Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de un chatbot para la recomendación de eventos o lugares de interés

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Héctor Jover Ibáñez

Tutor: Pedro José Valderas Aranda

2019-2020

Resumen

Este trabajo desarrolla un chatbot para la aplicación de mensajería instantánea Telegram, con la finalidad de acercar la tecnología a la ciudadanía. Este bot ofrece un servicio de geolocalización, fácil de usar, para encontrar distintos puntos de interés, cerca del usuario, en la ciudad de Valencia, España.

Los datos con los que se trabaja provienen del portal de datos abiertos del Ayuntamiento de Valencia, con las credenciales facilitadas por la Universidad Politécnica de Valencia. Las tecnologías utilizadas son JavaScript, como lenguaje de programación, Telegram y NodeJS.

Palabras clave: chatbot, bot, puntos de interés, Telegram, NodeJs, geolocalización.

Abstract

This work develops a chatbot for the Telegram instant messaging application, in order to bring technology closer to the public. This bot offers an easy to use geolocation service, to find different points of interest near the user, in the city of Valencia, Spain.

The data with which we work comes from the open data portal of the city council of Valencia, with the credentials provided by the Universidad Politécnica de Valencia. The technologies used are JavaScript, as the programming language, Telegram and NodeJS.

Key Words: chatbot, bot, Telegram, points of interest, NodeJs, geolocation.

Resum

Aquest treball desenvolupa un chatbot per a l'aplicació de missatgeria instantània Telegram, amb la finalitat d'acostar la tecnologia a la ciutadania. Aquest bot ofereix un servei de geolocalització, fàcil d'usar, per a trobar diferents punts d'interés, prop de l'usuari, a la ciutat de València, Espanya.

Les dades amb els quals es treballa provenen del portal de dades obertes de l'ajuntament de València, amb les credencials facilitades per la Universitat Politècnica de València. Les tecnologies utilitzades són Javascript, com a llenguatge de programació, Telegram i NodeJS.

Paraules clau: chatbot, bot, Telegram, NodeJs, geolocalització.

Tabla de contenidos

Contenido

Tabla de figuras

1. Introducción

La mensajería instantánea ha tenido una evolución muy significativa en los últimos años. Con la introducción en el mercado de los smartphones (teléfonos inteligentes) y su posterior desarrollo, las aplicaciones de mensajería instantánea se han convertido en una de las aplicaciones con más usuarios, llegando a su auge recientemente con la introducción de aplicaciones como son WhatsApp, Telegram, QQ, Line, Snapchat y muchas otras.

Telegram [1] es una aplicación de mensajería lanzada en agosto del 2013 por los hermanos Pavel y Nikolai Durov. Esta aplicación tiene diversas características como son las conversaciones entre usuarios, envío de archivos, llamadas, desarrollo de bots etc. Nuestro proyecto trata del desarrollo de un chatbot y por tanto hemos aprovechado la extensa documentación que tiene esta plataforma sobre su creación y puesta en marcha.

Un Bot es un programa informático que suele tener como objetivo automatizar tareas que serían tediosas para un humano. Existen diferentes tipos de bots en base a su función, por ejemplo, los que rastrean información en la web se denominan *web crawlers* y los que mantienen conversaciones son *chatbots* o en español bots conversacionales. Siendo un poco más específicos, en Telegram los bots son cuentas especiales que no requieren de número de teléfono para ser creadas y ofrecen un servicio a los usuarios que interactúan con estos, mediante, mensajes, comandos o peticiones.

En este trabajo, nos vamos a centrar en desarrollar un bot conversacional que ofrece la funcionalidad de localizar distintos tipos de sitios, restaurantes, museos, fuentes, etc. alrededor de nuestro usuario. Los datos de los lugares los recogemos del Ayuntamiento de Valencia, que pone a disposición del público una serie de API´s o puntos de acceso, que nos permiten acceder a distintos tipos de información, como puede ser, puntos de acceso wifi, paradas de taxi cercanas, estaciones de Valenbisi y más. Mediante la creación de este bot, ponemos a disposición de la ciudadanía un servicio capaz de utilizar estos datos, que, aunque son de acceso libre, es necesario tener unos conocimientos tecnológicos suficientes, para poder hacer uso de ellos.

Por tanto, mediante un bot podemos ofrecer un servicio sin la necesidad de mantener contacto humano, que obtiene los datos de los usuarios de forma amigable y sencilla y que se puede mantener operativo las veinticuatro horas del día.

1.1 Motivación

Personalmente mi principal motivo para embarcarme en este proyecto fue la buena experiencia que tuve en la asignatura de Integración de Aplicaciones. Asignatura de cuarto año de la rama de Tecnologías de la información del grado de Ingeniería informática.

Las características que comparten este proyecto con la asignatura, como son los intercambios de información y su posterior tratamiento, así como el uso y conocimiento de las interfaces de programación de aplicaciones (API), me resultan interesantes y además me serían de utilidad para la realización del trabajo.

Por tanto, la realización de este proyecto me permitiría ganar conocimientos adicionales y mejorar mis habilidades adquiridas en el ámbito de las Tecnologías de la información.

1.2 Objetivos

Los objetivos del proyecto son los siguientes:

- El objetivo principal es desarrollar un chatbot para la ciudad de Valencia, con el cual un usuario pueda interactuar para localizar ciertos lugares de valor cultural, social o de ocio cerca de él.
- Facilitar la interacción entre el usuario y el bot para mejorar su experiencia con nuestro servicio.

1.3 Estructura de la memoria

Este documento se ha organizado con el siguiente formato:

1. **Introducción**: Capítulo actual que nos presenta el contexto del proyecto, los objetivos y la motivación personal del autor.

2. **Estado del arte**: En este Capítulo se muestra servicios similares al propuesto en este trabajo y analizaremos sus similitudes y diferencias.

3. **Metodología**: En esta sección se comenta la metodología utilizada y se muestra como se ha desarrollado el proyecto.

4. Requisitos: Se muestran todas las tareas que el usuario puede realizar utilizando nuestro servicio.

5. **Arquitectura**: Aquí vemos de manera más técnica como trabaja Telegram y las consecuencias que esto tiene sobre nuestro proyecto.

6. **Diseño del bot**: Diagrama de flujo de todas las posibles interacciones que el usuario puede tener con nuestro bot.

7. **Desarrollo**: Explicación detallada de la implementación final del proyecto, así como de las herramientas utilizadas.

8. Prueba de funcionamiento: Se muestra como ha quedado finalmente el bot y se puede observar cómo ofrece el servicio.

9. **Conclusiones**: Finalmente se muestran las opiniones del autor y se comenta si los objetivos propuestos han sido cumplidos.

2. Estado del arte

En este capítulo vamos a analizar algunos bots que comparten semejanzas en cuanto a el servicio ofrecido por el nuestro, es decir, que muestre lugares en base a la localización del usuario. A su vez explicaremos qué son los datos abiertos y su finalidad en nuestro proyecto.

Este estudio de mercado nos mostrará las características de nuestra competencia y nos permitirá identificar que nos distingue del resto, para tener una mayor oportunidad a la hora de lanzar nuestro producto al mercado.

Para la realización de este estudio se ha utilizado la página web thereisabotforthat.com que como su nombre indica es un buscador de bots, en el cual podemos filtrar por tanto plataformas como por Tags (etiquetas).

2.1 GiveMeFoodBot

GiveMeFoodBot como su nombre indica “Dame comida, bot” es un sencillo programa desarrollado para Telegram que busca restaurantes alrededor de la posición del usuario. Se puede acceder a él desde [15] y podemos observar de manera más detallada su funcionamiento en [13], donde su creadora a puesto a libre disposición su código y documentación. En la actualidad este bot no se encuentra operativo, la última comprobación fue realizada a las 18:30 del 07/08/19.

En la imagen a continuación podemos ver todo el código de este simple bot.

```
require 'telegram/bot'
require 'google_places'

token = 'YOUR-TELEGRAM-BOT-TOKEN'
@googleClient = GooglePlaces::Client.new('YOUR-GOOGLE-PLACES-API-TOKEN')

Telegram::Bot::Client.run(token) do |bot|
  bot.listen do |message|
    if message.location
      l = message.location
      restaurants = @googleClient.spots(l.latitude, l.longitude, :types => ['restaurant','food'], :radius => 500)
      for r in restaurants
        bot.api.send_location(chat_id: message.chat.id, latitude: r.lat, longitude: r.lng)
      end
    end
    case message.text
      when '/food'
        kb = [Telegram::Bot::Types::KeyboardButton.new(text: 'Show me your location', request_location: true)]
        markup = Telegram::Bot::Types::ReplyKeyboardMarkup.new(keyboard: kb)
        bot.api.send_message(chat_id: message.chat.id, text: 'Hey!', reply_markup: markup)
    end
  end
end
```

Figura 1: Código de GiveMeFoodBot.

Fuente: [13]

Como podemos observar hace uso de la API de Google, más específicamente de Google Places, para localizar los restaurantes en un radio definido de 500 metros como podemos observar en la línea 11 de la figura 1.

Como nuestro bot, este consigue nuestra localización tras el uso de un comando, "/food", que habilita un botón que tras su pulsación envía la posición actual del usuario. Finalmente nos envía las localizaciones de cuantos restaurantes haya en ese radio (líneas 12 y 13).

2.2 Around Me

A diferencia del anterior, "Around Me" está desarrollado para Facebook y se puede ver más información sobre el en [16] y [17] donde su creador habla sobre él.

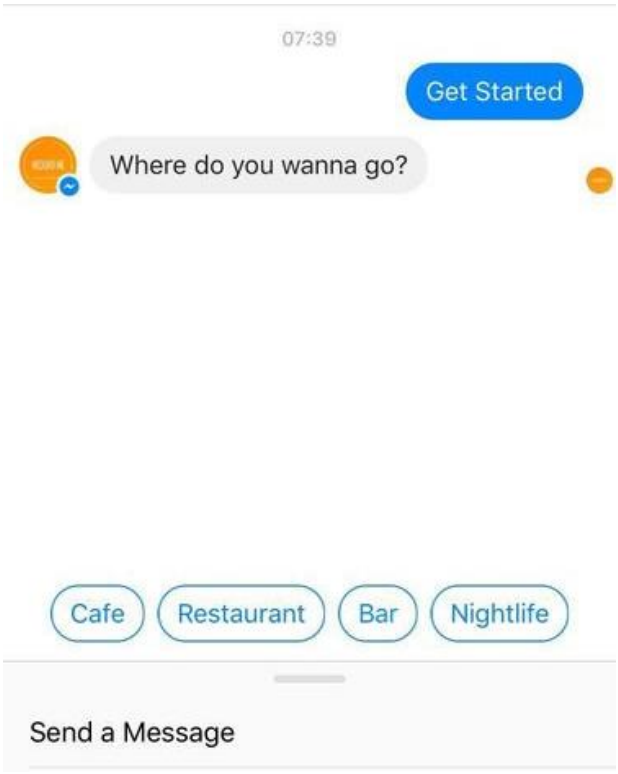

Figura 2: Interacción con around Me.

Fuente: [17]

La principal diferencia con GiveMeFoodBot es que podemos buscar distintas categorías de locales, restaurantes, cafés, bares y locales de ocio nocturno, incrementando la flexibilidad.

Una vez recibimos el mapa de Google con su posición, si pulsamos sobre él botón Get locations, nos carga una ruta hacia nuestro destino.

