YourPlacesBot - A Telegram Bot

David Quesada López y Mateo García Fuentes

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

UNIVERSIDAD
COMPLUTENSE
MADRID

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

Director: Carlos Gregorio Rodríguez

15 de junio de 2017

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumno/s y tutor/es del Trabajo Fin de Grado (TFG) en el Grado en ……de la Facultad de ……., autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TF) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Periodo de embargo (opcional):

- ☐ 6 meses
- ☐ 12meses

TÍTULO del TFG: .....
.....
Curso académico: 20..... / 20.....

Nombre del Alumno/s:
.....
.....

Tutor/es del TFG y departamento al que pertenece:
.....
.....
.....

Firma del alumno/s

Firma del tutor/es

Agradecimientos

Gracias a Nick Lee (https://github.com/nickoala) por desarrollar `Telepot`, un *framework* de `Python` para la *API* de `Telegram Bot` y ofrecerlo bajo una licencia `MIT`.

Índice general

Índice de figuras

Índice de abreviaturas

- **API:** del inglés *Application Programming Interface*, es una interfaz de programación de aplicaciones. Es una definición de los servicios (funciones/funcionalidades) que un determinado módulo *software* provee a otros módulos.
- **ESRI:** del inglés *Environmental Systems Research Institute*, es un proveedor de *software* de sistemas de información geográfica (GIS), aplicaciones web de GIS y aplicaciones de gestión de bases de datos geográficas.
- **GPL:** del inglés *General Public License*, es una licencia de derechos de autor de *sofware* libre creada por la *Free Software Foundation*.
- **GPS:** del inglés *Global Positioning System*, es un sistema de posicionamiento a nivel global que permite determinar la posición de un objeto en nuestro planeta.
- **HTTPS:** del inglés *Hypertext Transfer Protocol Secure*, es un protocolo seguro de transferencia de hipertexto. Consiste en la comunicación a través de HTTP (Hypertext Transfer Protocol) dentro de una conexión cifrada que asegura la autenticación del sitio web visitado y la protección de la privacidad e integridad de los datos intercambiados.
- **JSON:** del inglés *JavaScript Object Notation*, es un formato de archivo que utiliza texto legible para humanos para transmitir objetos de datos.
- **OTAN:** del francés *Organisation du Traité de lÁtlantique Nord*, es una alianza militar entre 28 estados de Europa y Norteamérica.
- **QWERTY:** es una distribución de teclado. Su nombre proviene de las primeras seis letras de su fila superior de teclas. Es la distribución más común aunque en algunos países europeos tienen ligeras modificaciones.
- **SQL:** del inglés *Structured Query Language*, es un lenguaje de consulta estructurada. Es un lenguaje diseñado para gestionar datos contenidos en un sistema de gestión de bases de datos relacionales.
- **PC:** del inglés *personal computer*, son las siglas que denominan un ordenador personal.

Resumen

`Telegram` es una aplicación de mensajería instantánea móvil y de escritorio basada en la nube. Esta aplicación, como la mayor parte de las aplicaciones de mensajería instantánea, permite intercambiar mensajes y archivos de todo tipo. En 2016 conseguía llegar a los 100 millones de usuarios y sus servidores procesaban 15 mil millones de mensajes al día.

Una de los principales atractivos de `Telegram` es su plataforma para bots. Los usuarios pueden crear sus propios bots y ponerlos en funcionamiento para que sean accesibles a todos los clientes de `Telegram` sin coste alguno para el desarrollador o para el consumidor. Estos bots proporcionan en su mayoría información, juegos o utilidades dentro de un *chat*, y aumentan en gran medida la funcionalidad de `Telegram`.

Gracias a esto, hoy en día `Telegram` no se limita sólo a la mensajería instantánea, sino que también actúa como un asistente personal virtual. Los asistentes virtuales son una característica fundamental en dispositivos móviles, facilitando el día a día de los usuarios y ofreciéndoles información que les es relevante en cualquier momento que lo necesiten. Los bots de `Telegram` le hacen especialmente potente en este ámbito, ya que continuamente aparecen bots que dotan a `Telegram` de nuevas capacidades como asistente personal.

A la hora de que un usuario interactúe con un bot, es especialmente interesante que el servicio que se le preste pueda depender de su ubicación geográfica y pueda tener un componente social. Por ello, este proyecto tiene como objetivo informar al usuario sobre qué establecimientos cercanos hay en base a su localización, ofreciendo la posibilidad de encontrarlos fácilmente y de ver datos proporcionados por otros usuarios sobre estos.

Palabras clave: bot, Telegram, geolocalización, noSQL, mapas de calor.

Abstract

`Telegram` is a cloud-based mobile and desktop instant messaging application. This application, like most instant messaging applications, allows you to exchange messages and files of all kinds. By 2016 it managed to reach 100 million users and their servers processed 15 billion messages a day.

One of the main features of `Telegram` is its bot support. Users can create their own bots and launch them to be available for everyone in `Telegram` without cost for neither the developer nor the client. These bots mainly offer information, games or utilities inside the chat and they increase greatly `Telegrams` functionality.

This opens the path for `Telegram` to act not only as an instant messaging application, but also as a virtual personal assistant. Virtual assistants are a vital feature of mobile dispositives, making the user's everyday life easier and offering them relevant information at any time they need it. `Telegram` bots make it especially powerful in this ambit, given that there are bots constantly being created that offer new personal assistant capabilities.

When an user interacts with a bot, it is of special interest that the response of the bot varies depending on the users location and on a social component. That's why this project aims to inform the user about what near by establishments there are depending on his location, offering the posibility to find them easily and to see information of them given by other users.

Keywords: bot, Telegram, geolocation, noSQL, heatmaps.

Capítulo 1: Introducción

1.1. Motivación

Si hacemos un análisis de la sociedad actual es fácil observar que el uso de dispositivos móviles es una constante en el día a día de cada persona. Y que la funcionalidad que ofrecen estos aumenta a una gran velocidad. Estos dispositivos son ya una parte esencial de nuestras vidas, y como consecuencia sus aplicaciones también.

Una de las fortalezas de estos dispositivos es la ayuda que nos ofrecen para procesar toda la información que hay en la red. Las aplicaciones con toda esa información nos ayudan a ampliar nuestra percepción del mundo y a tomar decisiones. Otra de las bazas importantes de las nuevas tecnologías son el carácter social que están adquiriendo, con recomendaciones, fotos, *likes* ... Dentro de estas aplicaciones sociales las de mayor uso son las de mensajería instantánea que nos ponen en contacto con nuestros amigos y conocidos, y nos permiten compartir todo tipo de información.

En algunas de las conversaciones con nuestros amigos había quejas de no disponer de una herramienta sencilla y amigable para encontrar bares, restaurantes y otros tipos de locales de ocio cercanos. En lo primero que piensas para buscar una solución a este problema es en la aplicación de `Google Maps`. Pero esta aplicación no es cómoda sino está embebida en otra aplicación debido a la enorme cantidad de información que ofrece.

En esta situación, pensamos que un bot de `Telegram` encajaría bien como solución. Por un lado, `Telegram` es una aplicación que usamos día a día y un bot resultaría fácil de manejar. Por otro lado, su carácter social nos permite tener un *feedback* con los usuarios que nos permita establecer cuales son las mejores opciones que nos ofrece `Google Maps`.

1.2. Objetivos

Al empezar este proyecto, la idea principal era crear un bot que funcionase como asistente virtual de los usuarios a la hora de ayudarles a encontrar locales cercanos a ellos. Valiéndonos de la plataforma que nos ofrece `Telegram`, somos capaces de facilitar el uso de este asistente a millones de usuarios sin tener que preocuparnos de la plataforma en la que lo vayan a usar, ya sean *smartphones*, *tablets* o incluso *pc's*. El bot estará en funcionamiento de manera ininterrumpida y mejorará sus prestaciones en medida del uso que le den otros usuarios, según le vayan aportando nueva información.

