ChatBot de automovilismo con Virtuoso, Snips y Telegram

Trabajo final sistemas basados en conocimiento abril 2018

Daniel Patricio Peñarreta Feijoo
Universidad Tecnica Particular de Loja
Departamento de Ciencias de la Computación y Electronica
e-mail: dppenarreta@utpl.edu.ec

Nelson Oswaldo Piedra Pullaguari
Universidad Tecnica particular de Loja
Departamento de Tecnologias Avanzadas de la Web y SBC
e-mail: nopiedra@utpl.edu.ec

Abstract

El proyecto se refiere sobre un chatbot el cual fue creado e implementado en lenguaje python con ayuda de la librería de SNIPS, la informacion fue cargada en virtuoso para poder ser obtenida mediante sentencias en lenguaje SQL, y para la interaccion entre el usuario final y el chatbot se utilizo la red social telegram como medio de comunicación fijo.

Palabras Claves: SKOS, SPARQL, Web, Telegram, Chatbot, Snips, Python.

I. INTRODUCCIÓN

El proyecto se encuentra construido con ayuda SNIPS que es la librería principal que nos permite hacer el reconocimiento de intenciones, realizar las consultas en lenguaje SPARQL y posteriormente la conexión con el API de Telegram para la presentación de datos e interacción con el usuario final. Además de SNIPS se utiliza el API de telegram para permitir la conexión, y Virtuoso para el almacenamiento de la información en formato RDF.

1. METODOLOGÍA.

1. SKOS

[1]Simple Knowledge Organization System (SKOS) es una familia de lenguajes formales, diseñada para la representación de tesauros, sistemas de clasificación, taxonomías o cualquier otro tipo de vocabulario estructurado. SKOS se basa en RDF y RDFS, y su objetivo principal es permitir la publicación fácil de vocabularios controlados y estructurados para la Web semántica. SKOS se desarrolla en el marco de la W3C y es uno de los vocabularios RDF utilizado en el proyecto SDMX-RDF para expresar el intercambio de datos y metadatos estadísticos en el estándar SDMX basado en RDF.
[2] [3]El modelo de datos SKOS es en realidad una ontología definida con OWL Full. Obviamente, al estar basado en RDF, SKOS estructura los datos en forma de tripletas que pueden ser codificadas en cualquier sintaxis válida para RDF. SKOS puede ser utilizado conjuntamente con OWL para expresar formalmente estructuras de conocimiento sobre un dominio concreto ya que SKOS no puede realizar esta función al no tratarse de un lenguaje para la representación de conocimiento formal. El conocimiento descrito de manera explícita como una ontología formal se expresa como un conjunto de axiomas y hechos. Pero un tesauro o cualquier tipo de esquema de clasificación no incluye este tipo de afirmaciones, sino que identifica y describe (con el lenguaje natural o expresiones no formales) ideas o significados a los que nos referimos como conceptos. Estos conceptos pueden organizarse en estructuras que carecen de una semántica formal y que no pueden considerarse como axiomas o hechos. Es decir, un tesauro únicamente proporciona un mapa intuitivo de cómo están organizados los temas dentro de procesos de clasificación y búsqueda de objetos (generalmente documentos) relevantes a un dominio específico. Para convertir un tesauro o esquema de clasificación en conocimiento formal, debe transformarse en una ontología, un proceso que resulta muy costoso.

2. Virtuoso

[4] [5] Virtuoso es un innovador servidor de datos multimolde de grado empresarial para empresas e individuos ágiles. Ofrece una solución agnóstica de plataforma inigualable para la administración, el acceso y la integración de datos. La exclusiva arquitectura de servidor híbrido de Virtuoso le permite ofrecer una funcionalidad de servidor tradicionalmente distinta dentro de una oferta de producto único que cubre las siguientes áreas:

- Gestión de datos relacionales
- Gestión de datos RDF

• Gestión de datos XML
• Gestión de contenido de texto libre e indexación de texto completo
• Servidor de datos vinculado
• Servidor de aplicaciones web

Configuración de virtuoso

a) Descarga del servidor virtuoso

Virtuoso de Open Link se puede descargar de la siguiente url: http: //virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSDownload.