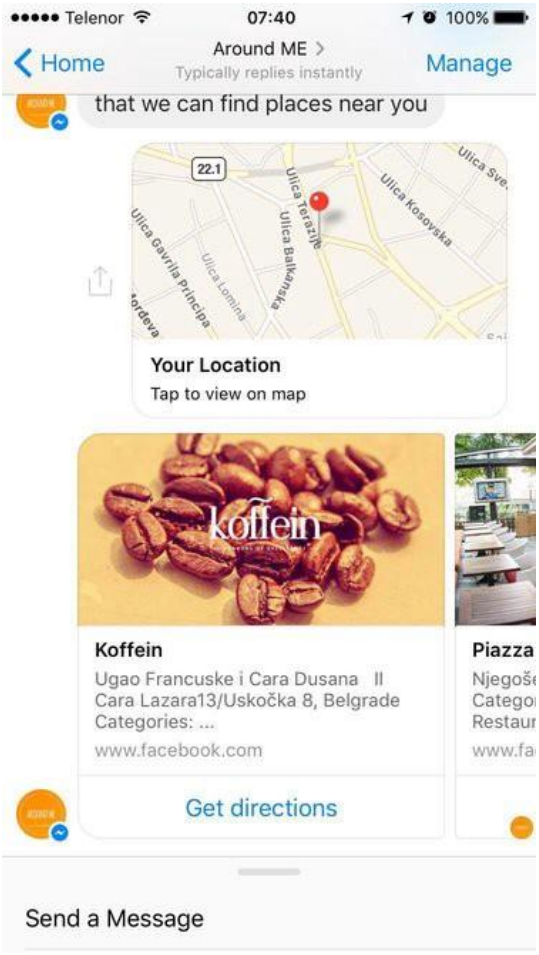

Figura 3: Segunda interacción con around Me.

Fuente: [17]

Podemos ver como este bot está más desarrollado que él mostrado anteriormente, aunque lo hace algo más tedioso de utilizar ya que ahora hay que mantener una conversación algo más larga que la de utilizar o escribir el comando /food.

Este último, se asemeja bastante más a nuestro programa, que el primero, debido tanto a la flexibilidad en cuanto a los lugares que pueden ser mostrados (aunque nosotros aumentamos más este número), como la forma que tiene el usuario para interaccionar con él.

2.3 Otros bots en Telegram

Utilizando la herramienta mencionada anteriormente, filtrando por Telegram, hemos encontrado una serie de bots que ofrecen una funcionalidad similar, mediante la búsqueda, “Finding places”, es decir “Encontrar sitios”. Sin embargo, lo común entre todos ellos es que como los 2 mencionados anteriormente estos se encuentran fuera de servicio.

Entre ellos tenemos:

- MeetPoint: Especializado en encontrar sitios en base a tu posición actual, mediante el uso de un comando /location, que, además permite compartir las localizaciones con amigos.
- Where to eat: Realiza búsquedas de sitios donde comer, al enviar tu posición.
- Fast Loo: Te ayuda a encontrar el baño más cercano enviando tu posición o escribiendo la dirección.
- LocalFindBot: Otro bot de localización, pero orientado a ser utilizado desde otros chats, escribiendo @nombredelbot seguido del nombre del sitio. Ordena la lista de sitios según la distancia a ellos.
- WhereBot y Search Near Places: Ofrecen un servicio de localización de forma similar a MeetPoint y LocalFindBot respectivamente.
- Mica, the Hipster Cat Bot: Este último se especializa en encontrar sitios “hípsters” en tu localización o simplemente en la ciudad que tu decidas. Las conversaciones son mucho más dinámicas que en los ejemplos anteriores, como se puede ver en la figura 4, en la que el usuario, al responder que su día podía ser mejor, recibe una imagen de un gato para intentar animarlo.

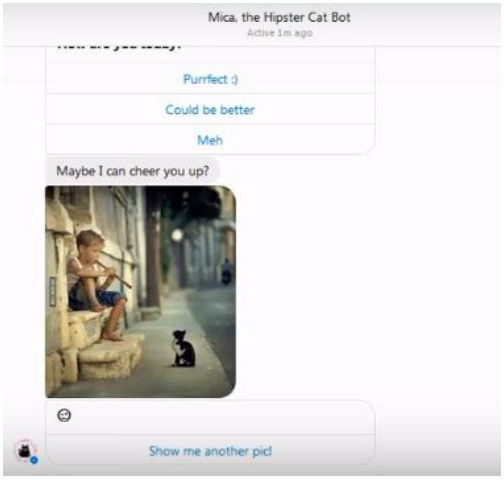

Figura 4: Mica, the Hipster cat Bot.

Fuente: [18]

2.4 Nearby Places Bot

Finalmente hemos buscado en otras plataformas como es Facebook Messenger y hemos encontrado un bot que si se encuentra operativo. Además, tiene una funcionalidad similar a los anteriores y más concretamente al que nosotros vamos a proporcionar.

Este bot comienza como el nuestro, pidiendo la localización del usuario (figura 5) y una vez proporcionada, nos muestra las categorías entre las cuales podemos elegir, 4 como se ve en la figura 6.

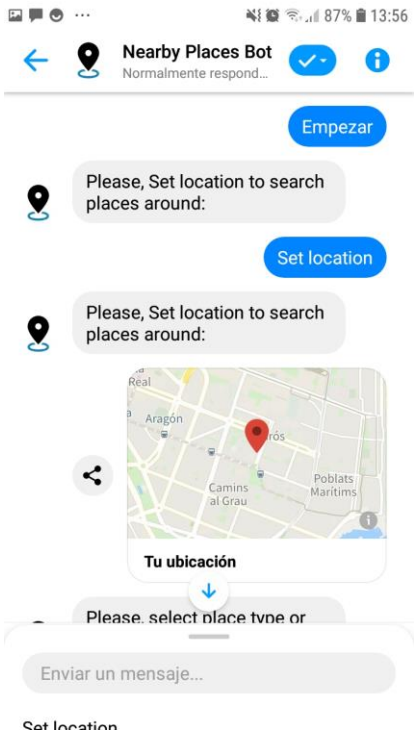

Figura 5: Comienzo de conversación.

Fuente: elaboración propia

Finalmente nos muestra los lugares más cercanos (figura 7), sin dejar al usuario proporcionar un radio de búsqueda, y como podemos observar tras apretar “food”, es decir, que buscamos locales para comer, el más lejano se encuentra solo a 40 metros de distancia y además nos muestra lugares erróneos (figura 8).

En conclusión, aunque la forma de interactuar con el usuario, como se podrá ver más adelante, es bastante similar a la que ofrecemos en nuestro servicio, tanto el número de categorías, como los datos que se muestran al usuario, dejan bastante que desear, y es ahí donde nuestro bot da un paso más allá para satisfacer al usuario en una mayor medida.

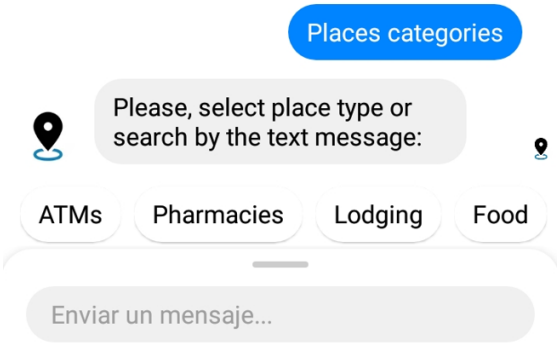

Figura 6: Categorías de Nearby Places Bot.

Fuente: elaboración propia

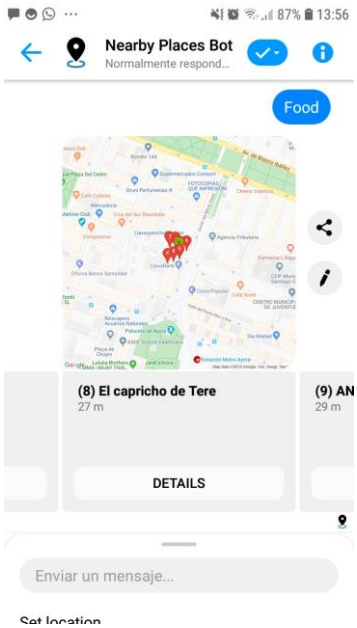

Figura 7: Restaurante cercano.

Fuente: elaboración propia

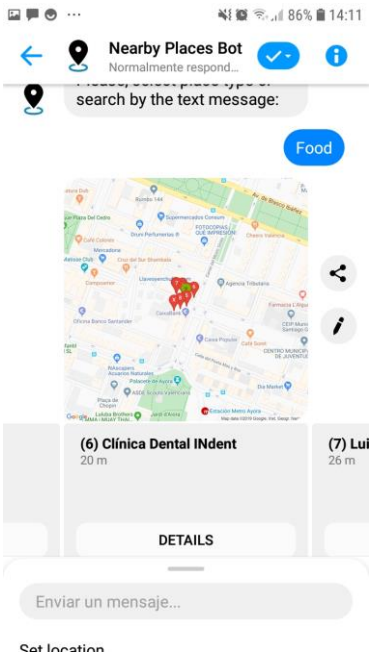

Figura 8: Clínica dental.

Fuente: elaboración propia

2.5 Datos abiertos

En nuestro proyecto hacemos uso de los datos abiertos provenientes del ayuntamiento de Valencia, con el fin de dar un servicio a sus ciudadanos o turistas que puedan estar interesados en descubrir o encontrar determinados lugares. Estos datos se obtienen específicamente de su portal [11] donde podemos encontrar 3 API´s de las cuales obtener distintos tipos de información. En nuestro caso utilizamos la API de datos Georreferenciados [12], la cual nos da acceso a multitud de información, como puede ser: aparcamiento libre más cercano, estaciones de Valenbisi, el tráfico actual, puntos de interés cerca del usuario, qué es lo que usaremos, y más.

Según el Ayuntamiento de Valencia los datos abiertos son una filosofía y práctica que persigue que determinados datos que no están sujetos a restricciones de privacidad, propiedad o seguridad, o aquellos que pudieran contener datos personales que infrinjan la legislación sobre protección de datos personales o datos que su publicación esté regulada de manera administrativa, sean visibles para todo el mundo. Además, los datos han de estar bien estructurados y en formatos conocidos para facilitar su utilización.

La libre disposición de esta información tiene un gran impacto sobre el sector público, permitiendo que personas u organizaciones aporten nuevos datos, conocimientos, mejoren procesos o creen nuevos servicios.

Los datos abiertos disponibles a través de gobiernos o comunidades autónomas son una fuente fiable de información, que incrementa la transparencia y con ello genera confianza y satisfacción en la población.

3. Metodología del proyecto

En este proyecto hemos utilizado la metodología incremental, metodología que se basa en incrementar de forma iterativa una implementación inicial y desarrollarla hasta cumplir nuestros objetivos y requerimientos

Cada incremento suele ser una nueva funcionalidad o una mejor versión de funcionalidades anteriores. De esta manera el producto se finaliza de forma progresiva sabiendo que cada funcionalidad esta implementada correctamente.

En la figura 9 podemos observar un esquema del modelo incremental y como cada incremento se basa en seis fases: análisis, diseño, código, prueba, integración y operación.

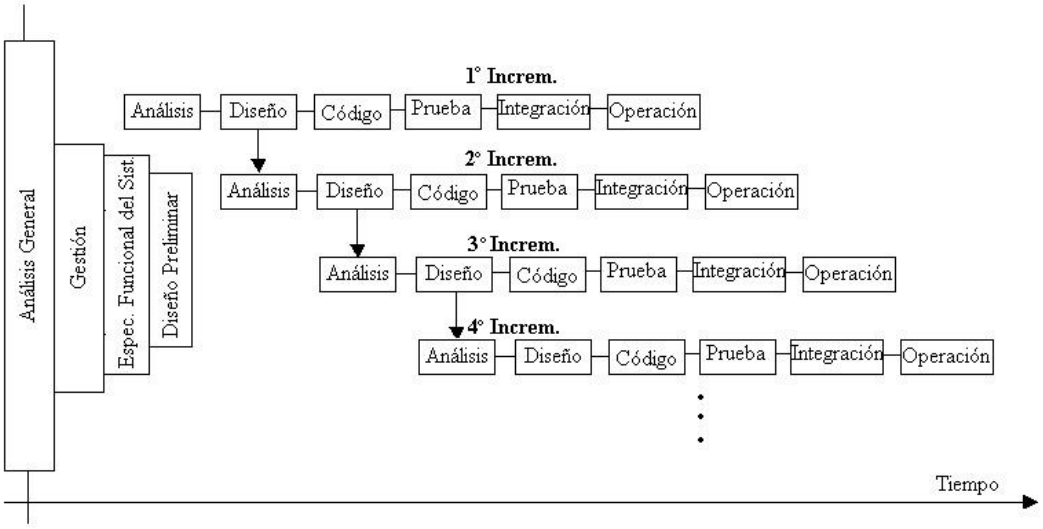

Figura 9: Metodología incremental.

