

Creación de CHATBOT usando TELEGRAM y ontologias.

Trabajo final de sistemas basados en el conocimiento

Abril – Agosto 2018

José Alberto Bustillos Ramón

Universidad Técnica Particular de

Loja

Departamento de Ciencias de la

Computación y Electrónica

e-mail: jabustillos@utpl.edu.ec

Nelson Oswaldo Piedra Pullaguary

Universidad Técnica particular de Loja

Departamento de Tecnologías

Avanzadas de la Web y SBC

e-mail: nopiedra@utpl.edu.ec

Abstract

Existen aplicaciones que logran interactuar en el mercado como es el servicio al cliente o responder inquietudes que tienen las personas ya que la mayoría de las personas con el pasar del tiempo desean tener respuestas inmediatas a sus preguntas. Hoy en día los chatbots son programas que simulan tener conversaciones con las personas logrando satisfacer la necesidad de sus inquietudes, para obtener información sobre el tema se usa un modelo estándar RDF junto con VIRTUOSO como un motor de base de datos y TELEGRAM para la creación del bot por el motivo de que es multiplataforma.

Palabras Claves: RDF, TELEGRAM, VIRTUOSO, chatbot.

I. INTRODUCCIÓN

Hoy en día existen muchas plataformas para la creación de chatbots, este tipo de aplicaciones permite que las personas logren obtener respuestas a sus inquietudes de forma automática y real para esto los bot mantienen una conversación con la personas obteniendo la información de una base de datos estructural.

Los chatbots pueden ser realizados con la mayoría de lenguajes de programación ya que esto permite dar más libertad a los desarrolladores como son conexión de base de datos, entre otros

El motivo de realizar este tipo de aplicaciones es lograr que las personas no tengan que esperar por ser atendidos o aún más esperar días por una respuesta que tal vez otra persona la realizó anteriormente y tenga que repetir la pregunta, lo que permite los chatbots es aparte de tener una conversación brinda información necesaria para lograr la satisfacción y conformidad.

II. METODOLOGÍA

a. Herramientas utilizadas

1. Telegram Bot

Telegram permite crear aplicaciones como son los bots que se ejecutan dentro de la app. Los usuarios logran interactuar con el robot de Plantas Ornamentales por medio de envío de mensajes ya que posee un procesamiento de lenguaje natural obteniendo las entidades para responder la pregunta junto con VIRTUOSO ya que ahí se encuentra almacenada la información, estas peticiones se realizan con protocolos HTTPS logrando comunicarse con nuestra API.[1]

a) API BOT

El API nos permite conectar con los robots a nuestra aplicación. Los motores de búsqueda de telegram son cuentas especiales que no requieren un número telefónico adicional de configurar. Estas cuentas sirven como una interfaz para la ejecución de código en algún lugar de su servidor.

Para usar esto, usted no necesita saber nada acerca de cómo funciona el protocolo de cifrado MTProto - nuestro servidor intermedio se encargará de toda la encriptación y la comunicación con la API de telegramas para usted. A comunicarse con este servidor a través de una sencilla interfaz HTTPS que ofrece una versión simplificada de la API de telegrama.

Funcionamiento de los bots.

Los usuarios pueden interactuar con los robots de dos maneras:[1]

- Enviar mensajes y [órdenes](#) a los robots mediante la apertura de una charla con ellos o mediante su inclusión en grupos. Esto es útil para los robots de chat o los robots de noticias como el funcionario [TechCrunch](#) bot.
- Enviar peticiones directamente desde el campo de entrada escribiendo @ nombre de usuario del robot y una consulta. Esto permite el envío de contenido de [los robots en línea](#) directamente en cualquier charla, grupo o canal.

