Telegram Bot - CryptoMarket

Autores: Carlos Piña Martínez y Diego del Corral Tercero

UNIVERSIDAD
COMPLUTENSE
MADRID

Trabajo de fin de grado del Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Director: Carlos Gregorio Rodríguez

Septiembre 2018

Índice

Índice de figuras

Resumen - summary

Español

La aparición de las criptomonedas ha supuesto una revolución en la manera de entender la economía introduciendo la idea de la gestión descentralizada y autónoma del dinero, donde no hay entidades ajenas encargadas del almacenaje y las transferencias. De este modo, son las propias personas las que toman el control sobre su economía.

CryptoMarket es un bot pensado como una herramienta gratuita mediante la cual poder acceder a la información más relevante de más de 5400 monedas y acercar así esta nueva tecnología a cualquier persona.

Haciendo una búsqueda rápida en los principales buscadores se pueden encontrar bots con una funcionalidad aparentemente similar, decimos aparentemente, porque tras estudiarlos nos dimos cuenta de que ninguno de ellos ofrecía una experiencia diferenciada para cada usuario y habitualmente su uso era complejo y enrevesado.

CryptoMarket ofrece la posibilidad de encapsular todas las monedas de interés para un usuario y así obtener una vista rápida del estado de estas. Esto se consigue mediante la cartera, en la cual los usuarios pueden añadir sus monedas y agruparlas.

Otro de los puntos diferenciadores de nuestro bot es la generación de gráficos en el momento, esto es, que cada vez que el usuario requiere un gráfico, este se genera descargando los datos más actuales, haciendo así que la información esté siempre actualizada. Además con los gráficos avanzados los usuarios menos experimentados optan a la generación, también en el momento, de señales de entrada y salida con las que pueden ver de una manera muy sencilla si sus valores están en un momento de crecimiento o por el contrario están debilitándose.

Uno de nuestros principales objetivos ha sido, dado que el bot trata una tecnología nueva y a menudo complicada como son las criptomonedas, acercarla a todos los usuarios, tanto los más experimentados como los más nuevos. Para conseguir esto hicimos una propuesta clara por la simplicidad y la usabilidad que creemos queda plasmada en la experiencia de los usuarios.

El bot final desarrollado se encuentra en el siguiente repositorio de GitHub:

https://github.com/dcorral3/CryptoPTB

English

The appearance of cryptocurrencies has become a revolution in the way of understanding the economy by introducing the idea of decentralized and autonomous management of money, where there are no outside entities responsible for storage and transfers. Now, it is the people themselves who take the power over their economy.

CryptoMarket is a bot designed as a free access tool through which get the most relevant information of over 5400 coins and bring closer this new technology to anyone.

Doing a quick search through the main search engines you can find bots with an apparently similar functionality, we say apparently, because after studying them we realized that none of them offered a differentiated experience for each user and its use was usually complex and convoluted.

CryptoMarket offers the possibility to encapsulate all the currencies of users interests and thus obtain a quick view of their status. This is achieved through the wallet, in which users can add their currencies and group them.

Another differentiating point of our bot is the generation of graphics on the fly, that is, whenever the user requires a graphic, it is generated by downloading the latest data making the information always up-to-date. In addition, the advanced graphics option lets less experienced users generate, also on the fly, entry and exit signals with which they can see in a very simple way if their values are in a moment of growth or conversely they are weakening.

Given that the bot deals with a new and often complicated technology such as cryptocurrencies, one of our main goals was to bring this new technology closer to both, experienced and new users. To achieve this we focused on simplicity and usability, we believe this is translated into a good user experience.

The final bot developed is in the following GitHub repository:

https://github.com/dcorral3/CryptoPTB

Palabras clave - keywords

	Español	English
1.	PTB (Python-Telegram-Bot)	PTB (Python-Telegram-Bot)
2.	CryptoMarket	CryptoMarket
3.	Matplotlib	Matplotlib
4.	CryptoCompare	CryptoCompare
5.	MongoDB	MongoDB
6.	Telegram Bot	Telegram Bot
7.	Criptomonedas	Cryptocurrencies
8.	Python	Python
9.	API (Interfaz de Programación de Aplicaciones)	API (Aplication Program Interface)
10.	Telepot	Telepot

Introducción

Español

Este apartado es una pequeña introducción para que el lector entienda el contexto y la organización del documento que va a leer.

En los últimos años la tecnología ha cambiado la forma en la que interactuamos con el mundo en prácticamente todos los aspectos. Uno de los paradigmas más revolucionarios surgido en las últimas fechas ha sido la aparición de las criptomonedas.

Desde la introducción de la moneda como valor de intercambio de bienes, las economías se han basado en un sistema de confianza y de regulación estatal que marcaba la producción, cantidad y valor de estas monedas basándose en un sistema centralizado, donde el estado y los mercados marcaban la norma. La creación de las criptomonedas cambia radicalmente el panorama económico introduciendo un sistema revolucionario y descentralizado, donde ningún poder estatal o económico puede imponer sus normas y son los propios poseedores los responsables de las transacciones y su validación, sin necesidad de intermediarios. Esta nueva forma monetaria trae consigo uno de los mayores avances, el sistema de transacciones basado en algoritmos matemáticos y el aprovechamiento de los recursos de internet. Esto ha hecho que las transacciones de capital no solo sean más seguras sino que son prácticamente instantáneas, cosa que hasta ahora no se había conseguido.

La idea de consolidar una economía descentralizada donde los usuarios tengan el control de su economía, nos llevó a la idea de crear una herramienta donde los usuarios tuvieran un acceso más directo al estado de las monedas virtuales. Para ello, después de valorar las distintas opciones, concluimos que Telegram era la herramienta perfecta para empezar a desarrollar este proyecto por su naturaleza OpenSource y la tipología de sus usuarios.

En la actualidad, Telegram es una aplicación multiplataforma que supera los 200 millones de usuarios activos por lo que dispondremos de un gran público al que poder ofrecer nuestro bot. Además, al ser multiplataforma el bot puede ser utilizado tanto en PC como en dispositivos móviles, lo que hace que sea aún más accesible. Otra de las ventajas de los bots sobre una app convencional es que no requiere de un proceso de instalación y registro. Usar nuestro bot es tan sencillo como iniciar una conversación con éste y automáticamente se añade a la lista de chats de Telegram. Gracias a los bots Telegram se puede convertir en una aplicación multiusos, ya que algunos de ellos pueden sustituir aplicaciones que utilizamos en el día a día, con el ahorro de espacio que conlleva, además de la comodidad de tener gran parte de las funcionalidades que cada usuario necesita concentradas en una sola aplicación.

Nuestro objetivo principal al crear este bot es crear una herramienta de acceso gratuito a todos aquellos usuarios interesados en esta nueva forma de comercio descentralizado y autogestionado. Una de las metas más importantes para nosotros era conseguir que el usuario tuviera una experiencia fluida y agradable a la hora de consultar el estado de sus monedas virtuales, proporcionando un servicio rápido, sencillo y visual. Por otro lado hemos puesto mucho esfuerzo en conseguir que el

proyecto siguiera una estructura clara y legible para facilitar las posibles aportaciones de terceros y mejorar así la mantenibilidad y la introducción de futuras mejoras.

Otro de nuestros objetivos más importantes era utilizar lenguajes y tecnologías que no habíamos aprendido durante la carrera y que nos pudieran aportar conocimientos bien valorados en el mundo laboral. Dado que en la carrera no habíamos aprendido Python, Git ni a usar bases de datos no relacionales, nos decidimos a hacer el proyecto utilizando principalmente estas tres herramientas. A medida que hemos ido avanzando en el desarrollo del bot, hemos ido conociendo más a fondo sus utilidades, ventajas y limitaciones, hasta conseguir un nivel más avanzado.

Por otra parte Telegram siempre ha sido una herramienta que nos ha suscitado un enorme interés, por su filosofía open source y su alto promedio de actualizaciones, en las cuales introducen siempre funcionalidades requeridas por los usuarios. Utilizando la API, hemos visto que no solo ofrece muchas posibilidades a los usuarios sino que también cuidan a los desarrolladores manteniendo una documentación clara y una API que ofrece infinidad de métodos para ayudar a crear un ecosistema dentro de Telegram en el cual poder desarrollar tus ideas.

A continuación se enumeran los distintos apartados de este documento con un breve texto resumiendo su contenido y como se debe afrontar la lectura de los mismos:

La estructura de esta memoria es la siguiente:

1. Comparativa de herramientas y librerías. En este apartado se realizará un análisis de las diferentes librerías y APIs que hemos valorado para realizar este trabajo y finalmente con cuales nos hemos quedado. Se habla también de las diferentes herramientas que se han utilizado para el desarrollo del mismo explicando las razones por las que se han utilizado.

2. Fase inicial, familiarización con el entorno y configuración. Apartado en el que se describe el proceso que se ha seguido para familiarizarse con las herramientas, APIs, lenguajes de programación, etc. que se han utilizado durante el proyecto. En los casos que se considera oportuno también se describe el proceso de instalación y configuración.

3. CryptoMarket bot. En este apartado se explica detalladamente el proceso de desarrollo del bot final, CryptoMarket. En él se analizan detalles técnicos y problemas encontrados durante el desarrollo de los distintos evolutivos y sus soluciones.

4. Implementaciones y mejoras futuras. Aquí se plasman ideas que tenemos pensadas para desarrollar en un futuro, así como mejoras del bot actual.

5. Conclusiones. En este apartado se describen nuestras experiencias realizando este proyecto y las correspondientes conclusiones que obtenemos de ellas.

English

This section acts as a brief introduction for the reader to understand the context and the organization of this document.

In the last years, technology has changed the way we interact with the world in almost every aspect. One of the most revolutionary paradigms introduced in the last times has been the emergence of cryptocurrencies.

Since the introduction of the coin as a method for the exchange of goods, the economies have been based in a trust model and state regulations that controlled the production, quantity and value of these coins, working in a centralized system, where the government and the markets imposed their rules. The appearance of cryptocurrencies changes radically the actual economic scene, by introducing a revolutionary and decentralized system, where no economic or governmental power can impose its rules, thus are the own users responsible for the transactions and validation with no need of brokers. This new monetary order brings one of the greatest advance, a transaction method built using mathematical algorithms and the exploitation of internet resources. This has made the flow of capital not only to be more secure but also almost instantaneous.

The idea of consolidating a decentralized economy where users have control of their personal economy, drove us to the idea of creating a tool to have a direct access to the state of the users virtual coins. To achieve this, we concluded that Telegram was the perfect environment to start developing this project, because it is open source and the typology of its active users.

Nowadays Telegram is a multiplatform application that overcomes the 200 million active users, this provides us with a great audience to which offer our bot. Because of its multiplatform nature, this bot can be used in PCs and Android or iOS devices which makes the tool more accessible. Another advantage of bots over conventional native applications is that bots do not require an installation and registry process. Using our bot is as easy as starting a new chat inside the Telegram application.
Thanks to bots, Telegram becomes a multifunction application, this happens because bots can substitute applications that we use on a daily basis, with great memory saving and the commodity of having all functionalities centered in a single application.

Our main goal when creating this bot is to make a free access tool for all those users interested in this new decentralized and self-managed way of commerce. One of our most important milestones was to achieve a fluid and comfortable user experience when consulting the state of virtual coins, supplying a fast, visual and easy service. In the other hand we have put a lot of our effort in having a clear and legible structure to facilitate possible contributions and this way, improving maintainability and easing the adoption of future features.

Also important was to use programming languages and technologies that we had not learned during our academic life that could give us well valued knowledge in the labour market. Given that we have not learned Python, Git nor non relational databases, we decided to start the project using these three technologies. During the

development process we got to know more in depth the advantages and limitations of these technologies and achieved a higher understanding level.

Telegram has always been a tool that has catched our attention, because of it open source philosophy
and the high rate of updates, always introducing new functionalities based on the community needs.
Using its API, we have seen that it offers not only a great number of possibilities to the users but also care about developers keeping a clear documentation and an API that provides infinite methods to help growing an ecosystem inside Telegram to develop new ideas.

Next you can find the different sections in this document with a brief text describing its content and a how to read guide:

The structure of this paper is as follows:

1. Comparative of tools and libraries. In this section you will find an analysis of the different libraries and APIs that we have taken into account to develop this project. Here also we talk about the libraries that were finally chosen and the reasons why they were chosen.

2. Early stage, familiarization with the environment and configuration. Brief description of the processes followed to get familiar with the tools, APIs and programming languages. In some cases there are also some instructions on how to install and configure.

3. CryptoMarket bot. In this section you will find a detailed explanation of the development process of CryptoMarket. More technical details and problems found will also be discussed over this part, talking about the new functionalities and solutions found.

4. Implementations and future work. Ideas for new features and improvements to be developed in the future.

5. Conclusions. Description of our overall experience during the development process and the obtained conclusions.

1. Comparativa de herramientas y librerías

1.1. Git

Como ya es habitual y prácticamente obligatorio, se ha utilizado GIT como herramienta de control de versiones. Ésta herramienta facilita el trabajo colaborativo, ya que permite el desarrollo seguro en paralelo y facilita la posterior integración de estos desarrollos. Además coger soltura con ésta herramienta será de gran utilidad en nuestro futuro laboral, ya que en casi la totalidad de las ofertas de trabajo se pide como requisito el manejo de herramientas de este tipo.

1.2. Python

Una de las premisas que nos propusimos con el TFG fue aprender cosas nuevas. Desde un principio se nos propuso utilizar este lenguaje para el desarrollo del TFG y tras analizar sus posibilidades, ventajas y desventajas no lo dudamos. Habíamos oído hablar del potencial del lenguaje y de su gran popularidad (está dentro del top 5 de lenguajes de programación más utilizados) pero nunca habíamos usado este lenguaje por lo que no podíamos desaprovechar la oportunidad que se nos ofrecía. La buena documentación publicada junto con la gran comunidad y la cantidad de APIs para extender la funcionalidad del lenguaje que hay disponibles nos ayudarían en nuestros desarrollos.

1.3. Telepot vs Python-Telegram-Bot

Telepot está publicada bajo la licencia MIT lo que nos permite usar la API libremente. Por otro lado Python-Telegram-Bot posee la licencia LGPLv3, ésta licencia nos permite usar la API siempre y cuando nuestro código sea público y bajo ésta misma licencia.

Ambas librerías son un envoltorio alrededor de la API oficial que proporciona Telegram. Esto significa que, esencialmente, adaptan los métodos y objetos de la API original y los traducen a métodos y objetos de Python. Con esto, se quiere conseguir la integración de esta herramienta en cualquier proyecto en Python y aprovechar así las virtudes de este lenguaje para un desarrollo más fluido. Este tipo de librerías existen también para otros lenguajes y son igualmente utilizadas en distintos proyectos software.

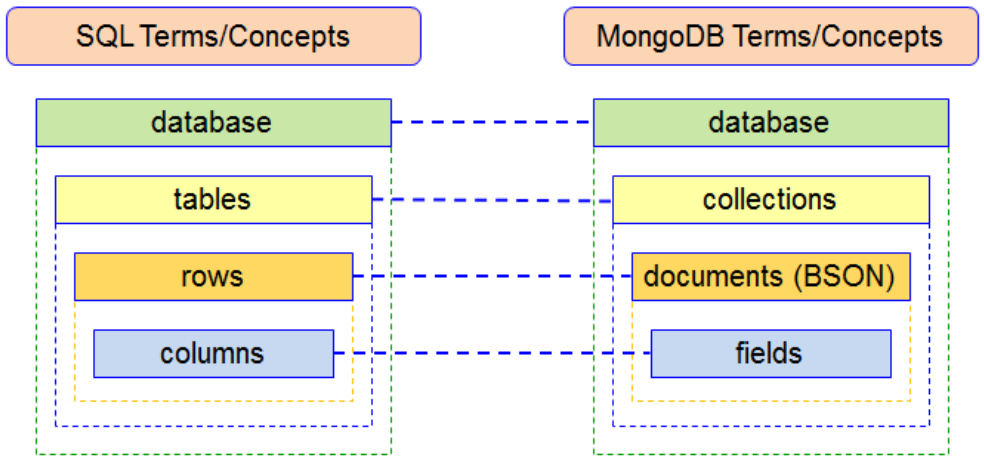
La diferencia entre estas dos librerías se reduce a los métodos complementarios que proporcionan a los desarrolladores. Estos métodos son una extensión particular de cada proyecto que usualmente se enfocan en facilitar la labor de los desarrolladores, se pueden encontrar en las páginas dedicadas a la documentación.