Fuente: [19]

3.1 Análisis general

La realización de este análisis tiene como objetivo especificar cuáles son nuestros requisitos funcionales para la elaboración del diseño iterativo. Para poder listar tales requisitos es primordial la elaboración de la misión del proyecto (sección 1.3) y el estudio de nuestros posibles usuarios.

A la hora de desarrollar un producto o servicio es necesario estudiar quienes son nuestros potenciales usuarios: ¿De dónde provienen?; ¿Qué edades tienen?;

¿Qué lenguaje utilizan?; ¿Qué esperan del producto?, y siendo este de un ámbito tecnológico ¿Qué nivel tienen?

En nuestro caso nos vamos a centrar en los usuarios interesados en conocer más la ciudad de Valencia, España y por tanto los idiomas imprescindibles para la aplicación son el castellano y el valenciano. Además, tras analizar los datos del turismo en la Comunidad Valenciana podemos incluir el inglés debido al significativo número de visitantes de países de habla inglesa y su uso masivo mundialmente. De un Total de 9,208,898 turistas en el año 2018 más de 2,900,000 provienen del Reino Unido [8].

Nuestro producto busca ser sencillo y fácil de usar con lo cual no es necesario ningún nivel tecnológico, sólo disponer de la aplicación de Telegram en un dispositivo móvil para la recogida de datos sobre la localización o acceder a través de su aplicación web, ya que no es posible recoger tales datos desde la aplicación de escritorio.

Nuestros requisitos funcionales son los siguientes:

1. Recogida y manipulación de datos de localización del usuario
2. Recogida de radio de búsqueda
3. Comunicación y tratamiento de datos de la API del Ayuntamiento de Valencia
4. Recogida de datos sobre la categoría de punto de interés y muestreo de ellos
5. Acceso a más información sobre un lugar específico
6. Selección de idiomas
7. Tratamiento de errores

3.2 Etapa iterativa

Como se ha comentado en el apartado anterior, en esta etapa es necesario ordenar las funcionalidades a implementar, normalmente se suele tener en cuenta la importancia que tienen en el proyecto y por ello se empiezan a implementar de mayor a menor trascendencia.

Una de las características de esta metodología es que, en cada iteración, el producto tiene que ser operacional, es decir, que podemos observar cómo interactuaría un usuario con él. Para ello es necesario en la primera iteración desarrollar una base de la aplicación sobre la cual trabajar.

En la tabla 1 que podemos ver a continuación se observa cada iteración con una pequeña descripción de la funcionalidad a desarrollar y un tiempo estimado de su implementación.

Nº	Descripción	Tiempo (horas)
1	Base de la aplicación	40
2	Función de localización y radio de búsqueda	10
3	Comunicación y tratamiento de datos del API	20
4	Función de más información	5
5	Traducción a distintos idiomas	8
6	Tratamiento de errores	5

Tabla 1: Etapas del desarrollo

Primera iteración

Al comienzo es necesario tanto elegir las herramientas con las que desarrollar el bot como diseñar el diagrama de flujo para ver la interacción usuario-bot.

Una vez hemos hecho lo anterior, en la misma iteración construimos la base del bot para que cualquier usuario de Telegram pudiera comunicarse con él.

Segunda iteración

En esta iteración implementamos tanto la función que recoge los datos de la localización del cliente, como la función para determinar el radio de búsqueda, es decir, cuantos metros alrededor de su posición desea observar.

Tercera iteración

Cuando ya tenemos los datos de la localización y el radio de búsqueda, preguntamos al cliente qué tipo de punto de interés quiere buscar, basándonos en las categorías de la API de geolocalización del Ayuntamiento de Valencia [12].

Una vez hemos recogido todos los datos, construimos una petición a la API de geolocalización en la cual incluimos las credenciales. Finalmente tratamos los datos recibidos para mostrárselos en un formato agradable al usuario.

Cuarta iteración

Una vez recibidos todos los lugares que se encuentran en el radio de búsqueda seleccionado, la siguiente función que implementamos sirve para poder preguntar sobre información más detallada de un lugar específico.

Quinta iteración

Para mejorar la experiencia y abordar un mayor número de usuarios hemos traducido la aplicación a distintos idiomas, que son el inglés y el valenciano.

Debido al método empleado a la hora de traducir los idiomas, el añadir más de ellos, solo incrementaría en un par de líneas de código, haciendo que la aplicación en este sentido sea altamente escalable.

Sexta iteración

Finalmente hemos manejado los posibles errores que podrían aparecer, tanto por las entradas de usuario como por mensajes vacíos de la API de geolocalización, mostrando mensajes de error al usuario para que pueda continuar con la utilización del bot.

3.3 Análisis de riesgos

En este proyecto tratamos principalmente con un riesgo de interoperabilidad, ya que hacemos uso de datos que provienen de terceros, en nuestro caso del Ayuntamiento de Valencia. Si el formato o la estructura de los mensajes del API de geolocalización fueran modificados, nuestro servicio es probable que cesara de funcionar hasta realizar las modificaciones necesarias.

Si fuera un cambio de estructura, las variaciones necesarias serían mínimas y por tanto se podría restablecer el servicio a las pocas horas de darse a conocer. En cambio, si fuera un cambio de formato, es decir se utilizará XML en vez de JSON, los cambios a realizar serían algo más extensos y por tanto la aplicación quedaría inutilizable durante al menos varios días.

Por último, existe la posibilidad de que se modificara el URL al cual lanzar nuestra petición y aunque la modificación sería simple de realizar, nuestros servicios quedarían inservibles hasta su arreglo.

Todos estos posibles riesgos tendrían un impacto importante en nuestro servicio y por tanto acarrearía un coste mayor del esperado.

4. Requisitos

Se ha realizado un diagrama de casos de uso (figura 10) para poder mostrar todas las acciones que el actor principal, es decir, el usuario, puede desempeñar al mantener una conversación con nuestro bot.

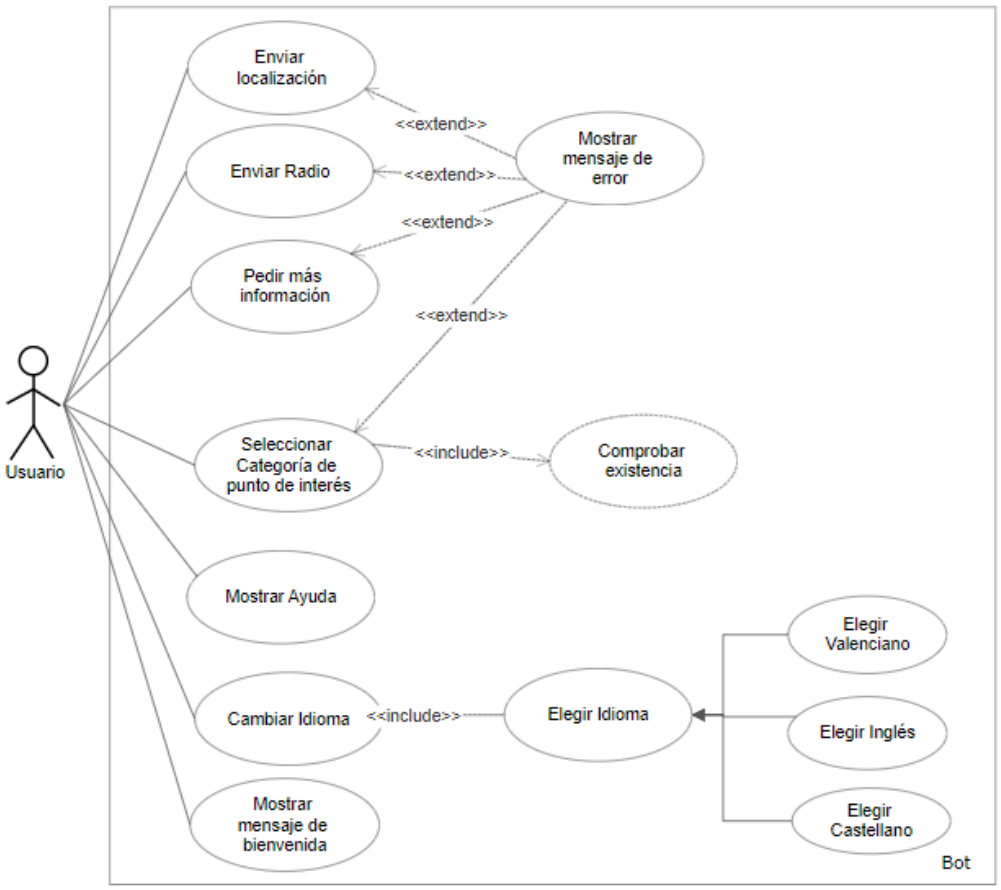

Figura 10: Diagrama de casos de uso

Fuente: elaboración propia

Mostrar mensaje de bienvenida

Al iniciar la conversación, el usuario recibe un mensaje que le comenta los primeros pasos para poder realizar la funcionalidad principal del bot y la posibilidad de cambiar de idioma. Si el usuario quiere, puede volver a leer el mensaje utilizando el comando */start*.

Enviar localización

Es el primer paso para poder encontrar los lugares cercanos, en los cuales el usuario esté interesado. Tras escribir el comando, se habilita un botón y tras su pulsación se envía la posición actual.

Enviar radio

El segundo paso es el de enviar el radio de búsqueda, en cualquier momento después de haber enviado nuestra localización se puede modificar el radio.

Seleccionar categoría de punto de interés

El último paso es el de seleccionar que tipo de lugar deseamos, el usuario recibe una lista de 18 posibilidades con un número asociado, siendo estas:

1. Bibliotecas
2. Centros juveniles
3. Centros sociales
4. Teatros
5. Oficinas de correo
6. Instalaciones deportivas
7. Centros educativos
8. Instalaciones sanitarias
9. Mercados
10. Museos
11. Oficinas municipales
12. Policía
13. Puntos de venta EMT
14. Alojamientos
15. Restaurantes
16. Fuentes
17. Pasarelas
18. Lava-pies

El usuario envía el número con el comando, se comprueba que ese número es correcto y finalmente recibe una lista de lugares de ese tipo seleccionado, en su radio de búsqueda definido.

Pedir más información

El listado de lugares también tiene un número asociado y si el usuario quiere saber más información de un lugar específico, puede utilizar el comando /info y su número, para recibir esta información.

Como el usuario introduce todos estos datos, es necesario realizar comprobaciones, y en caso de que algún dato no se haya enviado correctamente se muestra un mensaje de error especifico para cada situación, como podemos observar en la figura 10.

Mostrar ayuda

Muestra un mensaje con todos los comandos y palabras clave, junto a una pequeña descripción para ayudar al usuario.

Cambiar idioma

El usuario en cualquier momento tiene la posibilidad de cambiar el idioma en el cual el bot conversa con él. Se muestran tres botones por pantalla con los idiomas disponibles.

Elegir idioma

El usuario puede elegir entre tres idiomas: Castellano, inglés y Valenciano, pulsando su respectivo botón.

5. Arquitectura

La arquitectura empleada en este proyecto es un modelo básico de cliente-servidor, donde además tras completar unos requisitos nos conectamos a un servicio proporcionado por el Ayuntamiento de Valencia. Una vez los clientes establecen conexión con nuestro bot, los mensajes intercambiados entre ambos se realizan por medio de peticiones y respuestas HTTP en las cuales se envían objetos de tipo JSON. Sí el programa recibe todos los datos que necesita del usuario, cumpliendo los requisitos, este, envía una petición HTTP a la ubicación donde el Ayuntamiento de Valencia tiene alojado su servicio [20].

En nuestro caso. debido a las limitaciones de Telegram nuestros usuarios no pueden utilizar nuestros servicios desde la aplicación de escritorio como podemos observar en la siguiente imagen, que se traduce a: “Lo sentimos, compartir tu ubicación no está disponible actualmente en la aplicación de escritorio”. Por tanto, deberán usar nuestros servicios desde la aplicación móvil o desde la aplicación web.