b) Arquitectura chatbot Telegram

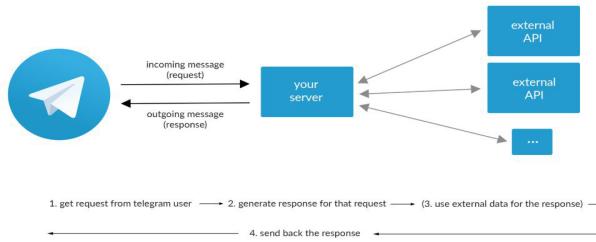


Ilustración 1 Arquitectura Telegram chatbot

En la ilustración 1 se puede apreciar el funcionamiento de un chatbot sencillo, lo cual se necesita un servidor que conecte con las API de mensajería a través de peticiones HTTP y dentro del código fuente del bot procesamos las peticiones entrantes generando respuestas según la entrada de usuario, se puede generar dos maneras respuestas como: basadas en reglas o usando técnicas de inteligencia artificial que sería más difícil pero muy eficiente bajo condiciones diferentes aprovechando más el lenguaje natural tanto escribo como hablado.[2] Prácticamente no tenemos límites, podemos también agregar botones, generar recibos, recibir pagos, subir fotos, videos, audios, y por supuesto texto.

2. Python

Python es un fácil de aprender, potente lenguaje de programación. Tiene estructuras de datos de alto nivel eficientes y un enfoque simple pero efectiva de programación orientada a objetos. Python elegante sintaxis y tipado dinámico, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para secuencias de comandos y el desarrollo rápido de aplicaciones en muchas áreas en la mayoría de las plataformas.[3]

a. Snips_nlu

Snips_nlu es una biblioteca de Python de entendimiento del lenguaje natural que permite analizar las frases

escritas en lenguaje natural, y extraer información estructurada.

Esta librería se basa en recursos lingüísticos que logra soportar varios idiomas como se muestran en la siguiente ilustración[4].

Idioma	código ISO
alemán	de
Inglés	en
Español	es
francés	fr
japonés	y
coreano	mi

Ilustración 2 Tabla de idiomas Snips

b. SPARQLWrapper

Esta es una envoltura alrededor de un servicio de SPARQL. Esto ayuda en la creación de la URI de consulta y, posiblemente, convertir el resultado en un formato más manejable. El paquete está disponible bajo licencia de W3C[5].

3. Virtuoso

Virtuoso Universal Server es compatible con un gran número de arquitecturas de despliegue.

Virtuoso es una base de datos SQL relacional de objetos de alto rendimiento. Como una base de datos, que permite realizar transacciones, un compilador de SQL inteligente, potente lenguaje-procedimiento almacenado con Java y .Net de alojamiento opcional del lado del servidor, copia de seguridad en caliente, soporte de SQL-99 y más. Tiene todas las interfaces de acceso a datos importantes, tales como ODBC, JDBC, ADO .NET y OLE / DB.

OpenLink Virtuoso soporta SPARQL incrustado en SQL para consultar datos RDF almacenados en la base de datos de Virtuoso. SPARQL se beneficia del apoyo de bajo nivel en el propio motor, tales como SPARQL-consciente de las reglas de tipo de fundición a presión y un dedicado tipo de datos IRI[6].

III. DESARROLLO.

La aplicación se desarrolló brinda información sobre plantas ornamentales como son: clasificación, definición,

traducción en quichua y mostrar una imagen sobre la planta que desea saber.

Esto se desarrollo por el motivo de que las personas logren identificar a este tipo de plantas ya que el propósito de estas plantas es para la decoración.

a) Arquitectura del Software

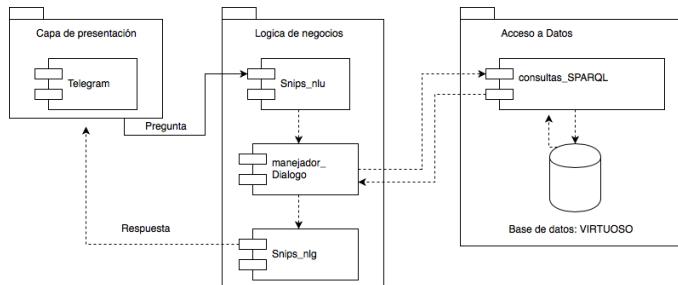


Ilustración 3 Arquitectura del chatbot

En la ilustración 3 se muestra la arquitectura del chatbot la cual fue desarrollado, mostrando como es la interacción del usuario con la aplicación telegram, realizando el script en código python para el procesamiento del lenguaje natural y la conexión con el VIRTUOSO.

b) Utilización de la librería Snips_nlu

La librería Snips_nlu se utiliza para el procesamiento de lenguaje natural, esta librería puede ser usada con Python 2 y Python 3.