Después de hacer distintas pruebas con estas dos librerías nos decantamos por desarrollar el bot final con Python-Telegram-Bot ya que proporciona una serie de métodos y objetos como los Handlers y el Dispatcher, con los cuales ya estamos familiarizados y nos ayudarían en la implementación y entendimiento de la estructura de nuestro proyecto.

1.4. SQL vs MongoDB

Para nuestro proyecto teníamos que decidir qué tipo de base de datos se ajustaría mejor a las necesidades del bot y el proceso de desarrollo. En un primer momento pensamos que con nuestra experiencia adquirida durante la carrera, nos sería más fácil usar una base de datos relacional como MySQL o PostgreSQL, ambas basadas en un sistema de tablas relacionadas y consultas en SQL.

Por otro lado estaban las bases de datos no relacionales como MongoDB o NoSQL. Este tipo de base de datos nos permitiría empezar el desarrollo del bot más rápidamente, ya que no requieren de un modelo predefinido. Además MongoDB trabaja con documentos en formato JSON, este formato es usado para el tratamiento de datos en las APIs de las cuales sacaremos la información que necesitamos sobre las criptomonedas. El factor que nos llevó a elegir MongoDB como base de datos fue su facilidad de instalación y de integración con el proyecto. Tras visitar la documentación oficial y los tutoriales introductorios nos decantamos por esta tecnología para nuestro back-end, que además nos serviría una vez más para aprender una nueva tecnología ya que ninguno habíamos usado una base de datos no relacional anteriormente.



1 Equivalencia entre SQL y MongoDB

1.5. Matplotlib

Una de las funcionalidades que queríamos dar al bot era la de mostrar gráficas, ya que es una forma atractiva y rápida de ver la evolución de los valores de las criptomonedas. Para implementar esta funcionalidad decidimos usar la librería Matplotlib, ya que era sencilla de usar y había gran cantidad de ejemplos ilustrativos online. Esta librería permite generar gráficas con un alto nivel de personalización de forma sencilla y en pocas líneas de código.

Matplotlib utiliza listas de datos para generar las gráficas, los datos que nosotros extraemos de las APIs nos son devueltos en formato JSON, por lo que nos es muy sencillo crear listas a partir de estos datos para que ésta librería genere las gráficas correspondientes.

1.6. APIs valoradas

En este apartado nos centrarnos en las APIs de criptomonedas que hemos valorado para obtener los datos que ofrecemos en CryptoMarket bot.

- **APIs seleccionadas:**

    - **CrpytoCompare**: API bajo la licencia CC BY-NC 3.0 (Creative Commons - Attribution Non-Commercial), ésta licencia nos permite utilizarla de forma gratuita siempre y cuando se atribuya la autoría a CryptoCompare y no se use con fines comerciales. Por lo que como nuestro bot es totalmente gratuito podemos utilizarla con total libertad.

      Esta API la hemos utilizado para la obtención de datos históricos sobre las criptomonedas, como por ejemplo el valor del bitcoin en los últimos días. Posteriormente hemos utilizado estos datos para la generación de gráficas mediante Matplotlib. También obtenemos el listado de todas las criptomonedas que poseen. Este listado es guardado en nuestra base de datos para tener un acceso más rápido a los datos, ya que éstos se usarán para formar distintas consultas a ésta misma API o a la de CoinMarketCap.

      URL: https://min-api.cryptocompare.com/

    - **CoinMarketCap**: según los términos y condiciones de uso de la API, se pueden usar sus datos siempre y cuando no tengan un fin comercial. Las demás restricciones no nos influyen ya que nosotros no utilizamos ni modificamos su código en ninguna parte, solamente usamos los datos que nos proporcionan sus servicios.

      De ésta API obtenemos datos más concretos de las criptomonedas así como distintos listados de criptomonedas que nos proporcionan, como por ejemplo, el Top 10.

      URL: https://coinmarketcap.com/es/api/

- **APIs descartadas:**

  También valoramos el uso de otras APIs de criptomonedas, como por ejemplo CoinApi.io y Coinigy, pero las principales razones por las que fueron descartadas fueron que, tanto CryptoCompare como CoinMarketCap nos facilitaban los datos que necesitábamos sin necesidad de solicitar un API key. Otra de las razones por las que también fueron descartadas algunas fue que para tener un número razonable de peticiones diarias a la API había que abonar diferentes tarifas premium.

1.7. Editores / IDEs

Inicialmente, al comenzar a familiarizarnos con Python y la API de Telepot (ésta API fue utilizada en los bots iniciales) usábamos editores de texto como Sublime Text o Atom, ya que el código era más sencillo y no necesitábamos grandes cantidades de

código para hacerlos funcionar, por lo que no nos surgió la necesidad de utilizar IDEs más complejos que nos facilitasen la implementación de éstos bots.

Una vez que el proyecto de CryptoMarket comenzó a crecer comenzamos a echar en falta que los editores nos ayudasen con la sintaxis para poder avanzar más rápidamente y de una forma más cómoda. Uno de nosotros optó por el IDE más recomendado por la comunidad de Python, PyCharm, mientras que el otro optó por Visual Studio Code.

Pycharm es un IDE muy potente para programar en Python, desarrollado por JetBrains (empresa escogida por Google para desarrollar el IDE por excelencia para la construcción de aplicaciones Android, Android Studio). La versión gratuita de PyCharm, denominada Community, es lo suficientemente potente como para agilizar la construcción de código en Python, permitiendo la instalación de plugins, marcando errores de sintaxis sin tener la necesidad de ejecutar el código, realizando sugerencias, permitiendo la navegación intuitiva por los proyectos, integración con herramientas de control de versiones como Git, etc.

Por otro lado, Visual Studio Code. Este editor de texto se ha convertido en los últimos años en una herramienta muy utilizada por los desarrolladores ya que su desarrollo se ha centrado en convertirse en una herramienta útil para desarrolladores de aplicaciones web. Con su sistema de extensiones puede dar soporte a cualquier lenguaje de programación. Este editor está registrado por Microsoft aunque el código cuenta con una licencia MIT que permite tanto su uso privado, como comercial, de modificación y de distribución. Entre las características más útiles de este editor se encuentra el buscador y organizador de archivos integrado completamente con Git, indicando en cada momento las modificaciones que se están llevando a cabo en cada archivo. Como otros muchos editores también cuenta con la tecnología IntelliSense que nos permite acceder a los métodos y atributos de cada variable, función u objeto a modo de sugerencia a la misma vez que escribimos el código.

1.8. Servidores (isabelita + personal + local)

Tanto para realizar pruebas con los bots que íbamos desarrollando como para dejar corriendo en segundo plano estos bots disponíamos de 3 entornos:

1. **Isabelita:** Servidor cedido por la Universidad Complutense de Madrid. A este servidor se nos concedió acceso mediante clave pública-privada. Como no disponemos de permisos de administrador en este servidor necesitamos solicitar a nuestro director, Carlos Gregorio, las instalaciones de librerías que necesitamos, entre otras cosas.

2. **Servidor personal:** Consiste en una máquina personal a la cual tenemos acceso a través de ssh con privilegios de administrador y en el cual podemos hacer las configuraciones e instalaciones que consideramos necesarias. Decidimos instalar la última versión de Ubuntu Server (Ubuntu Server 18.04) ya que es una de las distribuciones más extendidas y con más soporte por parte de la comunidad.

   Para poder acceder y comprobar el estado del servidor en cualquier momento, decidimos hacer un pequeño bot de Telegram, usando la librería Telepot.

Con este bot podíamos acceder a la IP pública de la máquina y actualizar la IP en el servicio DNS ofrecido por No-IP. Además este bot nos notificaría cada vez que se produjese un cambio de IP o un reinicio de la máquina. Esto ha sido totalmente necesario ya que no disponíamos de una IP estática.

Este servidor es usado para tener siempre activa la última versión funcional de nuestro bot.

3. **Localhost:** Como es ya habitual en todo tipo de desarrollos hemos utilizado entornos de prueba y corrección de errores en nuestros ordenadores personales. Estos entornos a diferencia de los mencionados anteriormente no eran accesibles de forma remota, se lanzan de manera ocasional y sólo con propósitos de testeo.

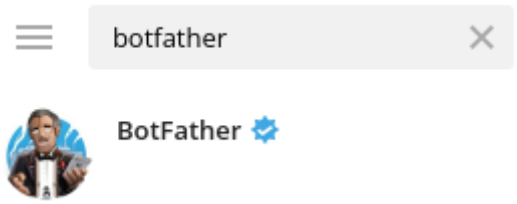
2. Fase inicial, familiarización con el entorno y configuración

2.1. Creación bots (BotFather)

La creación de bots en Telegram se hace a través de BotFather. Con este bot se pueden hacer todas las tareas básicas de gestión de bots: crear nuevos bots, eliminar bots, gestionar descripción, foto y comandos de un bot, etc.

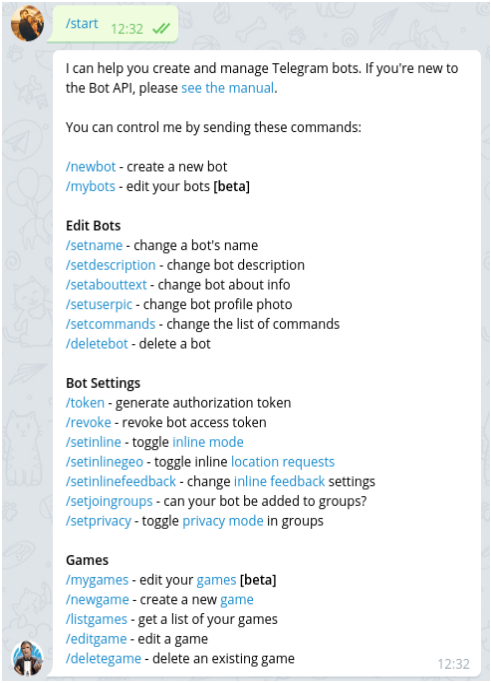
El proceso de creación de un bot sería es el siguiente:

En primer lugar se introduce 'BotFather' en la barra de búsqueda de Telegram.



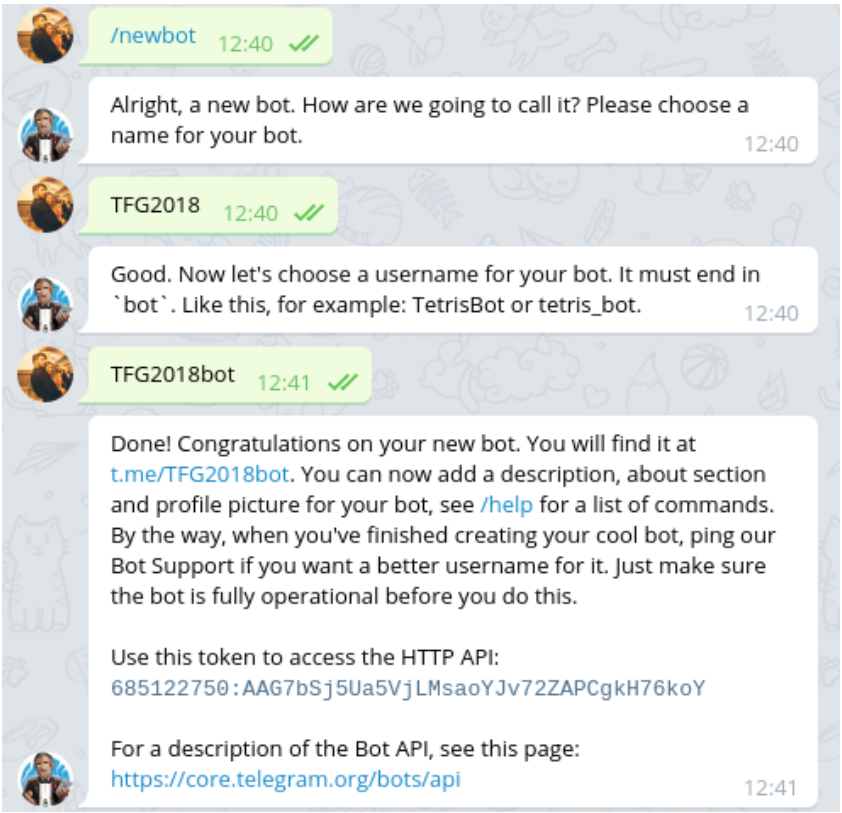
2 BotFather

Después se inicia el bot mandando el comando 'start' y nos mandará una lista con todos los comandos que podemos usar. En nuestro caso el que queremos usar es el de crear nuevo bot '/newbot'.



3 Comandos admitidos por BotFather

Después de introducir el nombre de nuestro bot y un identificador único, nuestro bot será creado y BotFather nos mandará el *token* con el que podremos acceder a la API HTTP que proporciona Telegram.



4 Creación de un bot con BotFather

Este token debe mantenerse siempre oculto ya que cualquier persona con acceso podría modificar el funcionamiento de nuestro bot.

2.2. Familiarización con Python

Para empezar a conocer el lenguaje de nuestro proyecto hicimos el tutorial que viene marcado en la documentación oficial de Python3+ (https://docs.python.org/3/tutorial/) En este tutorial se puede aprender la sintaxis del lenguaje así como métodos de depuración y buenas prácticas de programación. Una vez aprendidas las nociones básicas del lenguaje, comenzamos a hacer los tutoriales introductorios de las librerías que usaríamos para nuestro proyecto.

2.3. Familiarización con librerías

La familiarización con las librerías de Telepot y Python-Telegram-Bot se convierte en un proceso sencillo gracias a la buena documentación que ofrecen ambos proyectos.

Telepot, en su tutorial de iniciación te enseña a crear un bot básico que responderá a tus mensajes y comandos con una respuesta simple. De este modo se aprende el funcionamiento básico y el flujo por el que pasan los mensajes y respuestas, así como la información que se recoge en cada mensaje del usuario.

Por otro lado Python-Telegram-Bot en su tutorial 'Your first Bot' nos introduce los métodos de su librería que nos servirán para empezar a desarrollar cualquier tipo de Bot. También explican métodos para crear nuestro propio sistema de logs así como el manejo de errores.

Una vez vistos los tutoriales empezamos con la creación de nuestros propios Bots para entender más a fondo el funcionamiento de estas dos librerías.

2.4. Primeros bots

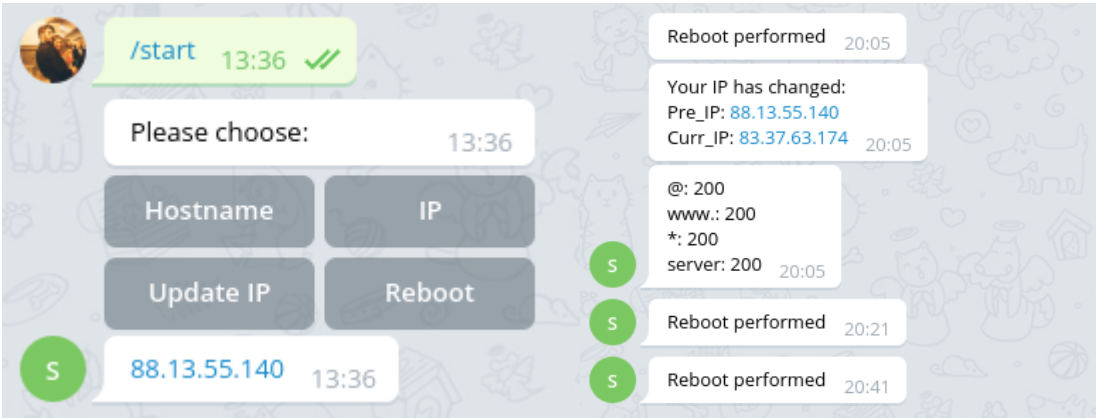
Empezamos creando dos bot con la librería Telepot. Uno de ellos devolvía en tiempo real el valor del Bitcoin en dólares americanos, el otro consultaba una API con los datos actualizados del popular juego Clash Royale, la información que facilitaba este bot era información pública del usuario que solicitábamos a la API *RoyaleAPI (https://docs.royaleapi.com/#/)*, ésta información se obtenía indicando a la API el id único del usuario del que se quería obtener la información. Estos dos bots sencillos nos sirvieron para familiarizarnos principalmente con el uso de las librerías json y request, ya que estas dos librerías son muy usadas en el bot final, CryptoMarket.

En una de las actualizaciones de la API de Clash Royale los desarrolladores decidieron añadir el uso de *tokens* para autorizar el uso de la API a los desarrolladores externos, ya que esta comenzó a tener mayor popularidad y necesitaban regular el uso por temas de estabilidad y sobrecargas de sus servidores. Adaptarnos a ésta actualización nos sirvió para familiarizarnos con el uso de *tokens* en las llamadas a las APIs, todo esto con ayuda de un pequeño tutorial que los desarrolladores de la API publicaron para distintos lenguajes de programación, Python entre ellos.