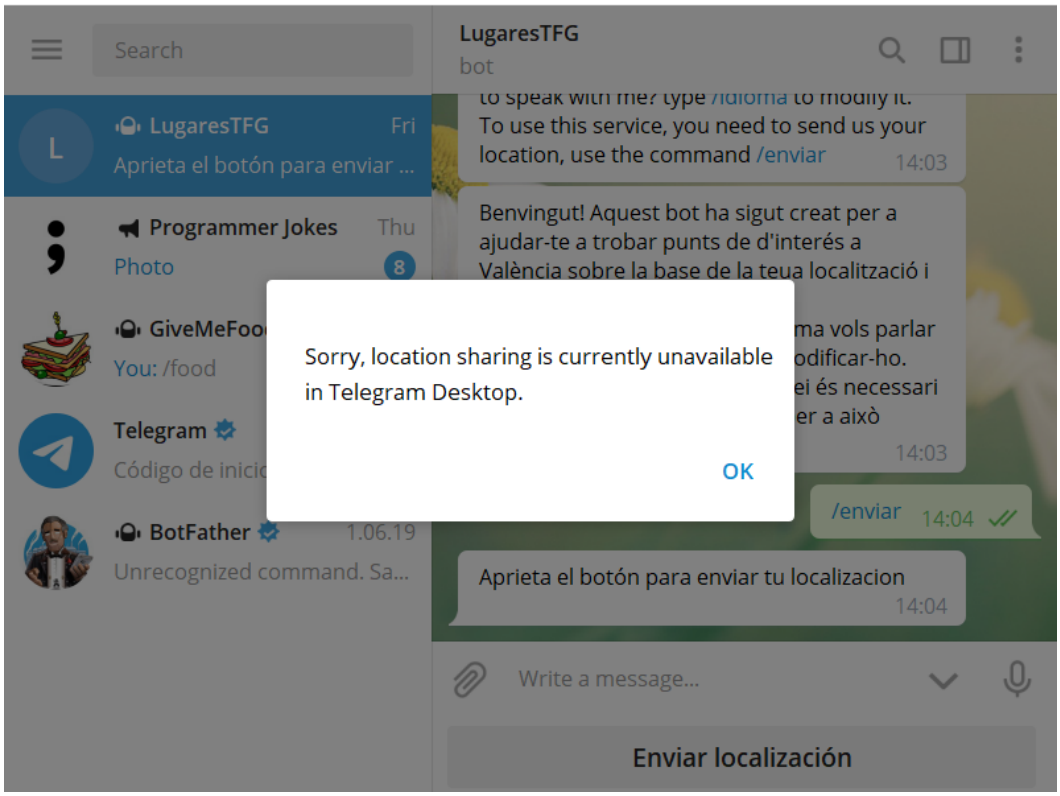

Figura 11: Telegram desktop.

Fuente: elaboración propia

Anteriormente hemos mencionado el protocolo de comunicaciones entre el cliente y el servidor, pero en el siguiente diagrama (ver figura 12) podemos observar de manera más detallada la influencia de todos los elementos en las comunicaciones [14].

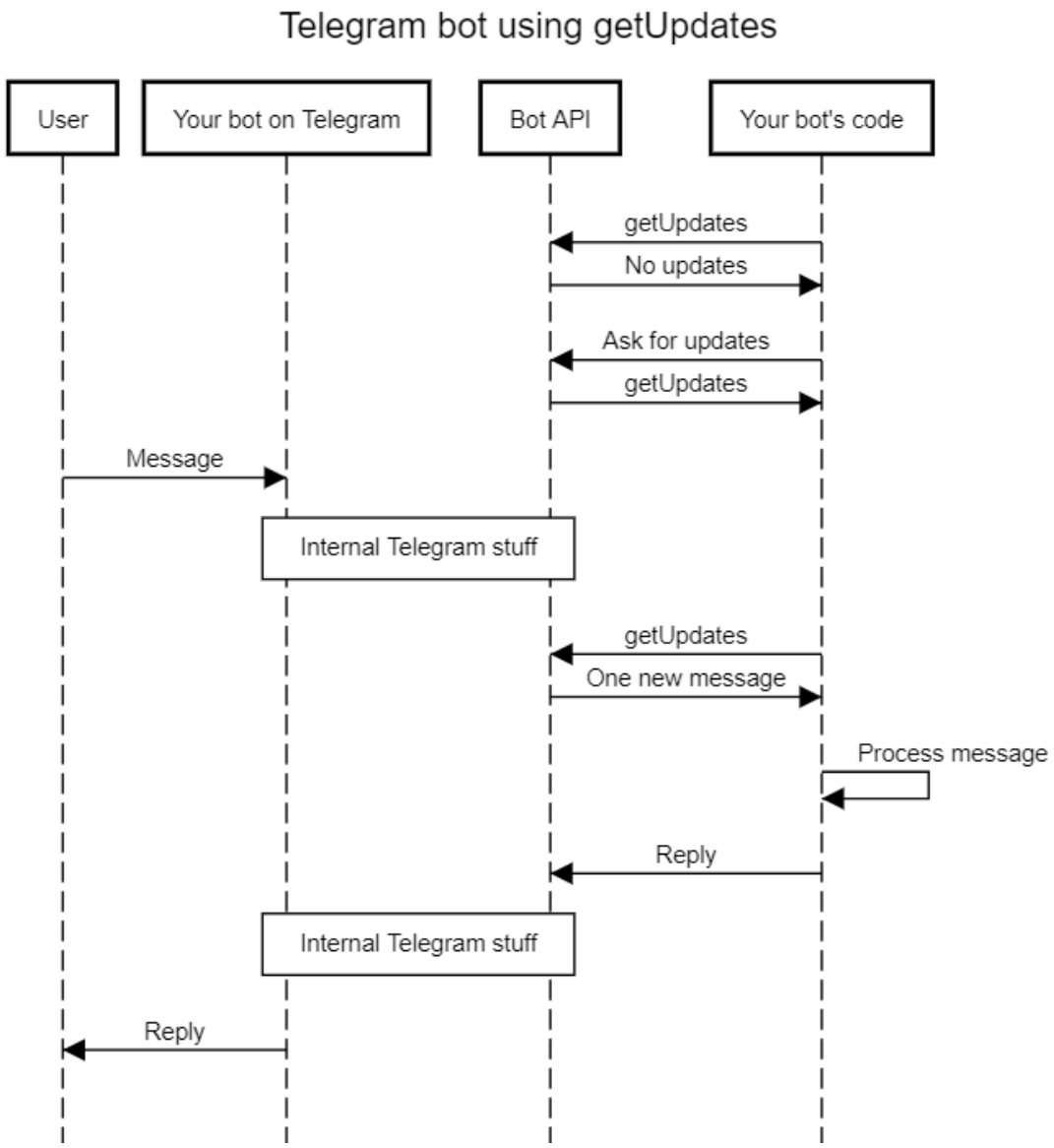

Figura 12: Ejemplo de comunicación cliente-bot.

Fuente: [14]

Los bots pueden recibir mensajes de dos formas distintas, mediante *push o pull* (empujar o tirar), *push* hace referencia a la utilización de lo que se denomina como *webhook* [9] mientras que *pull* es un método de recepción de mensajes en el que constantemente el código del bot pregunta a la Bot API si se ha producido un nuevo mensaje.

En nuestro caso utilizamos el método *pull* y más precisamente una técnica denominada como *long-polling* [7]. Esta técnica intenta minimizar la latencia y los recursos de la red. Esto se consigue porque el servidor en vez de enviar un mensaje vacío si no se produce ningún evento, en lugar de realizar constantes peticiones, mantiene la petición abierta y espera hasta que la información de respuesta esté disponible. Una vez la información está disponible esta se envía por medio de una respuesta HTTP, completando así la petición HTTP que se encontraba abierta. Al terminar se vuelve a abrir otra petición para que la latencia de respuesta (el tiempo transcurrido entre el momento en que la información está disponible por primera vez y la siguiente solicitud del cliente) que ocurre con los métodos normales de *polling* se elimine.

Utilizando el método **getUpdates** del api REST de Telegram que hace referencia a nuestro bot por medio del “Bot Token” (el cual se encuentra tapado), que sirve como autenticación, podemos observar un mensaje realizado por un usuario llamado Héctor (Figura 13).

El mensaje se realiza mediante un objeto JSON serializado, el cual contiene información que nos puede resultar muy útil. El campo **update_id** contiene un identificador único que sirve para ignorar actualizaciones repetidas. Dentro del mensaje se puede encontrar información sobre la persona que lo ha enviado, como su nombre o identificador, la fecha codificada en Unix, el tipo de chat desde donde ha sido enviado (privado, grupo, supergrupo o canal) y el texto escrito en tal mensaje, que se encuentra en el campo **text**.

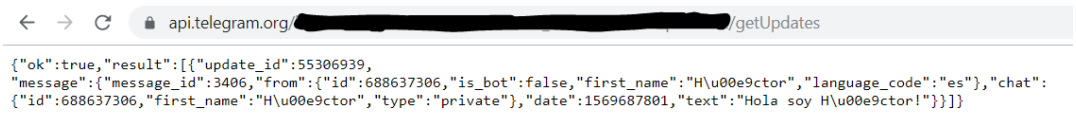

Figura 13: Estructura de un objeto JSON.

Fuente: elaboración propia

6. Diseño del bot

Una vez mostrado cual es el formato de los mensajes, ahora vamos a observar todas las posibles interacciones entre el usuario y el sistema, para ello hemos plasmado todas las posibilidades en un diagrama de flujo.

Primero nos vamos a centrar en si el usuario introduce un comando (Figura 14), si este es el caso la aplicación cuenta con 4 de ellos: ***/start,*** el cual muestra un mensaje de bienvenida en cada uno de los 3 idiomas en los que esta traducida la aplicación; **/idioma**, que nos ofrece la posibilidad de manejar las conversaciones en castellano, inglés o valenciano; ***/help,*** nos muestra una lista de todos los comandos y palabras clave que nos pueden ser de utilidad en caso de necesitar ayuda; y por último **/enviar**, que tras enviar un mensaje de información, nos añade un botón en el teclado de nuestro dispositivo para permitirnos enviar nuestra localización. Si fuera el caso de que ya ha sido usado este comando no se volvería a añadir otro botón.

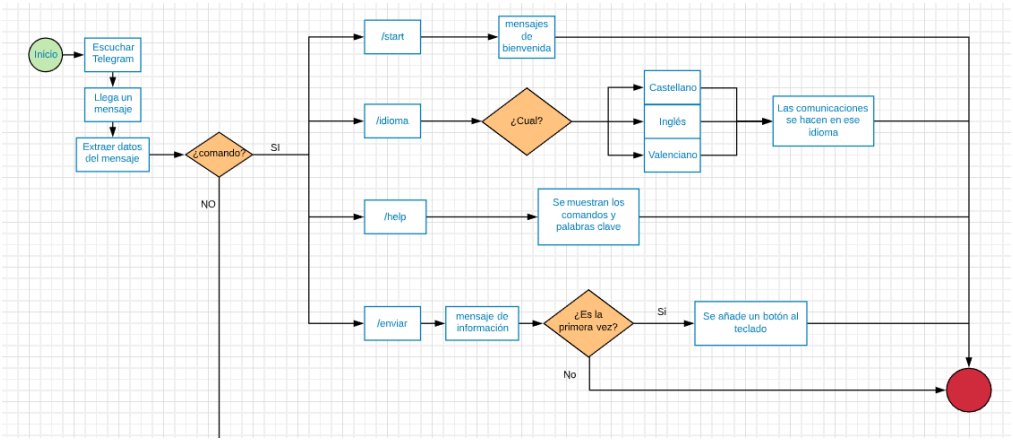

Figura 14: Diagrama de flujo, comandos.

Fuente: elaboración propia

Si no introducimos un comando tenemos otras dos posibilidades, pulsar sobre un botón o introducir un texto que no es un comando (Figura 15). Al pulsar sobre un botón o cambiamos de idioma como se ha mencionado antes o enviamos nuestra localización y el bot nos pide introducir un radio de búsqueda.