Una vez instalada la librería se debe descargar el recurso lingüístico con la que se va a trabajar, este caso solo se trabaja con el lenguaje español. Esto nos permite identificar las diferentes preguntas que ingresa el usuario y lograr identificar las entidades que ingresa para ser las peticiones a VIRTUOSO.

Esta librería trabajo con dos tipos de archivos de formato “txt” en la cual los nombres de estos archivos es fundamental para su funcionamiento, para que la librería logre diferenciar entre intentos y entidades se les nombre de forma distinta como: intent_ ‘intentos’, entity_ ‘entidades’. Para este proyecto existe un archivo de entidades donde se almacenan todos las entidades que se trabaja y 6 archivo intent_ ya que trabajamos con 6 niveles de preguntas.

c) Archivos de intentos

- intent_definicion.txt
- intent_grupo.txt
- intent_imagen.txt
- intent_saludo.txt
- intent_tipo.txt
- intent_traducion.txt

Ilustración 4 Archivos para intentos

En la siguiente ilustración 4 se muestra todos los archivos de las intenciones por cada nivel de pregunta. Las intenciones representan a las preguntas que pueden realizar las personas para obtener la información que necesita como se observa en la ilustración 5, 6 y 7, cada archivo posee su nivel de pregunta, el momento de que el usuario ingrese la pregunta se analizara los intentos y obtener el nombre del intento para poder realizar la detección de la entidad y poder realizar la correcta consulta a VIRTUOSO.

¿Qué tipos de [tipo:tipo](Arboles) conoces?
¿Cuántas [tipo:tipo](Arboles) conoces?
¿Qué tipos de plantas [tipo:tipo](Arboles) conoces?
¿Cuántas plantas [tipo:tipo](Arboles) conoces?
¿Qué plantas [tipo:tipo](Arboles) conoces?
tipos de [tipo:tipo](Arboles)
tipos de [tipo:tipo](Plantas_Ornamentales)
clasificación de las plantas [tipo:tipo](Arboles)
¿Cómo se clasifican las [tipo:tipo](Arboles)?
¿cuál es la clasificación de las [tipo:tipo](Arboles)?
Tipos de [tipo:tipo](Arboles)
Tipos de [tipo:tipo](Palmeras)
Tipo de [tipo:tipo](Palmeras)
¿Cuántos tipos de [tipo:tipo](Arboles) existen?
clasificación de [tipo:tipo](Arboles)

Ilustración 5 Intento para los tipos

Cual es la imagen de [imagen:imagen](Arboles)?
abes la imagen de [imagen:imagen](Arboles)
magen de [imagen:imagen](Arboles)
Cual es la imagen de plantas [imagen:imagen](Arboles)?
abes la imagen de plantas [imagen:imagen](Arboles)
magen de plantas [imagen:imagen](Arboles)
mágenes de [imagen:imagen](Arboles)
magen de [imagen:imagen](Arboles)
magen de [imagen:imagen](Caladium_Bicolor)

Ilustración 6 Intento para la imagen

```
[definicion:definicion](Arboles)
[definicion:definicion](Aro_Comun)
[definicion:definicion](Calta_Amarilla)
[definicion:definicion](Plantas_Ornementales)
¿Qué son las plantas [definicion:definicion](Arboles)?
¿Qué son [definicion:definicion](Arboles)?
define [definicion:definicion](Arboles)
¿Qué son los [definicion:definicion](Arboles)?
definición de [definicion:definicion](Plantas_Ornementales)
```

Ilustración 7 Intento para la definición

Existe la posibilidad de que no se registren todos los intentos ya que las personas realizan muchas formas de hacer una pregunta por una respuesta para esto se realizó llevar un registro de las fallas de las entidades para poder ser ingresados estos nuevos intentos y hacer que nuestra data cresca dando una mejor interacción al usuario.

