Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

Desarrollo de un asistente virtual conversacional proactivo basado en Dialogflow

Development of a proactive conversational virtual assistant based on Dialogflow .

Eduardo Borges Fernández

La Laguna, 30 de mayo de 2018

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y Profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Rosa María Aguilar Chinea**, con N.I.F. 43.778.956-C Catedrátrica de Ingeniería de sistemas y automática adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Desarrollo de un asistente virtual conversacional proactivo basado en Dialogflow.*"

ha sido realizada bajo su dirección por D. **Eduardo Borges Fernández**, con N.I.F. 78.858.652-R

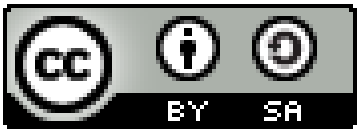
Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de mayo de 2018

Agradecimientos

A mi tutor, Jesús Miguel Torres Jorge y a mi cotutora Rosa María Aguilar Chinea por ayudarme, por su continua disposición y por su actitud positiva.

A mi familia, amigos y pareja por apoyarme y aportarme estabilidad emocional.

Licencia



Resumen

El objetivo de éste proyecto ha sido la investigación de la tecnología de procesamiento de lenguaje natural DialogFlow, el funcionamiento de la API de Google Places, de Telegram, de Discord y del uso de una base de datos MongoDB.

Todo esto con el fin de crear un agente virtual conversacional proactivo sobre turismo en Tenerife, llamado TenTurismo, que entienda el lenguaje natural y que sea capaz de aportar información de lugares por cercanía y por búsqueda de palabras clave a través de canales como Telegram y Discord.

Palabras clave: Turismo, Tenerife, Chatbots, Agentes conversacionales, Dialogflow, Telegram, Discord, TenTenerife

Abstract

The objective of this project has been the investigation of the natural language processing technology DialogFlow, the functioning of the Google Places API, the Telegram API, the Discord API and the use of a MongoDB database.

All this to create a proactive conversational virtual agent of tourism in Tenerife, called TenTurismo, that understands natural language and is abled to provide information of nearby places and look for them by keywords too, through channels such as Telegram and Discord.

Keywords: *Conversational agent, Turism, Tenerife, Chatbots, Dialogflow, Telegram, Discord, TenTenerife*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Los agentes virtuales conversacionales, también conocidos como chatbots, son programas informáticos que simulan el comportamiento de una conversación humana, son actualmente un tema candente en el mundo de la Inteligencia Artificial. Su principal objetivo es conseguir que el usuario pueda realizar acciones sobre un sistema sin necesidad de que exista un intermediario humano, algunos ejemplos pueden ser: pedir una pizza, solicitar información sobre un envío, hacer transacciones bancarias o cualquier aplicación en un campo específico.

1.1. Planteamiento del problema

Tenerife es conocido, junto con las otras Islas Canarias, por su gran afluencia de turismo, la mayoría de turistas que llegan suelen haber pagado con anterioridad en una agencia de viajes las actividades, los hoteles, el coche, etc. Esto ha provocado que hayan tenido que gastar dinero en tareas de organización, que se vean limitados a conocer solo una parte de la isla y que desconozcan la existencia de muchos lugares interesantes que no sabían que existían o que simplemente nadie les había recomendado. Dicho todo esto, éste proyecto plantea la solución de la creación de un chatbot que informe sobre todo tipo de lugares bajo demanda del usuario y que además los pueda recomendar en base a los gustos del turista, de esta manera, el turista podrá saber en qué sitios puede estar interesado y podrá elegir por su cuenta, en vez de aceptar lo que le venda una agencia.

1.2. Propuesta de solución

Existen actualmente diversas aplicaciones para el turismo en Tenerife, pero estas están sujetas a actualizaciones, consumen recursos del teléfono y son demasiado específicas, por ejemplo, búsqueda de rutas para el senderismo, de playas,etc. Por está razón la solución planteada es un chatbot que además procesará el lenguaje natural con el fin de poder entender al usuario sin que éste

se tenga que esforzarse en adivinar qué comando tiene que usar, como pasaría con los bots que se basan en comandos.
Además de todo esto, en el proyecto se persigue como mayor novedad que el chatbot o agente sea proactivo, con la finalidad de recomendar sitios que al turista le puedan interesar; todo esto para crear la mejor experiencia de usuario, necesitando únicamente un dispositivo móvil, hablar el mismo idioma que el chatbot y el uso de Discord o Telegram, aplicaciones de mensajería instantánea en las que el agente actualmente da servicio.

1.3. Objetivos

- Crear un chatbot multiplataforma usando los flujos de diálogos de DialogFlow y las API de busqueda de lugares
- Registrar los gustos y hacer recomendaciones de nuevos lugares.
- Mejorar la experiencia de usuario usando respuestas rápidas y mensajes estructurados.
- Permitir el uso del chatbot en chats grupales y evitar el intrusismo.

Capítulo 2

Contexto

En éste apartado se van a analizar los antecedentes y el estado del arte.

2.1. Antecedentes

La idea de que un programa informático pueda mantener una conversación con un humano se remonta a la época de Alan Turing, con el nacimiento del Test de Turing, que se basaba en conseguir que dada una conversación entre humano y máquina, el humano no supiese distinguir si estaba hablando con otro humano o con una máquina. Esta idea es cada día una realidad mayor gracias a los avances de Inteligencia Artificial y la rama de Machine Learning.

