Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Definiciones y vocabulario técnico

Antes de comenzar con el desarrollo del proyecto, es importante resaltar el significado de ciertos términos usados en el presente documento.

- **Aplicación de mensajería instantánea**: Software que permite comunicación en tiempo real entre dos o más personas basada en texto y archivos multimedia (audio, imagen, vídeo).
- **Facebook Messenger**: Aplicación de mensajería instantánea independiente de Facebook e integrada en dicha red social.
- ***Bot***: Software autónomo destinado a la realización de tareas que simulan e imitan comportamientos humanos.
- ***Bot* Conversacional o *ChatBot***: Software autónomo destinado a responder automáticamente a los mensajes de un usuario, simulando así una conversación real. Puede estar integrado o desarrollado para una aplicación de mensajería instantánea.
- **HTML5**: Lenguaje de marcado para estructurar y presentar contenido en la web.
- **Navegador o *browser***: *Software* que permite el acceso y la visualización de sitios *web*. Su principal función es renderizar el código HTML.
- ***Headless browser***: Navegador sin interfaz gráfica de usuario. Proporciona control automatizado sobre páginas *web*.
- **API**: *Application Programming Interface*, Conjunto de comandos, funciones, protocolos y objetos que permiten la interacción con un sistema externo.

- **Chrome/Chromium**: Navegadores de Google, a nivel funcional son muy parecidos, entre algunas de sus diferencias tenemos que Chromium es Open-Source y no incluye códecs propietarios para archivos multimedia.
- **Puppeteer**: *Headless browser* API de alto nivel para controlar Chromium o Chrome, tanto en modo *headless* como en modo normal.
- **DOM o *Document Object Model***: API que proporciona un conjunto estándar de objetos para representar documentos HTML y XML.
- ***Web Scrapping***: Técnica mediante la cual se extrae información de sitios *web*.
- **Javascript**: Implementación del estándar ECMAScript. Se puede usar tanto en el lado del cliente como del servidor.
- **Node.js**: Entorno de ejecución que permite el uso de Javascript en servidor.
- **Express**: Framework de Node.js que facilita el direccionamiento y la creación de *endpoints* para satisfacer las solicitudes HTTP de cliente.
- **JSON o *JavaScript Object Notation***: Formato de texto para el intercambio de datos.
- **MySQL**: Sistema de gestión de bases de datos relacional. Mediante el uso de SQL (*Structured Query Language*) se pueden realizar operaciones de lectura, escritura, actualización y borrado sobre las tablas de dichas bases de datos.
- **NLP o *Natural Language Processing***: Habilidad de un programa para 'entender' el lenguaje humano (Extraer información esquematizada y sintetizada de este).
- ***Plugin***: *Software* o Herramienta que extiende la funcionalidad de una aplicación.
- ***Emoji***: Se utilizará indistintamente junto con emoticono. Imagen o pictograma usado para expresar ideas, emociones o sentimientos en medios digitales.
- **WBS**: *Work Breakdown Structure*. Descomposición jerárquica del trabajo a realizar para cumplir con los objetivos de un proyecto.

Capítulo 2

Introducción

El auge de las comunicaciones a través de aplicaciones de mensajería instantánea, así como ciertas mejoras en las ciencias de la computación dedicadas al procesamiento del lenguaje natural (NLP) ha traido consigo un resurgimiento del interés por los *bots* conversacionales. Empresas de software relacionadas con la interacción entre usuarios como pueden ser Facebook (Messenger), Telegram, Microsoft (Skype) o Slack, por citar algunas de las más importantes, llevan ya tiempo dando soporte a desarrolladores facilitando documentación, herramientas y API's fácilmente accesibles para la creación de *chatbots* en sus plataformas. Estos *bots* suponen una oportunidad para las empresas, pues pueden facilitar tanto la comunicación con sus clientes, conformar otro canal de ventas, y, dependiendo de su planteamiento, incluso una vía para recabar información de sus posibles consumidores y su público objetivo.

Un estudio reciente [Ubisend, 2017] revela que la opinión que los usuarios tienen de los *bots* está cambiando progresivamente, el 69 % de los encuestados haría uso de un *chatbot* con tal de obtener una respuesta rápida, y el 21 % los usaría porque considera que son la manera más sencilla de contactar con las empresas. Además un 43 % considera que son innovadores y un 30 % considera que son útiles, si bien un 35 % considera que son menos personales en su servicio. La tendencia para los próximos años prevee que la implantación de *bots* por empresas continue. Otro estudio [Oracle, 2017] en el que se han consultado a 800 directores de ventas, directores ejecutivos de marketing y estrategia de Francia, Paises Bajos, Sudáfrica y Reino Unido, indica que para 2020 el 80 % de ellos tendrán algún tipo de *chatbot* accesible al público. Entre los sectores más interesados en desarrollar este nuevo canal de soporte y ventas a clientes, están el financiero y el de seguros. Dos ejemplos son Next Insurance [ZDNet, 2017], startup que lanzó en marzo de 2017 un *bot* para proveer seguros a PYMES, con muy buenos resultados, y Erica [CNBC, 2017], el *chatbot* de Bank of America, para dar soporte a usuarios sobre consejos financieros

y para transacciones cliente a cliente.

Ante este panorama surge una duda natural, ¿cómo asegurarnos de que dichos *bots* tienen la calidad necesaria para llevar a cabo una interacción agradable con sus usuarios? Una primera aproximación podría ser un análisis individual, pero dicho análisis puede estar sesgado por la opinión del que está probándolo. El objetivo de este Trabajo de Fin de Grado (TFG) es crear un analizador de *chatbots* de Facebook Messenger, capaz de extraer métricas útiles sobre su funcionamiento.

2.1. Origen del proyecto

La idea original del proyecto fue de Juanan Pereira, docente en la Universidad del País Vasco (UPV/EHU) del departamento de Lenguajes y Sistemas Informáticos.

Juanan había realizado un análisis de calidad de los *bots* más populares de messenger, junto con Oscar Díaz, también docente de la Universidad del País Vasco. En dicho análisis se obtenía de manera automatizada el número de *likes* en Facebook y el tiempo de respuesta de 100 *bots* diferentes (se analizaron los más populares de Facebook Messenger), y de manera semi-automatizada otras métricas como el entendimiento de lenguaje natural (NLP), variación de respuestas ante un mismo mensaje enviado al *bot*, soporte de comando *cancel*, gestión correcta de pequeños errores ortográficos en los mensajes que les son enviados, y posibilidad de obtención de soporte humano.

La idea tras este análisis era comprobar si se cumple la hipótesis de si existe correlación entre la popularidad de estos (medida en *likes*) y su calidad, medida mediante las distintas métricas expuestas anteriormente. El resultado de dicho análisis fue que no existe correlación entre dichas métricas y la popularidad de los bots.

2.2. Descripción y situación del proyecto

Este TFG parte de cero, y pretende ser una base o un punto de partida para TFG's posteriores que tengan por objetivo realizar análisis sobre la interacción con *bots* en Messenger.

El trabajo consiste en un analizador de *bots* de Facebook Messenger completamente automatizado, capaz de obtener el id asociado a un *bot* en Facebook messenger, el tiempo medio que tarda en responder, si deja algún mensaje sin responder o le toma más tiempo del habitual, si su manera de expresarse se adapta a la que se usa en aplicaciones de mensajería instantánea e incluye *emojis* en sus respuestas, si se excede en el uso de estos, o no los emplea. En mensajería instantánea también son populares los formatos multimedia, como audios, imágenes y vídeos. Se analizará también si el *bot* incluye entre sus respuestas mensajes de este tipo, así como si es capaz de soportar los mensajes que le llegen con este contenido, aportando una respuesta distinta de un simple "no te entiendo". En el análisis también se averiguará si el *bot* soporta el comando básico *help* (ayuda), y si se vale de una funcionalidad muy atractiva para los usuarios que es enviarles las opciones que tienen ante una pregunta en forma de botones. Esta funcionalidad

enriquece la experiencia de los usuarios, y si el *bot* soporta esta funcionalidad, se analizará si responde correctamente (de la misma manera) ante la pulsación de uno de dichos botones y el envío de la respuesta de manera escrita (el usuario no se debe ver limitado a pulsar un botón y abandonar la escritura, debe poder disponer de ambas opciones).

Dicho analizador estará disponible en un servidor, para ser ejecutado como un script al que se deberá pasar como parámetro el nombre del *bot* a analizar. También se desarrollará una interfaz gráfica simple que permita a los usuarios ver los resultados del análisis a medida que se van obteniendo, actualizando gráficas, para que pueda identificar los puntos en los que es posible que el *bot* flojee. (E.g.: ante un determinado mensaje tarda más de lo normal en responder, o no hace uso de *emojis* y contenido multimedia).

2.3. Alicientes para la elección del proyecto

Hay 5 motivos que han contribuido especialmente a que me decante por este proyecto.

Curiosidad

Ya había desarrollado previamente un *bot* para Telegram por iniciativa propia, pero la idea de crear scripts para analizar las respuestas de otros me pareció muy interesante. Surgieron dudas relacionadas con el NLP, con la calidad de las respuestas, heurísticos, y todas despertaban interés. Además algunas de las herramientas a usar me resultan novedosas, y eso siempre supone un reto añadido que contribuye a llamar la atención.

Tiene la capacidad de escalar

Dependiendo de como se plantee el análisis de un *bot*, se pueden llegar a analizar muchos aspectos de este, por lo que el proyecto puede servir de base a futuros desarrollos, que añadan al analizador nuevas capacidades de análisis, mejorándolo y contribuyendo a una mayor completitud de este.

Ausencia de referentes

Aunque a primera vista esto puede ser un punto negativo a la hora de tomar ejemplo y realizar un análisis del estado del arte para ver los puntos fuertes y débiles de las soluciones existentes, también tiene una parte muy positiva, que es que la solución que se desarrolle puede ser de mucha utilidad para futuros trabajos, que tendrán una base respecto de la cual partir.

Aprendizaje continuo

Durante el grado había hecho uso de algunas de las herramientas que utilizaría en el proyecto, pero nunca viene mal reforzar conceptos y aprender nuevos relacionados con lo ya aprendido, además el uso de un automatizador de navegadores es muy atrayente dado que es una nueva herramienta que me puede ser de mucha utilidad en el futuro dada su versatilidad. Generalmente se usan para la realización de pruebas automatizadas, lo cual es importante en el trabajo de desarrollador y es un conocimiento bastante demandado y útil.

Puede ser de ayuda a desarrolladores reales

Puede ser de mucha utilidad para los desarrolladores de *bots*, puesto que actualmente no hay nada similar en el mercado, no existen soluciones de este tipo. Existe una parecida, `testmybot`, que aborda pruebas desde otra perspectiva, comprobando únicamente los mensajes enviados y recibidos, intentando que se ajusten a un flujo previamente establecido, pero prescinde de otras características como pueden ser los elementos multimedia o la interacción mediante botones.

Capítulo 3

Planteamiento Inicial

En este apartado se exponen los objetivos, las herramientas usadas, el alcance, la planificación temporal y las evaluaciones económica y de riesgos realizadas para este proyecto. Así mismo, se muestra como funciona la interacción entre el analizador, el *bot* dentro de Messenger y la interfaz web en caso de que se haga uso de ella.

3.1. Objetivos

Se han identificado tres grupos de objetivos del proyecto. Así, es más sencillo identificarlos, y realizar un seguimiento de su cumplimiento.

Analizador de bots

Bajo este grupo de objetivos identificamos los que debe cumplir el analizador para resultar satisfactorio. Son los siguientes:

- **Identificación del ID - Nombre**: El analizador debe poder aceptar un id o un nombre de un bot para realizar el análisis, y deberá obtener el otro identificador posible (si recibe un nombre, caso más usual, obtendrá el id, y viceversa).
- **Obtención número de *likes* del bot**: El analizador debe obtener el número de likes que el bot analizado tiene en Facebook.
- **Obtención de porcentaje de mensajes con emoticonos**: El analizador debe poder obtener el porcentaje de emoticonos que emita el bot, tanto en mensajes simples como los que incluya en los botones que muestre al usuario.

- **Cálculo del tiempo de respuesta por mensaje y tiempo medio**: Se obtendrá el tiempo de respuesta ante cada mensaje enviado al bot, de esta manera se podrá calcular el tiempo de respuesta medio y se podrán identificar los mensajes ante los cuales el bot responde con mayor latencia.
- **Identificación de comando *help***: Se identificará si el *bot* es capaz de soportar el comando *help*, ayuda, comparando la respuesta del comando con la que devuelve ante un mensaje aleatorio.
- **Cálculo de porcentaje de mensajes multimedia**: Se identificarán los mensajes multimedia que envie al usuario el *bot*.
- **Tolerancia ante respuestas con botones y escritas**: Cuando el *bot* envíe al usuario botones, se comprobará si se tiene el mismo efecto escribiendo manualmente la respuesta o utilizando el botón proporcionado.

Interfaz web

En esta sección se muestran los objetivos principales que ha de cumplir la Interfaz web. Su completo desarrollo se ha basado en los siguientes objetivos:

- **Uso de gráficos**: Se utilizarán gráficos para mostrar los datos que se vayan obteniendo con el analizador, se mostrarán gráficos relativos al análisis de determinadas métricas, como latencia de las respuestas de todos los mensajes enviados, porcentaje de uso de emojis y archivos multimedia. Los gráficos deberán permitir identificar fácilmente detalles importantes, como por ejemplo los mensajes a los que se tarda más en ofrecer una respuesta.
- **Diseño atractivo y sobrio**: Los elementos de la página deberán estar ordenados y resultar agradables a la vista, con una disposición que facilite la comprensión de la información mostrada de un vistazo rápido.
- **Actualización en tiempo real**: Los datos que se muestren en la interfaz web durante el análisis deberán actualizarse en tiempo real a medida que los vaya generando el analizador, y se deberá tener *feedback* una vez el análisis haya terminado.

Generales y de documentación

En este apartado se muestran los objetivos comunes tanto para el analizador como para la interfaz web. Tambien se presentarán los objetivos a alcanzar con la documentación y la memoria del proyecto.

- **Documentación**: Objetivo común a los dos apartados anteriores (el analizador y la interfaz web resultante). Esta ha de estar bien definida, escrita en un lenguaje claro y dividida en secciones específicas. Así, en caso de surgir algún problema o error, será posible consultar dicho documento o los comentarios del el código para poder localizar fácilmente lo que ha fallado. Además, esto hará más sencillo las posibles modificaciones y actualizaciones futuras.
- **Preparado para modificaciones y nuevas funcionalidades**:El analizador y su correspondiente interfaz web deberán ser fáciles de modificar y se tiene que poder añadir nuevas funcionalidades sin problemas.

3.2. Herramientas usadas

En esta sección se especifican las diferentes herramientas, librerías y *plugins* que se utilizarán para el desarrollo del proyecto. También se explica, brevemente en qué partes del proyecto se va a usar cada una de las herramientas mencionadas.

- **Git**: Es un sistema de control de versiones de código abierto, haremos uso de él para tener un cierto orden a la hora de desarrollar los componentes que integran el trabajo. Su página oficial es: `https://git-scm.com/`
- **Github**: Github es una plataforma para gestionar proyectos haciendo uso de Git. Nosotros la usaremos para gestionar el código del proyecto, y tener la posibilidad de continuar trabajando en el proyecto de manera sencilla. Su página oficial es: `https://github.com/`
- **Nodejs**: Servidor asíncrono que utiliza como lenguaje javascript. Lo usaremos para el desarrollo de la integración del analizador con su correspondiente interfaz gráfica. `https://nodejs.org/en/`
- **Express**: *Framework* para Nodejs que facilita y simplifica el proceso de enviar los archivos al cliente. Se hará uso de él para enviar los archivos correspondientes a la interfaz web del analizador por parte del servidor. Su página oficial es: `http://expressjs.com/es/`
- **Google Chrome & Firefox**: Dos de los navegadores más populares que existen actualmente, se usarán para visualizar la aplicación web y comprobar

que su funcionamiento es correcto. Sus páginas oficiales son, respectivamente: `https://www.google.es/chrome/index.html` y `https://www.mozilla.org/es-ES/firefox/`

- **Puppeteer**: API de alto nivel para controlar los navegadores de Google Chromium y Chrome. El analizador hará uso continuo de esta API para interactuar con Facebook Messenger y sus bots simulando ser usuarios de esta aplicación de mensajería. Su repositorio en github es: `https://github.com/GoogleChrome/puppeteer`

- **HTML5**: Lenguaje de marcado para estructurar y presentar contenido en la web. Quinta revisión del estándar HTML4. Lo usaremos para establecer la estructura de la interfaz web que hará uso del analizador de bots.

- **JavaScript**: Lenguaje de programación interpretado usado principalmente en páginas web dinámicas para el frontend. Gracias a Nodejs y su diseño, lo usaremos también para la parte relativa al servidor (*backend*).

- **MySQL**: Sistema de gestión y administración de bases de datos de código abierto. Lo usaremos para añadir la opción de almacenar los datos que genere el analizador.

- **Overleaf**:Overleaf es una herramienta de escritura y publicación colaborativa online de LaTeX y texto enriquecido. En ella, se puede ver en tiempo real, sin necesidad de una compilación explícita cómo se va modificando el fichero PDF resultante del código que se escribe. Haremos uso de esta herramienta para redactar la memoria del proyecto en lenguaje LaTeX. Su página oficial es: `https://www.overleaf.com/`.

- **Sublime Text 2**: Editor de texto para la creación y edición de código. Ofrece resaltado de sintaxis y soporte multiplataforma. Esta herramienta será utilizada para desarrollar todo el código, tanto del analizador como de la interfaz web resultante. Su página oficial es: `https://www.sublimetext.com/`.

- **DocBlockr**: *Plugin* o paquete para Sublime Text2, que facilita la creación de comentarios para funciones, simplificando y ayudando bastante a documentar el código. Su repositorio en Github es: `https://www.github.com/spadgos/sublime-jsdocs`

- **Toms Planner**: Herramienta *software* para la realización de diagramas de Gantt en línea que permite a cualquier persona crear, colaborar y compartir diagramas de Gantt. Se usará para la creación del diagrama Gantt de este proyecto. Su página oficial es: `https://www.tomsplanner.es/`

- **Socket.io**: Librería para la implementación de *WebSockets* en javascript, ofreciendo así la posibilidad de establecer una comunicación en tiempo real y bidireccional entre cliente y servidor. Se hará uso de ella para la comunicación entre el servidor y el analizador y entre el servidor y la interfaz web. Su página oficial es: `https://socket.io/`

- **Chart.js**: Librería simple para la creación de gráficos en javascript. Se usará para los gráficos generados en la interfaz gráfica del analizador. Su página oficial es: `https://www.chartjs.org/`

- **Draw.io**: Herramienta *online* para la creación de diagramas. Se hará uso de ella para realizar los diagramas del proyecto relacionados con el desarrollo de código y con la descomposición del trabajo en WBS. Su página oficial es: `https://www.draw.io/`

- **Python**: Lenguaje de programación interpretado con una sintaxis muy sobria. Este será usado para crear un *script* que obtenga todos los *bots* registrados en chatbottle. Su página oficial es: `https://www.python.org/`

3.3. Arquitectura

El analizador hace uso de la librería puppeteer para comunicarse con chromium en modo *headless*, esto es, no se abre la interfaz gráfica del navegador sino que se simula. Gracias a esto es posible acceder a Messenger, identificarse con un usuario y comunicarse con el *bot* que ha sido indicado.