De cara a la implementación del bot, este se valdrá de la ubicación que se le envíe para mostrar locales cercanos. Para esto, vamos a establecer una serie de objetivos funcionales que el bot debe cumplir:

- Ser capaz de **recibir una localización** y devolver una serie de locales cercanos a ella. Estos locales deberán estar dentro de un cierto radio de distancia variable y por defecto se buscará que estén abiertos en el instante de la búsqueda.
- Cuando se seleccione un establecimiento, **se devolverá su ubicación** y la información de la que disponga el bot sobre él.
- Poderle dar una **puntuación** a los establecimientos y ser capaz de enviar fotografías al bot.
- Poder mostrar un **mapa de calor** en torno a tu localización reflejando las ubicaciones más concurridas por otros usuarios.
- Mostrar **información** de uso del bot de cara a su posterior desarrollo.

1.3. Estructura del documento

Vamos a estructurar este documento conforme al orden en el que fuímos desarrollando el bot. Para cada versión con nuevas funcionalidades, documentaremos qué hemos cambiado, cómo lo hemos hecho y a qué problemas nos hemos enfrentado.

A continuación mostramos sobre qué va a tratar cada capítulo:

- **Capítulo 2: Infraestructura para bots - `Telepot`**. En este capítulo introduciremos al lector al desarrollo de bots en `Telegram` contando cómo es la infraestructura que tienen y la *API* que ofrece `Telegram` a los

desarrolladores. También hablaremos de qué es y por qué hemos usado el *framework* de `Telepot` para nuestro bot en vez de la *API* nativa de `Telegram`.

- **Capítulo 3: Experimentación previa**. Esta parte tratará de nuestro primer acercamiento a la creación de bots. Hablamos de cómo empezamos a usar `Telepot` para codificar bots simples y lo que aprendimos para usar en un futuro en nuestro bot final.

- **Capítulo 4: Primera versión - Establecimientos cercanos**. En este apartado hablamos sobre el desarrollo de la primera versión estable de **YourPlacesBot**. Esta versión cumple con nuestro primer objetivo que era ofrecer los establecimientos cercanos al usuario.

- **Capítulo 5: Segunda versión - Base de datos e interacción**. Aquí tratamos el desarrollo de la versión que ya era capaz de realizar toda la interacción entre el bot y el usuario. Es la primera versión que pusimos a funcionar de manera continuada en `Telegram`.

- **Capítulo 6: Tercera versión - Opciones y *Heatmaps***. Este capítulo trata sobre la tercera versión, donde se introducen opciones sobre el funcionamiento del bot que cada usuario puede variar y mapas de calor en base a tu ubicación.

- **Capítulo 7: Cuarta versión - Escribir direcciones y estadísticas**. Este apartado trata sobre la última versión del bot, en la cual nuestro objetivo era mejorar y facilitar el uso del bot. Permitiendo al usuario escribir su ciudad, barrio, dirección en vez de tener que mandar la ubicación o comprobar las estadísticas de uso del bot a nivel global.

- **Capítulo 8: Conclusiones y trabajo futuro**. En este último capítulo sacamos conclusiones sobre el desarrollo del bot en `Telegram` y exponemos las mejoras futuras a implementar en el bot.

Chapter 1: Introduction

1.1. Motivation

If we make an analysis of our current society it's easy to observe that the use of mobile devices is a constant in the day to day of each person. And that the functionality they offer increases at a great speed. These devices are already an essential part of our lives, and as consequence their applications too.

One of the strengths of these devices is how they help us processing all the information on the network. Applications with all that information help us broaden our perception of the world and make decisions. Another important asset of the new technologies is the social character that they are acquiring, with recommendations, photos, likes ... Within these social applications the most used are instant messaging that put us in touch with our friends and acquaintances, and allow us to share all kind of information.

In some of the conversations with our friends there were complaints about not having a simple and friendly tool to find bars, restaurants and other types of nearby establishments. The first thing you think to look for a solution to this problem is in the `Google Maps` application. But this app isn't comfortable unless it's embedded in another applications due to the huge amount of information that it offers.

In this situation, we thought that a `Telegram` bot would fit well as a solution. On one hand, `Telegram` is an application that we use every day and a bot would be easy to handle. On the other hand, its social character allows us to have feedback with the users that helps us establish which are the best options that `Google Maps` offers us.

1.2. Objectives

At the beginning of this project, the main idea was to create a bot that would work as a virtual assistant for users to help them to find places near them.

Using the platform offered by `Telegram`, we are able to facilitate the use of these assistant to millions of users without having to worry about the platform that they'll use, whether they are smartphones, tablets or even pcs. The bot will be in operation in an uninterrupted way and will improve its performance in the measure of the use that other users give it, as they are contributing with new information.

In order to the implement of the bot, this will use the location that it's sent to show nearby establishments. For this, we will establish a series of functional objectives that the bot must comply:

- Be able **to receive a location** and return a series of nearby establishments. This premises must be within a certain radius of variable distance and will be searched by default at the time of the query.
- When an establishment is selected , its location and the information that the bot has on it will be returned.
- It can give a **rate** to the establishments and to be able to send photos to the bot.
- To be able to show a **heat map** around your location reflecting the locations most crowded by other users.
- To show **info** of use of the bot for its further development.

1.3. Structure of the document

We're going to structure this document according to the order in which we were developing the bot. For each version with new features, we'll document what we have changed, how we have done it and which problems we have faced.

Following is what will treat each chapter:

- **Chapter 2: Infrastructure for bots - `Telepot`**. In these chapter we'll introduce the reader to the development of bots in `Telegram` telling how is the infrastructure that they have and the API that `Telegram` offers to the developers. Also we'll talk what it's and why we have used the `Telepot` framework for our bot instead of the `Telegram` native API.
- **Chapter 3: Previous experimentation**. This part will deal with our first approach to the creation of bots. We talk about how we start to use `Telepot` to encode simple bots and what we learned to use in our final bot.

- **Chapter 4: First version - Nearby establishments**. In this chapter we talk about the development of the first stable version **YourPlaces-Bot**. This version carries out with our first objective that was to offer the the establishments near the user.
- **Chapter 5: Second version - Database and interaction**. Here we treat the development of the version which was already capable of performing all the interaction between the bot and the user. This is the first version that we put to run continuously in `Telegram`.
- **Chapter 6: Third version - Options and Heatmaps**. This chapter deals with our third version, which introduces options about the operation of the bot that each user can change and heat maps based on your location.
- **Chapter 7: Fourth version - Write addresses and statistics**. This chapter is about the latest version of the bot, in which our goal was to improve and facilitate the use of the bot. Allowing the user to type their city, neighborhood, address instead of having to send the location or check the statistics of use of the bot at global level.
- **Chapter 8: Conclusions and future work**. In this last chapter we write the conclusions about the development of our `Telegram` bot and expose the future improvements to be implemented in the bot.

Capítulo 2: Infraestructura para bots - `Telepot`

Para comenzar a desarrollar un bot, hay que familiarizarse con las herramientas que ofrece `Telegram` para ello. Una vez entendido el funcionamiento de la plataforma de `Telegram`, de su *API* y de `Telepot`, estaremos listos para empezar a codificar nuestros bots.

2.1. Plataforma de bots de `Telegram`

El servicio de mensajería `Telegram Messenger` ofrece una interfaz de programación para bots, llamada `Telegram Bot API`. Estos bots son aplicaciones de terceros que se ejecutan en `Telegram`. La forma de interactuar con estos bots es muy variada: mensajes, comandos, archivos, botones, etc. Las características usuales de estos bots son proporcionar notificaciones y noticias personalizadas, juegos o integración de servicios externos a `Telegram`.

Los mensajes, comandos y peticiones enviados por los usuarios se envían al *software* que se ejecuta en los servidores. Los servidores de `Telegram`, que son intermediarios entre nuestro servidor y el usuario, manejan el cifrado y la comunicación. Esta comunicación se realiza a través de una interfaz `HTTPS` que ofrece una versión simplificada de la *API* de `Telegram`.

Los desarrolladores deben crear sus bots a través de *@BotFather*. Este bot es el padre de todos los bots y proporciona *tokens* únicos para cada uno. También es necesario para administrar los bots y gestionar sus configuraciones. Además de esto, el *@BotFather* avisa a los desarrolladores de bots populares (300 solicitudes por minuto o más) si detecta que la tasa de conversión de solicitud/respuesta es muy baja. Está alerta consta de 3 botones de respuesta: Arreglado, Silenciar y Apoyo. Esta última opción abrira un nuevo *chat* con el `@BotSupport` que ayudará al desarrollador a detectar el problema o averiguar si hay algún problema en los servidores de `Telegram`.

