GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Diana: bot asistente de compras online en Telegram

Autor: Ángel Sánchez Sierra

Director: Mario Castro Ponce

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diana: bot asistente de compras online en Telegram

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/20 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Ángel Sánchez Sierra Fecha: 26/ 07/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Mario Castro Ponce Fecha: 26/07/2020

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Diana: bot asistente de compras online en Telegram

Autor: Ángel Sánchez Sierra

Director: Mario Castro Ponce

Madrid

Agradecimientos

En primer lugar, quiero agradecer a mi familia por el apoyo y esfuerzo realizado, gracias al cual he podido estudiar en esta universidad. Ni este TFG ni mi formación, hubieran sido posibles sin ellos. Entre ellos especialmente a mi abuelo, sin cuyo trabajo ni mi padre ni yo hubiéramos tenido la suerte de formarnos, y que a pesar de que ya no esté entre nosotros, estoy seguro de que se habría sentido muy feliz de verme donde estoy ahora.

En segundo lugar, me gustaría agradecer a mis amigos que me han apoyado durante estos años, tanto a aquellos que he conocido gracias a la universidad y que han demostrado ser grandes persona, como a mis amigos de toda la vida. Si he podido mantenerme positivo y caminando hacia delante estos años, ha sido gracias a ellos.

Finalmente me gustaría agradecer a esos profesores, algunos de ellos que incluso sin conocerme, me ayudaron y me enseñaron con paciencia. Y que me animaron a seguir adelante. Muchas gracias también a mi director de proyecto Mario Castro, por dedicarme su tiempo y ayudarme a llevar a cabo este proyecto.

Espero que este trabajo sea espejo del esfuerzo y dedicación de los últimos 4 años, y del apoyo de la gente que llevo detrás.

DIANA: BOT ASISTENTE DE COMPRAS ONLINE EN TELEGRAM

Autor: Sánchez Sierra, Ángel.
Director: Castro Ponce, Mario.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto consiste en el diseño de un bot (programa informático que es capaz de realizar actividades de manera autónoma), cuyo funcionamiento se dará en la plataforma de Telegram. El lenguaje de programación usado será Python 3.8. El principal uso de este bot será la búsqueda de productos en varias webs. Pudiéndose ampliar tanto las webs como las funciones del bot fácilmente, ya que no requiere de ningún hardware para funcionar más que un servidor.

Palabras clave: Telegram, Móvil, Bot, Python

1. Introducción

En los últimos años, el uso de internet como nueva forma de compra ha estado en auge continuo, especialmente en el año 2020, debido a los extraordinarios acontecimientos, la población ha evolucionado aumentando el uso del comercio electrónico.

Este proyecto pretende entrar a formar parte de este nuevo mercado emergente, ayudando a la gente a administrar y encontrar los productos que deseen en el mayor abanico posible de tiendas.

El objetivo principal del proyecto es acercar esta nueva forma de vida a la mayor parte de la población posible, a través de Diana, un bot gratuito que permite buscar en varias de estas tiendas fiables un producto para así, conseguir el producto al mejor valor para el consumidor.

2. Definición del Proyecto

En este proyecto se desarrollará un bot para la plataforma de mensajería Telegram. Para ello utilizaremos el lenguaje de programación Python 3.8, además esta será la primera vez que el autor del proyecto usa el lenguaje, por lo que el aprendizaje de dicho lenguaje se desarrolló a la par que el proyecto.

El principal uso de este bot será la búsqueda de productos en varias webs, y la creación de listas de seguimiento de productos mediante el uso de bases de datos, para así guardar productos que el usuario desee, a fin de mostrarlos al usuario cuando cambien de precio.

Pudiéndose ampliar tanto las webs como las funciones del bot fácilmente, ya que no requiere de ningún hardware para funcionar más que un servidor.

3. Descripción del sistema y esquema

El bot funcionará enteramente en un servidor, ese servidor puede estar situado tanto en un dispositivo físico personal (como puede ser un ordenador o teléfono móvil), o en un

servidor en la nube. Por otro lado, la base de datos que se utilizará, que será MongoDB, será una base de datos NoSQL que se podrá situar tanto en dispositivo físico, como en la nube (al igual que el bot).

El funcionamiento será el siguiente: el usuario se conecta a Telegram, el cual envía desde sus servidores la información del usuario; una vez el servidor recibe esa información, la analiza y responde al usuario a través de los servidores de Telegram. Por otro lado, después de recibir la información, el servidor puede requerir de algún elemento externo a él, como por ejemplo: el Email para enviar un mensaje al administrador; la web de divisas, para obtener información en tiempo real de las divisas y su cambio actual; la base de datos de MongoDB; las contraseñas para inicializar el correo o el email, que estarán en un archivo .cfg externo al servidor por temas de seguridad; o finalmente acceso a las webs de las cuales se quiere realizar la compra.

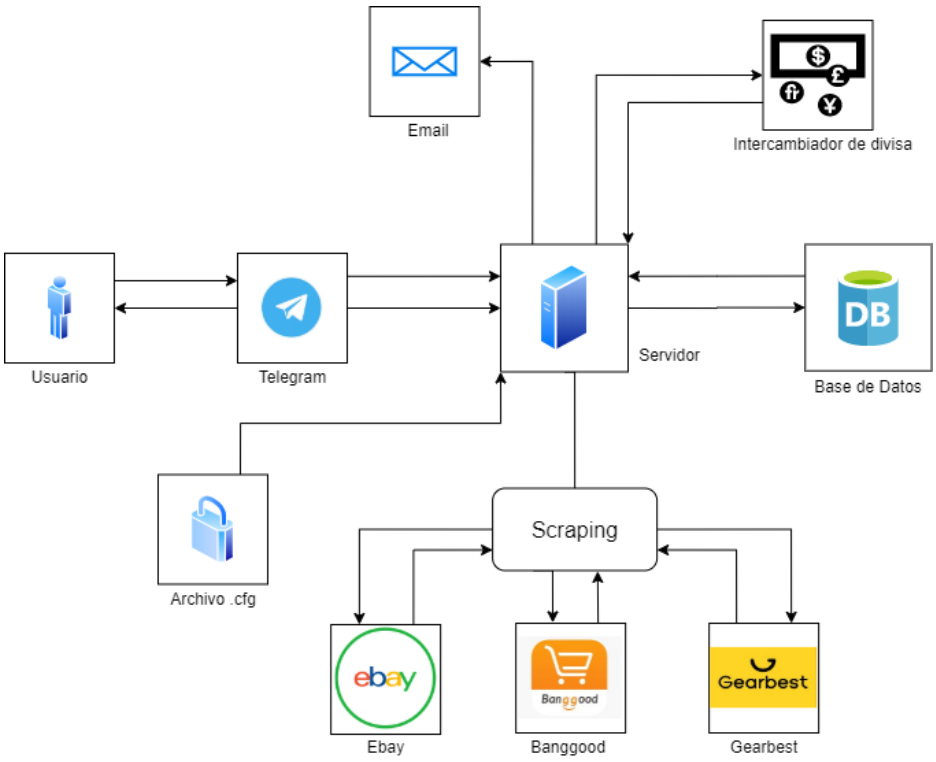

Ilustración 1 – Workflow General

4. Resultados

Tras finalizar este proyecto, se han conseguido los resultados esperados. Los objetivos cumplidos son:

- Un interfaz fácil de usar para todos los usuarios, independientemente de que no tengan gran experiencia en el sector de las compras online.

- Un interfaz con posible cambio entre inglés o español, para su uso no solo en España, sino a nivel internacional. También es posible un cambio de divisa para comprar no solo en euros sino en cualquier otra divisa.
- Una herramienta de búsqueda flexible, que permite cambiar el intervalo de precios y las tiendas en las que se desea buscar, a gusto de cada usuario.
- Una lista de productos capaz de guardar los productos que le resulten interesantes al usuario y realizar comprobaciones de precio continuas gracias al uso de una base de datos para almacenamiento externa al servidor de trabajo.
- El desarrollo de un proyecto fácilmente actualizable a lo largo del tiempo y con posibles mejoras, cuyos datos previos no se perderán, gracias a su almacenamiento en la base de datos.
- El uso de una herramienta completamente gratuita y sin costes ni anuncios, accesible para todo el mundo.

5. Conclusiones

En este proyecto se ha desarrollado un bot completamente desde cero, sin ningún conocimiento previo de ninguna de las herramientas usadas para ello, exceptuando unas bases de otros lenguajes de programación que han acelerado el aprendizaje de Python, pero que no son esenciales para la comprensión de este lenguaje de programación. Por ello, este proyecto puede extrapolarse a otros casos diferentes de desarrollo de bots, y no está completamente ligado a este caso específico, sino que se pueden usar los conocimientos plasmados en esta memoria para el desarrollo de un bot completamente diferente usando los scripts de comunicación del usuario con Telegram y el servidor, o el uso de la base de datos (lo que necesite cada persona), y adaptándolos a las necesidades de cada uno, ya que todo el código está desarrollado en diferentes módulos para hacer más fácil su uso y entendimiento.

Además, si bien el proyecto funciona correctamente y cumple con las expectativas iniciales, cabe mucho margen de mejora y de ampliación con nuevas funciones. Aunque se recomienda que para mejorar la velocidad y disminuir la carga del servidor, se desarrollen diferentes bots con funcionamientos más específicos en vez de uno con un funcionamiento más general, ya que al estar funcionando en la misma plataforma (Telegram) resulta fácil el acceso a varios de ellos y es más estructural y organizado que tenerlo todo en el mismo bot. Además, hace más fácil al usuario su uso y aprendizaje. Dicho esto, si se quiere añadir alguna mejora o expansión del bot, las principales que se recomiendan son:

- Uso de librerías de Python para desarrollar las comunicaciones con Telegram y el scraping web, ya que disminuye la carga de aprendizaje y tienen varias opciones que pueden ser útiles para otros proyectos.
- Uso de las API de las webs de compra (si existen), ya que aumenta la velocidad de obtención de información de ellas.

6. Referencias

[1] MongoDB. (marzo de 13 de 2020). *Manual MongoDB*. Obtenido de Manual MongoDB: https://docs.mongodb.com/manual/

[2] MongoDB. (9 de marzo de 2020). *MongoDBCompassCommunity*. Obtenido de MongoDBCompassCommunity: https://www.mongodb.com/try/download

[3] Python. (14 de julio de 2020). *python descargas*. Obtenido de python descargas: https://www.python.org/downloads/

[4] Telegram. (9 de julio de 2020). *Telegram Bot API*. Obtenido de Telegram Bot API: https://core.telegram.org/bots/api

[5] Wikimedia Foundation, Inc. (29 de Enero de 2020). *Wikipedia*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Telegram

DIANA: ONLINE SHOPPING ASSISTANT BOT IN TELEGRAM

Author: Sánchez Sierra, Ángel.
Supervisor: Castro Ponce, Mario.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project consists of the design of a bot (a computer program that can carry out activities in an autonomous way), whose operation will take place on the Telegram platform. The programming language used will be Python 3.8. The main use of this bot will be to search for products on various websites. The main idea is being able to expand both the webs and the functions of the bot easily since it does not require any hardware to operate apart from the server.

Keywords: Telegram, Mobile, Bot, Python

1. Introduction

In recent years, the use of the internet as a new form of purchase has been increasing continuously, especially in the year 2020, due to extraordinary events, the population has evolved, increasing the use of electronic commerce.

This project aims to become part of this newly emerging market, helping people to manage and find the products they want in the widest possible range of online shops.

The main objective of the project is to bring this new way of life closer to as much of the population as possible, through Diana, a free bot that allows you to search several of these reliable stores for a product so that the consumer can get the product at the best value.

2. Project definition

In this project, a bot will be developed for the Telegram messaging platform. For this it will be used the Python 3.8 programming language, also this will be the first time that the author of the project uses the language, so the learning of this language was developed along with the project.

The main use of this bot will be the search for products on various websites, and the creation of product tracking lists through the use of databases, in order to save products that the user wants, so the bot will show them to the user when they change their price.

The main idea is being able to expand both the webs and the functions of the bot easily since it does not require any hardware to operate apart from the server.

3. Description of the System and scheme

The bot will work entirely on a server, that server can be located both on a personal physical device (such as a computer or mobile phone), or on a server in the cloud. On the other hand, the database that will be used, which will be MongoDB, will be a NoSQL database that can be placed on both the physical device and the cloud (just like the bot).

The operation will be as follows: the user connects to Telegram, which sends the user's information from its servers; once the server receives this information, it analyzes it and responds to the user through the Telegram servers. After receiving the information, the server may require something external to it, such as: Email to send a message to the administrator; the currency website, to obtain real-time information on currencies and their current exchange rate; the MongoDB database; passwords to initialize mail or email, which will be in a .cfg file external to the server for security reasons; or finally access to the websites from which you want to make the purchase.

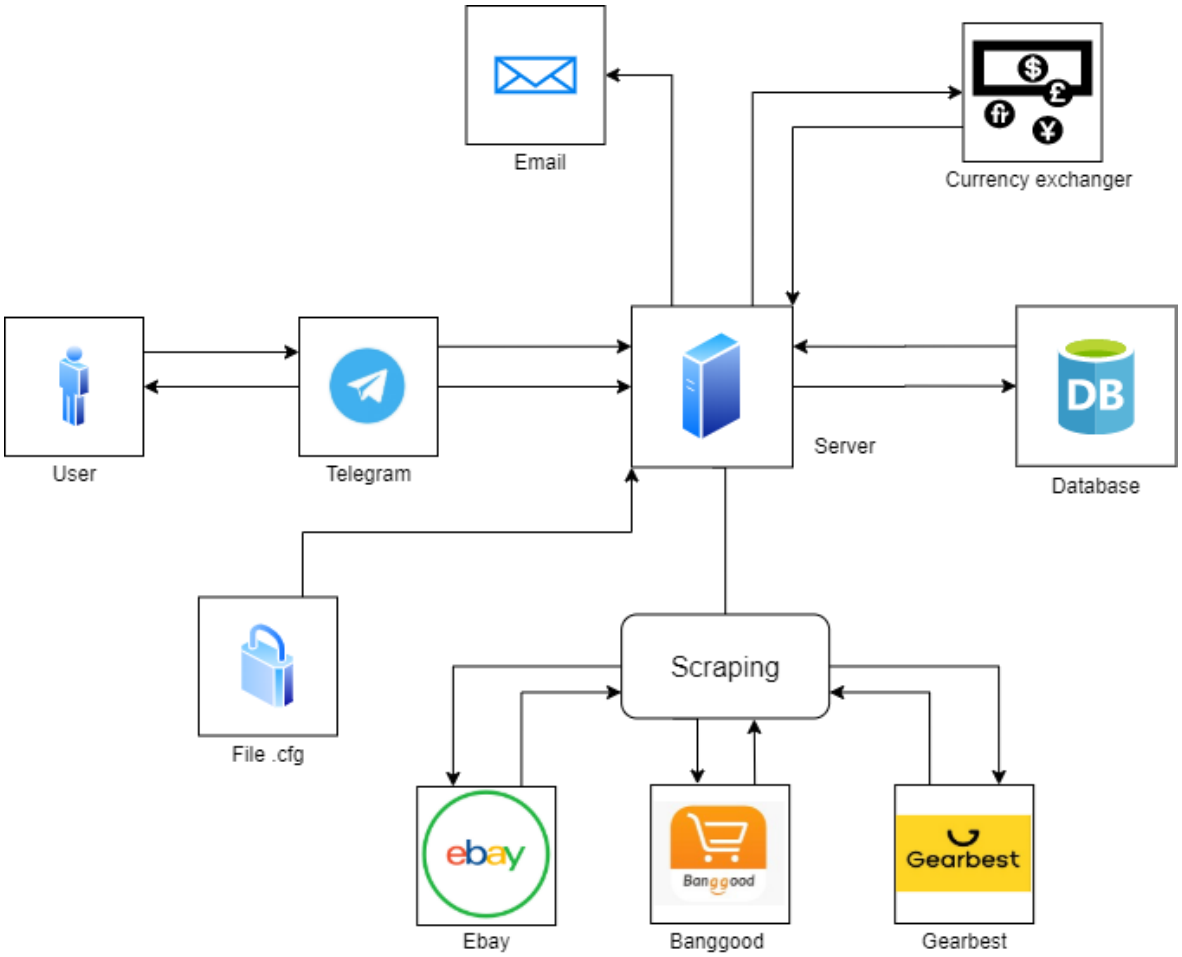

Ilustración 2 – General Workflow (English version)

4. Results

After completing this project, the expected results have been achieved. The objectives achieved are:

- An easy-to-use interface for all users, regardless of whether they have extensive experience in the online shopping industry.
- An interface allowing the possibility to change between English or Spanish, for use not only in Spain, but internationally. A currency exchange makes possible to buy not only in euros but in any other currency.

- A flexible search tool, which allows you to change the price range and the shops in which you want to search, to suit each user.

- A list of products capable of saving the products that are interesting to the user and carry out continuous price checks thanks to the use of a database for storage external to the work server.

- The development of a project that can be easily updated over time and with possible improvements, whose previous data will not be lost, thanks to its storage in the database.

- The use of a completely free tool with no costs or ads, accessible to everyone.

5. Conclusions

In this project, a bot has been developed completely from scratch, without any prior knowledge of any of the tools used for it, except for some bases of other programming languages that have accelerated the learning of Python, but that are not essential for the understanding of this programming language. For this reason, this project can be extrapolated to other different cases of bot development, and it is not completely linked to this specific case, but the knowledge expressed in this report can be used to develop a completely different bot using communication scripts with Telegram and the server, or the scripts of the database and adapting them to the needs of each one, since all the code is developed in different modules to make its use easier and understanding.

In addition, although the project works correctly and meets initial expectations, there is much room for improvement and expansion with new functions. However, it is recommended that to improve the speed and decrease the server load, different bots should be developed with more specific operations instead of one with more general operation. Since when operating on the same platform (Telegram) it is easy to access several of them and it is more structural and organized than having it all in the same bot. Also, it makes the user easier to use and learn this bot. That said, if you want to add any improvement or expansion of the bot, the main ones that are recommended are:

- Use of Python libraries to develop communications with Telegram and web scraping, since it reduces the learning load and they have several options that may be useful for other projects.

- Use of the APIs of the purchase websites (if they exist), since it increases the speed of obtaining information from them.

6. References

[1] MongoDB. (marzo de 13 de 2020). *Manual MongoDB*. Obtenido de Manual MongoDB: https://docs.mongodb.com/manual/

[2] MongoDB. (9 de marzo de 2020). *MongoDBCompassCommunity*. Obtenido de MongoDBCompassCommunity: https://www.mongodb.com/try/download

[3] Python. (14 de julio de 2020). *python descargas*. Obtenido de python descargas: https://www.python.org/downloads/

[4] Telegram. (9 de julio de 2020). *Telegram Bot API*. Obtenido de Telegram Bot API: https://core.telegram.org/bots/api

[5] Wikimedia Foundation, Inc. (29 de Enero de 2020). *Wikipedia*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Telegram

Índice de la memoria

Índice de figuras

Capítulo 1. INTRODUCCIÓN

Durante estos últimos años, el uso de internet como nueva forma de compra ha estado en auge continuo, especialmente tras los extraordinarios acontecimientos vividos en España y el resto del mundo durante el año 2020 por la crisis mundial del coronavirus que no ha permitido a la gente acudir a tiendas físicas. Esto ha hecho que surjan de manera continua e ininterrumpida tiendas online para este nuevo comercio electrónico. Sin embargo, al aparecer tantas tiendas online, surge la dificultad de no saber en cual buscar, ya que al vender todas productos similares o incluso iguales, es primordial encontrar una tienda segura, que además ofrezca el mejor precio.

El objetivo que se busca con este proyecto es acercar esta nueva forma de vida a todo el mundo, a través de Diana, un bot gratuito que permite buscar en varias de estas tiendas fiables un producto para así, conseguir el producto al mejor valor para el consumidor.

1.1 MOTIVACIÓN DEL PROYECTO

La razón por la que se ha escogido este sistema en una plataforma como Telegram en lugar de por ejemplo como una aplicación, es debido al auge también de todas las redes sociales y de mensajería, entre las que Telegram se incluye. Facebook, WhatsApp y Twitter, son algunos de los principales portales de mensajería y socialización, por los que al día circulan millones de usuarios, es por eso que Telegram, que también tiene al día millones de usuarios conectados, es preferible para el desarrollo de una app, ya que, al situarse dentro de la aplicación, se llega ya a millones de usuarios sin necesidad de publicitar una app externa. También es importante destacar, que, aunque bien es cierto que WhatsApp es en 2020, la plataforma de mensajería más utilizada en España, el crecimiento de Telegram lo sitúa como su principal competidor gracias a sus muchas ventajas sobre este, una de las cuales es el desarrollo de bots.

Gracias al desarrollo del bot en una plataforma gratuita ya establecida, se ahorran muchos costes tanto de mantenimiento, como de publicidad. Por otro lado, aunque el servicio que ofrece el bot es gratuito, existen actualmente programas de comercialización de las propias webs de ventas conocidos como Programas de Afiliación. Estos son programas publicitarios desarrollados por las webs con el objetivo de promocionarse, al que cualquier persona con una página web, blog o medio de comunicación puede suscribirse para obtener un id de afiliado. Este id se adjunta en los enlaces de los productos que el usuario registrado coloca en su página web, para así recibir una comisión de cada venta producida, estas comisiones oscilan entre el 2% y el 15% dependiendo del producto. Con esta forma de generar beneficio, no se molesta al comprador, ya que no produce ningún coste extra en su compra, y la persona que envía en enlace del producto recibe una compensación económica por sus servicios. Teniendo en cuenta que el bot puede mandar miles de enlaces a la hora y muchos de ellos serán de productos que finalmente serán comprados, el beneficio económico está asegurado, y una vez que el bot esté funcionando, el dinero se generará de manera automática y continuada con unos costes de mantenimiento del servidor del bot mínimos.

Además, este bot ayudará a las personas a comprar siempre de manera fiable y óptima, produciendo una ayuda económica a todo el mundo que lo use, ya que, al acceder al producto a mejor precio, ahorrará un dinero que no ahorraría si comprase siempre en el mismo lugar o de manera más convencional.

1.2 ESTRUCTURA DEL DOCUMENTO

Este documento se estructura en 8 capítulos. En este primer capítulo se hace una introducción del documento, explicando también la motivación tras él. En el Capítulo 2. se describen las tecnologías que se van a utilizar, esto son los programas que se van a usar y la documentación necesaria (más tarde en el ANEXO I: Instalación de programas se explicará la instalación de estos). En el Capítulo 3. se desarrolla el estado de la cuestión en el cual se explica la situación actual del mercado en el que se pretende que entre este proyecto.

En el Capítulo 4. se explican tanto la justificación del proyecto, como la metodología a seguir y la planificación económica y temporal. El **¡Error! No se encuentra el origen de la r eferencia.**explica las diferentes partes de Diana y cómo funcionan (todos los códigos usados en este capítulo estarán el ANEXO II: Código).

El Capítulo 6. es en el cual se explican los resultados obtenidos y se explica brevemente que mejoras podrían haberse dado para conseguir unos mejores (lógicamente esto solo se entiende después de haber llegado a estos primeros resultados). El Capítulo 7. explica las conclusiones del trabajo y posibles mejoras o futuros proyectos a partir de este, muchas de estas mejoras derivan de las explicadas en los resultados del Capítulo 6. El **¡Error! No se e ncuentra el origen de la referencia.**contiene toda la bibliografía utilizada en el documento.

Por otro lado, hay dos anexos: el ANEXO I: Instalación de programas explica la instalación de cada programa necesario, y el ANEXO II: Código contiene todo el código utilizado en el proyecto.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 SOFTWARE

En este apartado se describirán los software y aplicaciones que se utilizarán durante el proyecto.

2.1.1 PYTHON 3.8

Python es uno de los lenguajes de programación más comunes y usados actualmente. También su escasa complejidad y facilidad de utilizar otros códigos desarrollados previamente a través de sus muchas librerías han hecho que sea el idóneo para este proyecto. La versión que se utilizará durante todo el proyecto es la 3.8, es importante trabajar en esta versión, ya que el funcionamiento de las librerías para otras versiones no está asegurado.

Su instalación está explicada en el ANEXO I: Instalación de programas: **¡Error! No se e ncuentra el origen de la referencia.** para Mac como para Windows.

Ilustración 3 - Python

2.1.2 PYCHARM COMMUNITY EDITION 2019.3.2

Tanto para programar el código, como para ejecutarlo y depurarlo, se ha usado PyCharm Community Edition 2019.3.2, esta era la versión gratuita de PyCharm disponible en octubre de 2019. Sin embargo, no es necesario usar esa versión, y se recomienda utilizar la última versión disponible, ya que contendrá mejoras en el programa y nuevas opciones, y debería ser capaz de utilizar las mismas librerías siempre y cuando se use Python 3.8. No obstante,

en caso de haber algún problema, esta versión está disponible en la web de PyCharm (JetBrains, 2019), así como las versiones posteriores y anteriores.

Aunque este proyecto ha sido desarrollado a través de PyCharm, también pueden utilizarse otras aplicaciones con el mismo fin, como por ejemplo Spider o Visual Studio. En este apartado, se recomienda que se use la más cómoda para cada programador.

La instalación de este programada esta explicada en el ANEXO I: Instalación de programas: **¡Error! No se encuentra el origen de la referencia.**.

Ilustración 4 - PyCharm

2.1.3 MONGODB

Para la base de datos que se utilizará, se ha optado por escoger MongoDB, debido a su similitud con el formato JSON para distribuir la información. Al ser una base de datos NoSQL, no se requiere conocer de este tipo y aprender a programarlas, el objetivo de esto es simplificar su uso y aprendizaje. Para acceder a ella y trabajar con ella se utilizará una librería de Python llamada pymongo (MongoDB, 2020), y para ver y analizar los datos se utilizará MongoDBCompassCommunity (MongoDB, 2020), que se encuentra disponible de manera gratuita en la web de MongoDB.

La instalación de la aplicación MongoDBCompassCommunity se ha facilitado en el ANEXO I: Instalación de programas: **¡Error! No se encuentra el origen de la referencia.**.

Ilustración 5 - MongoDB

2.1.4 TELEGRAM API

Para las comunicaciones entre el servidor y Telegram, se utilizará la API desarrollada por Telegram específicamente para el uso de bots. Toda la información necesaria para entender y estudiar la API se encuentra en la web de Telegram y se puede ver en la bibliografía (Telegram, 2020).

2.1.5 LIBRERÍAS DE PYTHON

Durante este proyecto se utilizarán varias librerías de Python para diferentes tareas. Todas ellas disponen de documentación que se adjuntará en la bibliografía.