El analizador puede ser ejecutado como un *script* independiente, o si es necesario un feedback visual, se puede ejecutar con un *script* que además de ejecutar el analizador pone en funcionamiento un servidor encargado de comunicarse con el analizador por webSockets para llevar los resultados del análisis en tiempo real a una interfaz web.

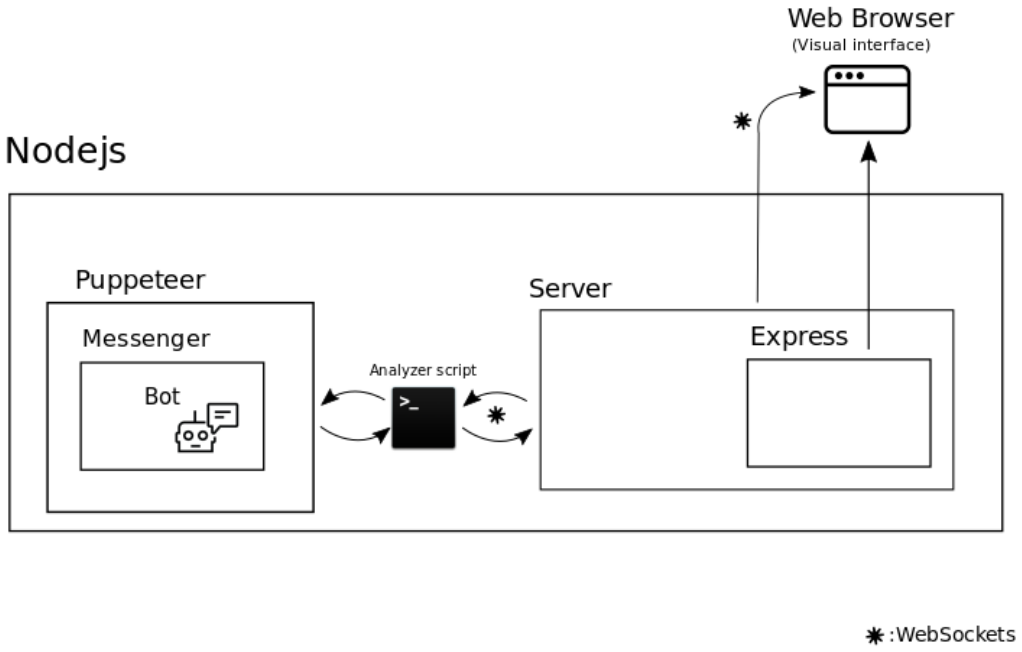

Figura 3.1: Arquitectura del servidor

Esta interacción entre el analizador con Facebook Messenger por una parte, y del servidor con la página web y con el analizador se realiza mediante webSockets, y se puede apreciar en la figura 3.1.

3.4. Alcance

Para definir el alcance de este proyecto se ha realizado un diagrama WBS (*Work Breakdown Structure*, estructura de descomposición de trabajo). Se han identificado los siguientes siete bloques: aprendizaje, organización, captura de requisitos, análisis y diseño, implementación y desarrollo, pruebas y documentación.

- **Organización**: Bloque que contiene las tareas necesarias para el correcto desarrollo del proyecto.
- **Aprendizaje**: Bloque al que pertenecen las tareas relacionadas con el estudio de las herramientas usadas durante el proyecto.
- **Captura de requisitos**: Bloque que agrupa las tareas en las cuales se recogen los datos necesarios para identificar las funcionalidades a realizar durante el desarrollo del proyecto.
- **Análisis y diseño**: En este bloque se encuentran las tareas que se encargan de describir gráficamente el código y la estructura del proyecto.
- **Implementación y desarrollo**: Este bloque alberga todas las tareas que impliquen la escritura de código en cualquier tipo de lenguaje de programación.
- **Pruebas**: Bloque en el que se incluyen las tareas con los métodos utilizados para la comprobación del correcto funcionamiento de todo lo implementado y desarrollado para el proyecto.
- **Documentación**: A este último bloque pertenecen las tareas relacionadas con la redacción de documentación sobre el proyecto. Se realizará durante el transcurso de este.

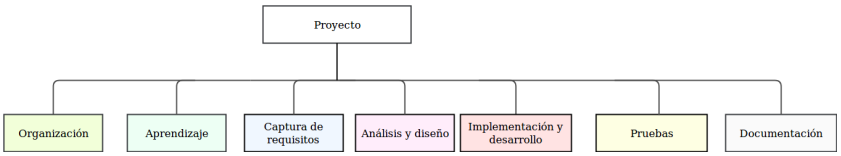

Figura 3.2: Diagrama WBS principal

En la figura 3.2 se puede reconocer el diagrama con cada uno de los bloques identificados. Cada bloque del WBS es precursor al siguiente descrito en la lista, exceptuando el aprendizaje y la documentación, y dentro del bloque de organización, la tarea correspondiente a reuniones tampoco es completamente precursora

de otras. La tarea de reuniones y las tareas de documentación se desarrollan en paralelo a las demás durante todo el proyecto, mientras que las tareas relacionadas con el aprendizaje se realizan durante los dos primeros meses del proyecto. Se va a proceder a detallar los datos de cada uno de los bloques identificados.

3.4.1. Organización

En esta sección se detallan las tareas que se han identificado para la correcta realización del proyecto. La figura 3.3 muestra el diagrama WBS con las tareas identificadas. En las tablas siguientes se muestra la información de cada tarea.

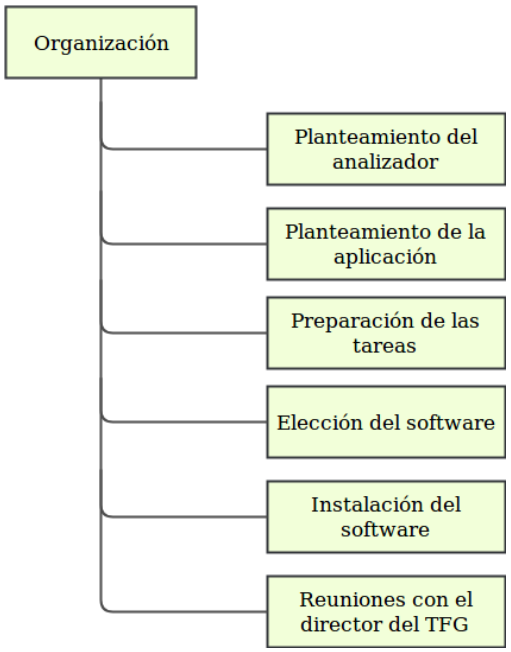

Figura 3.3: Diagrama WBS del bloque de organización

Planteamiento del analizador
Bloque de trabajo: Organización **Duración estimada**:3
Descripción: Tarea en la que se recogen las ideas de qué puede analizar el bot.
Salidas/Entregables: Documento con una lista de las posibles métricas a recoger por el analizador.
Recursos necesarios:Cuaderno y bolígrafo.

Planteamiento de la aplicación
Bloque de trabajo: Organización **Duración estimada**:1
Descripción: Tarea en la que se recogen las funcionalidades que puede tener la página web así como funcionalidades que se pueden añadir al proyecto.
Salidas/Entregables: Documento con una lista de las ideas aplicables al desarrollo de la página *web*.
Recursos necesarios:Cuaderno y bolígrafo.

Preparación de las tareas
Bloque de trabajo: Organización **Duración estimada**:4
Descripción: En esta tarea se identifican, se describen y se planifican temporalmente las tareas a realizar en el proyecto.
Salidas/Entregables: Diagrama WBS con todas las tareas a realizar , y documento en el que se explica cada una de estas.
Recursos necesarios:Cuaderno y bolígrafo.

Elección del *software*
Bloque de trabajo: Organización **Duración estimada**:1
Descripción: Elección del *software* más adecuado para la realización del proyecto.
Salidas/Entregables: Lista con el *software* y *plugins* con los que se realizará el proyecto
Recursos necesarios:Cuaderno y bolígrafo.

Instalación del *software*
Bloque de trabajo: Organización **Duración estimada**:3
Descripción: En esta tarea se refleja la instalación del *software* identificado para la realización del proyecto.
Salidas/Entregables: NO.
Recursos necesarios:Ordenador conectado a Internet.

Reuniones con el director del TFG
Bloque de trabajo: Organización **Duración estimada**:10
Descripción: Tarea que refleja las reuniones que tendrán lugar durante el desarrollo del proyecto con el director del mismo. En estas se analizarán los problemas que surgan, modificaciones en la planificación o funcionalidades del proyecto.
Salidas/Entregables: Documento con los puntos importantes de la reunión, como consejos, modificaciones o tareas pendientes identificadas en la reunión.
Recursos necesarios: Papel y bolígrafo.

3.4.2. Aprendizaje

En esta sección se muestran las tareas que tienen que ver con el aprendizaje de las herramientas, programas y librerías que se usarán en el proyecto. La figura 3.4 muestra el WBS con las tareas identificadas en el bloque de aprendizaje. En las tablas siguientes se detalla la información de cada tarea.

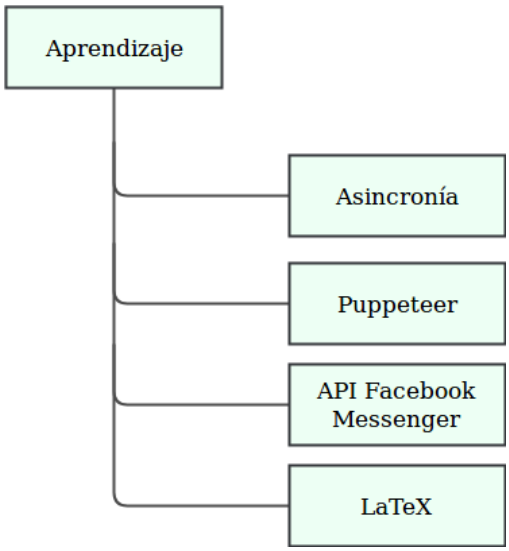

Figura 3.4: Diagrama WBS del bloque de aprendizaje

Asincronía
Bloque de trabajo: Aprendizaje **Duración estimada**:10
Descripción: Aprendizaje del uso de async/await en Javascript, así como el tratamiento de excepciones.
Salidas/Entregables: NO
Recursos necesarios:Sublime Text 2, Nodejs

Puppeteer
Bloque de trabajo: Aprendizaje **Duración estimada**:15
Descripción: Aprendizaje del uso de async/await en Javascript, así como el tratamiento de excepciones.
Salidas/Entregables: NO
Recursos necesarios:Sublime Text 2, Documentación de la API, Nodejs

API Facebook Messenger
Bloque de trabajo: Aprendizaje **Duración estimada**:2
Descripción: Aprendizaje del uso de la API de Facebook Messenger, construcción de un bot.
Salidas/Entregables: NO
Recursos necesarios:Sublime Text 2, Documentación de la API, Nodejs

LaTeX
Bloque de trabajo: Aprendizaje **Duración estimada**:13
Descripción: Aprendizaje de la sintáxis y el uso de LaTeXuso de la API de Facebook Messenger, construcción de un bot.
Salidas/Entregables: NO
Recursos necesarios:`https://www.overleaf.com/`, Documentación de LaTeX

3.4.3. Captura de requisitos

En este bloque se refleja la planificación de las tareas que identifican los requisitos que deben satisfacer tanto el analizador como la página *web* desarrollados durante el proyecto. En la figura 3.5 se muestra el desglose de las tareas que componen este bloque, las cuales vamos a proceder a detallar.

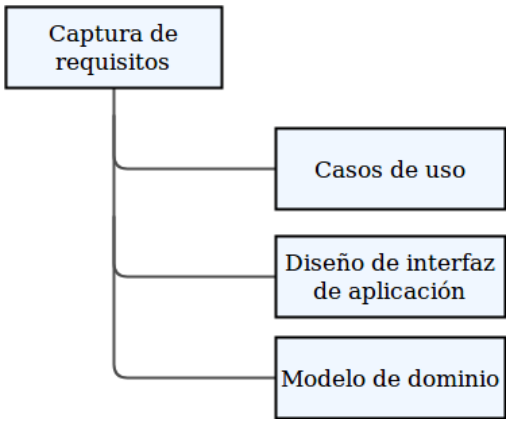

Figura 3.5: Diagrama WBS del bloque de captura de requisitos

Casos de uso
Bloque de trabajo: Captura de requisitos **Duración estimada**:4
Descripción: Tarea destinada a identificar en un diagrama los casos de uso correspondientes a la página *web*.
Salidas/Entregables: Diagrama de casos de uso
Recursos necesarios:Papel y bolígrafo.

Diseño de interfaz de aplicación
Bloque de trabajo: Captura de requisitos **Duración estimada**:17
Descripción: Esbozo de las interfaces de la página *web*, tanto en papel como digitalmente.
Salidas/Entregables: NO
Recursos necesarios:Papel y bolígrafo.

Modelo de dominio
Bloque de trabajo: Captura de requisitos **Duración estimada**:4
Descripción: En esta tarea se identifican las entidades necesarias en este proyecto y posteriormente reflejarlas en el diagrama del modelo de dominio.
Salidas/Entregables: Diagrama del modelo de dominio
Recursos necesarios: Papel y bolígrafo.

3.4.4. Análisis y diseño

En este bloque están las tareas correspondientes a análisis y diseño. En la figura 3.6 se pueden identificar el correspondiente desglose de estas. A continuación se exponen los detalles de cada una de ellas.

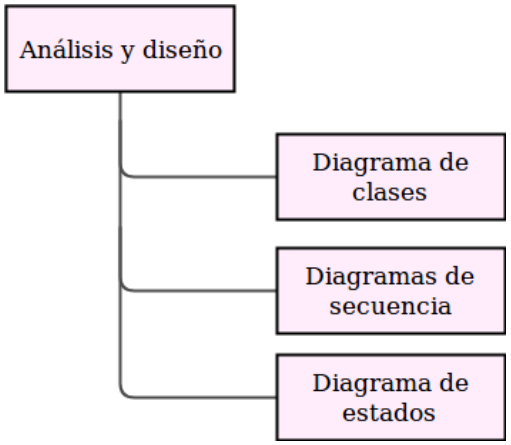

Figura 3.6: Diagrama WBS del bloque de análisis y diseño

Diagrama de clases
Bloque de trabajo: Análisis y diseño **Duración estimada**:4
Descripción: Creación del diagrama de clases que necesitará el analizador para satisfacer los objetivos del proyecto.
Salidas/Entregables: Diagrama de clases.
Recursos necesarios: Papel y bolígrafo.

Diagrama de secuencia
Bloque de trabajo: Análisis y diseño **Duración estimada**:8
Descripción: Realización de los diagramas de secuencia de los *scripts* más interesantes del analizador.
Salidas/Entregables: Diagramas de secuencia
Recursos necesarios: Papel y bolígrafo.

Diagrama de estado
Bloque de trabajo: Análisis y diseño **Duración estimada**:3
Descripción: Desarrollo del diagrama que identifica los diferentes estados del analizador durante la ejecución de sus *scripts*.
Salidas/Entregables: Diagrama de estados
Recursos necesarios: Papel y bolígrafo.

3.4.5. Implementación y desarrollo

En este bloque se reflejan las tareas a realizar durante la implementación y desarrollo del proyecto. Tal y como se muestra en la Figura 3.7 , este apartado se divide en dos secciones, las cuales se describen y se detallan a continuación, junto con las tareas que las componen.

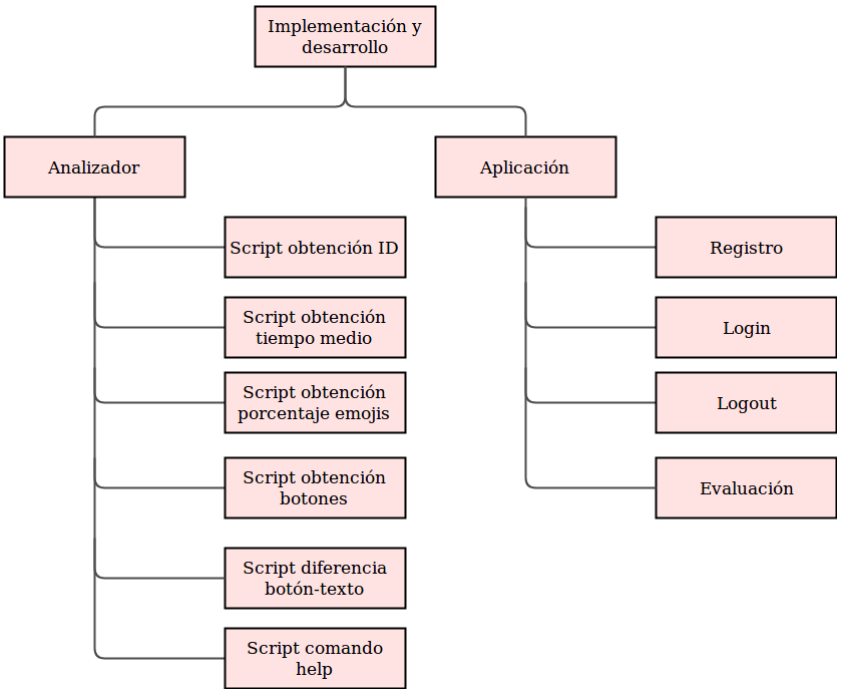

Figura 3.7: Diagrama WBS del bloque de implementación y desarrollo

Desarrollo del analizador

En esta sección se muestran las tareas que se corresponden con el desarrollo del analizador.