En el contexto de la extracción de información, un *intento* corresponde a la acción o la intención contenida en la consulta del usuario, que puede ser más o menos explícito.

d) Archivos de intenciones

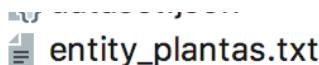


Ilustración 8 Archivo de entidades

El archivo de entidades sirve para que la librería logre identificar las entidades en las preguntas para poder realizar la consulta sobre el tema que deseó. Estas entidades van a las consultas SPARQL extrayendo la información del SKOS.

Cuando el usuario ingrese una pregunta y exista una nueva entidad que tal vez no fue registrada y se trata sobre el tema, esa entidad nueva se almacena en el archivo de registro con la palabra “*ALERTA*” esto le permitirá al desarrollador de que la entidad no ha sido encontrado.

Estas entidades tienen etiquetas especiales a partir de “snips/” y haciendo uso de ellos cuando es apropiado no solo dará mejores resultados, sino que también proporcionará alguna *resolución de entidades* tales como un formato ISO para una fecha.

```
Acuáticas, acuáticas, Acuática, acuática,acuáticas, Acuática, acuática
Alpinia_Purpurata, Alpinia Purpurata, alpinia purpurata, Alpinia purpurata
Arboles, arboles, Arbol, arbol, Arbol, árbol
Aro_Comun, aro comun, Aro Comun, Aro comun
Biganvilla, buganvilla,Buganvillas, buganvillas
Caladium_Bicolor, Caladium Bicolor, caladium bicolor, Caladium bicolor
Calta_Amarilla, calta amarilla, Calta Amarilla, Calta amarilla
Campanillas, campanillas, campanilla, Campanilla
Canna_Indica, canna indica, Canna indica
Caryota_Mitis, caryota mitis, Caryota mitis
Chamaecyparis_Pisifera, chamaecyparis pisifera
Chamaedorea_Elegans, chamaedorea elegans
Citrus_Medica, citrus medica
Dahlia_Hybrida, dahlia hybrida
Delonix_Regia, delonix regia
Enamorada_deLmuro, enamorada del muro
Flor_de_Loto, flor de loto
Hiedra, hiedra
Jacinto_Agua, jacinto agua
Ligustrum_Lucidum, ligustrum lucidum
Lilium_Spp, lilium spp
Nenufar_Blanco, nenufar blanco
Palmeras, palmeras, palmera, Palmera
Pasionaria, pasionaria, Pasionarias, pasionarias
Plantas_Ornementales, plantas, planta, Planta, plantas ornamentales
Trepadoras, trepadoras, trepadora, Trepadora
Tuberosas, tuberosas, tuberosa, Tuberosa
```

Entidades incorporadas y sus extractores subyacentes son mantenidos por el equipo de SNIPS. Puede encontrar la lista de todas las entidades incorporadas soportados por cada idioma en el SNIPS Ontología repositorio. El SNIPS UDE utiliza la poderosa Rustling biblioteca para extraer entidades incorporadas a partir del texto.

Por otro lado, las entidades que se declaran por el desarrollador se llaman *personalizados* entidades.

Los valores de la entidad y sinónimos, la primera cosa que puede hacer es añadir una lista de valores posibles para su entidad.