Al mismo tiempo comenzamos con la configuración del servidor y para ello decidimos crear otro bot que se encargase de darnos información relevante, así como

efectuar algunos comandos básicos para asegurar la conexión a este. Este bot se puede encontrar en el siguiente enlace y sus tareas principales son:

1. Devolver el nombre de la máquina en que se está ejecutando
2. Consultar la ip pública de la máquina
3. Restaurar la IP pública a través de la API de No-IP, servicio de DDNS gratuito.
4. Notificar cuando se produce un cambio de IP o cuando se reinicia el servidor



5 Chat con ServerBot (control del servidor personal)

Este bot se puede encontrar en el siguiente enlace:
https://github.com/dcorral3/serverbot

2.5. Instalación y familiarización con MongoDB

Como se ha mencionado anteriormente, decidimos usar MongoDB para nuestra base de datos. MongoDB está incluido en los repositorios oficiales de Ubuntu, pero después de informarnos nos dimos cuenta de que la mejor forma de instalarlo es mediante el repositorio oficial, ya que nos garantiza que siempre vamos a estar actualizados a la última versión. El proceso de instalación de MongoDB tanto en nuestros equipos personales como en los servidores (Isabelita y personal) fue el siguiente:

Lo primero que tuvimos que hacer fue importar la clave del repositorio oficial de MongoDB ya que Ubuntu necesita verificar la autenticidad de los paquetes que están firmados con claves GPG y posteriormente añadimos los detalles del repositorio para que el comando *apt* conozca de dónde tiene que descargar los paquetes para la instalación. Para esto se ejecutan los siguientes comandos en la terminal:

$ sudo apt-key adv --keyserver hkp: //keyserver.ubuntu.com: 80 --recv 0C49F3730359A14518585931BC711F9BA15703C6

$ echo "deb [arch = amd64, arm64] http://repo.mongodb.org/apt/ubuntu xenial / mongodb-org / 3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

$ sudo apt-get update

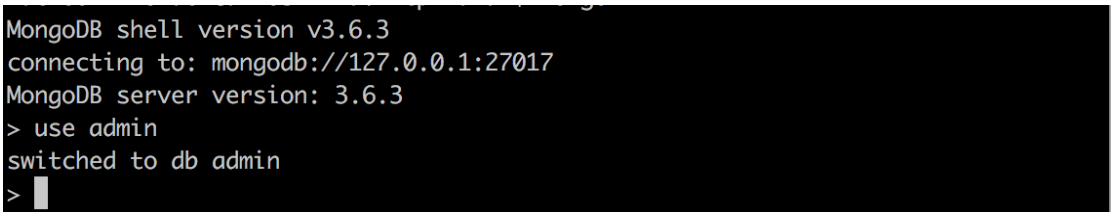
$ sudo apt-get install mongodb-org

Una vez termina de ejecutar el último comando ya es posible utilizar MongoDB normalmente escribiendo *'mongo'* en la terminal. Al ejecutar este comando nos dimos cuenta de que aparecían varios mensajes de *warning* que nos indicaban que la base de datos no era segura así que nos informamos de cómo securizarla. Para ello encontramos el siguiente tutorial:

https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-mongodb-on-ubuntu-16-04

En él se nos indicaba paso por paso como hacerlo explicando en cada uno de los pasos que era lo que estábamos haciendo.

Lo primero de todo fue añadir un usuario administrador, para ello, desde la terminal de MongoDB indicábamos que queríamos usar la base de datos 'admin'.



6 Shell MongoDB

A continuación insertamos un usuario en esta base de datos con la siguiente estructura :

```
> db.createUser ({
        usuario: "admin",
        pwd: "adminPwd",
        roles: [{role: "userAdminAnyDatabase", db: "admin"}]
 })
```

La parte más importante de la estructura anterior para añadir el usuario es la línea:

roles: [{role: "userAdminAnyDatabase", db: "admin"}]

Ésta línea indica que éste usuario tiene los permisos necesarios para ser administrador en todas las bases de datos de MongoDB que se creen en el equipo o servidor correspondiente.

El siguiente paso fue habilitar la autenticación de MongoDB ya que sino no se piden las credenciales del usuario administrador para interactuar con las bases de datos de MongoDB. Para habilitarlo había que modificar el archivo mongod.conf ubicado en la ruta /etc/mongod.conf y añadir (o descomentar en caso de que ya exista) las líneas:

```
security:
  authorization: "enabled"
```

Es importante que la primera línea (security:) no tenga espacios al principio y que la segunda línea (authorization: "enabled") tenga exactamente dos espacios de sangría al inicio ya que sino los cambios no tendrán efecto y la autorización no será requerida para interactuar con las bases de datos.

Después de guardar los cambios realizados en el archivo tenemos que reiniciar MongoDB para que los cambios tengan efecto.

Una vez hecho todo esto, creamos otro usuario para la base de datos que utilizaríamos en el bot CryptoMarket con la siguiente estructura:

```
> db.createUser ({
        usuario: "user",
        pwd: "userPwd",
        roles: [{role: "readwrite", db: "botdb"}]
 })
```

A este usuario sin embargo le asignamos permisos de lectura y escritura solo en la base de datos del bot (botdb en el caso del ejemplo).

Una vez realizados todos los pasos anteriores ya teníamos la base de datos lista para ser utilizada.

3. CryptoMarket bot

3.1. Idea inicial

Antes de empezar a trabajar en el proyecto hicimos una breve investigación sobre los bots que existen y sus diferentes utilidades, así como las necesidades emergentes de los usuarios frecuentes en Telegram. Como nosotros, la mayor parte de usuarios activos en telegram están muy relacionados con el mundo del software y las últimas tecnologías así que pensamos en juntar esta herramienta con otra de las novedades de los últimos años, las criptomonedas.

Empezamos por buscar bots desarrollados para Telegram relacionados con las monedas virtuales. Para nuestra sorpresa los bots eran muy básicos y solo te permitían ver el valor de algunas de las criptomonedas más comunes o eran demasiado complejos y difíciles de usar, normalmente la interacción no era muy intuitiva para el usuario. Por esta razón decidimos empezar a construir un bot sencillo, que proporcionase información relevante de la mayoría de las monedas y donde los usuarios pudieran ver rápidamente qué estaba pasando con las criptomonedas de su interés.

3.2. Evolución

En este apartado hablaremos de la evolución del proyecto y cómo fueron surgiendo las ideas que luego llevamos a cabo. Más adelante en el apartado de documentación funcional se hará un análisis técnico de cada funcionalidad y su estructura.

Empezamos desarrollando un sistema muy sencillo donde se pudiera consultar el precio de las monedas con más volumen de mercado mediante un teclado en pantalla. Ya habíamos creado un bot sencillo pero rápido.

Necesitábamos algo que nos distinguiese del resto del bots y dado a que las APIs que usábamos ofrecían infinidad de datos sobre miles de monedas (más de 5.400 monedas), comenzamos a pensar en una mejora para que los usuarios tuvieran acceso más rápido a las monedas que ellos desearan y no solo las que estuvieran en el top 10. Entonces decidimos introducir la Cartera.

Mediante la Cartera el usuario podría guardar todas las monedas de su interés y así no tener que buscar en una larga lista de valores y símbolos. En un principio, para ver que nuestro sistema de inserción funcionaba, hicimos las operaciones manualmente en nuestra base de datos. Posteriormente desarrollamos la funcionalidad para que el usuario pudiese elegir la moneda que quería añadir a su cartera, en el momento en el que se solicita al bot añadir una moneda a la cartera, este envía un mensaje al usuario pidiéndole que le envíe el nombre corto de la moneda virtual que desea añadir. En este momento nos dimos cuenta de que aprovechando esta funcionalidad podíamos realizar una búsqueda de cualquier moneda para consultar sus datos sin necesidad de añadirla a la cartera y así lo hicimos.

Hasta ahora todos los comandos que recibía nuestro bot a excepción del inicial, era mediante la pulsación de alguno de los botones en el chat, sin embargo, tanto para la búsqueda como para añadir una moneda a la cartera necesitábamos recibir el símbolo de la moneda en forma de una cadena de caracteres que el usuario nos escribiese por el chat. Nos surgió la necesidad de saber en qué vista estaba el usuario para poder redirigirle correctamente a la siguiente vista. Entonces añadimos un objeto 'contexto' al usuario que nos indicaría si este está realizando una búsqueda o añadiendo una moneda para interpretar el texto enviado. Esto es necesario ya que Python-Telegram-Bot hace una distinción muy marcada entre los distintos tipos de mensajes (callbackQuery, inlineQuery, textMessage, command, etc.) Una vez hecho esto ya teníamos una implementación de la cartera y una búsqueda global totalmente funcionales, donde el usuario podría añadir cualquiera de las más de 5400 monedas soportadas por las APIs de CryptoCompare y CoinMarketCap.

Lo siguiente fue introducir en la información de las distintas monedas su variación porcentual para distintos periodos de tiempo: variación en la última hora, último día y última semana. Con estos datos el usuario podría hacerse una idea general de lo que está ocurriendo con sus criptomonedas. Siguiendo el hilo de la última mejora y para ampliar la experiencia del usuario, complementamos la vista de las monedas con gráficas: última hora, último día, última semana y último mes. Estas gráficas son generadas en el momento con los datos históricos que proporciona CryptoCompare, por lo que las gráficas están siempre actualizadas con los últimos datos.

Llegados a este punto ya teníamos una vista bastante completa de cada moneda, sin embargo nos dimos cuenta de la naturaleza internacional de estas y decidimos dar un paso adelante para hacer el bot más plural. Concluimos que sería apropiado extender la conversión de las criptomonedas a otra monedas soportadas, así que incluimos las monedas más usadas (dólar americano, euro, yen japonés, dólar australiano, libra británica, dólar Hong Kong). Para ello era necesario que el usuario dispusiera de un apartado de configuración dentro del bot para elegir la moneda destino, ya que hasta el momento solo se consultaban los valores de las monedas en dólares americanos.

Sabiendo que tendríamos que hacer un apartado de configuración se nos ocurrió hacer un sistema basado en diccionarios para traducir el bot a cualquier idioma. Dado que nosotros solo manejamos Inglés y Castellano, no hemos podido traducirlo a más lenguajes, pero el método para añadir nuevas traducciones es muy sencillo para que cualquier persona pueda hacer sus aportaciones.

Tras una reunión con el director del proyecto en la recta final, decidimos implementar algunas de las funcionalidades que surgieron. Una de ellas fue hacer que el bot fuera más proactivo y mandase algún tipo de informe sobre su wallet a los usuarios si estos lo deseaban. Este informe se puede obtener dentro de la vista de wallet o activándolo en la configuración para que se mande todos los días a una hora determinada. Para conseguir la periodicidad del informe usamos la utilidad Cron que se integra en la mayoría de distribuciones Linux y permite lanzar scripts periódicamente, y un script en Python que se encargase de crear otro hilo para mandar el informe a los usuarios suscritos.

Viendo que aún había tiempo decidimos darle una vuelta de tuerca a los gráficos y añadir uno más avanzado con indicadores de entrada y salida basados en el indicador 'Moving Average Convergence Divergence' (MACD). Aunque nos llevó un tiempo entender las matemáticas detrás de este indicador, conseguimos generar la gráfica avanzada con indicadores de compra/venta bastante acertados.

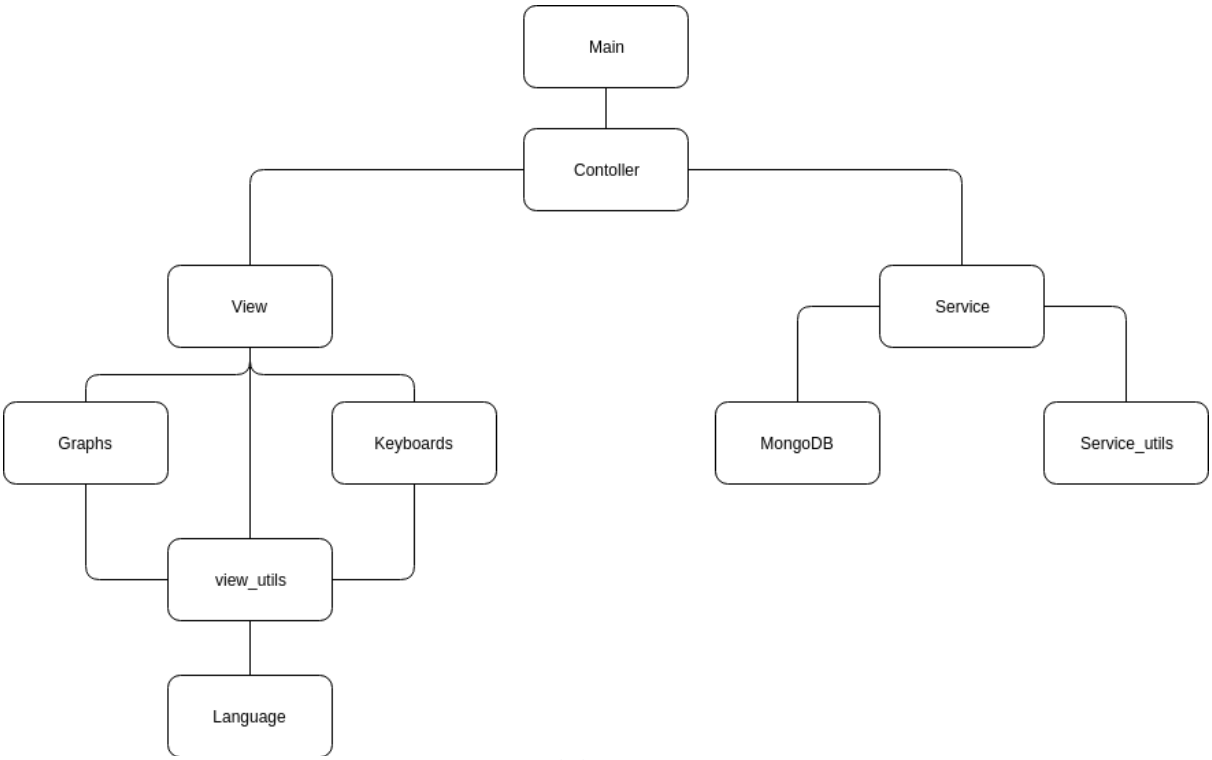
3.3. Esquema de proyecto y principales casos de uso

Para la organización del proyecto decidimos implementar el patrón MVC (Modelo-Vista-Controlador). En este modelo, hay tres clases principales:

- Controlador: Recibe los mensajes enviados por el usuario, obtiene los datos necesarios del modelo o servicios y pide la siguiente vista. En cierto modo actúa de nexo entre la vista y el modelo proporcionándole a cada uno los datos necesarios.

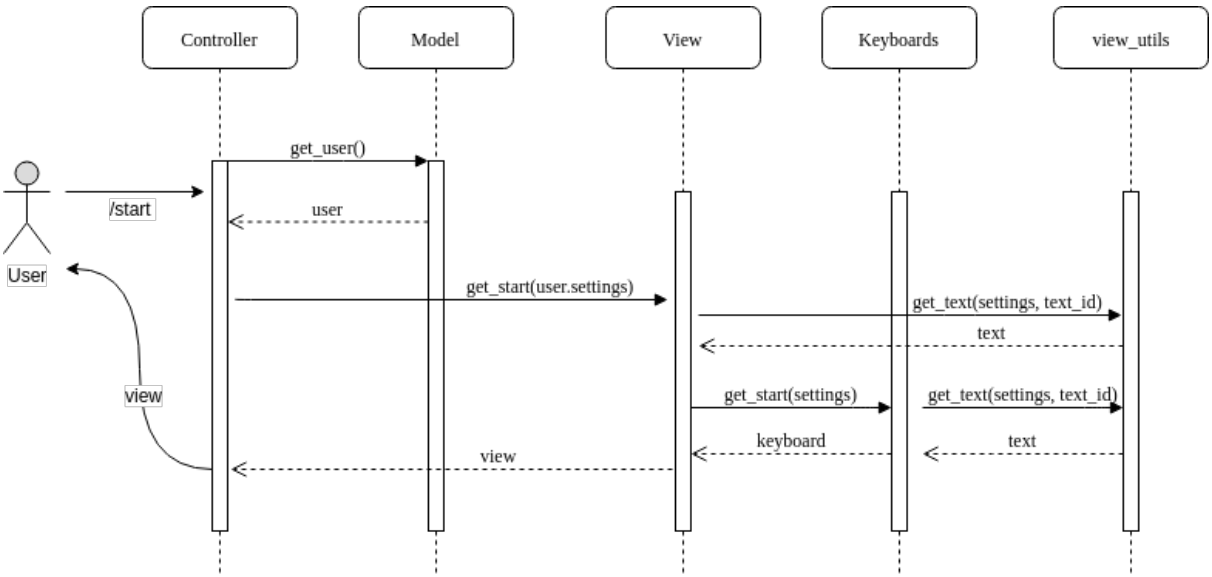
- Vista: Se encarga de construir las vistas, para ello utiliza los datos que el controlador le ofrece. El teclado o el gráfico solicitados utilizan la función get_text() del archivo view_utils.py que devuelve, en función de la configuración del usuario, el texto en el idioma solicitado, utilizando el diccionario languages.py

- Modelo: Se encarga de recoger los datos, tanto de la base de datos local, como de servicios externos como las APIs. También modifica los datos necesarios del contexto del usuario dependiendo de la vista en la que se encuentre.