Script obtención de ID
Bloque de trabajo: Implementación y desarrollo - Desarrollo del analizador **Duración estimada**:4
Descripción: *Script* encargado de obtener el ID que Facebook Messenger ha asignado al *bot* correspondiente.
Salidas/Entregables: Archivo Javascript capaz de obtener el ID del *bot*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Script obtención tiempo medio de respuesta
Bloque de trabajo: Implementación y desarrollo - Desarrollo del analizador **Duración estimada**:6
Descripción: *Script* encargado de obtener el tiempo de respuesta medio que tarda en responder el *bot* correspondiente.
Salidas/Entregables: Archivo Javascript capaz de obtener el tiempo medio de respuesta del *bot*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Script obtención porcentaje de *emojis*
Bloque de trabajo: Implementación y desarrollo - Desarrollo del analizador **Duración estimada**:6
Descripción: *Script* encargado de obtener el porcentaje de uso de *emojis* por parte del *bot* correspondiente.
Salidas/Entregables: Archivo Javascript capaz de obtener el porcentaje de *emojis* que usa el *bot*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Script obtención botones
Bloque de trabajo: Implementación y desarrollo - Desarrollo del analizador **Duración estimada**:6
Descripción: *Script* encargado de obtener los botones que envie el bot al usuario, las opciones que este tiene para responder.
Salidas/Entregables: Archivo Javascript capaz de obtener los botones que envíe el *bot* como respuesta (si el *bot* los envía).
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Script que compruebe la diferencia entre botón y texto
Bloque de trabajo: Implementación y desarrollo - Desarrollo del analizador **Duración estimada**:10
Descripción: *Script* encargado de analizar la diferencia entre responder al *bot* pulsando uno de los botones que ha enviado al usuario, o responderle con el texto que posee dicho botón, ambas opciones deberían ser aceptadas por este.
Salidas/Entregables: Archivo Javascript capaz de analizar la respuesta del *bot* ante una pulsación de un botón y escritura del texto del botón.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Script respuesta help
Bloque de trabajo: Implementación y desarrollo - Desarrollo del analizador **Duración estimada**:6
Descripción: *Script* encargado de analizar la respuesta del *bot* ante un mensaje *help* o ayuda. Se comparará la respuesta obtenida con la obtenida ante un conjunto de carácteres aleatorio (sin sentido.)
Salidas/Entregables: Archivo Javascript capaz de analizar la respuesta del *bot* ante el comando *help* o ayuda.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Desarrollo de la aplicación

En esta sección se muestran las tareas que se corresponden con el desarrollo de la aplicación *web*.

Registro
Bloque de trabajo: Implementación y desarrollo - Desarrollo de la aplicación **Duración estimada**:40
Descripción: Implementación del registro en la aplicación *web*.
Salidas/Entregables: Archivos con el código del desarrollo para registrarse en la aplicación *web*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Mozilla Firefox, Google Chrome, MySQL.

Login
Bloque de trabajo: Implementación y desarrollo - Desarrollo de la aplicación **Duración estimada**:13
Descripción: Implementación del login en la aplicación *web*.
Salidas/Entregables: Archivos con el código del desarrollo de la funcionalidad para identificarse en la aplicación *web*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Mozilla Firefox, Google Chrome, MySQL.

Logout
Bloque de trabajo: Implementación y desarrollo - Desarrollo de la aplicación **Duración estimada**:13
Descripción: Implementación del logout en la aplicación *web*.
Salidas/Entregables: Archivos con el código del desarrollo de la funcionalidad de deslogeo de la aplicación *web*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Mozilla Firefox, Google Chrome, MySQL.

Realizar evaluación
Bloque de trabajo: Implementación y desarrollo - Desarrollo de la aplicación **Duración estimada**:10
Descripción: Realización de evaluación de un bot y presentado de métricas analizadas y resultados.
Salidas/Entregables: Archivos con el código del desarrollo de la funcionalidad que hace uso del analizador en la aplicación *web*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Mozilla Firefox, Google Chrome, MySQL.

3.4.6. Pruebas

En este bloque se reflejan las tareas identificadas relacionadas con la evaluación tanto del analizador como de la aplicación *web*. En la Figura 3.8 se muestra el desglose de las tareas de este bloque. A continuación se detalla cada una de ellas.

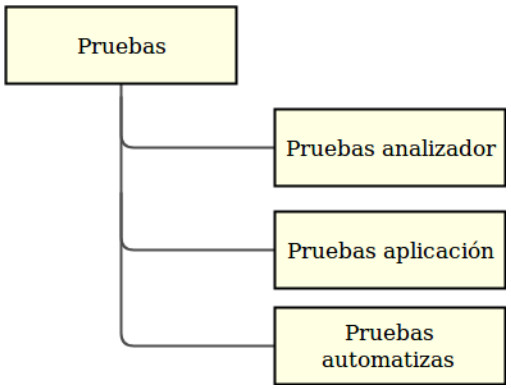

Figura 3.8: Diagrama WBS del bloque de pruebas

Pruebas relativas al analizador
Bloque de trabajo: Pruebas **Duración estimada**:20
Descripción: Se usa el analizador en un *script* para evaluar 100 *bots*. Se comprueba que los resultados son coherentes.
Salidas/Entregables: Archivos con el código que realiza la evaluación del analizador.*web*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Mozilla Firefox, Google Chrome, MySQL.

Pruebas relativas a la aplicación
Bloque de trabajo: Pruebas **Duración estimada**:30
Descripción: Se comprueba que el funcionamiento de la aplicación *web* es correcto mediante la realización de las pruebas correspondientes.
Salidas/Entregables: NO.
Recursos necesarios: Ordenador conectado a internet, Nodejs, Mozilla Firefox, MySQL.

3.4.7. Documentación

Este bloque se divide en dos secciones, documentación del código y memoria del proyecto. Esta división se puede apreciar en la Figura 10.2 ,ambas secciones se

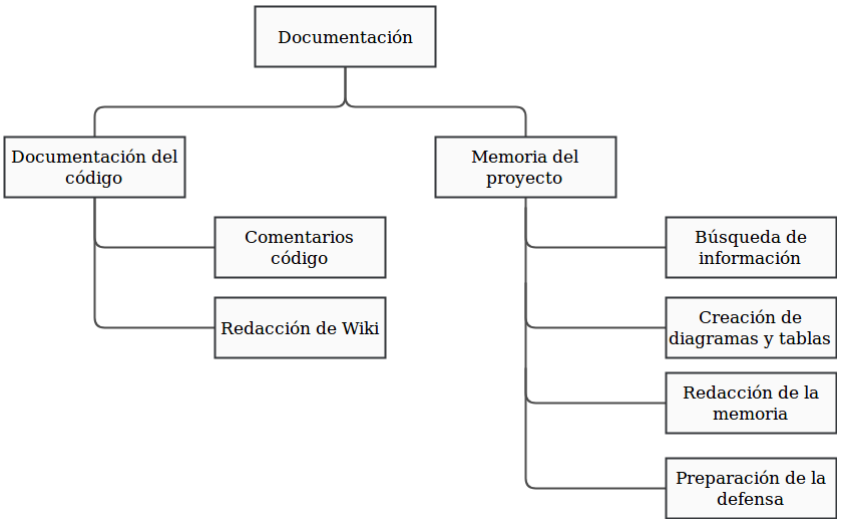

Figura 3.9: Diagrama WBS del bloque de documentación

describen y se detallan a continuación, junto con las tareas que las componen.

Documentación del código

En esta sección se muestran las tareas que se corresponden con el desarrollo de la documentación del código e instalación de lo desarrollado durante el proyecto.

Comentarios código
Bloque de trabajo: Documentación - Documentación código **Duración estimada**:5
Descripción: Se documenta el funcionamiento de cada método de los *scripts*, parámetros de entrada y tipo de salida.
Salidas/Entregables: Comentarios en código.
Recursos necesarios: Ordenador conectado a internet, Sublime Text2, plugin docblockr.

Redacción de Wiki
Bloque de trabajo: Documentación - Documentación código **Duración estimada**:15
Descripción: Se documentan los pasos necesarios para poder instalar el analizador y crear la Base de Datos de la que hace uso.
Salidas/Entregables: Wiki.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2.

Memoria del proyecto

En esta sección se muestran las tareas que se corresponden con el desarrollo de la memoria del proyecto.

Búsqueda de información
Bloque de trabajo: Documentación - Memoria del proyecto **Duración estimada**:15
Descripción: Búsqueda de información relevante para el proyecto.
Salidas/Entregables: Bibliografía.
Recursos necesarios: Ordenador conectado a internet.

Creación de diagramas y tablas
Bloque de trabajo: Documentación - Memoria del proyecto **Duración estimada**:4
Descripción: Creación de las tablas y diagramas de los que se hará uso en este documento.
Salidas/Entregables: Diagramas y tablas.
Recursos necesarios: Ordenador conectado a internet, Overleaf, Tomsplanner

Redacción de la memoria
Bloque de trabajo: Documentación - Memoria del proyecto **Duración estimada**:45
Descripción: Redacción de la memoria del proyecto.
Salidas/Entregables: Memoria del proyecto.
Recursos necesarios: Ordenador conectado a internet, Overleaf.

Preparación de la defensa
Bloque de trabajo: Documentación - Memoria del proyecto **Duración estimada**:20
Descripción: Preparación de la defensa del proyecto.
Salidas/Entregables: NO.
Recursos necesarios: Documentación y proyecto.

3.4.8. Síntesis de la planificación realizada

A continuación se muestran las tareas identificadas, cada una con su nombre, tiempo estimado y relaciones de dependencia. Como se puede observar en la 3.3 el número de horas totales que se estima para la duración del proyecto asciende a 341.

ID	Nombre tarea	Predecesores	Tiempo estimado
	Organización		24
1	Planteamiento del analizador	-	3
2	Planteamiento de la aplicación	-	1
3	Preparación de las tareas	1,2	6
4	Elección del software	3	1
5	Instalación del software	4	3
6	Reuniones con el director del TFG	3	10
	Aprendizaje		40
7	Asincronía	-	10
8	Puppeteer	7	15
9	Express	-	2
10	Latex	-	13
	Captura de requisitos		25
11	Casos de uso	3	4
12	Diseño de interfaz de aplicación	2,6	17
13	Modelo de dominio	11,12	4
	Análisis y diseño		15
14	Diagrama de clases	13	4
15	Diagrama de secuencia	14	8
16	Diagramas de estado	11	3

Cuadro 3.1: Duración y dependencias de los 4 primeros grupos de tareas

ID	Nombre tarea	Predecesores	Tiempo estimado
	Implementación y desarrollo		104
	Desarrollo del analizador	-	48
17	Script obtención de ID	1,8	4
18	Script obtención tiempo medio respuesta	1,8	6
19	Script obtención porcentaje de *emojis*	1,8	6
20	Script obtención botones	1,8	10
21	Script diferencia botón-texto	1,8	14
22	Script respuesta help	1,8	8
	Desarrollo de la aplicación		56
23	Registro	2,7,9,13	20
24	Login	2,7,9,13	13
25	Logout	2,7,9,13	13
26	Realizar evaluación - Mostrar datos	1,2,7,9,13,17-22	10

Cuadro 3.2: Duración y dependencias de las tareas correspondientes a implementación y desarrollo

ID	Nombre tarea	Predecesores	Tiempo estimado
	Pruebas		50
27	Pruebas relativas al analizador	17-22	20
28	Pruebas relativas a la aplicación	23-26	30
	Documentación		83
	Documentación código		10
29	Comentarios código	17-22,23-26	5
30	Redacción de Wiki	27,28	5
	Memoria del proyecto		73
31	Búsqueda de información	1,2	4
32	Creación de diagramas y tablas	3	4
33	Redacción de la memoria	10,29,30,31,32	45
34	Preparación de la defensa	34	20
	TOTAL	35	341

Cuadro 3.3: Duración y dependencias de las tareas correspondientes a documentación y pruebas

3.5. Planificación temporal

Ya conocemos las tareas a realizar y su duración, ahora hay que planificar su desarrollo en el tiempo estimado para el proyecto. En el diagrama Gantt que se muestra en la figura 3.10, se puede observar la planificación de los bloques analizados en el alcance del proyecto. Vamos a establecer un horario de 25 horas semanales. El proyecto comienza el día 10 de enero de 2018, y se ha planificado su finalización el día 16 de julio del mismo año, por lo que tenemos aproximadamente 6 meses. Teniendo en cuenta que un mes tiene de media 4 semanas, con la siguiente fórmula calculamos las horas resultantes:

$$Meses * semanas\,mes * horas\,trabajo\,semana = 25 * 4 * 6 = 600h$$

Se trabajará fines de semana, festivos y laborables indistintamente si fuese necesario, con el fin de cumplir con los plazos establecidos. Como se refleja en el diagrama Gantt, hay bastantes tareas que pueden realizarse en paralelo, dado que no son predecesoras unas de otras. Además teniendo en cuenta que la estimación de tiempo para el proyecto es de 341 horas (Sección 3.4.8), hay tiempo de sobra para llevar a cabo el proyecto.

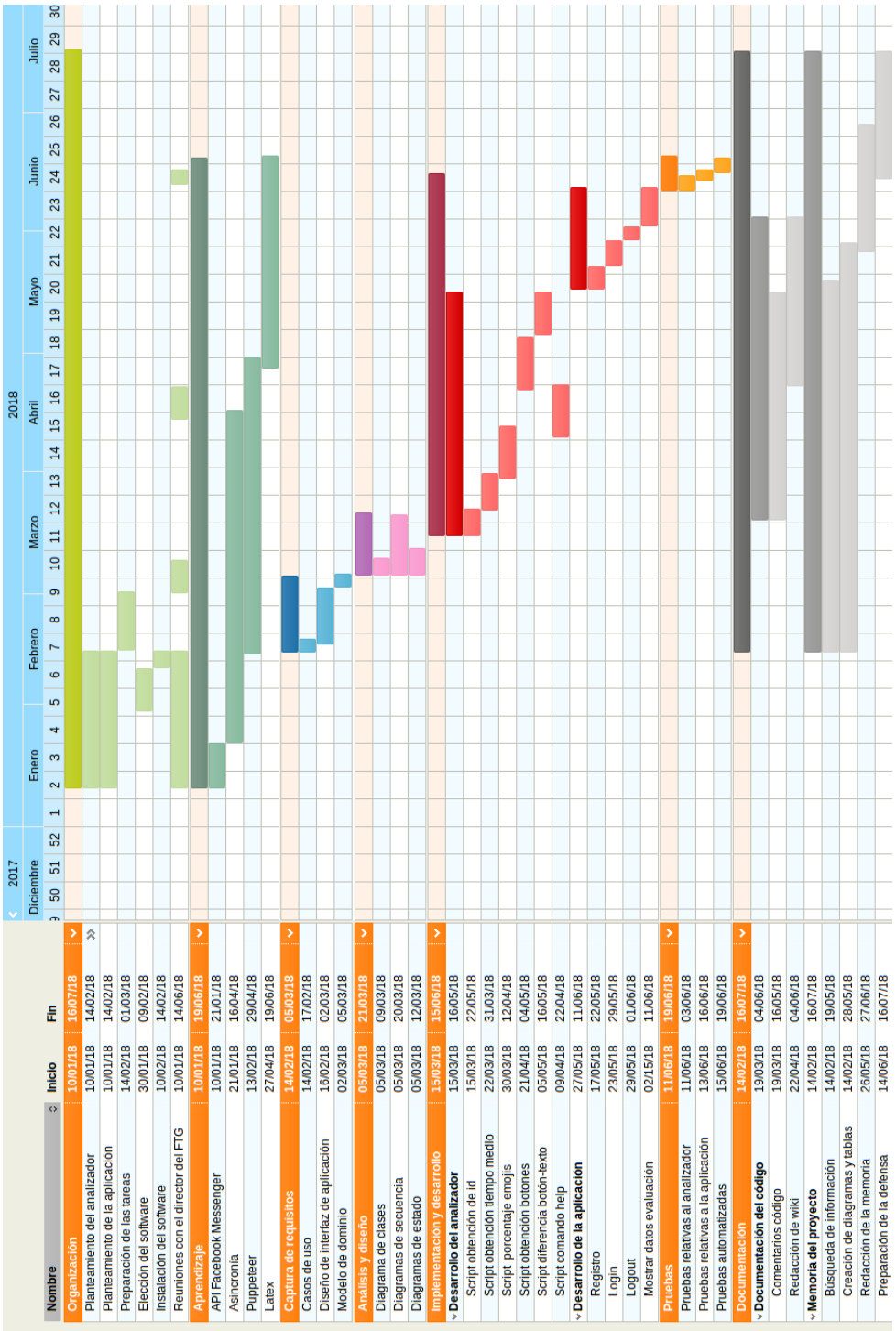

Figura 3.10: Diagrama Gantt de la planificación temporal del proyecto

3.6. Evaluación económica

En esta sección veremos la valoración económica del proyecto, pese a que no se espera comercializarlo, y las diferentes posibilidades de obtención de beneficios.

3.6.1. Mano de obra

En base al mercado laboral actual, vamos a suponer un sueldo bruto para un programador informático de 1500€ mensuales. Con la siguiente ecuación podemos calcular la remuneración por hora trabajada:

$$Sueldo/hora = \frac{Sueldo\ mes}{Semanas\ mes * Días\ trabajo\ semana * Horas\ trabajo\ día}$$

Un mes tiene de media 4 semanas de duración, cada semana reparte 40 horas de trabajo, generalmente en 5 días, a 8 horas cada uno. De la fórmula anterior obtenemos:

$$Sueldo/hora = \frac{1500}{4 * 5 * 8} = \frac{1500}{160} = 9,375$$

Con un total de 9,375 € por hora de trabajo. En la planificación del TFG, se ha estimado que se trabajarán 341 horas, de estas, 40 irán destinadas al aprendizaje del funcionamiento de las herramientas de las que se hará uso. Esas horas se reflejan en formación, y no van a ser consideradas al calcular los costes de mano de obra (aunque se podrían incluir, puesto que es tiempo que se ha invertido en la consecución del trabajo). Ignorando las horas de formación, tenemos:

$$Horas\ totales - Horas\ formación = 341 - 40 = 301h$$

Y multiplicado por el sueldo/hora calculado previamente obtenemos un total de:

$$Sueldo = Horas\ trabajo * Sueldo/hora = 301 * 9,375 = 2821,875$$

El gasto económico relativo a la mano de obra sería de un total de 2821,875€.

3.6.2. *Hardware*

Los costes asociados al *hardware* radican exclusivamente en el ordenador de sobremesa que se va a utilizar, si bien en ocasiones puntuales también se hará uso de un ordenador portátil, su impacto en el proyecto es despreciable y se omitirán los costes asociados a este. Así, tenemos que el ordenador de sobremesa a utilizar es un Innobo Plus, cuyo precio fue de unos 600 euros aproximadamente, a lo que hay que sumar una tarjeta gráfica Nvidia GeForce GTX 650 Ti,cuyo precio fue de unos 110 euros aproximadamente, tanto al ordenador de sobremesa como a la

tarjeta gráfica se les presupone una vida útil de 6 años, la amortización mensual resultante es:

$$Amortización\ mensual = Precio/Vida\ útil\ (en\ meses) = 710/(6 * 12) = 9,86e$$

Con una amortización mensual de 9,86€, y considerando para el TFG una duración de 6 meses, tenemos:

$$Amortización\ total = Amortización\ mensual * total\ meses = 9,86 * 6 = 59,16e$$

Lo cual nos indica que los costes asociados al *hardware* ascenderán a un total de 59,16€

3.6.3. *Software*

La totalidad del *software* que se piensa utilizar es gratuito, desde el sistema operativo (Ubuntu), la plataforma de control de versiones (Github con repositorio público), Overleaf para realizar la documentación, etc. No obstante, sería injusto considerar que en este apartado el gasto es de 0€, pues estaríamos olvidando los gastos asociados al servidor proporcionado por la universidad, en el cual se realizarán las pruebas del analizador y posterior despliegue de una página web que haga uso de este. Por ello, sabiendo que el servidor proporcionado por la universidad pertenece a Digital Ocean, podemos realizar una estimación de los gastos asociados al servidor en el caso de que tuvieramos que hacernos cargo de ellos. Puesto que no se haría un uso intensivo del servidor, nos bastaría con la opción más económica, un único *droplet* (máquina virtual orientada a desempeñar funciones de servidor), con un precio de 5€/mes, durante 6 meses, resultaría en un total de 30€ de gastos asociados al servidor. El gasto en *software* sería de 30€.