Al proporcionar una lista de valores de ejemplo para su entidad, ayuda a SNIPS UDE comprender lo que la entidad está a punto.

e) Conjunto de datos

Una vez de que ya estén identificados las entidades y los intentos se debe generar el conjunto de datos, ya que este archivo ayuda a la creación del archivo de entrenamiento para la librería todos estos archivos se encuentran en formato muy utilizado JSON como se muestran en la ilustración 9. Se debe tener en cuenta en lenguaje en el lenguaje que va a ser desarrollado para poder realizar sin complicación sino existirán errores de sintaxis por el lenguaje, ya que cada lenguaje posee su propio sintaxis.

```
'entities": {
    "definicion": {
        "automatically_extensible": true,
        "data": [],
        "use_synonyms": true
    },
    "español": {
        "automatically_extensible": true,
        "data": []
    },
    "use_synonyms": true
},
"grupo": {
    "automatically_extensible": true,
    "data": []
},
"use_synonyms": true
},
"imagen": {
    "automatically_extensible": true,
    "data": []
},
"use_synonyms": true
},
"plantas": {
    "automatically_extensible": true,
    "data": [
        {
            "synonyms": [],
            "value": "Abies religiosa"
        },
        {
            "synonyms": [],
            "value": "Acuaticas"
        },
        {
            "synonyms": [],
            "value": "Alpinia Purpurata"
        },
        {
            "synonyms": [],
            "value": "Arboles"
        },
        {
            "synonyms": [],
            "value": "Ara Comum"
        }
    ]
}
```

Ilustración 9 Datos del archivo dataset.json

f) Entrenamiento de motor.

Una vez obtenido el archivo dataset.json podemos lograr el entrenamiento con las entidades e intentos ingresados anteriormente y organizados con el formato JSON como se observa en la ilustración 10.

Ilustración 10 Datos del archivo trained.json

g) Extracción de la entidad

Al momento de que el usuario ingrese la pregunta y la envié el snips debe poder identificar la entidad y el intento para esto se usa el archivo trained.json ya que posee toda esa información estructurada como se ve en la ilustración 11.

```
snips_nlu.load_resources("es")
reload(sys)
sys.setdefaultencoding('utf8')
# Lectura del archivo de entrenamiento para identificar la
with io.open("trained.json") as f:
    engine_dict = json.load(f)
engine = SnipsNLUEngine.from_dict(engine_dict)

# metodo para obtener el detalle de la pregunta realizada
def pregunta(frase):
    r = engine.parse(unicode(frase))
    return json.dumps(r, indent=2)
```

Ilustración 11 Obtención de la entidad e intento

Al momento de realizar esta consulta nos retorna estos valor en una formato JSON en la cual se debe extraer la información sobre el nombre del intento y ademas extraer la entidad para poder obtener el resultado como se muestra en la ilustración 12.

```
# procesamiento del texto ingresado para su identificacion
respuesta_json = json.loads(pregunta(texto))
# archivo de control de palabras erroneas
f = open("registro.txt", "a")
try:
    # captura de la intencion
    intencion = respuesta_json['slots'][0]['entity']
    # captura de la entidad a preguntar
    entidad = respuesta_json['slots'][0]['value']['value']
```

Ilustración 12 Extracción de la entidad e intención

h) Consultas SPARQL.

Cuando se tenga la entidad y el intento correctamente identificado se direccionara por la condición correspondiente para realizar la consulta como se ilustra en la ilustración 13, esto enviara a la entidad a la sentencia SPARQL que se encuentran en el archivo snips_prueba.py ya que en este archivo se encuentra el script de la conexión y la estructura de la sentencia SPARQL hacia VIRTUOSO.

```

if len(lista_plantas) != 0:
    respuesta = ""
    # almacena los nombres de preguntas con varias respuestas
    lista_nombres_plantas = []
    # obteniendo los resultados para presentarlos
    for i in lista_plantas:
        lista_nombres_plantas.append(i[0:])
    # presentación de datos junto con el conteo
    respuesta = "Por el momento conozco " + str(len(lista_nombres_plantas)) + " plantas:\n"
    for j in lista_nombres_plantas:
        # condición para presentar varias respuestas
        if len(lista_nombres_plantas) == 1:
            respuesta = j
            break
        elif lista_nombres_plantas[-1] == j:
            respuesta = respuesta + " y " + j
        elif lista_nombres_plantas[-2] == j:
            respuesta = respuesta + j + " "
        else:
            respuesta = respuesta + j + ", "
    else:
        # imprimir mensaje de error enviando numero randomico
        respuesta = respuestaError[random.randint(1, 3)]
        print('error 1')
        # almacena la cadena de texto que ocasio error
        f.write(texto + '\n')
        f.close()
# condición para encontrar definiciones de palabras

