

YourPlacesBot - A Telegram Bot

David Quesada López y Mateo García Fuentes

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INGENIERÍA INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

Director: Carlos Gregorio Rodríguez

24 de mayo 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  
☐ 12 meses

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 API de Telegram Bot y desarrollarlo bajo una licencia MIT.

# Índice general

Índice de figuras	v
Índice de abreviaturas	vi
Resumen	vii
Abstract	viii
<b>Capítulo 1: Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	1
1.3. Estructura del documento . . . . .	2
<b>Capítulo 2: Infraestructura para bots - Telepot</b>	<b>4</b>
2.1. Plataforma de bots de Telegram . . . . .	4
2.2. Telepot . . . . .	4
<b>Capítulo 3: Experimentación previa</b>	<b>6</b>
3.1. Creación de los bots . . . . .	6
3.2. Teclados personalizados . . . . .	7
3.3. Gestión de mensajes - DelegatorBot y handlers . . . . .	8
<b>Capítulo 4: Primera versión - Establecimientos cercanos</b>	<b>10</b>
<b>Capítulo 5: Segunda versión - Base de datos e interacción</b>	<b>13</b>
5.1. Base de datos . . . . .	13
5.2. Interacción con el usuario . . . . .	14
5.3. Otros cambios . . . . .	15
<b>Capítulo 6: Tercera versión - Opciones y Heatmaps</b>	<b>16</b>
<b>Capítulo 7: Conclusiones y trabajo futuro</b>	<b>17</b>
<b>Apéndice</b>	<b>18</b>
<b>Bibliografía</b>	<b>19</b>

<b>Anexo</b>	<b>20</b>
<b>Glosario</b>	<b>21</b>

## Índice de figuras

## Índice de abreviaturas

# Resumen

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.

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

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.

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 possibility 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

La motivación para crear este bot surgió a partir de unas conversaciones con amigos. Se quejaban de no disponer de una herramienta sencilla para poder ver qué locales hay cercanos a nosotros y así elegir a dónde ir si no conoces el lugar.

Al intentar buscar una solución a este problema lo primero en lo que pensamos fue la aplicación de Google Maps. Google Maps tiene la funcionalidad de encontrar establecimientos cercanos a ti en base a tu localización y a qué buscas, pero el resultado fue que la aplicación es demasiado grande y no es práctico tener que abrirla, navegar entre todas las opciones y funcionalidades que ofrece y ponerte a buscar un local que te guste.

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, la función de compartir la ubicación de Telegram se adecúa perfectamente a la API de Google Maps para realizar consultas de búsqueda de establecimientos cercanos.

## 1.2. Objetivos

Al empezar este proyecto, la idea principal era aprender a hacer un bot de Telegram partiendo de los conocimientos previos adquiridos durante la carrera. Para esto, también teníamos como objetivo mejorar nuestra habilidad con Python, que no es un lenguaje que se use ampliamente durante el grado, y con bases de datos no SQL, en este caso MongoDB que se usa en alguna asignatura. Otro punto importante era ser capaces de buscar e integrar APIs que nos resultasen útiles para nuestro bot. Finalmente, también queríamos coger soltura escribiendo memorias de esta extensión utilizando LaTeX, que no es algo que se practique antes del trabajo de fin de

grado.

El objetivo final de este proyecto será la implementación de un bot de Telegram que valiéndose de la ubicación que se le envíe devuelva locales con las características indicadas. Para esto, vamos a establecer una serie de objetivos funcionales que el bot debe cumplir:

- Ser capaz de **recibir una localización y devolver una lista 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

### 1.3. Estructura del documento

Vamos a estructurar este documento conforme al orden en el que fuimos 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 ya cuenta con la funcionalidad y la

interfaz básicas y sirve como base sobre la que poder ir implementando mejoras.

- **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 apartado trata sobre la última versión del bot, 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: 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.

# Capítulo 2: Infraestructura para bots - Telepot

## 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 abre 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.

## 2.2. Telepot

Telepot es un framework desarrollado por Nick Lee (<https://github.com/nickoala/telepot>) que te ayuda a desarrollar bots de

Telegram usando Python en vez de HTTPS. 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.

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.

## 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:AAFvI3sUR1sOFbeDrUceVyAs44KlIfKgx-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,

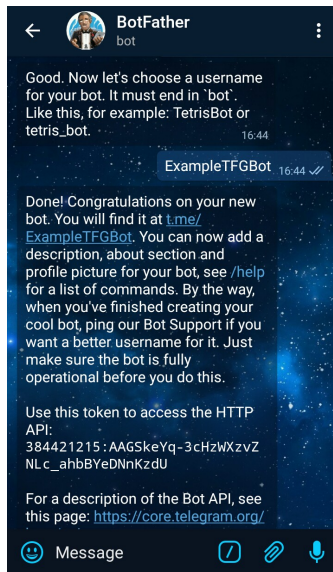


Figura 3.1: @BotFather provee los TOKEN

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.

```

1 def on_chat_message(msg):
2     content_type, chat_type, chat_id = telepot.glance(msg)
3
4     if content_type == 'text':
5         if msg['text'] == '/start':
6             bot.sendMessage(chat_id, 'Bienvenido.', reply_markup=
keyboard1)
7         elif msg['text'] == 'Comenzar' and not (chat_id in convers):
8             peticion(chat_id)
9         elif msg['text'] == 'Finalizar' and chat_id in convers:
10            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`.

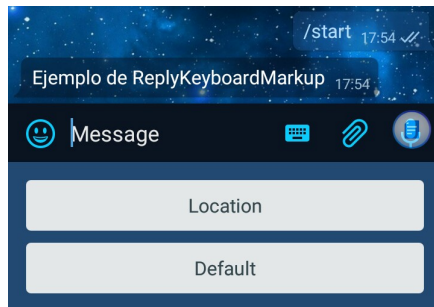


Figura 3.2: ReplyKeyboardMarkup

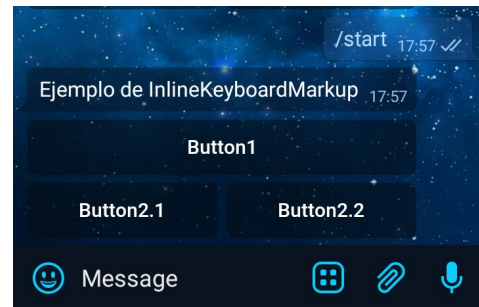


Figura 3.3: InlineKeyboardMarkup

### 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.