- **ELIZA** [1966]: proclamado como el primer chatbot que logró superar el test de turing, aunque dada la época y las limitaciones tecnológicas, nunca llegó a profundizar en las conversaciones y cayó en el olvido.
- **Clippy** [2007]: el conocido clip del word, que daba sugerencias a los usuarios dependiendo de lo que estuviesen haciendo. Su problema fue que era muy intrusivo y molesto para los usuarios
- **TayTweets** [2016]: el caso más reciente de los aquí nombrados, TayTweets es un chatbot que se encuentra en Twitter desarrollado por Microsoft, su peculiaridad y arma de doble filo a la vez era que aprendía de los usuarios, lo que provocó la burla y mala fé de los usuarios que causó un comportamiento racista por parte del chatbot en menos de 24 horas.

2.2. Estado del arte

Gracias a estos avances en la IA, actualmente los chatbot se están situando como una de las tecnologías más prometedoras para la sociedad, debido a que pueden encontrarse cada vez en más plataformas, existen más para distintos

campos de aplicación y es mayor el número de empresas que los demandan para aumentar la productividad.

Algunas plataformas que soportan chatbots:

- **Telegram**: aplicación de mensajería instantánea con un número de teléfono asociado.
- **Slack**: aplicación de mensajería instantánea destinada principalmente a grupos de trabajo.
- **Discord**: aplicación web/escritorio/móvil que sirve para hablar con grupos de personas por voz y por escrito.
- Facebook Messenger: sistema de mensajería instantánea de Facebook que requiere de una cuenta.

Algunas empresas que usan chatbots:

- **Dominos Pizza**: pizzería que permite realizar pedidos a través de su asistente (chatbot).
- **Expedia**: agencia de viajes que permite reservar hoteles y comparar precios a través de un asistente virtual.
- **Uber**: empresa de transporte privado, cuyo servicio lo dan los usuarios registrados en la aplicación. Esta empresa permite interactuar con un asistente virtual para solicitar transporte o rutas.

Actualmente no existen chatbots de turismo en Tenerife, los únicos productos que tratan el tema son aplicaciones móviles que tienen una serie de desventajas frente a los primeros:

- Consumo de recursos del dispositivo
- Sujetas a actualizaciones descargables
- No suelen existir tanto para Android como para iOS
- Muchas obligan al usuario a registrarse
- En su mayoría contienen publicidad molesta

Aplicaciones:

- **Audioguía Tenerife**: aplicación que contiene audios que guían y dan información sobre los sitios más emblemáticos de Tenerife.

- **GuachApp**: aplicación que permite localizar guachinches cercanos y ver información sobre ellos.
- **ApateApp**: aplicación orientada al senderismo en el sur de Tenerife. Describe las rutas, tiempo estimado y características de los senderos.

Capítulo 3

Aspectos técnicos

Un chatbot no es más que una entidad de código apoyado por un conjunto de tecnologías, que dada una entrada de lenguaje natural realiza acciones de código internas y le da una respuesta al usuario con la finalidad de conseguir realizar el trabajo que haría una persona real.

En base a la anterior definición de chatbot, éste se puede dividir en dos principales tareas de las cuales cada una requiere una tecnología concreta.

1) Entender que intenta hacer o conseguir el usuario.

Esta tarea se basa en el NLP (Natural Language Processor) que analiza lo que ha dicho el usuario y genera un JSON con información esencial.

2) Procesar las peticiones y resolver las acciones

Esta tarea se basa en el uso de un webhook que dado lo que determine el NLP realizará las acciones necesarias para resolver la petición del usuario.

Un webhook es la entidad de código alojado de cara a Internet que permite unificar la comunicación de las distintas aplicaciones que conforman el funcionamiento del chatbot.

3.1. Decisiones técnicas

Dialogflow es una tecnología que solo analiza texto, por lo que todo aquel mensaje que sea enviado directamente a él tendrá que tener texto o será descartado. Dado que nuestro agente necesita solicitar la ubicación del usuario para poder hacer recomendaciones por cercanía se ha decidido no integrar directamente el sistema de mensajería con la tecnología NLP (Dialogflow), provocando su paso por un webhook intermedio que será el encargado de manejar los mensajes no textuales, todo esto por medio de librerías que se analizarán más adelante.

Dado que los chatbot procesan mensajes procedentes de chats de mensajería instantánea, es usual que reciban una gran cantidad de peticiones de poco coste de procesamiento. Éste hecho es problemático cuando se utilizan métodos y funciones bloqueantes, dado que habría que esperar a que cada petición se resolviese para continuar con la siguiente, por lo tanto, es una necesidad el uso de asincronía en éste caso.

La API en la que se va a apoyar el webhook para buscar lugares es la de Google Places, ya que la de TripAdvisor no distribuye claves para proyectos académicos.

Una decisión técnica importante es decidir si se realizará una búsqueda textual o por cercanía. La búsqueda por cercanía se realizará cuando el usuario solicite al chatbot un lugar que esté definido como una entidad en Dialogflow y como un tipo de búsqueda en la API de Google Places, algunos de estos pueden ser bar, restaurante, parque, alquiler de coches, hotel, etc. Mientras que la búsqueda textual se realizará cuando el chatbot desconozca el tipo de local o que el usuario indique un municipio de Tenerife para un tipo conocido.

La base de datos irá incluyendo los lugares que van encontrando los usuarios con el fin de ser más dinámica en el descubrimiento de nuevos, además cuando el usuario requiera de más resultados se comprobará si existen en la base de datos para poder evitar consultas innecesarias a la API.

