55:06 [15398887724]: Monitoring chat to get messages --> 34671844950
2014  2017-06-28 10:55:07 [15398887724]: Messages successfully obtained --> 34671844950
2015  2017-06-28 10:55:07 [15398887724]: UpdateMode activated
```

Figura 3-32 - Log de monitorización de cuentas

4. Descripción técnica

Una vez se han explicado los objetivos y se han mostrado los resultados obtenidos, llega el momento de entrar en detalle en el cómo se ha conseguido desarrollar el proyecto desde un punto de vista más técnico. Es decir, voy a explicar cómo he logrado crear APIs de WhatsApp y posteriormente usarlas para realizar una sincronización real entre *León Literario* y WhatsApp.

Como expliqué anteriormente el proyecto se divide en tres subproyectos independientes que utilizan tecnologías muy diferentes. Aun así, el *corazón* de todo el sistema en conjunto es Node.js, y el *cerebro* la base de datos, alojada en SQL Server.

4.1 Descripción detallada de cada uno de los subproyectos

4.1.1 Autorización de cuentas

Después de analizar muchas posibles soluciones que cubriesen esta necesidad, se optó por la que se consideraba más viable. Esta solución consistía en recoger las cookies de WhatsApp Web tras iniciar sesión, almacenarlas en una base de datos y posteriormente introducirlas en el navegador para poder acceder a la cuenta.

Este objetivo se logró de la siguiente manera:

Un servidor Node.js se encarga de servir una interfaz que será visible para el usuario que desee autorizar a la plataforma el uso de su cuenta.

Cuando un usuario acceda a la interfaz, un script notificará al servidor Node.js a través de Socket.IO de su acceso a la interfaz web.

En ese momento en el lado del servidor se ejecutará, a través de Selenium, un navegador Chrome que estará completamente controlado por Node.js. Este navegador accederá a WhatsApp Web (web.whatsapp.com). Como es normal WhatsApp Web mostrará un código QR para iniciar sesión.

Node.js obtendrá ese código QR y lo enviará a través de Socket.IO a la interfaz web que se muestra al usuario.

Cuando el usuario escanee el código QR de la interfaz, en el navegador Chrome del lado del servidor, que está siendo controlado por Node.js, se iniciará sesión con la cuenta del usuario.

Una vez iniciada la sesión, Node.js obtendrá las cookies de WhatsApp Web del navegador. También obtendrá la foto de perfil del usuario, si este la tuviese, y su número de teléfono que, aunque WhatsApp no lo muestra en ningún momento está contenido en el link de la foto de perfil.

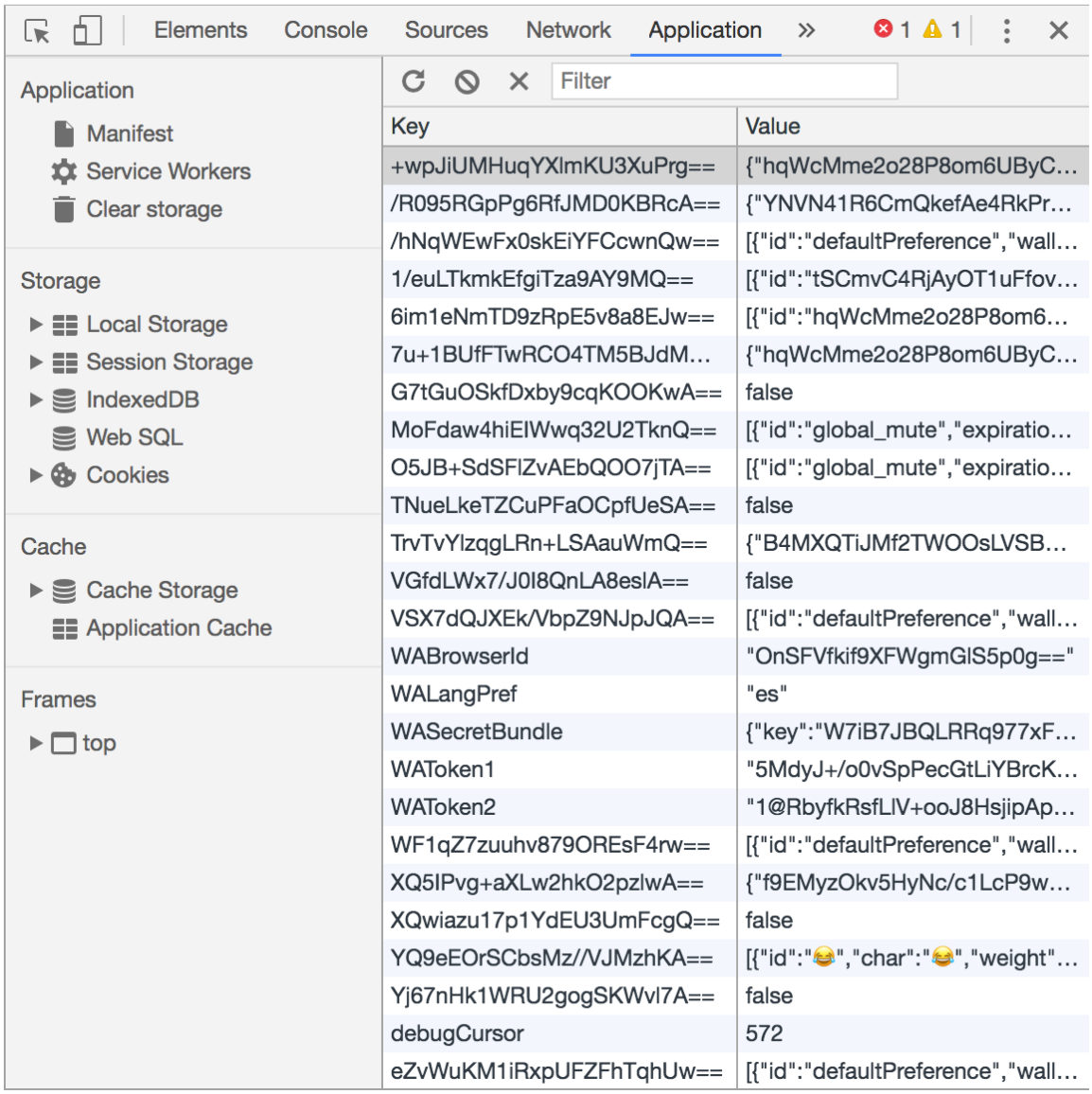

Figura 4-1 - Cookies de WhatsApp Web

Una vez se haya terminado de obtener estos datos se cerrará el navegador Chrome de forma automática desde Node.js. Node realizará una llamada a un procedimiento de la base de datos para almacenar todos los datos correspondientes a la sesión del usuario.

Por otro lado, se enviarán la foto de perfil y el número de teléfono a la interfaz web visible para el usuario para informarle de que la autorización se completó con éxito. Después de ese momento el usuario podrá cerrar la web, para que el servidor de por concluido el proceso de autorización.

En el desarrollo de este sistema surgieron dos problemas:

El primero, que se solucionó completamente, consistía en que si el usuario tardaba en escanear el código QR, este cambiaría invalidando el anterior. En el peor de los casos WhatsApp Web dejaría de mostrar códigos QR, solicitando un click para volver a mostrar un nuevo código QR. Se solucionó el problema enviando el nuevo código QR a interfaz visible cada vez que este cambiase y simulando el click del usuario cuando WhatsApp dejase de mostrar códigos QR.

El segundo problema solo se pudo solucionar parcialmente. Este problema consiste en que, si el usuario que se autoriza no tiene foto de perfil, no se podrá obtener tampoco su número de teléfono, debido a que este solamente se encuentra contenido en el link de la foto de perfil si la hubiese. La solución parcial consiste en tomar como identificador para el número de la cuenta la palabra "Desconocido" junto a un número que se iría incrementando sucesivamente. Ejemplo: Desconocido1, Desconocido2.

4.1.2 Envío automático de iteraciones

Debido a que las tecnologías utilizadas en el sistema de autorización dieron muy buen resultado se optó por usar algunas de estas tecnologías. En primer lugar, se debía de realizar el paso contrario a la autorización, es decir, seleccionar las cookies de las cuentas de WhatsApp almacenadas en la base de datos para inyectarlas en un navegador y acceder de esta forma a las sesiones de las cuentas de WhatsApp Web correspondientes para posteriormente ejecutar las iteraciones almacenadas.

Este objetivo se logró de la siguiente manera:

En primer lugar, se creó una herramienta en lenguaje java que consistía en un generador de scripts sobre una base de datos SQL. El cliente debía de introducir el número de teléfono asociado a una cuenta de WhatsApp autorizada desde la que se enviaría el mensaje, el número de teléfono del destinatario, la fecha en la que debía ser entregado el mensaje y el propio mensaje. Se generaría un script SQL que el usuario podría ver en todo momento y una vez ejecutado se almacenaría la iteración en la base de datos.

Un servidor Node.js consulta cada cierto tiempo la base de datos para buscar si existen nuevas iteraciones. Esta tarea se realiza durante toda la ejecución del sistema, independientemente de si hay o no nuevas iteraciones.

En caso de que no haya nuevas iteraciones se indicará con un mensaje en la terminal que no las hay.

Si por el contrario las hay, se informará del mismo modo, que existe un nuevo *batch* de iteraciones y que se procederá a su ejecución.

La ejecución consiste en:

- Primero, se comprueba que la cuenta del remitente esté autorizada en la plataforma.
- Después, se toman todas las iteraciones del *batch*.
- Se genera el script encargado de inyectar las cookies para esa cuenta de WhatsApp en WhatsApp Web.
- A través de Selenium se abre un navegador Chrome que está totalmente controlado por Node.js.
- Se accede a WhatsApp Web (web.whatsapp.com) y se ejecuta sobre la web el script que inyecta las cookies. De esta forma se iniciará la sesión de WhatsApp correspondiente al remitente de todas las iteraciones del *batch*.
- Se genera un script que utiliza *Web scraping* para poder ejecutar las iteraciones sobre la cuenta de Whatsapp iniciada en el navegador Chrome. El script funciona de la siguiente manera:
    - Se comprueba que la web de WhatsApp Web se ha cargado en el navegador de forma completa.
    - Se simula un click sobre el buscador de chats y se introduce el nombre del destinatario, una vez se ha cargado en la web la lista donde se muestra al destinatario, se hace un click sobre él.
    - Una vez se muestre la ventana del chat del destinatario se procederá a simular un simular un click sobre el *imput* donde se escribe el mensaje, para introducir a continuación el texto del mensaje. Posteriormente se simulará otro click en el botón que permite enviar el mensaje.
    - Utilizando DOM se introducirá un *div* con el id “finish” en la raíz del documento HTML.