```

Ilustración 13 Control de entidad y de errores

Antes de realizar este proceso pueda que dirija una entidad que no se encuentre en VIRTUOSO esto daría un error al momento de retornar el resultado dando un valor nulo y terminando por completo la ejecución del chatbot para esto se le realiza una condición para cuando este valor sea nulo envíe un mensaje de errores mostrando de que fue mal formulada la pregunta.

Para que la conversación no solo tenga una mensaje de error ya que puede existir fallas sé ha creado una lista con diferentes mensajes de error para que la conversación no sea repetitiva para poder realizar esto al momento de que

existe cualquier tipo de error se le envía la lista donde esta almacenados estos mensajes pero para que sean varios al momento de enviar la posición del mensaje se envía un número randomico para que se envié de diferentes formas de mostrarle al usuario de que no se entendió la pregunta y además que estos fallos existan poder actualizar nuestro registro de fallos.

i) Consulta de imágenes.

Sabemos que el API de telegram nos permite enviar varios recursos, me permite poder mostrar la imagen de la planta que el usuario o cliente desea saber para esto el usuario debe ingresar una pregunta para poder obtener el intento imagen como se muestra en la ilustración 14 y poder lograr entrar al método donde se encuentra la dirección del archivo esto es posible ya que las imágenes son almacenadas con el nombre de la entidad permitiendo encontrar la imagen, una vez obtenida la dirección de la imagen que deseamos mostrar la enviamos a la API de telegram para mostrar en la interfaz del chat como se muestra en la ilustración 15.



Ilustración 14 Lista de imágenes

```
elif (intencion == 'imagen'):
    # obtención de la dirección de la imagen
    uri = imagen(entidad)
    # saber si la imagen existe
    valor = os.path.exists(uri)
    # condición para enviar error si la imagen no existe
    if (valor == True):
        # cargar imagen |
        photo = open(uri, 'rb')
        # mensaje de la imagen consultada
        respuesta = 'Imagen de plantas ' + entidad
        # comando para enviar la imagen a la interfaz de chat
        bot.send_photo(chat_id, photo)
    else:
        respuesta = 'no encontramos la imagen'
```

Ilustración 15 Cargar imagen

Nuestro chatbot no solo puede mostrar imágenes, logra dar información sobre la definición de las plantas, la traducción en quichua, obtener en que grupo pertenece la planta y su clasificación todo esto se puede lograr gracias a nuestro SKOS cuando realizamos la organización de la información como se observa en la ilustración 16.

```
def tipos(entidad):
    print('***** Tipos *****')
    resultado = []
    print('tipos ' + entidad)
    # envío de la sentencia SPARQL a VIRTUOSO
    sparql.setQuery('''
        select ?tipos where {
            <example> ?entidad > http://www.w3.org/2004/02/skos/core#narrower > ?tipos
        }''')
    sparql.setReturnFormat(JSON)
    resultados = sparql.query().convert()
    print(resultados)
    # extracción del resultado del formato JSON
    for valor in resultados['results']['bindings']:
        dato = (valor['tipos']['value'])
        resultado.append(dato.strip('example:'))
    # retorno del resultado para ser presentado
    print(resultado)
    return (resultado)

def definicion(entidad):
    resultado = []
    print('***** DEFINICION *****')
    print(entidad)
    sparql.setQuery('''
        select ?tipos where {
            <example> ?entidad > http://www.w3.org/2004/02/skos/core#definition > ?tipos
        }''')
    sparql.setReturnFormat(JSON)
    resultados = sparql.query().convert()
    print(resultados)
    # for valor in resultados['results']['bindings']:
    #     dato = (valor['tipos']['value'])
    #     resultado.append(dato)
    print(resultado)
    return (resultado)
```