Una decisión menor ha sido que cuando el chatbot no detecte la intención del usuario, haga una búsqueda textual del mensaje.

3.2. Buscando un lenguaje para el webhook

Conociendo el requisito de la asincronía, se debe elegir un lenguaje que nos permita tratarla de una manera cómoda y eficiente. Para que un lenguaje pueda tratar la asincronía debe usar alguna de estas técnicas:

- Callbacks
- Promesas
- Futuros

Algunos lenguajes que usan estas técnicas:

- NodeJS (Promesas y Callbacks)
- Python (Promesas y Callbacks)
- C++ (Futuros e hilos)

Por lo tanto nos vamos a fijar en sus técnicas de concurrencia:

- NodeJS (Single thread)
- Python (Multi-threading)
- C++ (Multi-threading)

Mientras que para manejar concurrencia en Python y C++ hace falta manejar un pool de hilos por lo que la programación se hace más costosa, en NodeJS solo existe uno, esto es gracias a su arquitectura.

Por lo tanto, por su arquitectura asíncrona, porque va a manejar JSON para interactuar con el NLP y por su facilidad para instalar dependencias (npm) se ha optado por escribir el webhook en **NodeJS**.

3.3. Buscando una base de datos

Analizando el problema se puede apreciar que la única información que se necesita mantener es sobre lugares y sobre usuarios.

Por lo tanto, primero se ha hecho una comparación entre un modelo relacional y uno no relacional.

- Modelo **No Relacional**
  - Ventajas:
    - Capacidad de expansión horizontal (escalable)
    - Rendimiento y disponibilidad alto
    - Se basa en valores de clave - valor
    - Almacenamiento eficiente
  - Desventajas:
    - Consistencia de datos es baja
    - No soporta consultas SQL
- Modelo **Relacional**
  - Ventajas:
    - Consistencia e integridad de datos alta
    - Sistema de consultas sencillo
  - Desventajas:
    - Menor capacidad de escalado
    - Rendimiento y capacidad limitado

Dado que no existe ninguna relación entre usuarios y lugares, que pueden tener información incompleta, que se necesita aprovechar el almacenamiento para cubrir la mayor cantidad y que es necesario que la base de datos sea mucho más rápida que la API de Google Places respondiendo peticiones, se ha decidido usar un modelo no relacional.
Dentro de los modelos no relacionales se han analizado las siguientes posibilidades de almacenamiento:

Almacenamiento de **documentos**: los documentos (conjunto de pares clave - valor) pueden estar codificados en JSON o XML. Un ejemplo de base de datos que use éste modelo es MongoDB.

Almacenamiento de **columnas**: los datos se almacenan en columnas, en vez de en filas, se utiliza una clave para seleccionar un grupo de columnas, cada una contiene una tupla con valores ordenados y separados por comas.

Al ser el lenguaje elegido para interactuar con la base de datos NodeJS, se ha optado por el almacenamiento de documentos por la facilidad de usar JSON (JavaScript Object Notation), por lo tanto la mejor opción es **MongoDB**.

3.4. Buscando un sistema de mensajería instantánea

Los aspectos a tener en cuenta para tomar la decisión de elegir los sistemas de mensajería instantánea donde el chatbot dará el servicio son los son siguientes:

- Popularidad
- Potencia de la API
- Capacidad de manejar ubicaciones
- Menor número de limitaciones y problemas

Por popularidad, la primera opción sería whatsapp, pero no tiene una API para desarrolladores ni permite la integración de chatbots.

Valorando los aspectos en sistemas de mensajería instantánea que permitan la integración:

- **Telegram**
  - Popularidad: 200 millones de usuarios mensuales activos.

  - Potencia de API: tiene continuas actualizaciones, es muy flexible, permite Markdown, permite botones de respuestas rápidas y la gestión de los bots es muy sencilla por medio de @Botfather
  - Ubicaciones: Sí
  - Limitaciones y problemas: No encontrados

- **Facebook Messenger**
  - Popularidad: 1300 millones de usuarios mensuales activos.
  - Potencia de API: la más potente por su capacidad de crear estructuras de informaciones en un chat.
  - Ubicaciones: Sí
  - Limitaciones y problemas: Durante el desarrollo de éste proyecto, el escándalo de Cambridge Analytica ha provocado que Facebook limite y supervise la distribución de tokens para la creación de aplicaciones.

- **Twitter**
  - Popularidad: 313 millones de usuarios mensuales activos.
  - Potencia de API: muy básica en el formateo de respuestas y la integración con webhook en fase beta.
  - Ubicaciones: Sí
  - Limitaciones y problemas: la beta de la api para integrar webhooks está restringida.

- **Discord**
  - Popularidad: 130 millones de usuarios mensuales activos.
  - Potencia de API: avanzada, con muchos tipos de mensajes e interacciones con emociones y además una gran comunidad de desarrollo.
  - Ubicaciones: No, pero lo planean para el futuro.
  - Limitaciones y problemas: Falta de botones para realizar respuestas rápidas.

- **Slack**
  - Popularidad: 8 millones de usuarios mensuales activos.
  - Potencia de API: No se ha valorado.
  - Ubicaciones: No
  - Limitaciones y problemas: muy pocos usuarios.

Por ser tan completo, la primera opción ha sido Telegram, y por a pesar de no ser de las más populares ni reconocer ubicaciones, pero sí tener una gran comunidad y una buena API, la segunda opción ha sido Discord, aunque hubiese sido Facebook si no hubiera existido ese gran problema actual.
Por lo tanto, el chatbot dará servicio en **Telegram** y **Discord**.