3.6.4. Gastos indirectos

Internet

En este apartado en el cálculo de los costes hay que tener en cuenta que en el contrato con la compañia proveedora vienen incluidos otros servicios como pueden ser televisión y telefonía, por lo que se estima un precio de 15€ mensuales por internet, resultando en un total de 90€ al cabo de 6 meses.

Factura eléctrica

Quizás el apartado en el cual es más complejo realizar un cálculo de los costes debido a factores como pueden ser el contrato, precio de la electricidad variable de unas franjas horarias a otras, de un mes a otro, etc. Por simplificar, estimamos

un gasto medio por mes de 10 euros en electricidad, el desarrollo del proyecto va a tomar 6 meses, nos da un total de 60€ en electricidad.

3.6.5. Gasto total

Por último vamos a realizar el cálculo del gasto total que se esperaría para la realización del proyecto si tuviesemos en cuenta los gastos asociados al servidor. En la tabla 3.4 están reflejados todos los gastos analizados anteriormente junto con la suma total. El desarrollo completo del proyecto tiene un coste asociado de 3061,035 euros.

Tipo de gasto	Desembolso asociado (Euros)
Mano de obra	2821,875€
Gasto en *hardware*	59,16€
Gasto en *software*	30€
Gasto indirectos	150€
Internet	90€
Electricidad	60€
Total	3061,035€

Cuadro 3.4: Gastos totales del proyecto

3.6.6. Obtención de beneficios

Si bien el analizador de bots no está pensado para ser comercializado, se pueden considerar las siguientes opciones en caso de que se cambie de idea:

- **Inclusión de anuncios en página web**:Una posibilidad para la obtención de beneficios es la inclusión de anuncios no invasivos, relacionados con el ámbito del software en lugares concretos de la página web. El dinero generado por estos anuncios variará en función de las pulsaciones en estos y las visitas de la página.
- **Funcionalidades restringidas**:Otra opción es limitar ciertas funcionalidades del analizador, ofreciendo a los clientes la posibilidad de pagar por acceder a estas,
- **Soporte a empresas**:Por último, cabe la posibilidad de ofrecer soporte a las empresas que testean sus bots si sus resultados no son satisfactorios, con la posibilidad de llegar a acuerdos para desarrollos de *bots* propios.

Cabe destacar que estas opciones no son mutuamente excluyentes, en caso de considerarse apropiado se pueden aplicar varias o incluso todas ellas.

3.7. Evaluación de riesgos

En esta sección se analizarán los posibles riesgos que puedan surgir durante el transcurso del proyecto, la probabilidad de que se den y su plan de prevención. Se clasificará los riesgos en función de su gravedad y posible perjuicio al desarrollo del proyecto. Consideraremos la jornada laboral de 4 horas al día.

3.7.1. Problemas con el equipo informático

Daños imprevistos, pérdida de trabajo debido a fallo de algún componente o fallo en el suministro eléctrico.

- **Prevención**
  - Uso de herramientas para almacenar los datos en la nube, como pueden ser Github para almacenar el repositorio remoto al que se subirá periódicamente lo desarrollado y Overleaf para la redacción de la documentación.
  - Realizar una revisión periódica del ordenador de sobremesa.
- **Plan de contingencia**
  - En lo referente al *Hardware*, en el caso de que el ordenador principal sufriese algún error fatal, se solventaría el problema si este es simple (fallo de un componente no determinante) o se usaría otro ordenador.
- **Probabilidad e impacto**
  - Probabilidad de un fallo relacionado con el *hardware*: baja: 5 %. El impacto asociado sería bajo debido a disponerse de un ordenador auxiliar.

3.7.2. Enfermedad o lesión determinante

Indisposición temporal por tema de salud, puede estar asociada al trabajo o ser ajena a este. Consideramos lesión determinante aquella que afecta a nuestro correcto desempeño, como pueden ser aquellas lesiones que afectan a las extremidades superiores, los brazos.

- **Prevención**
  - Mantener una buena alimentación, correctos hábitos de sueño y de ejercicio, y una postura y descansos adecuados cuando se está trabajando.
  - Evitar actividades peligrosas, como pueden ser deportes de riesgo.
- **Plan de contingencia**
  - En caso de enfermedad, descansar el tiempo indicado por la revisión del personal sanitario, así como ser consecuente con las medidas prescritas por este.
- **Probabilidad e impacto**
  - Probabilidad de enfermedad: baja: 5 %. Supondremos una duración media de enfermedad de 3 días, eso nos dá un total de 12 horas perdidas, impacto medio.
  - Probabilidad de lesión determinante: muy baja: 0.01 %. Consideramos lesión determinante aquella que imposibilita el uso de una mano para trabajar. Como la duración de estas lesiones suele ser larga, su impacto podrá ser alto o muy alto.

3.7.3. Cambios en APIs o en estructura DOM de Facebook Messenger

La versión 1.2.0 de la API de Puppeteer está planeada para el 12 de abril de 2018. Esto puede llevar a errores en el código implementado. Así mismo, debido al *scrapping* que se realiza sobre el DOM de Facebook Messenger, si cambian el formato de la página es posible que el código se vea comprometido.

- **Prevención**
  - Estar atento a los posibles cambios que puedan realizar sobre la API de Puppeteer.
  - Evitar en la medida de lo posible trabajar con el DOM de Facebook Messenger con selectores muy ambiguos.
- **Plan de contingencia**
  - En caso de cambio en la API o en el DOM de Facebook Messenger, los cambios a realizar en el analizador no deberían ser muy grandes, por lo que con solventar los errores que puedan surgir aumentando las horas de trabajo se debería poder hacer frente a este escenario.

- **Probabilidad e impacto**
  - Probabilidad de cambio en API o en DOM de Facebook Messenger: baja: 5 %. Supondremos que los cambios a realizar deberían tomar un día o dos, eso nos dá un total de 8 horas perdidas, impacto bajo-medio.

Capítulo 4

Análisis de antecedentes

Actualmente hay muy pocas soluciones para medir la calidad de los *chatbots*, y todas ellas están centradas exclusivamente en facilitar la automatización de pruebas. Si los desarrolladores (que son los que se encargan del testeo del *chatbot* que desarrollan) han realizado un planteamiento pobre, las pruebas no detectarán la ausencia de calidad. El resultado será un *bot* que superará las pruebas puesto que estarán orientadas a probar que el funcionamiento desarrollado es correcto, pero con el cual no será agradable e intuitivo mantener una conversación.

Un ejemplo es `TestMyBot`, una librería desarrollada por Florian Treml y Christoph Börner para la automatización de creación de pruebas para *chatbots*.

TestMyBot se basa en Botium, También desarrollado por Florian Treml. Botium es una API para automatizar interacciones básicas con *chatbots*, (envio y recepción de mensajes simples). Actualmente soporta *chatbots* de muchas compañías, Facebook Messenger, Telegram, Skype, Watson, etc.

Con TestMyBot se puede guardar el flujo de conversación de un *bot*, almacenando los mensajes que le son enviados y las respuestas que este devuelve, para posteriormente realizar pruebas que recreen dicho flujo, las cuales fallarán si el *bot* no devuelve las mismas respuestas ante los mensajes que le son enviados.

Similar a la herramienta anterior tenemos la página `http://dimon.co/`, que también enfoca el análisis de *bots* de manera similar, haciendo énfasis en replicar flujos de conversaciones previamente guardadas.

Existen páginas como `http://bottesting.co/` que ofrecen la realización de análisis de la experiencia de usuario, pero dicho análisis se realiza mediante un uso manual del *bot* que se está poniendo a prueba. Puede ser una solución para algunos desarrolladores, pero a nosotros no nos ofrece una perspectiva del proceso de análisis que nos pueda ser de utilidad, pues nuestra intención es automatizar

dicho proceso.

Quizás sea interesante mencionar `facebook-chat-api`[1], una libería que permite emular un navegador para interactuar con *chatbots* en Messenger. Para ello, realiza las mismas peticiones GET y POST que se efectuan en una interacción normal en Messenger. Botium hace uso de esta API para comunicarse en Messenger con los *bots*, pero para nosotros es insuficiente puesto que no es capaz de procesar la información relativa a botones e imágenes, y tampoco es capaz de detectar los botones inferiores del chat cuando estos son mostrados (E.g.: Figura 4.1).

Viendo la ausencia de antecedentes en lo referente a testeo de calidad (y no de funcionamiento), y el estado del arte actual, se hace patente que la consecución del presente proyecto es una gran oportunidad para desarrollar una herramienta novedosa.

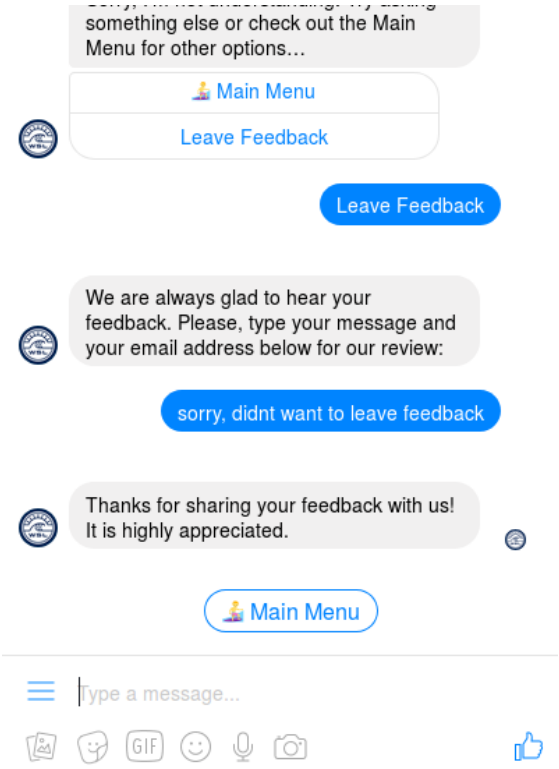

Figura 4.1: Ejemplo de *bot* que hace uso de un botón inferior

[1] `https://github.com/Schmavery/facebook-chat-api`

Capítulo 5

Captura de requisitos

A continuación se detallan los requisitos que debe cumplir el analizador desarrollado durante el proyecto, para satisfacer las necesidades de sus usuarios. Se ha dividido este capítulo en dos partes, una relacionada con el analizador como tal y otra relacionada con la interfaz web que hace uso del analizador. La parte de captura de requisitos de la interfaz web se ha dividido en las siguientes secciones: jerarquía de actores, casos de uso y modelo de dominio. La parte de captura de requisitos del analizador consta de una única sección, relacionada con los objetivos que se pretenden alcanzar.

El analizador consistirá en un script capaz de conectarse a Facebook Messenger y obtener información existente del bot que le sea indicado y capaz de generar información tras interactuar con este, información acerca de la calidad de sus respuestas, tanto en lo referente a información como a expeciencia de uso, a fin de cumplir con los objetivos definidos. (sección 3.1)Dicho script podrá ser lanzado desde la línea de comandos, en cuyo caso se mostrará información a medida que se vaya analizando el bot, para al terminar el análisis mostrarla al completo. El mismo script será el que utilice la interfaz web para realizar los análisis.

5.1. Jerarquía de actores

Debido a la naturaleza de la herramienta, la jerarquía de actores resultante es simple puesto que sólo existe un único perfil, el de usuario.

5.2. Casos de uso

Por la misma razón, los casos de uso de la herramienta son bastante escuetos y se pueden resumir en tres, tal como se puede apreciar en la figura 5.1.

Ejecución análisis

La ejecución del analisis se puede realizar a través de la línea de comandos, indicando como parámetro el nombre o id del bot a analizar.

Ejecución análisis con interfaz visual

De la misma manera se puede iniciar el análisis a través de la línea de comandos pero indicando que se desea tener un *feedback* visual a medida que se va realizando dicho análisis.

Ejecución análisis con guardado en BD

Por último se puede realizar el análisis indicando que se desea que al finalizar este se almacenen los resultados en una tabla en una base de datos.

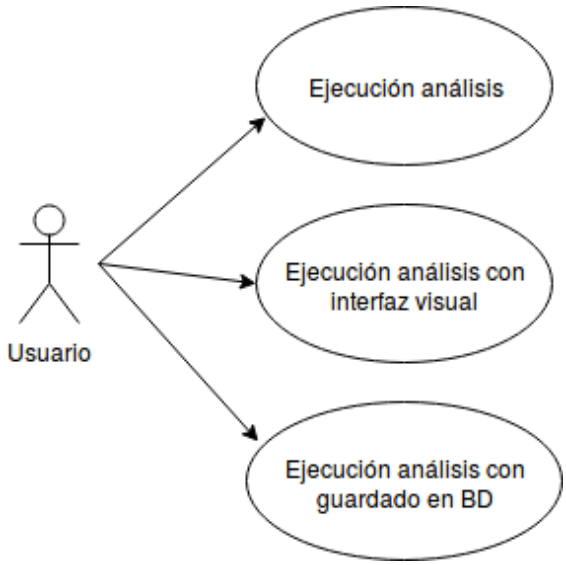

Figura 5.1: Casos de uso de la herramienta desarrollada

5.3. Modelo de dominio

El modelo de dominio es una represencación conceptual de las entidades, relaciones y atributos relacionados con un problema específico. En esta sección, se muestra el modelo de dominio desarrollado durante este proyecto.

A continuación, se procede a realizar una descripción más exhaustiva de cada una de las entidades que aparecen en la figura 5.2:

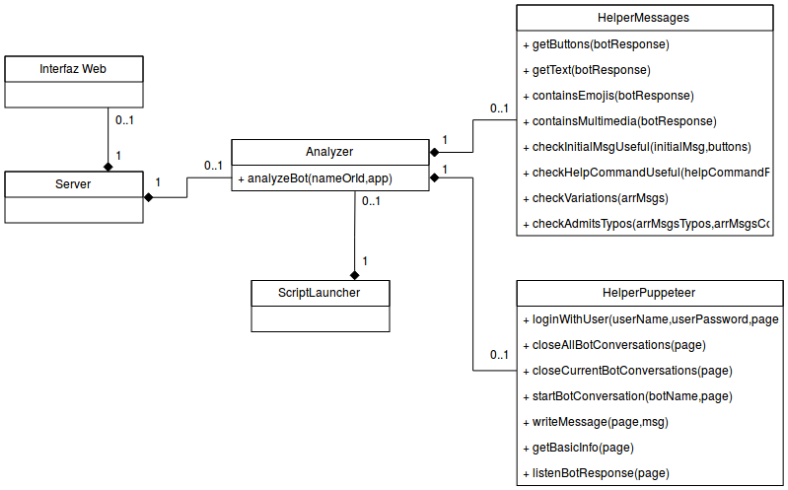

Figura 5.2: Modelo de dominio de la herramienta desarrollada

- **Analyzer**: Es la entidad principal, pues es la que se encarga de realizar el análisis del *chatbot*.
- **HelperPuppeteer**: Es la entidad encargada de realizar las interacciones principales con Messenger a través de puppeteer, tales como son la identificación para acceder a dicha página, el envío de mensajes al *bot* en cuestión, y el procesamiento de las respuestas de este.
- **HelperMessages**: Es la entidad que simplifica la obtención de información de las respuestas obtenidas de un *chatbot*.
- **ScriptLauncher**: Es la entidad que expone al usuario un *entry point* o punto de entrada para poder realizar análisis de *bots* mediante la consola de comandos.
- **Server**: Es la entidad que se encarga de iniciar un análisis y de actualizar una interfaz visual con los resultados del mismo a medida que se van conociendo.

Capítulo 6

Análisis y diseño

Distinguimos las mismas partes que en la captura de requisitos, por un lado vamos a exponer el análisis y el diseño del script que realiza la conexión con Facebook Messenger y realiza el análisis y por otro vamos a presentar el análisis y el diseño de la interfaz web que hace uso del analizador para mostrar la información a medida que se va obteniendo.

6.1. Analizador

A continuación se muestra un esquema del diagrama de flujo que detalla el funcionamiento de la función `analyzeBot`, que realiza el análisis del bot que le es indicado:

6.1.1. División de módulos

El analizador de bots hace uso de librerías externas mencionadas previamente, y de ciertos módulos desarrollados para posibilitar su tarea, el uso de estas librerías y de estos módulos está detallado a continuación:

6.1.2. Bot Listener

Detallamos en profundidad la función encargada de procesar las respuestas del *bot* que se analiza por ser una pieza clave en la consecución de este trabajo. El diagrama de flujo (figura 6.1) muestra los distintos pasos que se siguen al analizar un *bot*. Se realiza la conexión, se obtienen los datos básicos del *chatbot*, se comprueba si responde, y se obtiene información acerca de sus propiedades haciendo uso de heurísticos. Ante el mensaje inicial, se utiliza como heurístico la presencia de botones, la longitud del texto obtenido como respuesta, y la presencia de ciertas

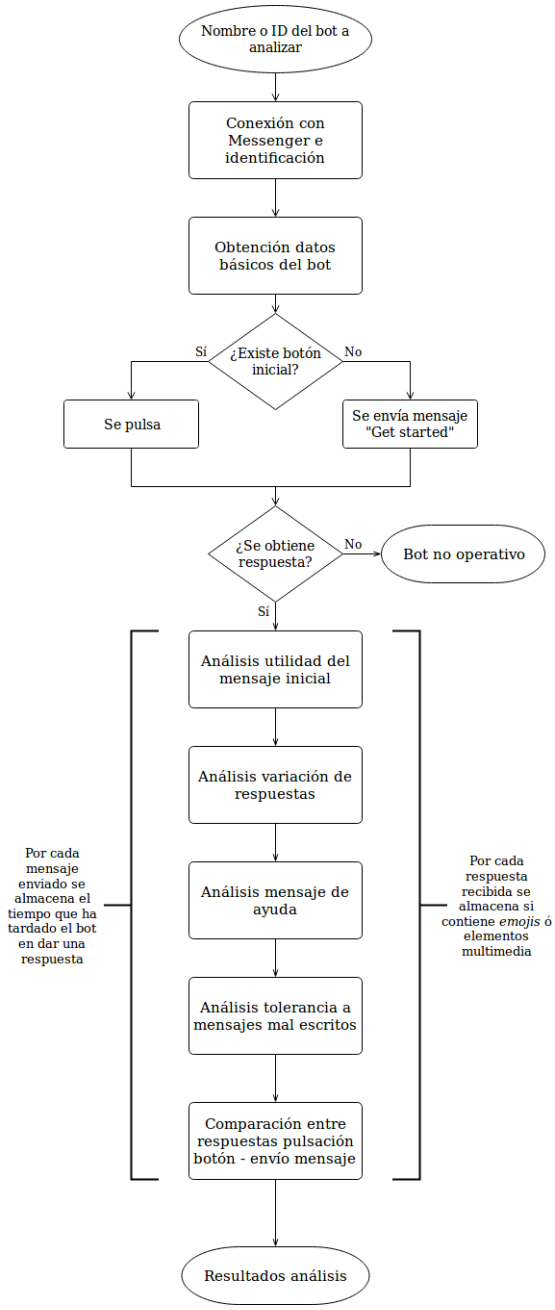

Figura 6.1: Diagrama de flujo del algoritmo de análisis

palabras clave. Con estas características se obtiene la utilidad del mensaje inicial. Ante un mensaje de ayuda, se analizan las mismas características, dado que el heurístico es muy similar. Destacables son los heurísticos usados para analizar si el *chatbot* tolera mensajes mal escritos, y si varía sus respuestas ante mensajes idénticos. Para comprobar si el *bot* tolera mensajes mal escritos, se le envían *utterances* (palabras usuales) escritas correctamente en primer lugar, y mal escritas posteriormente. Las respuestas obtenidas para los mensajes escritos de manera incorrecta son comparadas con la respuesta obtenida para el mensaje escrito correctamente, y en base a coincidencias se determina si el *bot* admite palabras mal escritas o *typos*. Para comprobar si el *bot* es capaz de variar sus respuestas, se le envían las mismas *utterances* o palabras varias veces, y se compueban las respuestas que proporciona ante dichos mensajes entre sí. En caso de coincidencias, se determina que el *chatbot* no varía sus respuestas. Como se puede apreciar en el diagrama de secuencia, también se comprueba si ante la pulsación de un botón y la escritura del mensaje contenido en el mismo se recibe el mismo mensaje, es decir, el *bot* admite y reconoce ambas maneras de recibir un mensaje.