Ilustración 16 Sentencias SPARQL

IV. CONCLUSIONES

- Se debe utilizar librerías de lenguaje identificación de entidades para obtener las URIS del parrafo.
- Se debe tener cuidado al momento de enviar la sentencia SPARQL para evitar que el servicio expire.
- La velocidad de obtención de las URIS depende del acceso a internet ya que utilizamos como base de conocimiento a dbpedia.

V. RECOMENDACIONES

- Para realizar software basado en RDF se debe tener conocimiento sobre la web semántica y su funcionamiento.
- Se debe tener conocimientos en SPARQL para realizar las sentencias de consulta y poder limitar el número de tripletes evitando que el sistema no reaccione al momento de presentar los resultados en la plantilla.

VI. REFERENCIAS

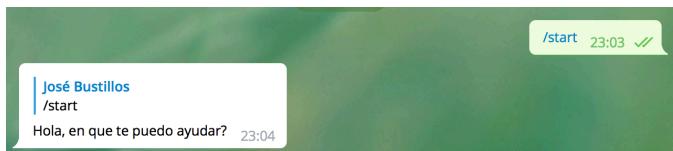
- [1] Core.telegram.org. (2018). *Bots: An introduction for developers.* [online] Available at: <https://core.telegram.org/bots> [Accessed 23 Jul. 2018].
- [2] Medium. (2018). *Los ChatBots en la Actualidad – Parkode – Medium.* [online] Available at: <https://medium.com/parkode/los-chatbots-en-la-actualidad-1d0aa68447ce> [Accessed 23 Jul. 2018].
- [3] Docs.python.org. (2018). *The Python Tutorial — Python 3.7.0 documentation.* [online] Available at: <https://docs.python.org/3/tutorial/index.html> [Accessed 23 Jul. 2018].

- [4] Snips-nlu.readthedocs.io. (2018). *Supported languages — Snips NLU 0.16.1 documentation*. [online] Available at: <https://snips-nlu.readthedocs.io/en/latest/languages.html#languages> [Accessed 23 Jul. 2018].
- [5] Rdflib.github.io. (2018). *SPARQL Endpoint interface to Python*. [online] Available at: <https://rdflib.github.io/sparqlwrapper/> [Accessed 23 Jul. 2018].

[6] Wikis.openlinksw.com. (2018). [online] Available at: <http://wikis.openlinksw.com/VirtuosoWikiWeb/VirtuosoProductOverview> [Accessed 23 Jul. 2018].

Anexos:

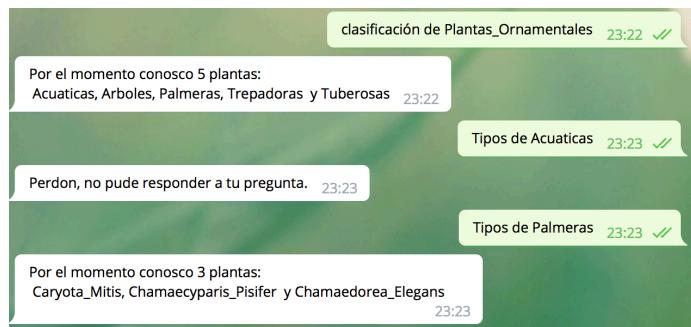
1. Inicio de la conversación.



2. Extraer la definición



3. Clasificación de plantas



4. Imagen de plantas



5. Traducción a Quichua



6. Registro de errores.

```
Perro *ALERTA*
Gato *ALERTA*
Pez *ALERTA*
Acuaticas
Acuaticas
Acuatica *ALERTA*
es que grupo pertenece las Acuaticas
en que grupo pertenece las Palmeras
Tipos de Acuaticas
en que grupo pertenece las Caryota_Miti
```