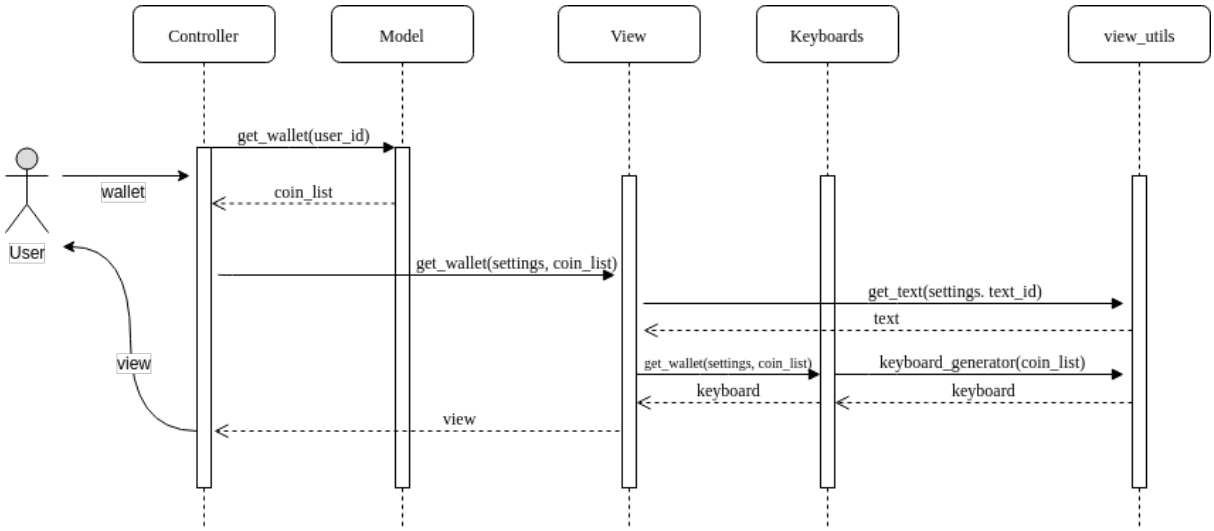
7 Arquitectura del proyecto CryptoPTB

El comando /start comprueba que el usuario está en la base de datos y a continuación pide la vista principal utilizando la configuración del usuario.



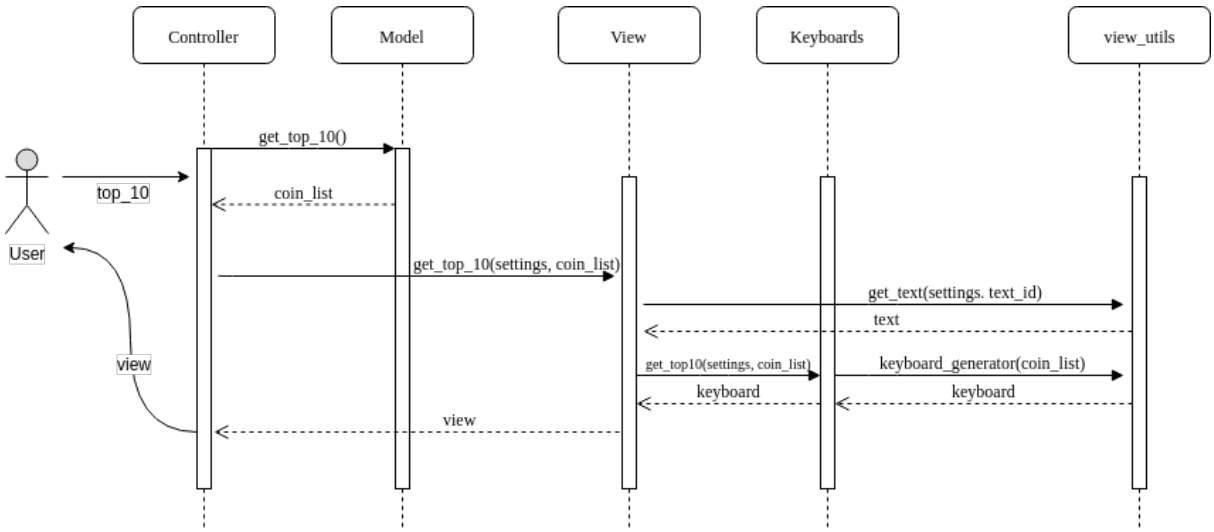
8 Diagrama de secuencia del comando /start

Cuando el usuario pulsa en su cartera se hace una petición a la base de datos que devuelve la lista de monedas en la cartera, esta lista de monedas es enviada a la vista que utiliza el generador de teclados para construir la nueva interfaz.



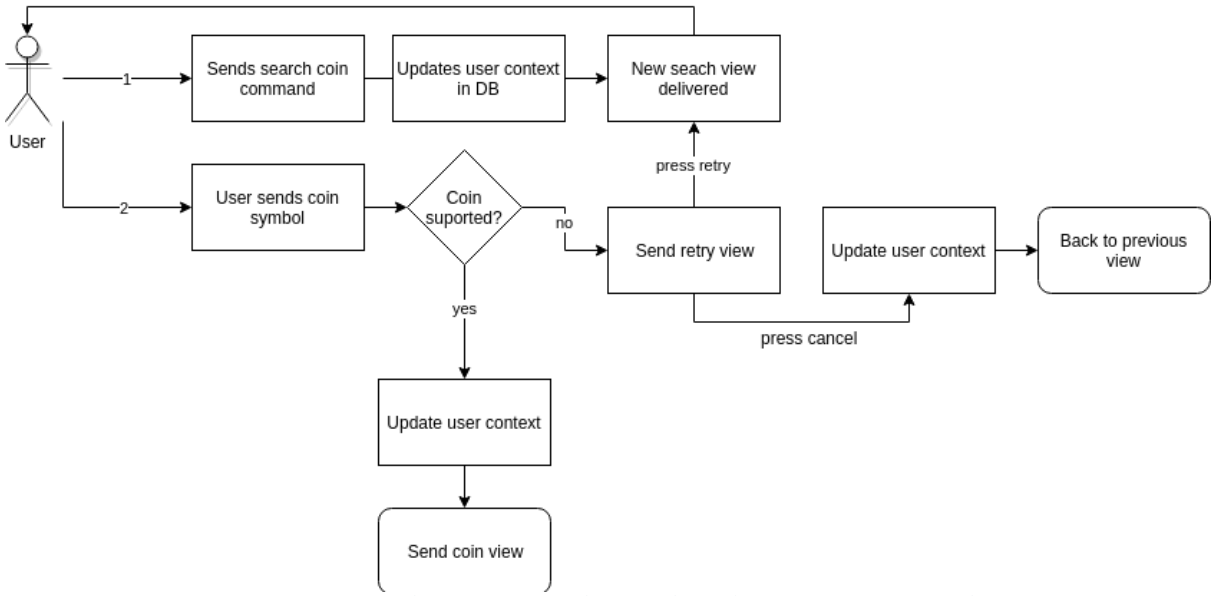
9 Diagrama de secuencia del Wallet

El comando Top 10 actúa de manera muy similar a la cartera solo que no necesita comprobar que el usuario existe, simplemente hace una petición al modelo para obtener el top 10 de monedas, por capitalización, en ese mismo instante.



10 Diagrama de secuencia del Top10

Cuando un usuario busca o añade una moneda se sigue un flujo algo distinto ya que se tiene que recibir el símbolo de la moneda a consultar a través de un mensaje en el chat. Además se tiene que actualizar el contexto del usuario para saber en cada momento si el usuario está realizando una búsqueda o no. Así quedaría el diagrama de flujo de este comando:

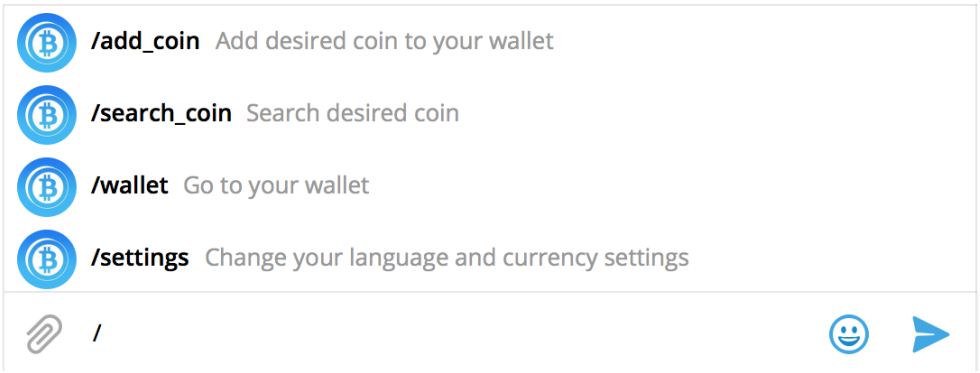


11 Diagrama de secuencia de añadir o buscar una moneda

3.4. Documentación funcional

- **Descripción del bot:** La descripción del bot debe incluir una explicación de lo que es capaz de hacer el bot y de cómo debe utilizarse. También debe mencionar las librerías utilizadas para no tener problemas con las licencias de éstas.

- **Comandos:**



12 Comandos de CryptoMarket Bot

  - **/start:** el bot debe enviar un nuevo mensaje con el menú principal en el que deben aparecer los botones: Wallet, Top 10, Search coin, Settings.

  - **/help:** el bot debe enviar un nuevo mensaje de ayuda en el que se debe explicar el funcionamiento del bot, los comandos disponibles...

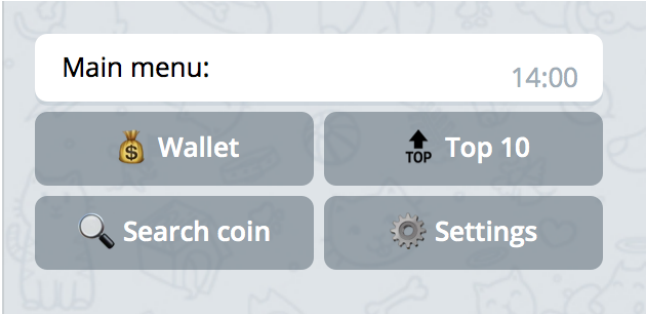
- **/wallet:** el bot debe enviar un nuevo mensaje en el que se muestre la vista “Wallet”. En ésta vista deben aparecer, en forma de botón, las criptomonedas que el usuario tenga añadidas a su cartera. También debe contener un botón para añadir una nueva criptomoneda a la cartera y un botón para volver al menú principal.

- **/settings:** el bot debe enviar un nuevo mensaje en el que se muestre la vista “Settings”. En ésta vista debe aparecer información con la configuración actual del usuario y dos botones, uno que permita cambiar el idioma y otro que permita cambiar la moneda de preferencia.

- **/search_coin:** el bot debe enviar un nuevo mensaje en el que se solicite al usuario que envíe el nombre corto de la criptomoneda que desea buscar. Este mensaje debe incluir un botón con el que se cancele la acción y posteriormente se vuelva al menú principal.

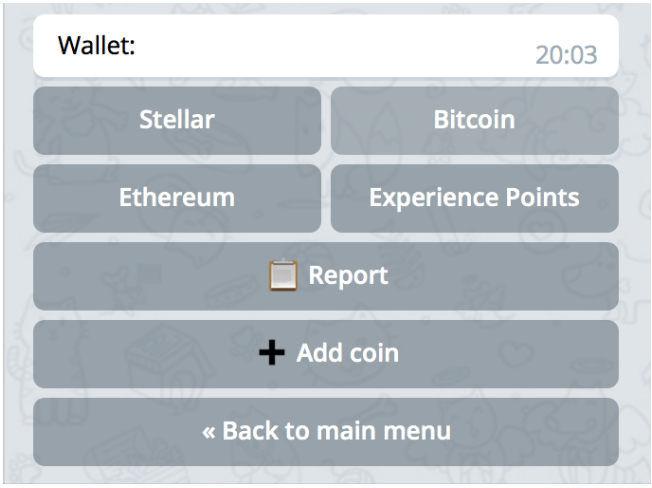
- **/add_coin:** el bot debe enviar un nuevo mensaje en el que se solicite al usuario que envíe el nombre corto de la criptomoneda que desea añadir a su cartera. Este mensaje debe incluir un botón con el que se cancele la acción y posteriormente se vuelva a la cartera del usuario.

- **Main menu:** el bot debe sustituir el mensaje actual. El nuevo mensaje debe disponer de al menos cuatro inline buttons: Wallet, Top 10, Search y Settings. Cada uno de estos botones, al ser pulsados, deben conducir a la vistas correspondientes a sus nombres descritas a continuación.



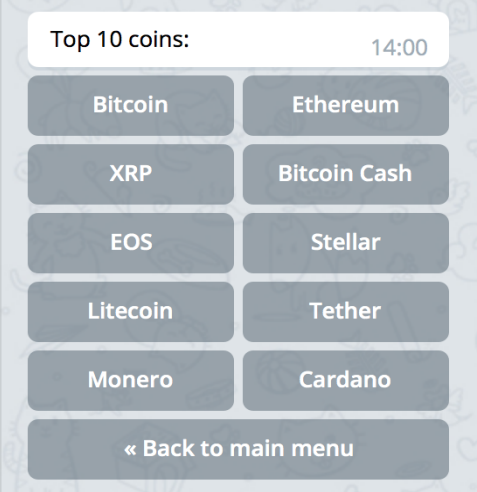
13 Vista del menú principal

- **Wallet:** el bot debe sustituir el mensaje actual. El nuevo mensaje debe mostrar un botón por cada criptomoneda que el usuario tenga añadida a su cartera. Cada uno de estos botones debe conducir a la vista "Detalle criptomoneda" descrita a continuación. También debe contener un botón que muestre un report de todo el wallet del usuario, otro que permita añadir una nueva moneda a la cartera y otro para volver al menú principal.



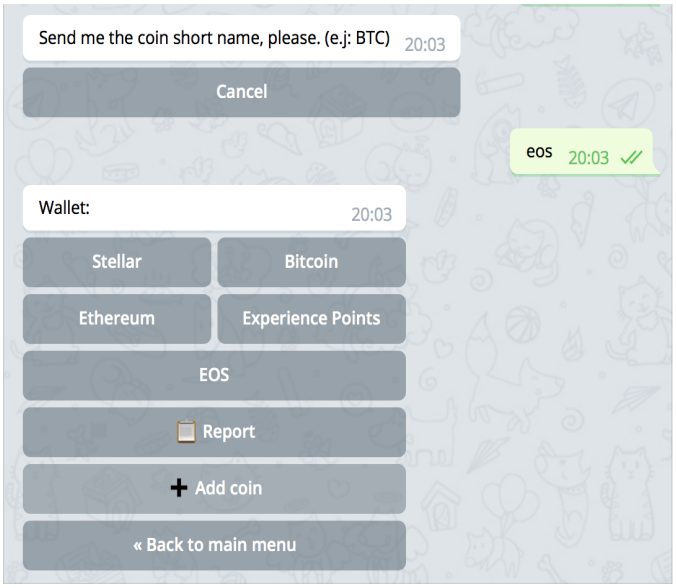
14 Vista del Wallet

- **Top 10:** el bot debe sustituir el mensaje actual. El nuevo mensaje debe mostrar un botón por cada una de las 10 monedas más relevantes del momento. Cada uno de estos botones debe conducir a la vista "Detalle criptomoneda" descrita a continuación. También debe disponer de un botón que conduzca a la vista del menú principal.