Las librerías que se utilizarán en este proyecto son:

- smtplib (Python Software Foundation, 2020). Se usará para mandar correos a través del bot. Estos correos no pueden inicialmente mandar caracteres no-ASCII, por lo que se necesitará utilizar a parte la librería email (Python Software Foundation, 2020).
- email (Python Software Foundation, 2020). Esta librería se utilizará únicamente para transformar los caracteres no-ASCII y que puedan ser enviados por correo por el usuario al administrador.
- configparser (Python Software Foundation, 2019). Esta librería se utilizará para leer los archivos protegidos (como contraseñas) de un archivo seguro.

- currency_converter (Prengère, 2020). Esta librería se utilizará para cambiar de divisa según dónde o qué queramos comprar.
- pymongo (MongoDB, 2020). Esta librería será necesaria para trabajar con MongoDB desde Python, y manipular así la base de datos.
- json (Python Software Foundation, 2019). Esta librería se utilizará para ver archivos en formato json, que es como vendrá la información de la API de Telegram y de MongoDB.
- time (Python Software Foundation, 2020). Esta librería se usará para aquellas funciones que requieran acceso a la hora.
- requests (Reitz, 2020). Esta librería es indispensable para obtener datos de internet y se usará principalmente a la hora de hacer Scraping para obtener los datos de la página web.
- bs4 (Richardson, 2020). Esta librería se usará para visualizar los datos obtenidos mediante la librería requests (Reitz, 2020), ya que inicialmente requests te saca todos los datos sin ordenar.

2.2 *HARDWARE*

El único hardware necesario para este proyecto es un ordenador con conexión a internet, que permita la instalación de los programas necesarios. El sistema operativo de ese ordenador puede ser tanto IOS como Windows o Linux, ya que los programas utilizados admiten cualquier sistema operativo de los mencionados.

Capítulo 3. ESTADO DE LA CUESTIÓN

3.1 COMERCIO ELECTRÓNICO

El comercio tradicional ha ido decayendo en los últimos años dando paso al comercio electrónico. La principal prueba de ello se llama Amazon y es la empresa gracias a la cual Jeff Bezos se convirtió en 2018 (conservando el título en 2019) en el hombre más rico del mundo según la revista Forbes (Forbes, 2020). Esta empresa produce sus ventas únicamente de manera online, demostrando que esta forma de venta es capaz de competir directamente con el comercio tradicional, e incluso superarlo.

Tras el éxito del gigante de ventas Amazon, han surgido varias alternativas. Algunas de ellas son Ebay, Gearbest, Banggood, Alibaba y Aliexpress. Tres de las cuales han sido incorporadas a este proyecto.

3.2 TIENDAS ONLINE

Es importante entender, que este proyecto no se postula como una alternativa a las tiendas online o como un portal de venta diferente a ellas. Sino que, por el contrario, es una forma de ayudar a estas tiendas online, mejorando y ampliando su público a través de los usuarios que accedan al bot y de su uso. Es por tanto una relación simbiótica entre el bot que da a los usuarios unas webs donde efectuar sus compras; y las webs, que al producir buenos resultados con productos fiables y de calidad, animan al usuario a seguir comprando en ellas, haciéndolo de manera más fácil a través de Telegram y por tanto del bot.

3.3 PORTALES DE BÚSQUEDA ONLINE

Actualmente, en 2020, hay varias plataformas auxiliares para buscar productos a buen precio, la más famosa en España es el Chollometro. Sin embargo, el funcionamiento de esta plataforma es diferente del funcionamiento del bot de Telegram, ya que el Chollometro

únicamente publica los mejores chollos, es decir, únicamente publica las ofertas puntuales que surgen de algunos productos, pero no publica continuamente los mejores precios de todos los productos.

Para explicarlo fácilmente con un ejemplo, imagina que Ebay saca el iPhone 10 con una rebaja de 200€, esa oferta saldría en el Chollometro y también al buscar iPhone 10 en el bot de Telegram. Sin embargo, una vez terminada la oferta, el iPhone 10 desaparecería de la web del Chollometro ya que el chollo ha caducado. Por otro lado, si se buscase iPhone 10 en el bot de Telegram, este seguiría encontrando varios resultados mostrando siempre el más barato, que en el caso de haberse acabado la oferta en Ebay, podría estar en Gearbest o en Banggood.

De esta manera, aunque el Chollometro es una buena opción para encontrar grandes ofertas puntuales, no lo es para compras necesarias en el momento o en el día a día. Para lo cual el bot de Telegram es de mayor ayuda.

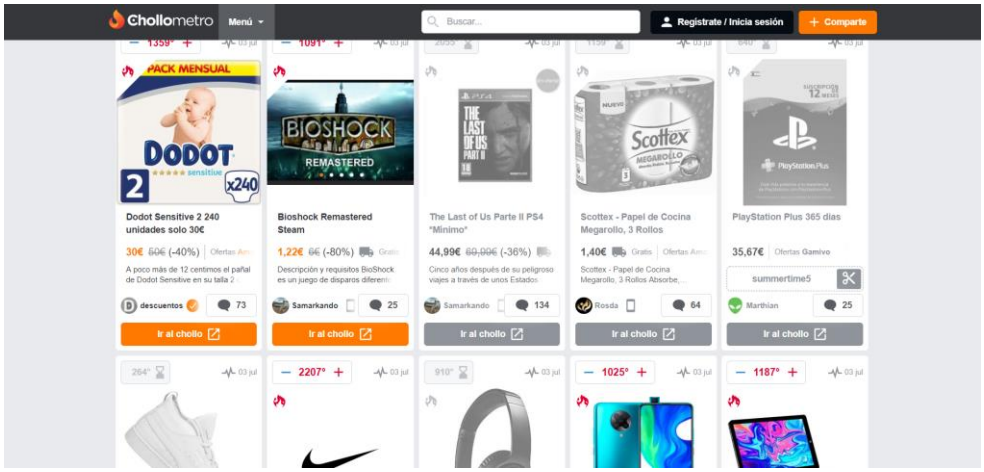

Ilustración 6 – Chollometro portal web

A parte del Chollometro existen varios comparadores de precios, uno de los más utilizados es Google Shopping. La principal ventaja que tiene el bot respecto a estos, es su uso a través de Telegram, ya que es mucho más rápido y directo utilizar una herramienta de búsqueda dentro de la aplicación de Telegram, cuando hablando con otro usuario aparece la necesidad de buscar un producto, que hacerlo con una herramienta externa como puede ser Google.

Además, aunque este primer proyecto únicamente cubre chats directos con el usuario (no grupos), está la opción de ampliar el bot para que pueda usarse en grupos y así poder ser utilizado rápidamente en mitad de una conversación.

3.4 CONCLUSIONES

La principal ventaja que tiene este proyecto contra lo que ya está establecido en internet, es formar parte de uno de los medios de comunicación más utilizados actualmente y que sigue en continuo crecimiento, Telegram. Además, su desarrollo en esta plataforma no requiere de ningún pago, y lo único necesario para su desarrollo es un ordenador.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Como ya se explicó en el apartado Motivación del proyecto, Telegram es uno de los principales portales de mensajería actuales. La cantidad de usuarios activos mensualmente en esta plataforma es de 400 millones según los datos publicados por Telegram en abril de 2020. Aunque la cifra de usuarios activos de WhatsApp es de 2000 millones y aún está lejos, se puede observar en la Ilustración 7 - Crecimiento de usuarios en Telegram en el tiempo, un crecimiento exponencial de los usuarios que apunta a mantenerse. Por lo que el alcance de Telegram a WhatsApp podría verse en los próximos años.

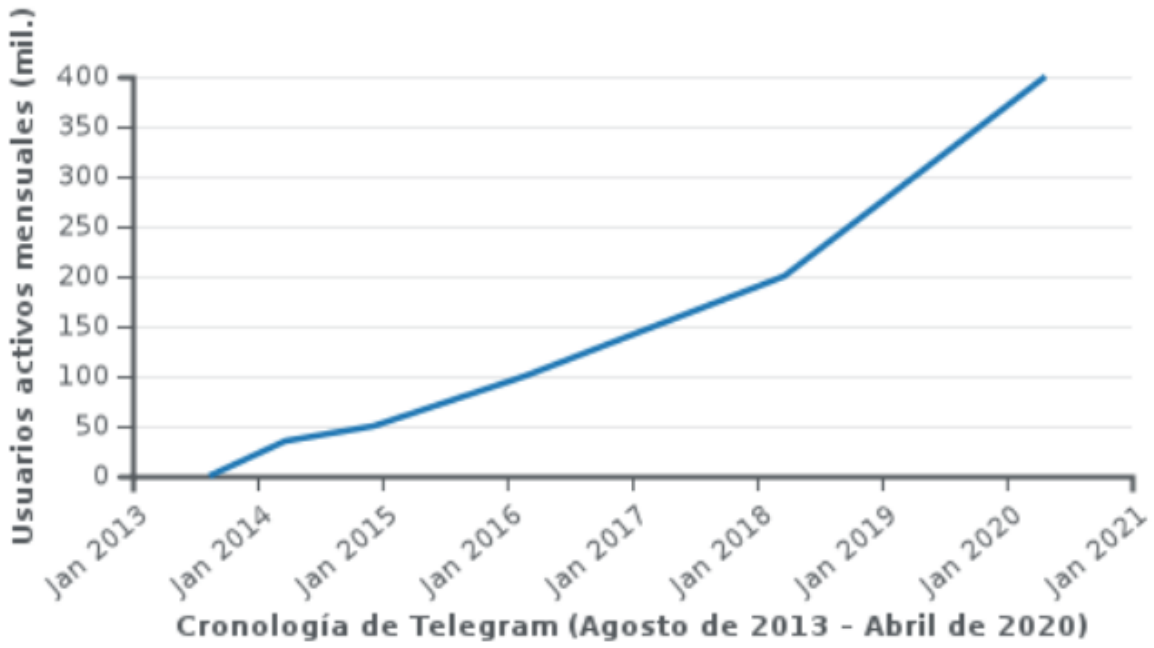

Ilustración 7 - Crecimiento de usuarios en Telegram en el tiempo

También, una de las grandes ventajas que tiene Diana, es que el único gasto de inversión en el proyecto es el sueldo del desarrollador, más tarde, ya que al funcionar en la plataforma de Telegram, el coste de mantenimiento del servidor es ínfimo, en un primer momento se puede

llevar a cabo con un ordenador o un servidor físico, y más tarde, se podría llevar a cabo de manera online, operando el servidor desde la nube. Este no es un gran coste, y los beneficios que se pueden conseguir modificando el bot una vez se consigan las cuentas afiliados, son constantes y continuos.

Además, para el usuario, el uso de una herramienta libre de anuncios y sobrecostes es algo de lo cual disfrutará y hará que permanezca usándola. Según los datos que se tienen de 2019, cada español gasta una media 2.098 € al año en compras online (itreseller, 2020). Si al menos cada usuario que use Diana consumiese por el bot el 50% de su gasto anual, es decir unos 1000 €, si se ingresase un 5% del gasto de los usuarios gracias a los programas de afiliados, el beneficio sería de 50 € anualmente por usuario, es difícil calcular aproximadamente cuantos usuarios podrían llegar a usar a Diana, pero si el número de usuarios en Telegram es de 400 millones, hay mucho margen de crecimiento.

4.2 OBJETIVOS

El objetivo principal de este proyecto es desarrollar una herramienta autónoma y eficiente, capaz de generar beneficios de manera continua y automática, con un coste de mantenimiento y de inversión inicial prácticamente nulos.

Los objetivos que se persiguen que tenga Diana, de cara al usuario y al desarrollador al acabar el proyecto son:

1. Hacer accesible al público estándar, sin grandes conocimientos de internet o de portales de compra online, la compra de productos a través de ellos.
2. Hacer una herramienta con un interfaz, tanto en inglés, para permitir su internacionalización, como en español, para aquellos usuarios hispanohablantes sin dominio de la lengua inglesa. De tal manera que gran parte del mundo tenga acceso a esta herramienta.

3. Ofrecer una herramienta fácilmente accesible, que no necesite ordenador, sino que se pueda acceder a través de cualquier sistema con Telegram (como un móvil o tableta).
4. Desarrollar un sistema fácil de entender, actualizar y mejorar, para más tarde introducir nuevas funcionalidades, páginas web o idiomas.

4.3 METODOLOGÍA

Para el desarrollo de este proyecto había dos posibles vías:

1. La primera opción y a la cual por la que ha sido desarrollado este proyecto es el desarrollo autónomo del bot, por parte de cualquier persona, partiendo de una base nula y aprendiendo conforme se desarrolla el proyecto, hasta tener una versión que puede incluso ser expandida o tener cambios en función de las ideas de cada persona.
2. La segunda opción que trataría de contratar a un informático o a alguien con formación previa en el tema, con lo que se ahorraría el tiempo de aprendizaje y únicamente habría que desarrollar los códigos del bot.

El desarrollo de este proyecto ha sido en base a la primera opción, por lo que, si se desease ir por la segunda vía y contratar a alguien con experiencia, únicamente se debe contratar a alguien con experiencia en Python, uso de bases de datos no relacionales (MongoDB), scraping (obtención de información a partir de páginas webs y ficheros JSON) y con conocimiento de la API de Telegram.

A continuación, se mostrará el orden de las actividades a realizar durante el proyecto.

4.3.1 APRENDIZAJE DE PYTHON

El aprendizaje de Python 3.8 es algo que se comenzará al principio del proyecto, pero se continuará a lo largo del proyecto debido a las diferentes librerías que se utilizarán y que habrá que aprender a usar.

Para su aprendizaje inicial se ha realizado un curso gratuito de Python a través de la plataforma (SoloLearn Inc., 2019), en ese curso se explica lo básico de Python y varios módulos muy útiles tanto para este proyecto como para posteriores. También se ha accedido a algunos libros para depurar conceptos, entre ellos Python Crash Course (Matthes, 2015) ha sido de los más útiles para entender algunos conceptos mejor, el resto de los libros estarán en el apartado de referencias. La mayoría de este material es en inglés y requiere de un nivel bueno de lectura en inglés. Sin embargo, aunque los libros son una ayuda, no son indispensables, ya que se puede entender muy bien los conceptos básicos usando la web de SoloLearn (SoloLearn Inc., 2019) y buscando las dudas en los foros de Stackoverflow (Stackoverflow, 2020).

El tiempo requerido para aprender lo básico con lo que iniciar el proyecto es de 10 horas ya que se tenían conocimientos previos de programación, pero podría llevar algo más comprender los conceptos si no se tienen bases previas. Una vez cubiertos los conocimientos básicos se puede comenzar con la siguiente parte.

No se requiere ni se recomienda un aprendizaje muy profundo de Python, ya que aprender módulos que no se vayan a usar en el proyecto únicamente requerirá más tiempo, sin aportar muchos beneficios a este. El aprendizaje de Python se seguirá desarrollando progresivamente durante todo el proyecto, junto con la introducción de nuevas librerías de Python. Durante estas siguientes etapas el aprendizaje del uso de las librerías utilizadas en Python ha sido de unas 20 horas.

El total de horas por tanto para esta primera parte del proyecto es de 30 horas.

4.3.2 APRENDIZAJE DE LA API DE TELEGRAM

En este apartado, al igual que en el anterior de Python es importante saber que no se necesita un conocimiento y estudio absoluto de toda la API de Telegram, únicamente se utilizará una parte de la API, no toda. Y tampoco es necesario aprenderla de memoria, sino que lo más recomendable es trabajar con la documentación de la API abierta en otra pestaña en el navegador web.

Para el entendimiento de esta parte y su aprendizaje se ha utilizado la web (Telegram, 2020), allí está desarrollada toda la documentación necesaria para desarrollar el bot. Aunque no se ha utilizado todo lo que hay en la web, si se requiere hacer una lectura exhaustiva de todo el documento (a pesar de que es bastante amplio) para entender correctamente el funcionamiento de esta API, y de todo lo que puede enviar y recibir.

El tiempo requerido para aprender de la API de Telegram es de 20 horas, en las cuales se han de aprender también los métodos de InlineQuery utilizados en el código para crear los menús.

4.3.3 CREACIÓN Y PRUEBA DE LAS COMUNICACIONES SERVIDOR-TELEGRAM

Una vez comprendido lo básico tanto del lenguaje de programación, como de la API de Telegram, se puede comenzar con el desarrollo del módulo de comunicación entre el servidor y Telegram. Aquí únicamente se busca que se sea capaz de recibir y enviar mensajes básicos por Telegram.

La idea de esta parte es distinguir la teoría de la práctica, ya que es aquí donde se ponen en práctica los conocimientos aprendidos durante las dos partes anteriores. Permitiendo así comprobar que efectivamente se ha entendido lo estudiado y que se puede empezar a trabajar, ya que en partes posteriores (Desarrollo del código y comprobación de errores) se requerirá de mayor destreza y comprehensión de lo estudiado.

Para el desarrollo de estas primeras comunicaciones ha sido especialmente útil el vídeo de Sourav Johar (Johar, 2019), en el cual desarrolla otro bot con aplicaciones diferentes, pero explica muy bien la organización y estructura de la información que se envía y recibe mediante la API.

El tiempo requerido para este primer código es de 20 horas para crear el bot en botfather y crear el código para su funcionamiento en PyCharm.

4.3.4 APRENDIZAJE DE SCRAPING

En este apartado se realizará el aprendizaje de las técnicas de Scraping, que en este proyecto se han realizado a mano, utilizando las librerías bs4 (Richardson, 2020) y requests. Al ser un proceso que se hará a mano en cada web, y que requerirá muchas pruebas para detectar los errores por los diferentes tipos de formatos que aparecerán, el aprendizaje debe ser a conciencia y debe entenderse especialmente bien la estructura de las páginas webs, y tanto como devuelve la información la librería requests, como la organización de esta información por la librería bs4 (Richardson, 2020).

Para esta parte, lo más importante es leer cuidadosamente la documentación de ambas librerías (tanto bs4 (Richardson, 2020) como requests), y los ejemplos que incluyen ambas. También ha resultado interesante el vídeo de Dev Ed (Ed, 2020), en el cual explica de manera muy básica y poco profunda el uso de ambas librerías y explica cómo usarlas para el Scraping web a través del ejemplo de la web de Amazon. Esto fue extrapolado y desarrollado con las demás webs, aunque por supuesto de manera mucho más profunda y desarrollada.

El tiempo de aprendizaje requerido para esta parte es de 25 horas.

4.3.5 APRENDIZAJE DE MONGODB

Para este punto se necesitará descargar MongoDBCompassCommunity (MongoDB, 2020). Tras eso, habrá que descargar la librería pymongo (MongoDB, 2020) para poder usar la base de datos MongoDB, en PyCharm. Tal y como se ha explicado previamente, la descarga de la aplicación MongoDBCompassCommunity, está explicada en ANEXO I: Instalación de programas: **¡Error! No se encuentra el origen de la referencia.**.

Para aprender a usar MongoDB y saber cómo se organizan los datos habrá que ver los tutoriales básicos disponibles en la web de MongoDB (Bradshaw, 2020). Toda la documentación de MongoDB está disponible en (MongoDB, 2020). Tras estos tutoriales que introducen la herramienta de MongoDB, llega el momento de comenzar a usar la librería de MongoDB para Python. Para ello hay que empezar por usar la documentación de la librería y sus ejemplos (MongoDB, 2020). Una vez leídos y entendidos los documentos, en la web

de MongoDB se puede acceder a un muy buen ejemplo del uso de la librería de MongoDB para Python (Walters, 2020).

El tiempo para el aprendizaje de esta herramienta es de 15 horas. Al igual que con las herramientas anteriores, MongoDB es una herramienta profesional, con muchas aplicaciones y usos, por lo que tiene muchas opciones y muchos tutoriales en la web. Sin embargo, solo se necesitará un conocimiento básico de la herramienta para poder incorporarla al proyecto.

4.3.6 DESARROLLO DEL CÓDIGO Y COMPROBACIÓN DE ERRORES

En este apartado, una vez se ha entendido y estudiado todo lo que se necesitará para el desarrollo completo del bot, se comenzará con él. Aquí se desarrollará todo el bot y sus diferentes módulos.

Para esta parte hay que apoyarse en todo la aprendido anteriormente, durante el desarrollo de esta parte se accede varias veces a los documentos previos estudiados y a la documentación que aparece en la bibliografía.

El tiempo dedicado a esta parte es el mayor de todo el proyecto, ya que será donde más código haya que escribir y donde se junte todo lo aprendido previamente. Además, habrá que depurar numerosas veces para encontrar los fallos y errores que aparecerán. Por ello el tiempo total es de 90 horas.

4.3.7 AMPLIACIÓN DEL PROYECTO CON NUEVAS DIVISAS Y MENSAJES DE EMAIL

En este último apartado se introducirá alguna característica extra al bot que se crea conveniente. En este proyecto se ha introducido el contacto por email con el administrador en caso de dudas del usuario o error en la ejecución, a través de la librería smtplib (Python Software Foundation, 2020); y la posibilidad de cambio de divisa a través de la librería currency_converter (Prengère, 2020).

Para el desarrollo de este último apartado se usó la documentación de ambas librerías y sus ejemplos, y concretamente para la librería smtplib (Python Software Foundation, 2020) encargada del correo, se utilizaron los conceptos aprendidos en el vídeo: Build A Python App That Tracks Amazon Prices (Ed, 2020), en el cual se explica brevemente el uso de esta librería.

En caso de desarrollar nuevamente este proyecto, o de ampliarlo, se anima al desarrollador a aplicar cualquier nueva característica o herramienta que vea útil o estime necesaria, y que la desarrolle con los conocimientos que reciba de este proyecto.

El tiempo de este apartado depende de la cantidad de herramientas nuevas desarrolladas, en este caso, al desarrollar el email y el convertidor de divisas, el tiempo es de 10 horas.

4.4 PLANIFICACIÓN TEMPORAL Y ESTIMACIÓN ECONÓMICA

El tiempo en el cual se ha desarrollado el proyecto se puede ver el diagrama de Gantt de la Ilustración 8 - Diagrama de Gantt del cronograma. En el cronograma se puede ver que, aunque las actividades son prácticamente secuenciales, algunas se superponen ligeramente entre ellas, eso es debido a que es óptimo empezar a trabajar en algunas partes prácticas del proyecto mientras se continua estudiando la teoría, ya que eso favorece a entender mejor las herramientas que se han utilizado (por supuesto para eso hace falta saber lo básico, es por eso que siempre hay un espacio de tiempo al empezar una nueva actividad antes de empezar la siguiente).

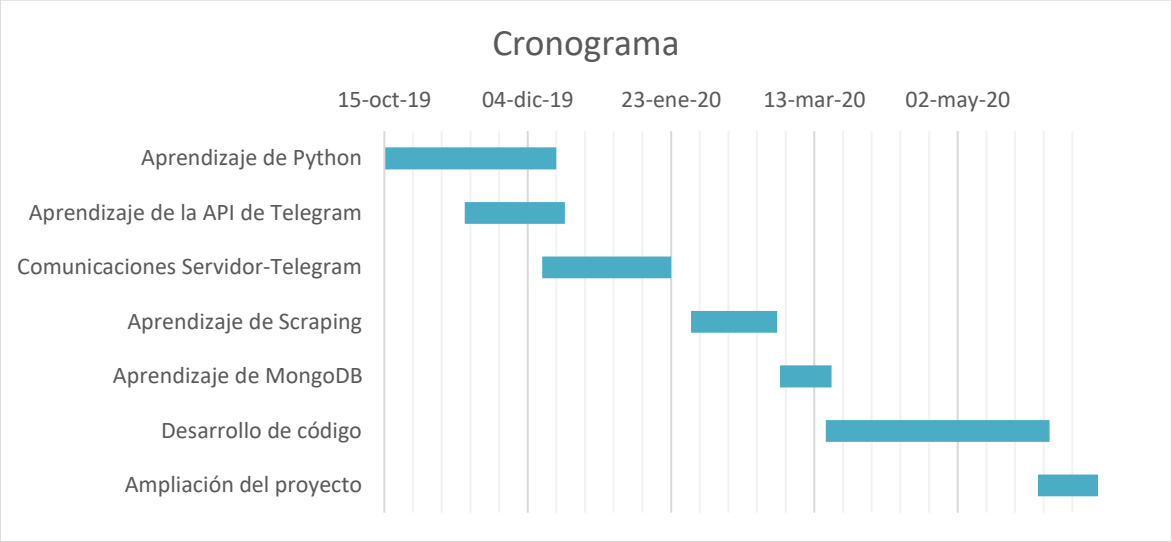

Ilustración 8 - Diagrama de Gantt del cronograma

A continuación, en la Ilustración 9 - Horas invertidas en cada actividad, se puede ver la cantidad de horas invertidas en cada actividad realizada durante el proyecto. Como era de esperar, la mayor parte del tiempo ha sido invertido durante la parte del desarrollo de código y comprobación de errores.

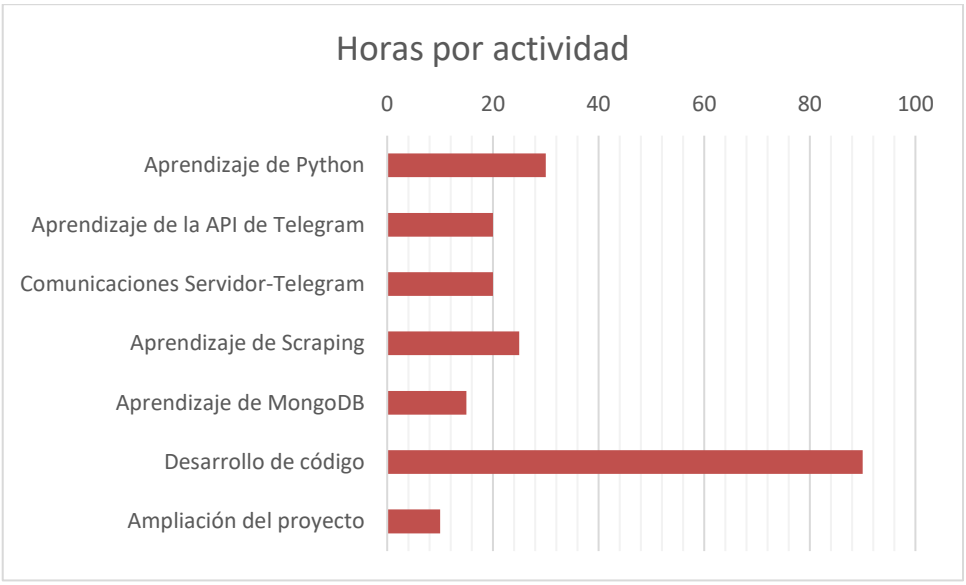

Ilustración 9 - Horas invertidas en cada actividad

El total de horas para la realización del proyecto es de 210, contando los periodos de aprendizaje de las herramientas. El saldo medio en España de un informático es de 14.11 €/h (indeed, 2020). Por lo que el coste total de desarrollo del proyecto es de 2.963 €. En este coste únicamente se recoge el salario del desarrollador, los posibles costes posteriores de mantenimiento de un servidor son prácticamente nulos, y no se necesitan para desarrollar el proyecto, ya que todas las pruebas realizadas han sido hechas mediante un servidor físico en el ordenador del desarrollador.