b) Inicio virtuoso

Virtuoso es un servidor portátil; por lo tanto, no requiere ninguna instalación. Solo necesita extraer el archivo Zip en algún directorio. La extracción Zip creará una carpeta con el nombre virtuoso-opensource que contiene todos los archivos del servidor. Dentro del directorio virtuoso-opensourcego to database y copie el archivo virtuoso.ini en el directorio bin. Ahora vamos a la dirección donde extrajimos los archivos luego a virtuoso-opensource \ bin usando la línea de comando y ejecute el siguiente comando: Virtuoso-t -f .

c) Carga de datos en el extremo virtual SPARQL

Para la carga de RDF con virtuoso conductor Este método es útil para cargar archivos RDF pequeños (por ejemplo, archivos de 100 o 200 MB). En este método, solo se puede cargar un archivo a la vez.
Aquí realizamos los siguientes pasos.
1. Vamos al enlace http: // localhost: 8890 / y hacemos clic en el conductor en el lado izquierdo.
2. Escriba dbaas, tanto la cuenta de inicio de sesión como la contraseña.
3. Haga clic en la pestaña Linked Data y luego en Quad Store Upload
4. Seleccione su archivo RDF (solo uno a la vez), dé un nombre de gráfico apropiado y haga clic en cargar
5. Si no hay ningún error de sintaxis, se agregará el archivo RDF al servidor virtuoso que puede consultar utilizando la interfaz pública dada en http: // localhost: 8890 / sparql .

3. SNIPS NLU

[6] [7]Los motores NLU se utilizan para alimentar cualquier chatbot o asistente de voz. Su objetivo es identificar la intención del usuario (también conocido como intención) y los parámetros (también llamados espacios) de la consulta. El desarrollador puede usar esto para determinar la acción o respuesta apropiada.
Algunas de las características que presenta snips nlu:
Comportamiento determinista
Lo primero que desea es que todos los ejemplos que proporcione para capacitar al modelo sean respaldados correctamente por el motor. Esto hace que el sistema sea predecible y fácil de usar: si una consulta no se analiza correctamente, agréguela al conjunto de datos y funcionará de inmediato.

Poder de generalización
Tener este comportamiento determinista es excelente para la solidez y la previsibilidad, pero un potente motor de NLU también necesita tener cierto poder de generalización. Desea que el sistema no solo reconozca los patrones provistos en el conjunto de entrenamiento, sino también todas las variaciones posibles que provienen del habla natural. Si volvemos al conjunto de datos anterior, es razonable esperar que el motor de NLU analice una consulta como: "¿Cuál es el clima en Beijing en este momento?", Aunque no forma parte de los ejemplos de capacitación.
Resolución de entidad
Por último, necesita algo llamado Resolución de Entidad. Esencialmente, extraer el trozo "el tercer domingo de marzo de 2018" de la oración "Necesito el clima para el tercer domingo de marzo de 2018" es un buen primer paso. Sin embargo, lo que quiere hacer a continuación es llamar a una API meteorológica para conocer el clima, y hay pocas posibilidades de que la API acepte cadenas de fechas sin procesar como entrada. Será más bien tomar una fecha en formato ISO: 2018–03–18. Lo posterior se conoce como el valor resuelto de la entidad.

[8]Para cumplir con estos objetivos: comportamiento determinista, poder de generalización y la capacidad de resolver entidades, construimos el canal de procesamiento descrito en la figura anterior. Recibe texto como entrada y emite una respuesta estructurada que contiene el intento y la lista de espacios. La unidad de procesamiento principal de la tubería es el motor NLU. Contiene dos analizadores intencionados que se llaman sucesivamente: un analizador intencional determinista y uno probabilístico.

- Ambiente de Snips nlu

El ecosistema de Snip NLU potencia todo lo relacionado con NLU en Snips. Snips NLU se utiliza para capacitar a los modelos generados en la consola web de Snips, en trabajadores de Python. Snips NLU Rust se utiliza para ejecutar la inferencia en cualquier lugar: en nuestra consola web usando un servidor Scala, o en el dispositivo, ya sea en Linux, iOS o