6.2. Interfaz Web

Para proporcionar un *feedback* visual al usuario del analizador, se proporciona también un comando que lanza una interfaz visual, la cual recibe los datos del análisis en curso y se actualiza en consecuencia. Se muestran datos en forma de gráficos relativos a la cantidad de mensajes con *emojis* y con elementos multimedia, así como del tiempo que tarda cada mensaje enviado al chatbot en ser respondido. La interfaz visual puede verse en uso en la figura 6.2

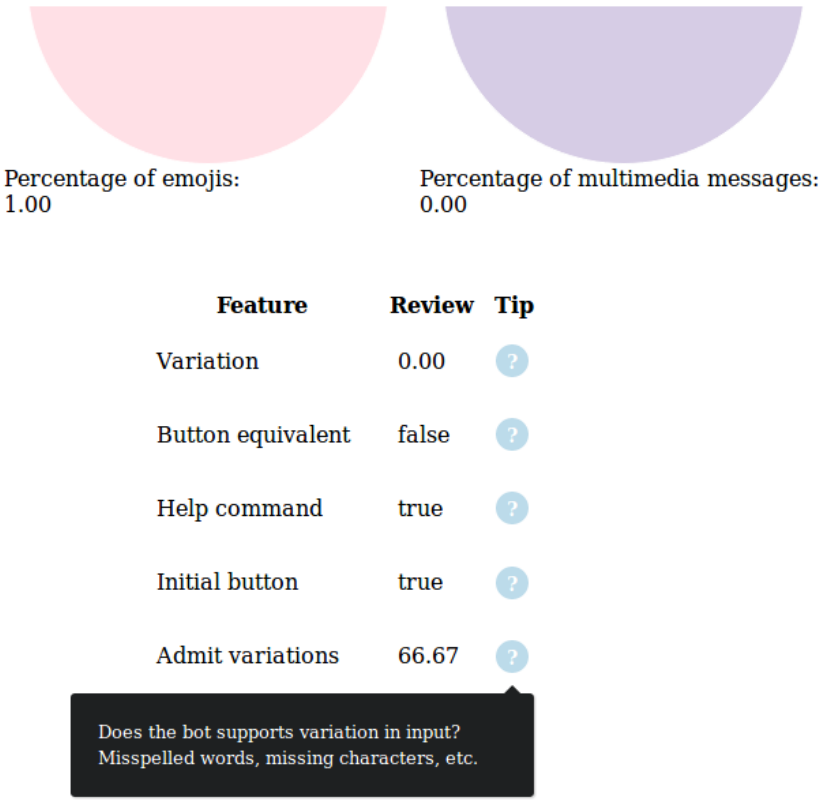

Figura 6.2: Interfaz visual con un análisis terminado

6.2.1. Diagrama de clases

Es una representación en forma de diagrama de la estructura de un sistema. En él, se indican las clases, métodos, atributos y relaciones entre clases. Durante el desarrollo del proyecto se realizaron algunas modificaciones en el diagrama de clases, debido principalmente al planteamiento de la interfaz web como una aplicación web completa, con posibilidad de que los usuarios no tuvieran que entretenerse en ajustes de credenciales en lo relativo a conexión con messenger. Se estudió la opción de usar oauth pero se descartó pues no hubiera solucionado el problema. En la Figura 6.3 se muestra la versión final del diagrama de clases correspondiente al analizador, sus componentes y clases que hacen uso de este.

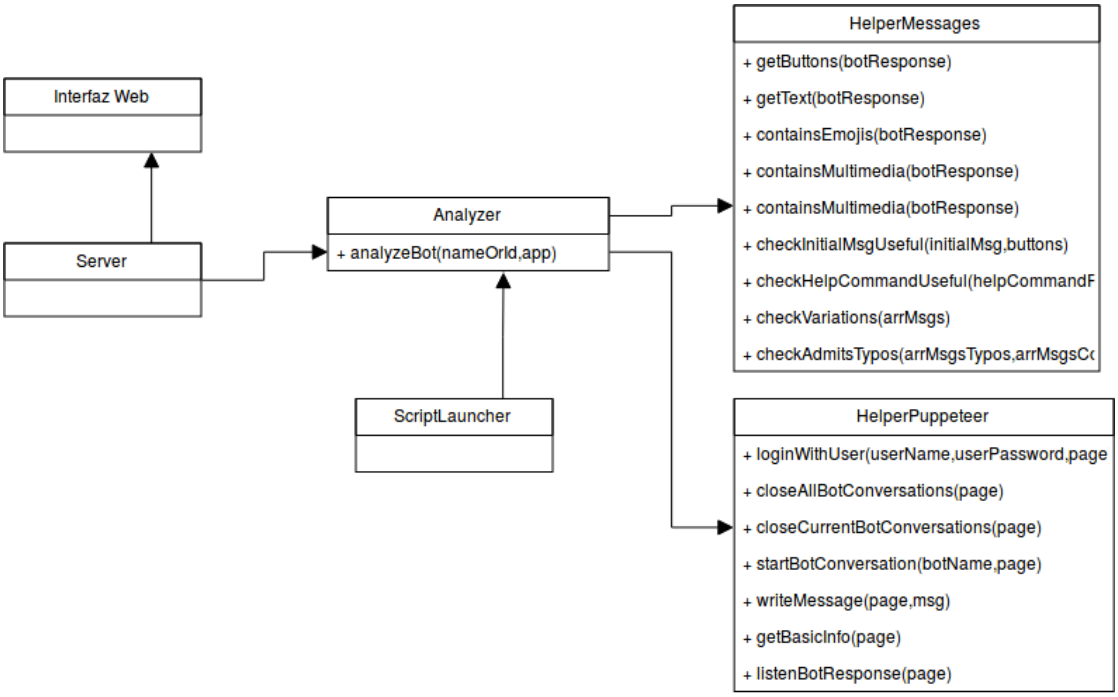

Figura 6.3: Diagrama de clases

- **Analyzer**: Es la clase principal, que contiene la función que se encarga de realizar el análisis completo del *chatbot*, obtención de *likes*, id en Messenger, porcentaje de *emojis* y elementos multimedia usados, tiempo que tarda el *bot* en responder a cada mensaje, si es capaz de procesar los mensajes correspondientes a los botones que presenta de la misma manera que si se hubiese pulsado el botón, si el mensaje inicial y el de ayuda son útiles, y por último si varia sus respuestas y si es capaz de entender mensajes mal escritos.

- **HelperPuppeteer**: Es la clase que realiza interacciones más complejas con Puppeteer, facilitando acciones básicas como pueden ser el *logeo* en Messenger, cerrado de conversaciones, inicio de estas con los bots indicados, y envío y recepción de mensajes.
- **HelperMessages**: Es una clase que contiene funciones de utilidad, usada para la extracción de información del objeto que representa la respuesta de un *bot*, tal como lo procesa la función correspondiente de la clase mencionada anteriormente, HelperPuppeteer. Algunas de sus funciones requieren de información adicional como parámetros, como pueden ser los textos de respuestas procesadas anteriormente asociadas a mensajes sin sentido enviados para conocer qué responde el *bot* que esté siendo analizado.
- **ScriptLauncher**: Es una clase muy simple que actúa como *wrapper* de la función de análisis de *chatbots*, para realizar análisis de *bots* directamente desde la consola de comandos.
- **Server**: Es una clase similar a la anterior, pues también hace las veces de *wrapper* y llama a la función de análisis, pero además de esto abre la interfaz web del análisis y comunica esta interfaz con el análisis en curso, actualizando los datos obtenidos del bot a medida que se van conociendo.

Todas las clases descritas en este apartado han sido creadas durante el desarrollo del proyecto.

6.2.2. Diagramas de secuencia

Los diagramas de secuencia tienen como objetivo representar gráficamente la interacción entre diferentes componentes de un sistema para facilitar y agilizar su comprensión. Para este proyecto, se han creado dos diagramas de secuencia, uno relativo a la función encargada de escuchar las respuestas que ofrecen los *bots* en Messenger, y otro referente a la comunicación entre el analizador y la interfaz web. Se han seleccionado estas interacciones para ser representadas mediante diagramas de secuencia por ser las más relevantes, complejas, y por ello interesantes de analizar.

Así mismo se va a realizar una descripción del flujo que tiene lugar en ambos.

Diagrama de función *listener*

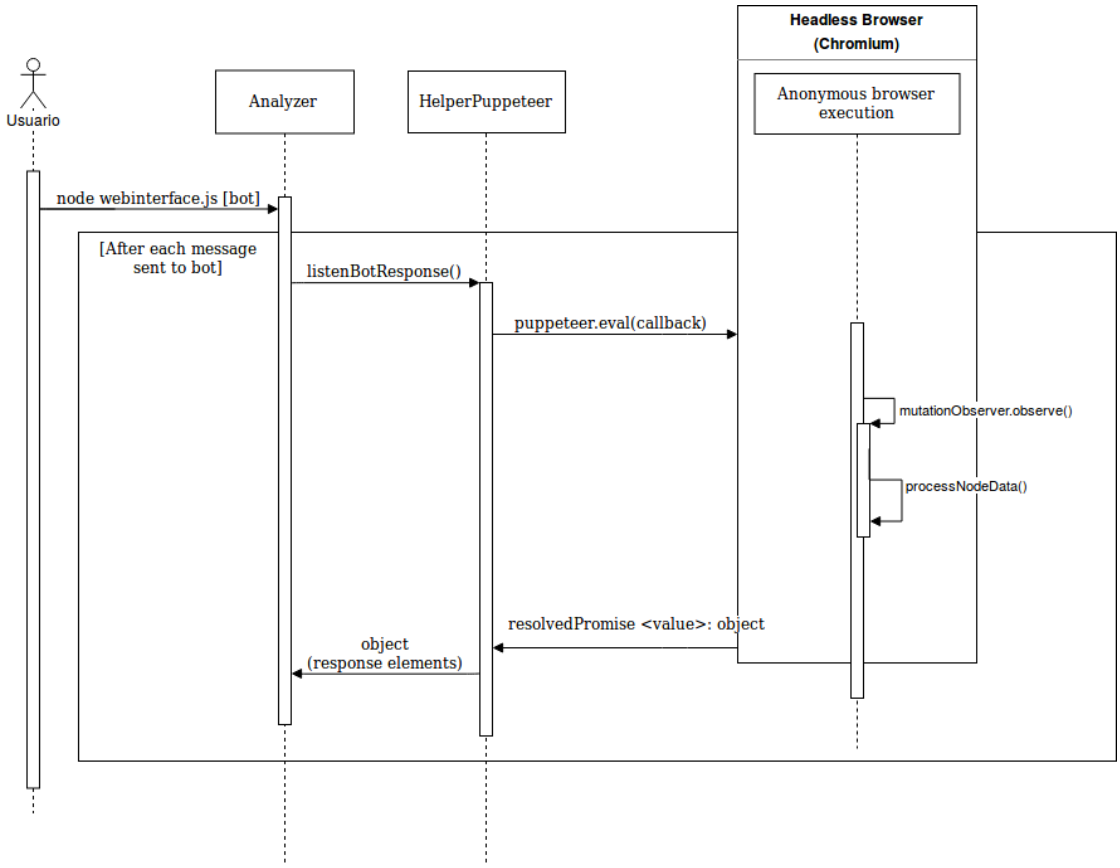

Figura 6.4: Diagrama de secuencia de la función *listener*

En la figura 6.4 se puede observar el diagrama de secuencia de la función

listenBotResponse(), encargada de devolver un objeto con los componentes de una respuesta devuelta por un *bot* en Messenger. Dicha función se encuentra en el módulo helperPuppeteer, y hace uso de una función de la librería puppeteer, eval(), que permite la ejecución de código en el propio navegador *headless* que se está usando por debajo. Gracias a esta función podemos hacer uso de la API MutationObserver, que nos permite detectar cambios en el DOM de la página de Messenger en la que estamos interactuando con el *bot*.

Así mismo, en el callback que es parámetro de la función `eval()` incluimos también tres funciones necesarias para el procesamiento de la respuesta que devuelve el *bot*: `processNodeData()`, `getBottomButtons()` y `getPath()`. Respectivamente, processNodeData analiza el contenido de los mensajes recibidos, y getBottomButtons se encarga de detectar si se muestran botones en la parte inferior de la conversación. Ambas funciones hacen uso de getPath, función de utilidad para obtener la ruta de los elementos que se están procesando. Esto es así debido a que tanto si se da el caso en el que hay botones inferiores como si los botones se muestran en un mensaje, es necesario obtener el path de estos para poder pulsarlos mediante una acción de puppeteer. En processNodeData también se obtiene el texto de los mensajes, si contienen imágenes o elementos multimedia y si contienen emojis.

Diagrama de interfaz web

En la figura 6.5 se puede observar el diagrama de secuencia que representa el flujo que tiene lugar al lanzar el análisis con interfaz gráfica. Cuando se quiere tener una interfaz gráfica, se pone en marcha un servidor, que es el que llama a la función de análisis, que, al ser llamada de esta forma se comunica con este servidor mediante webSockets (flechas rojas en el diagrama). El *script* posteriormente abre una página en navegador con la interfaz web del análisis. Una vez realizado esto, las sucesivas actualizaciones de información del análisis se notifican al servidor, y este las traslada a la interfaz web.

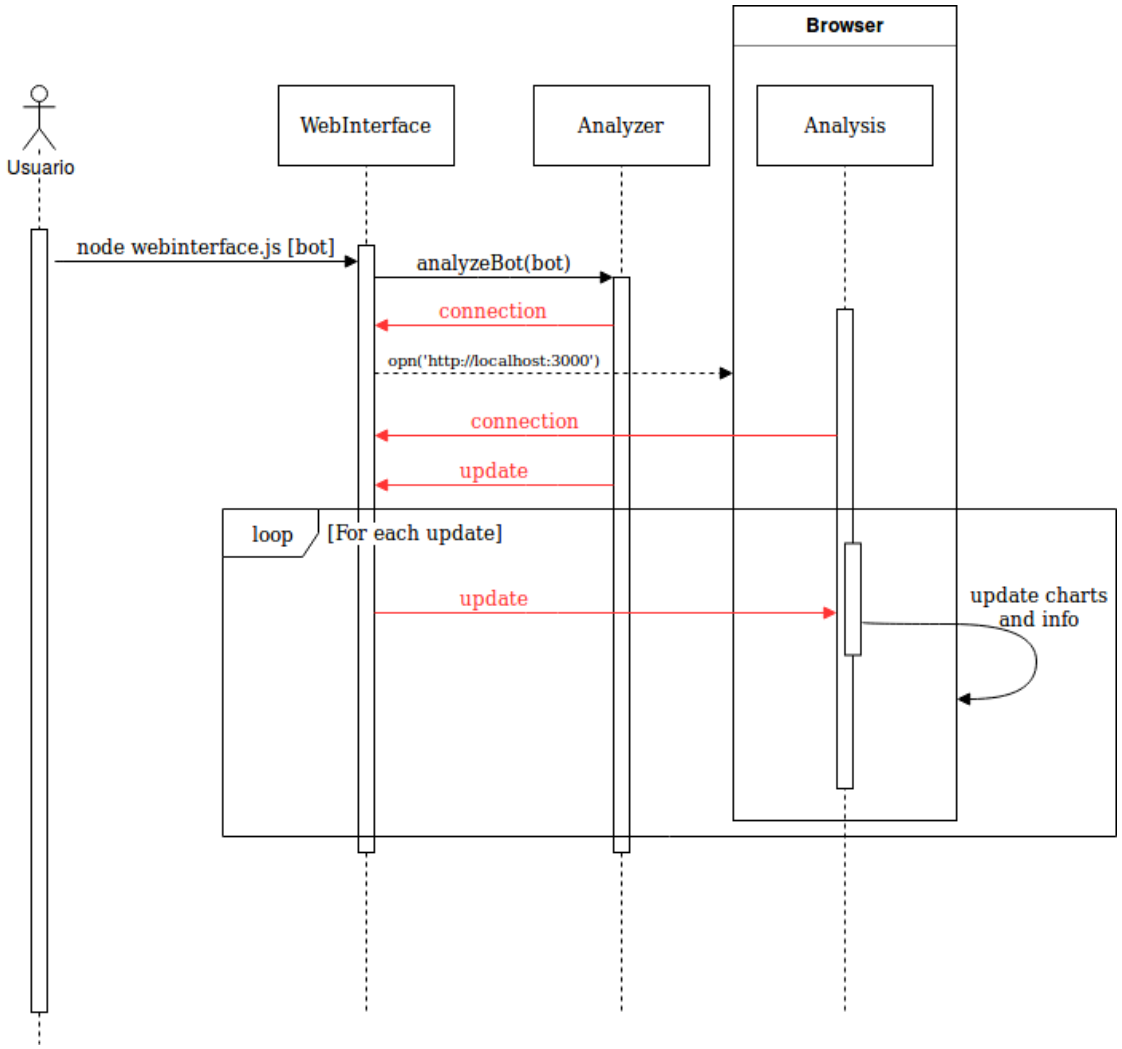

Figura 6.5: Diagrama de secuencia de la conexión del analizador y la interfaz gráfica

Capítulo 7

Elección de lenguajes y tecnologías

La elección de las herramientas correctas es de vital importancia para el desarrollo de un proyecto. Una mala selección de estas puede afectar de manera severa al tiempo dedicado al proyecto, conllevando por ello también pérdidas monetarias y en última instancia el colapso de una empresa.