4.5 OBJETIVOS DE LA ONU PARA EL DESARROLLO SOSTENIBLE Y BENEFICIOS MEDIOAMBIENTALES

Otra de las grandes facetas que actualmente se debe tener en cuenta, es el compromiso medioambiental, el cual tiene tanta importancia como el compromiso social o los beneficios económicos. Es por ello por lo que se ha desarrollado este proyecto, gracias al cual se ofrece una alternativa a los portales de venta físicos tradicionales, utilizando un programa autónomo electrónico, que requiere de un mantenimiento y recursos para su desarrollo prácticamente nulos. Esto se ha hecho con el objetivo de ahorrar el máximo número de recursos, para así reducir los residuos generados por la sociedad, e intentar ayudar de alguna manera a que todo el mundo avance por un camino más limpio, y con un uso de los recursos lo más óptimo posible.

Con esto se espera que se reduzcan el número de traslados de mercancías, ya que los productos irían directamente del almacén del vendedor a la casa del comprador, evitando así el traslado a tiendas y el consumo de recursos de estas. También se intenta reducir el consumismo, ya que al cambiar la actividad de ir a comprar a un lugar público con escaparates y tiendas, por una compra de lo necesario desde casa, se reduce el número de estímulos al que estamos expuestos que nos incitan a comprar, reduciendo así los recursos que se usan del planeta, y ayudando así a crear un mundo mejor.

Capítulo 5. DESARROLLO DE DIANA

Las fases de desarrollo del sistema ya han sido explicadas en el apartado Metodología, es por eso que este capítulo se centrará en la explicación del funcionamiento y de cómo se han hecho las diferentes partes del código de Diana. También se explicará la estructura de funcionamiento de Diana en su conjunto, y, por otro lado, la de su servidor específicamente.

También se explicará el funcionamiento de su interfaz. Para ver más en profundidad su código, se puede ver todo en el ANEXO II: Código.

5.1 ESQUEMA GENERAL

El funcionamiento de Diana puede simplificarse en el intercambio de datos entre cuatro entidades: el usuario, Telegram, el servidor y la base de datos. Este intercambio de datos se da siempre de la misma manera: primero el usuario envía un mensaje a Diana mediante Telegram; una vez ese mensaje llega a los servidores de Telegram, se envía ese mensaje con la información del usuario que lo ha enviado al servidor, luego el servidor analiza el mensaje y accede a la base datos para registrar al usuario; una vez que el servidor ha terminado de procesar el mensaje y de cumplir las funciones necesarias, guarda los datos necesarios en la base de datos y envía la respuesta de vuelta por Telegram, que a su vez se la manda al usuario.

El servidor por otro lado tiene también acceso a funciones que necesita para analizar la información: el email, un intercambiador de divisas, las webs a las que realiza el Scraping, y un archivo .cfg del cual obtiene las contraseñas del correo y del bot de Telegram.

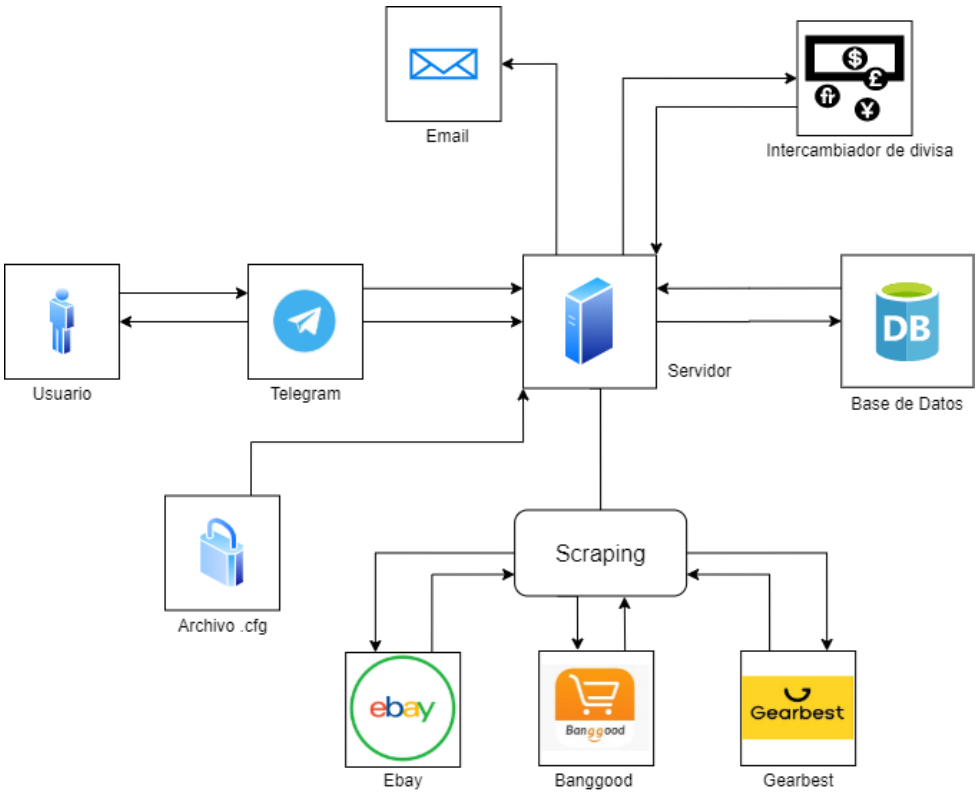

Ilustración 10 - Esquema de Diana

En la Ilustración 10 - Esquema de Diana, se pueden ver estas funciones descritas previamente. Este es un esquema general que sirve de introducción para entender la ruta de datos de Diana. Pero es necesario posteriormente de un esquema más en profundidad del servidor que se desarrollará en el apartado Esquema del Servidor.

A continuación, se explicará cada elemento del esquema.

5.1.1 USUARIO

Es todo aquel usuario registrado en la aplicación de Telegram, para ello se requiere de un teléfono y un nombre de usuario, cada usuario en Telegram tiene asociado un ID de usuario. Este ID será el ID que se utilizará en la base de datos para identificar a cada usuario y poder guardar la información de los productos que desea.

5.1.2 TELEGRAM

Los servidores de Telegram son los que harán de puente entre el servidor y el usuario. Estos enviarán información al servidor cada vez que haya un nuevo mensaje o información que enviar. Sin embargo, no almacenarán ni procesarán ningún tipo de dato, únicamente los enviarán directamente a el servidor.

5.1.3 SERVIDOR

Será el cerebro encargado de procesar toda la información, sin embargo, aunque inicialmente sí se pensaba que almacenase los datos el mismo servidor, finalmente se ha decidido que este no almacenará nada, sino que enviará los datos a almacenar a la base de datos.

Desde el servidor se realizan la mayor parte de conexiones y de tareas, entre ellas las relacionadas con el archivo seguro .cfg y tanto con el Scraping, como con el acceso al correo o al intercambiador de divisas. Todas estas tareas serán monitorizadas individualmente desde el servidor.

5.1.4 BASE DE DATOS

La base de datos que se va a utilizar es MongoDB, estará únicamente comunicada con el servidor, de manera que se aísle lo máximo posible del resto de funciones, a fin de mantener la información segura.

Durante este proyecto la base de datos se ha colocado en el mismo ordenador del servidor, pero podría haberse puesto en un servidor online y haberse accedido a él por medio de internet. Pero para un primer proyecto se ha decidido que sea físico para disminuir su complejidad.

5.1.5 EMAIL

Gracias a la librería smtplib se puede acceder desde el servidor al correo. Este envío de datos es unidireccional, ya que el servidor no recibe información de los correos, sino que únicamente se dedica a mandarlos.

5.1.6 INTERCAMBIADOR DE DIVISAS

Para este módulo también se utiliza una librería específica de Python llamada currency_converter. Esta envía una cifra y la moneda inicial y la final, recibiendo así la cifra en la divisa final.

5.1.7 SCRAPING

Aquí se agrupan tanto en el Scraping de Ebay, como el de Banggood y Gearbest. Aunque dentro del servidor, cada una de las webs tiene una función diferente para poder hacer el Scraping.

5.1.8 ARCHIVO .CFG

Es un archivo seguro en el cual se guardan previamente las contraseñas del bot y del correo. También se puede usar para guardar otra información y protegerla mejor que en el servidor.

5.2 ESQUEMA DEL SERVIDOR

En este punto se realizará un esquema de las funciones que existen dentro del servidor y como se relacionan entre ellas: quién llama a quién, y para qué sirve cada una.

El funcionamiento interno del servidor trata de un main llamado Server. Desde el main se ejecutan y llaman a otras funciones. El esquema puede parecer un poco enrevesado, pero se explicará cada elemento para que quede más claro.

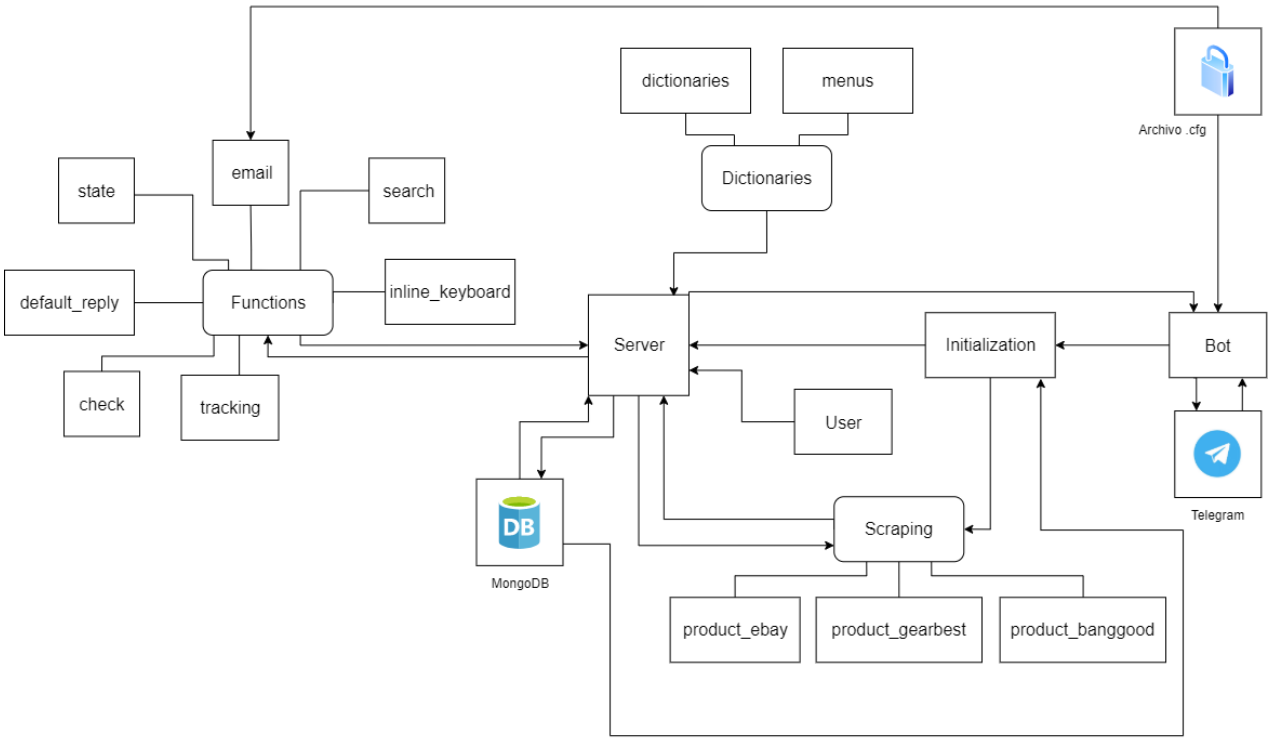

Ilustración 11 - Esquema del servidor

A continuación, se explicará cada elemento del esquema. Los códigos de cada elemento están en el ANEXO II: Código. Este código está comentado para entenderlo correctamente.

5.3 FUNCIONES

5.3.1 SERVER

Es el main del programa, desde allí se llama a Functions (Functions es la carpeta de funciones que se llaman desde el server) y se interactúa con ellas; de User y de Dictionaries se recibe la información de los diferentes diccionarios para los menús, y de la estructura de los usuarios; a través del archivo bot, se interactúa con los servidores de Telegram; a través de Scraping (Scraping es la carpeta en la que están todos los productos de Scraping) se accede a las webs deseadas; y también se tiene enlace directo con la base de datos MongoDB.

Su funcionamiento está basado en dos máquinas de estados: la primera sirve para ver si se ha llamado a algún comando específico de los disponibles en Telegram, si es así entra en

esta primera máquina de estados y no en la segunda; y la segunda es una máquina de estados que se entra si ya habíamos usado previamente en un comando, y sirve para entrar en una parte de la respuesta (por ejemplo si hemos seleccionado /language en el mensaje anterior, en el siguiente mensaje entraremos en la parte de la máquina de estados correspondiente a haber seleccionado /language previamente). Toda esta máquina de estados se va guardando en la base de datos, para saber en qué parte de ella está cada usuario.

También cada vez que entra un mensaje, revisa si es de un usuario que existía previamente en la base de datos, o si por el contrario es nuevo, le crea un perfil a ese nuevo usuario dentro de la base de datos. Esto se hace nada más recibir una update (un nuevo mensaje), al comienzo del main.

Para enviar el mensaje en inglés o español, se almacena primero el idioma de cada usuario en la base de datos, y cada vez que va a enviar un mensaje, se decide con un if en función de si el idioma del usuario es inglés o español.

Otro elemento que abunda en el main son los try:, except:, que se usan para comprobar si hay un error, y enviar así un mensaje al usuario explicando a que se puede haber debido el error. En el caso de que no sea un error del usuario, se envía un mensaje por correo al administrador para que revise el error.

Para ver y entender el código, se puede acceder al ANEXO II: Código: server. Allí está todo el código comentado.

5.3.2 INITIALIZATION

Aquí se inicializan la base de datos, el bot de Telegram y el convertidor de divisas, para luego usarlos dentro del main.

Para ver y entender el código, se puede acceder al ANEXO II: Código: initialization. Allí está todo el código comentado.

5.3.3 BOT

Aquí se define la clase telegram_bot, esta clase sirve para recibir y enviar mensajes por la plataforma de Telegram a través del bot, son las comunicaciones entre el servidor y Telegram. Tiene también una función para leer la contraseña del bot del archivo seguro .cfg. Es un módulo clave dentro del server, ya que se está usando continuamente desde el server para enviar mensajes al usuario.

Para ver y entender el código, se puede acceder al ANEXO II: Código: bot. Allí está todo el código comentado.

5.3.4 USER

Aquí se define la clase User, esta clase estructura la información que se recibe de Telegram en un JSON, para poder mandarla a la base de datos. Tiene diferentes estructuras según el tipo de dato que se envíe, a priori en este proyecto solo interesan los mensajes de texto y las respuestas de los inline keyboard, pero se han dejado estructuras extras (como de vídeo y de gift) con el fin de usarlas en una versión futura del proyecto que analice más tipos de mensajes.

Para ver y entender el código, se puede acceder al ANEXO II: Código: user. Allí está todo el código comentado.

5.3.5 DICTIONARIES

Aquí simplemente se almacenan los diccionarios de Python que se usarán más tarde en el server.

Para ver y entender el código, se puede acceder al ANEXO II: Código: dictionaries. Allí está todo el código comentado.

5.3.6 MENUS

Aquí simplemente se almacenan los menús para los inline keyboard de Telegram que se usarán más tarde en el server.

Para ver y entender el código, se puede acceder al ANEXO II: Código: menus. Allí está todo el código comentado.

5.3.7 CHECK

Esta función sirve para comprobar si alguno de los productos que siguen ha llegado a su precio deseado, si es así envía un mensaje, sino no envía nada.

Para ver y entender el código, se puede acceder al ANEXO II: Código: check. Allí está todo el código comentado.

5.3.8 DEFAULT_REPLY

Simplemente envía una respuesta por defecto al usuario.

Para ver y entender el código, se puede acceder al ANEXO II: Código: default_reply. Allí está todo el código comentado.

5.3.9 EMAIL

Es el módulo encargado de enviar un email tanto en el caso de que lo pida un usuario, como de aviso del servidor por un error en el mismo. Incluye la librería email con MIMEText para poder enviar caracteres no-ASCII como la ñ.

Para ver y entender el código, se puede acceder al ANEXO II: Código: email. Allí está todo el código comentado.

5.3.10 INLINE_KEYBOARD

Es el módulo encargado de la creación de los inline keyboard, estos inline keyboard son los menús flotantes que aparecen en la interfaz de Telegram. Para su creación únicamente hace falta un diccionario, y la función lo pone en formato JSON para enviarlo más tarde a Telegram.

Para ver y entender el código, se puede acceder al ANEXO II: Código: inline_keyboard. Allí está todo el código comentado.

5.3.11 SEARCH

Esta es la función para buscar un producto deseado en las webs disponibles. Dentro de ella llama a los módulos de Scraping de cada web y al final envía el producto que encuentre a mejor precio, si no encuentra ninguno manda un mensaje predeterminado.

Para ver y entender el código, se puede acceder al ANEXO II: Código: search. Allí está todo el código comentado.

5.3.12 STATE

Esta función se encarga de actualizar la máquina de estados en la base de datos, para así ir guardando el estado en el que estamos en cada momento. Nunca se puede estar en dos estados a la vez, por lo que cada vez que la llamamos ponemos todos los estados a 0 menos el que se está.

Para ver y entender el código, se puede acceder al ANEXO II: Código: stateinline_keyboard. Allí está todo el código comentado.

5.3.13 TRACKING

Esta función se encarga de obtener todos los usuarios que tengan una lista de tracking, y de comprobar si alguno de los elementos de sus listas se ha puesto al precio deseado, llamando para ello a la función check.

Para ver y entender el código, se puede acceder al ANEXO II: Código: tracking. Allí está todo el código comentado.

5.4 INTERFAZ

La idea de Diana es que sea un bot muy fácil de entender y de utilizar para todos los usuarios, pero que a su vez sea muy completo y tenga varias funciones. Con esta idea en mente se ha desarrollado un interfaz que responde y explica cada mensaje o duda que tiene el usuario (siempre y cuando se utilicen los comandos).

La primera pantalla que aparece antes de iniciar el bot es la de la Ilustración 12 - Inicio de Diana. Como se puede ver, no hay mucha complejidad en el tema de iniciar el bot, únicamente se debe hacer click en el botón de iniciar.

Ilustración 12 - Inicio de Diana

Lo siguiente que veremos una vez iniciado el bot es una breve introducción de los comandos, esta descripción de los comandos estará siempre disponible usando el comando /help, por si acaso se necesita recordar alguno de ellos.

Ilustración 13 - Comandos de Diana

No obstante, para hacerlo incluso más fácil, si se pulsa el botón / a la derecha de donde se escriben los mensajes, se despliega la lista de comandos con una breve explicación en inglés de cada uno de ellos, esto se puede observar en la Ilustración 14 - Funciones del Bot. Se explican en inglés porque cuando se despliegan pulsando / la explicación de cada uno es fija y no se puede cambiar de idioma, sin embargo, si usamos el comando /help, aparecerán todos los comandos explicados en el idioma del usuario, ya que es un mensaje que se envía desde el servidor de Python, y no un interfaz estático de Telegram.

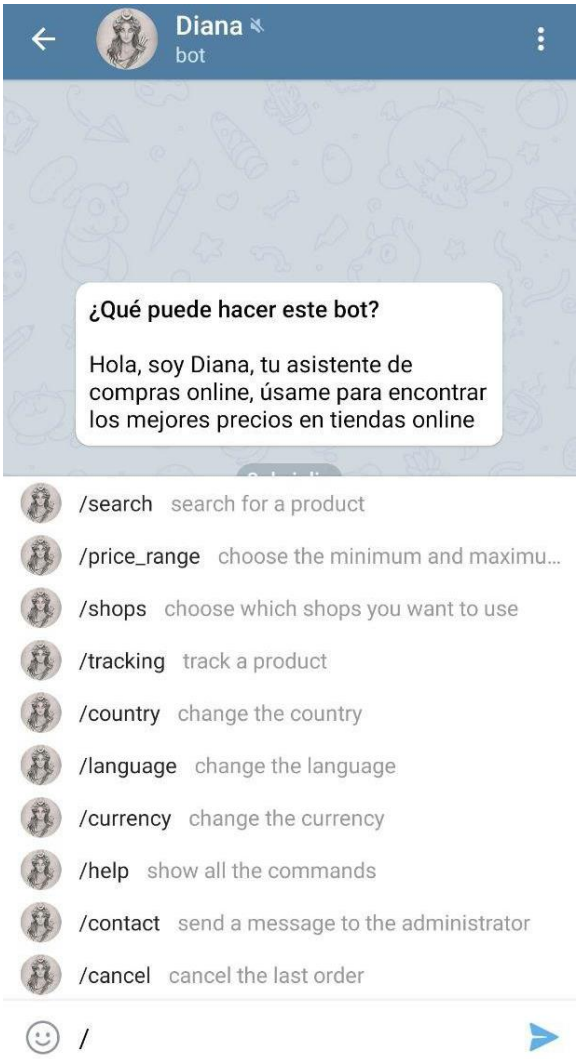

Ilustración 14 - Funciones del Bot

Como ya se ha explicado antes, Diana está hecha para responder a cualquier duda del usuario explicando que hacer lo mejor posible. Por ejemplo, en la Ilustración 15 - Dudas de usuario, Diana detecta que lo que le han mandado no es un comando, y como no tiene ningún otro comando pidiendo algo específico previamente, envía el mensaje predeterminado y le pide al usuario que la utilice por medio de los comandos, ya que no responde al lenguaje natural. Además, añade que puede usar el comando /help para obtener así la lista de comandos explicados.

Ilustración 15 - Dudas de usuario

A continuación, el usuario decide que quiere cambiar de idioma al inglés, y a través del comando /language abre el menú de idiomas y selecciona con el panel táctil el inglés. Tras eso, Diana actualiza en la base de datos el idioma del usuario e inmediatamente comienza a usar el inglés, enviando un mensaje de confirmación ya en inglés, y enviando la respuesta predeterminada en inglés también. Estos pasos se pueden observar en la Ilustración 16 - Cambio de idioma a inglés. Cabe señalar, que también desde el ordenador se puede acceder al menú de idiomas haciendo click con el ratón en la opción que se desee. Diana también está preparada para recibir el mensaje escrito y no es necesario hacer click en la opción, también se puede escribir directamente, aunque lógicamente es mucho más rápido utilizar los menús flotantes. En la Ilustración 17 - Cambio de idioma escrito se puede ver como tras

cambiar al inglés, el usuario vuelve al español, pero escribiendo Español tras usar el comando idiomas.

Ilustración 16 - Cambio de idioma a inglés

Ilustración 17 - Cambio de idioma escrito

Todos los comandos están explicados en Telegram, y su desarrollo está explicado en el código (ANEXO II: Código). A continuación, se verán imágenes de el mismo usuario usando el comando /tracking. Al principio el usuario que no tiene productos en su lista de seguimiento y tras pulsar My Items, Diana le explica que no tiene productos en su lista, entonces el usuario decide añadir uno nuevo y pulsa Add Item, tras eso Diana le pregunta el nombre y el precio del producto que quiere que busque y lo añade. Tras eso el usuario decide que quiere eliminar ese producto de su lista y borra el Iphone X, para comprobar que efectivamente está borrado, hace click en My Items y Diana efectivamente responde con que no hay ningún producto en su lista de seguimiento. Depués el usuario decide añadir un nuevo producto y añade el Xiaomi Redmi Note 8 a su lista, a continuación selecciona la opción de

Check Items para ver si hay alguno disponible por el precio que ha buscado, y efectivamente encuentra uno en Banggood por 184.24 €. Ahora simplemente debe hacer click en el link del producto para acceder a él.

El resto de comandos siguen la misma mecánica de acciones secuenciales y menús flotantes.

Ilustración 18 - Tracking paso 1

Ilustración 19 - Tracking paso 2

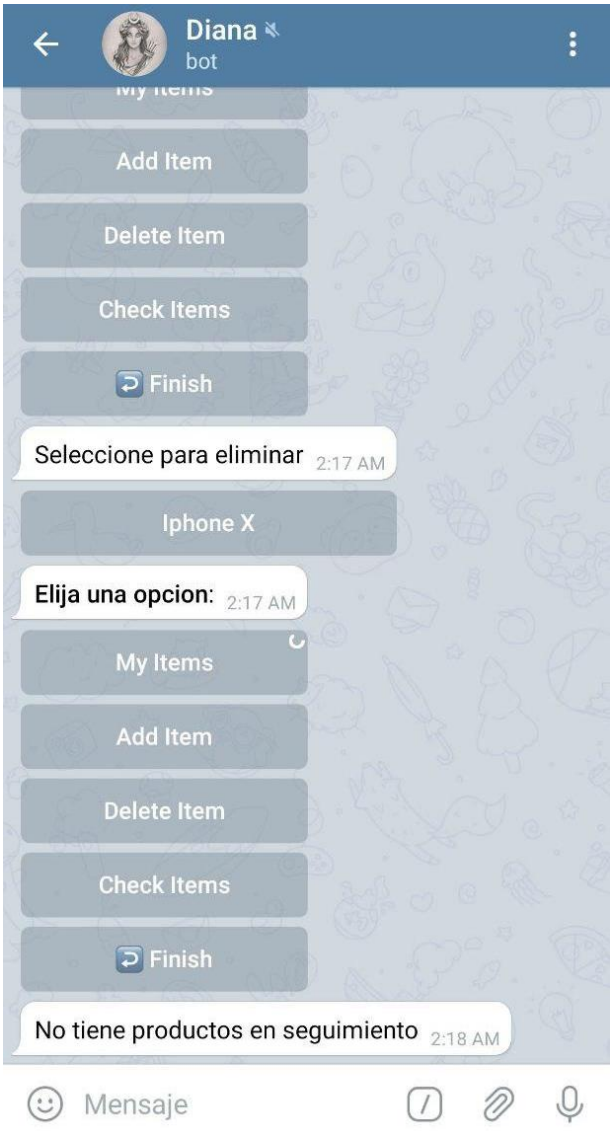

Ilustración 20 - Tracking paso 3

Ilustración 21 - Tracking paso 4

Ilustración 22 - Tracking paso 5

Capítulo 6. ANÁLISIS DE RESULTADOS

6.1 COMUNICACIONES SERVIDOR-TELEGRAM

Este es una de las partes que mejor resultado ha dado, gracias al desarrollo propio de un sistema de comunicación entre el servidor del bot y Telegram, se ha optimizado el tiempo para que se reduzca al máximo al ejecutar el código. Estando optimizado el código del bot, la velocidad de envío de información depende finalmente de la velocidad de la red. Aunque según las pruebas hechas con varios usuarios, el volumen de usuarios no parece afectar a la velocidad de envío de datos entre el servidor y Telegram, lo que quiere decir que cada intercambio de datos con cada individuo toma muy poco tiempo.