Finalmente, tenemos los mensajes de texto que no se incluyen en ninguna categoría, como se puede observar en la figura 15 hay un cierto orden a seguir o recibimos un mensaje de información el cual nos avisa de que hay cierta información que el usuario necesita proporcionar antes de avanzar. La letra “n” que podemos ver después de radio, categoría e info es un número que se introduce en el mismo mensaje, por ejemplo, “radio 500”, que sirve para limitar el radio a 500 metros. Una vez proporcionada la localización, el radio y la categoría, en ese

orden, la palabra clave info y un número que corresponde a un lugar mostrado nos permite recibir información adicional sobre ese lugar, como la distancia, el número de teléfono o su página web, si es que disponen de ellos.

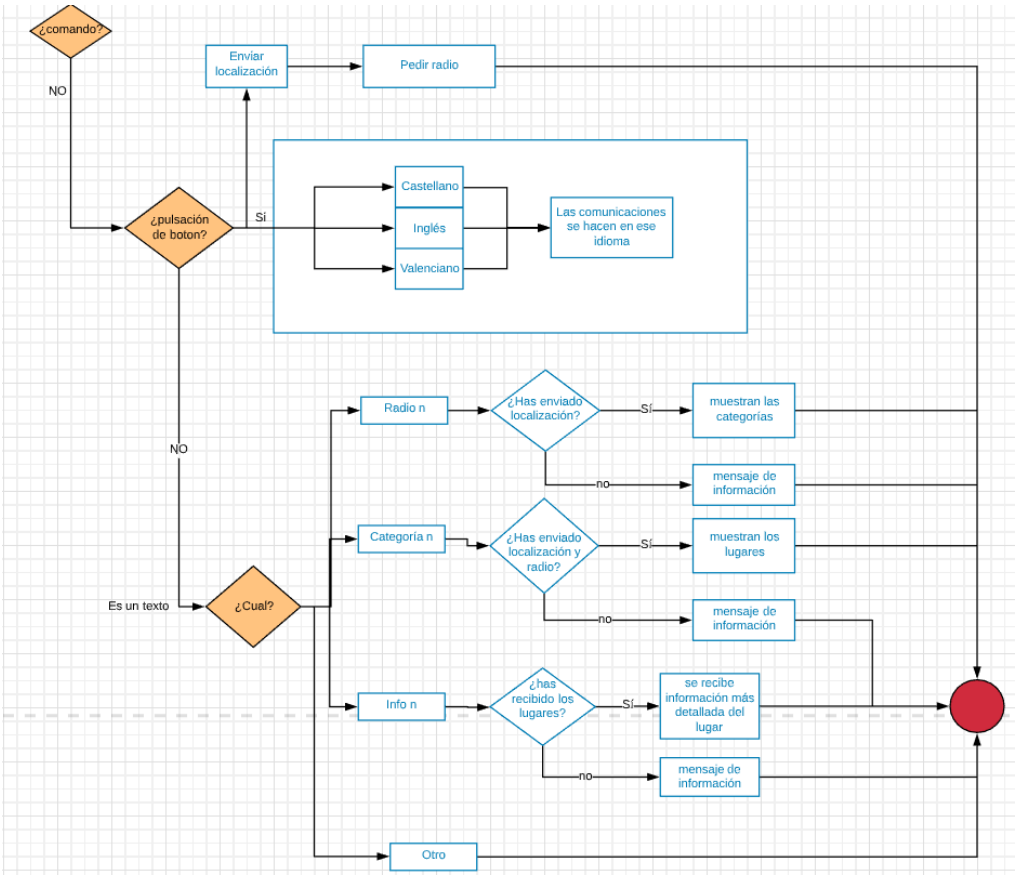

Figura 15: Diagrama de flujo, botones y textos.

Fuente: elaboración propia

7. Desarrollo

Llegados a esta sección, ahora vamos a ver un enfoque más práctico sobre la elaboración del proyecto, por ejemplo, las tecnologías utilizadas, la preparación del entorno y la codificación del bot.

7.1 Tecnologías

Telegram

La aplicación de mensajería instantánea para la cual hemos decidido implementar nuestro servicio.

Node JS

Es nuestro entorno de programación con el cual interpretar código JavaScript, que es el lenguaje por el cual hemos optado para el desarrollo del proyecto.

Telegraf

Un *framework* (que es similar a un API), para crear de forma simple nuestro bot [6].

JSON / Javascript Object Notation

Es un formato de estructuración de datos normalmente usado para transmitir datos entre cliente y servidor.

Visual studio code

Nuestro editor de texto seleccionado, donde hemos escrito todo nuestro código. El tener una terminal integrada nos facilita la ejecución de Node.

7.2 Preparación del entorno

La preparación del entorno es un paso fundamental a la hora de embarcarnos en cualquier desarrollo de software, por tanto, antes de nada, lo primero que necesitamos es instalar NodeJS [4] y crear una carpeta donde trabajar. Accediendo a ella desde la terminal creamos un archivo package.json, mediante el comando *npm init*, que contiene información básica del proyecto así como todas las librerías utilizadas.

```
{
  "name": "bottelegraf",
  "version": "1.0.0",
  "description": "bot tfg",
  "main": "index.js",
  "author": "Héctor Jover Ibáñez",
  "license": "ISC",
  "dependencies": {
    "config.json": "0.0.4",
    "request": "^2.88.0",
    "telegraf": "^3.29.0"
  }
}
```

Figura 16: package.json.

Fuente: elaboración propia

En la misma carpeta procederemos a instalar desde npm [5] todos los módulos que sean necesarios, empezando por Telegraf que es un *framework* para el desarrollo de bots, *request* que nos ayudara a la hora de realizar peticiones HTTP y config.json que facilita el manejo de objetos JSON en archivos de configuración.

```
$ npm install telegraf
```

Figura 17: Instalación de Telegraf

Cuando tenemos nuestro entorno de desarrollo preparado, lo que nos queda es construir nuestro bot, para ello Telegram tiene el padre de todos los bots, el cual es un bot, para ayudarnos con esta tarea [2]. BotFather nos permite mediante una conversación y una serie de comandos, elegir el nombre y la configuración que

queremos para éste, además nos permitirá conseguir el token que será necesario para dotar a nuestro bot de funcionalidad por medio del API de Telegam [3]. Nuestro bot recibe el nombre de InfoVLC, que viene de información en Valencia y además es comprensible para personas de habla inglesa y española.

Accediendo a BotFather desde Telegram e iniciando una conversación, se nos muestran todos los posibles comandos para su configuración (ver figura 18), le damos nombre (figura 19) y nos entrega nuestro token (figura 21). En la figura 20 se puede ver como utilizamos el comando */setname* para modificar el nombre de nuestro bot.

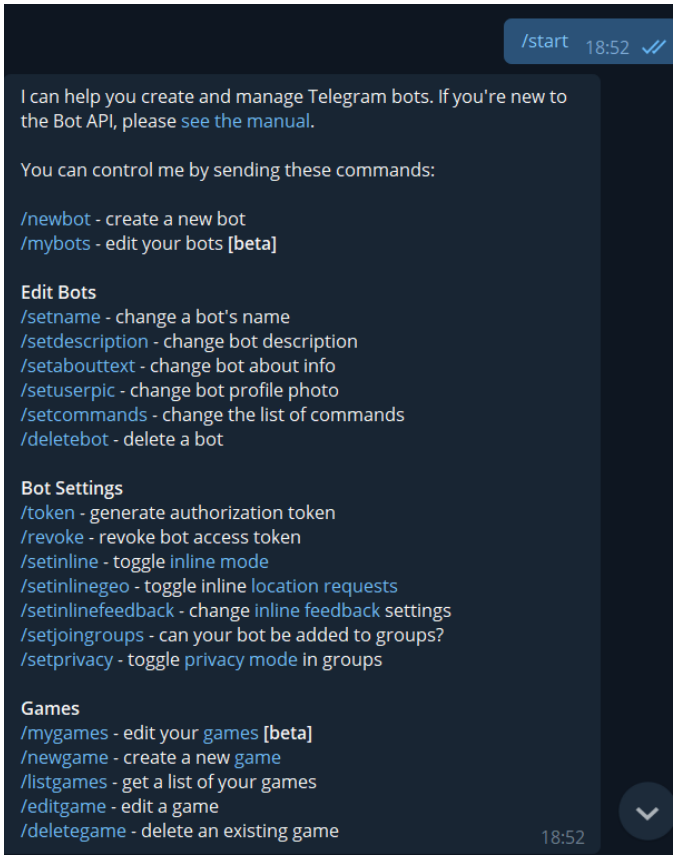

Figura 18: Comandos de BotFather.

Fuente: elaboración propia

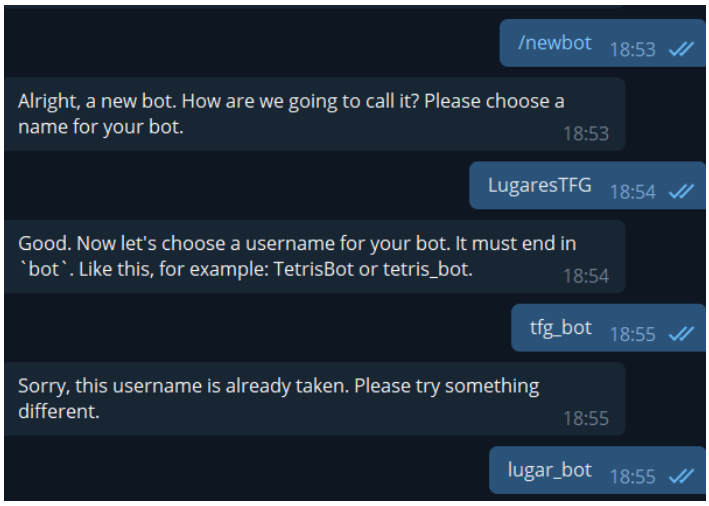

Figura 19: Creación del bot.

Fuente: elaboración propia

Figura 20: Cambio de nombre.

Fuente: elaboración propia

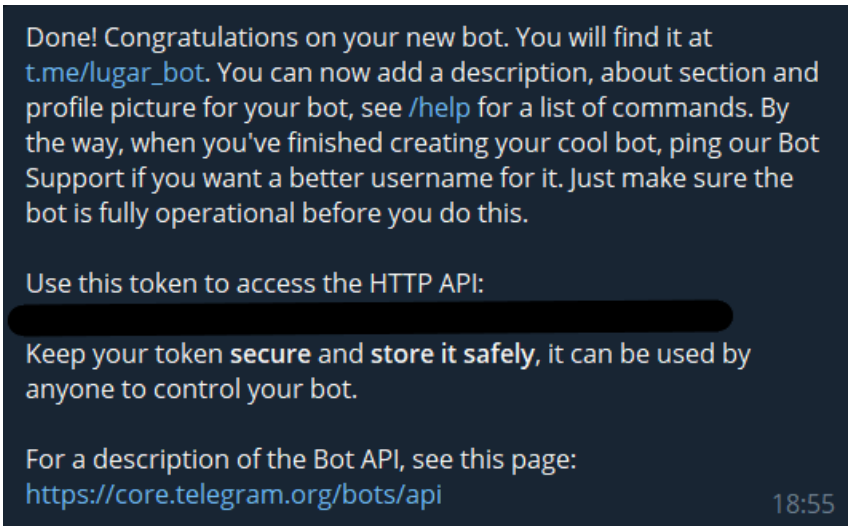

Figura 21: Token del bot.

Fuente: elaboración propia

7.3 Archivo de configuración

Por motivos de seguridad se ha creado un archivo de configuración en el cual se guarda información confidencial como, el token del bot y el usuario y contraseña para hacer uso del API del Ayuntamiento de Valencia, a ellos accederemos remotamente para su utilización.