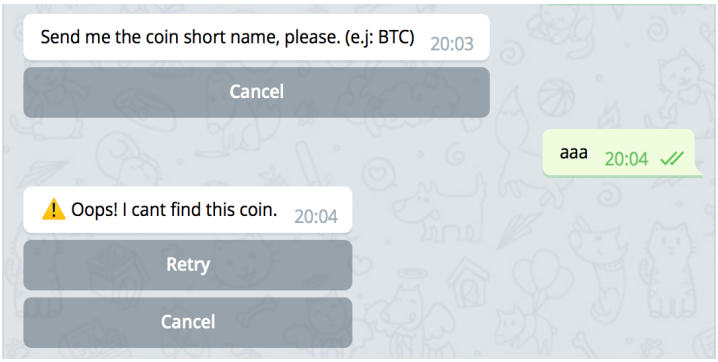
15 Vista del top 10

- **Search y Add coin:** el bot debe enviar un nuevo mensaje. Este mensaje debe mostrar un texto informando al usuario indicando que debe enviar el nombre corto de la criptomoneda que desea buscar o añadir. Una vez que el usuario introduzca un nombre corto correcto se debe mostrar la vista "Detalle de criptomoneda" en el caso de que se esté realizando una búsqueda, o "Wallet" en el caso de que se esté añadiendo una moneda, en caso contrario se debe mostrar la vista de "error al realizar una búsqueda o añadir criptomoneda" descrita a continuación.



16 Flujo y vistas de añadir o buscar criptomoneda

- **Error al realizar una búsqueda o añadir criptomoneda:** el bot debe sustituir el mensaje actual. Este debe mostrar un mensaje que informe al usuario de que la moneda introducida no es válida. Debe contener un botón que permita volver a la vista anterior (Search o Add coin, según corresponda) y otro botón que permita cancelar la operación. En el caso de cancelar la operación se debe volver al menú principal en caso de tratarse de una búsqueda o a la vista "Wallet" en caso de tratarse de añadir una moneda.

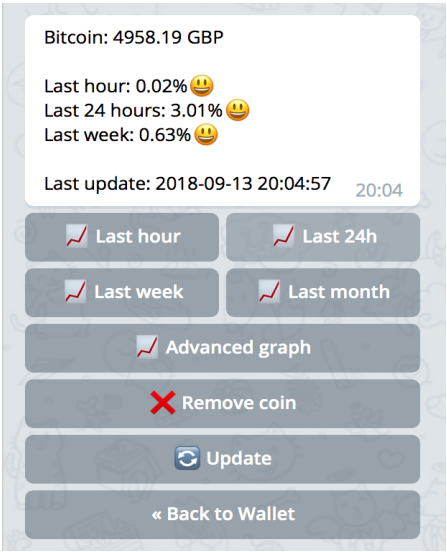


17 Flujo y vistas del error al realizar una búsqueda o añadir criptomoneda

- **Detalle criptomoneda:** el bot debe sustituir el mensaje actual. Este debe mostrar un mensaje que informe al usuario de lo siguiente:

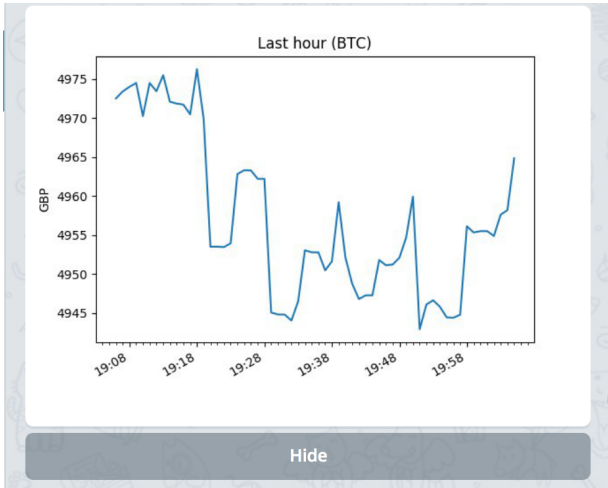
    - Nombre de la moneda.
    - Valor actual de la moneda en la divisa que el usuario tenga seleccionada en sus preferencias.
    - La variación porcentual del valor de la criptomoneda de la última hora, las últimas 24 horas y la última semana.
    - La última fecha en la que se ha actualizado la información en formato *Y-m-d H:M:S*

  También debe contener botones para mostrar las gráficas correspondientes a la evolución del valor de la criptomoneda en la última hora, las últimas 24 horas, la última semana y el último mes y otro botón para mostrar una gráfica avanzada de esta criptomoneda con indicadores de compra y venta. Además debe tener un botón para añadir o eliminar la criptomoneda a la cartera del usuario (dependiendo de si ya está añadida o no), otro botón que permita actualizar la información de ésta vista y otro que permita volver a la vista anterior.



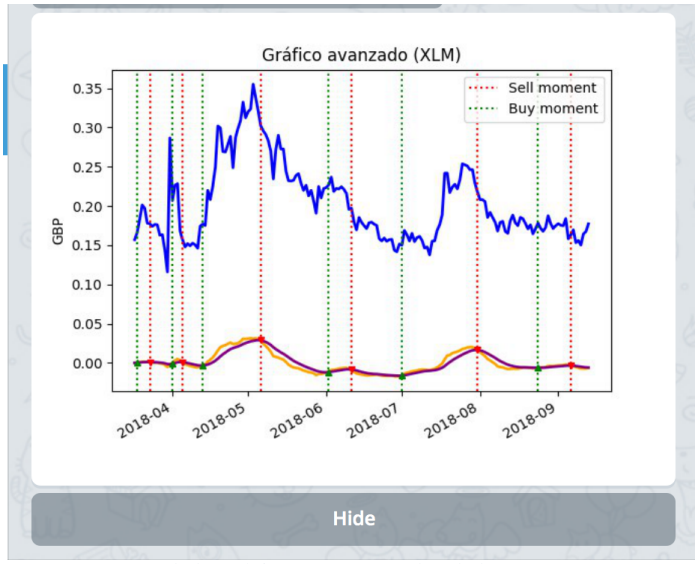
18 Vista del detalle de criptomoneda

- **Mostrar gráfica:** el bot debe enviar un nuevo mensaje. Este mensaje contendrá una imagen con la gráfica correspondiente. También dispondrá de un botón que permita ocultar este mensaje.



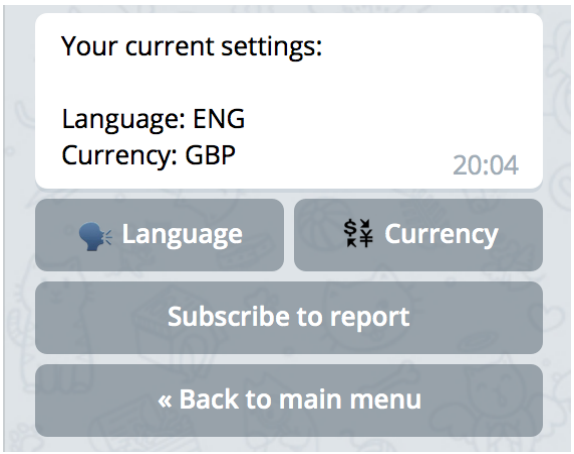
19 Vista del gráfico de la evolución de la última hora del BTC

- **Mostrar gráfica avanzada:** el bot enviará un nuevo mensaje. Este mensaje contendrá una imagen con la gráfica correspondiente en la que deben aparecer los indicadores de compra y venta descritos en el apartado 3.5. También dispondrá de un botón que permita ocultar este mensaje.



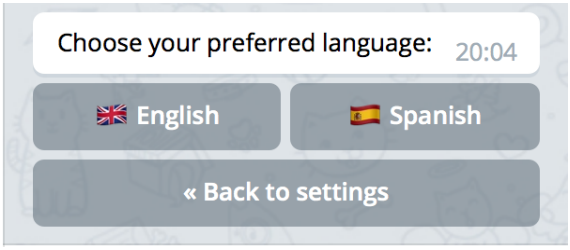
20 Vista del gráfico avanzado del XLM

- **Settings:** el bot debe sustituir el mensaje actual. Este mensaje debe informar de la configuración actual del usuario. También debe disponer de un botón que conduzca a la vista "Seleccionar idioma" descrita a continuación, otro que conduzca a la vista "Seleccionar moneda de preferencia" descrita a continuación, otro que permita suscribirse al report y otro que permita volver al menú principal.



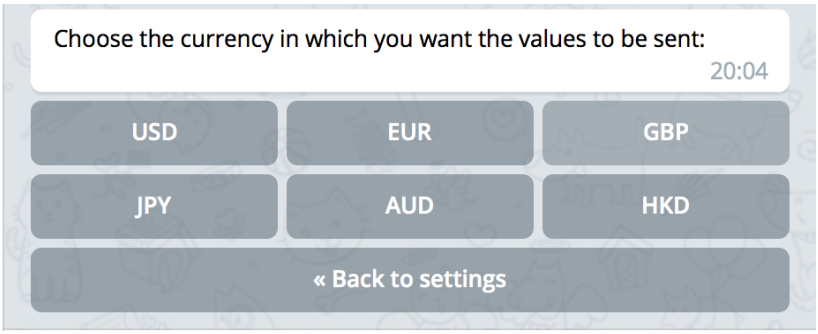
21 Vista de la configuración

- **Seleccionar idioma:** el bot debe sustituir el mensaje actual. Este mensaje debe contener un botón por cada idioma disponible. Cada uno de estos botones debe guardar la preferencia del usuario en base de datos. También debe contener un botón que permita volver a la vista "Settings".



22 Vista del selector de idiomas

- **Seleccionar moneda de preferencia:** el bot debe sustituir el mensaje actual. Este mensaje debe contener un botón por cada tipo de moneda disponible. Cada uno de estos botones debe guardar la preferencia del usuario en base de datos. También debe contener un botón que permita volver a la vista "Settings".



23 Vista del selector de divisa

3.5. Detalles de implementación

Como se ha mencionado anteriormente el bot se ha desarrollado con la ayuda de la librería de Python-Telegram-Bot. Para hacer funcionar un bot con esta librería es necesario crearse dos objetos principalmente, el objeto *Updater* y el objeto *Dispatcher*. El objeto *Updater* hay que instanciarlo pasándole por parámetro el token del bot y se encarga de recibir las actualizaciones que envía Telegram de ese bot y entregárselas al objeto *Dispatcher*. El objetivo del objeto *Dispatcher* es el de enviar todas las actualizaciones que recibe del objeto *Updater* y entregárselas a sus correspondientes *Handlers*. PTB dispone de una serie de handlers cuya función es gestionar los distintos tipos de eventos que recibe el bot. Para hacer funcionar estos handler es necesario añadir los que queramos gestionar al objeto *Dispatcher* de la librería de PTB. Nuestro bot intercepta los siguientes tipos de *Handlers*:

1. *CommandHandlers:* se encargan de gestionar los comandos del bot de Telegram. Los comandos de los bots de Telegram son mensajes que empiezan por el símbolo '/', seguido del nombre del comando. Los *CommandHandler* registrados en nuestro bot son: /start, /help, /wallet, /settings, /add_coin y /search_coin.

   Ejemplo de CommandHandler:

   ```
   start_handler = CommandHandler('start', controller.start)
   dispatcher.add_handler(start_handler)
   ```

2. *CallbackQueryHandler:* se encargan de gestionar los *callbacks* que envían los *InlineKeyboardButtons* que tenemos definidos en nuestro bot. Los *InlineKeyboardButtons* son botones que están asociados a un mensaje de Telegram. Cada *InlineKeyboardButton* dispone de un *callback* que se define a la hora de ser instanciado.

   Ejemplo de *CallbackQueryHandler*:

   ```
   button_handler = CallbackQueryHandler(controller.button)
   dispatcher.add_handler(button_handler)
   ```

   Ejemplo de *InlineKeyboardButton*:

   ```
   InlineKeyboardButton(text="🔝 Top 10", callback_data='top_10')
   ```

3. *MessageHandler:* se encarga de gestionar los mensajes de Telegram. Estos mensajes pueden tratarse de texto, multimedia o actualizaciones de estado. A la hora de declarar este handler hay que indicar el tipo de mensaje que se quiere gestionar.

   Ejemplo de *MessageHandler:*

   ```
   text_handler = MessageHandler(Filters.text, controller.text_messages)
   dispatcher.add_handler(text_handler)
   ```

```
updater = Updater(token=TOKEN)
dispatcher = updater.dispatcher
controller = Controller()

start_handler = CommandHandler('start', controller.start)
dispatcher.add_handler(start_handler)

help_handler = CommandHandler('help', controller.help)
dispatcher.add_handler(help_handler)

add_coin_handler = CommandHandler('add_coin', controller.add_coin)
dispatcher.add_handler(add_coin_handler)

settings_handler = CommandHandler('settings', controller.settings)
dispatcher.add_handler(settings_handler)

search_coin_handler = CommandHandler('search_coin', controller.search_coin)
dispatcher.add_handler(search_coin_handler)

wallet_handler = CommandHandler('wallet', controller.wallet)
dispatcher.add_handler(wallet_handler)

button_handler = CallbackQueryHandler(controller.button)
dispatcher.add_handler(button_handler)

text_handler = MessageHandler(Filters.text, controller.text_messages)
dispatcher.add_handler(text_handler)

updater.start_polling()
```

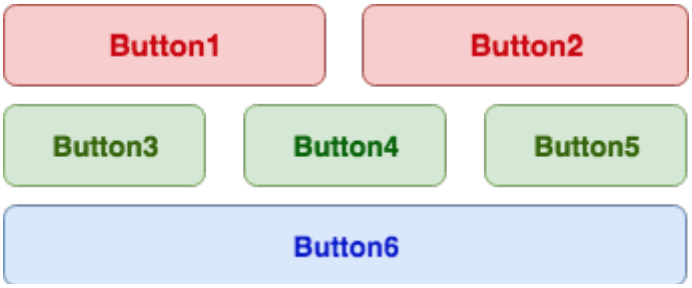
24 main.py

La principal manera de navegar por el bot es mediante los anteriormente explicados *InlineKeyboardButtons* ya que nos parece que es más atractiva para el usuario a la vez que más cómoda y limpia. Para garantizar un flujo completo de navegación por los distintos menús decidimos implementar un sistema de botones que nos permitiesen volver atrás cuando mostramos diferentes vistas. Cada uno de estos botones dispone de un callback diferente que recoge el objeto *Dispatcher* y posteriormente lo gestiona con el *CallbackQueryHandler*.

En ocasiones nos vemos en la necesidad de generar teclados con *InlineKeyboardButtons* de forma dinámica por lo que se ha implementado una función llamada *keyboard_generator()*. Esta función recibe los datos necesarios para generar los botones que se solicitan y devuelve un array que a su vez en cada posición posee otro array de *InlineKeyboardButtons*, ya que Telegram para crear la vista toma cada uno de estos arrays internos como una fila del *InlineKeyboardMarkup* y cada una de estas filas tendrá un número de columnas igual a la longitud de cada array interno. Esto es, si por ejemplo tenemos la siguiente estructura:

[[Button1, Button2], [Button3, Button4, Button5], [Button6]]

Telegram creará una vista de la siguiente forma:



25 Ejemplo de generación de teclado

Para la obtención de los datos que nos ofrecen las APIs de CryptoCompare y CoinMarketCapt utilizamos la librería *Request* de Python. Todas las llamadas a estas APIs son de tipo GET y no requieren ningún tipo de autenticación, al menos en el momento de escribir estas líneas, pero en el caso de pedir autenticación por medio de una API key ya sabremos cómo implementarlo gracias a la experiencia anterior con el bot de estadísticas de Clash Royale.