Tanto el uso de una librería de Telegram como el uso de las funciones descritas en este proyecto son ambas buenas opciones, para elegir entre una u otra se debe pensar principalmente en el volumen de usuarios que va a mover el bot, ya que, en caso de ser muy alto, se necesitará un código lo más optimizado posible. Pero, en caso de no necesitar una gran optimización, las librerías de Telegram son muy buena opción para utilizar, ya que permiten el uso de varias herramientas de Telegram sin tener que aprenderlas por la API.

6.2 BASE DE DATOS

Tras analizar varias bases de datos y elegir finalmente MongoDB, el resultado ha sido muy satisfactorio. Esto es debido a que el código JSON, que es el utilizado por Telegram para comunicarse con el servidor, es el mismo utilizado por la base de datos para almacenar la información. Haciéndola una herramienta muy útil, fácilmente extrapolable y con una gran variedad de usos para otros proyectos, y con una curva de aprendizaje no muy complicada.

Por otro lado, el tiempo usado para acceder a la base de datos y extraer información es minúsculo y no ralentiza nada las demás tareas. Esto es gracias a que esta base de datos es usada por varias empresas y redes sociales, por lo que está hecha para mover continuamente un gran volumen de intercambios de información.

Sin duda, una de las herramientas más importantes del proyecto, y que es muy importante comprender para poder usarla en futuros proyectos.

En este proyecto el servidor se ha situado en el ordenador que hace de servidor, pero se podría situar de igual manera en la nube para que funcione de manera automática y se pueda acceder desde varios dispositivos. Pero en esta primera versión del bot, se ha optado por resultar lo más simple posible para el desarrollador, con el fin de no requerir tiempo extra. Además, no hay ningún problema con el uso de esta base de datos en el ordenador, ya que cada usuario registrado ocupa muy poco espacio, gracias al uso del formato JSON.

6.3 SCRAPING

Aunque se ha conseguido una versión final de esta parte sin errores y cuyo funcionamiento es bueno. Esta es sin duda la parte que más podría mejorar en una versión siguiente de este proyecto. Principalmente esto es debido a que al realizar el Scraping de forma manual, se han dado multitud de errores debido a los diferentes formatos en los que aparece la información, estos errores han sido parcheados, pero aun así ha requerido demasiado tiempo y esfuerzo. También, al utilizar la librería requests (Reitz, 2020) para obtener la información de la web, se requiere mucho más tiempo que en las otras partes del proyecto para llevar a cabo esta actividad.

Ante esta serie de problemas, las posibles soluciones que existen actualmente son:

- Utilizar la librería scrapy de Python. Se puede utilizar una librería dedicada a la parte de Scraping, y que ya está desarrollada en Python. Esto podría reducir los errores que han aparecido durante el desarrollo del proyecto, aunque no disminuirían en gran medida el tiempo de ejecución, ya que se utiliza la librería requests (Reitz, 2020)

(Reitz, 2020) que es la que hace que se ralentice esta actividad al tener que obtener toda la información de la web de la tienda.

- Utilizar las APIs de las tiendas. Esta es la forma óptima sin duda de utilizar el Scraping, ya que, mediante librerías previamente desarrolladas en Python, es muy fácil acceder a la información de estas webs. El problema de esto radica en que no todas las tiendas online tienen una API desarrollada para estos fines, y las que sí la tienen desarrollada, requieren de una cuenta especial en la tienda y de unos permisos especiales, los cuales, si no se dispone de una web o un blog, es difícil obtener. En el caso de Amazon se requiere incluso una cuenta de AWS, la cual no es nada fácil de utilizar debido a su gran complejidad.

6.4 INTERFAZ DEL BOT

El interfaz desarrollado tanto en inglés como en español ha sido probado con varios usuarios mientras se realizaban las pruebas del bot, y según los usuarios que lo han utilizado es un interfaz muy amigable e intuitivo. Tanto como interactúa el bot, como sus funciones principales, han sido rápidamente comprendidas por usuarios tanto con experiencia media en Telegram, como con nula experiencia en la plataforma.

Según sus respuestas, tanto el comando /help, como la respuesta predeterminada del bot, han ayudado mucho en caso dudas o de no saber cuáles son las funciones del bot.

Los usuarios también han destacado la claridad del bot para explicarse en caso de haber fallos en la forma de escribir comandos u órdenes para el bot, o de simplemente responder a errores producidos por el usuario.

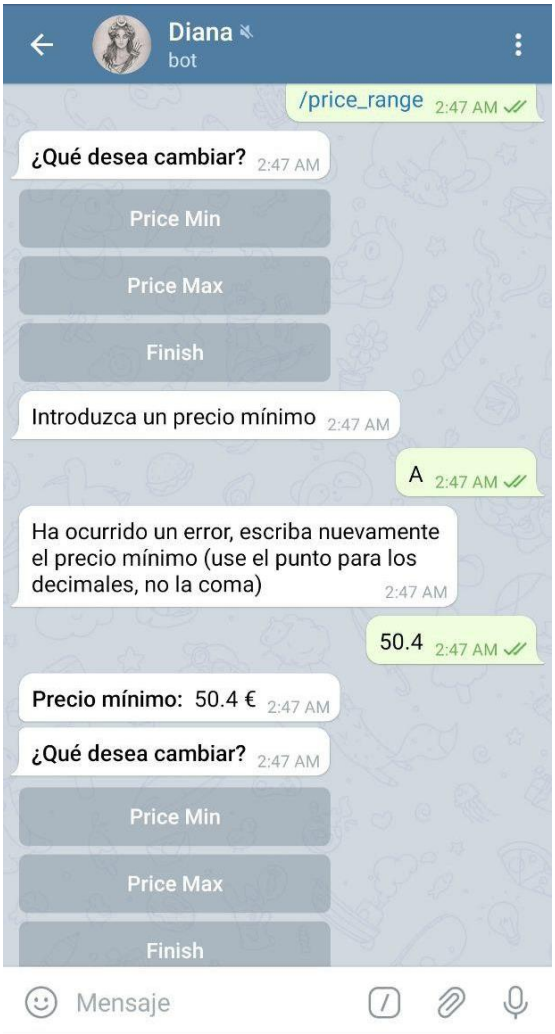

Ilustración 23 - Respuesta del interfaz a errores del usuario

También les ha gustado especialmente la existencia de menús flotantes dentro del bot, lo cual requirió mucha investigación en la API de Telegram y de varias pruebas, debido a la escasa documentación existente y a que resultó imposible encontrar ejemplos de su aplicación a Python en Internet.

Ilustración 24 - Menú flotante tracking

6.5 EMAIL

El email es una de las herramientas que se introdujeron en la etapa de ampliación del bot al final del proyecto. Sin embargo, según los usuarios que lo han utilizado, es una de las herramientas que más les ha llamado la atención, ya que les permite ponerse en contacto con el administrador del bot, para informarle de posibles mejoras o solución de errores.

También ha mostrado ser una herramienta fundamental para el desarrollador, ya que permite a este activar el bot, y que este le envíe un mensaje directo al correo en caso de que deje de funcionar por un error de programación.

Además, aunque esta ha sido una herramienta desarrollada brevemente en el bot, tiene un potencial uso muy amplio, ya que se puede programar el correo para que envíe datos determinados al desarrollador de forma diaria o semanal, en caso de estar interesado en almacenar información del número de usos diarios u otras cuestiones del bot.

Ilustración 25 - Mensaje email

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

En este proyecto se ha logrado con éxito alcanzar todos los objetivos iniciales que se perseguían, además se ha comprobado que efectivamente el diseño del interfaz es intuitivo y fácil de usar para los usuarios. Además de que las opiniones de los usuarios que lo han probado han sido muy positivas, el proyecto también ha suscitado interés en otros usuarios jóvenes a los que se les ha explicado, pero no han entrado dentro de la fase de prueba. Por esto, se tienen grandes expectativas a futuro del proyecto.

Por el lado del aprendizaje y el desarrollo de los conocimientos necesarios para el proyecto, se han podido establecer los periodos y herramientas necesarios para que el desarrollador pueda llevar a cabo este aprendizaje de la manera más completa, clara y rápida posible. Esto ha sido gracias a una investigación profunda (que ha llevado más tiempo del programado debido a que se ha necesitado de la investigación de múltiples fuentes), que ha permitido establecer las mejores herramientas para el proyecto. Además, estas herramientas son todas gratuitas, siguiendo la filosofía de mínimos costes e inversión que se ha seguido durante el desarrollo de todo el proyecto.

Durante el desarrollo del proyecto, han surgido cambios y nuevos desarrollos dentro de este. El primero de ellos es que inicialmente el proyecto se planteaba como una herramienta más general, sin un uso muy específico, pero fue después de realizar preguntas a varios usuarios, que se decidió finalmente, que una herramienta general sería poco práctica, y es por eso que se acabó centrando en las compras online en varias webs. También se planteó inicialmente el uso de otras webs, como Amazon o Aliexpress. Sin embargo, estas dos requieren de unas relaciones especiales con ambas dos empresas y de la creación de una cuenta especial, que no es nada fácil de conseguir. Es por eso por lo que se acabó escogiendo a Ebay, Gearbest y Banggood, que además de ser de las webs más usadas y fiables actualmente, también se pueden usar sin necesidad de usar una cuenta especial.

Cabe destacar también, que inicialmente no se planeaba el uso de una base de datos, y se pensaba hacer todo mediante el almacenamiento de datos en diccionarios de Python. Sin embargo, esto traía dos problemas: el primero que cada vez que se reinicie el servidor todos los datos almacenados en los diccionarios se perderían, a menos que se mandasen a algún otro tipo de archivo de almacenamiento; y el segundo problema, a la hora de acceder a los datos y analizarlos, además de usarlos, el uso de diccionarios ralentiza mucho el proceso y lo limita. Ante estos problemas se decidió el uso de MongoDB, que ha demostrado ser una de las herramientas más prácticas y útiles de este proyecto.

Otra de las ideas que mejor han sido recibidas y que surgió en el desarrollo del proyecto, es el uso de menús flotantes, en un primer momento únicamente se pensaba usar mensajes de texto para comunicarse con el usuario. Sin embargo, fue al usar el bot de Telegram llamado botfather (que se usa para crear el bot en Telegram y modificar su imagen o su estado), que se pensó que igual esta herramienta podría implementarse también a Diana. Y aunque fue complicado encontrar una documentación aplicada a Python de cómo realizar esto, finalmente tras varias pruebas se consiguió encontrar la manera de construir estos menús y de recibir la información al pulsarlos. Al final esta herramienta ha ayudado a dar un interfaz más limpio, sencilla y amigable para el usuario.

Finalmente, la última herramienta que se desarrolló y que fue sin duda las más compleja del proyecto, fue el tracking, esta herramienta se había pensado teóricamente en un principio, pero se le dio forma en las últimas etapas del desarrollo del código del proyecto. Es sin duda la herramienta más útil y que más ha gustado, de la cual dispone Diana. Después de ver sus resultados, se puede afirmar que su uso es más que satisfactorio. Además de permitir al usuario un seguimiento de su producto, permite al desarrollador recopilar información de cuáles son los productos que más busca la gente, y esto podría permitir en el futuro, la creación de una función que sea capaz de recomendar al usuario en base a lo que tiene en su lista de tracking, y a lo que tienen los demás usuarios. Sin duda además de ser una de las herramientas más útiles, también es de las más prometedoras.

7.2 TRABAJOS FUTUROS Y MEJORAS AL PROYECTO

Es importante destacar que el desarrollo de este proyecto es muy personalizable y puede variar en función de los deseos del programador, tanto la interfaz como varias partes del código pueden cambiar en función de los deseos del desarrollador, por lo que en sí cada posible desarrollo de este proyecto debería ser distinto de los otros. Sin embargo, esto no impide que haya varias mejores aplicables al proyecto en rasgos generales. Estas mejoras ya han sido comentadas brevemente en los resultados del trabajo, pero se incidirá de manera más profunda sobre ellas en esta sección.

7.2.1 PYTHON-TELEGRAM-BOT

Python-telegram-bot (Toledo, 2020) es una librería de Python que sirve de alternativa a diseñar todas las comunicaciones entre el Servidor y Telegram de forma manual mediante el uso de la API. Esta librería no es fácil de usar ni está exenta del estudio de la API de Telegram, además se necesita estudiar también la documentación de esta librería para aprender a usarla apropiadamente. Es por eso por lo que inicialmente se decidió realizar la comunicación entre el Servidor y Telegram de forma manual. Sin embargo, una vez se llegó a la parte del proyecto en el cual se necesitaba del uso de menús flotantes, la utilización de esta librería hubiera ayudado a agilizar el desarrollo del bot, ya que hay poca información en Internet, tanto en inglés como en español, de las herramientas más avanzadas de la API de Telegram. Finalmente se logró incorporar la creación de menús flotantes a este proyecto, pero si se va a desarrollar un bot que necesite de herramientas más complejas de Telegram, como por ejemplo InlineQuery o uso y procesado de imágenes u otros datos, y no se quiere perder mucho tiempo en la API de Telegram, es más sencillo comenzar el proyecto usando esta librería de Python, y construir todo el código partiendo de esa librería.

Ilustración 26 - Python-telegram-bot

7.2.2 SCRAPY

Scrapy es una librería de Python dedicada especialmente al Scraping de páginas webs, esta herramienta permite la automatización de los procesos que se han desarrollado manualmente en este proyecto, además disminuye el número de errores producidos. Es necesario realizar un estudio a parte de la propia librería y sus comandos, además del estudio básico sobre el Scraping de páginas web y el uso de la librería requests (Reitz, 2020), que se llevan a cabo durante la etapa de Aprendizaje de Scraping. Sin embargo, esta herramienta permite ahorrar una gran cantidad de tiempo a la hora de programar la parte de Scraping en el bot. Especialmente si hablamos de ampliar el proyecto introduciendo nuevas webs, esta herramienta será imprescindible si se quiere mejorar el rendimiento y velocidad del desarrollador.

Ilustración 27 - Scrapy Python

7.2.3 APIS WEBS

Aquí es donde mayor margen de mejora hay, y donde se debería comenzar a mejorar el proyecto. Durante el proyecto se ha usado un Scraping manual. Sin embargo, algunas webs tienen disponible, al igual que Telegram, su propia API para manejar los datos. El uso de estas APIs aumentaría de manera significativa la velocidad y el número de datos que se pueden procesar, además existen librerías en Python para usar estas APIs, una de ellas es Python-amazon-simple-product-api, la cual sirve para obtener la información de cualquier producto de Amazon. Estas APIs requieren de cuentas especiales, pero sin duda sería un gran paso obtener una de ellas, ya que ahorraría toda la parte de tener que aprender a usar Scraping, y de scrapear de forma manual.

7.2.4 AMPLIACIÓN DE IDIOMAS Y WEBS

La forma más sencilla de mejorar el proyecto es incluir en la interfaz nuevos idiomas y webs a las que acceder. Para esto no se necesita aplicar nuevos conocimientos, sino que se trata de hacer lo mismo que se ha hecho para construir la interfaz en inglés y de introducir más webs a las cuales aplicar las técnicas de Scraping. Para este proyecto inicial es más que suficiente con los idiomas y webs disponibles, pero si este mismo proyecto se quiere aplicar a un país vecino en el cual el idioma oficial sea distinto de los dos desarrollados (por ejemplo, Italia), en ese caso sí sería necesario introducir este idioma. Por la parte de las nuevas webs, se recomienda que se utilice si está disponible la API de estas, y, sino que se acceda a estas mediante la herramienta Scrapy de Python previamente explicada en el apartado 7.2.2 Scrapy. Algunas webs recomendadas que actualmente tienen API desarrollada son Amazon y Aliexpress. Sin embargo, estas dos requieren de una cuenta de afiliados con servicios especiales que es difícil de conseguir, y que para ello es necesario contactar primero con las webs.

Ilustración 28 - Web de Amazon

7.2.5 EMAIL

Esta es una herramienta potencialmente muy útil, tanto para este proyecto como para cualquier otro que necesite de informes periódicos. En este proyecto se usa únicamente para ponerse en contacto con el administrador y para informar de errores que sucedan durante la ejecución del bot. Sin embargo, se puede utilizar para enviar información regular al administrador sobre el número de usuarios que utilizan el bot diariamente o para enviar un correo a los usuarios que lo deseen cuando, por ejemplo, el producto que deseen se encuentre disponible. Las posibilidades de uso de la herramienta del correo son ilimitadas y es un campo donde se pueden realizar ampliaciones importantes del bot.

7.2.6 BASE DE DATOS

La base de datos utilizada, MongoDB, ha demostrado funcionar de manera óptima en el proyecto, debido a que no es excesivamente difícil de aprender a usar, y es una base de datos muy potente. Por ello, aunque existen otras opciones viables y buenas en cuanto a bases de datos, se recomienda ampliar el uso de esta en vez de usar otra. Dicho esto, hay mucho espacio de mejora y ampliación de la base de datos, ya que hay muchos datos que se pueden aprovechar. Por ejemplo, el número de interacciones diarias del bot, para determinar su uso, y si está creciendo o disminuyendo, y cuáles son los días con mayor tráfico en el bot y los

que menos, para realizar mantenimientos del bot esos días, y así importunar al usuario en la menor medida. También hay muchos otros datos que se podrían almacenar y que pueden ser muy útiles, como, por ejemplo, el precio al que se ha encontraba un producto cuando se buscó y su evolución a lo largo del tiempo, para mostrar así a los usuarios una evolución del precio de este a lo largo del tiempo. Esta es una de las aplicaciones más interesantes para ampliar el bot, sin embargo, requerirá un mayor uso de la base de datos y mayor espacio de almacenamiento debido a los miles de productos que habría que analizar.

Capítulo 8. BIBLIOGRAFÍA

[1] Bradshaw, S. (10 de marzo de 2020). *Tutorial MongoDB*. Obtenido de Tutorial MongoDB: https://university.mongodb.com/courses/M001/about

[2] Ed, D. (20 de febrero de 2020). *Youtube: Build A Python App That Tracks Amazon Prices!* Obtenido de Youtube: Build A Python App That Tracks Amazon Prices!: https://www.youtube.com/watch?v=Bg9r_yLk7VY

[3] Escobin, M. (10 de julio de 2020). *Medium.com*. Obtenido de Medium.com: https://medium.com/@chutzpah/telegram-inline-keyboards-using-google-app-script-f0a0550fde26

[4] Forbes. (21 de junio de 2020). *Forbes*. Obtenido de Forbes: https://forbes.es/listas/65224/lista-forbes-2020-de-los-mas-ricos-del-mundo/

[5] indeed. (29 de junio de 2020). *indeed*. Obtenido de indeed: https://es.indeed.com/salaries/informatico-Salaries?period=hourly

[6] itreseller. (10 de julio de 2020). *itreseller*. Obtenido de itreseller: https://www.itreseller.es/en-cifras/2020/01/los-espanoles-gastan-en-sus-compras-online-una-media-de-2098-euros#:~:text=Los%20espa%C3%B1oles%20han%20gastado%20en,eleva%20hasta%20los%202.205%20euros.

[7] JetBrains. (15 de octubre de 2019). *Pycharm*. Obtenido de Pycharm: https://www.jetbrains.com/es-es/pycharm/

[8] Johar, S. (10 de diciembre de 2019). *Youtube: Gangsta: A Telegram Chatbot with Python from scratch*. Obtenido de Youtube: Gangsta: A Telegram Chatbot with Python from scratch: https://www.youtube.com/watch?v=5nhdxpoicW4

[9] Lutz, M. (2013). *Learning Python.* Sebastopol: O'Reilly.

[10] Matthes, E. (2015). *Python Crash Course.* San Francisco: No Starch Press.

[11] MongoDB. (marzo de 13 de 2020). *Manual MongoDB*. Obtenido de Manual MongoDB: https://docs.mongodb.com/manual/

[12] MongoDB. (9 de marzo de 2020). *MongoDBCompassCommunity*. Obtenido de MongoDBCompassCommunity: https://www.mongodb.com/try/download

[13] MongoDB. (12 de marzo de 2020). *pymongo.readthedocs.io*. Obtenido de pymongo.readthedocs.io: https://pymongo.readthedocs.io/en/stable/tutorial.html

[14] Prengère, A. (25 de mayo de 2020). *pypi.org/CurrencyConverter*. Obtenido de pypi.org/CurrencyConverter: https://pypi.org/project/CurrencyConverter/

[15] Python. (14 de julio de 2020). *python descargas*. Obtenido de python descargas: https://www.python.org/downloads/

[16] Python Software Foundation. (2 de noviembre de 2019). *python.org/3/configparser*. Obtenido de python.org/3/configparser: https://docs.python.org/3/library/configparser.html

[17] Python Software Foundation. (30 de octubre de 2019). *python.org/json*. Obtenido de python.org/json: https://docs.python.org/3/library/json.html

[18] Python Software Foundation. (3 de julio de 2020). *python.org/email*. Obtenido de python.org/email: https://docs.python.org/3/library/email.html

[19] Python Software Foundation. (30 de mayo de 2020). *python.org/smtplib*. Obtenido de python.org/smtplib: https://docs.python.org/3/library/smtplib.html

[20] Python Software Foundation. (20 de abril de 2020). *python.org/time*. Obtenido de python.org/time: https://docs.python.org/3/library/time.html

[21] Reitz, K. (2 de febrero de 2020). *requests*. Obtenido de requests: https://requests.readthedocs.io/en/master/

[22] Richardson, L. (5 de febrero de 2020). *Beautiful Soup Documentation*. Obtenido de Beautiful Soup Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[23] Sierra, Á. S. (19 de julio de 2020). *Github/diana*. Obtenido de Github/diana: https://github.com/Diana-bot-TFG/Diana

[24] SoloLearn Inc. (20 de Octubre de 2019). *sololearn*. Obtenido de sololearn: https://www.sololearn.com/

[25] Stackoverflow. (20 de mayo de 2020). *Stackoverflow*. Obtenido de Stackoverflow: https://stackoverflow.com/

[26] Telegram. (9 de julio de 2020). *Telegram Bot API*. Obtenido de Telegram Bot API: https://core.telegram.org/bots/api

[27] Toledo, L. (20 de Marzo de 2020). *python-telegram-bot en github*. Obtenido de python-telegram-bot en github: https://github.com/python-telegram-bot/python-telegram-bot

[28] Walters, R. (18 de marzo de 2020). *mongodb.com*. Obtenido de mongodb.com: https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb

[29] *Wikipedia usuarios telegram.* (14 de julio de 2020). Obtenido de Wikipedia usuarios telegram: https://es.wikipedia.org/wiki/Anexo:Usuarios_de_Telegram

ANEXO I: INSTALACIÓN DE PROGRAMAS

PYTHON 3.8

Python 3.8 es la versión de Python que se utilizará durante todo el proyecto. Es la herramienta más importante, ya que tener instalada otra versión, puede dar errores a la hora de utilizar las librerías.

Este lenguaje de programación es compatible con muchos sistemas operativos, entre ellos: Windows, Mac y Linux.

Para este proyecto se ha empleado la versión Python 3.8.0, que es la que estaba disponible en octubre de 2019. Sin embargo, no debería haber ningún problema con usar una versión posterior dentro de la versión 3.8 de Python. Es por eso por lo que se recomienda seleccionar la última disponible.

Para iniciar su descarga, debemos acceder a la web oficial de Python (https://python.org/downloads/). Una vez en la web, si se tiene un sistema Windows, se puede descargar la última versión haciendo click en el botón de debajo de Download the latest version for Windows, esto descargará la última versión disponible, que si está dentro de la serie de Python 3.8, no debería dar ningún problema con las librerías utilizadas. Todo esto se puede ver en la Ilustración 29 - Web de descarga de Python.

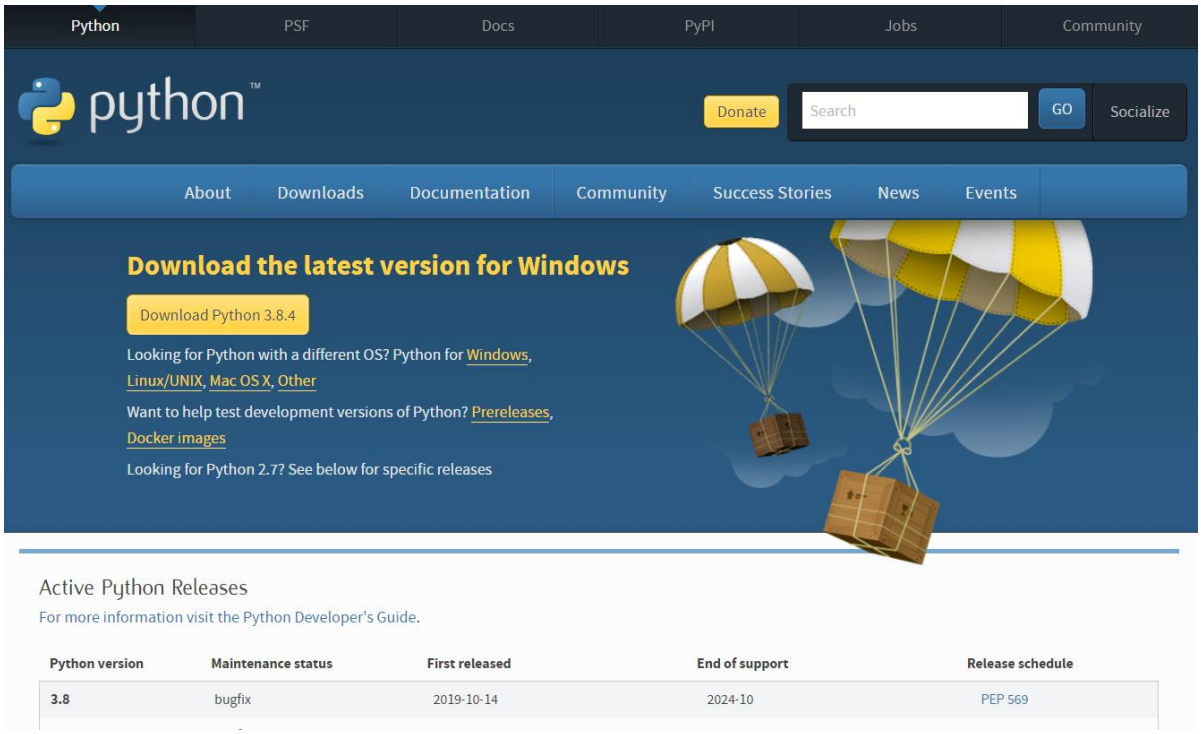

Ilustración 29 - Web de descarga de Python

Si se experimenta algún problema bien durante la instalación o bien después de instalar Python, a la hora de usar las librerías descritas en el capítulo 2.1.5: Librerías de Python, hay que observar la parte donde pone: Looking for Python with a different OS? Y hacer click en el sistema operativo que se necesite. Esto nos llevará a una nueva ventana donde estarán disponibles para su descarga todas las versiones anteriores de Python, para el sistema operativo seleccionado. Si hacemos click en Windows, veremos una ventana como la de la Ilustración 30 - Python para Windows, allí hay que buscar la versión de Python que se desea utilizar, en este caso, tanto la 3.8.0 como la 3.8.1 funcionan correctamente, y se ha de hacer click donde pone Download Windows x86-64 executable installer, en el caso de que se tenga un ordenador de 32 bits (caso poco probable ya que la mayoría son ya de 64 bits), se debe instalar la versión donde pone Download Windows x86 executable installer.