```
function triggerMouseEvent (node, eventType) {

    var clickEvent = document.createEvent ('MouseEvents');
    clickEvent.initEvent (eventType, true, true);
    node.dispatchEvent (clickEvent);

}

function click(element){

    triggerMouseEvent (element, 'mouseover');
    triggerMouseEvent (element, 'mousedown');
    triggerMouseEvent (element, 'mouseup');
    triggerMouseEvent (element, 'click');

}

function inputEvent(input, value){

    input.value = value;
    var evt = document.createEvent('TextEvent');
    evt.initTextEvent ('input', true, true, window, input.value, 0, 'en-US');
    input.innerText = value;
    input.focus();
    input.dispatchEvent(evt);

}

function start(){

    setTimeout(function(){

        if (document.getElementsByClassName('icon icon-search').length > 0 && document.getElementsByClassName('pane-header pane-list-
            header')[0].getElementsByClassName('avatar-image is-loaded').length > 0 && document.getElementsByClassName('menu menu-
            horizontal').length > 0) {

            click(document.getElementsByClassName('icon icon-search')[0]);
            inputEvent(document.getElementsByClassName('input input-search')[0], tlf);

            openChat();

        }else{

            start();

        }

    }, 100);

}
```

Figura 4-2 - Fragmento del script para el envío de iteraciones

- Una función en Node.js cada cierto tiempo comprueba si ha finalizado el script con las iteraciones. Para comprobar que el script ha finalizado comprueba que en el código fuente HTML de la página de WhatsApp Web cargada en el navegador existe un *div* con id “finish”.
- Si pasado un tiempo no se encuentra este *div*, se tomará como una ejecución fallida el conjunto de iteraciones de ese *batch*.
- Después se realizará desde Node.js una llamada a un procedimiento para actualizar el estado de las iteraciones almacenadas, indicando si se ejecutaron correctamente o no.
- Para finalizar se imprimirá un mensaje en la terminal del servidor Node.js indicando que el *batch* ha terminado de ejecutarse.

4.1.3 Monitorización de datos

Este sistema fue sin duda el más difícil de desarrollar. Se utilizaron tecnologías muy diferentes a las explicadas en los puntos anteriores.

En primer lugar, surgió un nuevo problema, la dificultad de poder manejar la cantidad de datos que contiene WhatsApp Web. Primero, hubo que realizar un estudio interno de cómo esta implementado WhatsApp Web y que datos nos puede ofrecer y cuales verdaderamente nos interesan. Concluido este estudio, se analizaron las tecnologías que podían ser utilizadas para este fin. Se llegó a la conclusión de que utilizar un navegador independiente controlado por Node.js a través de Selenium no era viable ni eficiente por cuestiones que explicaré más adelante.

Se encontró por fin una herramienta que cumplía con las necesidades del sistema y que sería la solución que nos permitiría realizar la captura de datos de una cuenta de WhatsApp a través de WhatsApp Web. Esta herramienta era NW.js, un software que nos permitiría crear aplicaciones de escritorio que funcionarán dentro de un servidor Node.js. Lo cual nos brinda la posibilidad de cargar páginas web completas y tener acceso a toda su estructura DOM desde Node.js.

En primer lugar, la aplicación muestra una interfaz para la selección de usuarios. Desde Node.js realizamos una consulta a la base de datos para obtener todas las cuentas de WhatsApp autorizadas en la plataforma. Las cuentas serán mostradas en una lista indicando el número de teléfono y la foto de perfil, si la hubiese, de cada cuenta. Una vez seleccionada se podrá empezar a realizar la monitorización pulsando el botón iniciar.

Cuando se pulsa ese botón se crea inmediatamente un script para la inyección de las cookies de la cuenta que se ha seleccionado y se inyecta sobre el navegador de NW.js. Posteriormente en este navegador se carga la web de WhatsApp Web (web.whatsapp.com).

Cuando se inicie WhatsApp Web se mostrará directamente la sesión de WhatsApp correspondiente con las credenciales que se almacenaron en la base de datos en el proceso de autorización. Es decir, se accederá a la cuenta de WhatsApp asociada a la persona que autorizó a la plataforma a usar su cuenta, en lugar de mostrar un código QR.

Ahora voy a explicar con cierta profundidad el proceso de captura de datos de la cuenta de WhatsApp que se muestra en la aplicación NW.js.

En primer lugar, se comprueba si la web de WhatsApp Web se ha cargado completamente en NW.js, si no es así se esperará un breve periodo de tiempo y se volverá a comprobar de nuevo si ha cargado. Este proceso se repetirá hasta que la web este cargada.

```
function inicialize(){

  setTimeout(function(){

    if (root.getElementsByClassName("icon icon-search").length > 0 && root.getElementsByClassName("pane-header
        pane-list-header")[0].getElementsByClassName("avatar-image is-loaded").length > 0 && root.
        getElementsByClassName("menu menu-horizontal").length > 0) {

      connector.connectDB();

      var profilePicture = root.getElementsByClassName("pane-header pane-list-header")
          [0].getElementsByClassName("avatar-image is-loaded")[0];
      userID = functions.getPhoneNumberFromPicure(profilePicture);

      createExitButton();

      if (updateMode) {

        checkUpdates();

      }else {

        startMonitoring();

      }

    }else {

      inicialize();

    }

  }, 100);