Las APIs nos devuelven los datos en forma de objetos JSON, con los cuales tenemos experiencia previa, lo que ha hecho que manejar los datos recibidos no suponga de un aprendizaje durante el desarrollo del TFG. Para hacer más ágil el manejo de las urls hemos implementado un método que hemos denominado *url_generator()*. Este método recibe por parámetro el tipo de url que se solicita y los datos necesarios para completar dicha url de forma dinámica.


```
{
   Response: "Success",
   Type: 100,
   Aggregated: false,
 - Data: [
     - {
           time: 1536796800,
           close: 6492,
           high: 6534.89,
           low: 6337.08,
           open: 6337.08,
           volumefrom: 65684.53,
           volumeto: 425090869.76
       },
     - {
           time: 1536883200,
           close: 6477.63,
           high: 6591.59,
           low: 6393.84,
           open: 6492.17,
           volumefrom: 40217.23,
           volumeto: 262042900.16
       }
   ],
   TimeTo: 1536883200,
   TimeFrom: 1536796800,
   FirstValueInArray: true,
 - ConversionType: {
       type: "direct",
       conversionSymbol: ""
   }
}
```

```
{
 - data: {
     - 1: {
           id: 1,
           name: "Bitcoin",
           symbol: "BTC",
           website_slug: "bitcoin",
           rank: 1,
           circulating_supply: 17267512,
           total_supply: 17267512,
           max_supply: 21000000,
         - quotes: {
             - USD: {
                   price: 6483.49283015,
                   volume_24h: 4080195739.3539,
                   market_cap: 111953790247,
                   percent_change_1h: 0.25,
                   percent_change_24h: -0.36,
                   percent_change_7d: 0.3
               }
           },
           last_updated: 1536940045
       }
   },
 - metadata: {
       timestamp: 1536939540,
       num_cryptocurrencies: 1950,
       error: null
   }
}
```


26 Ejemplo de respuesta de CryptoCompare (izquierda) y CoinMarketCap (derecha)

Como ya se ha mencionado anteriormente nuestra base de datos es no relacional, en concreto es una base de datos MongoDB. MongoDB ha supuesto un gran descubrimiento ya que ha conseguido agilizar en gran medida el desarrollo del bot. La principal razón es que, cuando hemos decidido cambiar la forma de guardar los datos no hemos tenido que rehacer dicha base de datos, esto es debido a que

MongoDB guarda los datos con una estructura JSON, lo que hace que las estructuras de datos no sean rígidas y sean fácilmente modificables. MongoDB almacena los datos en lo que denomina colecciones, lo que sería el equivalente a una tabla en SQL. A su vez cada colección organiza sus datos en documentos, cada documento almacenará un objeto JSON con los datos que nosotros deseemos. Añadir un campo más a cualquier documento de una colección se convierte en una tarea muy sencilla en MongoDB, ya que solo sería necesario invocar a la función update y especificarle el nombre del nuevo campo con los datos asociados a este, lo que incluso permite que en una misma colección haya documentos con diferente estructura. Para operar con MongoDB en nuestro proyecto hemos utilizado la librería PyMongo, ésta librería nos proporciona una interfaz para poder gestionar MongoDB en Python. Una de las funciones que nos ha resultado más útil de PyMongo es *update()* junto con el parámetro 'upsert'. Este parámetro permite que, si al intentar actualizar un documento, este no existe, se crea con los datos que se facilitan, por lo que hace que no tengamos que comprobar si el documento ya existe y actualizarlo o, en el caso de que no exista, insertarlo.

Nuestra base de datos actualmente se compone de dos colecciones:

1. Colección 'coins': en esta colección almacenamos toda la lista de criptomonedas disponible en la API de CryptoCompare. Cada documento tiene una estructura del tipo:

```
{
  "_id" : 1,
  "name" : "Bitcoin",
  "symbol" : "BTC"
}
```

2. Colección 'users': en esta colección almacenamos los usuarios que utilizan nuestro bot. La información que almacenamos en cada documento es la siguiente:

```
{
  "_id" : 123456,
  "wallet" : [],
  "context" : {
    "add_coin" : false,
    "search_coin" : false
  },
  "settings" : {
    "language" : "ENG",
    "currency" : "USD"
  }
}
```

3. Colección 'report': en esta colección almacenamos los ids de los usuarios que tienen activado el report automático:

```
{ "_id" : 123456}
```

A la hora de generar las gráficas se ha utilizado la librería Matplotlib. En primer lugar, para probar ésta librería, construimos un nuevo proyecto para realizar pruebas sin riesgo de originar bugs por error en el proyecto original. Cuando habíamos realizado las pruebas más sencillas comenzamos a incluir datos reales que obteníamos de las APIs de criptomonedas hasta que obtuvimos los resultados deseados. Una vez llegados a este punto introducimos éste código en nuestro proyecto con las modificaciones pertinentes para adaptarlo al código ya existente. Para nuestro proyecto hemos implementado gráficas para mostrar la evolución en distintos periodos de tiempo de las criptomonedas. Los datos los obtenemos de la librería CryptoCompare, que nos facilita lo datos históricos de más de 5.400 criptomonedas en distintos periodos de tiempo: por minutos, por horas y por días. Por nuestra parte decidimos mostrar gráficas sobre la última hora, las últimas 24 horas, la última semana y el último mes.

- Gráfica última hora: representa la evolución de los últimos 60 minutos de la criptomoneda correspondiente y con la divisa que el usuario ha elegido en sus preferencias.

- Gráfica últimas 24 horas: representa la evolución de los 1440 últimos minutos, ya que son los minutos que hay en 24 horas. Se decidió hacer así ya nos permite mostrar una gráfica más precisa.

- Gráfica última semana: representa la evolución de las 168 últimas horas, que son las correspondientes a las horas que hay en los últimos 7 días. Al igual que en el caso de la gráfica de las últimas 24 horas esto se hizo para poder representar una gráfica más precisa.

- Gráfica del último mes: representa la evolución de las últimas 720 horas, que son las correspondientes a los últimos 30 días. Al igual que en el caso de la gráfica de las últimas 24 horas y de la última semana esto se hizo para poder representar una gráfica más precisa.

Al intentar representar gran cantidad de datos en un espacio tan pequeño, Matplotlib solapaba la leyenda de los ejes X e Y, por lo que tuvimos que indicar a ésta librería distintos rangos en los que queríamos que indicase los valores. Una vez generadas las gráficas las guardamos en formato .png en una carpeta llamada 'graphs' en el servidor. Para evitar problemas de memoria cuando se han solicitado gran cantidad de gráficas estas imágenes son eliminadas una vez que se han enviado al usuario que las solicita. Para garantizar que no se eliminan las gráficas equivocadas nombramos a los archivos con el formato <userId>.png.

Para hacer más personal y a la vez pueda ser usado sin problemas por mucha más gente incluimos la posibilidad de que el usuario elija, tanto el idioma del bot (por el momento solo están disponibles inglés y español), como la moneda en la que quiere recibir el valor actual de la criptomoneda (dólar americano, euro, yen japonés, dólar Australiano, libra Británica, dólar Hong Kong). Estas preferencias son guardadas en nuestra base de datos y por defecto se inicia con el bot en inglés y la moneda en dólar americano.

Para realizar la implementación multi idioma lo más escalable y óptimo posible se ha realizado mediante diccionarios. La búsqueda en un diccionario es constante O(1) en

la mayoría de los casos, llegando a ser como máximo lineal en el número de elementos O(n) en el peor de los casos, por lo que los tiempos de acceso en nuestro diccionario son despreciables. La estructura de estos diccionarios correspondería con la siguiente:

```
HashMap = {
        key (type String) : value (type HashMap) = {
                key (type String) : value (type String)
        }
}
```

El primer String de esta estructura de datos corresponde con la denominación corta del idioma en cuestión siguiendo el estándar ISO 639-2. Los siguientes dos Strings corresponden con el identificador (clave del HashMap) que hemos asignado a cada texto (valor del HashMap).

En la generación de gráficas avanzadas es necesario traer una gran cantidad de datos para su posterior procesamiento. Para esto Pandas, es la librería ideal ya que introduce una nueva estructura de datos y funciones para su manipulación, el dataframe.

	datetime	low	high	open	close	volumefrom	volumeto
0	2018-03-17	6292.52	6839.39	6732.55	6404.29	17998.32	1.176954e+08
1	2018-03-18	5954.41	6734.97	6404.06	6681.96	30695.31	1.921577e+08
2	2018-03-19	6606.93	7100.32	6681.96	6972.23	27049.14	1.847861e+08
3	2018-03-20	6750.44	7414.75	6972.29	7290.64	22080.96	1.563475e+08
4	2018-03-21	7150.73	7481.52	7289.26	7223.15	17913.16	1.311014e+08
5	2018-03-22	6903.35	7361.65	7221.90	7080.46	16788.16	1.188445e+08

27 Dataframe antes de StockStats

Pasándole estos datos a la librería StockStats, ésta genera otras columnas aplicando las fórmulas necesarias para conseguir la implementación del indicador MACD.

	datetime	low	high	open	close	...	close_12_ema	close_26_ema	macd	macds	macdh
0	2018-03-17	6292.52	6839.39	6732.55	6404.29	...	6404.290000	6404.290000	0.000000	0.000000	0.000000
1	2018-03-18	5954.41	6734.97	6404.06	6681.96	...	6554.694583	6548.464808	6.229776	3.460986	5.537578
2	2018-03-19	6606.93	7100.32	6681.96	6972.23	...	6717.658753	6700.719527	16.939226	8.984855	15.908742
3	2018-03-20	6750.44	7414.75	6972.29	7290.64	...	6898.526540	6865.635512	32.891028	17.083152	31.615752
4	2018-03-21	7150.73	7481.52	7289.26	7223.15	...	6986.725574	6948.544596	38.180978	23.359278	29.643399
5	2018-03-22	6903.35	7361.65	7221.90	7080.46	...	7009.507971	6974.966200	34.541771	26.390356	16.302831
6	2018-03-23	6730.67	7230.27	7078.41	7226.89	...	7058.016010	7019.769541	38.246469	29.390822	17.711294

28 Dataframe después de StockStats

Con los datos de MACD y MACDS ya podemos obtener las señales de compra y venta, pues cuando estos dos valores se cruzan marcan una señal de salida o de entrada, indicando que el valor en cuestión están en una tendencia alcista o bajista.

Para encontrar los puntos de corte solo tuvimos que recorrer el dataframe y comparar con el registro anterior. Si los valores MACD y MACDS se cruzaban, guardabamos el registro en su array correspondiente:

Buy_signal = MACDS <= MACD & Previous_short >= Previous_long
Sell_signal = MACDS >= MACD & Previous_short <= Previous_long

Para la implementación del informe empezamos creando un botón en el menú de la cartera para que recogiese los datos más relevantes de las últimas 24 horas de las monedas que el usuario había añadido previamente y los mostrase en una vista compacta. Los datos que se muestran para cada moneda son:

- Símbolo
- Variación en porcentaje
- Valor actual
- Valor mínimo alcanzado
- Valor máximo alcanzado

Al final del informe se hace una suma de la variación en porcentaje de todas la monedas y se muestra un mensaje diciendo si es positiva o negativa.

Una vez creada esta función decidimos hacer el bot más proactivo añadiendo la posibilidad de mandar este informe una vez al día. Para ello modificamos la vista de configuración y añadimos un botón de activar/desactivar informe. Al pulsar este botón se añade el Id del usuario a una nueva colección en nuestra base de datos llamada reports. De la misma manera al volver a pulsar el botón se desactiva la suscripción borrando el Id del usuario de la colección 'report. Todos los usuarios dentro de esta colección reciben el informe de su cartera una vez al día a las 12:00.

El desarrollo de esta función necesitaba la creación de un script independiente, que se generase como un proceso distinto y que pudiera ser llamado por una tarea Cron de Linux. Creamos el archivo report_cron.py que contiene una instancia de la clase MongoDB y otra de Views. Su única función es mandar a todos los usuarios dentro de la colección report el informe de sus monedas.

La tarea Cron debe ser instalada en el servidor en el que se aloje el proyecto. Para cambiar la periodicidad en la que se manda el report, basta con cambiar los parámetros de la tarea Cron.

Ejemplos:

- * * * * * *python3 report_cron.py:* Ejecutará el script de python cada minuto.
- * 12 * * * python3 report_cron.py: Ejecutará el script todos los días a las 12:00

3.6. Problemas encontrados durante el desarrollo y soluciones

El primer problema que nos encontramos a la hora de enfrentarnos a este proyecto fue adaptarnos al lenguaje de programación Python. Nunca habíamos usado este lenguaje de programación y sobre todo su sintaxis es bastante diferente a otros lenguajes que habíamos usado, como Java o C++. Una de las cosas a las que cuesta acostumbrarse es a su forma de organizar el código, ya que Python lo interpreta mediante tabulaciones de las líneas, sin embargo los lenguajes de programación que habíamos usado delimitan las funciones, bucles y condiciones mediante paréntesis y llaves. Esto es totalmente subsanado en el momento en el que se usa un IDE o editor de texto específico para este lenguaje.

Al igual que con Python, nos ocurrió con MongoDB. A ambos nos llamaban la atención las bases de datos no relacionales, pero ninguno de nosotros había tenido la oportunidad de trabajar con ellas. Lo primero a lo que nos tuvimos que enfrentar fue entender la forma que tiene, en este caso, MongoDB, de organizar las colecciones y los documentos. En el momento en el que entendimos la forma de organizarse y de funcionar de MongoDB decidimos utilizarlo sin pensarlo ya que consideramos que iba a cubrir todas nuestras necesidades y queríamos aprender a utilizarlo. Instalar MongoDB fue muy rápido, pero dotar a nuestras bases de datos de seguridad no fue tanto, ya que, aun siguiendo un tutorial, tuvimos problemas para configurarlo en uno de nuestros PCs personales debido a que en uno de los pasos de la configuración no se debió realizar correctamente y los siguientes pasos no producían los resultados esperados. Esto fue solucionado reinstalando MongoDB en esta máquina y volviendo a configurar todo desde cero. Una vez teníamos configurada correctamente la base de datos de nuestro bot comenzamos a realizar pruebas pero tuvimos la suerte de que la curva de aprendizaje fue rápida y conseguimos estar trabajando con ella en poco tiempo.

Otra de las principales dificultades fue entender el funcionamiento de los bots de Telegram y a su vez de la librería que hemos utilizado para trabajar con ellos. Interiorizar que todos los usuarios que están utilizando el bot y que todas sus acciones son gestionadas por la misma instancia del código cambia los esquemas a los que estamos acostumbrados, ya que hasta ahora las aplicaciones que habíamos desarrollado se compilaban y se instalaban en un dispositivo y cada una de estas instalaciones era la encargada de gestionar las acciones del usuario que usaba el dispositivo en cuestión por separado. Sin embargo en estos bots una única instancia del código se encuentra corriendo en un servidor y es la encargada de gestionar las peticiones de todos los usuarios que usan dicho bot. Esto influye en la forma de pensar los algoritmos y también en la forma de idear la arquitectura del proyecto.Por otro lado entender cómo la librería de Python-Telegram-Bot gestiona todo esto no fue tan fácil como en un principio pensábamos. Una vez que se había entendido cómo funcionaban los objetos *Updater, Dispatcher* y sus correspondientes *Handlers* se facilita en gran medida el trabajo que hay que realizar.

Durante el desarrollo del bot nos dimos cuenta de que, el botón de 'Update' que se encarga de la actualización de la información en el detalle de la criptomoneda, a veces se quedaba cargando durante un largo periodo de tiempo sin que la librería informase de ningún fallo. Esta funcionalidad utiliza un método de la librería que sirve para editar el texto de un mensaje, después de realizar gran cantidad de pruebas nos dimos cuenta de que la librería, cuando el nuevo texto que queríamos visualizar en el mensaje era exactamente el mismo que el que ya había se quedaba sin bloqueada durante un tiempo. Para solucionar esto simplemente comprobábamos si el nuevo texto del mensaje era distinto al que había actualmente para solicitar la edición del mismo.

Para implementar la integración de distintos idiomas en nuestro bot, empezamos con un sistema basado en condicionales. Al ver que el código era cada vez mayor y menos legible empezamos a pensar en un sistema más escalable y modular que facilitara las aportaciones futuras. Tras varias reuniones e ideas, llegamos a la conclusión de que la mejor manera de conseguir nuestro propósito era utilizando diccionarios. Utilizando un diccionario con las traducciones y una función auxiliar que asociase un identificador con un texto, los cambios para introducir nuevos lenguajes se convertía

en un proceso mucho más sencillo. Introducimos un nuevo archivo llamado 'languages.py' que contiene un diccionario con una estructura sencilla que se compone de una clave de idioma y un diccionario anidado que asocia los identificadores con su texto correspondiente.

Este problema y la resolución del mismo nos lleva directamente a una de las dificultades que más hemos experimentado en el proceso: La toma de decisiones y el impacto de sus implementaciones a largo plazo. Al ser este uno de nuestros primeros proyectos y dado que no tenemos mucha experiencia, nos hemos encontrado muchas veces repitiendo y reorganizando nuestro código para hacerlo más legible, modular y escalable. Para evitar esto nos pusimos como obligación hacer reuniones periódicas para, entre los dos, acordar la mejor manera de afrontar la implementación de nuevas funcionalidades. En estas reuniones establecíamos plazos para terminar objetivos, hablábamos de soluciones para mejorar el trabajo ya existente y nos repartíamos las tareas en función de la disponibilidad de cada uno.

Uno de los temas que salía a menudo en estas reuniones eran las licencias de las distintas APIs y librerías. Debido a que nuestro proyecto es un proyecto llevado a cabo sin ánimo de lucro y que las distintas herramientas que usamos son públicas y de software libre no encontramos problemas a la hora de usarlas libremente, sin embargo, sí que tuvimos que comprobar bajo qué condiciones se podían utilizar y las condiciones de uso.