3.5. Tecnologías y herramientas

En esta sección se describirán las tecnologías utilizadas

NodeJS

Entorno de ejecución de código abierto para JavaScript caracterizado por su asincronía, su flexibilidad y su popularidad.

NPM

Node Package Manager, el sistema de manejo de paquetes para NodeJS que permite instalar fácilmente dependencias y cualquier paquete que exista en la comunidad.

Dialogflow

Tecnología NLP (Natural Language Processor) que permite procesar el lenguaje natural con el fin de entender qué acción requiere el usuario y con qué parámetros. Sus elementos básicos son:

- Intenciones (intents), se definen como frases o patrones de frase que ha realizado el usuario para determinar una acción o la entrada en un contexto.
- Entidades (entities), son grupos de palabras, que pueden tener sinónimos, que pertenecen a un mismo grupo y que cada vez que son mencionadas por el usuario se capturan como un parámetro.
- Contextos (contexts), son estados en los que entra el chatbot para tener la capacidad de propagar los parámetros anteriores de la conversación entre intenciones.

Genera un JSON de salida con la información necesaria para su procesamiento en el webhook.

mLab

Plataforma conocida por al servicio de usuarios el mayor número de bases de datos MongoDB en la nube, permite una versión gratuita con 0.5 GB de almacenamiento que es la que se ha usado para éste proyecto.

API Google Places

Google Developers es una plataforma que nos permite conocer el uso de sus API, en éste caso la de Google Places, capaz de encontrar lugares por todo el mundo de forma fácil. Su versión gratuita sin activar la facturación, permite 1500 peticiones diarias, y con la facturación activada supera las 10000.

Heroku

Plataforma que da servicios (PaaS) a usuarios en la nube con el fin de que puedan abstraerse de la infraestructura y puedan desplegar sus aplicaciones, las que son denominadas dynos. En el dyno gratuito que permite Heroku, está hosteado el webhook del proyecto.

OpenShift

Plataforma que da servicio (Paas) a usuarios en la nube, usa un cluster de Kubernetes que hay que configurar para que los servicios puedan estar visibles hacia Internet.

3.6. Paquetes npm

Paquete	Descripción
date-and-time	Permite obtener la hora actual con precisión, se usa para calcular la caducidad de la localización del usuario.
discord.js	Permite interactuar con la API de Discord a través de eventos y solo necesitando el token del bot.
google-translate-api	Permite traducir palabras entre idiomas o hacer predicciones de lo que se intenta decir; se usa para traducir los hashtag de lugares no conocidos por el chatbot.
mongodb	Permite instanciar un cliente para interactuar con la base de datos de mLab conectándose a través de una URI.
node-telegram-api	Permite interactuar con la API de Telegram solo necesitando el token del bot y la dirección del webhook.
request	Permite realizar peticiones http.
restify	Permite crear un servidor que escuche en un puerto y definir los path de los métodos POST y GET.
js-recommender	Permite calcular de manera cuantitativa los posibles gustos de los usuarios sobre sitios desconocidos.

Tabla 3.1: Explicación paquetes npm

Capítulo 4

TenTurismo

El resultado de éste proyecto ha sido un agente virtual conversacional proactivo de turismo llamado TenTurismo. En éste capítulo se analizarán los distintos elementos que lo conforman.

4.1. Funcionamiento en Telegram

Para encontrar el contacto de TenTurismo en Telegram lo primero es ir al apartado de contactos y buscarlo por su nombre.

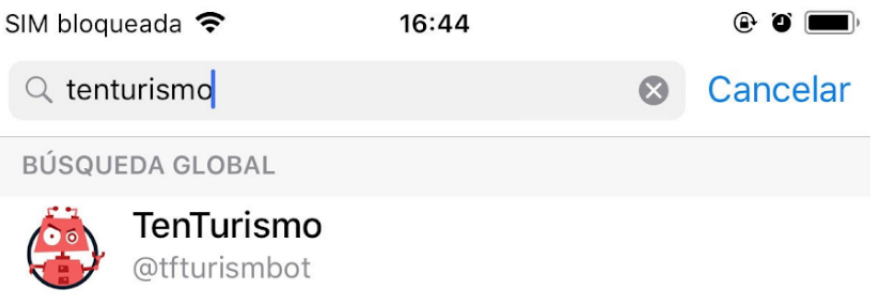


Figura 4.1: Buscando a TenTurismo en Telegram

Una vez seleccionado el contacto aparecerá una breve descripción del chatbot y sus funcionalidades. A usuarios nuevos les aparecerá “Iniciar bot”, mientras que a un usuario que haya tenido contacto previo ,“Reiniciar bot”.