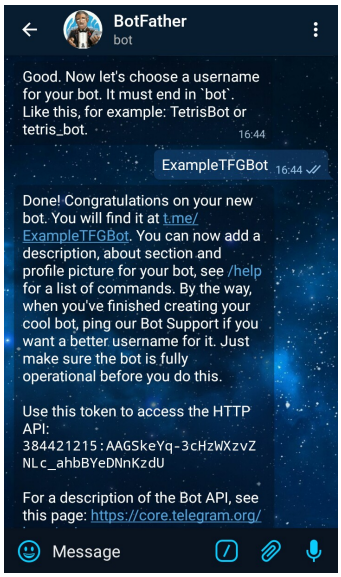

Figura 2.1: *@BotFather* provee los *TOKEN*

2.2. `Telepot`

`Telegram` está desarrollado bajo una licencia GPL, es decir, es *software* libre. Por lo que todo el mundo puede ver su código. Esto ha permitido que programadores de todo el mundo hayan creado diversos *frameworks* para la plataforma de bots de `Telegram` adaptándose a diferentes lenguajes u otras necesidades.

`Telepot` es uno de estos *frameworks*, desarrollado por Nick Lee (`https://github.com/nickoala/telepot`) te ayuda a desarrollar bots de `Telegram` usando `Python`. La funcionalidad básica de `Telepot` consiste en servir de *wrapper* de todas las funciones de la *API* de `Telegram` para así poder programar el código de los bots en `Python`. Como comentábamos en la sección anterior el uso de `Telegram Bot API` se realiza mediante solicitudes `HTTPS` a este servicio.

Además de esto, también ofrece funcionalidades para hacer más cómoda la gestión de servidor de tu bot. Cuando creas un bot, tienes que ir pidiendo a los servidores de `Telegram` *updates* para que te envíen una lista con los mensajes que le han sido enviados. `Telepot` te proporciona métodos para que no tengas que preocuparte de montar toda una estructura que gestione su recepción. `Telepot` se encarga de crear hilos de ejecución por cada *chat* nuevo que se abre con el bot y de pararlos cuando estos llevan demasiado tiempo sin realizar ninguna tarea.

Nick Lee ofrece también una guía de referencia de `Telepot` (`http://telepot.`

`readthedocs.io/en/latest/reference.html`) en la que explica en detalle las funciones de las que dispone el *framework*. El uso conjunto del amplio repertorio de ejemplos de código del repositorio de `GitHub` junto con esta guía de referencia hacen que se agilice bastante el proceso de aprender a codificar bots.

Cabe destacar también que el repositorio está siendo actualizado con frecuencia para adaptarse a las nuevas funcionalidades que va añadiendo `Telegram`.

Capítulo 3: Experimentación previa

De cara a afrontar este proyecto, nos encontramos con que debíamos aprender a hacer un bot sin tener conocimientos previos ni de la *API* de `Telegram` ni de `Python`. Por tanto, para coger soltura con el lenguaje y con el funcionamiento de los bots dedicamos la primera parte del proyecto a crear bots que realizasen tareas sencillas. Para crear estos bots, fuímos guiándonos del tutorial a base de ejemplos de `Telepot` (`http://telepot.readthedocs.io/en/latest/`).

3.1. Creación de los bots

El primer paso básico para crear un bot es hablar con *@BotFather* para crear tu bot con un nombre y que te envíe el *token* de tu bot. Este *token* es necesario para poder pedirle al servidor de `Telegram` las *updates* de tu bot. Una vez obtenido el *token*, hay que crear un objeto Bot con el *framework* de `Telepot` para obtener de manera periódica las *updates* de nuestro bot. Este proceso sería:

```
1 TOKEN = '255866015:AAFvI3sUR1sOFbeDrUceVyAs44KlfKgx-UE'
2
3 bot = telepot.Bot(TOKEN)
4 bot.message_loop(on_chat_message)
```

La clase Bot de `Telepot` es el *wrapper* principal de las funciones en `HTTPS` de la *API* de `Telegram` a `Python`. Llamando al método de *message_loop* conseguimos que todos los *updates* nuevos los trate la función que le indiquemos por argumento, en este caso *on_chat_message(msg)*.

Todos los mensajes que maneja el bot tienen formato *JSON*. Estos mensajes se envían primero a los servidores de `Telegram`, y son ellos los que se encargan de redirigirlos a su destino. Cuando el bot recibe los mensajes, se puede emplear la función *telepot.glance(msg)* para saber qué contiene (texto, imágenes, audio,...), en qué tipo de *chat* se ha enviado (grupo o privado) y el id de la persona que lo

envió. También se puede acceder a esta información accediendo directamente al *JSON* del mensaje.

```
def on_chat_message(msg):
    content_type, chat_type, chat_id = telepot.glance(msg)

    if content_type == 'text':
     if msg['text'] == '/start':
      bot.sendMessage(chat_id, 'Bienvenido.', reply_markup=
    keyboard1)
     elif msg['text'] == 'Comenzar' and  not (chat_id in convers):
    peticion(chat_id)
     elif msg['text'] == 'Finalizar' and  chat_id in convers:
      terminarPareja(chat_id)
```

Ejemplo de un *on_chat_message*

3.2. Teclados personalizados

En estos primeros bots también experimentamos con los distintos tipos de teclado que ofrece la *API* de `Telegram`, los *ReplyKeyboardMarkups* y los *InlineKeyboardMarkups*. La diferencia básica entre estos dos teclados es la posición que ocupan en la pantalla. El primero se sitúa fuera del propio *chat* sitúandose en el lugar que le correspondería al teclado *QWERTY* y el segundo está dentro de la propia conversación obligatoriamente acompañado de un mensaje de texto. Otra diferencia es la manera de procesar las pulsaciones, mientras las pulsaciones a los botones de los *ReplyKeyboardMarkups* se escriben automáticamente en el *chat* como mensaje del usuario, las pulsaciones de los botones de los *InlineKeyboardMarkups* se procesan internamente, cada vez que se pulsa uno de estos botones se crea una clase *ButtonHandler*. Estos botones se procesan como *inline* queries al ser pulsados, de manera que al llegar al bot se diferencian del resto de los mensajes del *chat*.

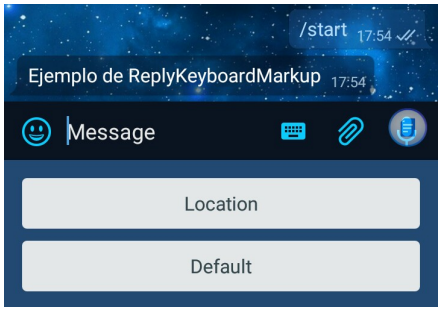

Figura 3.2: ReplyKeyboardMarkup

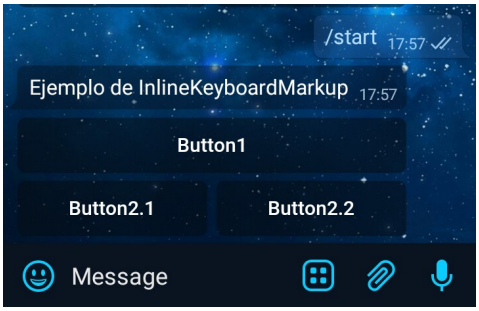

Figura 3.3: *InlineKeyboardMarkup*

3.3. Gestión de mensajes - *DelegatorBot* y *handlers*

Lo último con lo que hicimos pruebas en esta primera fase fue con la gestión del servidor. En un principio, nuestros bots procesaban los *updates* que les llegaban de manera secuencial. Cuando ya empezaron a realizar tareas más complejas, tuvimos que crear servidores simples que lanzasen un hilo por cada nuevo *chat_id* que les llegase en un mensaje. Así, el bot disponía de un diccionario con los *chat_id* y una cola, para que cada mensaje proveniente de un *chat_id* ya conocido se encolase para su hilo correspondiente.

```
def comienzo(chat_id):
 q = Queue()
 t = threading.Thread(target=partida, args=(chat_id, q,))
 t.setDaemon = True
 partidas[chat_id] = q
 t.start()
```