Hablando de software libre llegamos a una de las herramientas que más contribuye a que estos proyectos puedan llevarse a cabo de forma colaborativa. Git nos ha ayudado mucho, aunque también nos a traido algún que otro quebradero de cabeza. Nuestra falta de experiencia y el desconocimiento hizo que la curva de aprendizaje fuera algo más pesada. Aun así pusimos nuestro esfuerzo en organizar el trabajo desarrollado utilizando este gestor de versiones tan completo. Además, no solo hemos aprendido a utilizar Git sino que también nos hemos familiarizado con Github que a día de hoy es el servicio de repositorios, públicos y privados, más usado del mundo.

Como ya hemos mencionado antes en este documento en la apartado de evolución, nos costó un tiempo entender la matemática que hay detrás de los indicadores usados en los mercados de valores. Decidimos usar el indicador MACD porque es uno de los más populares y es sencillo de implementar. Una vez lo entendimos, surgió el problema de cómo plasmar esa información en un gráfico generado con python. Para ello usamos las librerías 'pandas' y 'StockStats' que nos facilitaron el trabajo. Mientras que la librería Pandas se encargaba de agrupar los datos históricos en una imagen de datos fácilmente tratable, StockStats se encargaba de aplicar las fórmulas necesarias para generar los datos del indicador. Con todos los datos formateados solo teníamos que aplicar las mismas técnicas para mostrar las gráficas que ya habíamos utilizado.

Para terminar la sección queremos remarcar otro conflicto con el que tuvimos que lidiar, organizar y compaginar nuestro tiempo. Por motivos de estudio y trabajo, hemos tenido que hacer una planificación del proyecto teniendo en cuenta las condiciones de cada uno. Con las reuniones periódicas logramos repartirnos el tiempo para poder llevar a cabo la mayor parte de las ideas iniciales y plasmarlas en el bot.

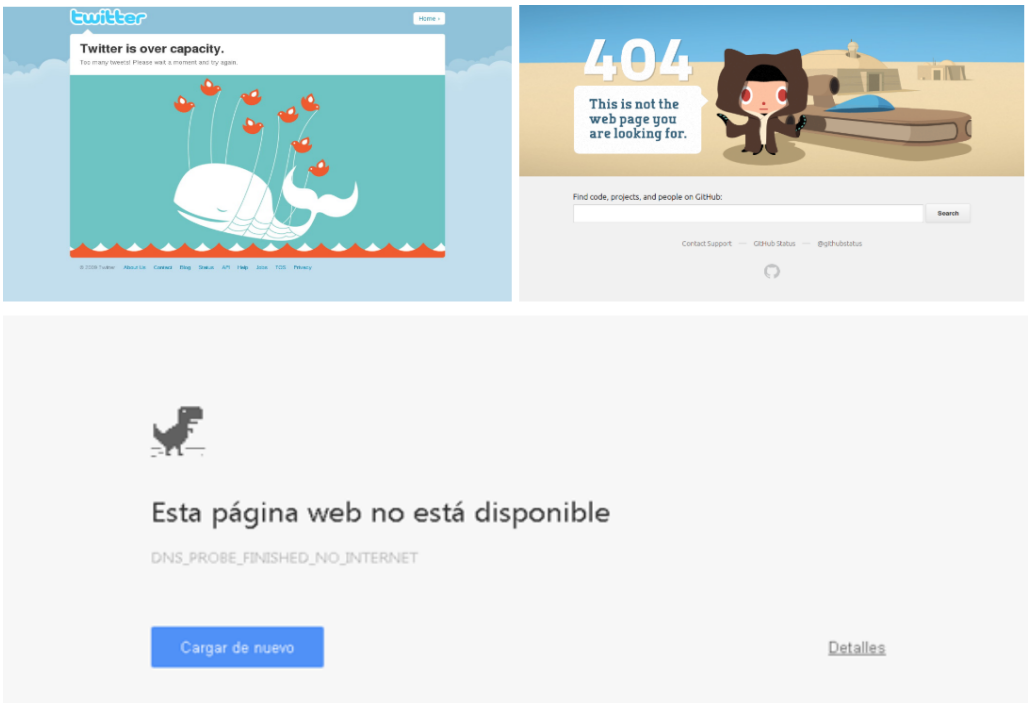
4. Implementaciones y mejoras futuras

Este TFG fue escogido porque a ambos nos llevaba tiempo llamando la atención el mundo de los bots y éramos conscientes de que Telegram ofrecía una API para desarrolladores. Unido a que los dos somos seguidores de las criptomonedas decidimos desarrollar este bot. Por estas razones nuestra intención es seguir mejorando el bot y ampliando su funcionalidad. Por el momento, las mejoras y funcionalidades que tenemos pensadas para nuestro bot son las siguientes:

- **Opción de añadir cantidad de cada criptomoneda en Wallet:** Normalmente, cuando un usuario añade una criptomoneda a su Wallet es porque posee cierta cantidad de ella y desea saber qué beneficio (o pérdida) obtendría en el caso de vender dicha cantidad, por supuesto, éste valor sería dado en la divisa que el usuario tenga seleccionada en sus preferencias. Por ejemplo, si un usuario posee 0,16 BTC y el esta criptomoneda en ese momento tiene un valor de 7.000€, dicho usuario posee 1.120€ (en el caso de que la divisa seleccionada en sus preferencias fuese el Euro).

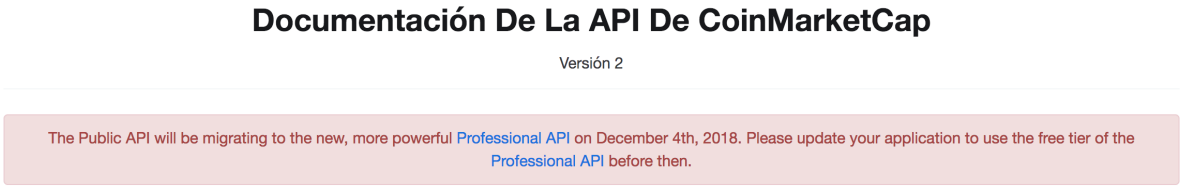
- **Lenguaje más natural:** En la actualidad se presta mucha atención a la experiencia de usuario ya que puede ser una característica diferencial a la hora de que un usuario utilice un software por tiempo prolongado. Si cogemos como ejemplos a dos grandes y famosos software, Siri y Google Assistant (aunque no sean exactamente bots como el nuestro), se nota desde el primer momento el cariño que, tanto Apple como Google, están depositando en la experiencia de usuario, ya que estos dos asistentes intentan tener una conversación lo más amistosa posible y ganarse al usuario, incluso son capaces de contar chistes o hacer guiños a series o películas famosas. Por esto queremos hacer que nuestro bot ofrezca la mejor experiencia de usuario posible.

- **Mejoras en la gestión de errores:** Esta funcionalidad en cierta manera va ligada a la anterior, ya que un software que presenta errores puede terminar frustrando al usuario y provocando que éste deje de usarlo. Es posible que haya errores que se descubran a medida que la popularidad del bot vaya aumentando por lo que habría que ir corrigiéndolos para hacer que nuestro bot sea lo más estable posible. Incluso se podría pensar una forma amigable de gestionar los errores, como ya se ha mencionado anteriormente, la experiencia de usuario es muy importante y en el ámbito de los errores también se puede aplicar, empresas como GiitHub, Twitter o la propia Google ofrecen mensajes de error personalizados que intentan que la experiencia de usuario no sea tan negativa. Ejemplos:



29 Ejemplos de errores personalizados

- **Mantenimiento, adaptación del bot a las actualizaciones de las librerías:** como ya nos ocurrió con la API que utilizábamos en el bot de estadísticas de Clash Royale, puede suceder que cualquiera de las APIs que actualmente utilizamos para obtener los datos sea actualizada y cambie la forma de usarse. Actualmente ambos estamos unidos a dos grupos de Telegram que posee la API de Python-Telegram-Bot, por lo que cuando la API recibe una actualización nos enteramos al momento. También, en el momento de escribir estas líneas, al revisar las web de las APIs para consultar información nos hemos dado cuenta de que una de ellas, CoinMarketCap, va a migrar la API el día 4 de Diciembre, por lo que antes de que esto ocurra debemos actualizar nuestro bot para que sea totalmente compatible y no dejen de responder algunas de sus funciones.



30 Anuncio de cambios en la API de CoinMarketCap

- **Generar una gráfica con todas las monedas que un usuario tiene añadidas en Wallet:** en una de las reuniones que tuvimos con nuestro director de TFG, Carlos Gregorio, surgió la idea de aprovechar que nuestro bot ofrecía gráficas con información de criptomonedas para ofrecer una gráfica en la que se represente la evolución de todas las monedas virtuales que un usuario tiene añadidas a su wallet. Esta funcionalidad serviría para que el usuario pueda comparar cuáles de sus criptomonedas tienen mejor evolución y así poder tomar decisiones acorde a esta información.

- **Alertas de tus monedas:** Con esta funcionalidad queremos dar al usuario la opción de marcar alertas de precios sobre sus monedas. De este modo, se enviará una alerta cada vez que el precio suba o baje del valor marcado por el usuario.

- **Sistema de recomendaciones:** basándonos en las búsquedas y consultas de los usuarios podremos deducir que monedas están suscitando más interés entre los usuarios y así recomendar y ofrecer estadísticas del estado de estas.

- **Para cada criptomoneda en que divisa está más rentable:** ésta funcionalidad también surgió en una de las reuniones con nuestro director de TFG. Debido a las distintas tasaciones y fluctuaciones que las divisas tienen, puede ser que el cambio de las criptomonedas a unas de estas divisas sea más rentable que a otras. Ésta funcionalidad puede ser interesante para usuarios que quieran vender alguna de sus criptomonedas y quiera sacar el máximo beneficio Antes de desarrollar esta funcionalidad debemos estudiar su viabilidad.

5. Conclusiones

Español

Cuando elegimos este proyecto lo hicimos porque a ambos nos llamó la atención la oportunidad de poder desarrollar un bot de telegram. Los dos somos usuarios muy activos de la plataforma y defensores de ella frente a su competencia, ya que ofrece un mundo de posibilidades, en parte, gracias a los bots y por su naturaleza open source. Aún así sabíamos que nos enfrentábamos a un mundo totalmente desconocido y que nos iba a costar mucho esfuerzo y tiempo, sobre todo, familiarizarnos con el entorno, lenguajes de programación y las nuevas herramientas que necesitaríamos para llevar a cabo nuestro proyecto.

Otro punto importante al que hemos tenido que enfrentarnos ha sido el de auto gestionar nuestro propio proyecto. Hasta ahora todos los proyectos que habíamos llevado a cabo venían predefinidos o bien por la universidad o bien por nuestro trabajo. Estos proyectos ya estaban pensados y nosotros simplemente nos habíamos dedicado a desarrollarlos cumpliendo los requisitos impuestos y en algunas ocasiones generando documentación. Sin embargo en este proyecto hemos tenido total libertad, teniendo la posibilidad de poder elegir las herramientas con las que nos encontrásemos más cómodos, las librerías que considerásemos que nos iban a resultar mejores para el cometido de nuestro proyecto, etc. Esto a priori puede parecer una ventaja, pero analizar los pros y los contras de todo lo anterior mencionado conlleva una responsabilidad que hasta ahora no habíamos asumido, ya que una mala decisión podía ocasionar grandes pérdidas de tiempo, y el tiempo es crucial cuando un proyecto tiene fechas de entrega, como es el caso. Gracias a este proyecto una de las cosas que hemos aprendido ha sido esto, ser capaces de analizar qué herramientas se adaptan mejor al proyecto y ponerlas en práctica, aunque éstas herramientas fuesen desconocidas.

Con respecto a la parte más técnica, nos hemos tenido que enfrentar a aprender desde cero el lenguaje de programación Python. Al enfrentarnos a esta situación nos

hemos dado cuenta de una habilidad que hemos adquirido durante la carrera: poder adaptarnos a cualquier lenguaje de programación sin necesidad de mucho esfuerzo apoyándonos en los conocimientos previos que hemos adquirido. El periodo de adaptación fue relativamente corto, aun siendo Python un lenguaje con una sintaxis y estructura que difiere bastante de otros lenguajes de programación orientados a objetos que ya conocíamos, como Java o C++. También decidimos utilizar MongoDB en nuestro proyecto, encontrándonos con la misma situación, ninguno habíamos usado nunca MongoDB, pero tampoco bases de datos no relacionales. El resultado fue similar, la curva de aprendizaje fue parecida a la que experimentamos con Python.

Decidir entre Telepot y Python-Telegram-Bot fue otra decisión importante que tuvimos que tomar. Ambas nos proveían una interfaz para trabajar con la API para el desarrollo de bots de Telegram pero conseguimos entender mejor cómo funcionaba la librería de PTB, por lo que decidimos utilizarla, ya que entendiendo cómo funcionaba nos iba a resultar más fácil plasmar nuestras ideas usando esta librería. Además, la librería de PTB tiene dos grupos en Telegram, @pythontelegrambotgroup y @pythontelegrambottalk. En el primero se resuelven todas las dudas correspondientes en la librería de PTB en concreto, en el segundo también resuelven dudas de Python y temas relacionados. En alguna ocasión hemos tenido que consultar alguna duda y la respuesta ha sido prácticamente inmediata.

Estamos muy satisfechos con el trabajo realizado y con el funcionamiento del bot. Hemos conseguido implementar casi todas las funciones que queríamos y lo hemos hecho sin descuidar nuestro trabajo. Cuidando la estructura y aplicando las buenas prácticas a la hora de programar, logramos mantener un proyecto fácil de entender, escalable y modular. Además en el proceso hemos aprendido a utilizar nuevas tecnologías y hemos puesto en práctica las técnicas de gestión aprendidas en la carrera.

CryptoMarket es ahora un bot totalmente funcional y que ayudará a muchos usuarios a acercarse al mundo de las criptomonedas, además, por su carácter de código abierto, cualquier miembro de la comunidad podrá implementar nuevas funciones o aportar nuevas ideas.

English

We chose this project because both of us were attracted by the opportunity to develop a Telegram bot. We are both active users and defenders of the platform against its rivals, as it offers a world of possibilities, in part, thanks to the bots and its open source nature. We knew that we were facing a totally unknown world and that it would cost us a lot of effort and time to familiarize ourselves with the environment, programming languages and the new tools we would need to develop our project.

Another important point that we have had to face has been the management of our own project. So far, all the projects we have carried out, were predefined either by the university or by our work. These projects were already organised and we only had to focus on developing them with the requirements imposed and, in some cases, generating documentation. However, in this project we have had total freedom,

having the possibility to choose the tools with which we were more comfortable, the libraries that we thought would be better for the purpose of our project, etc. This, a priori, may seem an advantage, but analyzing the pros and cons it brings a responsibility that until now we had not assumed, since a bad decision could cause great losses of time, and time is crucial when a project has delivery dates, as is the case. Thanks to this project, one of the things we have learned is to be able to analyze which tools are best suited to the project and use them, even if these tools were unknown.

On the technical side, we had to learn the Python programming language from scratch. Facing this situation we realized an ability that we have acquired during the studies: to be able to adapt to any programming language without much effort, based on previous knowledge. The adaptation period was relatively short, even though Python is a language with a syntax and structure that differs a lot from other object oriented programming languages that we already know, such as Java or C ++. We also decided to use MongoDB in our project, finding the same situation, none of us had ever used MongoDB or any other non-relational database. The learning curve was similar to the one we experienced with Python.

Deciding whether to use Telepot or Python-Telegram-Bot was another important decision that we had to make. Both provided an interface to work with the API for the development of Telegram bots but finally we got to understand better how the PTB library worked, so we decided to use it, since this better understanding would make it easier for us to transform our ideas into real functionalities. In addition, the PTB library has two groups in Telegram, @pythontelegrambotgroup and @pythontelegrambottalk. In the first, all the doubts that corresponds to PTB library are solved, in the second one, they also solve Python related questions and other topics. We have had to ask some questions and the answer has been almost immediate.

We are very satisfied with the work done and with the performance of the bot. We have managed to implement almost all the features we wanted and we have done it without neglecting our work. Caring for the structure and applying good practices when programming, we have managed to maintain a project that is easy to understand, scalable and modular. Also in the process, we have learned to use new technologies and we have put into practice the management techniques learned in the career.

CryptoMarket is now a fully functional bot that will help many users to get closer to the world of cryptocurrencies, also, due to its open source nature, any member of the community can implement new functions or contribute with new ideas.

Contribuciones personales

En general, ambos integrantes hemos participado en todas las partes del proyecto exceptuando los bots iniciales, los cuales fueron desarrollados individualmente, aunque siempre cooperando entre nosotros y compartiendo el código en git.