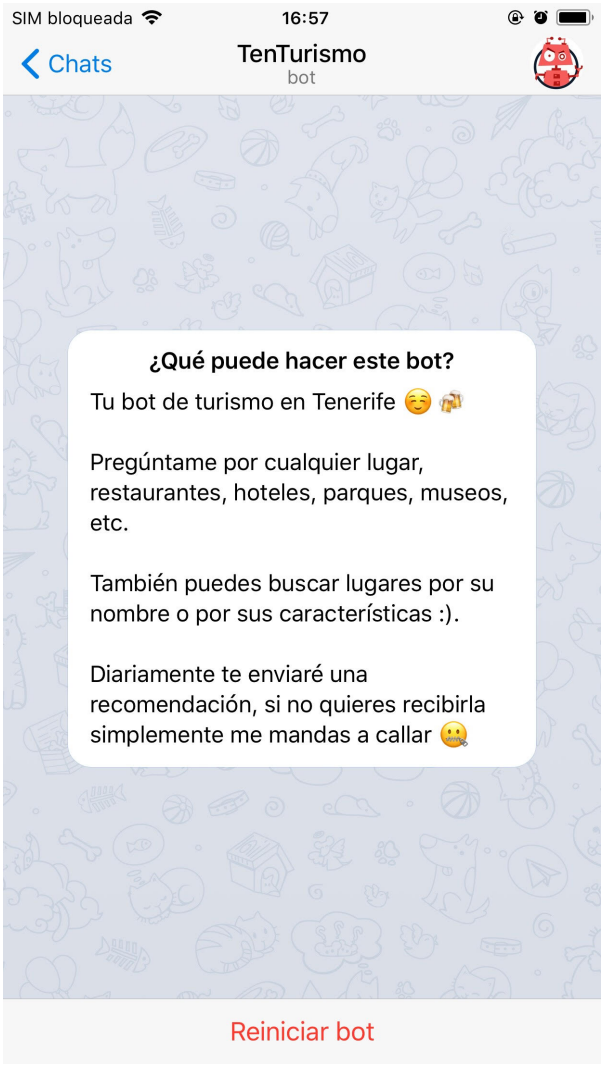


Figura 4.2: Primera vista del chat de TenTurismo

Al iniciarlo, TenTurismo saluda y podremos hacer cualquier petición, por ejemplo, buscar un lugar general como puede ser un parque. Los lugares generales son aquellos que tanto el chatbot como la API de Google Places reconoce como tipo de lugar. En estos casos como se ha mencionado anteriormente, se realizará una búsqueda por cercanía, por lo tanto es necesario solicitar al usuario la ubicación actual en el caso de que le haya caducado (caducan cada 5 minutos). Una vez se le envíe la localización a TenTurismo, nos mostrará el resultado más cercano que cumple con nuestra petición.

Figura 4.3: Búsqueda de lugares generales

En la siguiente imagen, en la conversación de la izquierda se ha desplegado el panel avanzado, que permite añadir a favoritos el lugar, obtener la localización, reviews de usuarios, teléfono de contacto, horario de apertura, web, más fotos u obtener otro resultado.

En la conversación de la derecha, se han obtenido otros resultados pulsando varias veces la respuesta rápida “Otro”.