Ejemplo de gestión de mensajes nuevos

Tras crear algunos bots que utilizaban este sistema de servidor con hilos por paso de mensajes, nos dimos cuenta de que `Telepot` proporciona una clase para gestionar todo esto de manera cómoda: la clase *DelegatorBot* y los *handlers*. Esta clase tiene la particularidad de que el objeto bot que creas trata de manera diferente los *updates* que le llegan dependiendo del tipo que sean. Además, este objeto se encarga de lanzar hilos por cada nuevo *chat_id* que le llega y los tiene todos registrados por debajo en un diccionario, para poder redirigirles los mensajes que les correspondan a cada uno y poder cerrarlos cuando sea necesario.

```
bot = telepot.DelegatorBot(TOKEN, [
    pave_event_space()(
        per_chat_id(), create_open, UserHandler, timeout=180),
    pave_event_space()(
        per_callback_query_origin(), create_open, ButtonHandler,
    timeout=180),
])

bot.message_loop(run_forever='Listening')
```

Ejemplo de creación de un *DelegatorBot*

Para gestionar cada mensaje de una forma u otra es necesario definir un *handler* del tipo que corresponda: para gestionar lo relacionado con mensajes en un *chat* se necesita un *ChatHandler* y para tratar las pulsaciones en teclados *inline* se necesita un *CallbackQueryOriginHandler*. Una vez creadas las clases de los *handlers*, debe ponerse un *pave_event_space()* por cada tipo de *updates* diferentes que vaya a tratarse. En los argumentos de esta función se especifica qué evento genera un nuevo hilo, en el caso de ejemplo serían *per_chat_id()* para el *ChatHandler* y *per_callback_query_origin()* para las pulsaciones en los teclados *inline*.

El uso del *DelegatorBot* y los *handlers* facilita en gran medida la creación de bots más complejos. Una vez que te acomodas a los diferentes eventos de creación de hilos y de *handlers* que te ofrece `Telepot`, te ahorra el tener que gestionar tú los hilos y los mensajes, siempre y cuando comprendas bien la gestión interna de estos hilos por parte del *DelegatorBot*.

Capítulo 4: Primera versión - Establecimientos cercanos

Tras la experimentación previa con `Telepot` y el desarrollo de varios bots básicos, comenzamos a programar el bot del proyecto. Nuestro principal objetivo en esta primera versión era ofrecer los locales cercanos según la localización que nos ofrecía el usuario.

4.1. Funcionalidad básica

Al iniciar el bot, el usuario recibía un mensaje que rezaba *Share your location!* a la vez que se le ofrecía un *ReplyKeyboardMarkup* con un botón que al pulsarlo compartía la localización del usuario. Después de recibir la localización, el bot continúa preguntando que tipo de establecimiento estaba buscando el usuario, dando varias opciones a través de un teclado *inline*. Cuando el usuario elegía que tipo de local que quería el bot le ofrecía una lista de los establecimientos de ese tipo a un radio de 500 metros. El usuario elegía entre alguno de los locales ofrecidos y el bot le mandaba la localización de este.

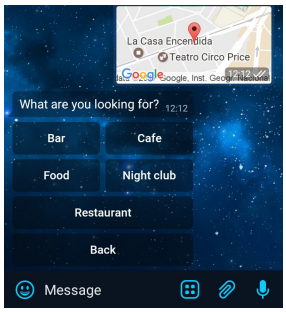

Figura 4.4: Enviar localización

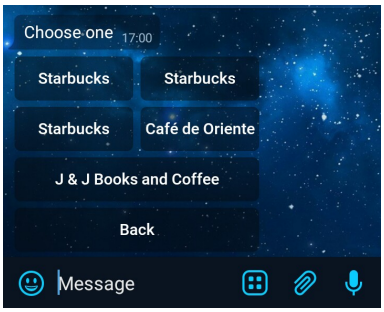

Figura 4.5: Escoger tipo

4.2. Uso de la *API* de `Google Maps`

Para conseguir los establecimientos en un radio de una localización concreta decidimos utilizar el cliente que proporcionan los servicios de `Google Maps` para `Python`. Utilizamos la función *places_nearby* que al pasarle como parámetros distintos aspectos de la consulta como la localización, el radio, el tipo de local, etc. devolvía una lista en formato *JSON* con todos los establecimientos, y sus datos, que cumplían los parámetros establecidos. Al recibir estos datos nos encargamos de procesarlos para mostrar parte de esta información a los usuarios.

```
1    js = mapclient.places(None, location=(latitude, longitude),
     radius=settings['radius'], language='es-ES', min_price=None,
     max_price=None, open_now=settings['openE'], type=
     establishmentType)
```

Consulta a la *API* de `Google Maps`

Decidimos usar los servicios prestados por `Google Maps` por ser la mayor aplicación a nivel mundial que ofrece las geolocalizaciones (entre otros datos) de establecimientos de ocio. Para poder utilizar estos servicios, tuvimos que registrarnos en `Google Developers` para obtener una clave de la *API* del maps de manera gratuíta con un límite de 1000 consultas diarias.

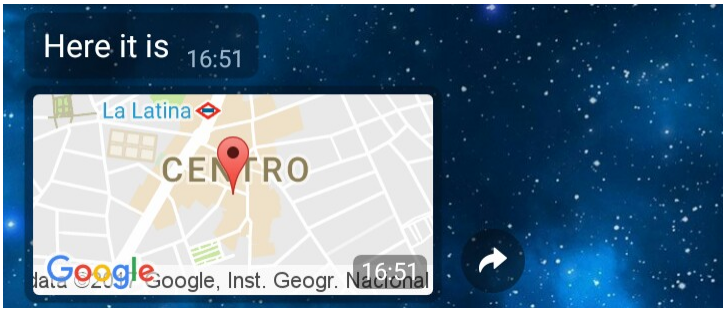

Figura 4.6: Escoger local

4.3. Limitaciones de `Telepot`

En este punto nos dimos cuenta de una de las limitaciones del los *handlers*. Al definirlos en la llamada a *DelegatorBot*, también hay que poner qué tiempo de *timeout* tienen. Una vez pasado este tiempo y si el *thread* de un *handler* no ha recibido ningún *update*, el *DelegatorBot* para ese hilo y destruye el objeto *handler* que hubiese. Si después de esto llegase un *update* en el mismo *chat* del hilo que se paró, se crearía un nuevo objeto *handler* y se lanzaría en un nuevo

hilo. Esto supone que si se declaró algún atributo privado en el *handler* este se perdió durante el *timeout*, por lo que hay que tenerlo en cuenta y guardar todos los datos relevantes de los *handlers* en la función *on_time_out()* definidas dentro de la clase del *handler* correspondiente, que es la que *DelegatorBot* llama antes de hacer el *timeout*.

Otro problema que observamos era que el uso del bot iba llenando el *chat* de mensajes y botones. Pero queríamos conseguir que la interfaz del bot se desarrollase en un solo mensaje y así evitar también que el usuario pueda pulsar botones de manera inesperada. Para mantener el flujo de interacción con el bot por medio de botones *inline* utilizamos continuamente la función *editMessageText* proporcionada por `Telepot`. Esta función es un *wrapper* de la función de la *API* de `Telegram` que permite editar un mensaje enviado anteriormente y cambiar su texto y el teclado que proporciona.

```
1         self.editor.editMessageText(msg, reply_markup=None)
```

Edición de mensajes previos

Capítulo 5: Segunda versión - Base de datos e interacción

A medida que íbamos desarrollando el bot nos encontrabamos con los primeros problemas, observabamos la necesidad de almacenar los datos y además queríamos mejorar la interacción con el bot para poder establecer un *feedback* con los usuarios.

5.1. `MongoDB`

Como indicabamos en el capítulo anterior el uso del *DelegatorBot* tenía el inconveniente de perder los atributos privados si no hay actualizaciones en un período determinado. Además queríamos guardar datos como las localizaciones que nos enviaba el usuario para utilizarlos en funcionalidades posteriores. Ante esta coyuntura decidimos crear una base de datos para el bot, algo indispensable en casi cualquier aplicación. La problemática con la que nos encontrábamos era si utilizar una base de datos relacional o no. Nuestra decisión final fue utilizar `MongoDB`, porque nos ofrecía la capacidad de almacenar nuestros datos que no tenían la misma estructura, pero si parecida, de una manera cómoda. Además nos permitía una futura escalabilidad de esas semi-estructuras que sabíamos que íbamos a necesitar con las futuras funcionalidades del bot.