El proceso de desarrollo de esta memoria ha sido colaborativo y todo su contenido ha sido previamente consensuado por ambos miembros del grupo. A su vez, dicho contenido ha sido revisado por nuestro director, Carlos Gregorio.

Contribuciones personales Diego del Corral

En este apartado hablaré de mis contribuciones personales tanto al proyecto CryptoMarket como al proceso de desarrollo, aprendizaje y mantenimiento del mismo.

Mi primera aportación ha sido la configuración de un servidor personal donde poder tener total acceso a un entorno de desarrollo completo para el lanzamiento de bots. Este servidor no es más que un ordenador antiguo que he restaurado para poder utilizar la última versión de Ubuntu Server (18.04) en el cual han sido instaladas todas las herramientas y librerías necesarias para el correcto funcionamiento del bot.

Paralelamente, desarrollé utilizando la librería Telepot, un bot muy sencillo para la monitorización del servidor y así poder actuar en caso de fallo. Este bot permite saber el nombre del host, la IP pública, así como hacer un reinicio de la máquina y recibir avisos si la IP pública ha cambiado. También permite a través de la API proporcionada por No-ip, cambiar la IP que apunta al subdominio dcorral1.hopto.org en caso de que esta haya cambiado. De este modo las conexiones ssh se pueden configurar para tener siempre acceso al servidor sin importar la IP pública de la máquina. Este bot se ayuda de tareas que yo mismo elaboré y que se llevan a cabo usando el archivo crontab de linux. En este archivo se pueden definir tareas (cron job) que se llevarán a cabo de forma periódica. Por ejemplo la IP pública se consulta cada 10 minutos y se compara con la anterior guardada previamente en un fichero.

Durante el desarrollo de este bot encontré otro wrapper en Python de la API oficial de Telegram, Python-Telegram-Bot. Esta librería, a parte de tener una comunidad más grande y activa, facilita una serie de métodos como CommandHandler() y Dispatcher() que hacen que el código se adapte más a la estructura inicial que habíamos pensado. Por ello, tras hablar con mi compañero y tener el visto bueno del director, inicié el traspaso de todo el desarrollo que teníamos ya implementado usando Telepot a esta nueva librería Python-Telegram-Bot. Este proceso me llevó algo más de una semana, el bot ya se encontraba en un estado muy avanzado y trasladar las funcionalidades de una librería a otra era en ocasiones una tarea que requería mucha concentración y trabajo.

Decidí crear un nuevo repositorio que acabaría siendo el repositorio definitivo donde llevar a cabo el resto de implementaciones. Llevé a cabo la creación y configuración de un nuevo repositorio git alojado en GitHub.com dando acceso a mi compañero para que pudiera subir sus cambios.

Como medida de seguridad, investigué cómo hacer que el bot se conectase a la base de datos de forma segura. Para ello fue necesario cambiar la configuración de la instancia de MongoDB y se creó un archivo de configuración donde irían los datos necesarios para la conexión del bot con la base de datos así como el Token que proporciona Telegram para establecer la conexión con el bot. Este archivo, no se puede hacer público por la naturaleza privada de sus datos, por ello proporcionamos un archivo de muestra, config_template.py, para que el bot pueda ser ejecutado de forma segura en cualquier servidor.

Entrando en implementaciones concretas dentro del proyecto se encuentra una de mis aportaciones más importantes, el sistema de traducciones para integrar nuevos idiomas. Cuando decidimos dar al usuario la opción de elegir el idioma, empezamos desarrollando esta función basándonos en condicionales que se repetían a lo largo de todo el código, pronto nos dimos cuenta de que no era una solución escalable y que convertiría nuestro código en algo incomprensible con la aparición de nuevos idiomas. Por ello decidí pensar una nueva manera de conseguir las traducciones manteniendo la modularidad y escalabilidad del proyecto. Tras un periodo de pruebas y reorganización de código llegué a la solución final.El sistema de traducciones se centra la encapsulación de todas las traducciones en un diccionario 'languages.py' y se accede proporcionando un identificador único del objeto que vaya a mostrar ese texto.

La estructura del diccionario es la siguiente:

```
LANG = {
        'ENG' : {
                'id'  : 'text',
                'id2': 'text'...
        },
        'ESP' : {
                'id'  : 'texto',
                'id2': 'texto'...
        }...
}
```

Mediante una función auxiliar get_text() con el identificador y el idioma del usuario como parámetros de entrada, se devuelve el texto en el idioma correcto. De este modo añadir nuevos idiomas se convierte en algo totalmente trivial donde solo habría que añadir la nueva traducción a este diccionario y no se repite tanta cantidad de código a lo largo del proyecto. Este desarrollo contribuyó en gran medida a la legibilidad y estructuración del código, además de facilitar las posibles aportaciones futuras.

Siguiendo la línea de esta implementación seguí reestructurando el proyecto creando archivos cuyas funciones fueran más concretas y reutilizables. Por ejemplo cambié la manera de crear los teclados de las vistas, encapsulando todas las funciones implicadas en la generación de estos en un archivo llamado keyboards.py. Con esto conseguí estandarizar la manera de crear estos teclados dentro del proyecto.

Estas y otras implementaciones se pueden consultar revisando los commits realizados en el GitHub del proyecto

La creación del bot de monitorización, el sistema de traducciones, la creación y configuración del repositorio en GitHub y la reestructuración del proyecto son mis aportaciones más importantes, sin embargo como ya hemos comentado, el resto del trabajo se llevó a cabo de forma conjunta y colaborativa donde ambos integrantes hemos realizado tareas complementarias para la finalización del bot CryptoMarket.

Contribuciones personales Carlos Piña

Para comenzar a familiarizarnos con los bots de telegram y la librería *Telepot* comenzamos a desarrollar cada uno un bot. El bot que yo he realizado ha sido el de estadísticas de jugadores de Clash Royale obteniendo los datos de la API RoyaleAPI (*https://docs.royaleapi.com/#/*). RoyaleAPI no es una API oficial de Supercell (Empresa desarrolladora del juego Clash Royale) y por tanto la estabilidad era mínima, ya que cuando la API recibía demasiadas peticiones de los usuarios que la utilizaban dejaba de responder. Para evitar esto los desarrolladores añadieron autenticación mediante *token* para controlar el uso de esta. En este momento el bot dejó de funcionar y tuve que actualizarlo para que las llamadas a la API incluyesen una autenticación por medio del *token* mencionado anteriormente. La actualización para dar compatibilidad al bot con este medio de seguridad la realicé con ayuda de un pequeño tutorial que los desarrolladores publicaron en su web.

En un principio el bot CryptoMarket disponía de una lista de las monedas más populares para las cuales se podía consultar su precio actual. Esta lista se mostraba en forma de *InlineKeyboard,* siendo cada botón una moneda diferente. Una de las primeras aportaciones al código del bot fue incluir un botón de actualizar la información de la criptomoneda en el mismo mensaje en el que se mostraba dicha información. Este botón permitía mejorar la usabilidad del bot, ya que hacía que no fuese necesario volver a iniciar el bot, solicitar todas las monedas y pulsar en una de ellas para que se volviese a solicitar su información. Este botón hacía uso de la función *editMessageText()* que nos facilitaba *Telepot*. A esta función se le pasa el id del usuario y el id del mensaje, junto con el texto a mostrar y el InlineKeyboard asignado al mensaje (en caso de tenerlo).

Ligado a la funcionalidad anterior se añadió un botón 'back' en la vista que mostraba la información de cada moneda. Este botón se añadió para evitar tener que iniciar el flujo desde cero cada vez que se quería consultar otra criptomoneda distinta a la que se estaba mostrando actualmente, permitiendo volver a mostrar la vista con el *InlineKeyboard* de criptomonedas desde la vista de información una moneda.

Posteriormente añadí el comando *top10*. Este comando hacía una petición a la API de CoinMarketCap solicitando las 10 monedas más populares y las mostraba en el *InlineKeyboard* que antes mostraba las monedas de forma estática. Esta petición utilizaba la librería *request,* y los datos los recogíamos con la librería json, ambas librerías de Python.

Telepot no facilitaba una interfaz clara para la gestión de las distintas formas de interacción que proporcionan los bots de Telegram (texto plano, pulsaciones en botones que se devuelven en forma de *callback_query,* comandos, etc.) por lo que se

construyó una clase que gestionaba los comandos *commandHandler.py*. Esta clase se encargaba de gestionar los distintos comandos que había añadidos a nuestro bot, lo que facilitaba la comprensión del código cuando el bot comenzó a tener varios comandos.

Después de implementar nuestros propios Handlers nos dimos cuenta de que había otra librería de Python para los bots de Telegram llamada Python-Telegram-Bot que nos facilitaba estas clases. Esta librería disponía de gran cantidad de Handlers implementados y decidimos utilizar esta para CryptoMarket. Esto supuso rehacer lo que ya habíamos hecho con *Telepot*, ahora con la nueva librería *PTB*. Una vez realizada esta tarea, comencé a interactuar con la base de datos en MongoDB que teníamos añadida al bot. Por lo que configure MongoDB en mi portátil personal como se describe en el punto 2.5. de este documento. Para interactuar con MongoDB desde Python utilizamos la librería PyMongo. Como nunca había utilizado MongoDB y por consiguiente, tampoco PyMongo, empecé a buscar información de cómo se interactuaba con este tipo de base de datos. Lo primero que hice fue guardar los usuarios en la base de datos, de los usuarios, inicialmente guardábamos solamente el id que Telegram les asigna y una lista de las criptomonedas que cada usuario tenía añadidas a su Wallet. Posteriormente también se guarda la configuración del bot (idioma y divisa preferida) y dos valores booleanos para facilitar la navegación del bot. Una vez guardados estos datos también añadí la funcionalidad necesaria para cargar el Wallet del usuario desde la base de datos.

En lo referente a la base de datos también implementé la funcionalidad que guarda todas las monedas que nos facilita la API de CryptoCompare para no tener que realizar una llamada cada vez que necesitemos saber si una criptomoneda está disponible en nuestro bot, así como mejorar la velocidad del bot.

Una vez guardadas todas las monedas disponibles en nuestra base de datos podíamos implementar de forma eficiente la funcionalidad de añadir moneda a Wallet. Fuí el encargado de implementar este flujo que funciona de la siguiente manera:

1. Desde el propio Wallet el usuario hace click en el botón correspondiente a añadir moneda.

2. El bot enviará un nuevo mensaje al usuario pidiéndole que le envíe el nombre corto de la criptomoneda, por ejemplo: BTC.

3. El bot comprueba si esa moneda existe en la base de datos. En caso de existir, añadirá la moneda al Wallet y mostrará la vista de este (con la moneda añadida). En caso de no existir la moneda en la base de datos editará el mensaje indicandolo y dando la posibilidad de repetir la búsqueda.

Ligado a la funcionalidad de añadir, implementé la funcionalidad de buscar criptomoneda. Esta funcionalidad es muy similar a la anterior pero en vez de redirigir a la vista de Wallet cuando la moneda existe, se muestra la información referente a la criptomoneda.

Para hacer el bot más fácil de usar se nos ocurrió añadir los botones de “añadir criptomoneda” y “eliminar criptomoneda” en la vista de detalle de cada

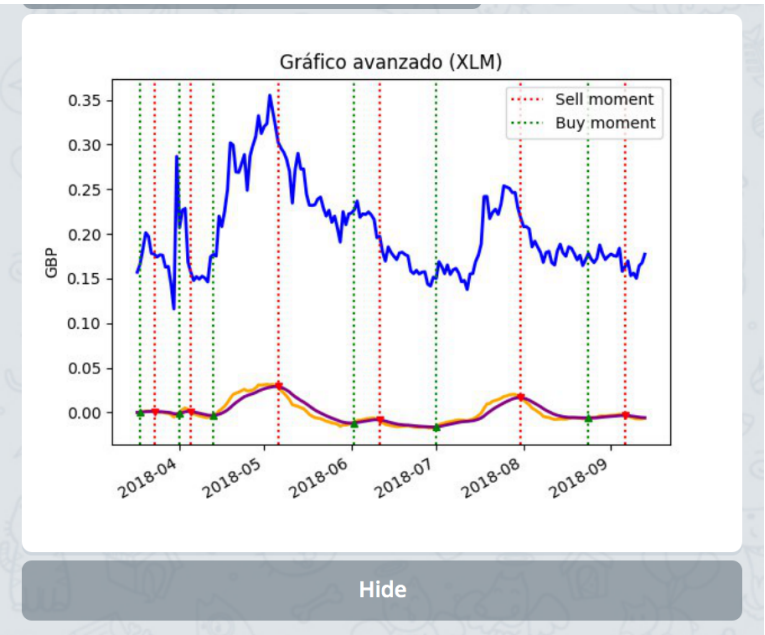
criptomoneda. El botón de “añadir criptomoneda” aparece cuando dicha moneda no se encuentra actualmente en el wallet del usuario, mientras que el botón de “eliminar criptomoneda” aparece en el caso contrario. El flujo que sigue el boton de añadir criptomoneda es el mismo que he descrito anteriormente.

Nos pareció interesante hacer el bot un poco más personal por lo que se nos ocurrió añadir un apartado de configuración. Fui el encargado de desarrollar esta funcionalidad. Por el momento las configuraciones disponibles en nuestro bot son el idioma (por el momento inglés y español) y la divisa preferida en la que el usuario quiere recibir los valores de las criptomonedas (dólar americano, euro, yen japonés, dólar australiano, libra británica, dólar de Hong Kong). Lo primero fue añadir los campos necesarios a los usuarios que guardabamos en base de datos, lo que resultó bastante fácil gracias a la forma en la que MongoDB gestiona los datos. En un principio, tanto para saber el idioma como la divisa en la que mostrar los mensajes simplemente se consultaba la base de datos y se usaba un texto u otro en los mensajes. Más adelante esta funcionalidad fue refactorizada en la parte de los idiomas para hacerla más manejable y escalable con un sistema basado en diccionarios.

También realicé la funcionalidad de mostrar gráficas con los históricos de datos que CryptoCompare nos facilita. Esta funcionalidad la realicé utilizando la librería de Matplotlib. Como nunca había utilizado dicha librería comencé creándome un proyecto vacío para ir probando sin comprometer el código de CryptoMarket, además las pruebas eran más rápidas en un proyecto pequeño y más manejable. Una vez conseguí los resultados deseados implementé la funcionalidad en el bot principal. El bot permite crear gráficas actualizadas de todas las criptomonedas que nos facilita la API de CryptoCompare en distintos periodos de tiempo: última hora, últimas 24 horas, última semana y último mes.

Aprovechando que la API de CryptoCompare nos permitía obtener los datos de varias criptomonedas con una sola consulta también he implementado un report en el que se muestra información de las criptomonedas que el usuario tiene en el Wallet. Se realiza una consulta a CryptoCompare en la que se indican las monedas que el usuario tiene añadidas a su wallet y se muestra información individual de cada una de ellas y se indica al usuario el balance total de las últimas 24 horas teniendo en cuenta estas monedas.

Por último he incluido la funcionalidad de gráfica avanzada en nuestro bot, digo incluido porque el estudio de cómo generar ésta gráfica ha sido realizado por ambos integrantes del grupo debido a nuestro desconocimiento de sobre los indicadores de compra y venta, . Esta funcionalidad ha sido de las que más nos ha costado entender, a los que en el mundo de la economía se denominan Trend Trading. Una vez realizado el análisis de los indicadores y realizadas las pruebas en un proyecto vacío fui el encargado de implementar esta funcionalidad en nuestro bot. En la gráfica se muestra la evolución, tanto del valor de la criptomoneda (línea azul), como del indicador MACD (Moving Average Convergence Divergence) (línea naranja) y su señal (línea morada). Las líneas verticales verdes y rojas indican cuando es un buen momento para comprar y vender, respectivamente.



31 Vista del gráfico avanzado del XLM

Referencias y bibliografía

- *Python: https://docs.python.org/3/*
- *Telegram API: https://core.telegram.org/bots/api*
- *Telegram: https://telegram.org/*
- *Telepot: https://telepot.readthedocs.io/en/latest/reference.html*
- *Python-Telegram-Bot: https://python-telegram-bot.readthedocs.io/en/stable/*
- *MongoDB: https://docs.mongodb.com/manual/*
- *PyMongo: https://api.mongodb.com/python/current/api/pymongo/index.html*
- *Matplotlib: https://matplotlib.org/2.2.3/index.html*
- *Requests: http://docs.python-requests.org/en/master/*
- *RoyaleApi: https://docs.royaleapi.com/#/*
- *CryptoCompare: https://min-api.cryptocompare.com/*
- *CoinMarketCap: https://pro.coinmarketcap.com/api/*
- *Pandas: https://pandas.pydata.org/pandas-docs/stable/*