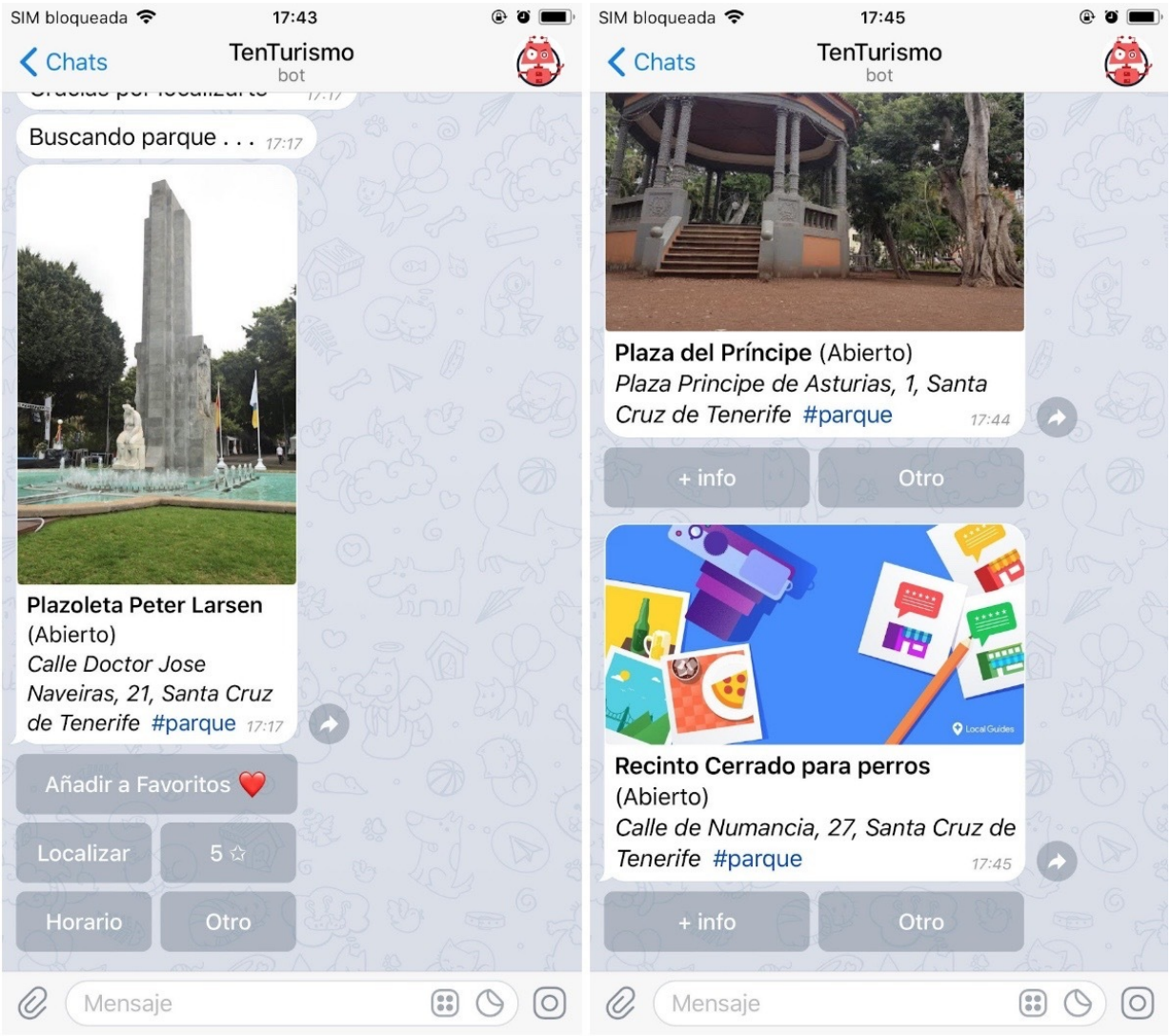


Figura 4.4: Navegación por las opciones de respuestas rápidas

La petición del usuario puede estar incompleta o equivocada, el chatbot maneja contextos, lo que permite añadir parámetros a la búsqueda sin que se tenga que reescribir toda la petición. Por ejemplo, el usuario quería buscar en La Laguna y no en la cercanía, por lo que podría indicarlo a continuación, además del municipio donde se encuentra el lugar, también puede añadir los requisitos máximos o mínimos de puntuación por usuarios o el estado (si está abierto o cerrado).

Figura 4.5: Usando peticiones contextuales

Si un usuario decide buscar lugares concretos como podría ser el Loro Parque, un lugar muy común entre turistas, TenTurismo también es capaz de procesar y responder correctamente a esa petición. Además de la búsqueda de lugares, permite listar los favoritos filtrados por el tipo de lugar, permitiendo recuperar la información sin necesidad de hacer una búsqueda.

Figura 4.6: Búsqueda de lugares concretos y listando favoritos respectivamente

Además, TenTurismo envía diariamente a los usuarios que tengan favoritos la recomendación de un lugar que viene determinado por el recomendador que está alojado en otro webhook y estará explicado más adelante.

Figura 4.7: Recomendación diaria

Todas estas acciones, exceptuando la de recomendación las puede realizar, tanto en chats privados como de grupo, pero para que no sea intrusista en conversaciones grupales hará falta mencionarlo **@tfturismbot** si se desea enviarle peticiones.

4.2. Funcionamiento en Discord

En Discord el funcionamiento es similar al que se explicó en el punto anterior sobre Telegram, aunque no existen algunas opciones, como la búsqueda por cercanía, la localización, más fotos o reviews, y el sistema de recomendación. Esto se debe a la falta de mecanismos de la API para enviar grupos de fotos, botones que permitan las respuestas rápidas y la imposibilidad de compartir la ubicación.

El primer paso para añadir al chatbot a un servidor de discord es buscar al bot por su id (442751764446314496) en la web de Discord, permitirle el acceso y ya queda preparado para dar servicio en ese servidor.

Figura 4.8: Añadiendo el chatbot al servidor y listando los favoritos

Conserva la posibilidad de hacer una búsqueda textual, contextual, listar favoritos, interacción en chats privados y grupos (con mención) e incluye la novedad de añadir y eliminar lugares a favoritos por medio de reacciones.

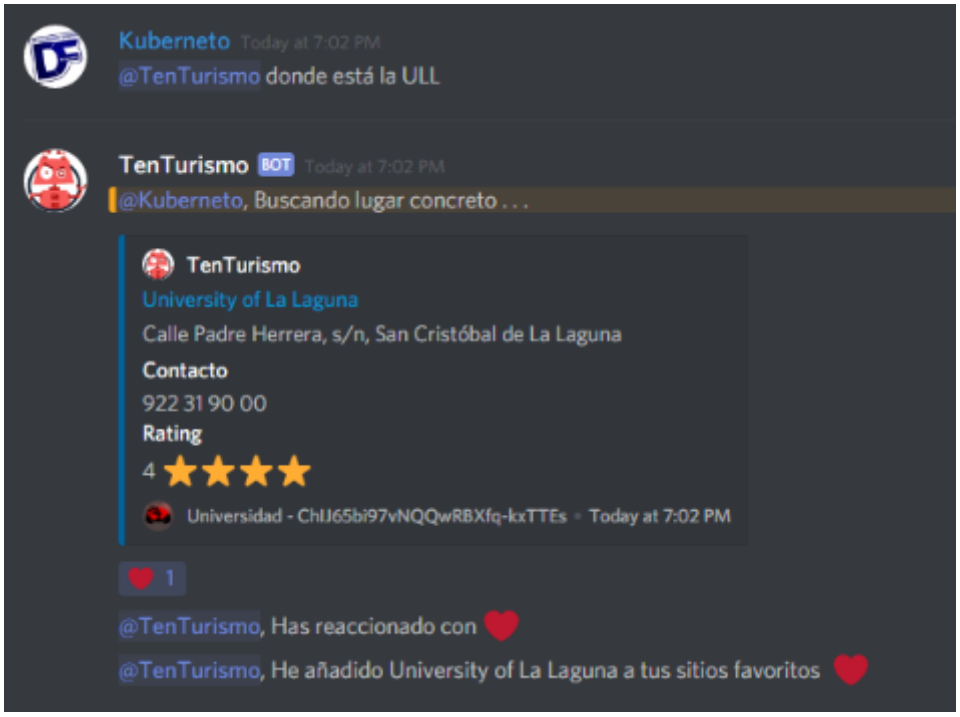


Figura 4.9: Buscando un lugar concreto (La Universidad de La Laguna)

4.3. Funcionamiento del recomendador

El recomendador se ejecuta en un segundo webhook, el que se encuentra en OpenShift, debido a que es una tarea cada vez más costosa a medida que crece el número de usuarios y lugares en la base de datos. Se basa en el uso de una gran tabla, en la que califica con 10 puntos las celdas que corresponden a un lugar favorito para determinado usuario.