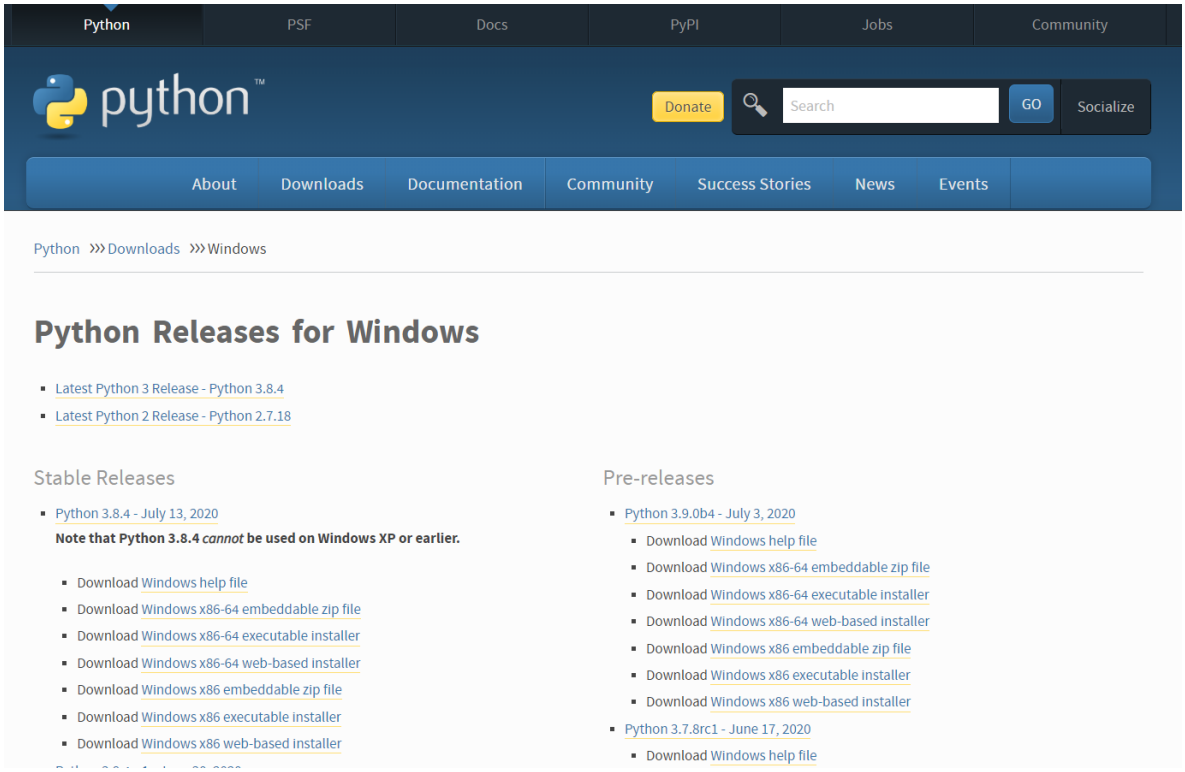

Ilustración 30 - Python para Windows

En el caso de que se vaya a trabajar con un sistema operativo Mac OS X, simplemente se deberá hacer click en Mac OS X en vez de en Windows, y nos abrirá otra pestaña distinta como la de la Ilustración 31 - Python para Mac, allí hay que hacer click en Download macOS 64-bit installer, en la versión que se desee (3.8.0 o 3.8.1) y esto nos descargará un archivo de instalación para Mac.

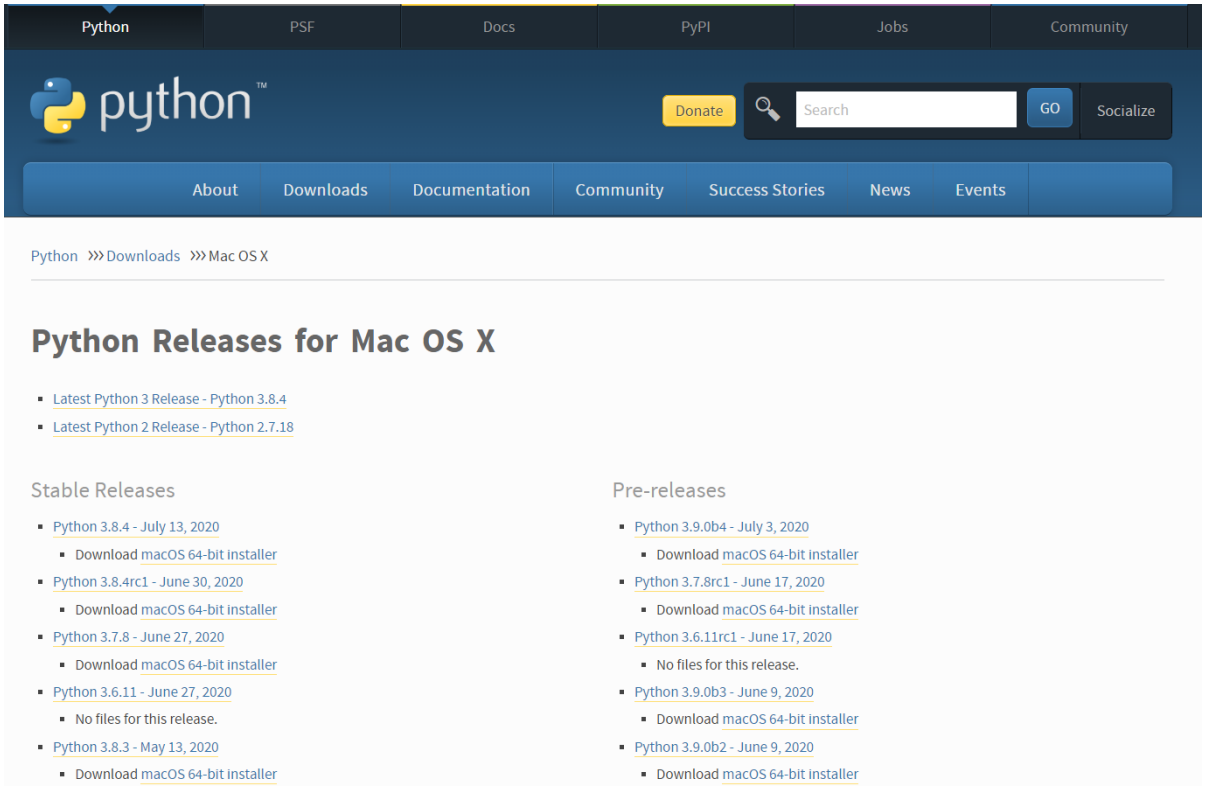

Ilustración 31 - Python para Mac

Una vez descargado el archivo, aparecerá un setup para su instalación, aquí se podrá elegir la ruta de instalación, aunque se recomienda usar la de por defecto en Install Now tal como se ve en la Ilustración 32 - Instalación Python paso 1. Una vez se haga click, el instalador instalará todas las librerías predeterminadas de Python y los paquetes necesarios. Es importante saber que no instalará las librerías del apartado Librerías de Python, estas librerías deben ser instaladas más tarde de manera manual desde PyCharm.

Ilustración 32 - Instalación Python paso 1

Una vez la instalación se ha completado exitosamente aparecerá la Ilustración 33 - Instalación Python paso 2. Si ha sucedido un error, intente repetir los pasos anteriores, y si el error persiste, póngase en contacto con los desarrolladores de Python mediante la web en la que están disponibles las descargas.

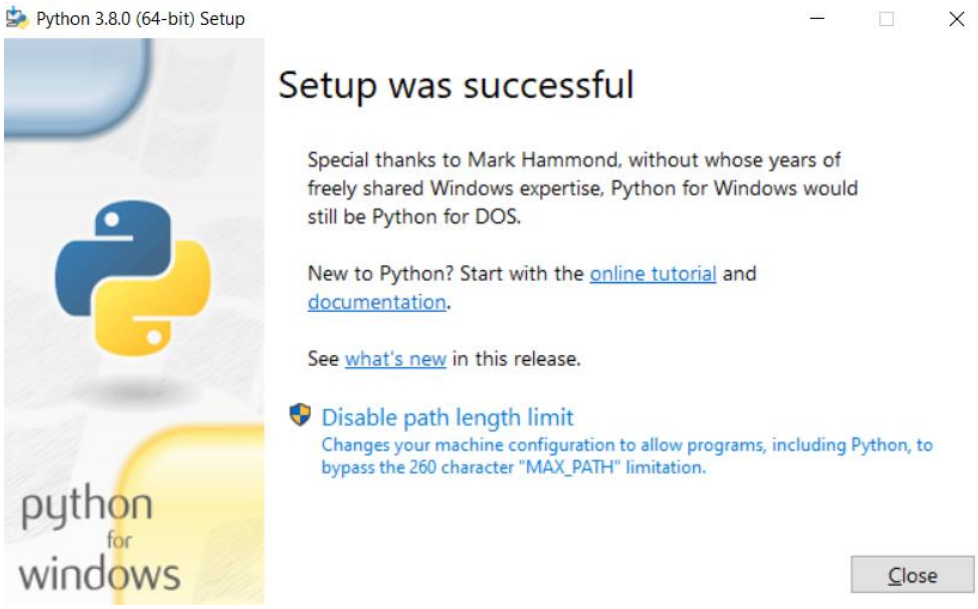

Ilustración 33 - Instalación Python paso 2

PYCHARM COMMUNITY EDITION 2019.3.2

Para la programación del bot, se ha utilizado la versión PyCharm Community Edition 2019.3.2. Sin embargo, se recomienda utilizar la última disponible, ya que esto no debería dar ningún tipo de error con las librerías. Para proceder a su descarga se debe acceder a la web de JetBrains (https://www.jetbrains.com/es-es/pycharm/) y al apartado de PyCharm. Una vez allí, veremos una interfaz como la de la Ilustración 34 - Web PyCharm, para proceder a su descarga únicamente hemos de hacer click en el botón: Descargar.

Ilustración 34 - Web PyCharm

Una vez hacemos click en descargar, se abrirá una nueva ventana como la de la Ilustración 35 - Descargar PyCharm, allí tendremos que elegir el sistema operativo que se desee utilizar, y descargar la versión Community, que es la gratuita. Tras hacer click en el botón descargar, comenzará la descarga del archivo de instalación.

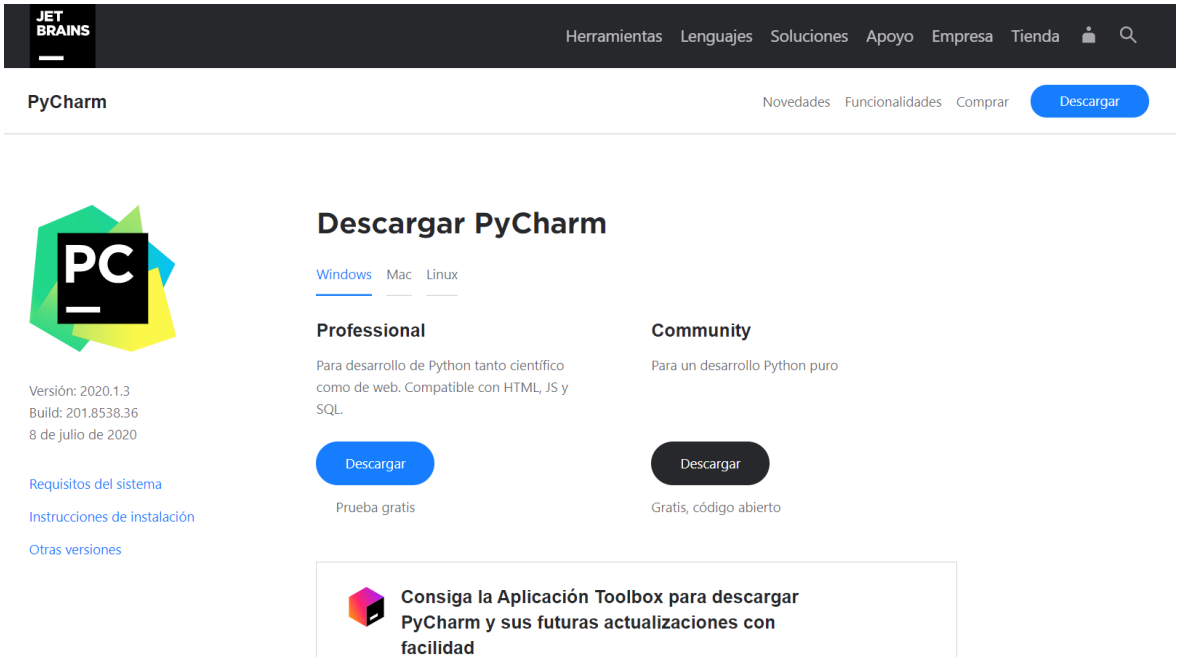

Ilustración 35 - Descargar PyCharm

Una vez haya finalizado la descarga, abrimos el archivo y aparecerá la imagen de la Ilustración 36 - Instalación PyCharm paso 1, hacemos click en Next >. Tras eso, aparecerá una imagen como de la Ilustración 37 - Instalación PyCharm paso 2, aquí se puede elegir la ruta de instalación que se desee. Sin embargo, lo más recomendable es no cambiarla y dejar la predeterminada para evitar posibles errores. En la siguiente ventana que se nos abrirá, veremos una imagen como la de la Ilustración 38 - Instalación PyCharm paso 3, aquí podemos crear un acceso directo en el escritorio dejando seleccionado el cuadrado de Create Desktop Shortcut, pero no es necesario. Simplemente hacemos click en Next > una vez más, y nos abrirá una nueva ventana como la de la Ilustración 39 - Instalación PyCharm paso 4, aquí debemos hacer click en Install, y comenzará la instalación del programa. Finalmente, si la instalación termina sin problemas, nos aparecerá una imagen como la de Ilustración 40 - Instalación PyCharm paso 5. Hacemos click en el botón Finish, y ya estaría instalado el programa.

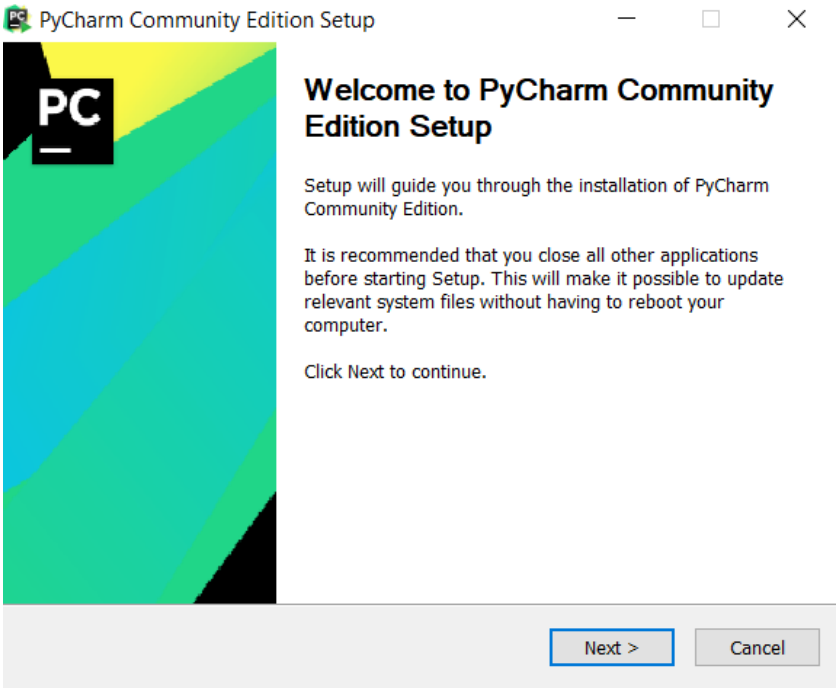

Ilustración 36 - Instalación PyCharm paso 1

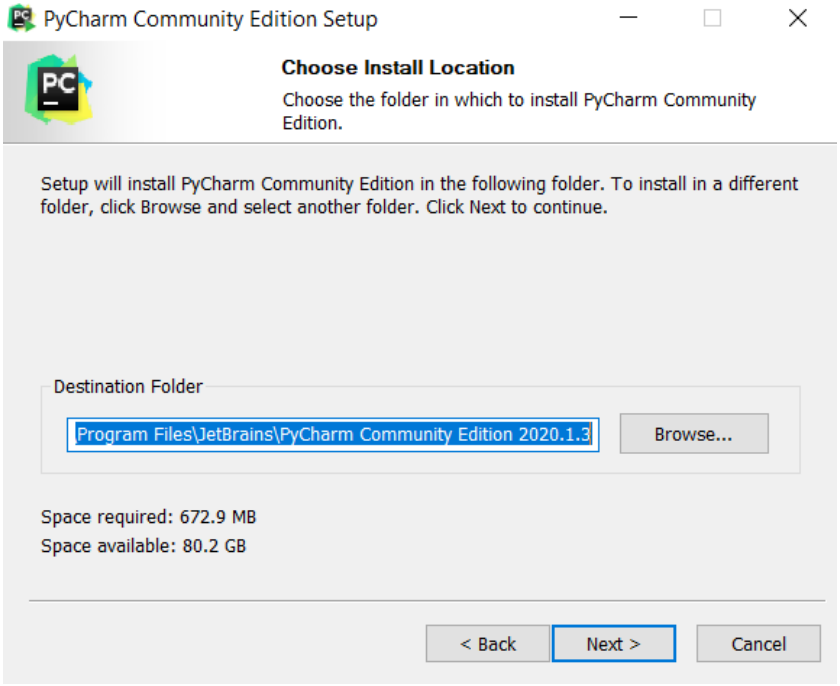

Ilustración 37 - Instalación PyCharm paso 2

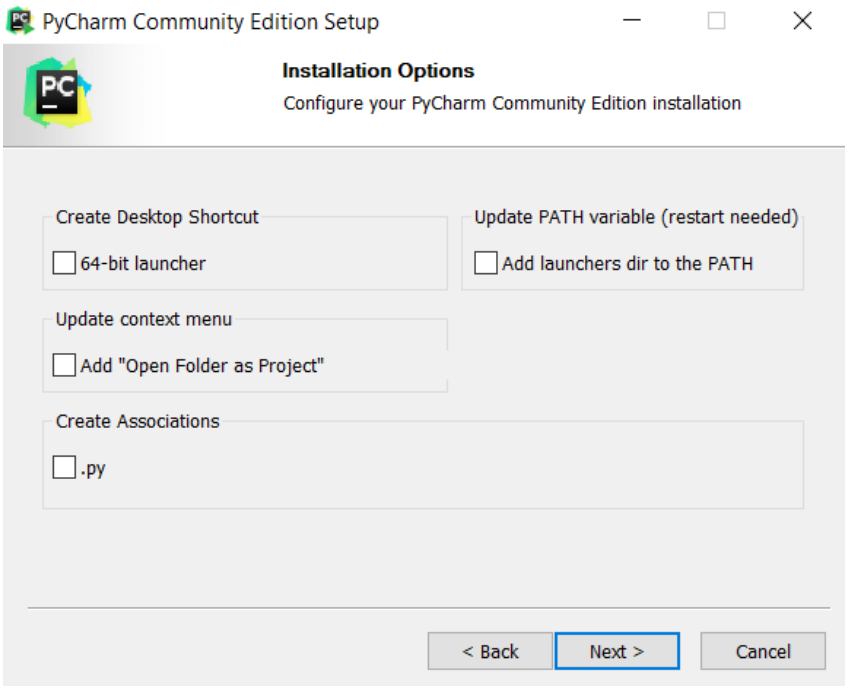

Ilustración 38 - Instalación PyCharm paso 3

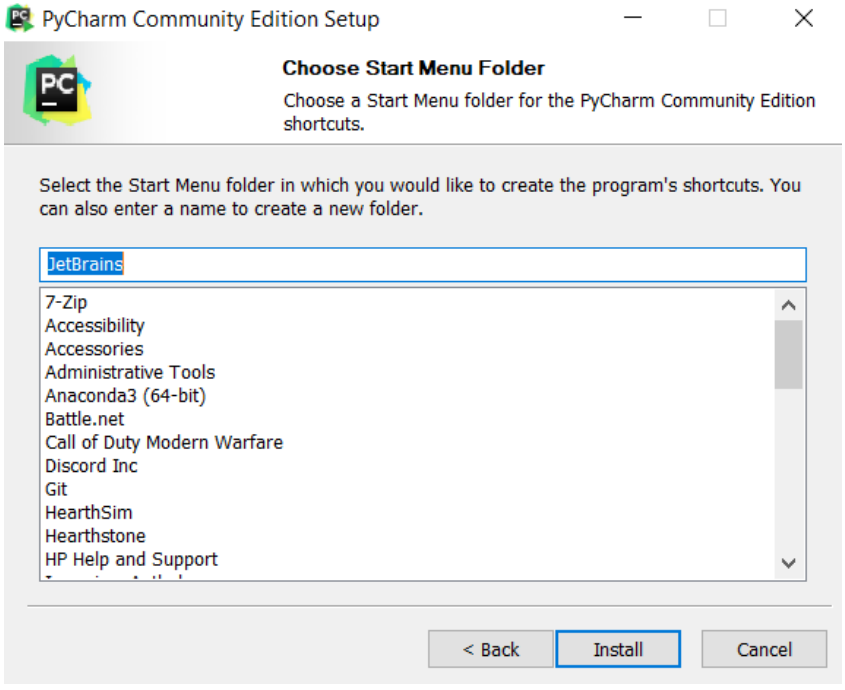

Ilustración 39 - Instalación PyCharm paso 4

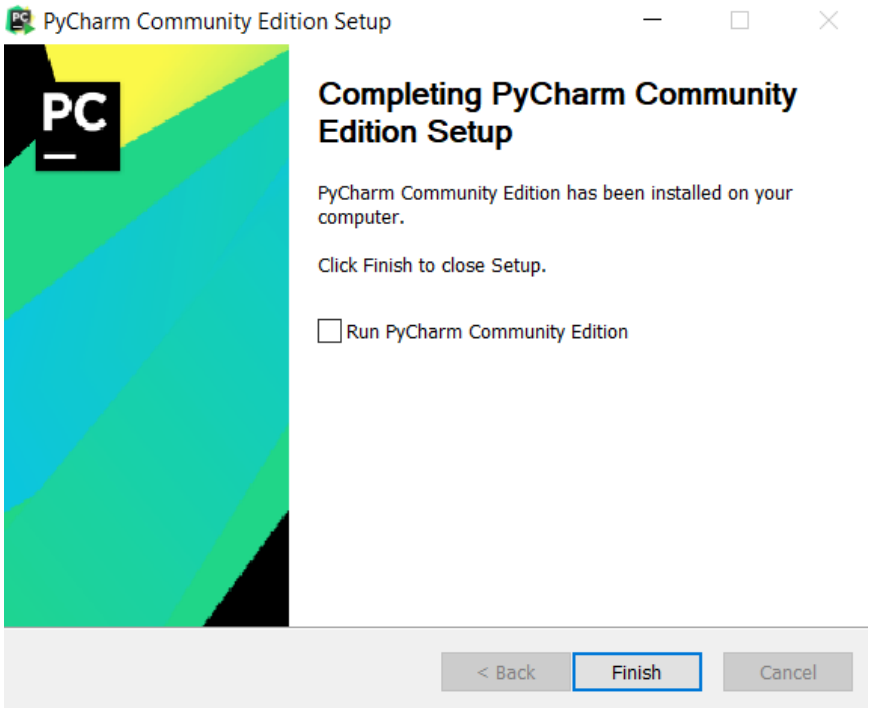

Ilustración 40 - Instalación PyCharm paso 5

MONGODB COMPASS COMMUNITY

Para el almacenamiento de datos, se ha utilizado el programa MongoDBCompassCommunity. Para proceder a su descarga se debe acceder a la web de MongoDB (https://www.mongodb.com/try/download) y al apartado de descargas. Una vez allí, veremos una interfaz como la de la Ilustración 41 - Web MongoDB, aquí hay tres diferentes tipos de descarga según como vayamos a usar la base de datos. Como usaremos la base de datos en el mismo ordenador, hacemos click en On-Premises y seleccionamos MongoDB Community Server. Veremos una imagen como la de la Ilustración 42 - MongoDB Community Server, allí seleccionamos la última versión disponible, el sistema operativo que vayamos a usar el formato msi.