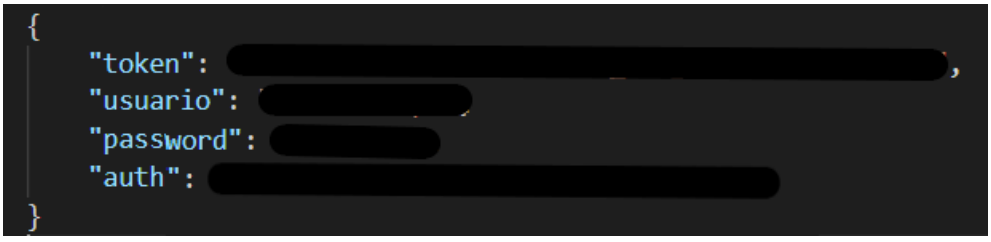

Figura 22: Usuario y contraseñas.

Fuente: elaboración propia

7.4 Variables y constantes

Como punto de partida a la hora de explicar el desarrollo del bot, en la figura 23 se muestran las variables y constantes declaradas, así como las librerías importadas de las cuales haremos uso.

```
const Telegraf = require('telegraf')
const config = require('./config.json')
const request = require('request')
const Markup = require('telegraf/markup')
const Extra = require('telegraf/extra')

const bot = new Telegraf(config.token)

//variable radio de búsqueda
var radio = null;

//variables latitud y longitud
var lat = null;
var lon = null;

//variable de la categoria de punto de interes y array de números que corresponden a una categoria
var categoria = null;
const numbers = [1,2,3,4,5,6,7,8,9,10,11,12,46,50,57,-1,-2,-3];

//variable JSON
var json = null;
var lenjson = null;

//variable idioma
var idioma = 'cas';

//json de textos de distintos idiomas
const id = {
    "cas":['De acuerdo, las comunicaciones serán en castellano','Aprieta el botón para enviar tu localizacion','
    "eng":['Ok, communications will now be in english','Press the button to send your location','Send location',
    "val":['D\'acord, les comunicacions seran en Valencià','Pitja el botó per a enviar la teua localització','En
}
```

Figura 23: Constantes y variables.

Fuente: elaboración propia

Las variables y constantes cuentan con una pequeña descripción de lo que representan, cabe mencionar la constante bot, la cual es una inicialización de un bot de Telegraf al cual debemos pasarle como argumento el token que hemos conseguido por medio de BotFather.

Las demás variables contendrán la información que el usuario nos vaya entregando, como es el radio de búsqueda, su posición (latitud y longitud), la categoría (restaurantes, museos…) y el idioma.

Otras variables:

1) numbers: es un array que contiene los números que representan las distintas categorías que se pueden obtener en la API de datos georreferenciados del Ayuntamiento de Valencia.
2) json y lenjson: son las variables encargadas de contener la información recibida del API antes mencionado, así como su tamaño (el número de objetos).
3) Id: es un objeto JSON con un array de los mensajes para cada idioma en el cual está preparada la aplicación. Si quisiéramos añadir otro idioma y que la aplicación siguiera siendo funcional, solamente habría que añadir otra dupla clave-valor en la cual los textos estuvieran en el mismo orden que los demás y añadir como veremos posteriormente un botón que devuelva el nombre de la clave.

Las librerías tanto Telegraf como request y config.json ya han sido mencionadas anteriormente, la inclusión de markup y extra son necesarias para la implementación de los botones que veremos más adelante.

7.5 Comandos

En el diagrama de flujo hemos visto cual es la secuencia de mensajes en las interacciones cliente-bot y ahora vamos a entrar en más detalle sobre el funcionamiento de los cuatro comandos que están implementados.

/idioma

En la figura 24 tenemos el código referente al comando “idioma”, el cual nos añade un botón para cada idioma, estos botones son del tipo *callbackbutton,* los cuales al ser apretados nos devuelven el valor del segundo argumento.
Una vez creado el array de botones, utilizamos el método inlinekeyboard (que es un teclado que se integra en los mensajes), para poner los botones en el teclado de pantalla y lo enviamos junto a un mensaje de selección de idiomas.

```
//commando para cambiar idioma
bot.command('idioma', (ctx) =>{
    let botidiomas = [
    Markup.callbackButton('Castellano','cas'),
    Markup.callbackButton('Valencià','val'),
    Markup.callbackButton('English','eng')
    ]
    let extra = Markup.inlineKeyboard(botidiomas).extra()
    ctx.reply('Selecciona el idioma / Select the language / Selecciona l\'idioma', extra)
})
```

Figura 24: Comando idioma.

Fuente: elaboración propia

El comando idioma no serviría de nada sin un manejador que esté escuchando a la interacción o evento *callback_query* con uno de los botones (ver figura 25). El objetivo del manejador es recuperar el valor del botón, es decir, 'cas', 'eng' o 'val' y guardarlo en la variable idioma.

La manera en la que utilizamos la variable idioma es muy peculiar, tenemos el objeto JSON id con tres claves "eng", "val" y "cas" las cuales coinciden con los valores devueltos tras apretar los botones referentes a esos idiomas. Con ello al tener cada mensaje traducido en la misma posición de cada array, dependiendo del valor de la variable idioma sacaríamos el mensaje pertinente.
En este caso sería 'De acuerdo, las comunicaciones serán en castellano' o 'Ok, communications will now be in english' o 'D\'acord, les comunicacions seran en Valencià'.

```
//manejador de botones de idiomas
bot.on('callback_query', ctx => {
    ctx.answerCbQuery()
    idioma = ctx.update.callback_query.data
    console.log(idioma)
    ctx.reply(id[idioma][0])
})
```

Figura 25: Manejador de botones de idiomas.

Fuente: elaboración propia

/enviar

Este comando actúa de una forma muy similar al anterior, de forma que una vez enviado nos añade a nuestro teclado, el cual se encuentra debajo de la pantalla (difiere del teclado integrado, el *inline keyboard*), un botón especial denominado *locationRequestButton* que sirve para recoger nuestra latitud y longitud.

```
//------------------Enviar la localización------------------//
bot.command('enviar', (ctx) => {
    var rp = id[idioma][1]
    var bt = id[idioma][2]
    console.log(idioma)
    return ctx.reply(rp, Extra.markup((markup) => {
    return markup.resize()
        .keyboard([
            markup.locationRequestButton(bt)
        ])
    }))

})
```

Figura 26: Comando enviar

Fuente: elaboración propia

Escuchar sobre el evento *'location'* un tipo de *UpdateType*(tipo de actualización) nos permite recibir la posición del usuario. Como se ve en la figura 27, transformamos la latitud y longitud a string para después poder quitar el punto. Esto se debe a que tanto la API de Telegram como la API del Ayuntamiento de Valencia usan distintos formatos en la latitud y longitud, y si no son transformados son inutilizables. Cuando queremos usarlos con la API de Telegram necesitamos los puntos y lo contrario con la del Ayuntamiento de Valencia, por tanto, se verá cómo se vuelven a añadir los puntos más adelante.

```
//handler para la localización
bot.on('location', (ctx) => {
    console.log(ctx.update.message.location['latitude'])
    //toPrecision(8) debido a fallos en la hora de enviar la latitud
    lat = ctx.update.message.location['latitude'].toPrecision(8).toString()
    lon = ctx.update.message.location['longitude'].toString()
    lat = Number(lat.replace('.', ''));
    lon = Number(lon.replace('.', ''));
    console.log(lon)
    console.log(lat)
    ctx.reply(id[idioma][8])
})
```

Figura 27: Manejador del evento *location*.

Fuente: elaboración propia

/start y /help

Estos dos últimos comandos solamente muestran por pantalla mensajes, en el caso de *start* (figura 28) se muestra un mensaje de bienvenida por cada idioma y al ejecutar el comando *help* (Figura 29) recibimos una guía de todos los mensajes y palabras clave que podemos utilizar para interactuar con el bot.

```
//Se inicia el bot y se muestra el mensaje de bienvenida
bot.start((ctx) => {
    ctx.reply(id['cas'][3])
    ctx.reply(id['eng'][3])
    ctx.reply(id['val'][3])
})
```

Figura 28: Comando *Start*.

Fuente: elaboración propia

```
//menu de comandos o palabras clave

bot.help((ctx) => {
    ctx.reply(id[idioma][7])
})
```

Figura 29: Comando *help*

Fuente: elaboración propia

7.6 Manejador de textos y funciones

Este bot además de hacer uso de comandos también interpreta los mensajes de texto mediante el método *hears* (escuchar), que a través de expresiones regulares comprueba si los mensajes de texto siguen uno de los patrones que están definidos.

Radio n

En el caso de la figura 30, esperamos que el usuario nos envié el radio de búsqueda respecto del cual quiere localizar lugares. Éste debe enviarlo escribiendo radio seguido de un número, como se observa en la expresión regular. Se comprueba si anteriormente se ha enviado la localización y después si la variable “r”, es decir, lo que sigue el mismo patrón a la expresión regular, se trata de la palabra radio y de un número entero mayor o igual a 50 (unidad en metros). En caso de que no fuera así, se avisa al usuario de que falta información o del posible error, si todo está correcto entonces se le muestra al usuario las categorías, restaurantes, museos… y se le pide elegir una, escribiendo categoría y su número correspondiente.

```
bot.hears(/[rR]adio (.+)/, ctx => {
    var r = parseInt(ctx.match[0].substring(5))
    console.log(r)
    if(lat != null && lon != null){
        if(typeof(r) == 'number' &&  r > 49 ){
            radio = r;
            ctx.reply(id[idioma][4])
        } else {
            ctx.reply(id[idioma][6])
        }
    } else {
        ctx.reply(id[idioma][5])
    }
})
```

Figura 30: Escuchar radio n.

Fuente: elaboración propia

Categoría n

La parte más importante del desarrollo transcurre cuando el usuario finalmente llega a este punto y escribe que categoría quiere. Hay 18 categorías que puede seleccionar, la API del Ayuntamiento de Valencia los referencia mediante estos números: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 46, 50, 57, -1, -2, -3 siendo cada uno una categoría distinta. Por tanto, para no confundir al usuario simplemente lo hemos enumerado hasta 18 y hacemos un cambio como se puede ver en los condicionales de la figura 31, por ejemplo, si escribe 13, lo modificamos a 46.

Si se sigue el patrón de la expresión regular, comprobamos que la variable “c” sea un número entero y que éste, esté incluido en el array de valores que corresponden con una categoría. Si este es el caso, construimos el URL con el cual acceder al servicio REST (figura 31 y especificado en [12]), que se construye a partir de los datos recibidos del usuario. La dirección del servicio es necesaria para construir la petición que necesitamos lanzar, también necesitamos proporcionar el método, en nuestro caso, *GET*, para recibir información. Las cabeceras que nos permitirán autenticarnos son: *user, pass* y *Authorization* que sus valores están disponibles en el fichero de configuración.

Con la petición construida, la enviamos por medio del método *request* y guardamos la respuesta recibida en la variable *json*. Si el objeto no está vacío, después de esperar un segundo, ejecutamos el método *muestravenues.* Este método es necesario para mostrar en orden los lugares, ya que la función *setTimeout* por sí sola no bloquea la ejecución si lo utilizamos dentro de un bucle.