}
```

Figura 4-4 - Fragmento de la función para iniciar la monitorización

Después, utilizando DOM se procederá a crear un botón que será insertado junto a los botones "Nuevo chat" y "Menú", cuya función es redirigir al usuario a la interfaz anterior, en la se eligen las cuentas para monitorizar.

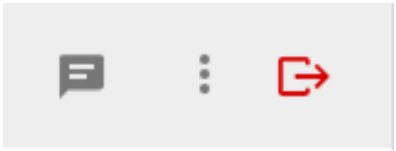

Figura 4-3 - Botones de WhatsApp Web

Luego se comprobará si la monitorización es de todos los chats, o solamente de uno en particular. Aquí existen dos vías:

- Si se desea monitorizar un usuario en particular, en primer lugar, se simulará un click en el buscador de chats y se introducirá su nombre o su número de teléfono. Después se comprobará si se muestra la lista de chat con el único chat que se quiere monitorizar. Si aún no se muestra se esperará un tiempo y se volverá a comprobar. Una vez que se muestre se añadirá el *div* que contiene el chat a una lista de chats de Node.js.
- Por otro lado, si no se indica ningún usuario en particular se monitorizará la totalidad de los chats que se muestran. Debido a que WhatsApp Web no almacena en el DOM toda la lista de chats, se simulará un evento *scroll* sobre la lista para que se carguen en el DOM de WhatsApp Web todos los chats. Una vez se tienen todos los chats en el DOM se añaden a la lista de chats de Node.js

Después se cogerá el ultimo chat de la lista de chats que hemos creado y se simulará un click sobre él, de esta forma se abrirá el chat y se mostrará en la interfaz. Inmediatamente después se borrará el último chat de la lista, ya que no lo necesitamos porque ya lo tenemos abierto.

Mediante DOM seleccionamos el link de la foto de perfil del chat que acabamos de abrir y mediante una librería de Node.js extraemos, a partir del link, la foto de perfil en base64 y la guardamos en una variable dentro de Node.

Posteriormente simulamos un click sobre la barra que se muestra en figura 4-5.

Figura 4-5 - Header de usuario en WhatsApp Web

Hemos conseguimos que se muestre el número de teléfono de la persona o el nombre del grupo.

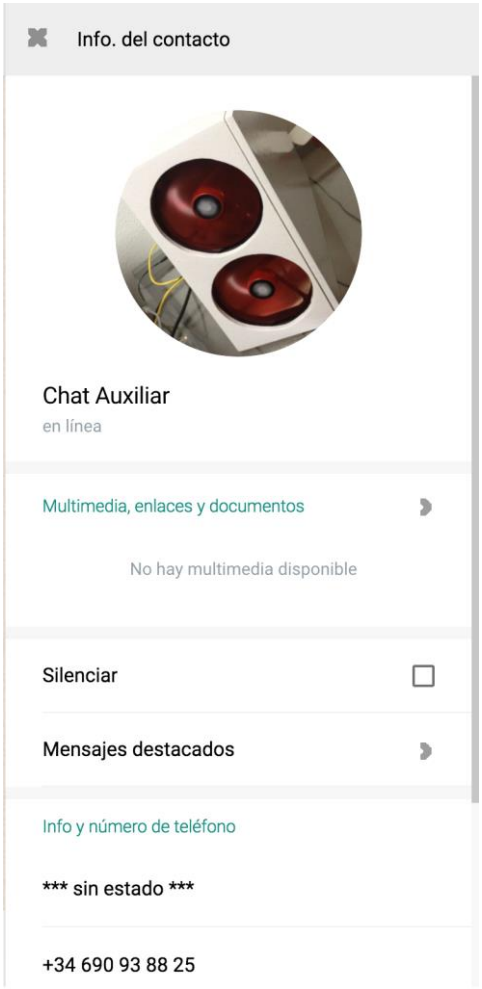

Figura 4-6 - Información de usuario en WhatsApp Web

Tomamos este dato como id del chat y lo guardamos en una variable de Node. Ahora simulamos un click sobre el icono de cerrar la información del chat para regresar a la interfaz donde se muestra el chat con las iteraciones.

Llegó el momento de seleccionar cada una de las iteraciones que se muestran en el chat. En primer lugar, se simula un evento *scroll* dentro del chat para que se carguen en el DOM todos los mensajes desde que se inició el chat. Una vez estén contenidos todos los mensajes en el DOM de WhatsApp Web se procederá a seleccionarlos todos y almacenarlos en una lista de Node.

Después se recorrerá la lista y se analizará cada uno de los mensajes extrayendo la mayor cantidad de información posible de cada uno. Se contemplará el tipo de iteración, es decir si es un mensaje de texto, una foto, un vídeo, un contacto, una ubicación, un audio, o un documento. También se seleccionará la fecha, la hora, el texto, los *emojis*, los links, las imágenes, una pre visualización de los vídeos, una pre visualización de los documentos, las coordenadas de una ubicación o el número de teléfono asociado de un contacto.

Toda esta información será guardada en forma de JSON, junto con la foto de perfil y el id del chat que guardamos anteriormente, y será almacenada en la base de datos mediante la llamada a un procedimiento de SQL.

Cuando se termine de procesar todos los mensajes, desde Node se volverá a llamar a la función de seleccionar chats. Se seleccionará el último chat de la lista, simulando un click sobre este, y se repetirá todo el proceso que he explicado hasta ahora con cada chat de la lista.

Cuando se termine la captura de datos, es decir cuando se haya terminado de monitorizar todos los chats de la lista se abrirá un chat auxiliar, y se activará el "*modo update*".

El *modo update* consiste en una función que cada cierto tiempo comprueba si hay mensajes nuevos, si no los hay se auto llama la función. De esta manera se simula un hilo que está a la espera de nuevas iteraciones.

Cuando un chat recibe nuevas iteraciones, automáticamente se seleccionará dicho chat en el DOM y se simulará un click sobre él. De esta forma se mostrará el chat en la interfaz. Solo se seleccionarán los mensajes nuevos. Para poder seleccionar estos mensajes simplemente debemos de buscar en el DOM el elemento *div* que contiene el mensaje "X MENSAJES SIN LEER" como se muestra en la figura 4-7:

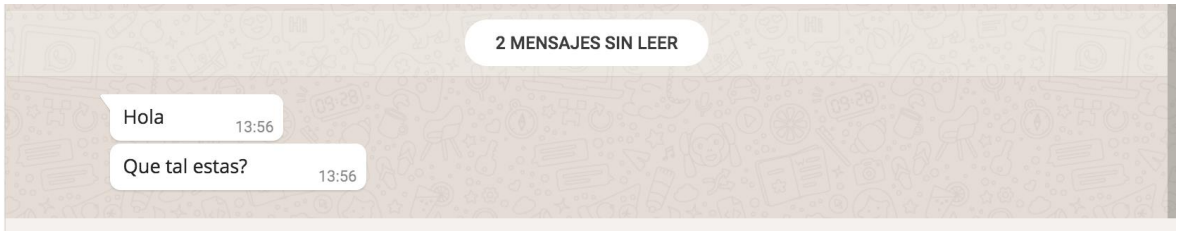

Figura 4-7 - Mensajes sin leer

Seleccionamos todos los mensajes que están colocados por debajo de este mensaje. Los procesamos uno a uno como se explicó con anterioridad. Una vez procesados se abrirá el chat auxiliar y se seguirá buscando nuevas iteraciones.

El chat auxiliar se corresponde con un chat entre el usuario y un destinatario que utiliza un número de *HumedoValley*. En este chat nunca existirán mensajes.

La función de este chat es solucionar un problema. Este problema consiste en que, si el último chat que ha sido monitorizado recibe mensajes nuevos, el "*modo update*" no los detectará. Esto se debe a que, si se reciben mensajes en un chat que está abierto en la interfaz, WhatsApp no los considera mensajes nuevos. De esta forma si cuando no se está monitorizando se muestra un chat que nunca va a recibir mensajes, se soluciona este problema.

4.1.4 Interfaz web para mostrar los resultados

Como se explicó con anterioridad, la idea de crear una interfaz web para mostrar los datos de la monitorización de cuentas surgió como solución para resolver el problema de manejar esta gran cantidad de datos.

Para no llevar a una equivocación se debe resaltar que la interfaz web solo mostrará los datos almacenados en la base de datos, nunca los datos de WhatsApp en tiempo real.

Una vez concretado esto, me dispongo a explicar desde un punto de vista técnico el desarrollo de esta interfaz.

Disponemos de un servidor Node.js que sirve todo el conjunto que compone la interfaz web. La interfaz web realiza peticiones por medio de AJAX a diversos *endpoints* de una API REST de Node.js. Node debe realizar consultas sobre la base de datos para obtener los datos necesarios y devolver estos como respuesta a las peticiones que realiza la interfaz.

Cuando se inicia la interfaz, esta realizara una petición GET por medio de AJAX a un *endpoint* de la API REST de Node para recibir la lista de cuentas de WhatsApp autorizadas en la plataforma. Node realizará la llamada a un procedimiento de SQL para obtener esta lista y enviarla como respuesta a la petición.

Una vez en el lado del cliente se haya recibo la lista de cuentas, se pintará sobre la interfaz web indicando el número de teléfono asociado a la cuenta, la foto de perfil y la fecha en la que se realizó la autorización en la plataforma.

También contamos con la existencia de un buscador, cuya función es mostrar únicamente las cuentas que contienen el texto que buscamos. Para esto, recorremos, usando el DOM, la lista de cuentas y eliminamos de la interfaz las cuentas que no contienen el parámetro de texto introducido.

Una vez se seleccione la cuenta cuyos resultados se desean ver se cargará una nueva página web, que contendrá en la URL el id de la cuenta (número de teléfono) que se desea monitorizar. Se obtendrá ese id, y se realizará una petición GET a través de AJAX, en la que se enviará como parámetro el id de la cuenta para poder recibir todos los chats asociados a esta. Node realizará una llamada al procedimiento de SQL para obtener la lista de chats asociados a esa cuenta y los devolverá como respuesta a la petición.

Cuando se reciban todos los chats de la cuenta en forma de lista, esta se recorrerá y se pintarán uno por uno todos los chats. Cada chat consta del nombre de destinatario y la foto de perfil de este, en caso de no existir se pondrá una foto por defecto.

También contamos con un buscador, que nos permite mostrar únicamente las cuentas que contengan en el nombre el parámetro de texto que hayamos escrito. El funcionamiento es el mismo que el del otro buscador. Este buscador no distingue entre mayúsculas y minúsculas.

Cuando se seleccione un chat, se abrirá una nueva página que incluirá en su URL el id de la cuenta, y el id del destinatario que deberá ser un número de teléfono de un persona o el nombre de un grupo. Se realizará una petición GET por medio de AJAX a un *endpoint* de Node donde se enviará el id de la cuenta y el del destinatario. Node responderá a la petición enviando una lista con todos los mensajes entre el remitente de la cuenta y el destinatario.

Una vez que se reciban los mensajes, se procederá a pintarlos sobre la interfaz, distinguiendo claramente quien es el remitente y el destinatario como se muestra en la figura 4-8.

Figura 4-8 - Mensajes de chat monitorizado

Al igual que en el resto de las páginas web que componen esta interfaz se ha implementado un buscador para poder mostrar únicamente los mensajes que contengan el texto introducido en él.

4.1.5 *Logs* del sistema

Todo el sistema en conjunto, tanto el software de autorización, como el del envío de mensajes y monitorización de cuentas, tiene integrado un historial donde se recoge cada una de las iteraciones del usuario con el sistema. En este historial podemos ver todo el conjunto de acciones que han sido realizadas por un usuario en el sistema, indicando aquellas en las cuales se ha producido algún tipo de error.

Todos los *logs* se realizan en el servidor Node.js y consisten en guardar en un archivo de texto las acciones que se van realizando sucesivamente, indicando la fecha de realización de cada una. A la vez que estas acciones se guardan en el archivo de texto también se muestran simultáneamente en la consola de Node.

4.2 Web scraping

El *web scraping* es una técnica utilizada para extraer información de páginas web de forma automatizada.

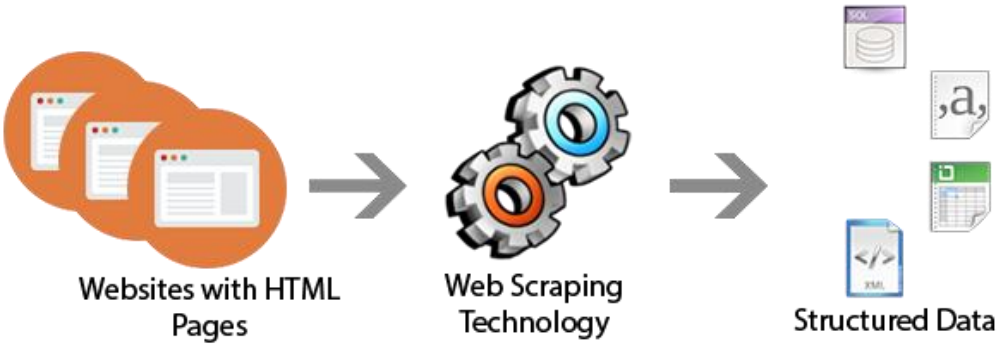

Figura 4-9 - Explicación de Web Scraping

Para realizar esta técnica un sistema automatizado se hace pasar por un usuario que navega por la web, en la que se quieren extraer los datos. Este sistema automatizado irá recogiendo los diferentes elementos que se desee de la web para luego almacenarlos.

En el sistema de autorización utilizamos esta técnica para extraer el código QR de WhatsApp Web y posteriormente las cookies y la información del usuario.

En cuanto al software de envío automático de iteraciones, utilizamos esta técnica para comprobar si se ha iniciado correctamente la sesión de WhatsApp Web y para comprobar que las iteraciones se han enviado correctamente.

En el sistema de monitorización de usuarios se utiliza esta técnica constantemente para la mayoría de las tareas de captura de datos.

Antes de utilizar *web scraping* se ha realizado un análisis exhaustivo de cómo están dispuestos los datos de la web en la que se desea aplicar esta técnica. Por lo tanto, se debe conocer perfectamente la estructura interna del DOM de la web.