Para gestionar la base de datos con el bot implementamos un nuevo módulo utilizando `Pymongo`, que es una librería de `Python` que ayuda a gestionar las bases de datos con `MongoDB`. Dentro de la nueva base de datos establecimos dos colecciones, una de usuarios y otra de establecimientos. Estas dos colecciones no implementan funciones *join*, dado que en `MongoDB` y en *NoSQL* en general son muy lentas. El bot cuando quiere guardar datos o extraerlos de la base de datos llama a una función específica del nuevo módulo, y este se encargará de conectar con la base de datos para extraer o persistir los datos. Para solucionar el problema del *timeout*, cuando este período de tiempo transcurre invocamos al método *on_close()* que persiste los atributos que van a expirar en la base de datos.

5.2. Interacción con el usuario

Llegados a este punto, aumentamos la funcionalidad del bot para que una vez que te enviase la localización de algún local y también te diese más opciones a realizar con ella. Pusimos la posibilidad de darle una valoración al local que se haya seleccionado, de cero a cinco estrellas, que también se muestra a la hora de enseñarte los locales cercanos a tí. Introducimos también la posibilidad de mandar imágenes de los locales, de forma que si un local tiene imágenes en la base de datos damos la opción de ir viéndolas a través de la interfaz con botones.

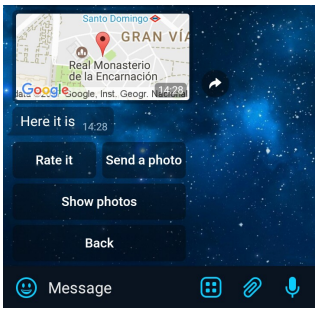

Figura 5.7: El usuario puede valorar y ver o envíar fotos

Cuando se le envía una imágen al bot, este recibe un *JSON* como si se tratase de un mensaje normal, pero una de las claves contiene el *id* que `Telegram` le da a la imagen. Es posible descargar el archivo de la imagen si se desea, pero con tener esta clave se puede reenviar y se recibirá como una imagen normal. De este modo, sólo es necesario guardar el *id* de las imágenes que se reciben en el documento de su local correspondiente. Usar `MongoDB` viene bien es este caso para poder guardar dentro de una misma colección documentos con distintas claves, ya que puede haber locales que no tengan puntuación o imágenes mientras que otros sí.

También para mejorar y dar más información sobre los locales a los usuarios, les proporcionamos la distancia a la que están respecto a su posición actual. Para calcularla utilizamos la fórmula del *haversine* que es una ecuación que calcula la distancia entre dos puntos de un globo sabiendo su longitud y su latitud. Esta función es sólo una aproximación, porque la Tierra no es una esfera perfecta, sino que su radio varía ascendentemente desde los polos hasta el ecuador debido a la forma geoidal del planeta. Como radio utilizamos la media geométrica, 6.367,45 kilómetros.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Figura 5.8: Fórmula de *haversine* desarrollada para calcular la distancia

5.3. *Steps*

Como ahora el bot dispone de más opciones para interactuar con él, también debía disponer de un botón en su interfaz para volver atrás. Como el flujo del bot se desarrolla parecido a una máquina de estados, creamos un módulo *steps* para definir los diferentes estados en los que puede encontrarse el bot, y así saber en cada momento a dónde debe ir en caso de recibir un *update* o de querer volver atrás. Estos estados ayudan también en el caso del *timeout*, ya que en la función *on_close()* se puede guardar en la base de datos el estado en el que se quedó un usuario, y cuando vuelva pueda continuar usando el bot desde el punto en el que lo dejó. Ahora que disponemos de base de datos, todas las variables locales que utiliza el *handler* también se pueden guardar en caso de *timeout*.

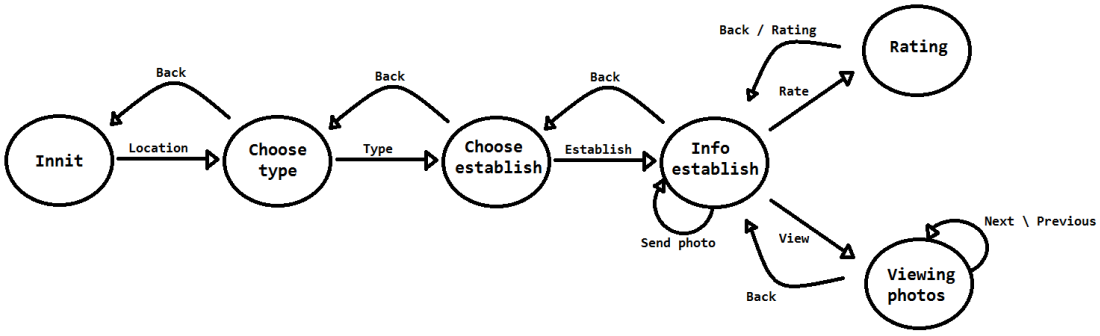

Figura 5.9: Estados posibles del bot

5.4. Otros cambios

Tras diversas comprobaciones del funcionamiento del bot, observamos que los establecimientos que nos enviaba el método *places_nearby* no coincidía totalmente con los locales que mostraba `Google Maps`. Esta consulta no devolvía todos los locales cercanos, sino sólo algunos. Por lo que en su lugar comenzamos a usar el método *places* de la misma *API*, que realiza una búsqueda más

exhaustiva. Pasándole por parámetro campos como la localización actual, el radio de la consulta o el tipo de establecimiento que el usuario está buscando.

```
1    js = mapclient.places(None, location=(latitude, longitude),
     radius=settings['radius'], language='es-ES', min_price=None,
     max_price=None, open_now=settings['openE'], type=
     establishmentType)
```

Consulta a la *API* de `Google Maps`

Capítulo 6: Tercera versión - Opciones y *Heatmaps*

Durante un tiempo estuvimos preparando el bot para que tuviese varios parámetros ajustables en cuanto a su funcionalidad. La *API* de `Google Maps` permite cambiar opciones de búsqueda como el radio de distancia, buscar sólo establecimientos abiertos o buscar establecimientos dentro de un rango de precios. También podíamos poner el bot tanto en español como en inglés. Juntando todo esto, podíamos crear un nuevo menú el en bot donde cada usuario pudiese cambiar estas opciones si lo desea y después lo persistimos en la base de datos.

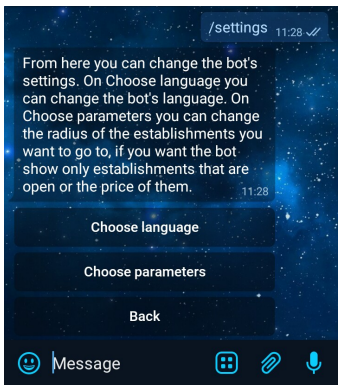

Figura 6.10: Menú de ajustes

6.1. Idiomas

Para tener el texto del bot en varios idiomas creamos un nuevo módulo que contiene todos los mensajes de texto y botones que puede enviar el bot. A la hora de enviar un mensaje por el bot, para introducir el texto se hace una llamada a la función correspondiente de este módulo y se le pasa por parámetro

una variable que determina si el texto se devolverá en inglés o en español. Por defecto, el bot enviará los mensajes en inglés y podrá cambiarse desde las opciones a español. Para añadir nuevos idiomas al bot, habría que traducir los métodos dentro del módulo *translate.py* y añadir la opción con el nuevo idioma en las opciones del bot. De este modo resulta cómodo poder añadir nuevos idiomas al bot.

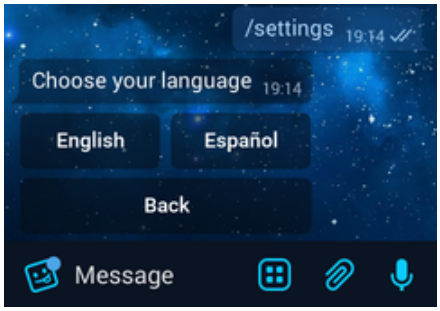

Figura 6.11: Elegir idioma

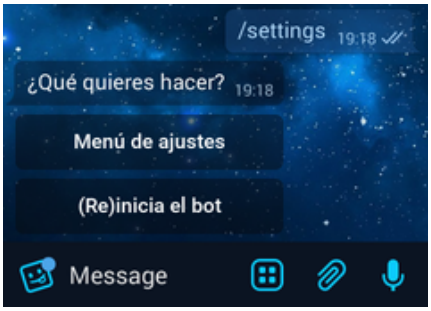

Figura 6.12: Tras cambiar a español

6.2. Opciones

De los parámetros variables de la función *places*, al final dejamos el radio de búsqueda, el número de establecimientos que se muestran en una búsqueda y buscar sólo establecimientos abiertos en el momento.