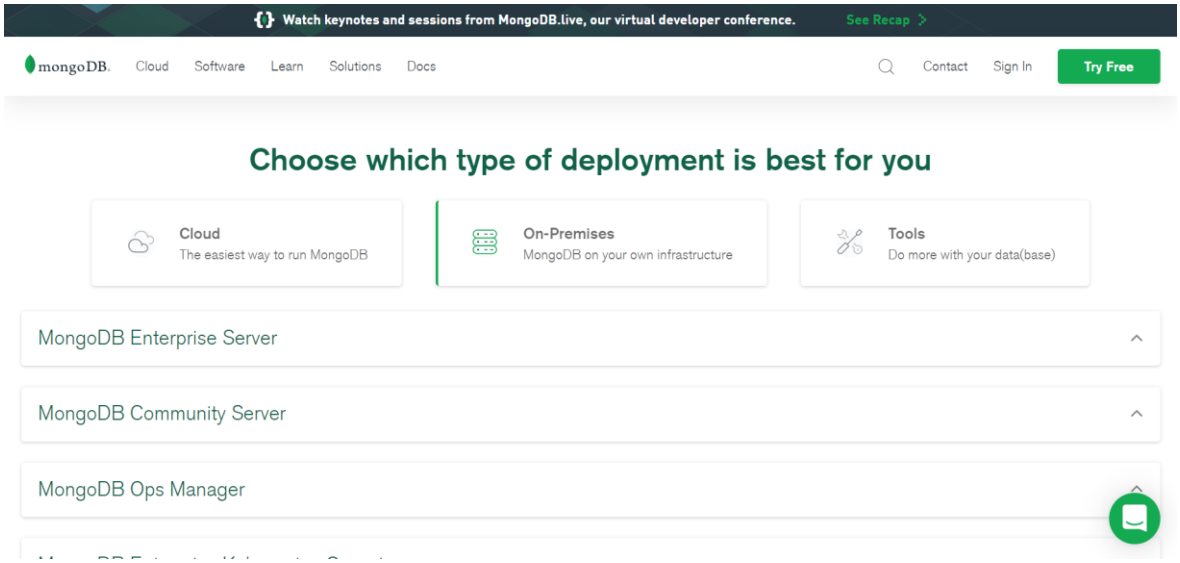

Ilustración 41 - Web MongoDB

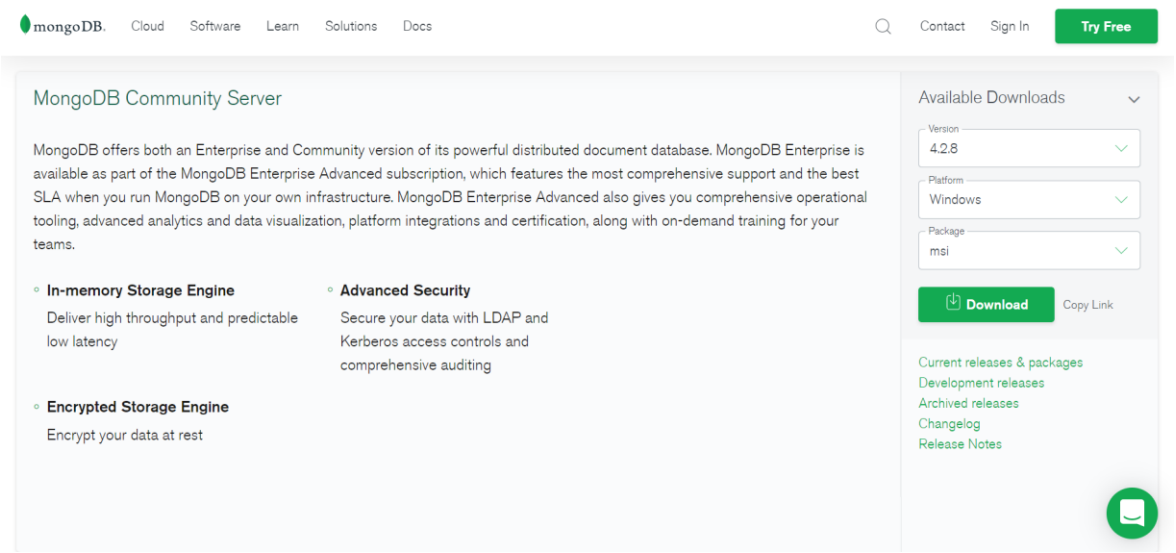

Ilustración 42 - MongoDB Community Server

Una vez descargado el archivo, procedemos a su instalación. Lo primero que aparecerá sera la imagen de laIlustración 43 - Instalación MongoDB paso 1, hacemos click en Next. Tras eso, aparecerá una imagen como de laIlustración 44 - Instalación MongoDB paso 2, tras leer los términos de la licencia si estamos de acuerdo le damos a Next (hemos de darle para proceder con la instalación). A continuación, elegimos una instalación completa en la Ilustración 45 - Instalación MongoDB paso 3, haciendo click en Complete y Next. Luego, en el service configuration (Ilustración 46 - Instalación MongoDB paso 4), se deja todo en predeterminado para evitar posibles errores y se hace click en Next. Nos saldrá una nueva ventana (Ilustración 47 - Instalación MongoDB paso 5) y volvemos a hacer click en Next.

Finalmente, en la última ventana nos aparecerá una imagen como la de la Ilustración 48 - Instalación MongoDB paso 6, en la cual únicamente hemos de hacer click en el botón Install y comenzará la instalación. Si todo ha salido correctamente aparecerá una nueva ventana una vez finalizada la instalación como la de la Ilustración 49 - Instalación MongoDB paso 7. Damos a Finish, y ya estaría instalado el programa.

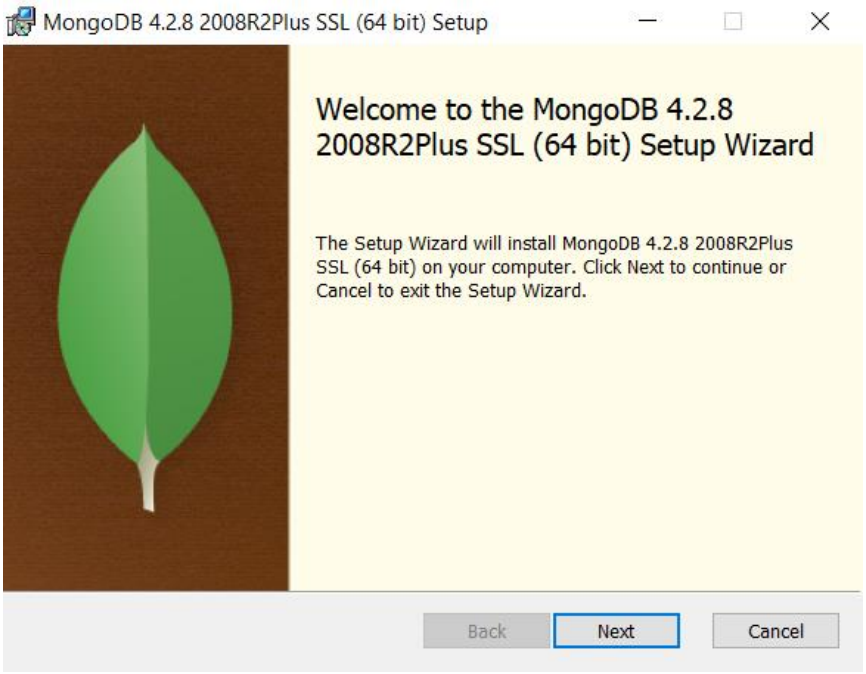

Ilustración 43 - Instalación MongoDB paso 1

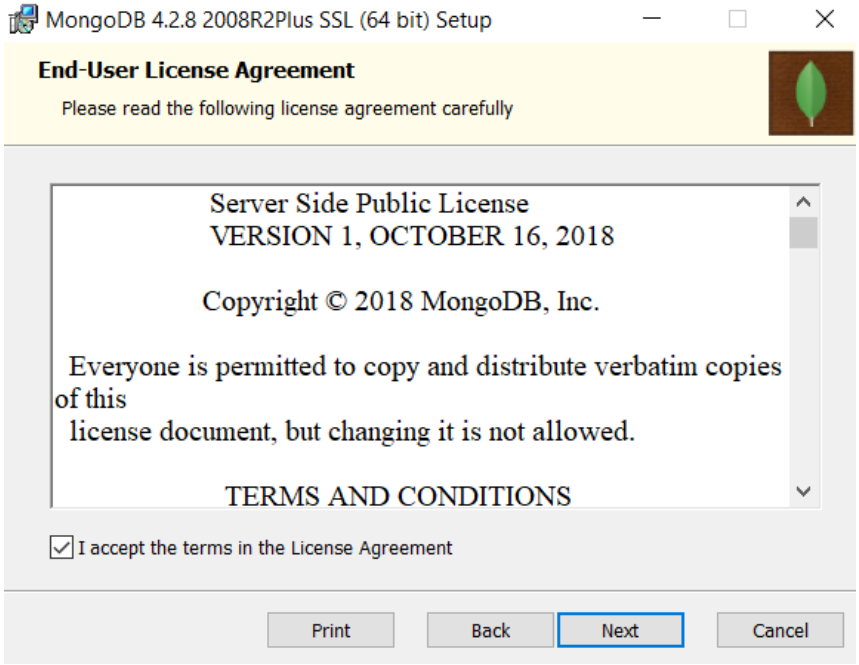

Ilustración 44 - Instalación MongoDB paso 2

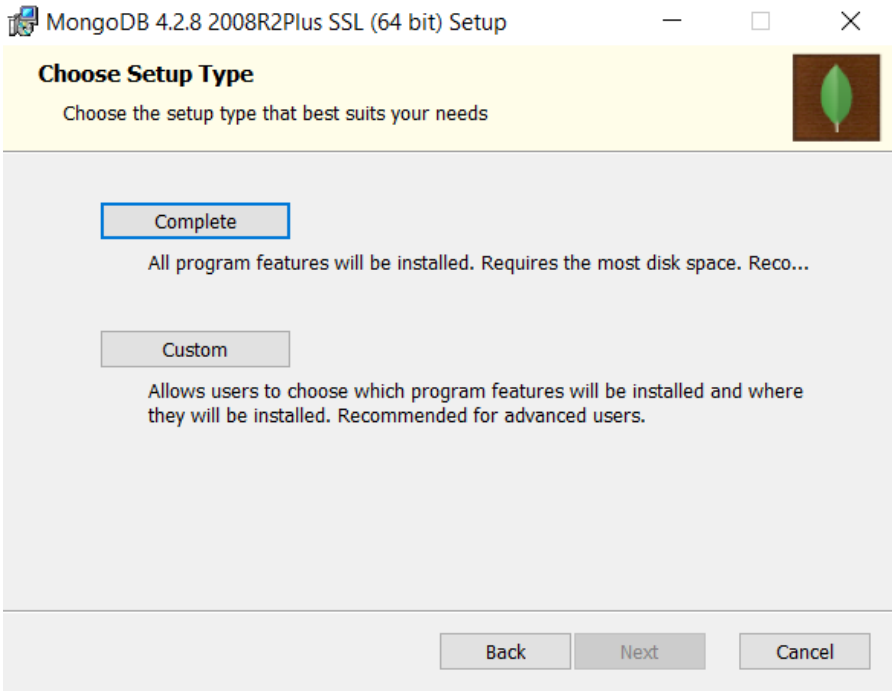

Ilustración 45 - Instalación MongoDB paso 3

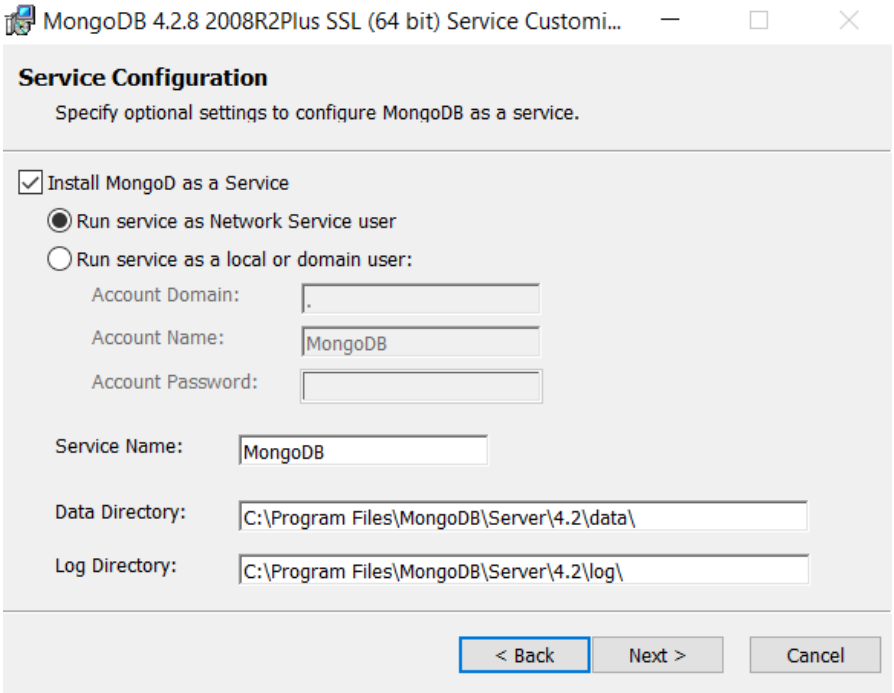

Ilustración 46 - Instalación MongoDB paso 4

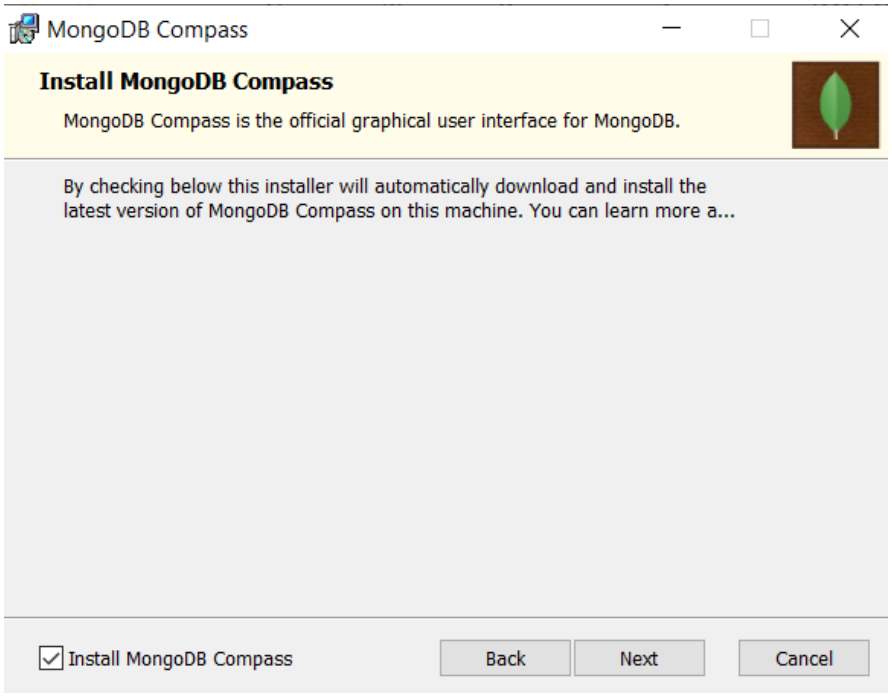

Ilustración 47 - Instalación MongoDB paso 5

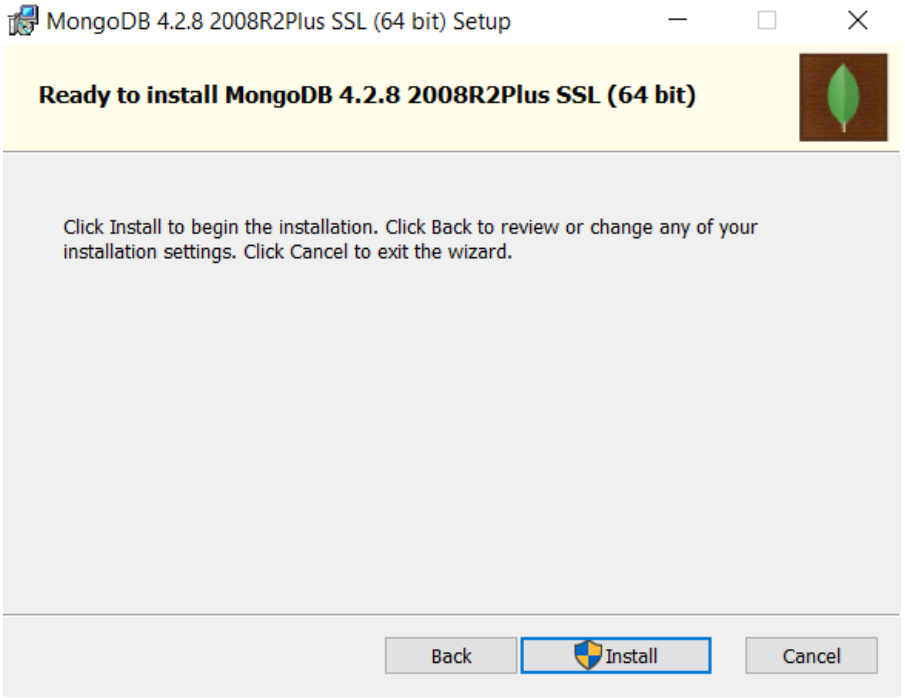

Ilustración 48 - Instalación MongoDB paso 6

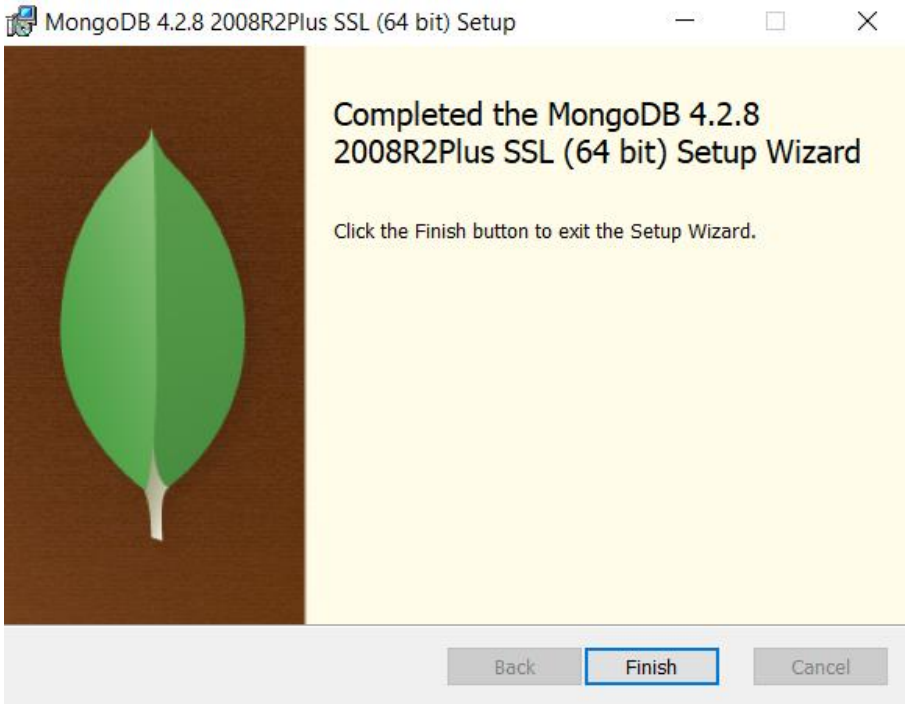

Ilustración 49 - Instalación MongoDB paso 7

ANEXO II: CÓDIGO

Todo el código se adjuntará aquí, aunque para un acceso más fácil, claro y limpio se puede acceder a través de la web de Github al proyecto (https://github.com/Diana-bot-TFG/Diana).

DICTIONARIES

DICTIONARIES

```
# Diccionarios
paises = {"Spain": "es"}
languages = {"Español": "es", "English": "en"}
divisas = {"EUR": "EUR", "USD": "USD", "GBP": "GBP", "JPY": "JPY", "CHF": "CHF",
"AUD": "AUD"}
divisas_simbolo = {"EUR": "€", "USD": "$", "GBP": "£", "JPY": "¥", "CHF": "Fr.",
"AUD": "A$"}
```

MENUS

```
# Menus
menu_price_range = {"Price Min": "Price Min", "Price Max": "Price Max", "Finish":
"Finish"}
menu_shops = {'Ebay': 'Ebay', 'Gearbest': 'Gearbest', 'Banggood': 'Banggood',
"Links": "Links", "↩ Finish": "Finish"}
menu_tracking = {"My Items": "My Items", "Add Item": "Add Item", "Delete Item":
"Delete Item",
                 "Check Items": "Check Items", "↩ Finish": "Finish"}
```

FUNCTIONS

CHECK

```
# Módulos
from Scraping.product_ebay import Ebay_Product
from Scraping.product_banggood import Banggood_Product
from Scraping.product_gearbest import Gearbest_Product
from Python_server.initialization import bot
from Dictionaries.dictionaries import divisas_simbolo


# Función
def check(id, Product, Country, Currency, Min, Max, Select_Ebay, Select_Gearbest,
Select_Banggood, usuario):
    # Únicamente se comprueban las webs seleccionadas por el usuario
    if Select_Ebay:
        ebay = Ebay_Product(Product, Min, Max, Currency, Country)
    if Select_Gearbest:
        gearbest = Gearbest_Product(Product, Min, Max, Currency, Country)
    if Select_Banggood:
        banggood = Banggood_Product(Product, Min, Max, Currency, Country)
    # Se comprueba si el producto de menor precio encontrado, cumple los márgenes
de precio establecido y se envía
    if Select_Ebay:
        try:
            min_ebay = ebay.Precio_producto.index(min(ebay.Precio_producto))
            if usuario['language'] == 'es':
                bot.send_message(
                    "[{}]\n\nHe encontrado su producto *{}* en
*Ebay*:\n\n{}\n\n*Precio:*\n{} {}\n\n*Link:*\n[{}]".format(
                        ebay.Imagen_producto[min_ebay], Product,
ebay.Nombre_producto[min_ebay],
                        ebay.Precio_producto[min_ebay],
divisas_simbolo[Currency],
                        ebay.Link_producto[min_ebay]), id)
            else:
                bot.send_message(
                    "[{}]\n\nI have found your product *{}* on
*Ebay*:\n\n{}\n\n*Precio:*\n{} {}\n\n*Link:*\n[{}]".format(
                        ebay.Imagen_producto[min_ebay], Product,
ebay.Nombre_producto[min_ebay],
                        ebay.Precio_producto[min_ebay],
divisas_simbolo[Currency],
                        ebay.Link_producto[min_ebay]), id)
        except:  # En caso de no haber un producto en el rango deseado, no se
envía nada
            pass
    if Select_Gearbest:
        try:
            min_gearbest =
```

```
gearbest.Precio_producto.index(min(gearbest.Precio_producto))
            if usuario['language'] == 'es':
                bot.send_message(
                    "[{}]\n\nHe encontrado su producto *{}* en
*Gearbest*:\n\n{}\n\n*Precio:*\n{} {}\n\n*Link:*\n[{}]".format(
                        gearbest.Imagen_producto[min_gearbest], Product,
gearbest.Nombre_producto[min_gearbest],
                        gearbest.Precio_producto[min_gearbest],
divisas_simbolo[usuario['currency']],
                        gearbest.Link_producto[min_gearbest]), id)
            else:
                bot.send_message(
                    "[{}]\n\nI have found your product *{}* on
*Gearbest*:\n\n{}\n\n*Precio:*\n{} {}\n\n*Link:*\n[{}]".format(
                        gearbest.Imagen_producto[min_gearbest], Product,
gearbest.Nombre_producto[min_gearbest],
                        gearbest.Precio_producto[min_gearbest],
divisas_simbolo[usuario['currency']],
                        gearbest.Link_producto[min_gearbest]), id)
        except:
            pass
    if Select_Banggood:
        try:
            min_banggood =
banggood.Precio_producto.index(min(banggood.Precio_producto))
            if usuario['language'] == 'es':
                bot.send_message(
                    "[{}]\n\nHe encontrado su producto *{}* en *Banggood* on
*Banggood*:\n\n{}\n\n*Precio:*\n{} {}\n\n*Link:*\n[{}]".format(
                        banggood.Imagen_producto[min_banggood], Product,
banggood.Nombre_producto[min_banggood],
                        banggood.Precio_producto[min_banggood],
divisas_simbolo[usuario['currency']],
                        banggood.Link_producto[min_banggood]), id)
            else:
                bot.send_message(
                    "[{}]\n\nI have found your product *{}* on
*Banggood*:\n\n{}\n\n*Precio:*\n{} {}\n\n*Link:*\n[{}]".format(
                        banggood.Imagen_producto[min_banggood], Product,
banggood.Nombre_producto[min_banggood],
                        banggood.Precio_producto[min_banggood],
divisas_simbolo[usuario['currency']],
                        banggood.Link_producto[min_banggood]), id)
        except:
            pass
```

DEFAULT_REPLY

```
# Módulos
from Python_server.initialization import bot
```

```
# Función
def default_reply(usuario):  # Respuesta por defecto
    if usuario['language'] == "es":
        bot.send_message(f"¿{usuario['name']}, qué quiere que haga ahora?
Recuerde usar los comandos, si tiene alguna "
                         f"duda pulse /help", usuario['id'])

    else:
        bot.send_message(f"{usuario['name']}, what do you want me to do now?
Remember to use the commands, if you "
                         f"have any doubt press /help", usuario['id'])
```

EMAIL

```
# Librerías
import smtplib
import configparser as cfg
from email.mime.text import MIMEText


# Función
def send_email(msg):
    # Creamos un servidor
    server = smtplib.SMTP('smtp.gmail.com: 587')
    server.starttls()

    # Sacamos la contraseña de un archivo config.cfg
    parser = cfg.ConfigParser()
    parser.read('config.cfg')
    password = parser.get('email', 'email_diana')

    # Iniciamos sesión
    server.login('dianabot2020@gmail.com', password)

    # Hacemos que se puedan enviar caracteres no-ASCII
    msg_tosend = MIMEText(msg, 'plain', 'latin-1').as_string()

    # Enviamos el mensaje
    server.sendmail('dianabot2020@gmail.com', 'dianabot2020@gmail.com',
msg_tosend)

    # Cerramos el servidor
    server.quit()
```

INLINE_KEYBOARD

```
# Función
def inline_keyboard(dictionary):  # Crea un inline_keyboard a partir de un
diccionario
    lista = []
    keys = list(dictionary.keys())
    # Creación del vector lista a partir del diccionario
    for i in range(len(dictionary)):
        lista.append([{"text": keys[i], "callback_data": keys[i]}])
    # Creación del inline_keyboard a partir de la lista
    keyboard = {"inline_keyboard": lista}
    return keyboard
```

SEARCH

```
# Módulos
from Scraping.product_ebay import Ebay_Product
from Scraping.product_banggood import Banggood_Product
from Scraping.product_gearbest import Gearbest_Product
from Python_server.initialization import bot
from Dictionaries.dictionaries import divisas_simbolo


# Función
def search(item, Select_Ebay, Select_Gearbest, Select_Banggood, Link_Activos,
usuario):
    # Fase de búsqueda en las webs seleccionadas por el usuario
    if Select_Ebay:
        ebay = Ebay_Product(
            item, usuario['range']['min'], usuario['range']['max'],
usuario['currency'], usuario['country'])

    if Select_Gearbest:
        gearbest = Gearbest_Product(
            item, usuario['range']['min'], usuario['range']['max'],
usuario['currency'], usuario['country'])

    if Select_Banggood:
        banggood = Banggood_Product(
            item, usuario['range']['min'], usuario['range']['max'],
usuario['currency'], usuario['country'])

    # Fase de envío del link si así lo pide el usuario
    if Link_Activos:
        if Select_Ebay:
            bot.send_message("*Ebay*\n[{}]".format(ebay.link), usuario['id'])

        if Select_Gearbest:
```

```
            bot.send_message("*Gearbest*\n[{}]".format(gearbest.link),
usuario['id'])

        if Select_Banggood:
            bot.send_message("*Banggood*\n[{}]".format(banggood.link),
usuario['id'])

    # Fase de envío de la mejor opción, si no encuentra nada que cumpla los
límites, envía un mensaje predeterminado
    if Select_Ebay:
        try:
            min_ebay = ebay.Precio_producto.index(min(ebay.Precio_producto))
            bot.send_message("[{}]\n\n*Ebay:*\n{}\n\n*Precio:*\n{}
{}\n\n*Link:*\n[{}]".format(
                ebay.Imagen_producto[min_ebay], ebay.Nombre_producto[min_ebay],
                ebay.Precio_producto[min_ebay],
divisas_simbolo[usuario['currency']],
                ebay.Link_producto[min_ebay]), usuario['id'])

        except:
            if usuario['language'] == 'es':
                bot.send_message("*Ebay:*\nNo he encontrado nada en Ebay",
usuario['id'])
            else:
                bot.send_message("*Ebay:*\nI have not found anything on Ebay",
usuario['id'])

    if Select_Gearbest:
        try:
            min_gearbest =
gearbest.Precio_producto.index(min(gearbest.Precio_producto))
            bot.send_message("[{}]\n\n*Gearbest:*\n{}\n\n*Precio:*\n{}
{}\n\n*Link:*\n[{}]".format(
                gearbest.Imagen_producto[min_gearbest],
gearbest.Nombre_producto[min_gearbest],
                gearbest.Precio_producto[min_gearbest],
divisas_simbolo[usuario['currency']],
                gearbest.Link_producto[min_gearbest]), usuario['id'])

        except:
            if usuario['language'] == 'es':
                bot.send_message("*Gearbest:*\nNo he encontrado nada en
Gearbest", usuario['id'])
            else:
                bot.send_message("*Gearbest:*\nI have not found anything on
Gearbest", usuario['id'])

    if Select_Banggood:
        try:
            min_banggood =
banggood.Precio_producto.index(min(banggood.Precio_producto))
            bot.send_message("[{}]\n\n*Banggood:*\n{}\n\n*Precio:*\n{}
{}\n\n*Link:*\n[{}]".format(
```

```
                banggood.Imagen_producto[min_banggood],
banggood.Nombre_producto[min_banggood],
                banggood.Precio_producto[min_banggood],
divisas_simbolo[usuario['currency']],
                banggood.Link_producto[min_banggood]), usuario['id'])

        except:
            if usuario['language'] == 'es':
                bot.send_message("*Banggood:*\nNo he encontrado nada en
Banggood", usuario['id'])
            else:
                bot.send_message("*Banggood:*\nI have not found anything on
Banggood", usuario['id'])
```