```
function getTextMessage(message){

  var data = message.getElementsByClassName("bubble-text")[0];
  var json = {};

  json.receiverID = idChat;
  json.receiverName = contactName;
  json.userID = userID;
  json.profilePicture = profilePicture;
  json.type = "text";
  json.time = data.getElementsByClassName("message-datetime")[0].innerText;
  json.id = data.getElementsByClassName("message-text")[0].getAttribute("data-id");
  json.message = data.getElementsByClassName("selectable-text")[0].innerText;

  if (message.getElementsByClassName("message-out").length > 0) json.status = "sended";
  if (message.getElementsByClassName("message-in").length > 0) json.status = "received";

  var datetime = data.getElementsByClassName("message-pre-text")[0].innerText.split("]")[0].trim();
  json.date = datetime.substring(10, datetime.length);

  var emojis = data.getElementsByClassName("emoji");
  json.emojis = [];

  if (emojis.length > 0) {

    for (var i = 0; i < emojis.length; i++) {
      json.emojis.push(emojis[i].getAttribute("alt"));
    }

  }

  var bigEmojis = data.getElementsByClassName("large-emoji selectable-text");

  if (bigEmojis.length > 0) {

    for (var i = 0; i < bigEmojis.length; i++) {
      json.emojis.push(bigEmojis[i].getAttribute("alt"));
    }

  }

  var links = data.querySelectorAll("a.selectable-text");
  if (links.length > 0) {

    json.links = [];

    for (var i = 0; i < links.length; i++) {
      json.links.push(links[i].innerText);
    }

  }

  connector.insertDB(json);
  messageList.push(json);

}
```

Figura 4-10 - Uso de Web Scraping

4.3 Comparativa entre Selenium y NW.js

Selenium es un entorno de pruebas de software basadas en la web que nos permite crear, editar y depurar casos de prueba, que pueden ser ejecutados de forma automática. Estas acciones se realizan por medio de APIs en varios lenguajes. En nuestro caso utilizaremos estas APIs desde Node.js.

En la arquitectura del software de autorización de cuentas de usuarios y en la del envío automático de iteraciones se utiliza esta tecnología. Selenium nos permite ejecutar un navegador Chrome que podemos manejar desde Node. De esta forma podemos extraer e inyectar cookies, así como insertar las iteraciones de usuario de forma automática.

NW.js es una herramienta que permite desarrollar aplicaciones de escritorio, se basa en Chrome y Node.js, de tal forma que el navegador Chrome se ejecuta dentro de un servidor Node.js, esto es posible porque ambos utilizan el mismo motor de JavaScript. Esto nos brinda la posibilidad de acceder a todo el DOM del navegador Chrome desde Node. (1)

La arquitectura del software de monitorización de cuentas se basa principalmente en esta herramienta porque permite el acceso completo desde Node al DOM de WhatsApp Web.

Ahora voy a explicar porque se ha utilizado NW.js en lugar de Selenium.

En primer lugar, Selenium nos permite, de forma simultánea, controlar varios navegadores Chrome desde un mismo servidor Node.js, esto se traduce en un mayor rendimiento y optimización respecto a NW.js, que únicamente nos permite monitorizar una cuenta a la vez. La desventaja consiste en que el script que se inyecta en el navegador Chrome, se ejecuta fuera de Node.js por lo tanto no tiene acceso a la base de datos ni a las funciones de Node.js.

Se intentó implementar tanto el uso de AJAX como el de Socket.io para comunicar el script con Node.js pero WhatsApp Web bloquea las conexiones salientes. Para comprobar si se ejecutó correctamente el script debemos crear un elemento en el DOM de la web que indique que se ejecutó con éxito, como se muestra en la figura 4-11, para posteriormente comprobar desde Node si existe ese elemento y actualizar el estado de la iteración en la base de datos.

Figura 4-11 - Señal de envío correcto de un mensaje

En cambio NW.js nos permite, en todo momento, realizar consultas sobre la base de datos desde el mismo script en el que se accede al DOM. Por lo tanto, no necesitamos enviar los datos a un servidor con acceso a la base de datos, ni usar el DOM para comunicárselo a Node por medio de “señales”.

En el software de monitorización de cuentas se accede continuamente al DOM de WhatsApp Web para extraer datos, por lo que se necesita ejecutar multitud de consultas sobre la base de datos para almacenarlos. El uso de Selenium para este fin no era viable, pero NW.js es la herramienta perfecta.

4.4 Socket.IO

Socket.IO es una librería que se puede instalar en Node.js y que nos permite manejar eventos en tiempo real mediante una conexión TCP. Es rápido y potente y permite hacer transmisiones de datos de forma segura en tiempo real. (6)

En el sistema de autorización se utiliza esta librería para realizar todas las transmisiones entre Node.js y la interfaz web donde el usuario nos autoriza a usar su cuenta.

Para implementar Socket.IO únicamente debemos crear una instancia de este objeto en el servidor Node.js e inicializarla sobre un servidor *http* como se muestra en la figura 4-12:

```
var app = require('express')();
var http = require('http').createServer(app);
var io = require('socket.io')(http);
```

Figura 4-12 - Implementación de Socket.IO en Node

Después únicamente bastaría con usar esta conexión en el cliente de la siguiente forma:

```
var socket = io.connect();
```

Figura 4-13 - Implementación de Socket.IO en el cliente

Principalmente se utiliza Socket.IO para enviar y recibir datos. En la siguiente figura se puede observar cómo se implementa, de forma sencilla, el envío y la recepción de datos utilizando Socket.IO.