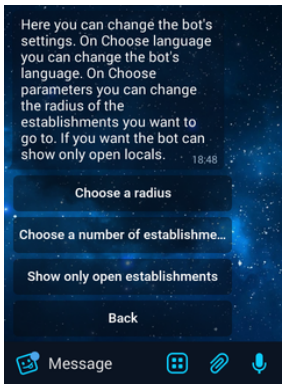

Figura 6.13: Opciones variables del bot

La opción de buscar establecimientos según un rango de precios de 0 a 4 daba resultados inesperados cuando la estuvimos probando. Al usar esta opción a veces no mostraba casi ningún establecimiento a pesar de ponerle que buscase

en todo el rango, de 0 a 4. Suponemos que esto se debe a que algunos establecimientos no tienen información acerca del rango de precios en el que están, y por tanto no aparecían en búsquedas que usasen ese parámetro. Por tanto, dado que resultó ser menos útil de lo esperado, decidimos no añadir esta opción al bot.

6.3. *Heatmaps*

El cambio más grande de funcionalidad que introdujimos en esta versión fue la capacidad de envíar mapas de calor al usuario. Para recibirlo el usuario debe introducir el comando */heatmap*, y el bot generará y enviará una imagen con un mapa de calor, cuyo centro es la última localización guardada del usuario que hace la petición. En este mapa de calor se muestran las zonas con más concurrencia cerca del usuario, usando las localizaciones de todos los usuarios.

Para realizar los mapas utilizamos el paquete de `Python Basemap`, que es una librería para trazar mapas 2D. Permite trazar contornos, imágenes, vectores, líneas o puntos en las coordenadas que se le proporcionen. Para construir el mapa es necesario proporcionarle a la librería los puntos geográficos que corresponden con la esquina inferior izquierda y la esquina superior derecha. Para calcular estos límites, teniendo en cuenta que conocemos el centro de la imagen, utilizamos la fórmula de *haversine* pero a la inversa. Si recapítulamos, recordamos que utilizabamos *haversine* para calcular la distancia entre dos puntos en un globo. Nosotros en este caso conocemos uno de los puntos, el ángulo y la distancia, por tanto si despejamos nuestra nueva incógnita tenemos como resultado, lo siguiente:

```
1    latup = math.asin(math.sin(lat)*math.cos(kmeters/R) + math.cos
     (lat)*math.sin(kmeters/R)*math.cos(bearing))
2    lonup = lon + math.atan2(math.sin(bearing)*math.sin(kmeters/R)
     *math.cos(lat), math.cos(kmeters/R)-math.sin(lat)*math.sin(
     latup))
```

Haversine inverso en `Python`

La funcionalidad básica de `Basemap` sólo ofrece mapas dibujados, lo cúal no nos interesa porque queríamos mayor precisión a nivel ciudad o incluso barrio. Pero sí ofrece la posibilidad de sustituir estos dibujos por otras imagénes. Índicando un código *EPSG*, acrónimo de *European Petroleum Survey Group*, que indican sistemas de referencia de coordenadas. En nuestro caso usamos el código 4326 que indica el sistema *World Geodetic System 1984* en el que está proyectado todo el planeta. Este sistema es usado por los *GPS* y por sistemas militares desarrollados por la *OTAN*. Y para obtener las imágenes vía satélite utilizamos el servicio `ArcGIS`, de *ESRI*, en concreto el *World Imagery* que tiene 24 niveles de resolución y imágenes en alta definición de casi todo el planeta.

```
1    map = Basemap(llcrnrlon=llcrnrlon,llcrnrlat=llcrnrlat,
     urcrnrlon=urcrnrlon,urcrnrlat=urcrnrlat, epsg=4326)
2    map.arcgisimage(service='World_Imagery', xpixels = 1500,
     verbose= True)
```

Consultas a `Basemap` y a `ArcGIS`

Capítulo 7: Cuarta versión - Escribir direcciones y estadísticas

Con la tercera versión ya cumplíamos los objetivos que habíamos establecido antes de empezar con el desarrollo del bot. En esta nueva versión nos centramos en hacer más amigable el uso del bot, de manera que el uso sea más rápido y cómodo.

7.1. Direcciones

En anteriores versiones la única forma de que el bot supiera tu localización era compartiéndola o utilizando la que ofrecemos por defecto. En este versión añadimos la posibilidad de escribir la dirección en formato texto, es decir, en el momento en el que el bot te pide la localización el usuario puede escribir la calle, el barrio o la ciudad en la que está. Este texto lo procesamos con la ayuda de otro servicio que ofrece la *API* de `Google Maps`, `geocode`, para transformarlo en unas coordenadas geográficas. Mostramos la dirección formateada en la conversación por si hay algún error debido que en distintas ciudades pueden haber calles con el mismo nombre. Y a partir de aquí seguiría el proceso de selección de local de la misma manera que si hubiese compartido la localización.

7.2. Cambios al mostrar los locales

Anteriormente al mostrar los locales mostrábamos primero en el texto uno por uno todos los establecimientos con la distancia a la que se encontraba el usuario y la puntuación media que tenía y luego una lista de *InlineKeyboardMarkups* de cada uno de los locales. Pero esto en lugares donde hay muchos locales provoca un mensaje muy largo por lo que decidimos cambiarlo. Ya que además como indicabamos en la sección de motivación uno de los problemas que nos encontramos en los servicios para buscar locales de ocio era la gran cantidad de información que nos ofrecían y que éramos incapaces de procesar.

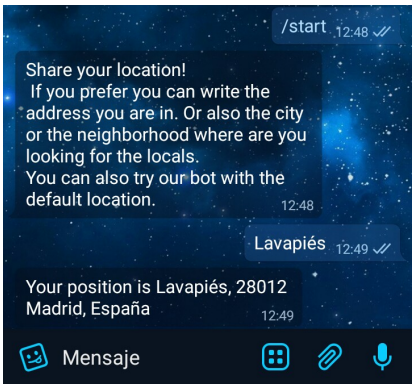

Figura 7.14: Envío de dirección escrita

Por ello a partir de esta versión, después de elegir el tipo de establecimiento al que el usuarios quiera ir ofrecemos, por mensaje de texto, los tres locales más cercanos a su posición y los tres mejor valorados por el resto de usuarios en el radio que el cliente tenga establecido. Y también mostramos la lista de *InlineKeyboardMarkups* de cada uno de los locales para que el usuario pueda seleccionar el que quiera.

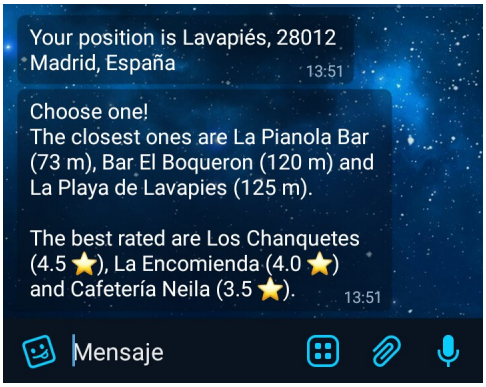

Figura 7.15: Cambios al mostrar los locales

7.3. Estadísticas

Esta nueva opción solo está disponible para los desarrolladores, se trata de un comando */stats*, que ofrece las estadísticas de uso del bot. Tras introducir el comando, que no mostramos en la lista de comandos, comprueba si el cliente que introdujo el comando es un superusuario. Los dos únicos superusuarios somos

los dos desarrolladores. Si un usuario no autorizado introduce el comando se le enviará un mensaje que le advierte que no tiene permiso para ver ese contenido y se le ofrece la única opción de volver hacia atrás. En el caso de que sí sea un superusuario se muestra el total de usuarios, los usuarios activos durante los últimos siete días, el total de local valorados y los que tienen fotos, y también las estadísticas en cuanto al idioma utilizado.