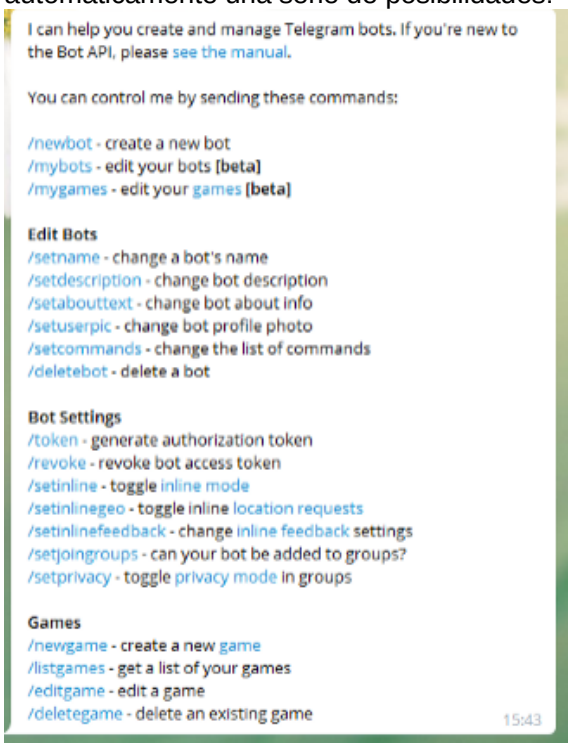
Android. Para obtener el mismo código para ejecutar en entornos tan diversos y restringidos, apostamos fuertemente por Rust . Este lenguaje moderno ofrece alto rendimiento y baja sobrecarga de memoria, así como seguridad de la memoria y compilación cruzada como ciudadanos de primera clase. Se utiliza una serialización JSON de modelos entrenados como interfaz en las bibliotecas Snips NLU.
Esto hace que Snips NLU sea la primera biblioteca de código abierto NLU que es totalmente portátil.

4. Telegram API

[9]Telegram es una plataforma de mensajería móvil. Telegram ofrece servicios de mensajería altamente encriptados para que los usuarios puedan enviar mensajes, fotos y videos a contactos seleccionados de forma privada. La API de Telegram permite a los desarrolladores acceder e integrar la funcionalidad de Telegram con otras aplicaciones. Algunos ejemplos de métodos API incluyen la administración de contactos, la verificación de teléfonos y el envío y recuperación de mensajes, fotos y videos.

1. Bot Father

Bot Father, como su propio nombre indica, es el padre de todos los bots en Telegram. Sí, los desarrolladores de esta app de mensajería ese día estaban muy graciosos y optaron por llamarle así. Lo primero que se debe de hacer es buscar al BotFather utilizando el buscador de Telegram. Cuando lo encontremos, debemos iniciar una conversación. El cual luego ofrece automáticamente una serie de posibilidades:



Como es de esperar, lo primero que debemos hacer es clickar en “/newbot”. Al clickar BotFather nos desplegara un mensaje en el que nos indicara que asígnenos un nombre al bot un nombre. Tras comprobar que todo está correcto, BotFather lanzará el token.

5. Automovilismo.

El automovilismo es una competición o prueba de velocidad entre vehículos terrestres propulsados mecánicamente, sobre tipos distintos de pistas. Los competidores corren en grupo o en solitario, en cuyo caso son cronometrados por separado.

El origen de la práctica del automovilismo, comienza a causa de los avances tecnológicos y de la invención del automóvil. El primer auto alimentado a combustible derivado del petróleo, fue inventado a finales de 1.885 con un Motor-Wagen por Karl Benz. Rápidamente surgió el interés por competir y realizar pruebas de velocidad de estos vehículos terrestres, con avances mecánicos y en diversas pistas. En las primeras carreras intervenían también vehículos a vapor.

La primera carrera se realizó en 1.887, desde París a Versalles. A partir de esta época se realizan varias competiciones en Estados Unidos y otros lugares. Para 1.900, se realiza en Lyon (Francia) el Primer Campeonato Internacional de Automovilismo, del que participaron cinco pilotos de cuatro países distintos. El auto ganador fue un Panhard francés, que mantuvo una velocidad promedio de 62 kilómetros por hora.

En 1.906 comienzan a llevarse a cabo las carreras del Gran Premio. La primera edición se realiza en Francia. Empiezan a realizarse distintas carreras, como las de circuitos cerrados, las de Turismo de Carretera, y en distintas pistas como autocross, carreras sobre pistas de hierba, subidas a puertos, karts, rallyes, carreras a campo traviesa, carreras fuera de pista, carreras de carros de serie y carreras de Grand Prix. Hay muchas subdivisiones y clases de vehículos. El automovilismo es un deporte en crecimiento, que incrementa el número de competiciones, con mejoras constantes de autos y circuitos.