```
socket.emit('message', mensaje);
//Enviamos un mensaje

socket.on('message', function(data) {
    //Recibimos un mensaje.
});
```

Figura 4-14 - Envío y recepción de datos con Socket.IO

4.5 API REST

La interfaz web utilizada para mostrar las iteraciones de los usuarios autorizados utiliza una API REST para comunicarse con el servidor Node.js. Se decidió que el uso de una API REST daría la flexibilidad necesaria para implementar de manera rápida y eficaz las comunicaciones. A demás, puede facilitar la portabilidad y el mantenimiento de cara a futuros cambios.

El uso que se hace de la API REST consiste en la utilización de AJAX en el lado del cliente para enviar peticiones, con o sin parámetros, a *endpoints* de Node.js y capturar la respuesta del servidor.

A continuación, se describirá uno a uno los *endpoints* que se han desarrollado.

Tabla 4-1 - *Endpoint* Deauthorize

Deauthorize		
Tipo: POST		
Endpoint: /deauthorize		
Parámetros: Phone – El número de teléfono asociado a la cuenta de WhatsApp en formato JSON. Este parámetro solo es numérico por lo que cualquier otro tipo de caracteres producirán respuesta de tipo 400.		
Descripción: Las llamadas a este *endpoint* se realizarán para retirar la autorización hacia esta cuenta a por parte de la plataforma.		
Response:	Códigos:	200 o 400
	Mensajes:	Deauthorization completed. 200
		Missing or invalid phone parameter. 400
		SQL-> <error> 400
Detalles técnicos: En este *endpoint* se recibe como parámetro el número de teléfono asociado a la cuenta y se realiza una llamada al procedimiento *HvS_WhA_DeAuthorize* que cambiará el campo *Wh_authorized* a -1.		

Tabla 4-2 - *Endpoint* List_elements

List_elements		
Tipo: GET		
Endpoint: /list_elements		
Parámetros: Ninguno.		
Descripción: Las llamadas a este *endpoint* se realizarán para obtener la lista de cuentas de WhatsApp autorizadas.		
Response:	Códigos:	200 o 400
	Mensajes:	<JSON userList>. 200
		SQL-> <error> 400
Detalles técnicos: En este *endpoint* se realiza una llamada al procedimiento *HvS_WhY_AnalyticITEMS__list* para obtener la lista completa de las cuentas de WhatsApp asociadas a la plataforma.		

Tabla 4-3 - *Endpoint* Delete_element

Delete_element		
Tipo: POST		
Endpoint: /delete_element		
Parámetros: ID – El número de teléfono asociado a la cuenta de WhatsApp en formato JSON. Este parámetro solo es numérico por lo que cualquier otro tipo de caracteres producirán respuesta de tipo 400.		
Descripción: Las llamadas a este *endpoint* se realizarán para retirar la autorización del uso de esta cuenta por parte de la plataforma.		
Response:	Códigos:	200 o 400
	Mensajes:	OK. 200
		SQL-> <error> 400
Detalles técnicos: En este *endpoint* se recibe como parámetro el número de teléfono asociado a la cuenta y se realiza una llamada al procedimiento *HvS_WhA_DeAuthorize* que cambiará el campo *Wh_authorized* a -1.		

Tabla 4-4 - *Endpoint* Get_chats

Get_chats
Tipo: GET **Endpoint**: /get_chats **Parámetros**: ID – El número de teléfono asociado a la cuenta de WhatsApp en formato JSON. Este parámetro solo es numérico por lo que cualquier otro tipo de caracteres producirán respuesta de tipo 400. **Descripción**: Las llamadas a este *endpoint* se realizarán obtener la lista de chat asociados a esta cuenta. **Response**: Códigos: 200 o 400 Mensajes: <JSON chatList>. 200 SQL-> <error> 400 **Detalles técnicos**: En este *endpoint* se recibe como parámetro el número de teléfono asociado a la cuenta y se realiza una llamada al procedimiento *HvS_WhY_GetChats* para obtener todos los chats asociados a la cuenta.

Tabla 4-5 - *Endpoint* Get_messages

Get_messages
Tipo: GET **Endpoint**: /get_messages **Parámetros**: userID – El número de teléfono asociado a la cuenta de WhatsApp en formato JSON. Este parámetro solo es numérico por lo que cualquier otro tipo de caracteres producirán respuesta de tipo 400. receiverID – El número de teléfono o nombre del grupo que se considera destinatario en el chat. **Descripción**: Las llamadas a este *endpoint* se realizarán obtener la lista de mensajes asociados a un chat. **Response**: Códigos: 200 o 400 Mensajes: <JSON messageList>. 200 SQL-> <error> 400 **Detalles técnicos**: En este *endpoint* se recibe el número de teléfono del remitente asociado a la cuenta y el id del destinatario con el que se comunica y se realiza una llamada al procedimiento *HvS_WhY_GetMessages* para obtener todos los mensajes entre ambos.

4.6 Modelo de datos

4.6.1 Autorización de cuentas

El modelo de datos del software de autorización está diseñado para guardar las credenciales de las cuentas de WhatsApp de los usuarios asociados a *León Literario*.

4.6.1.1 Tablas

Se utiliza una única tabla para almacenar toda la información de los usuarios. Esta tabla está compuesta por un identificador único llamado "ID" que hace referencia a la clave primaria de la tabla. El campo "Phone" se utiliza para almacenar el número de teléfono de la cuenta de WhatsApp. En el campo "Photo" se almacena la foto de perfil del usuario. El campo "secrets" contiene las cookies necesarias para iniciar sesión en esa cuenta. El campo Wh_authorized solo podrá contener los valores 1 y -1 para indicar si el usuario mantiene la autorización de su cuenta.

ID	Phone	Photo	_active	_timestamp	secrets	Wh_authorized	_logCode
7	15398887724	...	-33	2017-05-30 12:19:35:190	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been AUTHORIZED
16	15398887724	NULL	-26	2017-05-30 20:12:38:350	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been RE-AUTHORIZED
24	Desconocido	NULL	0	2017-06-01 10:27:13:223	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been RE-AUTHORIZED
29	15398887724	...	-19	2017-06-02 17:42:57:697	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been RE-AUTHORIZED
2	15398881756	...	0	2017-05-08 18:51:24:103	{"s":{"O5JB+SdSFIZvAEbQOO7jTA==":"[{\"id\...	1	Whatsapp phone has been AUTHORIZED
17	15398887724	NULL	-25	2017-05-30 20:49:43:837	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been RE-AUTHORIZED
18	15398887724	NULL	-24	2017-05-30 20:54:03:597	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been RE-AUTHORIZED
20	15398887724	NULL	-22	2017-06-01 10:08:07:890	{"s":{"1/euLTkmkEfgiTza9AY9MQ==":"[{\"id\":...	1	Whatsapp phone has been RE-AUTHORIZED

Figura 4-15 - Tabla autorización

Para el *logueo* contamos con los siguientes 3 campos:

- _active: Indica el orden de las acciones sobre la cuenta, siendo 1 la actual. Cada vez que se produzca una acción en la cuenta de esta tabla se restara 1 a todos los campos active que pertenezcan a dicha cuenta.
- _timestamp: Indica la fecha de la acción realizada.
- logCode: Aporta una descripción a la acción que se realizó.

4.6.1.2 Procedimientos almacenados

Los procedimientos con los que cuenta este modelo son tres: autorizar cuentas, desautorizarlas y devolver todas las cuentas de la tabla incluyendo todos sus campos.

Tabla 4-6 - Procedimiento almacenado Authorize

HvS_WhA_Authorize
Parámetros de entrada: @Phone, @Photo y @secrets. Solo @Photo puede ser nulo. **Parámetros de salida**: @outputCODE. **Descripción**: Este procedimiento tras recibir los parámetros de entrada, almacena o modifica un registro en la tabla que contiene las credenciales de la cuenta. **Response**: Códigos: 200 o 400. **Detalles técnicos**: En este procedimiento, tras recibir los parámetros, se comprueba que sean válidos y dependiendo si existe o no un registro anterior para esta cuenta, crea uno nuevo o modifica el anterior restando 1 al número almacenado en el campo "_active". En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

Tabla 4-7 - Procedimiento almacenado DeAuthorize

HvS_WhA_DeAuthorize
Parámetros de entrada: @Phone. No puede ser nulo. **Parámetros de salida**: @outputCODE. **Descripción**: Este procedimiento tras recibir los parámetros de entrada, modifica el campo "Wh_Authorized" de la cuenta a -1 para desautorizarla. **Response**: Códigos: 200 o 400. **Detalles técnicos**: En este procedimiento, tras recibir los parámetros, se comprueba que sean válidos y busca el último registro para esa cuenta y modifica el número almacenado en el campo "_active" restándole 1. También modifica el campo "Wh_Authorized" de la cuenta a -1. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

Tabla 4-8 - Procedimiento almacenado GetSesions

HvS_WhA_GetSesions
Parámetros de entrada: Ninguno.
Parámetros de salida: @outputCODE.
Descripción: Este procedimiento se encarga de devolver todas las cuentas autorizadas.
Response: Códigos: 200 o 400.
Detalles técnicos: En este procedimiento, se seleccionan todas las cuentas que tengan 1 como valor en los campos "_active" y "Wh_autorized" y se devuelven en el *recordset*. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

4.6.2 Envío automático de iteraciones

El modelo de datos de este sistema está diseñado para almacenar las iteraciones del usuario y establecer un orden de procesamiento mediante lo que se denomina *batchs*. Una vez que se vayan procesado las iteraciones se ira actualizando el estado de cada una de ellas.