```
1 def comienzo(chat_id):
2     q = Queue()
3     t = threading.Thread(target=partida, args=(chat_id, q,))
4     t.setDaemon = True
5     partidas[chat_id] = q
6     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.

```
1 bot = telepot.DelegatorBot(TOKEN, [  
2     pave_event_space()(  
3         per_chat_id(), create_open, UserHandler, timeout=180),  
4     pave_event_space()(  
5         per_callback_query_origin(), create_open, ButtonHandler,  
6         timeout=180),  
7 ])
8 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

La primera versión del bot que realizamos cumplía el objetivo básico del bot. Según la localización del usuario, el bot mostraba los establecimientos cercanos. 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 al 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.

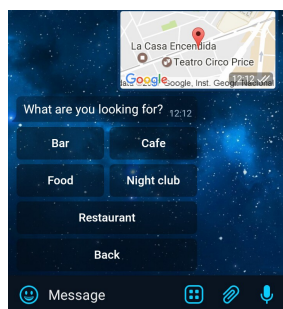


Figura 4.4: Enviar localización

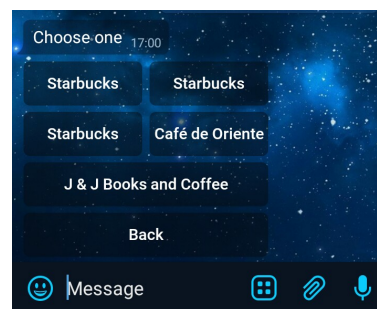


Figura 4.5: Escoger tipo

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 mostrarle 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 gratuita.

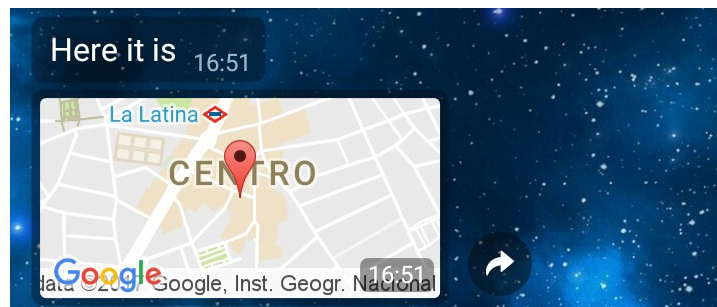


Figura 4.6: Escoger local

En este punto nos dimos cuenta de una de las limitaciones de 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.

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

Al hacer esto, se consigue que la interfaz del bot se desarrolle en un solo mensaje y así evitamos también que el usuario pueda pulsar botones de manera inesperada si se dejasen sin eliminar en el chat.

## Capítulo 5: Segunda versión - Base de datos e interacción

A medida que íbamos desarrollando el bot nos encontrábamos con los primeros problemas, observábamos la necesidad de almacenar los datos y además queríamos mejorar la interacción con el bot.

### 5.1. Base de datos

Como indicábamos 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*. Dentro de la nueva base de datos establecimos dos colecciones, una de usuarios y otra de establecimientos. 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

En esta segunda versión comenzamos a desarrollar funcionalidades para que hubiese una interacción entre el usuario el bot. En principio focalizamos esta interacción en que el usuario pudiera dar *feedback* sobre los establecimientos. Aprovechando que la mayor parte de los clientes de Telegram lo utilizan desde su dispositivo móvil, a pesar de su versión de escritorio, implementamos la capacidad de poder enviar al bot fotografías de los locales para que así otros usuarios pudieran consultarlas antes de realizar su decisión. Además de estos también añadimos la capacidad de que el usuario pueda valorar, con una nota de 0 a 5, el local. Y así mostrar a todos los usuarios la puntuación media que nuestros clientes han dado a determinado local.

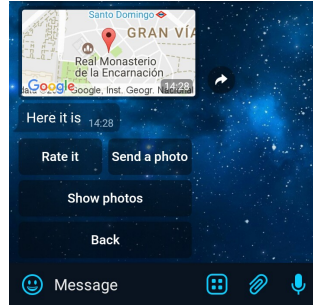


Figura 5.7: El usuario puede valorar y ver o envíar fotos

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

Con el propósito de que el usuario pudiera navegar por el bot cómodamente, creamos un nuevo módulo que se encarga de gestionar los pasos en los que esta cada usuario. Los diferentes pasos son las distintas etapas en las que puede estar la conversación con el bot, inicio, eligiendo tipo de

establecimiento, enviando una foto, etc ... Así este módulo es invocado cuando el usuario avanza o retrocede y es el encargado de comunicar cuál es el siguiente paso a tomar. Estos pasos se guardan en local como un atributo, pero en el momento en el que el método *on\_close* es invocado la etapa actual se guarda en la base de datos.

### 5.3. 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

## Capítulo 7: Conclusiones y trabajo futuro

## Apéndice

# Bibliografia

- [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/>

## Anexo

## Glosario