2.1. Disciplinas:

Las disciplinas automovilísticas pueden tener diferentes clasificaciones: por tipo de automóvil (monoplaza, turismo, stock, de producción, gran turismo, clásicos...), por el tipo de competición (circuito de asfalto, de tierra o de hielo, rally, campo a través) y por el objetivo (velocidad, resistencia, derrapadas). Algunas de las más destacadas e ilustrativas de lo anterior, son:

2.1.1. Monoplaza

Los monoplazas son vehículos diseñados especialmente para competición. Llevan alerones y neumáticos anchos para agarrarse al piso lo más posible, y las ruedas no están por lo general cubiertas. Son vehículos muy bajos, pues rondan el metro de altura, y hay solamente lugar para una persona (de ahí el nombre monoplaza).

Fórmula 1 es la categoría más popular, sobre todo en Europa. Los equipos, generalmente divisiones de fabricantes (Ferrari, Mercedes, Renault y Red Bull) utilizan presupuestos de cientos de millones de euros para desarrollar las últimas tecnologías que les permitan ganar centésimas de segundo en la pista

2.1.2. Rally

Las competiciones de rally se desarrollan por vías públicas cerradas al tránsito rodado; los participantes (piloto y copiloto) deben recorrer un camino predeterminado en el menor tiempo posible. Cada automóvil sale con un minuto respecto del siguiente, por lo que no hay contacto visual ni físico entre ellos. Generalmente los automóviles son derivados de los de producción; según la categoría se modifican más partes y en mayor medida.

El Campeonato Mundial de Rally utiliza automóviles del segmento C con diferentes preparaciones. Los más potentes y utilizados por los equipos oficiales son los World Rally Car usan motores de 1.6 L con turbocompresor y altamente modificados. En los últimos años varias marcas compiten arduamente por la victoria: Citroën, Volkswagen o Ford, entre otras. Entre los rallys más famosos se encuentran Montecarlo, Finlandia, Suecia, Acrópolis, Córcega (Francia), el RAC (Reino Unido) o el Catalunya-Costa Daurada (España).

2.1.3. Turismo

Esta categoría se corre en circuitos cerrados de asfalto con automóviles de turismo. Para emparejar las prestaciones y bajar costos, los automóviles tienen muchos elementos en común con sus hermanos de producción, con modificaciones en aspectos como la seguridad, motor, frenos, y suspensiones.

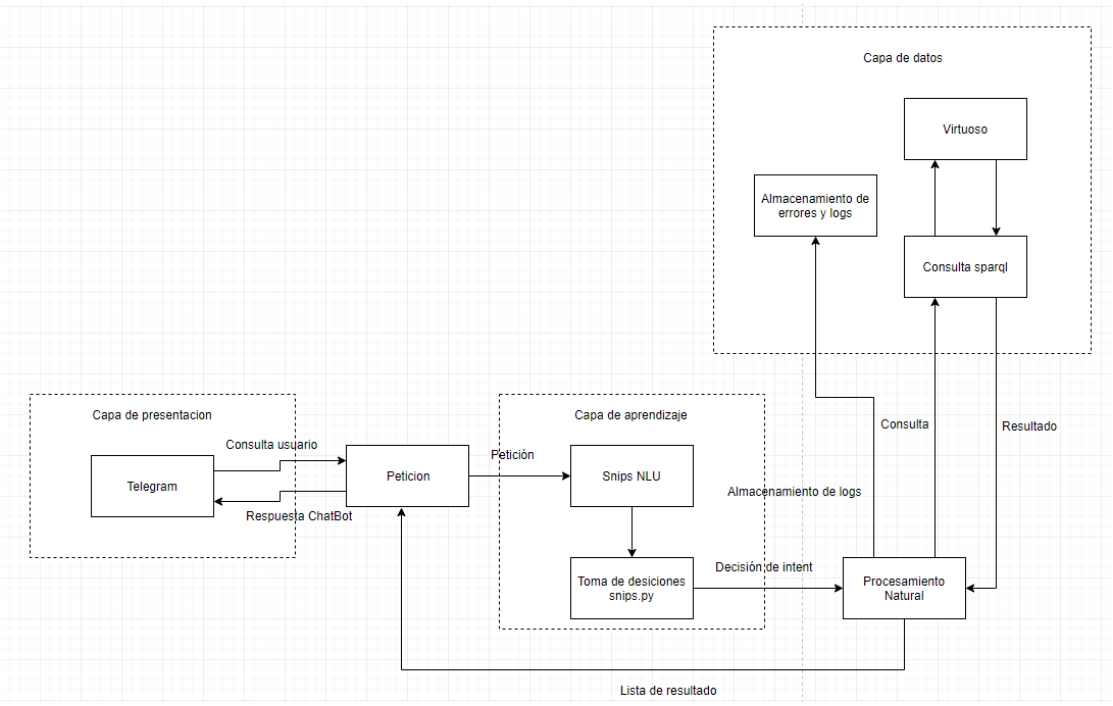
Debido a tener carrocería más fuerte y a ser carreras cortas (generalmente de entre media y una hora), los automóviles de turismo suelen tener más contacto físico que los monoplazas o los GT.