4.6.2.1 Tablas

Al igual que en el software de autorización se utiliza únicamente una tabla. Esta tabla se compone, en primer lugar, por un identificador único denominado "ID" que es la clave primaria. El campo "Phone" se utiliza para almacenar el número de teléfono asociado a la cuenta de WhatsApp. El campo "Receiver" contendrá el id del destinatario que puede ser el número de teléfono de una persona o el nombre de un grupo. El campo "QUEUE_ID" contiene el identificador que se asigna a cada iteración. "Creation_timestamp" almacenará la fecha en la que se ejecutó la consulta para almacenar la iteración. Por otro lado "ExpectedProcess_timestamp" almacenará la fecha establecida por el usuario para que se ejecute la iteración". El campo "Content" contendrá el mensaje que se quiere enviar. "BatchNum" almacenará el número del *batch* asociado a esta iteración. "isToBeProcessed" contendrá un número que será 1 si la iteración tiene que ser procesada, o un 0 si ya se ha procesado o está en proceso. "isBeingProcessed" almacenará un número que será 1 si la acción se procesó con éxito, o un -1 si hubo errores. "isProcessedSucces" contendrá un número que será 1 si la iteración se procesó correctamente, o un -1 si no se produjeron errores.

ID	Phone	Receiver	QUEUE_ID	Creation_timestamp	Creation_timestamp	Content	BatchNum	isToBeProcessed	isBeingProcessed	isProcessedSucces
1	15398887724	671844950	2239	2017-06-19 14:43:04:783	2017-06-19 14:43:04:783	Hola	3694	0	0	1
2	15398887724	671844950	2240	2017-06-19 14:43:13:003	2017-06-19 14:43:13:003	¿Que tal?	3694	0	0	1
3	15398887724	671844950	2241	2017-06-19 14:43:38:147	2017-06-19 14:43:38:147	Mensaje enviado a las 18:48	3694	0	0	1
4	15398887724	671844950	2242	2017-06-28 12:19:47:693	2017-06-28 12:19:47:693	Estoy probando el sistema	3694	0	0	1
5	15398887724	671844950	2243	2017-06-28 12:26:29:920	2017-06-28 12:26:29:920	Estoy probando el sistema	3705	0	0	1
6	15398887724	671844950	2244	2017-06-28 12:29:00:800	2017-06-28 12:29:00:800	Estoy probando el sistema	3727	0	0	1
7	15398887724	671844950	2245	2017-06-28 12:33:16:067	2017-06-28 12:33:16:067	Estoy probando el sistema	3737	0	0	1
8	15398887724	671844950	2246	2017-06-28 12:34:21:053	2017-06-28 12:34:21:053	Estoy probando el sistema	3754	0	0	1

Figura 4-16 - Tabla envío automático de iteraciones

4.6.2.2 Procedimientos almacenados

Los procedimientos con los que cuenta este modelo son tres: Uno para añadir iteraciones a la cola, otro para crear *batches* a partir de un conjunto de iteraciones y un último para actualizar el estado de las iteraciones después de que estas se hayan ejecutado.

Tabla 4-9 - Procedimiento almacenado addACTION

HvS_WhV_QUEUE__addACTION
Parámetros de entrada: @Phone, @Receiver, @ActionCODE, @ExpectedProcess_timestamp y @Content. Ninguno puede ser nulo.
Parámetros de salida: @outputCODE.
Descripción: Este procedimiento recibe los parámetros y crea un nuevo registro en la tabla para esa iteración.
Response: Códigos: 200 o 400
Detalles técnicos: En este procedimiento tras recibir los parámetros se comprueba que sean válidos y dependiendo si existe o no un registro anterior con esos datos se crea uno nuevo o se arroja un error. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

Tabla 4-10 - Procedimiento almacenado updateACTION

HvS_WhV_QUEUE__updateACTION
Parámetros de entrada: @BatchNum, @ID, @isSuccessful y @logMessage. Solo @logMessage puede ser nulo.
Parámetros de salida: @outputCODE.
Descripción: Este procedimiento recibe los parámetros y modifica el registro en la base de datos con los nuevos parametros.
Response: Códigos: 200 o 400
Detalles técnicos: En este procedimiento tras recibir los parámetros se comprueba que sean válidos y se modifica en la base de datos el registro de la iteración que corresponda con el id que se toma como parámetro de entrada. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

Tabla 4-11 - Procedimiento almacenado getBATCH

HvS_WhV_QUEUE__getBATCH
Parámetros de entrada: Ninguno.
Parámetros de salida: @outputCODE.
Descripción: Este procedimiento recoge una serie de registros y los agrupa en un *batch* para ser procesados en grupo.
Response: Códigos: 200 o 400
Detalles técnicos: En este procedimiento se recoge una serie de registros que cumplan unas condiciones y posteriormente se devuelven en el *recordset*. Las condiciones son: que no esté asociado ya con un *batch* y que la fecha de ejecución no sea mayor que la actual. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

4.6.2.3 Secuencias

En este modelo de datos se utilizan dos secuencias, una para establecer el "QUEUE_ID" y otra para establecer el "BatchNum".

La secuencia "HvS_WhV_QUEUE_ID" establecerá el "QUEUE_ID" de cada iteración incrementando en 1 el de la iteración anterior. Empieza por el número 2001.

La secuencia "HvS_WhV_QUEUE_BatchN" establecerá el "BatchNum" para las iteraciones, incrementando en 1 el de número del anterior. También empieza por el número 2001.

4.6.3 Monitorización de datos

El modelo de datos del software de monitorización de cuentas está diseñado para almacenar los datos extraídos de las diferentes cuentas de WhatsApp asociadas a la plataforma.

4.6.3.1 Tablas

Al igual que el resto de modelos se utiliza una única tabla. Esta tabla tiene como identificador principal un campo "ID" que debe ser único. Contamos con el campo "userID" que hace referencia al número de teléfono asociado a la cuenta de WhatsApp. El campo "receiverID" almacenará en su interior el id del destinatario, con el que se comunica el remitente, que debe ser el número de teléfono de una persona o el nombre de un grupo. "receiverName" hace referencia al nombre que el remitente asigna a un chat. "IterationTYPE" almacenará el tipo de dato que se guarda y este podrá ser texto, imagen, vídeo, documento, ubicación, audio o contacto. "_timestamp" contiene la fecha en la que se realizó la consulta para almacenar el dato. Y por último en el campo "InteractionJSON" se almacenará toda la información relacionada con una iteración, como por ejemplo la fecha, la hora, el texto, los *emojis*, los links, las imágenes, la pre visualización de los vídeos, la pre visualización de los documentos, las coordenadas de una ubicación o el número de teléfono asociado a un contacto.

ID	receiverID	receiverName	userID	InteractionTYPE	_timestamp	InteractionJSON	ACTIVE
1	5218117788673	+52 1 81 1778 8673	15398887724	text	2017-06-18 16:06:40:087	{"receiverID":"5218117788673","receiverName":"+...	1
2	5218117788673	+52 1 81 1778 8673	15398887724	text	2017-06-18 16:06:40:167	{"receiverID":"5218117788673","receiverName":"+...	1
3	34650640840	Pedro HumedoValley	15398887724	text	2017-06-18 16:06:40:807	{"receiverID":"34650640840","receiverName":"Pe...	1
4	34650640840	Pedro HumedoValley	15398887724	text	2017-06-18 16:06:40:807	{"receiverID":"34650640840","receiverName":"Pe...	1
5	34650640840	Pedro HumedoValley	15398887724	text	2017-06-18 16:06:40:900	{"receiverID":"34650640840","receiverName":"Pe...	1
6	34650640840	Grupo para pruebas	15398887724	text	2017-06-18 16:06:42:430	{"receiverID":"34650640840","receiverName":"Gr...	1
7	34663535521	Jesus Uni	15398887724	text	2017-06-18 16:06:44:040	{"receiverID":"34663535521","receiverName":"Jes...	1
8	34663535521	Jesus Uni	15398887724	text	2017-06-18 16:06:44:040	{"receiverID":"34663535521","receiverName":"Jes...	1
10	34663535521	Jesus Uni	15398887724	text	2017-06-18 16:06:44:103	{"receiverID":"34663535521","receiverName":"Jes...	1
12	34663535521	Jesus Uni	15398887724	text	2017-06-18 16:06:44:120	{"receiverID":"34663535521","receiverName":"Jes...	1
13	34663535521	Jesus Uni	15398887724	text	2017-06-18 16:06:44:167	{"receiverID":"34663535521","receiverName":"Jes...	1
14	34671844950	Yoo	15398887724	text	2017-06-18 16:06:44:773	{"receiverID":"34671844950","receiverName":"Yoo...	1
16	34671844950	Yoo	15398887724	text	2017-06-18 16:06:44:790	{"receiverID":"34671844950","receiverName":"Yoo...	1

Figura 4-17 - Tabla de monitorización de datos

4.6.3.2 Procedimientos almacenados

Los procedimientos almacenados utilizados en este modelo son tres: el primero para almacenar los datos de la monitorización, y el segundo y el tercero para obtenerlos.

Tabla 4-12 - Procedimiento almacenado add

HvS_WhY_AnalyticLOG__add
Parámetros de entrada: @userID, @receiverID, @receiverName, @InteractionType, @InteractionJSON. Ninguno de los parámetros puede ser nulo. **Parámetros de salida**: @outputCODE. **Descripción**: Este procedimiento se encarga de almacenar las iteraciones capturadas en la monitorización de las cuentas de WhatsApp. **Response**: Códigos: 200 o 400 **Detalles técnicos**: En el procedimiento se almacenan las iteraciones de las cuentas de WhatsApp. En el procedimiento simplemente se recogen los parámetros y se insertan en la base de datos. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

Tabla 4-13 - Procedimiento almacenado GetChats

HvS_WhY_GetChats