En la figura 32 podemos observar un ejemplo de respuesta recibida y así podemos entender mejor porque el método *muestravenues* (figura 33) recorre todos los objetos buscando el título y el texto. Como se ha mencionado anteriormente Telegram opera con la latitud y longitud de forma distinta y por tanto tenemos que modificar el formato de las latitudes y longitudes añadiendo un punto en la posición necesaria y ajustando la longitud por medio de *toPrecision(8).* Finalmente utilizamos el método *sendVenues* [3] para mostrar al usuario la localización de cada lugar, asociándole un número (x) en el título del punto de interés para poder identificarlo más fácilmente (1 - ejemplo). Se puede observar cómo jugamos con los tiempos en los *setTimeout* para mostrar los mensajes de forma secuencial y al terminar de enviar todos los lugares, dos segundos más tarde, se envía un mensaje comentando la posibilidad de escribir *info* y su número asociado para recibir información más específica.

```
//Escuchar la categoria

bot.hears(/[cC]ategor[ii]a (.+)/, ctx => {
    if(radio != null && lat != null){
        var c = parseInt(ctx.match[0].substring(9))
        if(c == 13){
            c = 46
        }
        if(c == 14){
            c = 50
        }
        if(c == 15){
            c = 57
        }
        if(c == 16){
            c = -1
        }
        if(c == 17){
            c = -2
        }
        if(c == 18){
            c = -3
        }
        if(typeof(c) == 'number' && numbers.includes(c)){
            categoria = c
            console.log(categoria)
            var http = 'http://mapas.valencia.es/lanzadera/puntoInteres/infociudad?radio='
            http += radio +'&lang=es&lat=' + lat +'&lon=' + lon + '&filtros='+ categoria;
            console.log(http)
            var options= {
                url: http,
                method: 'GET',
                headers: {
                    'user':config.usuario,
                    'pass':config.password,
                    'Authorization':config.auth
                }

            }
            request(options, function(err, res, body){
                json = JSON.parse(body);
                //console.log(json)
                lenjson = json.length
                if(isEmpty(json)){
                   ctx.reply(id[idioma][9])
                } else {
                    ctx.reply(id[idioma][10])
                    setTimeout(function() {
                        muestravenues(json,ctx);
                    },1000)
                    setTimeout(function(){
                        ctx.reply(id[idioma][11])
                    },1000 * lenjson + 2000)
                }
            })
        }
        else{
            ctx.reply(id[idioma][12])
        }
    }
    else {
        ctx.reply(id[idioma][13])
    }
    return json
})
```

Figura 31: Manejador de categoría.

Fuente: elaboración propia

```
[{"titulo":"BIBLIOTECA MUNICIPAL VICENT CASP I VERGER","texto":"C/ PERIS BRELL, 6\nAyuntamiento de
Valencia","imagen":"http://mapas.valencia.es/WebsMunicipales/layar/img/infociudad_1.png","lat":39465062,"lon
":-347405,"distancia":635,"tipo":1,"acciones":[{"nombre":"Más
información","tipo":"text/html","uri":"https://www.valencia.es/ayuntamiento/infociudad_accesible.nsf/vBuscar
Callejerovalen/id000027?OpenDocument&lang=1"},{"nombre":"Llamar","tipo":"tel","uri":"+34963301473"}]},
{"titulo":"BIBLIOTECA MUNICIPAL TOMÁS VICENT TOSCA","texto":"\nAyuntamiento de
Valencia","imagen":"http://mapas.valencia.es/WebsMunicipales/layar/img/infociudad_1.png","lat":39472278,"lon
":-350937,"distancia":665,"tipo":1,"acciones":[{"nombre":"Más
información","tipo":"text/html","uri":"https://www.valencia.es/ayuntamiento/infociudad_accesible.nsf/vBuscar
Callejerovalen/id000016?OpenDocument&lang=1"},{"nombre":"Llamar","tipo":"tel","uri":"+34963525478"}]},
{"titulo":"BIBLIOTECA PÚBLICA MUNICIPAL MARÍA MOLINER","texto":"C/ SERPIS, 9 y 11\nAyuntamiento de
Valencia","imagen":"http://mapas.valencia.es/WebsMunicipales/layar/img/infociudad_1.png","lat":39476254,"lon
":-349078,"distancia":827,"tipo":1,"acciones":[{"nombre":"Más
información","tipo":"text/html","uri":"https://www.valencia.es/ayuntamiento/infociudad_accesible.nsf/vBuscar
Callejerovalen/id000009?OpenDocument&lang=1"},{"nombre":"Llamar","tipo":"tel","uri":"+34963931333"}]},
{"titulo":"BIBLIOTECA MUNICIPAL GREGORI MAYANS I SÍSCAR","texto":"C/ TRAFALGAR, 34\nAyuntamiento de
Valencia","imagen":"http://mapas.valencia.es/WebsMunicipales/layar/img/infociudad_1.png","lat":39461391,"lon
":-345096,"distancia":967,"tipo":1,"acciones":[{"nombre":"Más
información","tipo":"text/html","uri":"https://www.valencia.es/ayuntamiento/infociudad_accesible.nsf/vBuscar
Callejerovalen/id000014?OpenDocument&lang=1"},{"nombre":"Llamar","tipo":"tel","uri":"+34963525478"}]}]
```


Figura 32: Respuesta JSON del API del Ayuntamiento de Valencia.

Fuente: elaboración propia

```
//función para comprobar si está vacio un objeto
function isEmpty(obj){
    return !Object.keys(obj).length;
}
//funcion necesaria para mostrar en ORDEN los lugares
function muestravenues(json,ctx){
    var x = 0
    for(x ; x < json.length; x++){
    //for(x in json){
        setTimeout(function(x){
            console.log(json[x].titulo)
            var tit = json[x].titulo;
            var txt = json[x].texto;
            console.log(json[x].lat)
            var xlat = Number(json[x].lat.toString().substring(0,2)+'.'+json[x].lat.toString().substring(2)).toPrecision(8)
            console.log(xlat)
            console.log(json[x].lon)
            var xlon = Number(json[x].lon.toString().substring(0,1)+'.'+json[x].lon.toString().substring(1))
            console.log(xlon)
            bot.telegram.sendVenue(ctx.from.id,xlat,xlon, x +' - '+tit,txt)
        },1000 * x, x);

    }
}
```

Figura 33: Funciones *isEmpty* y *muestravenues*.

Fuente: elaboración propia

Info n

Finalmente tenemos la última funcionalidad implementada, en ella escuchamos, como en otras ocasiones, a la palabra clave info y un número que corresponda con un lugar mostrado anteriormente. Si el número coincide, recorremos el objeto en concreto, es decir “n”, creando un mensaje con información que podría resultar interesante, como es, la distancia, el teléfono y la página web del punto de interés. Debido a que no todos los lugares tienen esta información, por ejemplo, una fuente, que no dispondrá de un número de contacto, antes de añadir un mensaje vacío comprobamos que la información si esté disponible.

```
//escuchar info n para devolver más información sobre n
bot.hears(/[iI]nfo (.+)/, ctx =>{
    var mensaje = '';
    if(json != null){
        var ncat = Number(ctx.match[0].substring(4))
        if(ncat >= 0 && ncat < lenjson){
            for (var k in json[ncat]){
                if(k == 'texto' || k == 'titulo'){mensaje += json[ncat][k]+'\n'}
                if(k == 'distancia'){
                    mensaje += id[idioma][14] + json[ncat][k]+ id[idioma][15]+'\n'
                }
            }

            if(json[ncat]['acciones'] != null){
                    for(var i in json[ncat]['acciones']){
                        console.log(json[ncat]['acciones'].length)
                        if(json[ncat]['acciones'][i]['tipo'] == 'tel'){
                            mensaje += id[idioma][16] + json[ncat]['acciones'][i]['uri'] + '\n'
                        }
                        if(json[ncat]['acciones'][i]['tipo'] == 'text/html'){
                            mensaje += id[idioma][17] + json[ncat]['acciones'][i]['uri'] +'\n'
                        }
                    }
            }
            ctx.reply(mensaje)

        }else{
            ctx.reply(id[idioma][18])
        }
    } else{
        ctx.reply(id[idioma][19])
    }
})
```

Figura 34: Manejador de *info n*.

Fuente: elaboración propia

7.7 Mensajes de error y lanzamiento

A lo largo de todo el código, en cada funcionalidad, se puede observar como el código está preparado para cualquier mensaje por parte del usuario. Así evitamos que nuestro servicio deje de funcionar y además podemos avisar de los fallos, intencionados o no, que se han producido.

Es especialmente necesario preparar estos mensajes en las secciones en las cuales se escucha determinadas palabras, ya que el usuario podría o no, continuar con un número, como en principio se podría esperar.

Para terminar, comentar brevemente el método de la figura 35 el cuál sirve para lanzar la aplicación en modo *long-polling*, como se comentó en el capítulo cuatro.

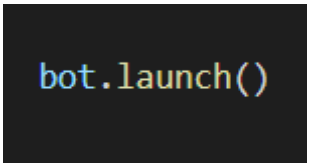

Figura 35: Lanzamiento(*polling*)

Fuente: elaboración propia

8. Prueba de funcionamiento

8.1 Despliegue

Para iniciar los servicios de nuestro bot y probar su funcionamiento, accedemos desde la terminal de comandos a la carpeta donde se ha estado trabajando y ejecutamos el comando *node index.js* (nombre del fichero .js). Una vez ejecutado, nuestro bot estará inicializado y por tanto ya podemos comunicarnos con él a través de Telegram.

8.2 Pruebas

Se han realizado pruebas de funcionamiento para demostrar que el bot funciona como debe. Iniciamos la conversación con */start* y recibimos tres mensajes de bienvenida con instrucciones de uso. Para no realizar tres imágenes del mismo comando, la figura 36 ha sido realizada a través de la aplicación de escritorio mientras que las demás han sido tomadas desde la aplicación móvil.

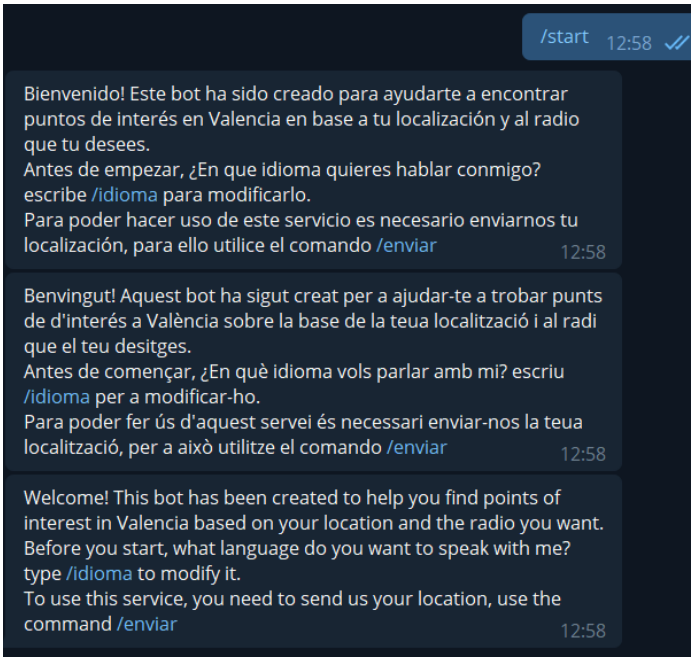

Figura 36: Prueba */start*.

Fuente: elaboración propia

Al escribir /idioma y pulsar sobre uno de los botones podemos observar cómo se cambian las comunicaciones, en la figura 37 se ha pulsado sobre inglés y después sobre castellano.

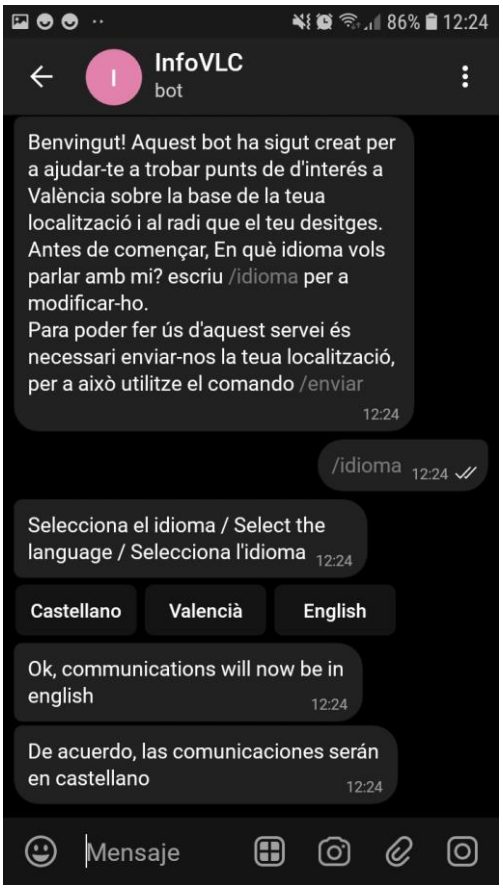

Figura 37: Prueba */idioma*.