2.1.4. Aceleración

Las carreras de aceleración o picadas (drag racing en inglés) es una disciplina de automovilismo en la que generalmente se ven envueltos dos autos en una pista recta de, típicamente, 1/4 de milla o 1/8 de milla (402 y 201 metros respectivamente). La finalidad de tal carrera es llegar antes que el contrario. Esta disciplina difiere de las otras en la escasa duración de cada carrera, menos de diez segundos con los automóviles más potentes.

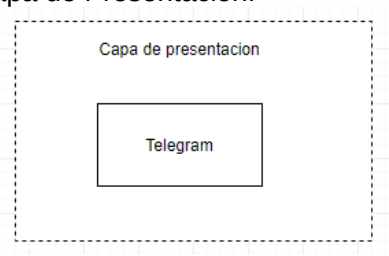
6. DESARROLLO

a) Arquitectura ChatBot

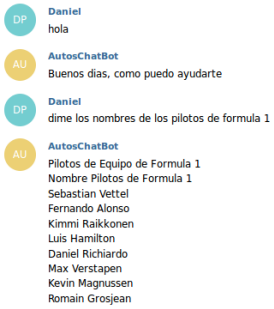


En esta arquitectura en N-capas se han diseñado 3 capas para la construcción del chatbot:

1) La capa de Presentación:

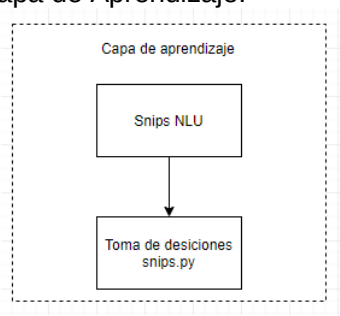


En esta capa se encuentra nuestra API de Telegram, la actuará el usuario con nuestro chatbot, aquí se encuentra representada por telegram que será la herramienta de conexión del usuario con el chatbot.



Con la utilización del API de Telegram, nuestro chatbot solo debe encontrase levantado en la máquina que tengamos el servicio o la información que se encuentra en virtuoso para poder acceder, aquí podemos acceder mediante la aplicación móvil o la aplicación web de telegram para poderlo utilizar.

2) La capa de Aprendizaje:



La capa de aprendizaje es la capa en la cual contaremos con snips nlu, la cual nos ayudara identificando la intención del usuario y posteriormente nos ayudara a seleccionar la dirección a la cual debe de ir nuestra información hacia la consulta de sparql a virtuoso que se encuentra en nuestra capa de datos o si debe de retornar un mensaje de que el chatbot no entendió la consulta hacia la capa de presentación, en caso de que el mensaje sea que no entendió le retornara un mensaje al usuario e iniciara de nuevo en la capa de presentación.

Esta capa es la más importante del proyecto, debido a que en esta capa esta la toma de decisiones que debe de tener el chatbot, al momento de enviar una petición a la capa de datos, esta envía la información necesaria para la consulta, esta capa además de enviar y recibir la información, también es la encargada de transformar nuestra información que es enviada en formado JSON a un texto entendible y legible por el usuario final.