Al tratarse de un proyecto en el que la totalidad del producto final implica la implementación de código, es verdaderamente importante hacer una selección de los lenguajes de programación, librerías y programas a utilizar. La correcta elección de estos puede ahorrar una gran cantidad problemas y horas de trabajo. Es por ello que en este proyecto se plantearon varios lenguajes con los que trabajar. En esta sección se muestran los planteamientos que se realizaron a la hora de escoger las herramientas utilizadas.

7.1. API control navegador *headless*

Para interaccionar con Facebook Messenger, se sopesaron 2 opciones, Puppeteer y Selenium, dos de las más famosas APIs de automatización de navegadores.

Los puntos a favor de Selenium eran la compatibilidad con múltiples navegadores y el tiempo que tiene de desarrollo detrás, que asegura estabilidad y robustez, pero teniendo en cuenta que no necesitábamos realizar tests en varios navegadores, sino simplemente ayudarnos de uno para obtener información de la interacción con *bots*, nos decantamos por Puppeteer, puesto que ofrece un control mayor sobre el navegador al estar desarrollado por Google y hacer uso del navegador Chromium.

7.2. Elección del lenguaje de programación

Cuando hablando con Juanan Pereira sobre este proyecto, salió el tema del lenguaje de programación que se podría usar, me comentó que él trabajaba con PHP, Python y Node.js, y que considerando la naturaleza del trabajo y algunos de mis trabajos realizados anteriormente en la asignatura de DAWE, Node.js podría ser una excelente elección. Otros aliciente fueron la naturaleza asíncrona del proyecto, a la cual Node.js se adapta perfectamente, y el uso de javascript como lenguaje de programación, que se usaría también para la interfaz web del analizador.

Para la parte de *scrapping* de la página `chatbottle`, se eligió python por su sintaxis sobria y simple. Como paquetes adicionales se escogieron `beautifulSoup` y `MySQLdb`, para poder trabajar con el HTML de la página y guardar los resultados en una base de datos respectivamente. También se usó el módulo urllib2 para establecer y gestionar la conexión con la página.

7.3. Elección de la librería de gráficos

Para la elección de la librería de gráficos para representar los datos del análisis en la interfaz web, se decidió primar la simplicidad, tanto visual como estructural, para no complicar en exceso la inserción de los datos del análisis. Se consideraron varias opciones, `C3.js`, `tauchart`, `chart.js` y `chartist.js`. Se eligió `chart.js` por su sencillez, puesto que no necesitábamos realizar gráficos complejos, sino sobrios en cuanto a contenido.

Las otras opciones también podrían haber sido aceptables, y en lo referente a documentación todas las opciones analizadas tienen una gran cantidad ejemplos y muy útiles.

7.4. Comunicación servidor

Para comunicar el servidor con el analizador, y el servidor con la interfaz web, se eligió `socket.io`, pues permite implementar comunicación por websockets de una manera muy sencilla y rápida, y altamente eficiente. Por la necesidad de informar desde el analizador a la interfaz web en tiempo real de los resultados del análisis mientras se están obteniendo, era necesario hacer uso de websockets u otra tecnología similar, y dado que ya había trabajado con la librería socket.io, decidí volver a usarla para simplificar este proceso.

Capítulo 8

Desarrollo

En este capítulo se detallan los pasos dados a la hora de desarrollar el analizador y la aplicación web que hace uso de este. Así mismo, se indican los problemas encontrados, y las soluciones estudiadas y las adoptadas ante ellos. De entre los problemas generales, hay que destacar los relacionados con la plataforma de Facebook Messenger, pues son los que más han afectado al desarrollo del proyecto, como por ejemplo el bloqueo de las cuentas que se usan para realizar la conexión a Messenger y la restricción de no poder abrir más de un número determinado de bots en un mismo día.

8.1. *Necesidades del cliente*

De manera introductoria, antes de analizar cuál fue el proceso que se siguió durante el desarrollo, se han de explicar las necesidades que el director y cliente, Juanan Pereira, identificó cuando se se realizó el diseño del proyecto.

8.1.1. Analizador

El analizador de *bots* de Messenger recaba información de los *chatbots* realizando una interacción constante con estos. La información que debe obtener debe ser, por lo menos:

- **análisis del mensaje inicial**: Se debe recibir información acerca de si el mensaje inicial es de utilidad a los usuario o no.
- **análisis de botón inicial**: Se indicará si el *bot* posee un botón inicial que facilite la interacción con este, y que ayuda a saber que se va a proceder a interactuar con un *chatbot*.

- **análisis del mensaje de ayuda**: Se debe obtener información acerca de si ante una petición de ayuda, se reconoce esta acción y el mensaje de ayuda que se proporciona es de utilidad a los usuarios.
- **análisis del uso de *emojis***: Se debe analizar el porcentaje estimado de *emojis* que son usados por el *chatbot* en sus interacciones con a los usuarios.
- **análisis del uso de elementos multimedia**: De la misma manera, se debe analizar el porcentaje estimado de elementos multimedia que son usados por el *chatbot* en sus interacciones con a los usuarios. Entre estos elementos se incluyen imágenes, audios y vídeos.
- **Comparativa entre botón y texto**: Se analizará si se obtiene el mismo resultado ante la pulsación de un botón y el envío del mensaje asociado al botón, en *bots* que estén correctamente desarrollados ambas interacciones se procesarán de la misma manera, dando lugar a una conversación coherente.
- **Análisis del tiempo medio de respuestas**: Se analizará el tiempo medio que tardan los *bots* en presentar una respuesta al usuario ante sus mensajes.

8.1.2. Interfaz visual

Para poder tener un *feedback* progresivo se consideró oportuno el desarrollo de una interfaz visual, que deberá actualizarse a medida que reciba la información procedente del analizador.

8.2. *Listener*

La función de *listener* que procesa las respuestas de los bots para obtener un objeto con sus elementos es probablemente de las que más problemas han dado. En momentos puntuales se llegó a considerar y estudiar reescribir el código desarrollado para Puppeteer en Selenium.

Las características del *scrapping* que se tiene que realizar son complejas (los nodos que se corresponden con los mensajes comparten los mismos atributos y clases, por lo que no es posible de manera sencilla obtener el último mensaje recibido). Para solventar esto, se ha hecho uso de la API MutationObserver, que se encarga de notificar de los cambios observados en el nodo del DOM que le sea indicado.

Con la función evaluate de puppeteer, perteneciente a la clase page (utiliza su contexto), ejecutamos en el navegador la función encargada de procesar los mensajes recibidos del bot, en la cual se hace uso de MutationObserver para obtener los

nodos añadidos (mensajes nuevos), procesarlos y así tener la información necesaria de sus elementos (texto, imágenes, botones, etc).

Tras esta primera aproximación, se descubrió que algunos mensajes se procesaban correctamente (se obtenía la información asociada a ellos), pero otros no. Este problema fue un verdadero quebradero de cabeza, puesto que no había nada que indicase un error o un fallo en el procesamiento de los mensajes. Hasta que tras muchas vueltas se descubrió que al enviar un mismo mensaje a un bot (y con ello recibir la misma respuesta dos veces), la segunda respuesta obtenida sí que se procesaba correctamente.

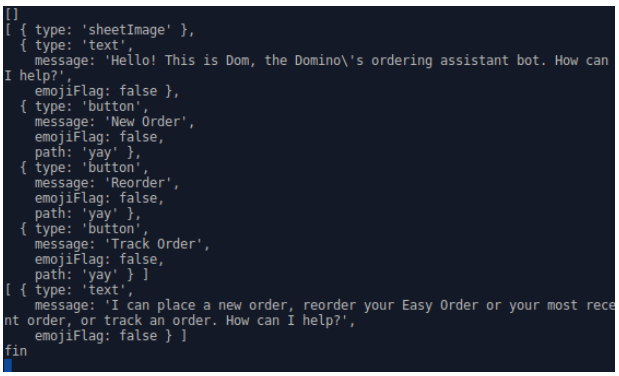

Figura 8.1: Información obtenida ante mensajes *Get started - Get started - Help*

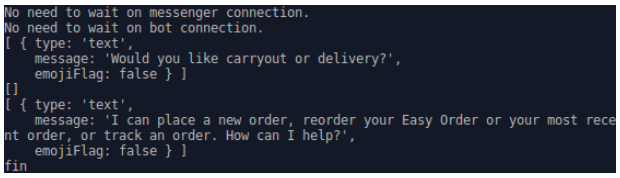

Figura 8.2: Información obtenida ante mensajes *randomMsg - Get started - Help*

Esto se debe al pseudo elemento ::after, y a su funcionamiento. Aunque a simple vista todo el nodo se carga de manera homogénea, lo cierto es que con este pseudo elemento se carga información en él, haciendo que si se obtiene el nuevo nodo en el instante que es añadido, no se tiene todo su contenido, sino simplemente su HTML correspondiente más externo.

Para solventar esto, simplemente almacenábamos en una variable el selector del nuevo nodo añadido, para al cabo de un breve intervalo de tiempo (medio segundo) realizar la selección de este. De esta forma, obteníamos el nodo completo, con todos sus nodos descendientes, hasta los añadidos en el pseudo elemento ::after.

8.3. *Script analizador*

A la hora de establecer una conexión con Messenger, se estudiaron dos opciones. Por un lado realizar la conexión manualmente, con un navegador en modo *headless* controlado por alguna API como Selenium o Puppeteer, y por otro hacer uso de una librería ya existente, facebook-chat-api, para no tener que reinventar la rueda y poder simplificar la tarea de interaccionar con Messenger y con bots en la aplicación.

Al final nos decantamos por la primera opción, pues la segunda, si bien era más sencilla y simple, tenía una gran desventaja, que era que simplemente procesaba el texto de los mensajes recibidos. A la hora de recibir botones no se obtenía ninguna información de la existencia de estos, y eso suponía una gran limitación para el desarrollo del proyecto. Tampoco se obtenía información de elementos multimedia.

Por esto fue necesario realizar la conexión a través de un navegador *headless* y las interacciones con este mediante una API. Nos decantamos por Puppeteer por el control que es capaz de ofrecer.

8.4. *Script* obtención lista *bots* populares

En un primer momento se quiso automatizar el proceso de análisis de *bots* hasta el punto de lanzar el analizador para todos los *bots* de la página *chatBottle* mediante un *script*, posteriormente, se decidió que simplemente nos ayudaríamos de esta página para obtener los *likes* de todos los *bots* registrados en ella para así poder ordenarlos por dicha métrica y centrarnos en los más populares (los que más relevancia tienen), para así analizar su calidad en lo referente a experiencia de uso. El *script* usado para obtener la lista de *bots* junto con sus *likes* está detallado en el anexo B.

8.5. *Wiki* realizada

Para facilitar la modificación del trabajo realizado, se ha realizado una pequeña *wiki* en GitHub. En dicha *wiki* también se hace mención al proceso de instalación y configuración, así como los comandos para la utilización del analizador, con y sin interfaz gráfica. `https://github.com/petuscov/puppeteerFacebook/wiki`

8.6. Contribución a proyecto *testMyBot*

Como parte del análisis del estado del arte, y motivado por Juanan, se usó la librería testMyBot para realizar análisis de *bots*, al encontrarse fallos en dicha librería, se procedió a buscar una solución a estos, corregirlos y realizar un *pull request* en el repositorio de la librería para que las correcciones estén disponibles a todo el mundo. El contribuir a un proyecto ajeno a través de la plataforma GitHub es una enriquecedora experiencia, los cambios realizados se detallan en el anexo B.

Capítulo 9

Pruebas

En este capítulo se detallan las pruebas realizadas para comprobar el correcto funcionamiento del analizador. Es importante señalar que por la naturaleza del proyecto, estas pruebas no pueden ser automatizadas. Las pruebas se han centrado en la función principal del analizador por una parte (la que se encarga de procesar las respuestas dadas por el bot) y del analizador como un todo por otra parte.

9.1. *Listener* de bots

Para comprobar el correcto funcionamiento del analizador, se comprobaba si el objeto con los elementos del mensaje o mensajes analizados por la función de *listener* de las respuestas del bot coincidía con su representación en el navegador. Se hacía especial énfasis en aquellos mensajes devueltos por bots que tenían alguna particularidad, botones, plantillas específicas, etc.

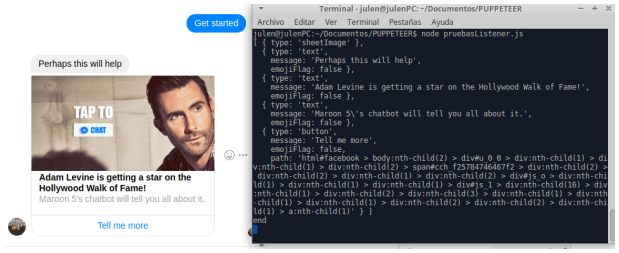

Figura 9.1: Comparación entre respuesta en Messenger y objeto resultante del procesamiento de dicha respuesta

9.2. Pruebas analizador

Para comprobar el correcto funcionamiento del analizador, se ha usado para analizar los 15 bots más populares de Facebook Messenger, así como con algunos adicionales que se han considerado relevantes por ciertas características que poseían. Previamente, antes de realizar los análisis haciendo uso del analizador, interaccionamos desde Messenger con el *bot* que se va a analizar. calculamos el porcentaje aproximado de mensajes con *emojis* y con elementos multimedia (imágenes, vídeos, audios), consideramos la utilidad tanto del mensaje inicial como de la respuesta dada ante el comando/mensaje *help*, variación de respuestas ante un mismo mensaje y si es capaz de entender mensajes mal escritos. Tras este análisis manual, lanzamos el analizador sobre el mismo *bot*, y comparamos los resultados obtenidos con los de nuestro propio análisis.

Nombre	Responde	Posee botón inicial	Mensaje inicial es útil	Posee mensaje de ayuda y útil	Equivalencia botón - texto	Variación en respuestas	Reconocimiento de *typos*	Emojis	Multimedia	Tiempo medio de respuesta	*Likes*
cnn	Sí	Sí	Sí	Sí	No	33 %	100 %	5/1	0/6	2.80	30M
50cent	No	-	-	-	-	-	-	-	-	-	30M
victoriassecret	Sí	Sí	Sí	Sí	Sí	0 %	66 %	5/1	1/5	1.47	29M
maroon5	Sí	Sí	Sí	Sí	No	0 %	100 %	5/1	0/6	0.93	38M
dominos	Sí	No	Sí	Sí	Sí	66 %	100 %	5/1	0/6	1.44	19M
cocacolanorway	No	-	-	-	-	-	-	-	-	-	107M
McDonaldsPK	Sí	Sí	Sí	Sí	No	66 %	66 %	2/4	0/6	2.06	77M
katyperry	Sí	Sí	No	Sí	Sí	0 %	100 %	5/1	4/2	2.65	68M
vonvon.me	No	-	-	-	-	-	-	-	-	-	63M
DavidGuetta	Sí	Sí	Sí	Sí	Sí	33 %	100 %	5/1	0/6	2.80	52M
HuaweimobileNO	Sí	No	No	No	No buttons	0 %	66 %	0/6	0/6	6.5	51M
KFC	No	-	-	-	-	-	-	-	-	-	51M
netflixfrance	Sí	Sí	Sí	Sí	No	66 %	66 %	2/4	0/6	2.06	47M
nba	Sí	Sí	No	Sí	Sí	0 %	66 %	0/6	0/6	1.48	34M
lorealparisnorge	No	-	-	-	-	-	-	-	-	-	34M
hmdeutschland	No	-	-	-	-	-	-	-	-	-	33M
transformersmovie	Sí	Sí	Sí	Sí	No	33 %	33 %	0/6	0/6	1.59	32M
NutellaChile	No	-	-	-	-	-	-	-	-	-	32M
adidasoriginals	No	-	-	-	-	-	-	-	-	-	32M
hiponcho	Sí	Sí	No	No	No buttons	0 %	100 %	0/6	0/6	0.93	15k

Cuadro 9.1: Análisis de calidad de los *chatbots* con mayor popularidad en Facebook Messenger actualmente

De estos análisis hay que comentar aspectos reseñables en uno de ellos. Llama la atención el tiempo tan elevado que tiene Huawei de respuesta. El analizador espera un tiempo máximo de 7 segundos antes de dejar de 'escuchar' las respuestas de un *bot* ante un mensaje, y que huawei se acerque tanto a esa cifra no es un buen síntoma. Al comprobar el flujo de la conversación en Messenger, podemos observar que el *bot* responde a mensajes sueltos de manera pseudoaleatoria, por lo que es muy probable que el servidor en el que está alojado está recibiendo muchas peticiones, o hay un diseño erróneo en la arquitectura del *bot*.

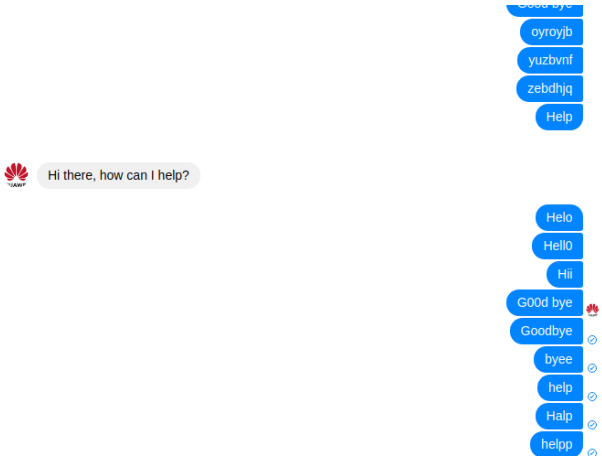

Figura 9.2: Flujo de conversación con *bot* Huawei

Capítulo 10

Conclusiones

En este último capítulo, se procede a efectuar un análisis entre la planificación original, realizada al comienzo del proyecto y la gestión del tiempo final. También se enumeran y describen lineas futuras e ideas que pueden ampliar el proyecto. Se hace un breve repaso de las licencias utilizadas y, finalmente, una pequeña reflexión personal.

10.1. Análisis entre planificación estimada y real

Si bien se han alcanzado todos los objetivos planteados al comienzo del proyecto, algunos de estos han variado respecto de su planteamiento inicial. Esto se debe a que este planteamiento inicial era demasiado ambicioso para la herramienta que se ha realizado. La gestión del tiempo y la planificación han tenido ligeras deficiencias, debido principalmente al tiempo que se ha podido dedicar a este. Por esta misma razón, algunos objetivos secundarios no se han podido alcanzar.