Fuente: elaboración propia

Cuando hemos decidido que idioma utilizar tenemos que usar el comando /enviar como se comentaba en el mensaje de bienvenida. Una vez enviado, en la parte de debajo de nuestra pantalla se observa el botón que hay que pulsar para enviar nuestra localización (Es necesario darle a Telegram permisos para usar la localización). Inmediatamente después, si todo está correcto, se pregunta al usuario por el radio en el cual quiere buscar.

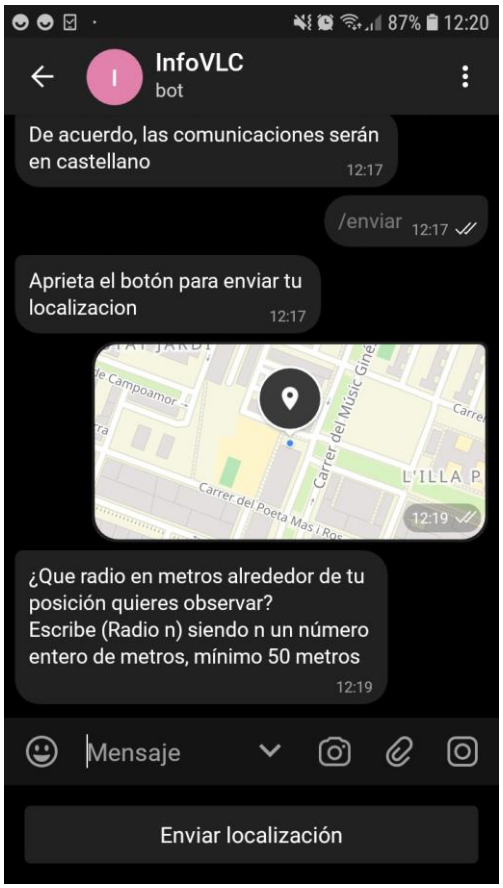

Figura 38: Prueba */enviar*

Fuente: elaboración propia

Escribimos radio 1000 y a diferencia de la figura 40 donde escribimos “radio patata” como todo está correcto (ver figura 39), se nos muestran todas las categorías.

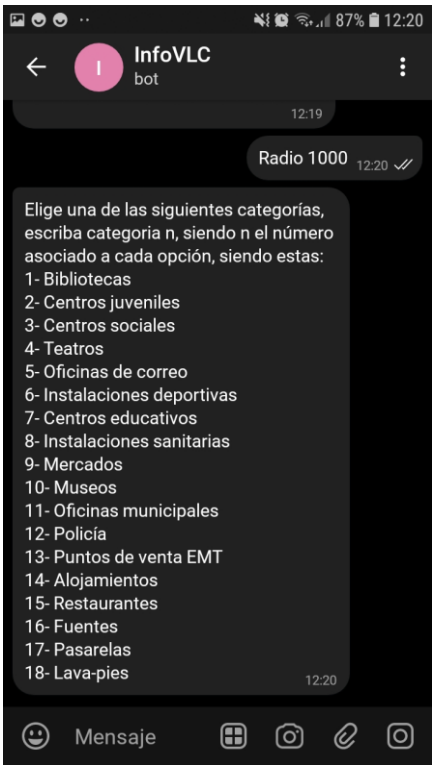

Figura 39: Prueba radio 1000

Fuente: elaboración propia

Figura 40: Error en el radio.

Fuente: elaboración propia

Elegimos una categoría y si todos los datos se han pasado correctamente, se nos muestran todos los lugares que corresponden a esa categoría, en el radio de búsqueda definido (ver figuras 41 y 42).

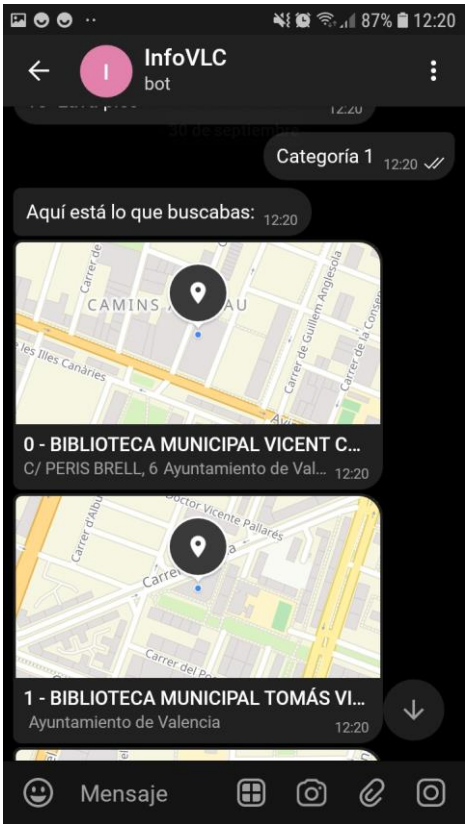

Figura 41: Prueba categoría 1, parte 1.

Fuente: elaboración propia

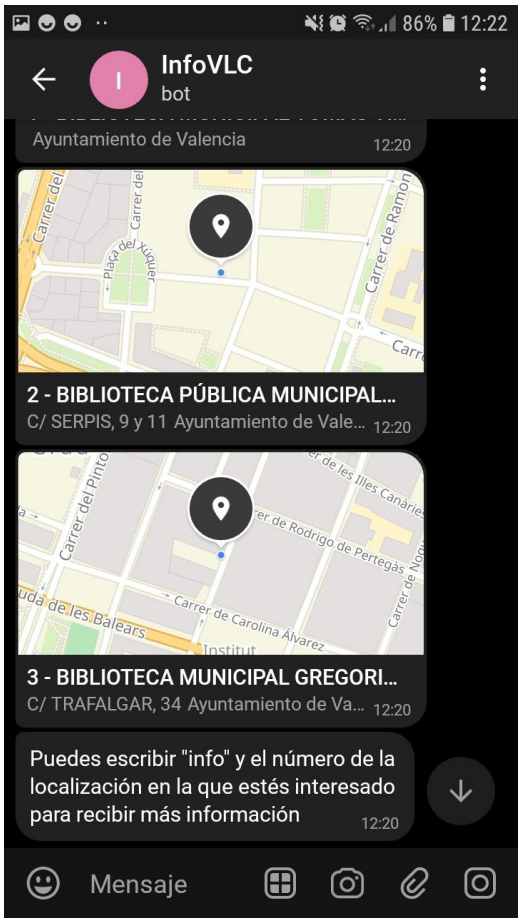

Figura 42: Prueba categoría 1, parte 2.

Fuente: elaboración propia

Finalmente, como se muestra al final de la figura 42 tenemos la opción de preguntar por más información. A continuación, escribimos Info 3 y
recibimos un mensaje con el nombre del lugar, la dirección, la distancia, su página web y su número de teléfono (Figura 43).

Figura 43: Prueba info 3.

Fuente: elaboración propia

9. Conclusiones

Los chatbots, así como cualquier bot, es una tecnología que ahora mismo se encuentra en alza, con la mejora constante en el área de la inteligencia artificial y el uso masivo de aplicaciones móviles, probablemente, cada vez sea más común interactuar con un bot.

Nuestro bot InfoVLC entra en esta era tecnológica como un servicio de localización totalmente a mano y fácil de usar, cumpliendo los objetivos que se determinaron antes de su desarrollo.

De forma más personal, la idea de seleccionar este trabajo de final de grado surge después de cursar la asignatura, Integración de aplicaciones de cuarto año del grado de Ingeniería informática y aunque los conocimientos estudiados en la carrera no son exactamente los necesarios para desarrollar un proyecto de estas características, si cubren lo esencial, por ejemplo, el lenguaje de programación utilizado, JavaScript, que se cubre en alto modo en la rama de Tecnologías de la información.

Mediante la realización de este proyecto he adquirido muchos conocimientos técnicos, tanto por las tecnologías utilizadas, como por la labor informativa necesaria para la elaboración de este documento.

9.1 Posibles mejoras

Como con cualquier proyecto o trabajo, existen limitaciones, sean presupuestarias o temporales, que limitan el alcance o las funcionalidades del proyecto.

En este proyecto se ha trabajado de una manera totalmente local y si queremos desplegar el servicio debemos de hacerlo de forma manual y consumir nuestros propios recursos. Por tanto, una mejora sería utilizar cualquier plataforma de despliegue como puede ser en la nube, que, aunque no son gratuitas, mejorarían la disponibilidad de nuestro servicio, por ejemplo, podríamos utilizar, *Amazon Web Services, Microsoft Azure, Google Cloud.*

Otra mejora que considerar es la expansión de nuestro servicio, es decir, llegar a más ciudades. Esto se podría realizar mediante el uso del API de Google llamado Places, aunque acarrearía grandes cambios en la aplicación, esta posibilidad se comentó con el tutor y se decidió por utilizar los datos abiertos del Ayuntamiento de Valencia.

Finalmente se podría realizar algunos cambios para evitar el uso del teclado y facilitar así las interacciones del usuario, mediante la inclusión de botones, aunque esto limitaría la flexibilidad existente. Cuestión que se podría considerar después de hacer un estudio y analizar el interés por el cambio.

10. Bibliografía

[1] Página oficial de Telegram. URL: https://telegram.org/

[2] Introducción a los Bots de Telegram. URL: https://core.telegram.org/bots

[3] Bot API de Telegram. URL: https://core.telegram.org/bots/api

[4] Página oficial de NodeJs. URL: https://nodejs.org/es/

[5] Página oficial de Npm. URL: https://www.npmjs.com/

[6] Documentación oficial de Telegraf. URL: https://telegraf.js.org/#/

[7] RFC 6202 - Known Issues and Best Practice for the Use of Long Polling and Streaming in Bidirectional HTTP. URL: https://tools.ietf.org/html/rfc6202#section-2.1

[8] Balance turístico en la comunidad Valencia 2018. URL:http://www.turisme.gva.es/turisme/es/files/pdf/estadistiquesdeturisme/anuarios/Balance_anyo%202018c.pdf

[9] Prashant Ram, “What is a webhook?” URL: https://codeburst.io/what-are-webhooks-b04ec2bf9ca2

[10] Node.js tutorial in Visual Studio Code. URL:https://code.visualstudio.com/docs/nodejs/nodejs-tutorial

[11] API´S del Ayuntamiento de Valencia. URL:http://gobiernoabierto.valencia.es/en/info-api/

[12] Documento del API de geolocalización.
URL:http://www.valencia.es/ayuntamiento/DatosAbiertos.nsf/0/2113BD9D1693D7EAC1257C6600449981/$FILE/API%20APPCIUDAD%20v3.pdf?OpenElement&lang=1

[13] GiveMeFoodBot, información y desarrollo.
URL:https://github.com/barbaramartina/ruby-telegram-bot

[14] Introduction to the Telegram Bot API, Part 1 por Jiayu Yi.
URL:https://chatbotslife.com/introduction-to-the-telegram-bot-api-part-1-2ae36f7b30a4

[15] Acceso al bot de Telegram GiveMeFoodBot (Inoperativo).
URL:https://telegram.me/GiveMeFoodBot

[16] AroundMe bot en Facebook (Inoperativo).
URL:https://www.facebook.com/aroundme.bot/

[17] Chatbotmagazine. URL: https://chatbotsmagazine.com/messenger-bot-called-around-me-that-shows-you-cool-places-around-you-wherever-you-are-efca185b8ba8

[18] Mica, the Hipster Cat Bot. URL: https://hipstercatbot.com/

[19] Archivo de Wikipedia, modelo incremental.
URL:https://es.wikipedia.org/wiki/Archivo:Modelo_Iterativo_Incremental.jpg

[20] Ubicación del servicio del API de datos georreferenciados.
URL:http://mapas.valencia.es/lanzadera

[21] Instant Node.js starter [electronic resource]: program your scalable network applications and web services with Node.js. Por, Pedro Teixeira.
URL: https://ebookcentral.proquest.com/lib/bibliotecaupves-ebooks/detail.action?docID=1214993

[22] Practical Bot Development [electronic resource]: Designing and Building Bots with Node.js and Microsoft Bot Framework. Por, Szymon Rozga.
URL: https://www.oreilly.com/library/view/practical-bot-development/9781484235409/?ar