Parámetros de entrada: @userID. No puede ser nulo. **Parámetros de salida**: @outputCODE. **Descripción**: Este procedimiento es el encargado de devolver todos los chats asociados a una cuenta de WhatsApp de un usuario. **Response**: Códigos: 200 o 400 **Detalles técnicos**: En el procedimiento se recoge el parámetro @userID y se comprueba que existe un usuario con este id. Si existe se devolverá en el *recordset* todos los chats asociados a este. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

Tabla 4-14 - Procedimiento almacenado GetMessages

HvS_WhY_GetMessages
Parámetros de entrada: @userID y @receiverID. Ninguno puede ser nulo.
Parámetros de salida: @outputCODE.
Descripción: Este procedimiento es el encargado de devolver todos los mensajes asociados a un chat.
Response: Códigos: 200 o 400
Detalles técnicos: En el procedimiento se recogen los parámetros @userID y @receiverID y se comprobará si son correctos. En caso afirmativo se devolverá en el *recordset* todos los mensajes asociados al chat entre el remitente y el destinatario. En el "outputCODE" se almacena un código para indicar si se han producido o no errores en la ejecución.

4.7 Librerías de terceros utilizadas

En este punto se tratarán todas las librerías de terceros utilizadas durante la realización de este proyecto, con una breve descripción de su uso.

4.7.1 Librerías de Node.js

- **Body-parser** https://www.npmjs.com/package/body-parser. Consiste en un *midelware* para trabajar con las respuestas JSON que se utilizan en los *endpoints* con método post. (16)
- **Chalk**. https://www.npmjs.com/package/chalk. Utilizada para mostrar con diferentes colores los mensajes que se imprimen en la terminal. (10)
- **Express**. https://www.npmjs.com/package/express. Utilizada como *framework* básico para la creación de la infraestructura web de manera rápida. (17)
- **Express-session**. https://www.npmjs.com/package/express-session. Se utilizada en conjunto con la librería Express para poder crear una sesión. (18)
- **Mssql.** https://www.npmjs.com/package/mssql. Librería que permite y simplifica el uso de conexiones con bases de datos SQL de Microsoft. (13)
- **Request**. https://www.npmjs.com/package/request. Librería que nos permite hacer llamadas HTTP, en nuestro caso es utilizada para extraer la imagen en base64 de un link. (14)
- **Socket.io**. https://www.npmjs.com/package/socket.io. Librería que simplifica la gestión de web sockets. Utilizada para realizar transmisiones de datos en tiempo real. (3)

- **Wd**. https://www.npmjs.com/package/wd. Librería que se utiliza para controlar desde Node.js el navegador Chrome que ejecuta a través de Selenium, permitiendo diversas acciones sobre este. (8)

4.7.2 Librerías para las interfaces web

- **Materialize**. http://materializecss.com/. Framework para desarrollar páginas web siguiendo el diseño material de google. Ofrece gran cantidad de elementos y plantillas. (7)
- **Font Awesome**. http://fontawesome.io/icons/. Ofrece iconos vectoriales que pueden ser personalizados, para mostrar en una página web. (2)

Todas las librerías tratadas en este punto, tanto las de Node.js como las de las interfaces web, son de código abierto y de uso libre.

5. Manual de usuario

En este apartado se explicará de la forma más clara y sencilla posible el funcionamiento de todo el software tratado en este proyecto. Para ello desarrollaré un pequeño manual para los 4 sistemas presentados que componen Sync WhatsApp.

5.1 Autorización

1) En primer lugar, ejecutaremos dos terminales y nos situaremos, en ambas, en el directorio *HvS_WhA* donde se encuentra ubicado este subproyecto.
2) Ahora deberemos ejecutar Selenium en una de las dos consolas, para realizar esta acción bastará con introducir el comando *java -jar selenium-server-standalone-3.3.1.jar*, como se puede ver en la imagen.

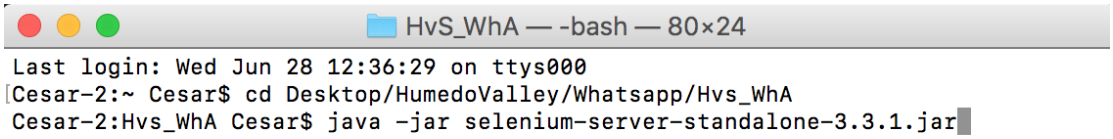

Figura 5-1 - Terminal para iniciar Selenium

3) La otra consola se utilizará para ejecutar el servidor Node.js. Si es la primera vez se ejecutará el comando *npm install* para instalar las librerías necesarias, y después el comando *node index.js*.

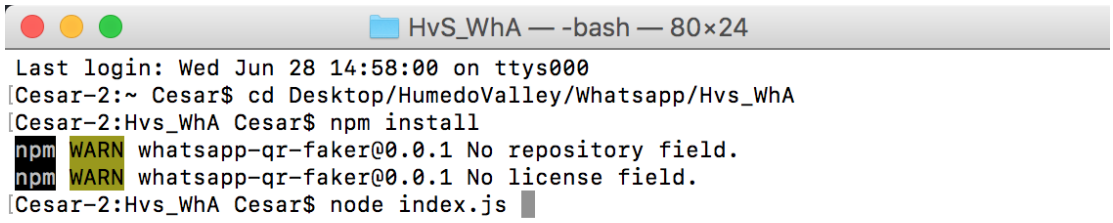

Figura 5-2 - Terminal para iniciar Node.js

4) El usuario que desee dar su autorización deberá introducir la siguiente URL en un navegador web: http://localhost:8080, y se le mostrará un código QR que deberá escanear desde la aplicación de WhatsApp de su Smartphone.
5) Si todo se ha realizado correctamente el usuario ya estaría autorizado y en la base de datos estarían guardadas sus credenciales.

6) Para desautorizar un usuario, se debe hacer una petición POST a la dirección http://localhost:3000/deauthorize, y en el cuerpo de esta petición, en formato JSON, se deberá¤ escribir el identificador del usuario en cuestión.

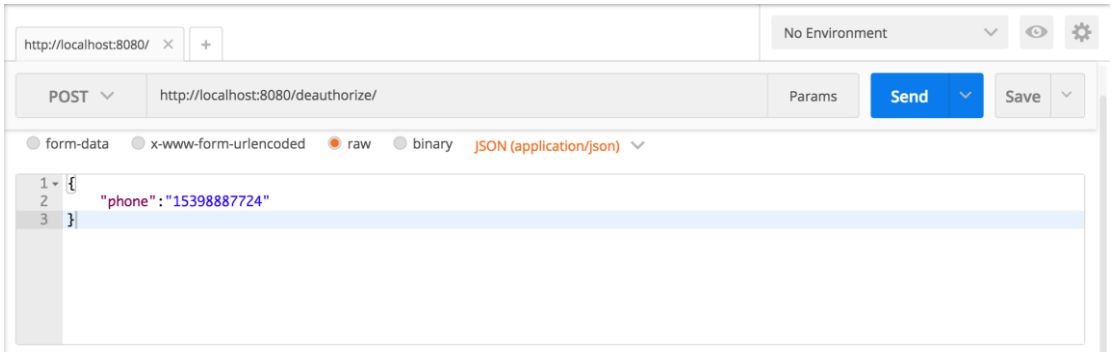

Figura 5-3 - Envío de una petición POST con Postman

5.2 Envío automático de iteraciones

1) En primer lugar, ejecutaremos una terminal y nos situaremos en el directorio *HvS_WhV* donde se encuentra ubicado este subproyecto.
2) La consola se utilizará para ejecutar el servidor Node.js. Si es la primera vez se ejecutará el comando *npm install* y después se ejecutará el comando *node index.js*. Una vez ejecutado, el sistema ya estará funcionando y buscando nuevas iteraciones.
3) El usuario ha de añadir nuevas iteraciones a la base de datos, para ello se utilizará la herramienta *Queue.jar*, como se muestra en la imagen.

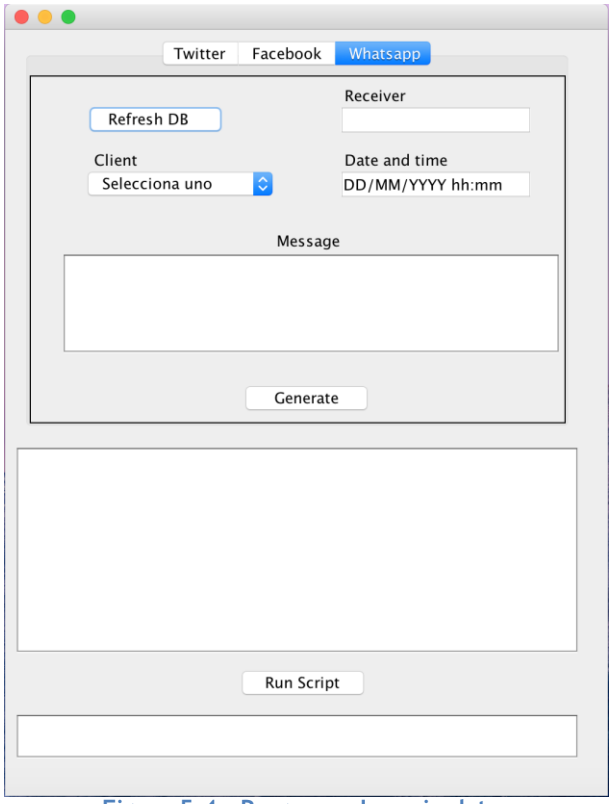

Figura 5-4 - Programa Java sin datos

4) Se deberá completar todos los campos de la herramienta: el remitente de la cuenta, el destinatario, el contenido del mensaje y la fecha en la que se desea que se publique.

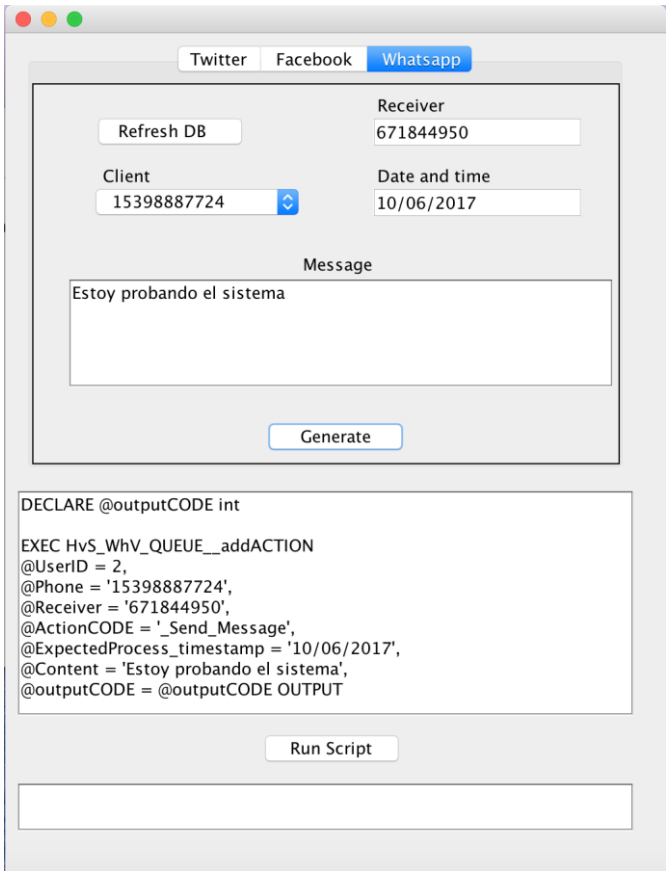

Figura 5-5 - Programa Java con datos

5) Después habrá que generar el script correspondiente a la iteración, para esto bastará con pulsar el botón *Generate*. Posteriormente se pulsará *Run script* para ejecutar el script sobre la base de datos.

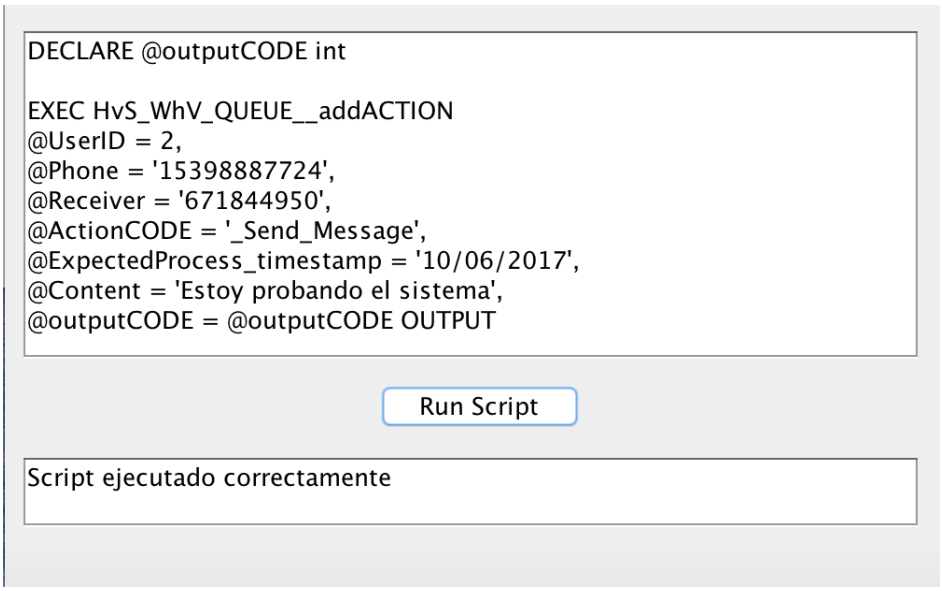

Figura 5-6 - Herramienta Java ejecutando consultas

6) El servidor Node.js iniciado anteriormente se encargará de publicar todas las iteraciones que hayan sido enviadas.

5.3 Monitorización de datos

1) Antes de empezar, para que el sistema funcione correctamente, dispondremos en un directorio cualquiera que contendrá en su interior dos directorios, uno se llamará *nw_js* y contendrá la herramienta NW.js con todos sus archivos, y otro será el directorio *HvS_WhY* donde se encuentra ubicado este subproyecto.

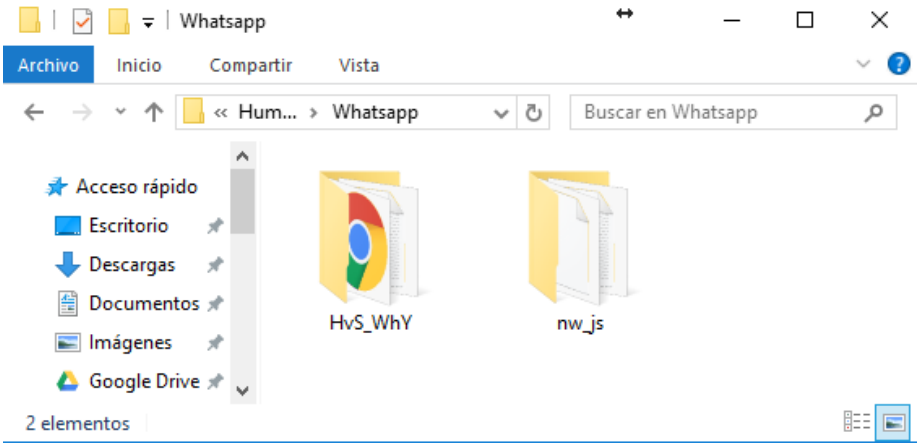

Figura 5-7 - Disposición de directorios

2) Ejecutaremos una terminal y nos situaremos en el directorio *nw_js* y posteriormente ejecutaremos el siguiente comando: *nw ../Hvs_WhY*. Si todo se realizó de la forma correcta se ejecutará el software de monitorización.

3) El usuario podrá observar una interfaz en la que se muestra una lista de usuarios para monitorizar y deberá seleccionar uno de ellos y pulsar el botón *Iniciar* para empezar la captura.

4) Inmediatamente se iniciará la interfaz de monitorización, donde se observará en todo momento el progreso de la captura de datos.

5) Si se desea detener el proceso habrá que hacer click en el botón que se muestra en la figura 5-8 y de esta forma se retrocederá a la interfaz donde se eligen las cuentas para monitorizar.

Figura 5-8 - Botón salir

5.3.1 Interfaz web para mostrar los resultados

1) En primer lugar, ejecutaremos una terminal y nos situaremos en el directorio *HvS_Int/WhatsApp* donde se encuentra ubicado el servidor Node.js y la interfaz web.
2) La consola se utilizará para ejecutar el servidor Node.js. Si es la primera vez se ejecutará el comando *npm install* y después se ejecutará el comando *node index.js*. Una vez ejecutado, Node servirá la interfaz que se encuentra alojada en este directorio.
3) El usuario deberá acceder desde un navegador a la siguiente URL: http://localhost:8082.
4) Inmediatamente se podrá observar una interfaz web donde se muestran las cuentas de WhatsApp autorizadas.
5) Desde esta interfaz el usuario puede desautorizar cuentas, como se muestra en la figura 5-9, o utilizar un buscador para filtrar las cuentas que desea ver.

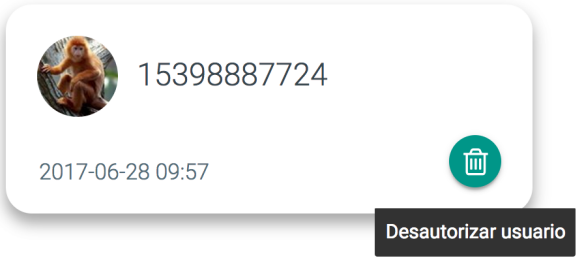

Figura 5-9 - Botón desautorización

6) Cuando seleccione una cuenta se mostrará una página web donde se pueden ver los chats asociados a esa cuenta. El usuario también cuenta con un buscador.
7) Al seleccionar un chat se mostrará una nueva página web con todos los mensajes de asociados a este. También está presente un buscador para poder filtrar mensajes.

6. Conclusiones

Una vez finalizado el proyecto puedo decir que los objetivos han sido cumplidos correctamente en su mayoría. Es cierto que algunas partes del proyecto, me han supuesto grandes esfuerzos de trabajo que no había previsto, también es cierto que a medida que avanzaba, el desarrollo iba siendo más ameno. Por lo tanto, el balance global es positivo.