STATE

```
# Módulos
from Python_server.initialization import cluster


# Función
def state(id, search, price_range, shops, tracking, country, language, currency,
contact):  # Modifica la state machine
    cluster.update_one(
        {'id': id},
        {'$set':
            {
                'state_machine.search': search,
                'state_machine.price_range': price_range,
                'state_machine.shops': shops,
                'state_machine.tracking': tracking,
                'state_machine.country': country,
                'state_machine.language': language,
                'state_machine.currency': currency,
                'state_machine.contact': contact
            }
        }
    )
```

TRACKING

```
# Módulos
from Python_server.initialization import cluster
from Functions.check import check


# Función
def tracking():
    # Extraemos todos los usuarios que tengan productos
```

```
    usuarios = cluster.find({"products.name_product": {"$ne": None}})

    # Vamos usuario a usuario y enviamos un mensaje si el producto deseado está
al precio establecido
    for usuario in usuarios:
        for i in range(len(usuario['products'])):
            check(usuario['id'], usuario['products'][i]['name_product'],
usuario['country'], usuario['currency'],
                  usuario['products'][i]['price_product_min'],
usuario['products'][i]['price_product_max'],
                  usuario['shops']['Ebay'], usuario['shops']['Gearbest'],
usuario['shops']['Banggood'], usuario)
```

PYTHON_SERVER

INITIALIZATION

```
# Librerías
from currency_converter import CurrencyConverter
from pymongo import MongoClient

# Módulos
from Telegram_server.bot import telegram_bot

# Telegram
bot = telegram_bot('config.cfg')

# Divisas
c = CurrencyConverter()

# Base de datos
client = MongoClient('localhost', 27017)
db = client.diana
cluster = db.diana_usuarios
```

SERVER

```
# Librerías
import json
import time

# Archivos
from Telegram_server.user import User
from Python_server.initialization import bot, c, cluster

# Diccionarios
from Dictionaries.dictionaries import paises, languages, divisas, divisas_simbolo

# Menús
from Dictionaries.menus import menu_price_range, menu_shops, menu_tracking

# Funciones
from Functions.check import check
from Functions.default_reply import default_reply
from Functions.inline_keyboard import inline_keyboard
from Functions.search import search
from Functions.state import state
from Functions.tracking import tracking
from Functions.email import send_email

# ------------------------------------------------------------------------------
-------------------

# Main
update_id = None

while True:
    # Mira a ver si hay un mensaje nuevo
    updates = bot.get_updates(offset=update_id)
    updates = updates["result"]

    # Actualiza los tracking de todos los usuarios a las 2:30am (para hacerlo a
una hora con poca carga en el bot)
    if time.strftime("%X") == "02:30:00":
        tracking()

    # Se entra en el if si se recibe un mensaje
    if updates:
        print(json.dumps(updates, indent=2, sort_keys=True))
        user = User(updates)
        update_id = user.update_id

        # Base de datos
        if user.from_ != "":  # Si no hay id del usuario es porque es un mensaje
no analizable (fotos, vídeos, etc)
            if cluster.find_one({'id': user.from_}):  # Comprobación de si el
usuario está en la base de datos
```

```
                usuario = cluster.find_one({'id': user.from_})  # Si está en la
base de datos se carga su información
            else:
                # Comprobación de si se tiene el idioma del usuario, si no se le
aplica el inglés
                if user.language in languages.values():
                    pass
                else:
                    user.language = "en"
                # Definición del usuario, en caso de no estar previamente en la
base de datos
                usuario = {
                    'id': user.from_,
                    'name': user.chat_name,
                    'username': user.chat_username,
                    'language': user.language,
                    'country': paises["Spain"],
                    'currency': divisas["EUR"],
                    'range':
                        {
                            'min': 0,
                            'max': 1000
                        },
                    'shops':
                        {
                            'Ebay': True,
                            'Gearbest': True,
                            'Banggood': True,
                            'Links': True
                        },
                    'state_machine':
                        {
                            'search': 0,
                            'price_range': 0,
                            'shops': 0,
                            'tracking': 0,
                            'country': 0,
                            'language': 0,
                            'currency': 0
                        }
                }
                # Inserción del usuario en la base de datos
                cluster.insert_one(usuario)

        # Se entra al if si es un mensaje escrito o de menú y viene de un chat
privado
        if user.type_chat == "private" and user.type_message == "message" or
user.type_message == "callback_query":
            try:
                # Primera máquina de estados
                if user.message_text == "/start" or user.message_text == "/help":
                    # Se distingue si el idioma del usuario es español o inglés,
esto se hará continuamente durante el código
```

```
                    if usuario['language'] == "es":
                        bot.send_message(
                            f"Hola {usuario['name']}, me llamo Diana, soy un Bot
orientado a las compras en Internet, "
                            "puedes usarme para buscar algo que necesites"
                            "\n\n*Comandos*" +
                            "\n/search - busca un producto por su nombre" +
                            "\n/price\_range - elige el mínimo y máximo precio
(inicialmente 0-1000€)" +
                            "\n/shops - selecciona las tiendas que quieres usar"
+
                            "\n/tracking - muestra los productos en lista de
seguimiento" +
                            "\n/country - cambia el país de búsqueda" +
                            "\n/language - cambia el idioma" +
                            "\n/currency - cambia la divisa" +
                            "\n/help - muestra todos los comandos" +
                            "\n/contact - envía un mensaje al administrador" +
                            "\n/cancel - cancela la última orden", usuario['id'])
                    else:
                        bot.send_message(
                            f"Hello {usuario['name']}, my name is Diana, I am a
Bot oriented to Internet purchases, "
                            "you can use me to find something you need"
                            "\n\n*Commands*" +
                            "\n/search - search for a product by name" +
                            "\n/price\_range - select the minimum and maximum
price (inicially 0-1000€)" +
                            "\n/shops - select which shops you want to use" +
                            "\n/tracking - show the tracking list" +
                            "\n/country - change country" +
                            "\n/language - change language" +
                            "\n/currency - change currency" +
                            "\n/help - show all commands" +
                            "\n/contact - send a message to the administrator" +
                            "\n/cancel - cancel the last order", usuario['id'])

                elif user.message_text == "/search":

                    if usuario['language'] == "es":
                        bot.send_message("Introduce el nombre del producto que
quieras", usuario['id'])
                    else:
                        bot.send_message("Send the name of the product you want",
usuario['id'])

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 1, 0, 0, 0, 0, 0, 0)

                elif user.message_text == "/price_range":

                    if usuario['language'] == "es":
                        bot.send_message("*¿Qué desea cambiar?*", usuario['id'],
```

```
inline_keyboard(menu_price_range))
                    else:
                        bot.send_message("*What do you want to change?*",
usuario['id'],
                                         inline_keyboard(menu_price_range))

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 1, 0, 0, 0, 0, 0, 0)

                elif user.message_text == "/shops":

                    if usuario['language'] == "es":
                        bot.send_message("*Seleccione las tiendas que quiera usar
y pulse Finish*", usuario['id'],
                                         inline_keyboard(menu_shops))
                    else:
                        bot.send_message("*Select the shops you want to use and
press Finish*", usuario['id'],
                                         inline_keyboard(menu_shops))

                    # Se ponen todas las tiendas en OFF
                    cluster.update_one(
                        {
                            'id': usuario['id']
                        },
                        {
                            '$set':
                                {
                                    'shops.Ebay': False,
                                    'shops.Gearbest': False,
                                    'shops.Banggood': False,
                                    'shops.Links': False
                                }
                        }
                    )

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 1, 0, 0, 0, 0, 0)

                elif user.message_text == "/tracking":

                    if usuario['language'] == "es":
                        bot.send_message("*Elija una opcion*:", usuario['id'],
inline_keyboard(menu_tracking))
                    else:
                        bot.send_message("*Select an option*:", usuario['id'],
inline_keyboard(menu_tracking))

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 0, 1, 0, 0, 0, 0)

                elif user.message_text == "/country":
```

```
                    if usuario['language'] == "es":
                        bot.send_message("*Elija su pais*", usuario['id'],
inline_keyboard(paises))
                    else:
                        bot.send_message("*Select your country*", usuario['id'],
inline_keyboard(paises))

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 0, 0, 1, 0, 0, 0)

                elif user.message_text == "/language":

                    if usuario['language'] == "es":
                        bot.send_message("*Elija su idioma*", usuario['id'],
inline_keyboard(languages))
                    else:
                        bot.send_message("*Select your language*", usuario['id'],
inline_keyboard(languages))

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 0, 0, 0, 1, 0, 0)

                elif user.message_text == "/currency":

                    if usuario['language'] == "es":
                        bot.send_message("*Elija su moneda*", usuario['id'],
inline_keyboard(divisas))
                    else:
                        bot.send_message("*Select your currency*", usuario['id'],
inline_keyboard(divisas))

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 0, 0, 0, 0, 1, 0)

                elif user.message_text == "/contact":
                    if usuario['language'] == "es":
                        bot.send_message("Escriba el mensaje a enviar",
usuario['id'])
                    else:
                        bot.send_message("Write the message to send",
usuario['id'])

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 1)

                elif user.message_text == "/cancel":

                    if usuario['language'] == "es":
                        bot.send_message("Operacion cancelada", usuario['id'])
                    else:
                        bot.send_message("Operation cancelled", usuario['id'])

                    # Actualización del estado de la máquina de estados
```

```
                    state(usuario['id'], 0, 0, 0, 0, 0, 0, 0)

                # Segunda máquina de estados
                elif usuario['state_machine']['search'] == 1:

                    if usuario['language'] == "es":
                        bot.send_message("Buscando...", usuario['id'])
                    else:
                        bot.send_message("Searching...", usuario['id'])
                    # Búsqueda
                    search(user.message_text, usuario['shops']['Ebay'],
usuario['shops']['Gearbest'],
                           usuario['shops']['Banggood'],
usuario['shops']['Links'], usuario)
                    # No se sale de este estado, para poder buscar varias cosas
seguidas

                elif usuario['state_machine']['price_range'] == 1:

                    if user.message_text == "Price Min":

                        if usuario['language'] == "es":
                            bot.send_message("Introduzca un precio mínimo",
usuario['id'])
                        else:
                            bot.send_message("Enter the minimum price",
usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 2, 0, 0, 0, 0, 0)

                    elif user.message_text == "Price Max":

                        if usuario['language'] == "es":
                            bot.send_message("Introduzca un precio máximo",
usuario['id'])
                        else:
                            bot.send_message("Enter the maximum price",
usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 3, 0, 0, 0, 0, 0)

                    elif user.message_text == "Finish":

                        if usuario['language'] == "es":
                            bot.send_message(
                                f"*Precio Min:* {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}\n"
                                f"*Precio Max:* {usuario['range']['max']}
{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])
                        else:
```

```
                            bot.send_message(
                                f"*Precio Min:* {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}\n"
                                f"*Precio Max:* {usuario['range']['max']}
{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 0, 0, 0, 0)

                elif usuario['state_machine']['price_range'] == 2:

                    try:
                        float(user.message_text)
                        # Insertamos el nuevo precio mínimo
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'range.min': user.message_text}})

                        # Descarga del nuevo usuario con el nuevo precio mínimo
                        usuario = cluster.find_one({'id': usuario['id']})

                        if usuario['language'] == "es":
                            bot.send_message(
                                f"*Precio mínimo:*  {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])
                        else:
                            bot.send_message(
                                f"*Minimum Price:*  {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 1, 0, 0, 0, 0, 0)

                        if usuario['language'] == "es":
                            bot.send_message("*¿Qué desea cambiar?*",
usuario['id'], inline_keyboard(menu_price_range))
                        else:
                            bot.send_message(
                                "*What do you want to change?*", usuario['id'],
                                inline_keyboard(menu_price_range))

                    except:
                        if user.message_text == "Finish":
                            if usuario['language'] == "es":
                                bot.send_message(
                                    f"*Precio Min:* {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}"
                                    f"\n*Precio Max:* {usuario['range']['max']}
{divisas_simbolo[usuario['currency']]}",
                                    usuario['id'])
                            else:
```

```
                                bot.send_message(
                                    f"*Min Price:* {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}"
                                    f"\n*Max Price:* {usuario['range']['max']}
{divisas_simbolo[usuario['currency']]}",
                                    usuario['id'])

                            # Actualización del estado de la máquina de estados
                            state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)

                        else:
                            if usuario['language'] == "es":
                                bot.send_message(
                                    "Ha ocurrido un error, escriba nuevamente el
precio mínimo "
                                    "(use el punto para los decimales, no la
coma)", usuario['id'])
                            else:
                                bot.send_message(
                                    "An error has occurred, write the minimum
price again "
                                    "(use the decimal point for not comma)",
usuario['id'])

                elif usuario['state_machine']['price_range'] == 3:
                    try:
                        float(user.message_text)
                        # Insertamos el nuevo precio máximo
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'range.max': user.message_text}})

                        # Descarga del nuevo usuario con el nuevo precio máximo
                        usuario = cluster.find_one({'id': usuario['id']})

                        if usuario['language'] == "es":
                            bot.send_message(
                                f"*Precio máximo:*  {usuario['range']['max']} "
                                f"{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])
                        else:
                            bot.send_message(
                                f"*Maximum price:*  {usuario['range']['max']} "
                                f"{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 1, 0, 0, 0, 0, 0, 0)

                        if usuario['language'] == "es":
                            bot.send_message("*¿Qué desea cambiar?*",
usuario['id'], inline_keyboard(menu_price_range))
                        else:
                            bot.send_message(
```

```
                                "*What do you want to change?*", usuario['id'],
inline_keyboard(menu_price_range))

                    except:
                        if user.message_text == "Finish":
                            if usuario['language'] == "es":
                                bot.send_message(
                                    f"*Precio Min:* {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}"
                                    f"\n*Precio Max:* {usuario['range']['max']}
{divisas_simbolo[usuario['currency']]}",
                                    usuario['id'])
                            else:
                                bot.send_message(
                                    f"*Min Price:* {usuario['range']['min']}
{divisas_simbolo[usuario['currency']]}"
                                    f"\n*Max Price:* {usuario['range']['max']}
{divisas_simbolo[usuario['currency']]}",
                                    usuario['id'])

                            # Actualización del estado de la máquina de estados
                            state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)

                        else:
                            if usuario['language'] == "es":
                                bot.send_message(
                                    "Ha ocurrido un error, escriba nuevamente el
precio máximo "
                                    "(use el punto para los decimales, no la
coma)", usuario['id'])
                            else:
                                bot.send_message(
                                    "An error has occurred, write the maximum
price again "
                                    "(use the decimal point for not comma)",
usuario['id'])

                elif usuario['state_machine']['shops'] == 1:

                    if user.message_text == 'Ebay':

                        if usuario['language'] == "es":
                            bot.send_message("*Ebay activa*", usuario['id'])
                        else:
                            bot.send_message("*Ebay active*", usuario['id'])

                        # Activación de Ebay
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'shops.Ebay': True}})

                    elif user.message_text == 'Gearbest':

                        if usuario['language'] == "es":
```

```
                            bot.send_message("*Gearbest activa*", usuario['id'])
                        else:
                            bot.send_message("*Gearbest active*", usuario['id'])

                        # Activación de Gearbest
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'shops.Gearbest': True}})

                    elif user.message_text == 'Banggood':

                        if usuario['language'] == "es":
                            bot.send_message("*Banggood activa*", usuario['id'])
                        else:
                            bot.send_message("*Banggood active*", usuario['id'])

                        # Activación de Banggood
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'shops.Banggood': True}})

                    elif user.message_text == "Links":

                        if usuario['language'] == "es":
                            bot.send_message("*Links activos*", usuario['id'])
                        else:
                            bot.send_message("*Links active*", usuario['id'])

                        # Activación de Links
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'shops.Links': True}})

                    elif user.message_text == "↩ Finish":
                        # Descarga del nuevo usuario
                        usuario = cluster.find_one({'id': usuario['id']})
                        if usuario['shops']['Ebay'] == False and
usuario['shops']['Gearbest'] == False and \
                                usuario['shops']['Banggood'] == False:
                            if usuario['language'] == "es":
                                bot.send_message("*No ha seleccionado ninguna
tienda*", usuario['id'])
                            else:
                                bot.send_message("*You have not seleccted any
shop*", usuario['id'])
                        else:
                            default_reply(usuario)
                            # Actualización del estado de la máquina de estados
                            state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)

                elif usuario['state_machine']['tracking'] == 1:

                    if user.message_text == "My Items":
                        try:
                            if len(usuario['products']) == 0:
                                if usuario['language'] == "es":
```

```
                                    bot.send_message("No tiene productos en
seguimiento", usuario['id'])
                                else:
                                    bot.send_message("You have no products on
tracking", usuario['id'])
                            else:
                                if usuario['language'] == "es":
                                    text = ""
                                    i = 0
                                    for i in range(len(usuario['products'])):
                                        text = text + "\n\n📦*Producto*: {}
\n💰*Precio*: {} {} - {} {}".format(

usuario['products'][i]['name_product'],

usuario['products'][i]['price_product_min'],
                                                divisas_simbolo[usuario['currency']],

usuario['products'][i]['price_product_max'],
                                                divisas_simbolo[usuario['currency']])
                                    bot.send_message(text, usuario['id'])
                                else:
                                    text = ""
                                    i = 0
                                    for i in range(len(usuario['products'])):
                                        text = text + "\n\n📦*Item*: {}
\n💰*Price*: {} {} - {} {}".format(

usuario['products'][i]['name_product'],

usuario['products'][i]['price_product_min'],
                                                divisas_simbolo[usuario['currency']],

usuario['products'][i]['price_product_max'],
                                                divisas_simbolo[usuario['currency']])
                                    bot.send_message(text, usuario['id'])

                        except:
                            if usuario['language'] == "es":
                                bot.send_message("No tiene productos en
seguimiento", usuario['id'])
                            else:
                                bot.send_message("You have no products on
tracking", usuario['id'])

                    elif user.message_text == "Add Item":
                        # Borramos lo temporal
                        try:
                            cluster.update_one({'id': usuario['id']}, {"$unset":
{"products_temporal": ""}})
                        except:
                            pass
```

```
                        if usuario['language'] == "es":
                            bot.send_message("Introduzca el *nombre del
producto*", usuario['id'])
                        else:
                            bot.send_message("Enter the *name of the product*",
usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 2, 0, 0, 0, 0)

                    elif user.message_text == "Delete Item":
                        diccionario_productos = {}
                        try:
                            i = 0
                            for i in range(len(usuario['products'])):

diccionario_productos[usuario['products'][i]['name_product']] =
usuario['products'][i]
                                    ['name_product']
                            if usuario['language'] == "es":
                                bot.send_message("Seleccione para eliminar",
usuario['id'],

inline_keyboard(diccionario_productos))
                            else:
                                bot.send_message("Select to delete",
usuario['id'],

inline_keyboard(diccionario_productos))

                            # Actualización del estado de la máquina de estados
                            state(usuario['id'], 0, 0, 0, 5, 0, 0, 0, 0)

                        except:
                            if usuario['language'] == "es":
                                bot.send_message("No tiene productos en
seguimiento", usuario['id'])
                            else:
                                bot.send_message("You have no products on
tracking", usuario['id'])

                            # Actualización del estado de la máquina de estados
                            state(usuario['id'], 0, 0, 0, 1, 0, 0, 0, 0)

                    elif user.message_text == "Check Items":
                        try:
                            i = 0
                            for i in range(len(usuario['products'])):
                                check(
                                    usuario['id'],
usuario['products'][i]['name_product'], usuario['country'],
                                    usuario['currency'],
```

```
usuario['products'][i]['price_product_min'],
                                    usuario['products'][i]['price_product_max'],
usuario['shops']['Ebay'],
                                    usuario['shops']['Gearbest'],
usuario['shops']['Banggood'], usuario)

                                if usuario['language'] == "es":
                                    bot.send_message("*Elija una opcion*:",
usuario['id'], inline_keyboard(menu_tracking))
                                else:
                                    bot.send_message("*Select an option*:",
usuario['id'], inline_keyboard(menu_tracking))

                            except:
                                if usuario['language'] == "es":
                                    bot.send_message("No tiene productos en
seguimiento", usuario['id'])
                                else:
                                    bot.send_message("You have no products on
tracking", usuario['id'])

                                # Actualización del estado de la máquina de estados
                                state(usuario['id'], 0, 0, 0, 1, 0, 0, 0, 0)

                        elif user.message_text == "↩ Finish":
                            default_reply(usuario)

                            # Actualización del estado de la máquina de estados
                            state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)

                    elif usuario['state_machine']['tracking'] == 2:  # Add Item
                        # Insertamos el nombre del item
                        cluster.update_one(
                            {'id': usuario['id']}, {"$set":
{'products_temporal.name_product': user.message_text}})

                        if usuario['language'] == "es":
                            bot.send_message("Introduzca un *precio mínimo*",
usuario['id'])
                        else:
                            bot.send_message("Enter the *minimum price*",
usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 3, 0, 0, 0, 0)

                    elif usuario['state_machine']['tracking'] == 3:  # Add Item
                        try:
                            float(user.message_text)
                            # Insertamos el precio max del item
                            cluster.update_one(
                                {'id': usuario['id']}, {"$set":
{'products_temporal.price_product_min': user.message_text}})
```

```
                        if usuario['language'] == "es":
                            bot.send_message("Introduzca un *precio máximo*",
usuario['id'])
                        else:
                            bot.send_message("Enter the *maximun price*",
usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 4, 0, 0, 0, 0)

                    except:
                        if usuario['language'] == "es":
                            bot.send_message(
                                "Ha ocurrido un error, escriba nuevamente el
precio mínimo "
                                "(use el punto para los decimales, no la coma)",
usuario['id'])
                        else:
                            bot.send_message(
                                "An error has occurred, write the minimum price
again "
                                "(use the decimal point for not comma)",
usuario['id'])

                elif usuario['state_machine']['tracking'] == 4:  # Add Item
                    try:
                        float(user.message_text)

                        # Insertamos el precio min del item
                        cluster.update_one(
                            {'id': usuario['id']},
                            {"$set": {'products_temporal.price_product_max':
user.message_text}})

                        # Descarga del nuevo usuario
                        usuario = cluster.find_one({'id': usuario['id']})

                        # Introducimos finalmente el producto y precios en la
base de datos (lo otro era temporal)
                        cluster.update_one(
                            {
                                'id': usuario['id']
                            },
                            {
                                "$addToSet": {'products':
                                    {
                                        'name_product':
usuario['products_temporal']['name_product'],
                                        'price_product_min':
usuario['products_temporal']['price_product_min'],
                                        'price_product_max':
usuario['products_temporal']['price_product_max']
```

```
                                    }
                                }
                            }
                        )

                        # Descarga del usuario final
                        usuario = cluster.find_one({'id': usuario['id']})

                        # Mensaje de confirmación
                        if usuario['language'] == "es":
                            bot.send_message(
                                f"Se ha añadido
*{usuario['products'][len(usuario['products']) - 1]['name_product']}* entre"
                                f" *{usuario['products'][len(usuario['products'])
- 1]['price_product_min']}* "
                                f"{divisas_simbolo[usuario['currency']]} y "
                                f"*{usuario['products'][len(usuario['products'])
- 1]['price_product_max']}* "
                                f"{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])
                        else:
                            bot.send_message(
                                f"*{usuario['products'][len(usuario['products'])
- 1]['name_product']}* has been added "
                                f" *between
{usuario['products'][len(usuario['products']) - 1]['price_product_min']}* "
                                f"{divisas_simbolo[usuario['currency']]} and "
                                f"*{usuario['products'][len(usuario['products'])
- 1]['price_product_max']}* "
                                f"{divisas_simbolo[usuario['currency']]}",
                                usuario['id'])

                        # Envío de menu_tracking
                        if usuario['language'] == "es":
                            bot.send_message("*Elija una opcion*:",
usuario['id'], inline_keyboard(menu_tracking))
                        else:
                            bot.send_message("*Select an option*:",
usuario['id'], inline_keyboard(menu_tracking))

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 1, 0, 0, 0, 0)

                    except:
                        if usuario['language'] == "es":
                            bot.send_message(
                                "Ha ocurrido un error, escriba nuevamente el
precio máximo "
                                "(use el punto para los decimales, no la coma)",
usuario['id'])
                        else:
                            bot.send_message(
                                "An error has occurred, write the maximum price
```

```
again "
                                "(use the decimal point for not comma)",
usuario['id'])

                elif usuario['state_machine']['tracking'] == 5:  # Delete Item
                    # Se elimina del producto
                    cluster.update_one(
                        {'id': usuario['id']}, {"$pull": {'products':
{'name_product': user.message_text}}})

                    # Descarga del nuevo usuario
                    usuario = cluster.find_one({'id': usuario['id']})

                    # Envío del menú tracking
                    if usuario['language'] == "es":
                        bot.send_message("*Elija una opcion*:", usuario['id'],
inline_keyboard(menu_tracking))
                    else:
                        bot.send_message("*Select an option*:", usuario['id'],
inline_keyboard(menu_tracking))

                        # Se muestra toda la lista
                        try:
                            if usuario['language'] == "es":
                                text = ""
                                i = 0
                                for i in range(len(usuario['products'])):
                                    text = text + "\n\n📦*Producto*: {}
\n💰*Precio*: {} {} - {} {}".format(
                                        usuario['products'][i]['name_product'],
usuario['products'][i]['price_product_min'],
                                        divisas_simbolo[usuario['currency']],
usuario['products'][i]['price_product_max'],
                                        divisas_simbolo[usuario['currency']])
                                bot.send_message(text, usuario['id'])
                            else:
                                text = ""
                                i = 0
                                for i in range(len(usuario['products'])):
                                    text = text + "\n\n📦*Item*: {} \n💰*Price*:
{} {} - {} {}".format(
                                        usuario['products'][i]['name_product'],
usuario['products'][i]['price_product_min'],
                                        divisas_simbolo[usuario['currency']],
usuario['products'][i]['price_product_max'],
                                        divisas_simbolo[usuario['currency']])
                                bot.send_message(text, usuario['id'])
```

```
                        except:
                            if usuario['language'] == "es":
                                bot.send_message("No tiene productos en
seguimiento", usuario['id'])
                            else:
                                bot.send_message("You have no products on
tracking", usuario['id'])

                    # Actualización del estado de la máquina de estados
                    state(usuario['id'], 0, 0, 0, 1, 0, 0, 0, 0)

                elif usuario['state_machine']['country'] == 1:
                    pais = user.message_text
                    if pais in paises:
                        # Insertamos el nuevo país
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'country': paises[pais]}})

                        # Respuesta de confirmación
                        if usuario['language'] == "es":
                            bot.send_message("*Ha elegido:* {}".format(pais),
usuario['id'])
                        else:
                            bot.send_message("*You have selected:*
{}".format(pais), usuario['id'])

                        # Respuesta automática
                        default_reply(usuario)
                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)

                    else:
                        if usuario['language'] == "es":
                            bot.send_message("Ha ocurrido un error, seleccione un
país de la lista", usuario['id'])
                        else:
                            bot.send_message("An error has occurred, select a
country from the list", usuario['id'])

                elif usuario['state_machine']['language'] == 1:
                    language = user.message_text
                    if language in languages:
                        # Insertamos el nuevo idioma
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'language': languages[language]}})

                        # Descarga del nuevo usuario
                        usuario = cluster.find_one({'id': usuario['id']})

                        # Respuesta de confirmación
                        if usuario['language'] == "es":
                            bot.send_message("*Ha elegido:* {}".format(language),
usuario['id'])
```

```
                        else:
                            bot.send_message("*You have selected:*
{}".format(language), usuario['id'])

                        # Respuesta automática
                        default_reply(usuario)
                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 0, 0, 0, 0)

                    else:
                        if usuario['language'] == "es":
                            bot.send_message("Ha ocurrido un error, elija un
idioma de la lista", usuario['id'])
                        else:
                            bot.send_message("An error has occurred, select a
language from the list", usuario['id'])

                elif usuario['state_machine']['currency'] == 1:
                    divisa = user.message_text
                    if divisa in divisas:
                        # Actualizamos el rango de precios al cambio (en tiempo
real) con la divisa
                        cluster.update_one(
                            {
                                'id': usuario['id']
                            }, {
                                '$set':
                                    {
                                        'range.max': round(
                                            c.convert(usuario['range']['max'],
usuario['currency'], divisas[divisa]),
                                            2),
                                        'range.min':
round(c.convert(usuario['range']['min'], usuario['currency'],

divisas[divisa]), 2)
                                    }
                            }
                        )

                        # Insertamos la nueva divisa después de hacer el cambio
de precios para tener la anterior divisa
                        cluster.update_one({'id': usuario['id']}, {'$set':
{'currency': divisas[divisa]}})

                        # Respuesta de confirmación
                        if usuario['language'] == "es":
                            bot.send_message("*Ha elegido:* {}".format(divisa),
usuario['id'])
                        else:
                            bot.send_message("*You have selected:*
{}".format(divisa), usuario['id'])
```

```
                        # Respuesta automática
                        default_reply(usuario)

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)

                    else:
                        if usuario['language'] == "es":
                            bot.send_message("Ha ocurrido un error, elija una
divisa de la lista", usuario['id'])
                        else:
                            bot.send_message("An error has occurred, select a
currency from the list", usuario['id'])

                elif usuario['state_machine']['contact'] == 1:
                    try:
                        send_email(user.message_text)
                        if usuario['language'] == "es":
                            bot.send_message("Se ha enviado su mensaje",
usuario['id'])
                        else:
                            bot.send_message("Your message has been sent",
usuario['id'])

                        # Actualización del estado de la máquina de estados
                        state(usuario['id'], 0, 0, 0, 0, 0, 0, 0, 0)
                    except:
                        if usuario['language'] == "es":
                            bot.send_message("Ha habido un problema al enviar su
mensaje, pruebe a no incluir "
                                             "caracteres especiales, si el
problema persiste escriba el mensaje en "
                                             "inglés", usuario['id'])
                        else:
                            bot.send_message("An error has occurred, try to not
include special characters",
                                             usuario['id'])

                else:  # A cualquier otra orden el bot solo hace la respuesta por
defecto
                    default_reply(usuario)

            except:  # En caso de error se envía un correo al administrador para
informarle de que algo ha fallado
                send_email("Ha ocurrido un error, es necesario realizar un
mantenimiento")
                print("Error")
                pass
```