El recomendador se basa en el uso de técnicas de filtrado colaborativo, por lo que para que logre su mejor funcionamiento en la predicciones es necesario manejar una gran cantidad de datos, tanto en filas (usuarios) como en columnas (lugares), además este sistema necesita de la participación del usuario puntuando los lugares o de lo contrario sería denominado como oveja gris (Grey Sheep) y no entraría a formar parte de él, en este caso concreto se ha tomado como esa puntuación el hecho de la existencia de un lugar en la lista de favoritos de un usuario. Por lo tanto, aquel usuario Y que tenga en la lista un lugar Z, se considerará que la celda (Y,Z) será del valor de 10, mientras que los lugares que no existan en la lista de favoritos serán incógnitas.

Un sistema recomendador colaborativo puede ser basado en memoria (matriz de datos y correlaciones) o en modelo (redes neuronales, lógica difusa, etc), en este caso se es basado en memoria cuyo proceso se puede resumir en los cuatros puntos siguientes:

- Calcular el grado de similiridad entre usuarios
- Obtención de los K-vecinos y recalcular vectores y patrones
- Calcular las predicciones de los lugares
- Recomendar los N-lugares más valorados

El algoritmo que intenta seguir la librería js-recommender se basa en tener un vector interno de gustos por cada usuario, con cada posición del vector asociada a un patrón , por ejemplo: actividades de montaña, actividades de playa, hosteleria, etc, que simplemente define como agrupar un conjunto de lugares. Cada lugar específico tendrá también un vector de ese mismo tamaño que definirá sobre 1 cuanto se ajusta a cada categoría, dado que este vector es una incógnita cuando el lugar es insertado en la tabla, será calculado basándose en la primera calificación recibida por un usuario y su vector interno de gustos, aunque posteriormente variará a medida que puntuen más usuarios. Un ejemplo para que se entienda mejor:

Vector de gustos: [fiesta, relax]
Vector de gustos del usuario A: [0, 10]

Si el usuario le da la **máxima nota** (significa que es probablemente un lugar de relax):
Vector de categorías del lugar:
Antes de recibir la primera calificación: [?, ?]
Después de recibir la primera calificación: [0 , 1]

Si el usuario le da la **mitad de la nota máxima** (significa que no está totalmente definido que es):
Vector de categorías del lugar:
Antes de recibir la primera calificación: [?, ?]
Después de recibir la primera calificación: [0.5 , 0.5]

Si el usuario le da la **peor nota** (significa que es probablemente un lugar de festejos):
Vector de categorías del lugar:
Antes de recibir la primera calificación: [?, ?]
Después de recibir la primera calificación: [1 , 0]

Además de esto, el vector de usuario será recalculado en base a la puntuación que le otorgue a un lugar y el vector de categorías del lugar, esto se hace debido a que el usuario puede variar sus gustos por estaciones, épocas, etc. La idea básica es que los vectores de los usuarios entrenan los vectores de los lugares y viceversa, con el fin de que ambos se ajusten lo mejor posible a la opinión de los usuarios.

Js-recommender permite abstraerse de todos estos cálculos y búsqueda de patrones, por lo que solo es necesario rellenar la tabla y usar la función que calcula las predicciones.

Lugar	Usuario A	Usuario B	Usuario C	Usuario D
Subway de La Laguna	10	?	?	?
Universidad de La Laguna	?	?	10	10
Museo de la Ciencia	?	10	?	?
Siam Park	?	?	10	?
Loro Park	?	?	10	10
Parque La Granja	?	?	?	?
Parque La Vega	?	?	?	?
TLP Weekend	?	?	10	?

Tabla 4.1: Tabla previa de lugares favoritos

En este ejemplo, antes de realizar el cálculo de los valores desconocidos gracias a la librería js-recommender, se puede identificar que el usuario C con el D tienen patrones de gustos muy parecidos, por lo que la nota calculada para el usuario D en TLP Weekend o Siam Park será cercana a 10. El recomendador elige el valor desconocido calculado con mayor valor como lugar a recomendar, sin tener en cuenta los que ya conoce (favoritos), y se comunica con el otro webhook en '/recomendation'.

Al procesar la tabla con el recomendador quedaría de la siguiente forma:

Lugar	Usuario A	Usuario B	Usuario C	Usuario D
Subway de La Laguna	-	**10**	**9**	9
Universidad de La Laguna	9	8	-	-
Museo de la Ciencia	**10**	-	9	9
Siam Park	9	8	-	**10**
Loro Park	9	8	-	-
Parque La Granja	NaN	NaN	NaN	NaN
Parque La Vega	NaN	NaN	NaN	NaN
TLP Weekend	8	7	-	10

Tabla 4.2: Predicción de la tabla hecha por el recomendador

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

La conclusión de este documento es que se ha obtenido un chatbot de turismo en Tenerife que cumple con todos los objetivos definidos y metas personales, algunos de estos son, que TenTurismo sea fácil e intuitivo de usar, que sea flexible en las consultas, que de servicio en distintos servicios de mensajería, que sea proactivo y que se hayan seleccionado correctamente las tecnologías para desarrollarlo.