10.1.1. Objetivos

Al comienzo del proyecto, se planteó entre los objetivos el desarrollo de una aplicación web, con posibilidad de registro de usuarios, almacenamiento de análisis, etc. Esto no pudo ser llevado a cabo, puesto que por la naturaleza del analizador es necesario hacer uso de una cuenta de Messenger. Si dicha cuenta la provee la propia aplicación, los usuarios de esta aplicación estarían compartiendo dicha cuenta para realizar sus análisis, Messenger detectaría muchas interacciones de dicha cuenta con *bots* y la bloquearía. Por el funcionamiento del analizador, no se puede realizar una autentificación a través de oauth para posteriormente hacer uso de las cuentas con las que se identifiquen los usuarios, oauth está pensado para efectuar una identificación de usuarios, no para hacer uso de ellos.

Por todo esto este objetivo se modificó para conformar la interfaz web que se ha desarrollado finalmente. Esto hace que los usuarios deban escribir sus credenciales de Messenger en un archivo de configuración, de manera similar a como trabaja testMyBot.

10.1.2. Alcance

Por lo mencionado en el punto anterior, el alcance se ha visto afectado. Si bien los bloques de tareas principales definidos en el diagrama WBS general no se han visto afectados, dentro de algunos de estos bloques si que se han experimentado modificaciones.

Implementación y desarrollo

Como se ha mencionado anteriormente, uno de los objetivos inciales (aplicación web) varió respecto de su planteamiento original, por lo que las tareas de implementación y desarrollo asociadas a este variaron enormemente.

Así, ahora tenemos que la aplicación tal como se planteó en un principio ha mutado en una interfaz gráfica que hace uso del analizador.

Interfaz web
Bloque de trabajo: Implementación y desarrollo - Interfaz web **Duración final**:12
Descripción: Interfaz web para mostrar los resultados del analizador, y conexiones entre estos componentes y servidor simple.*Script* encargado de obtener el ID que Facebook Messenger ha asignado al *bot* correspondiente.
Salidas/Entregables: Archivo Javascript capaz de obtener el ID del *bot*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

Server
Bloque de trabajo: Implementación y desarrollo - Interfaz web **Duración estimada**:4
Descripción: Servidor encargado de abrir una página en el navegador con la interfaz web, y de comunicar dicha interfaz con el análisis en curso.
Salidas/Entregables: Archivo Javascript relativo al servidor.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Puppeteer API, MySQL.

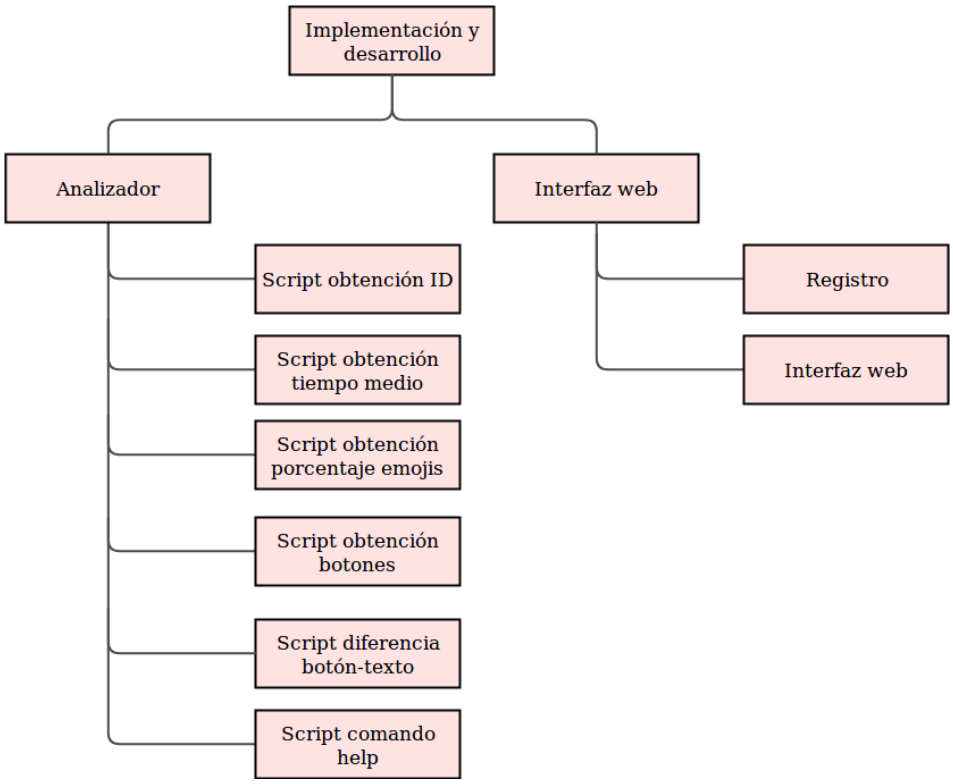

Figura 10.1: Diagrama WBS final del bloque de implementación y desarrollo

Pruebas

La creación de pruebas automatizadas no es posible por la naturaleza de la herramienta desarrollada, y la interfaz web no necesita de pruebas, por lo que no aparecen reflejadas en el diagrama gantt final. En su lugar, se han añadido las pruebas relativas al *listener* de messenger, pues es una parte muy sensible e importante del analizador.

Pruebas relativas a función *listener*
Bloque de trabajo: Pruebas **Duración estimada**:20
Descripción: Se ejecuta la función *listener* en un *bot* tras enviarle un mensaje, para comprobar de manera manual si el objeto resultante del procesado de la respuesta obtenida de este coincide con la representación visual de la respuesta.
Salidas/Entregables: Documentación relativa a los resultados de las pruebas.*web*.
Recursos necesarios: Ordenador conectado a internet, Sublime Text 2, Nodejs, Mozilla Firefox, Google Chrome.

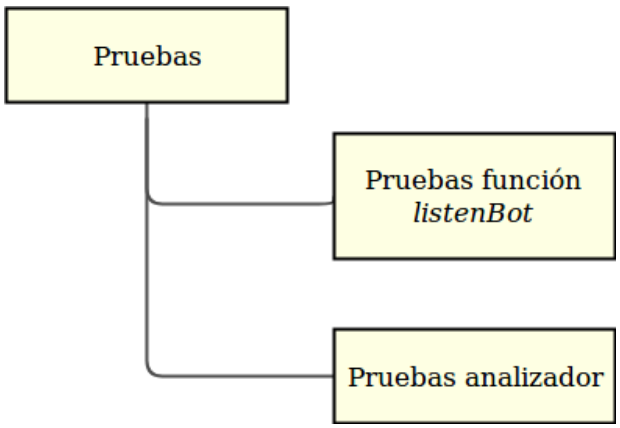

Figura 10.2: Diagrama WBS final del bloque de pruebas

Duración de las tareas

Los cambios que ha experimentado el proyecto desde su planificación estimada originalmente y su desarrollo real afectan al diagrama gantt, y la diferencia de tiempo en las respectivas tareas se puede apreciar en las tablas 10.1, 10.2, y 10.3. El diagrama gantt final puede verse en la figura 10.3

ID	Nombre tarea	Tiempo estimado	Tiempo real
	Organización	24	24
1	Planteamiento del analizador	3	3
2	Planteamiento de la aplicación	1	1
3	Preparación de las tareas	6	6
4	Elección del software	1	1
5	Instalación del software	3	2
6	Reuniones con el director del TFG	10	10
	Aprendizaje	40	35
7	Asincronía	10	5
8	Puppeteer	15	15
9	Express	2	2
10	Latex	13	13
	Captura de requisitos	25	20
11	Casos de uso	4	4
12	Diseño de interfaz de aplicación	17	12
13	Modelo de dominio	4	4
	Análisis y diseño	15	13
14	Diagrama de clases	4	4
15	Diagrama de secuencia	8	6
16	Diagramas de estado	3	3

Cuadro 10.1: Comparativa entre tiempo estimado y final de los 4 primeros grupos de tareas

ID	Nombre tarea	Tiempo estimado	Tiempo real
	Implementación y desarrollo	104	100
	Desarrollo del analizador	48	44
17	Script obtención de ID	4	4
18	Script obtención tiempo medio respuesta	6	6
19	Script obtención porcentaje de *emojis*	6	6
20	Script obtención botones	10	10
21	Script diferencia botón-texto	14	10
22	Script respuesta help	8	8
	Desarrollo de la interfaz web	56	60
23	Interfaz web	20	22
24	Login	13	-
25	Logout	13	-
26	Realizar evaluación - Mostrar datos	20	22

Cuadro 10.2: Comparativa entre tiempo estimado y final correspondiente a implementación y desarrollo

ID	Nombre tarea	Tiempo estimado	Tiempo real
	Pruebas	50	50
27	Pruebas relativas al analizador	20	25
28	Pruebas función listenBot	30	25
	Documentación	83	83
	Documentación código	10	10
29	Comentarios código	5	5
30	Redacción de Wiki	5	4
	Memoria del proyecto	73	80
31	Búsqueda de información	4	5
32	Creación de diagramas y tablas	4	7
33	Redacción de la memoria	45	48
34	Preparación de la defensa	20	20
	TOTAL	341	340

Cuadro 10.3: Comparativa entre tiempo estimado y final correspondiente a documentación y pruebas

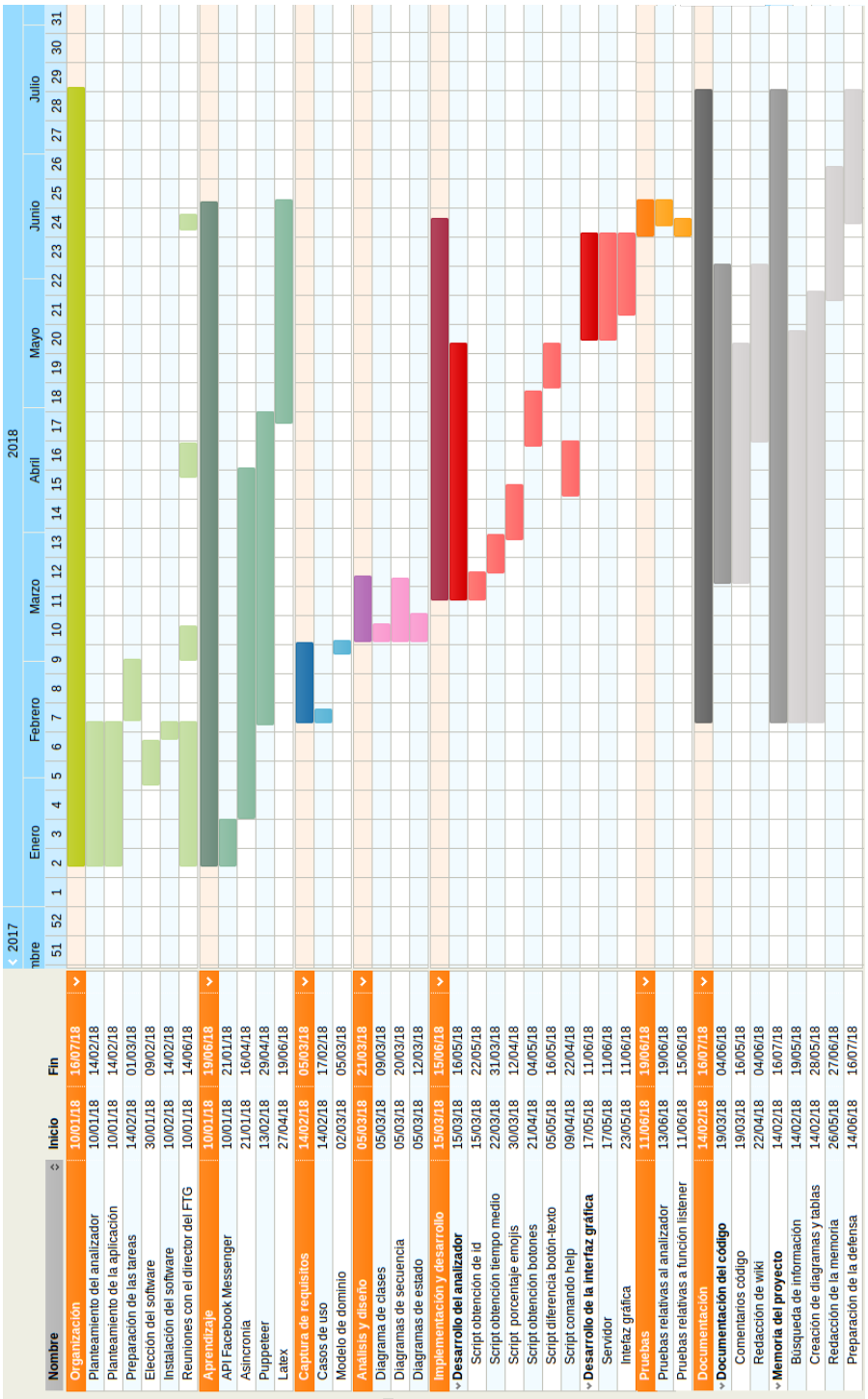

Figura 10.3: Diagrama Gantt final del bloque de pruebas

10.2. Lineas futuras

Por la naturaleza de este proyecto y la inexistencia de trabajos o aplicaciones similares, se ha pretendido que lo realizado sirva de base para futuros desarrollos y análisis.

Los heurísticos utilizados en las funciones de análisis son susceptibles de ser modificados, y se pueden crear nuevas funciones de análisis partiendo del *listener* principal del analizador. A continuación se enumeran algunas líneas a considerar para proyectos futuros:

- **Identificación de procesamiento de audios**: Si bien es una característica que muy pocos bots pueden soportar en el estado del arte actual, es muy probable que en la próxima década se realicen esfuerzos en soporte de NPL con audios en *bots*, y que vayan apareciendo bots que admitan este tipo de mensajes. Para analizar si un bot es capaz de admitir y 'entender' audios, una primera aproximación podría ser enviarle un "*Hello*", y comparar la respuesta recibida con la que se obtiene ante un *Hello* escrito y con la que se obtiene ante mensajes aleatorios (sin significado).

- **Identificación de imágenes**: De manera muy similar al punto anterior, se puede intentar analizar que el *bot* que se está analizando es capaz de reconocer que ha recibido una imagen como mensaje, y ofrecer una respuesta acorde a esto, en lugar del típico mensaje que usar cuando no sabe que otra cosa decir; "no te entiendo".

  Tanto este punto como el anterior tienen una implementación algo compleja, puesto que se requiere enviar archivos a Messenger, automatizar la pulsación de botones es sencillo, pero en el caso del audio no se puede enviar un mensaje pregrabado porque de manera natural hay que hacer uso del micrófono, y en el caso de enviar una imagen hay que seleccionar una del sistema, por lo que la simulación de estas acciones puede requerir de un enfoque distinto y/o más complejo.

- **Identificación de enlaces a páginas externas**: Los botones que no sustituyen a un mensaje que se le puede enviar al bot sino que redirigen a páginas externas ya se detectan por el *listener*, crear una función para el analizador que indique si un bot posee botones de este tipo no supondría apenas esfuerzo.

- **Identificación de idioma**: La interfaz de Messenger toma el idioma que el usuario tiene configurado en Facebook, pero los *bots* no son capaces de reconocer automáticamente esto. Los que poseen botón inicial se pueden

ayudar de este, pues al pertenecer a la interfaz de Messenger si que cambia de un idioma para otro, (*Get started* en inglés, Empezar en castellano). Una vez que se ha recibido el primer mensaje no tiene por qué ser necesario que el *bot* sea capaz de soportar otros idiomas, pero es una característica que se puede analizar.

- **Identificación de comando *cancel***: La identificación de comando *cancel* también puede ser interesante, aunque es más complicada de abordar que el resto de propuestas presentadas hasta ahora, pues es necesario reconocer cuando se está en medio de una conversación, y para ello habría que utilizar heurísticos bastante avanzados y que pueden no adecuarse para la mayoría de *bots*.
- **Análisis del menú de ayuda**: En Messenger todos los bots disponen de un menú de ayuda, el cual también es susceptible de análisis para comprobar si los comandos reflejados en él son útiles. Es una característica que puede ayudar al análisis de la calidad que posee un bot y de la experiencia de usuario que ofrece.
- **Ponderación de elementos**: Se puede intentar realizar una ponderación de todos los elementos analizados, y otorgar en un análisis una puntuación al bot que esté siendo sujeto del mismo. En dicha ponderación se pueden incluir al margen de las características analizadas en este trabajo otras relacionadas con la experiencia de usuario, tales como son el uso de menús, el uso de botones, el uso de plantillas con imágenes, etc. Elementos que enriquecen la experiencia de uso, pues no hay que olvidar que el objetivo de los *bots* es facilitarle al usuario la obtención de información, y para ello nunca está de más hacer uso de las herramientas que estén disponibles.

10.3. Licencias

Todo lo desarrollado en este proyecto se encuentra bajo la licencia MIT. Las herramientas de las que se hace uso poseen licencias diferentes, pero todas permiten la modificación, distribución, uso personal y uso comercial de cada una, de mayor a menor importancia en el proyecto tenemos:

- **Puppeteer**: Apache License 2.0
- **Express**: MIT
- **Socket.io**: MIT
- **Chart.js**: MIT

10.4. Reflexión personal

Finalmente, quiero terminar comentando mis impresiones sobre el proyecto desarrollado.

Debo comenzar mencionando que en un principio consideré que el proyecto iba a resultar más sencillo. En momentos concretos la interacción con los *bots* resultaba un poco desesperante, especialmente la obtención de información asociada a las respuestas de estos, pero esto es debido a la manera que tiene Facebook (y por tanto también Messenger) de hacer las cosas. Inicialmente pensé que el trabajo se iba a orientar más a la creación de heurísticos y a la realización de análisis que a 'pegarme' con Messenger. Los conflictos no han derivado de las herramientas, pues estas son buenas y se han elegido correctamente dado que se adecuaban al trabajo, los conflictos han derivado únicamente de la propia plataforma de Messenger. Los *bots* están pensados para ser usados por humanos, y por ello traen muchos quebraderos si se intenta automatizar una interacción con ellos. Esto me ha llevado a mejorar mi paciencia y mi capacidad de esfuerzo. Además con esto he aprendido a usar nuevas herramientas y he afianzado mis conocimientos sobre otras, por destacar una sobre las demás, me quedo con el paradigma de programación asíncrona, y más teniendo en cuenta que a pesar de usar promesas simples, no me he quedado ahí y he sido capaz de hacer uso de una nueva forma introducida en ES6, las palabras clave async/await.

Es necesario mencionar que, aún habiendo comenzado el proyecto desde cero, he conseguido llevarlo a un estado correcto en aproximadamente medio año.

Quiero destacar por encima de todo la oportunidad de haber podido contribuir a un proyecto novedoso, haber sido capaz de marcar un rumbo y de realizar un trabajo que pueda ser de utilidad para futuros desarrollos y análisis.

Por todo esto, debo dar las gracias a Juanan, por permitirme tener esta experiencia y por su atención para con los problemas que iban surgiendo durante el proyecto.

Bibliografía

H. T. CNBC. Bank of america launches ai chatbot erica, 2017.

Oracle. What is the future role for humans in delivering customer experience?, 2017.

Ubisend. Chatbot Survey, 2017.