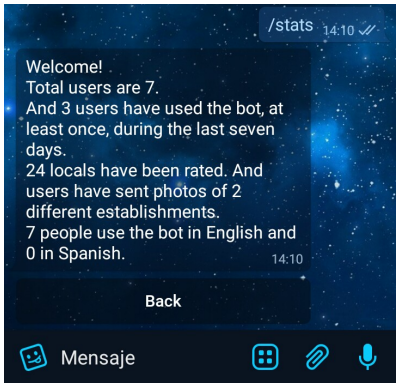

Figura 7.16: Estadísticas

7.4. Ayuda

Implementamos un nuevo comando en el bot, */help*, este comando está presente es casi todos los bots de `Telegram`. Y su única utilidad es la de resumir como funciona el bot, como utilizarlo y cuales son sus opciones.

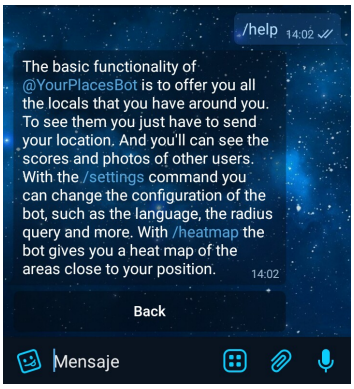

Figura 7.17: Ayuda

Capítulo 8: Conclusiones y trabajo futuro

Finalmente, vamos a recapitular sobre lo que hemos conseguido durante este proyecto de cara a las capacidades adquiridas y a las tecnologías empleadas y sobre el trabajo que queda por realizar en el futuro.

8.1. Bots de `Telegram`

Al empezar este proyecto, conocíamos los bots de `Telegram` y habíamos usado alguno antes, pero no sabíamos nada acerca de su creación. De cara a afrontar la codificación del bot, tuvimos que apoyarnos en nuestros conocimientos previos y aprender sobre la marcha las capacidades necesarias.

Los bots son una herramienta definitivamente potente de `Telegram`. La capacidad de poder crear herramientas a disposición de millones de potenciales usuarios sin tener que realizar más que unos pocos pasos de creación a través de *@BotFather*, junto con toda la plataforma que te ofrece `Telegram` ya montada pone a disposición de cualquiera la creación de bots. Esta plataforma ayuda en gran medida a la robustez de los bots, al sólo tener que encargarse de comunicarse con los servidores de `Telegram` para obtener *updates* y enviar mensajes.

También es cierto que por ello los bots tienen que adaptarse a la estructura que ofrece `Telegram`, y hay que ceñirse a las posibilidades de interacción que permite un *chat* de una aplicación de mensajería. A cambio de poder utilizar la interfaz de `Telegram`, tienes que asumir las limitaciones que presenta. Aún así, compensa adaptarse a su interfaz, ya que ofrece bastantes posibilidades y se van ampliando sus capacidades con bastante frecuencia.

Una vez asumida y comprendida la interfaz, el hecho de estar desarrollando un bot en `Telegram` se diluye y pasa a ser más un medio que un fin: estás realizando una aplicación que se sirve de la plataforma de `Telegram` para funcionar, simplemente ciñes la funcionalidad que quieres que tenga tu bot a las

posibilidades de interacción que ofrece `Telegram`, pero por debajo sigue siendo una aplicación que podrías haber adaptado a otro medio distinto.

En definitiva, los bots de `Telegram` son una muy buena posibilidad a la hora de decidir una plataforma para aplicaciones en dispositivos móviles, dadas las facilidades que ofrecen y la gran visibilidad que puedes llegar a lograr sin coste alguno.

8.2. `Python` y `Telepot`

En primer lugar, el lenguaje elegido fue `Python` para poder aprovechar el uso de `Telepot`. `Python` no es un lenguaje ampliamente utilizado durante el grado, por lo que no partimos con conocimientos sobre él. Sin embargo, sí que habíamos usado antes lenguajes orientados a objetos y lenguajes interpretados, por lo que pudimos adaptarnos a él tras el periodo de experimentación previa.

La sencillez al la hora de escribir código en `Python` hace bastante fácil su aprendizaje y uso, pero su mejor característica es la amplia gama de librerías de las que dispone.

En un principio, el uso de `Telepot` sólo sirve para codificar los bots en `Python`, pero una vez comprendido el uso de los *handlers* y de *DelegatorBot*, la carga de trabajo se ve reducida notablemente.

8.3. Base de datos

Para la base de datos, estamos en un entorno adecuado para *NoSQL*. Dada la diversidad de los datos y el formato *JSON* que utiliza `Telegram` para hacer llegar los mensajes enviados al bot, resultó una buena opción `MongoDB`. Este lenguaje orientado a documentos funciona bien fuera del entorno *cluster* al que están dirigidos gran parte de los lenguajes *NoSQL*.

Además, es sencillo integrarlo con `Python` a través de `Pymongo`. Como este lenguaje sí lo habíamos practicado antes, pudimos aplicar lo que ya sabíamos de su uso en terminal, dado que el funcionamiento es el mismo que con `Pymongo`.

Por ahora no tenemos pensado usar un *cluster* para la base de datos, ya que el tráfico y el tamaño no son suficientes. Aún así, si en un futuro queremos montarlo es posible crear un conjunto de réplica dado que ya partimos de usar `MongoDB`.

8.4. Creación de la memoria

Durante el grado, nunca nos habíamos enfrentado a la tarea de generar un documento de estas dimensiones y características. Por tanto, de cara a afrontarlo, decidimos emplear `LaTeX`, un sistema de composición de textos orientado a tareas de este estilo.

A pesar de que al principio cuesta un poco comprender la sintaxis, después resulta bastante sencillo su uso, dando facilidades a la hora de insertar citas al código y a la hora de estructurar el documento.

8.5. Trabajo futuro

Los objetivos marcados antes del desarrollo del bot los hemos cumplido, pero a medida que se íbamos desarrollando el código nos surgieron nuevas ideas y funcionalidades para añadir al bot. Y también mejorar algunas de las partes que ya son funcionales en el bot.

La idea de añadir reconocimiento de imágenes para que los usuarios nos pudieran enviar fotos de las cartas de comida o bebidas y poder mostrar la información al resto de los usuarios, surgió cuando implementamos la capacidad de poder enviar y ver fotos en el bot. Pero debido a las restricciones temporales tuvimos que decidir si implementar los mapas de calor o el reconocimiento de imágenes, y al final dimos prioridad a los mapas de calor.

Nos gustaría aumentar la funcionalidad de los mapas de calor. Aprovechando el navegador interno de `Telegram` que permite usar `HTML5` poder hacer un navegador sobre un mapa con todos los datos de nuestros usuarios. Es cierto que `Google Maps` ofrece un servicio parecido en su *API*, pero debido a las restricciones de uso de `Telegram Bot API` no lo podemos utilizar, ya que no está permitido abrir enlaces externos dentro del navegador. Además nos gustaría ampliar la funcionalidad temporal de los *heatmaps*, es decir, que se pudieran mostrar los mapas de calor según el día elegido o la hora. Para esto también tendríamos que añadir nuevos campos o incluso una nueva colección a nuestra base de datos.

Otra idea que se nos ocurrió pero debido a la falta de tiempo y de datos en nuestra base de datos aún no pudimos implementar. Se trata de recomendaciones de locales, según lo que otros usuarios que hayan valorado con una buena puntuación los mismos establecimientos a los que el usuario también haya dado una buena puntuación, ofrecer otros locales con buenas valoraciones por parte de estos usuarios en los que el usuario aún no haya estado.

Además de esto, como `Telegram Bot API` es una plataforma que está en con-

tinuo crecimiento, con nuevas actualizaciones y nuevas funcionalidades cada cierto tiempo nuestra idea es actualizar el bot a la vez que lo hace la plataforma. E implementaremos nuevas funcionalidades si siguen creciendo las opciones, que sean interesantes para el uso del bot, de la plataforma.

Chapter 8: Conclusions and future work

To conclude, we are going to over what we have acomplished during this project regarding the abilities we acquiered and the technologies we used. We will also talk about the future work that is left to be done.

8.1. `Telegram` bots

When we began this project, we knew about `Telegram` bots and we had used some of them before, but we didn't know anything about how they are created. When facing the coding of our bot, we had to rely on our previous knoledge and learn on the fly the necessary skills.

Bots are a definitely powerful tool of `Telegram`. The ability of being able to create tools that are at the disposition of millions of potencial users by following a few steps through *@BotFather*, along with the platform that `Telegram` makes it possible for anyone to create a bot. This platform also makes bots more robust, given that you only have to communicate with the `Telegram` servers to obtain updates and send messages.