Por otra parte, este proyecto me ha aportado una buena cantidad de conocimientos que permite mi desarrollo como Ingeniero Informático, como pueden ser, el aprendizaje de NodeJS, Dialogflow, MongoDB, de distintas APIs como las de Google Places, Telegram y Discord, el uso de variables de entorno, del manejador de paquetes de node, del manejo de http requests (POST, GET, uso del bearer token, etc), desplegar aplicaciones en Heroku, desplegar aplicaciones configurando un cluster a través de Openshift, reforzar conocimiento sobre uso de callbacks y promesas, sobre el concepto de webhook y el desarrollo de la capacidad de comprensión y síntesis.

5.2. Líneas futuras

A pesar de que TenTurismo es bastante completo, existe una gran conjunto de posibles mejoras. Algunas de ellas son:

- Realizar una batería de tests para comprobar el correcto funcionamiento del webhook
- Integrar en más plataformas
- Añadir más parámetros de filtrado en las consultas

- Permitir al usuario calificar los lugares
- Entrenar más las intenciones del chatbot
- Soportar consultas por audio
- Añadir más APIs, por ejemplo para la búsqueda de eventos
- Añadir un modo comando
- Permitir interactuar con algunos locales (reservas, compras, etc)
- Permitir el manejo y la creación de grupos de usuarios
- Añadir más seguridad

Capítulo 6

Summary and Conclusions

6.1. Coclusion

The conclusion of this document is that a tourism chatbot has been obtained in Tenerife that meets all the defined objectives and personal goals, some of these are, that TenTurismo is easy and intuitive to use, that it is flexible with the queries, integrated in different messaging services, that is proactive and that the technologies have been correctly selected to develop it. This project has given me a good amount of knowledge that allows my improvement as a Computer Engineer, such as, the learning of NodeJS, Dialogflow , MongoDB, of different APIs such as Google Places, Telegram and Discord, the use of environment variables, the node packet manager, the management of http requests (POST, GET, use of the bearer token, etc), deploying applications in Heroku, deploying apps on a cluster with Openshift, reinforce knowledge about the use of callbacks and promises, about the concept of webhooks and the improvement of the capacity for understanding and synthesis.

6.2. Future work

- Make a battery of unit tests to check the correct functioning of the webhook.
- Integrations in more plataforms
- More filter parameters for the queries
- Options to qualify places for the user
- To train more the chatbot intents
- Support audio queries
- Adding more APIs, to look for events

- Adding a command mode
- Allow to interact with some stores (reservations, purchases, etc)
- Group management
- More security

Capítulo 7

Presupuesto

En este capítulo se desglosará el coste fijo y de mantenimiento del proyecto en el caso de necesitar contratar una mejora en los servicios debido a la escabilidad que puede necesitar el sistema.

7.1. Costes de desarrollo

Actividad	Horas	Precio (euros)
Análisis de documentación y bibliografía	20	240
Diseño y planificación	20	240
Implementación del chatbot	190	2280
Realizar y analizar tests	10	120
Integración en plataformas	20	240
Realización de una memoria	30	360
Total	300	**3600**

Tabla 7.1: Costes fijos del proyecto

7.2. Costes de recursos

Recurso	Precio (euros/mes)
Base de datos en mLab (40GB)	174
Total	174

Tabla 7.2: Costes de mantenimiento

Apéndice A

Recursos github

A.1. Repositorio DialogFlow

AUTOR: Eduardo Borges Fernández

FECHA: 25/05/2018

DESCRIPCIÓN: Lógica del flujo de conversación del chatbot exportada de DialogFlow (formato JSON)

ENLACE: https://github.com/alu0100885613/DialogFlowLogic

A.2. Repositorio Recomendador

AUTOR: Eduardo Borges Fernández

FECHA: 25/05/2018

DESCRIPCIÓN: Webhook JavaScript que está desplegado en OpenShift y actúa como recomendador de TenTuris

ENLACE: https://github.com/alu0100885613/TenTurismo-recommender

A.3. Repositorio TenTurismo

AUTOR: Eduardo Borges Fernández

FECHA: 25/05/2018

DESCRIPCIÓN: Webhook JavaScript que es el nexo de unón de TenTurismo con todos los componentes

ENLACE: https://github.com/alu0100885613/TenTurismo

Bibliografía

[1] da-14, Python Vs Node. https://da-14.com/blog/python-vs-nodejs-which-better-your-project.

[2] DialogFlow Docs. https://dialogflow.com/docs.

[3] Discord Developers Web. https://discordapp.com/developers/docs/intro.

[4] Discord.js library docs. https://discord.js.org//docs/main/stable/general/welcome.

[5] Google Places Developers Docs. https://developers.google.com/places/web-service/intro.

[6] Heroku Devcenter NodeJS. https://devcenter.heroku.com/categories/nodejs-support.

[7] js-recommender npm Docs. https://www.npmjs.com/package/js-recommender.

[8] MongoDB Docs. https://docs.mongodb.com/.

[9] NodeJS Tutorials. https://www.w3schools.com/nodejs/.

[10] Openshift Blog, Delivering Container Storage-as-a-Service , SHILPI SRIVASTAVA. https://blog.openshift.com/delivering-container-storage-as-a-service/.

[11] Pandorafms, Base de datos NoSQL, Sara Martín. https://blog.pandorafms.org/es/bases-de-datos-nosql/.

[12] Telegram Bot Api. https://core.telegram.org/bots/api.