E. B. ZDNet. Next Insurance launches Facebook Messenger chatbot to replace the insurance agent, 2017.

Apéndice A

Tipos de respuestas de bots en Messenger

A continuación se realiza un pequeño análisis de las formas en las que puede responder un *bot* y los elementos de los que puede hacer uso.

- Respuestas simples: La forma más básica y simple de comunicación. No hacen uso de las posibilidades de la plataforma para enriquecer la experiencia de usuario.

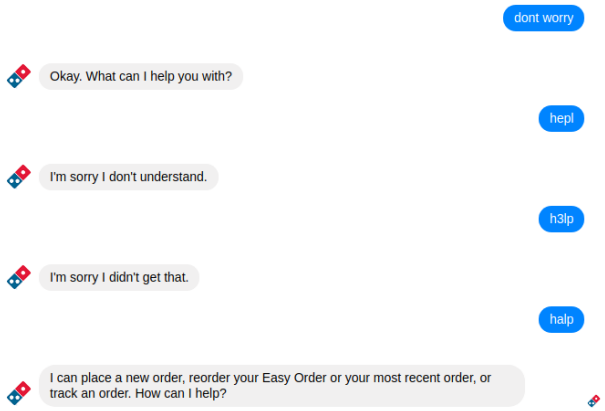

Figura A.1: Ejemplo de respuestas simples consistentes únicamente en texto

- Imágenes (estáticas o *gifs*): Elemento que aporta riqueza a la experiencia de usuario. Pueden presentarse de manera independiente o integrados en plantillas.
- Lista con opciones: Como se puede observar en la figura los botones también pueden contener emojis.

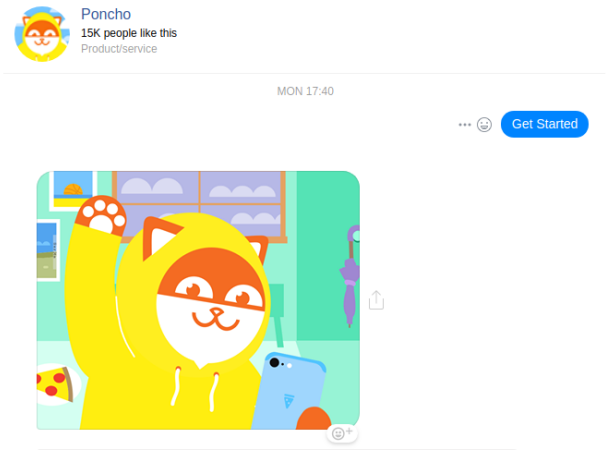

Figura A.2: Ejemplo de imagen de respuesta

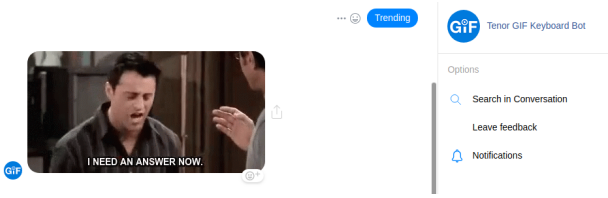

Figura A.3: Ejemplo de gif animado

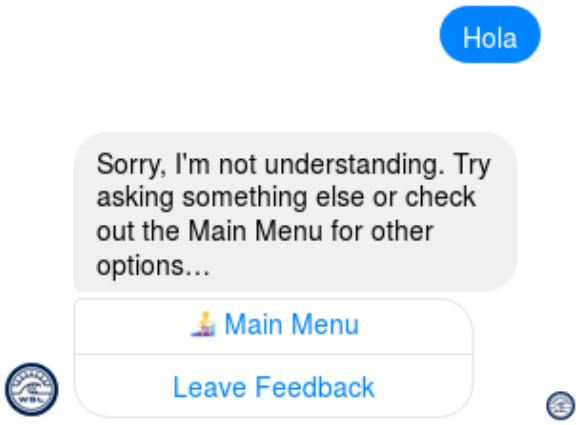

Figura A.4: Ejemplo de uso de un *emoji* en un botón

- Botones al pie de la conversación: Los *bots* también pueden responder mostrando botones en la parte inferior de la conversación, permitiendo al usuario conocer respuestas que el *bot* es capaz de entender a la par que facilitar el acceso a dichas respuestas. Una vez seleccionada una respuesta, o escrita esta de manera normal, estos botones desaparecen.
- Vídeos y Audios: No son muy comunes, pero Messenger soporta estas respuestas.

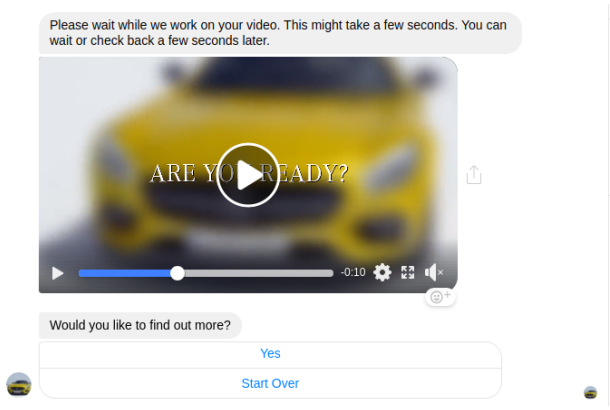

Figura A.5: Ejemplo de vídeo de respuesta

- Plantillas complejas: Son respuestas en las que se hace uso de varios elementos, siendo lo más común encontrarnos con una imagen acompañada de un texto, y una lista de botones para responder fácilmente.

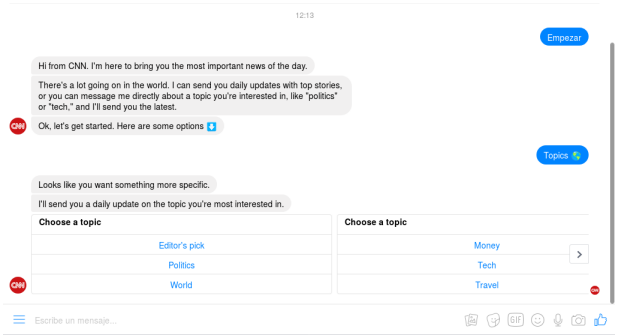

Figura A.6: Plantilla con lista de opciones horizontal o *carrousel*

- Combinaciones de elementos: Los elementos anteriores se pueden presentar de manera simultánea.

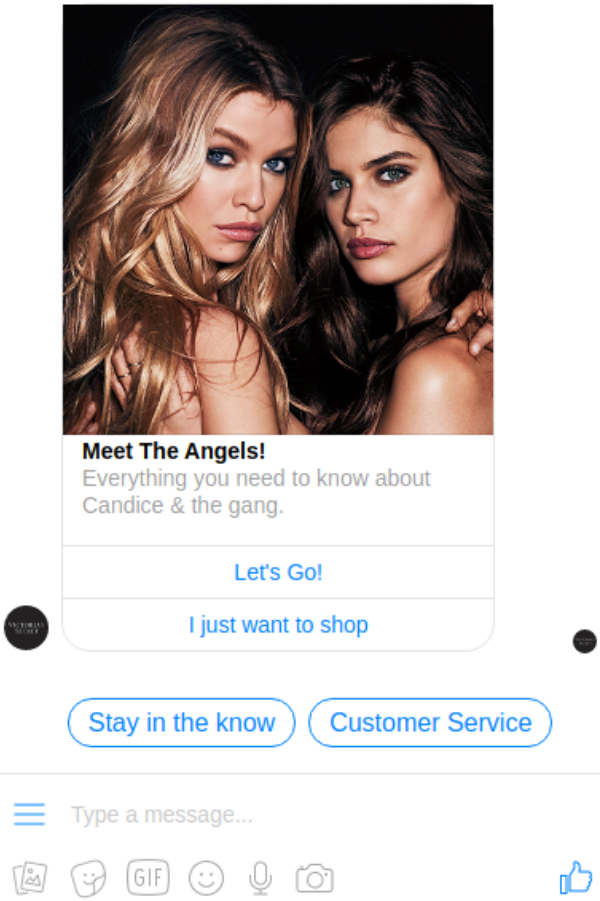

Figura A.7: Ejemplo de uso de botones inferiores, combinados con una plantilla con botones normales

Apéndice B

Script obtención *likes* bots

Se ha realizado un pequeño *script* en Python para conocer los *bots* más populares de Messenger y así poder usarlos en las pruebas del analizador. Con él se *parsea* la información de todas las páginas de *bots* del dominio chatbottle.

Para ello se ha hecho uso de las siguientes librerías:

- **urllib2**: Encargada de gestionar la conexión a la url indicada y así obtener el HTML correspondiente a esta. Obtenemos el HTML de todas las páginas de bots de chatbottle.
- **BeautifulSoup**: Para poder realizar el *scrapping* de las páginas de información de los *bots*.
- **mysqldb**: Gestiona la conexión con Mysql desde python. Tras obtener la url de la página de chatbot del bot y sus likes, realizamos un *insert* en la base de datos.

Hay que destacar que chatbottle tiene actualmente 500 páginas con 20 bots cada una, el análisis de todos los bots lleva aproximadamente unos 20 minutos. En cada página se ignoraban los 3 primeros elementos, el primero por no corresponderse con un bot, sino con publicidad, y los dos siguientes por repetirse en todas las páginas (*bot* del día y *bot* de la semana), sólo se analizan en la primera página. Se ha incluido en el proyecto el archivo sql con todos los datos que se recogen en la tabla usada en el *script*.

Tras esto con el comando `select * from chatbottle order by likes desc limit 200;` obtenemos una lista de los 200 *bots* más populares:

Como se puede apreciar en la imagen, hay *bots* que están repetidos o pertenecen a la misma compañia, por lo que a la hora de realizar las pruebas se han descartado las repeticiones de estos.

Figura B.1: Número de bots en chatbottle

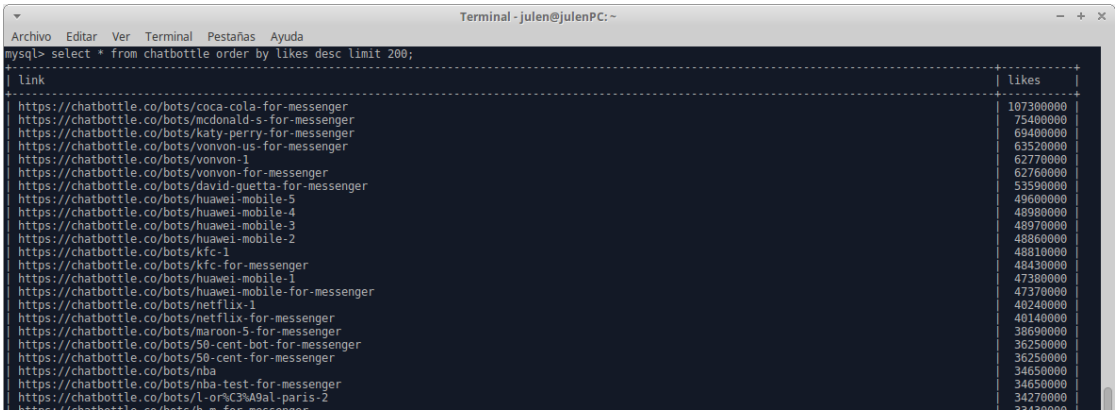

Figura B.2: Bots más populares ordenados por *likes*

Apéndice C

Cambios en testmybot

Como parte del análisis del estado del arte, se ha intentado utilizar la libería de automatización de tests de *bots* testmybot, motivado por Juanan. Dentro de la librería, para trabajar con Messenger hay que acceder a la carpeta fbdirect. Esta librería se encarga de 'grabar' una conversación con el comando `npm run emulator` haciendo uso de la librería facebook-chat-api, y tras finalizar la grabación de la conversación, generar un archivo convo.txt en la carpeta spec/convo con los mensajes enviados al bot desde la consola y las respuestas del bot que se quiere *testear*. Podemos comprobar que se ha grabado correctamente la conversación que ha tenido lugar. Tras esto, se ejecuta el comando `npm run test`, que hace uso del archivo convo.txt generado anteriormente para identificar si el *bot* presenta algún fallo ocasionado por modificaciones realizadas sobre este. Al descubrir Juanan que la librería presentaba fallos con la automatización de tests en Messenger (dicha librería soporta *bots* de muchas aplicaciones), motivó a solventar el error e intentar mejorar la parte de tests.

```
1  conversationInitiated
2
3  #me
4  help
5
6  #bot
7  Need a hand? Tap "Customer Service" for our FAQs or to contact an associate, or tap "Angels 101" to meet the Angels!
   Options are below or in the menu icon (☰).
8
9  #me
10 thanks
```

Figura C.1: Conversación 'grabada' por test my bot en archivo *conversationInitiated.convo.txt*

Tras realizar la grabación de los mensajes enviados al *bot* y las respuestas de este en el archivo convo.txt, observamos el problema a la hora de lanzar el test, pues se obtiene un error que nada tiene que ver con la conversación grabada.

Debido a la naturaleza de la librería, posee una estructura muy particular para poder hacer uso de las mismas funciones de análisis con los *bots* de todas las pla-

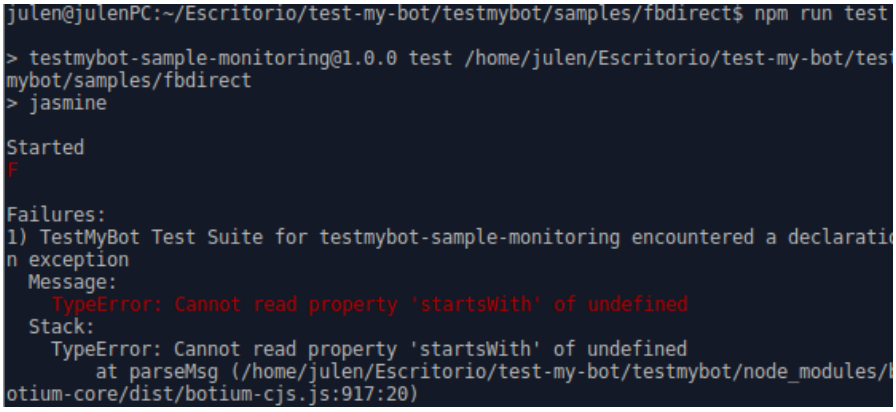

Figura C.2: Fallo existente en testMyBot

taformas que soporta, por lo que el cambio necesario fue difícil de identificar. Tras llevarlo a cabo, y realizar las pruebas para asegurarnos de que estaba solventado , se solicitó un *pull request* para *mergear* la nueva rama en la que se había realizado el commit con el cambio. El desarrollador principal de la librería, Florian Treml dió el visto bueno y realizó el *merge* de los cambios realizados.

Figura C.3: Prueba correcta en la que se supera el test

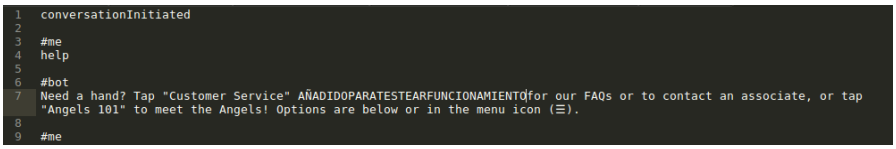

Figura C.4: Cambio realizado para probar el caso negativo

Tras esto, se quiso contribuir intentando mejorar la librería, Juanan propuso incluir un análisis del tiempo que tarda el bot en contestar cada mensaje, y que

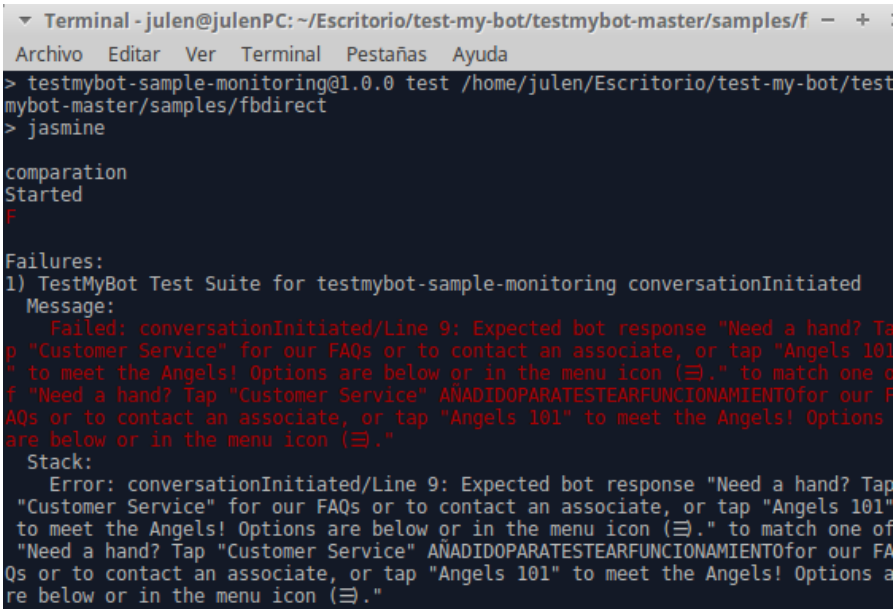

Figura C.5: Prueba del caso negativo en el que no se pasa el test

el test fallase en caso de superarse un umbral de tiempo establecido al dar una respuesta. Se estudió este cambio, pero por la estructura de la librería, dicho cambio se desestimó por su complejidad, en su lugar se decidió cambiar la forma en la que funcionaban los inputs al grabar las conversaciones: como se puede apreciar en la figura , cuando se están enviando los mensajes al bot a través de la consola, no se puede hacer uso de las flechas del teclado, puesto que si se intenta cambiar el cursor del teclado pulsando la tecla de izquierda para modificar una letra mal escrita, se obtienen carácteres no deseados. De la misma forma, si queremos repetir un mensaje enviado previamente, no podemos hacer uso de la tecla 'arriba' como en otras aplicaciones de consola que almacenan los comandos introducidos previamente, en su lugar obtenemos los carácteres asociados a la tecla 'arriba'. La corrección de dichas 'molestias' puede suponer una mejora de la experiencia de uso de la librería, por lo que se decidió solventarlas. Tras un primer intento que no satisfacía tests de prueba para *bots* de IBM Watson, se consiguió realizar la corrección para que funcionase en las grabaciones de todos los tipos de *bots*, y Florian realizó el *merge* de la rama en la que había realizado los cambios.

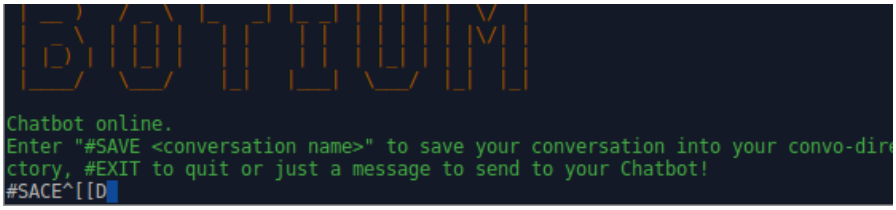

Figura C.6: Molestia generada al usar las flechas del teclado al escribir un mensaje.