Los nuevos conocimientos y la experiencia obtenida en el manejo de APIs hacen que merezca la pena desarrollar este proyecto desde cero, pese a que posee una gran complejidad.

El proyecto abarca muchos ámbitos de la programación, pero debido al elevado número de objetivos que se debían cumplir, muchos de esos puntos podrían mejorarse. Dicho esto, siempre se ha buscado la mejor solución a todos los problemas, esto en muchas ocasiones me ha llevado a estancarme buscando tecnologías que cumplieran con los objetivos propuestos.

Al principio, la idea era crear únicamente un software capaz de permitir a los usuarios dar autorización a una plataforma de terceros para utilizar sus cuentas de WhatsApp, de esta forma los usuarios de la plataforma podrían publicar sus iteraciones en WhatsApp de forma automatizada desde esta. Más tarde se decidió crear un sistema que monitorizara estas cuentas para extraer información con el objetivo de mejorar la experiencia de los usuarios en la plataforma.

La interfaz web para mostrar los resultados de la monitorización dio solución al manejo de un gran volumen de datos y servirá más adelante como la base para realizar informes con la información obtenida.

HumedoValley ya contaba con una tecnología de sincronización similar, utilizada para la sincronización de *León Literario* con cuentas de Twitter. Mientras yo desarrollaba *Sync WhatsApp*, otra persona desarrollaba paralelamente un sistema de sincronización con cuentas de Facebook.

Cabe destacar que la seguridad, aunque se tuvo en cuenta, se implementó pensando en el uso en local, con lo que si necesitaran portarse los componentes a maquinas en diferentes lugares, sería necesaria la implementación de un sistema de verificación de acceso que no forma parte de los objetivos este proyecto.

El testeo del software en entornos de altas peticiones sería otro punto necesario que el proyecto no ha abarcado. Con las tecnologías utilizadas la escalabilidad y mantenimiento del código se puede realizar de manera muy fácil por lo que no supondrá ningún problema en el futuro.

Como resumen de las conclusiones se ha desarrollado una arquitectura completamente funcional cumpliendo la mayoría de los objetivos de manera satisfactoria, pero, que se podría mejorar en algunos aspectos y evolucionar en una arquitectura más potente y escalable.

7. Líneas futuras

Todo el software en conjunto se ha desarrollado correctamente siguiendo las especificaciones iniciales, aun así, se han pensado algunas nuevas funcionalidades que se podrían implementar de cara al futuro.

En un futuro habría que realizar la integración completa de todo el sistema con la plataforma *León Literario*. Por ahora se han realizado pequeñas pruebas de integración, pero aún no se ha introducido en la plataforma.

La idea de esta integración es conseguir que los usuarios puedan iniciar sesión en *León Literario* mediante su cuenta de WhatsApp. Esto permitiría realizar una sincronización entre León Literario y WhatsApp. Es decir, se podría publicar en WhatsApp las iteraciones de un usuario de la plataforma. En la figura 7.1 se puede observar cómo sería el inicio de sesión en *León Literario* con una cuenta de WhatsApp.

Figura 7-1 - Iniciar sesión con WhatsApp en León Literario

Otra idea para implementar en el futuro es usar la información de los usuarios de WhatsApp almacenada en la base de datos SQL para realizar estudios y posteriormente elaborar perfiles concretos. De esta forma podríamos ofrecer al usuario una mejor experiencia en *León Literario* y mostrarle eventos concretos basándonos en la información recabada de su cuenta de WhatsApp.

Por otro lado, en cuanto al software encargado del envío automático de iteraciones se podría añadir una funcionalidad más. Esta funcionalidad consiste en, no solo poder enviar mensajes de texto, si no también fotos y vídeos.

Esta función es complicada de implementar debido a que se debería conocer perfectamente la implementación del "evento" que utiliza WhatsApp para poder realizar el envió de archivos multimedia, para luego poder simularlo.

Se ha intentado simular un evento de este tipo sin éxito, debido a que WhatsApp reconoce algunos de los eventos que no son realizados por un usuario físico y los rechaza.

Se ha propuesto una solución que consistiría en simular un *drag and drop* con un programa externo. Este programa debería de arrastrar un archivo desde el directorio que se le indique hasta el interior del navegador, donde esta WhatsApp Web. De esta manera, WhatsApp no tendría forma de detectar si el evento es un evento simulado por un sistema o un evento real de un usuario físico.

Por otro lado, el proyecto está pensado para funcionar de manera local y con pocos usuarios, por lo que, si se desea integrar en la plataforma de la empresa, es probable que haya que hacer algunas adaptaciones. Estas adaptaciones consistirían principalmente en temas de seguridad y de una mayor capacidad para almacenar los datos.

8. Agradecimientos

En primer lugar, quiero agradecer a mi tutor, Pedro Pérez Grande, la oportunidad que me brindó ofreciéndome este proyecto. Durante la realización de este, he aprendido infinidad de cosas. Pedro siempre me ha ayudado y me ha guiado en los momentos difíciles, pero sobre todo me ha dado una gran libertad en cuanto a la realización de tareas y uso de las tecnologías necesarias, lo que me ha permitido desarrollarme más como ingeniero e investigar nuevas tecnologías.

Quiero agradecer a Borja Fernández, compañero de clase y de *HumedoValley*, el apoyo y la ayuda que me ha ofrecido siempre desde el primer día. Borja y yo realizamos proyectos similares paralelamente. Su proyecto consistía en el uso de APIs de Facebook para la integración con *León Literario*.

No puedo olvidar el apoyo y la colaboración recibida por parte del resto de miembros de *HumedoValley*, en especial de Mario Corral. Mario me aportó un proyecto en el que se utilizaban APIs de Twitter para la integración con una plataforma. Este me sirvió como una guía realizar mi proyecto, debido a que en él se trataba la división en tres subproyectos: uno para realizar la autorización de cuentas de usuarios, otro para la automatización del envío de iteraciones y un tercero para analizar toda la actividad de las cuentas.

Me gustaría darle las gracias a Héctor Alaiz Moretón por su labor como tutor y su valiosa ayuda en todo momento.

También me gustaría agradecer a mis padres sus sabios consejos y su comprensión durante esta etapa.

Para finalizar, quiero dedicar las últimas palabras del trabajo de fin de grado a alguien muy especial para mí, mi abuelo. Quiero agradecerle todo el apoyo que he recibo durante toda mi vida, no solo durante esta etapa. Él siempre ha estado conmigo, ayudándome en los momentos difíciles. Cuando pensaba que no era capaz, cuando quería dejarlo porque ya no podía más, cuando no podía abarcar tanto volumen de trabajo, cuando creía que no podía mejorar y no valía, cuando pensaba que yo no era tan bueno como resto, él siempre estaba ahí para convencerme de lo contrario. Me enseñó a rendir, no a rendirme.

9. Lista de referencias

1. **Adrian Manjarrez.** Medium. [En línea] https://medium.com/@Andarms/creando-aplicaciones-de-escritorio-con-tecnologia-web-nw-js-8c97a0f2c87f/.

2. **Awesome, Font.** Font Awesome. [En línea] http://fontawesome.io/.

3. **darrachequesne.** Npmjs. [En línea] https://www.npmjs.com/package/socket.io.

4. **DNN.** DNN. [En línea] http://www.dnnsoftware.com/platform/start/architecture.

5. **dotnetdev.** Stackoverflow. [En línea] http://stackoverflow.com/questions/4844637/what-is-the-difference-between-concurrency-parallelism-and-asynchronous-methods.

6. **Francisco López.** Nodehispano. [En línea] http://www.nodehispano.com/2012/09/introduccion-a-socket-io-nodejs/.

7. **Gogole.** MaterializeCSS. [En línea] http://materializecss.com/.

8. **jlipps.** Npmjs. [En línea] https://www.npmjs.com/package/wd.

9. **Meyer, Jan Christian.** Quora. [En línea] https://www.quora.com/What-are-the-differences-between-parallel-concurrent-and-asynchronous-programming.

10. **qix.** Npmjs. [En línea] https://www.npmjs.com/package/chalk.

11. **Rocio Muñoz.** Cantabriatic. [En línea] http://www.cantabriatic.com/node-js/.

12. **Sherman, Matt.** Stackoverflow. [En línea] http://stackoverflow.com/questions/6133574/how-to-articulate-the-difference-between-asynchronous-and-parallel-programming.

13. **Simek, Patrick.** Npmjs. [En línea] https://www.npmjs.com/package/mssql.

14. **simov.** Npmjs. [En línea] https://www.npmjs.com/package/request.

15. **Utaal.** Stackoverflow. [En línea] http://stackoverflow.com/questions/14795145/how-the-single-threaded-non-blocking-io-model-works-in-node-js/.

16. **Wilson Doug.** Npmjs. [En línea] https://www.npmjs.com/package/body-parser.

17. **Wilson Doug.** Npmjs. [En línea] https://www.npmjs.com/package/express.

18. **Wilson Doug.** Npmjs. [En línea] https://www.npmjs.com/package/express-session.