SCRAPING

PRODUCT_BANGGOOD

```
# Módulos
from Python_server.initialization import c


# Clase
class Banggood_Product:

    def __init__(self, text, rango_min, rango_max, divisa, country):
        self.name = text
        self.country = country
        self.link = self.create_link()
        self.ID_producto = []
        self.Precio_producto = []
        self.Nombre_producto = []
        self.Link_producto = []
        self.Imagen_producto = []

        # Rango de precios dado por el usuario
        rango_precios = [rango_min, rango_max]

        # Inicialización de request
        headers = {"User-Agent": 'Mozilla\5.0'}
        page = requests.session().get(self.link, headers=headers)
        page_soup = BeautifulSoup(page.content, "html.parser")  # también vale
lxml

        # Selección de los productos que son los que tienen la etiqueta shopslist
        items = page_soup.select(".shopslist ", namespaces=None)

        # Análisis de cada producto encontrado
        for i in range(len(items)):

self.Nombre_producto.append(items[i].contents[1].contents[3].contents[1]["title"]
)
            self.ID_producto.append(items[i]["data-pid"])
            # Se usa find() para el precio para disminuir las interacciones para
encontrar el producto y simplificar el código
            self.Precio_producto.append(
                round(c.convert(float(items[i].find("span", {"class": "price
wh_cn"})['oriprice']), 'USD', divisa), 2))

self.Link_producto.append(items[i].contents[1].contents[3].contents[1]["href"])

self.Imagen_producto.append(items[i].contents[1].contents[1].contents[1].contents
[1]["data-original"])

        # Selección de aquellos productos que están en el intervalo de precios
```

```
        if rango_precios:
            numero_productos = len(self.Precio_producto)
            for j in range(numero_productos):
                if float(rango_precios[0]) < float(self.Precio_producto[j]) <
float(rango_precios[1]):
                    self.ID_producto.append(self.ID_producto[j])
                    self.Precio_producto.append(self.Precio_producto[j])
                    self.Nombre_producto.append(self.Nombre_producto[j])
                    self.Link_producto.append(self.Link_producto[j])
                    self.Imagen_producto.append(self.Imagen_producto[j])
                else:
                    pass
        for j in range(numero_productos):
            self.ID_producto.pop(0)
            self.Precio_producto.pop(0)
            self.Nombre_producto.pop(0)
            self.Link_producto.pop(0)
            self.Imagen_producto.pop(0)
        else:
            pass

        # Se imprimen por pantalla los productos para que los vea el
desarrollador (opcional)
        print(self.Nombre_producto)
        print(self.Precio_producto)
        print(self.Link_producto)
        print(self.Imagen_producto)

    # Función
    def create_link(self):  # Crea el link de búsqueda en internet
        key_word = self.name.replace(" ", "%20")
        url = "https://www.banggood.com/search/{}.html".format(key_word)
        return url
```

PRODUCT_EBAY

```
# Módulos
from Python_server.initialization import c


# Clase
class Ebay_Product:

    def __init__(self, text, rango_min, rango_max, divisa, country):
        self.name = text
        self.country = country
        self.link = self.create_link()
        self.ID_producto = []
        self.Precio_producto = []
        self.Nombre_producto = []
        self.Link_producto = []
        self.Imagen_producto = []

        # Rango de precios dado por el usuario
        rango_precios = [rango_min, rango_max]
        headers = {"User-Agent": 'Mozilla\5.0'}
        page = requests.get(self.link, headers=headers)
        page_soup = BeautifulSoup(page.content, "html.parser")  # también vale
lxml

        # Selección de los productos que son los que tienen la etiqueta sresult
lvresult clearfix li shic
        items = page_soup.select(".sresult.lvresult.clearfix.li.shic",
namespaces=None)

        # Análisis de cada producto encontrado
        for i in range(len(items)):
            try:

self.Nombre_producto.append(items[i].contents[1].contents[1].contents[1].contents
[1]["alt"])
            except:

self.Nombre_producto.append(items[i].contents[1].contents[1].contents[5].contents
[1]["alt"])
            self.ID_producto.append(items[i]["listingid"])
            try:
                precio =
str(items[i].contents[5].contents[1].contents[1].contents[0])
                number = re.findall(r"[-+]?\d*\.\d+|\d+", precio)
                resul = float(number[0]) + (float(number[1]) / 100)
                self.Precio_producto.append(round(c.convert(resul, 'USD',
divisa), 2))
            except:
                try:
                    precio =
```

```
str(items[i].contents[5].contents[1].contents[1].contents[1].contents[0])
                    number = re.findall(r"[-+]?\d*\.\d+|\d+", precio)
                    resul = float(number[0]) + (float(number[1]) / 100)
                    self.Precio_producto.append(round(c.convert(resul, 'USD',
divisa), 2))
                except:
                    try:
                        precio =
str(items[i].contents[7].contents[1].contents[1].contents[0])
                        number = re.findall(r"[-+]?\d*\.\d+|\d+", precio)
                        resul = float(number[0]) + (float(number[1]) / 100)
                        self.Precio_producto.append(round(c.convert(resul, 'USD',
divisa), 2))
                    except:
                        try:
                            precio =
str(items[i].contents[7].contents[1].contents[1].contents[1].contents[0])
                            number = re.findall(r"[-+]?\d*\.\d+|\d+", precio)
                            resul = float(number[0]) + (float(number[1]) / 100)
                            self.Precio_producto.append(round(c.convert(resul,
'USD', divisa), 2))
                        except:
                            try:
                                precio =
str(items[i].contents[9].contents[1].contents[1].contents[0])
                                number = re.findall(r"[-+]?\d*\.\d+|\d+", precio)
                                resul = float(number[0]) + (float(number[1]) /
100)

self.Precio_producto.append(round(c.convert(resul, 'USD', divisa), 2))
                            except:
                                try:
                                    precio =
str(items[i].contents[9].contents[1].contents[1].contents[0])
                                    number = re.findall(r"[-+]?\d*\.\d+|\d+",
precio)
                                    resul = float(number[0]) + (float(number[1])
/ 100)

self.Precio_producto.append(round(c.convert(resul, 'USD', divisa), 2))
                                except:
                                    try:
                                        precio =
str(items[i].contents[11].contents[1].contents[1].contents[0])
                                        number = re.findall(r"[-+]?\d*\.\d+|\d+",
precio)
                                        resul = float(number[0]) +
(float(number[1]) / 100)

self.Precio_producto.append(round(c.convert(resul, 'USD', divisa), 2))
                                    except:
                                        precio = str(
```

```
items[i].contents[11].contents[1].contents[1].contents[1].contents[0])
                                        number = re.findall(r"[-+]?\d*\.\d+|\d+",
precio)
                                        resul = float(number[0]) +
(float(number[1]) / 100)

self.Precio_producto.append(round(c.convert(resul, 'USD', divisa), 2))
            try:

self.Link_producto.append(items[i].contents[1].contents[1].contents[1]["href"])
            except:

self.Link_producto.append(items[i].contents[1].contents[1].contents[5]["href"])
            if len(self.Link_producto[i]) > 300:    # recorta los links muy
largos
                recorta1 = self.Link_producto[i].index("?")
                recorta2 = self.Link_producto[i].index("&")
                recorta3 = self.Link_producto[i][140:].index("&")
                self.Link_producto[i] = self.Link_producto[i][0:(recorta1+1)] +
self.Link_producto[i][(recorta2+1):(140+recorta3)]
            try:

self.Imagen_producto.append(items[i].contents[1].contents[1].contents[1].contents
[1]["src"])
            except:

self.Imagen_producto.append(items[i].contents[1].contents[1].contents[5].contents
[1]["src"])

        # Selección de aquellos productos que están en el intervalo de precios
        if rango_precios:
            numero_productos = len(self.Precio_producto)
            for j in range(numero_productos):
                if float(rango_precios[0]) < float(self.Precio_producto[j]) <
float(rango_precios[1]):
                    self.ID_producto.append(self.ID_producto[j])
                    self.Precio_producto.append(self.Precio_producto[j])
                    self.Nombre_producto.append(self.Nombre_producto[j])
                    self.Link_producto.append(self.Link_producto[j])
                    self.Imagen_producto.append(self.Imagen_producto[j])
                else:
                    pass
        j = 0
        for j in range(numero_productos):
            self.ID_producto.pop(0)
            self.Precio_producto.pop(0)
            self.Nombre_producto.pop(0)
            self.Link_producto.pop(0)
            self.Imagen_producto.pop(0)
        else:
            pass

        # Se imprimen por pantalla los productos para que los vea el
```

```
desarrollador (opcional)
        print(self.Nombre_producto)
        print(self.Precio_producto)
        print(self.Link_producto)
        print(self.Imagen_producto)

    # Función
    def create_link(self):  # Crea el link de búsqueda en internet
        key_word = self.name.replace(" ", "%20")
        url = "https://www.ebay.es/sch/i.html?_nkw={}".format(key_word)
        return url
```

PRODUCT_GEARBEST

```
# Librerías
from bs4 import BeautifulSoup
import requests

# Módulos
from Python_server.initialization import c


# Clase
class Gearbest_Product:

    def __init__(self, text, rango_min, rango_max, divisa, country):
        self.name = text
        self.country = country
        self.link = self.create_link()
        self.ID_producto = []
        self.Precio_producto = []
        self.Nombre_producto = []
        self.Link_producto = []
        self.Imagen_producto = []

        # Rango de precios dado por el usuario
        rango_precios = [rango_min, rango_max]

        # Inicialización de request
        headers = {"User-Agent": 'Mozilla\5.0'}
        page = requests.session().get(self.link, headers=headers)
        page_soup = BeautifulSoup(page.content, "html.parser")  # también vale
lxml

        # Selección de los productos que son los que tienen la etiqueta
gbGoodsItem
        items = page_soup.select(".gbGoodsItem", namespaces=None)

        # Análisis de cada producto encontrado
        for i in range(len(items)-1):   # A veces el ultimo item que registra no
es un objeto y da errores
            try:

self.Nombre_producto.append(items[i].contents[3].contents[1]["title"])
            except:

self.Nombre_producto.append(items[i].contents[1].contents[1]["title"])
            self.ID_producto.append(items[i]["data-goods-id"])
            try:  # estos tienen descuento, por eso tienen un componente mas

self.Precio_producto.append(round(c.convert(items[i].contents[5].contents[3].text
, 'USD', divisa), 2))
                self.Link_producto.append(items[i].contents[3]["href"])
                self.Imagen_producto.append(items[i].contents[3]["data-img"])
```

```
            except:

self.Precio_producto.append(round(c.convert(items[i].contents[3].contents[3].text
, 'USD', divisa), 2))
                self.Link_producto.append(items[i].contents[1]["href"])
                self.Imagen_producto.append(items[i].contents[1]["data-img"])

        # Selección de aquellos productos que están en el intervalo de precios
        if rango_precios:
            numero_productos = len(self.Precio_producto)
            for j in range(numero_productos):
                if float(rango_precios[0]) < float(self.Precio_producto[j]) <
float(rango_precios[1]):
                    self.ID_producto.append(self.ID_producto[j])
                    self.Precio_producto.append(self.Precio_producto[j])
                    self.Nombre_producto.append(self.Nombre_producto[j])
                    self.Link_producto.append(self.Link_producto[j])
                    self.Imagen_producto.append(self.Imagen_producto[j])
                else:
                    pass
        for j in range(numero_productos):
            self.ID_producto.pop(0)
            self.Precio_producto.pop(0)
            self.Nombre_producto.pop(0)
            self.Link_producto.pop(0)
            self.Imagen_producto.pop(0)
        else:
            pass

        # Se imprimen por pantalla los productos para que los vea el
desarrollador (opcional)
        print(self.Nombre_producto)
        print(self.Precio_producto)
        print(self.Link_producto)
        print(self.Imagen_producto)

    # Función
    def create_link(self):  # Crea el link de búsqueda en internet
        key_word = self.name.replace(" ", "%20")
        url = "https://www.gearbest.com/sale/{}".format(key_word)
        return url
```

TELEGRAM_SERVER

BOT

```
# Librerías
import requests
import json
import configparser as cfg


# Clase
class telegram_chatbot():


    def __init__(self, config):
        self.token = self.read_token_from_config_file(config)
        self.base = "https://api.telegram.org/bot{}/".format(self.token)


    # Función que sirve para ver si ha habido un mensaje nuevo al bot en Telegram
    def get_updates(self, offset=None):
        url = self.base + "getUpdates"
        if offset:
            url = url +
"?offset={}&timeout=100&allowed_updates=[“message”]".format(offset+1)
        r = requests.get(url)
        return json.loads(r.content)


    # Función que envía un mensaje o un menú al usuario
    def send_message(self, msg, chat_id, reply_keyboard=None):
        if reply_keyboard is not None:
            url = self.base +
"sendMessage?chat_id={}&text={}&parse_mode={}&reply_markup={}".format(chat_id,
msg, "Markdown", json.dumps(reply_keyboard))
        else:
            url = self.base +
"sendMessage?chat_id={}&text={}&parse_mode={}".format(chat_id, msg, "Markdown")
```

```
        if msg is not None:
            requests.get(url)


    # Función que lee del config.cfg la contraseña del Totem
    def read_token_from_config_file(self, config):
        parser = cfg.ConfigParser()
        parser.read(config)
        return parser.get('token', 'token_telegram')
```

USER

```
# Clase

class User:


    def __init__(self, updates):

        for item in updates:

            self.update_id = item["update_id"]

            # El mensaje puede venir en diferentes formatos según el tipo de
mensaje que sea

            try:    # Mensaje de texto normal

                self.from_ = item["message"]["from"]["id"]

                self.chat_name = str(item["message"]["from"]["first_name"])

                self.date = item["message"]["date"]

                self.message_text = str(item["message"]["text"])

                self.language = str(item["message"]["from"]["language_code"])

                try:

                    self.chat_username = str(item["message"]["from"]["username"])

                except:

                    self.chat_username = None

                self.type_chat = item["message"]["chat"]["type"]

                self.type_message = "message"

            except:

                try:    # Mensaje que viene de un inline_keyboard

                    self.from_ = item["callback_query"]["message"]["chat"]["id"]

                    self.chat_name =
str(item["callback_query"]["from"]["first_name"])

                    self.date = item["callback_query"]["message"]["date"]

                    self.message_text = str(item["callback_query"]["data"])

                    self.language =
str(item["callback_query"]["from"]["language_code"])

                    try:

                        self.chat_username =
str(item["callback_query"]["message"]["from"]["username"])
```

```
                    except:

                        self.chat_username = None

                    self.type_chat =
item["callback_query"]["message"]["chat"]["type"]

                    self.type_message = "callback_query"

                except:  # Los siguientes tipos de mensajes son inútiles para el
Bot actualmente

                    # Únicamente se necesita del último except:, pero se deja el
resto de tipos para futuras ampliaciones del bot

                    try:    # Mensaje editado

                        self.from_ = item["edited_message"]["chat"]["id"]

                        self.chat_name =
str(item["edited_message"]["from"]["first_name"])

                        self.date = item["edited_message"]["date"]

                        self.message_text = str(item["edited_message"]["text"])

                        self.language =
str(item["edited_message"]["from"]["language_code"])

                        try:

                            self.chat_username =
str(item["edited_message"]["from"]["username"])

                        except:

                            self.chat_username = None

                        self.type_chat = item["edited_message"]["chat"]["type"]

                        self.type_message = "edited_message"

                    except:

                        try:    # Foto

                            self.from_ = item["message"]["from"]["id"]

                            self.chat_name =
str(item["message"]["from"]["first_name"])

                            self.date = item["message"]["date"]

                            self.message_text =
str(item["message"]["photo"][2]["file_id"])

                            self.language =
str(item["message"]["from"]["language_code"])

                            try:
```

```
                                self.chat_username =
str(item["message"]["from"]["username"])

                            except:

                                self.chat_username = None

                            self.type_chat = item["message"]["chat"]["type"]

                            self.type_message = "photo"

                        except:

                            try:    # Video

                                self.from_ = item["message"]["from"]["id"]

                                self.chat_name =
str(item["message"]["from"]["first_name"])

                                self.date = item["message"]["date"]

                                self.message_text =
str(item["message"]["video_note"]["file_id"])

                                self.language =
str(item["message"]["from"]["language_code"])

                                try:

                                    self.chat_username =
str(item["message"]["from"]["username"])

                                except:

                                    self.chat_username = None

                                self.type_chat = item["message"]["chat"]["type"]

                                self.type_message = "video"

                            except:

                                try:    # Gif

                                    self.from_ = item["message"]["from"]["id"]

                                    self.chat_name =
str(item["message"]["from"]["first_name"])

                                    self.date = item["message"]["date"]

                                    self.message_text =
str(item["message"]["animation"]["file_id"])

                                    self.language =
str(item["message"]["from"]["language_code"])

                                    try:
```

```
                                        self.chat_username =
str(item["message"]["from"]["username"])

                                    except:

                                        self.chat_username = None

                                    self.type_chat =
item["message"]["chat"]["type"]

                                    self.type_message = "animation"

                                except:

                                    try:    # Documento

                                        self.from_ =
item["message"]["from"]["id"]

                                        self.chat_name =
str(item["message"]["from"]["first_name"])

                                        self.date = item["message"]["date"]

                                        self.message_text =
str(item["message"]["document"]["file_id"])

                                        self.language =
str(item["message"]["from"]["language_code"])

                                        try:

                                            self.chat_username =
str(item["message"]["from"]["username"])

                                        except:

                                            self.chat_username = None

                                        self.type_chat =
item["message"]["chat"]["type"]

                                        self.type_message = "document"

                                    except:

                                        try:  # Sticker

                                            self.from_ =
item["message"]["from"]["id"]

                                            self.chat_name =
str(item["message"]["from"]["first_name"])

                                            self.date = item["message"]["date"]

                                            self.message_text =
str(item["message"]["sticker"]["file_id"])

                                            self.language =
str(item["message"]["from"]["language_code"])
```

```
                                            try:
                                                self.chat_username =
str(item["message"]["from"]["username"])
                                            except:
                                                self.chat_username = None
                                            self.type_chat =
item["message"]["chat"]["type"]
                                            self.type_message = "sticker"
                                        except:
                                            try:    # Mensaje de voz
                                                self.from_ =
item["message"]["from"]["id"]
                                                self.chat_name =
str(item["message"]["from"]["first_name"])
                                                self.date =
item["message"]["date"]
                                                self.message_text =
str(item["message"]["voice"]["file_id"])
                                                self.language =
str(item["message"]["from"]["language_code"])
                                                try:
                                                    self.chat_username =
str(item["message"]["from"]["username"])
                                                except:
                                                    self.chat_username = None
                                                self.type_chat =
item["message"]["chat"]["type"]
                                                self.type_message = "voice"
                                            except:
                                                try:  # Ubicación
                                                    self.from_ =
item["message"]["from"]["id"]
                                                    self.chat_name =
str(item["message"]["from"]["first_name"])
                                                    self.date =
item["message"]["date"]
```

```
                                                    self.message_text =
[item["message"]["location"]["latitude"],

item["message"]["location"]["longitude"]]

                                                    self.language =
str(item["message"]["from"]["language_code"])

                                                    try:

                                                        self.chat_username =
str(item["message"]["from"]["username"])

                                                    except:

                                                        self.chat_username = None

                                                    self.type_chat =
item["message"]["chat"]["type"]

                                                    self.type_message =
"location"

                                                except:

                                                    try:  # Ubicación en tiempo
real

                                                        self.from_ =
item["edited_message"]["from"]["id"]

                                                        self.chat_name =
str(item["edited_message"]["from"]["first_name"])

                                                        self.date =
item["edited_message"]["date"]

                                                        self.message_text =
[item["edited_message"]["location"]["latitude"],

item["edited_message"]["location"]["longitude"]]

                                                        self.language =
str(item["edited_message"]["from"]["language_code"])

                                                        try:

                                                            self.chat_username =
str(

item["edited_message"]["from"]["username"])

                                                        except:

                                                            self.chat_username =
None
```

```
                                                    self.type_chat =
item["edited_message"]["chat"]["type"]
                                                    self.type_message =
"live_location"
                                                except:
                                                    try:  # Encuesta
                                                        self.from_ =
item["message"]["from"]["id"]
                                                        self.chat_name = str(
item["message"]["from"]["first_name"])
                                                        self.date =
item["message"]["date"]
                                                        self.message_text =
item["message"]["poll"]["id"]
                                                        self.language = str(
item["message"]["from"]["language_code"])
                                                        try:
self.chat_username = str(
item["message"]["from"]["username"])
                                                        except:
self.chat_username = None
                                                        self.type_chat =
item["message"]["chat"]["type"]
                                                        self.type_message =
"poll"
                                                    except:
                                                        try:  # Música
                                                            self.from_ =
item["message"]["from"]["id"]
                                                            self.chat_name =
str(
item["message"]["from"]["first_name"])
```

```
                                                    self.date =
item["message"]["date"]
                                                    self.message_text
= item["message"]["audio"]["file_id"]
                                                    self.language =
str(
item["message"]["from"]["language_code"])
                                                    try:
self.chat_username = str(
item["message"]["from"]["username"])
                                                    except:
self.chat_username = None
                                                    self.type_chat =
item["message"]["chat"]["type"]
                                                    self.type_message
= "audio"
                                                except:
                                                    try:  # Contacto
                                                        self.from_ =
item["message"]["from"]["id"]
self.chat_name = str(
item["message"]["from"]["first_name"])
                                                        self.date =
item["message"]["date"]
self.message_text = item["message"]["contact"]["vcard"]
                                                        self.language
= str(
item["message"]["from"]["language_code"])
                                                        try:
self.chat_username = str(
```

```
item["message"]["from"]["username"])
                                                                except:

self.chat_username = None

self.type_chat = item["message"]["chat"][
                                                                    "type"]

self.type_message = "contact"
                                                            except: # Para
cualquier otra cosa
                                                                self.from_ =
""

self.chat_name = ""
                                                                self.date =
""

self.message_text = ""
                                                                self.language
= ""

self.chat_username = ""

self.type_chat = ""

self.type_message = "kk"
```