However, it's also true that because of this, all the bots have to adapt to the structure offered by `Telegram`, and you have to stick to the ways of interaction with the user that a `Telegram` chat offers. In exchange to being able to use the interface of `Telegram` you have to assume its limitations. Even so, all the posibilities offered by `Telegram`, which are also being extended with regularity, compensate this.

Once you have comprenhended the interface, you stop thinking about developing a `Telegram` bot and begin thinking of `Telegram` as a medium and not as an end. You are developing an application that runs on `Telegram`'s platform, so you stick the funcionality of your bot to the possibilities of interaction offered by `Telegram`, but the application you are developing could have been adapted to a different platform.

All in all, Telegram bots are a very good possibility when you think about developing an application for mobile devices, given the advantages they offer and all the visibility that you can achieve without any cost.

8.2. Python and Telepot

When we began this project, we decided to code it in Python in order to be able to benefit ourselves from Telepot. Python is not widely used during the degree, so we had no previous knowledge of it. However, we had used previously object oriented languages and interpreted languages, so we could adapt easily to it after some time.

The easiness of writing code in Python makes learning it very comfortable, but its best characteristic is the wide array of libraries that it has.

At first, using Telepot is only useful to be able to code your bots in Python, but when you comprehend how handlers and the *DelegatorBot* works, it saves you a great amount of work.

8.3. Database

For the database, we are in a suitable environment for NoSQL. Given the diversity of the data and the JSON format used by Telegram to get the messages sent to the bot, it was a good option MongoDB. This document-oriented language works well outside the cluster environment to which most of the NoSQL languages are addressed.

In addition, it's easy to integrate it with Python through Pymongo. As this language we had practiced before, we could apply what we already know of its use in terminal, since the operation is the same as with Pymongo.

For now we don't plan to use a cluster for the database, since traffic and size aren't enough. However, if in the future we want to mount it it's possible to create a replica set since we are already using MongoDB.

8.4. Creating Memory

During the grade, we had never faced to the task of generating a document of these dimensions and characteristics. Therefore, in order to face it, we decided to use LaTeX, a system of task-oriented text composition of this style.

Although it's a little difficult to understand the syntax at first, then it is quite easy to use, giving easy access at the time to insert cites to the code and the time to structure the document.

Aportaciones individuales

En general ambos autores hemos participado en todos los aspectos del proyecto. La parte de la experimentación previa la realizamos de forma individual, aunque trabajando aspectos muy parecidos y utilizando `Telepot`.

David Quesada López

Cuando comenzó el proyecto conjunto, monté el repositorio de git que hemos utilizado durante todo el proyecto. Decidí ponerle una licencia `MIT` como la usada por `Telepot`, para que si alguien se encuentra con el proyecto y le puede ser de utilidad pueda valerse de él libremente mencionándonos.

Al comenzar a codificar el bot, creé la estructura base del bot a partir de los bots que había desarrollado en la fase de experimentación. Durante esta fase, me había centrado especialmente en el uso de los *handlers*, los *delegator* y los teclados *inline*, por lo que la estructura inicial fue lo bastante robusta como para soportar el resto de la codificación del bot.

Me encargué también de preparar la *API* de `Google Maps` para poder usarla, registrándose y obteniendo la clave. Hice las primeras pruebas del método *places_nearby*, aunque en este momento del desarrollo aún no nos dimos cuenta de que este método no funcionaba correctamente.

Implementé la base de datos en `MongoDB` con `Pymongo` en un nuevo módulo. Este nuevo módulo contendría todas las funciones relativas a la interacción con la base de datos. Monté la estructura de lo que empezaríamos guardando en la colección *usuarios*, y dado que usamos `MongoDB` es fácil cambiarlo en un futuro para guardar nuevos datos o añadir colecciones. Monté también el *script* para lanzar la base de datos en el servidor.

Junté en otro módulo todos los teclados que utilizamos en el bot, para que fuese más sencillo encontrarlos y realizar cambios en ellos. De esta forma para introducir un teclado a un mensaje lo llamamos como a una función. De cara a la funcionalidad del bot, me encargué de hacer la parte de recibir fotos de

cada local y guardarlas en la base de datos. Finalmente, cambié la forma de guardar localizaciones de establecimientos en la base de datos de cara a crear un índice geoespacial en un futuro.

En cuanto a la memoria, comencé escribiendo el resumen y el capítulo 1, montando la estructura que seguiríamos luego. Escribí el capítulo 2 y el capítulo 3, sobre los pasos previos a la creación de nuestro bot. El capítulo de la segunda versión lo escribimos a medias, encargándome yo de las secciones de `MongoDB` y de los *steps*. En la tercera versión hice las dos primeras secciones. Finalmente, escribí el último capítulo excepto el trabajo futuro.

Mateo García Fuentes

En la primera versión, me encargué de procesar los datos que devolvía el método *places_nearby* de manera que guardaba el nombre y las coordenadas de los locales para así mostrar los restaurantes en botones y al pulsar estos mandar la localización del establecimiento pulsado. Después también me encargué de filtrar los establecimientos por tipo, es decir, el método *places_nearby* acepta como parámetro unos determinados tipos de locales de los cuales mostramos cinco al usuario y una vez pulsado se le muestran al usuario los establecimientos de ese tipo.

En la segunda versión, me encargué de iniciar el primer esqueleto de *steps*, aún en el archivo principal, que pasaría a ser un módulo que controla los pasos de cada usuario en el bot. También me encargue de implementar la opción que permite ir hacia atrás a los usuarios, en el módulo *steps* ya separado que se invoca desde un *InlineKeyboardMarkup*. Después de que observamos que el método *places_nearby* no ofrecía todos los establecimientos posibles me encargué de cambiar este método por el nuevo *places* que ofrece el mismo servicio de `Google Maps`. De la nueva parte de la interacción con el usuario me encargué de implementar que el usuario pudiera realizar las puntuaciones del local que haya elegido y que mostrase la puntuación de todos los establecimientos en el paso de elegir local. También me encargué de investigar e implementar la fórmula de *haversine* para calcular la distancia entre la ubicación del usuario y el establecimiento. También implementé la funcionalidad de mostrar las fotos de los locales que las tuvieran.

En la tercera versión, implementé la funcionalidad de *settings* que son los ajustes que el usuario puede cambiar como el idioma, el radio de las consultas o si solo quiere ver los establecimientos abiertos. También me encargué de que estos ajustes se tuvieran en cuenta en la consulta que hacemos a la *API* de `Google Maps` donde le introducimo por parámetro los ajuste que tenga seleccionados el cliente. Programé el nuevo módulo de esta versión que era el encargado de traducir el bot, como explicamos anteriormente el bot recibe el idioma del

usuario y devuelve la frase correspondiente en el idioma indicado. La mayor parte de la implementación de los *heatmaps* lo hicimos en conjunto, excepto la función *calculateBounds* que se encarga de calcular la esquina inferior izquierda y la superior derecha de la imagen del mapa de calor utilizando la fórmula del *haversine*, como explicamos en un capítulo anterior.

En la cuarta versión, implementé la funcionalidad de ayuda, también añadí la nueva funcionalidad de escribir la dirección por texto con el nuevo método de *geocode*. Además me encargué de mejorar la manera de mostrarle la información de los establecimientos a los usuarios de forma que fuera más amigable para este.

En la memoria, me encargué de explicar los tipos de teclados que ofrece `Telegram Bot API` y escribí el capítulo de la primera version. De la segunda version expliqué las secciones de interacción con el usuario (valoraciones, *haversine*, ...) y de otros cambios (el cambio al método *places*). En el capítulo de la tercera versión me encargué de explicar como implementamos los mapas de calor. Además también escribí el capítulo de la cuarta versión y el índice de abreviaturas. Tras enviar el primer borrador me encargué de corregir el resumen y la introducion además de los capítulos 2 y 4.

Bibliografía

[1] LaTex ->http://texdoc.net/texmf-dist/doc/latex/memoir/memman.pdf

[2] Python ->https://docs.python.org/2.7/

[3] API Google Maps Docs ->http://googlemaps.github.io/google-maps-services-python/docs/2.4.5/

[4] Telegram Bots ->https://core.telegram.org/bots/api

[5] API Telepot ->http://telepot.readthedocs.io/en/latest/