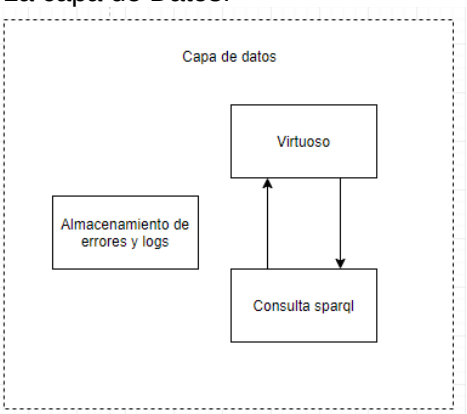
```
respuesta_json = json.loads(pregunta(texto))
print(pregunta(texto))
print('\nRESPUESTA JSON: ')
print(respuesta_json)
try:
    if (len(respuesta_json['slots'][0]['entity'])!=1):
        intencion = respuesta_json['slots'][0]['entity'] #pilotos
        probabilidad = respuesta_json['intent']['probability']
        formula1 = respuesta_json['slots'][0]['value']['value']
        equiposf1 = respuesta_json['slots'][0]['value']['value']
        campeonesf1 = respuesta_json['slots'][0]['value']['value']
        categorias = respuesta_json['slots'][0]['value']['value']
        teamF1 = respuesta_json['slots'][0]['value']['value']
        categoriasF1 = respuesta_json['slots'][0]['value']['value']
        print("esta es la intencion:")
        print(intencion)
        print("esta es la probabilidad:")
        print(probabilidad)
```

En la imagen anterior, la capa de aprendizaje realiza la descomposición de la pregunta del usuario en los diferentes Entity e intención encontrados, para el tratamiento de su información y posterior consulta.

```
if (intencion=="formula1"):
    print("entro a pilotos:")
    print(formula1)

    lista_formula1 = consulta_formula1(formula1)
    if(len(lista_formula1)!=0 or len(formula1)!=0) :

        respuesta = ""
        lista_nombres_formula1 = []

        for i in lista_formula1:
            lista_nombres_formula1.append(i)
            respuesta = respuesta + i + '\n'
    else:
        respuesta = [random.choice(listaError)]
```

Mientras que en el grafico anterior, podemos ver que de acuerdo a la intención encontrada el método ingresada a los diferentes intent en el cual se realizara la consulta SPARQL y posteriormente entrega del resultado para presentar al chatbot, además de los procesos que realiza el proceso intermedio de respuesta en el cual en caso de no entrar a ningún método o de no reconocer algún intent o Entity el mensaje que presentara al usuario es de un error.

3) La capa de Datos:



Esta capa es la encargada de almacenar toda la información que necesitemos, además de realizar las consultas a nuestro servidor en virtuoso vía HTTP, aquí la información que nos entrega la capa de aprendizaje realiza a una consulta en sparql y posteriormente a virtuoso, el cual nos retornara una lista de datos con la información solicitada por el usuario en formato JSON para que sea descompuesta por la capa de aprendizaje y luego entregada a la capa de presentación.

Aquí almacenamos la información en lenguaje RDF y se le asigna una URL para que esta pueda ser consultada de manera fácil.

```
sparql3 = SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
sparql3.setQuery("""
            select distinct ?Nombre where {
                skos:Automovilismo skos:narrower ?categorias .
                ?categorias skos:prefLabel ?Nombre .
                }
                """)
# definition
sparql3.setReturnFormat(JSON)
results3 = sparql3.query().convert()
```

Para este método primero se enlaza la URL en la cual se encuentra el conjunto de tripletas a consultar, posteriormente entre comillas se envía la consulta en formato SPARQL para así obtener un resultado y presentarle al usuario, este resultado se encuentra en formato JSON por lo cual vamos a transformarlo luego y descomponerlo para que la información necesaria y pertinente sea entregada al usuario.

Entre las consultas que hemos utilizado en cada sentencia sparql tenemos:

- Sentencia para consultar el nombre de los pilotos de fórmula 1:

```
Select ?Nombre
where
{
?Pilotos skos:narrower skos:PilotosF1 .
?Pilotos rdfs:label ?Nombre .
}
```

- Sentencia para consultar los equipos que actualmente se encuentran compitiendo en la formula 1:

```
Select ?Nombre
where
{
?EquiposF1 skos:narrower skos:NombreEquiposF1 .
?EquiposF1 rdfs:label ?Nombre .
}
```

- Sentencia para consultar el nombre de los pilotos que pertenecen a algún equipo de fórmula 1:

```
select ?Nombre
where
{
?EquiposF1 skos:narrower skos:NombreEquiposF1 .
?EquiposF1 rdfs:label ?NomEquiposF1 .
?Pilotos skos:narrower ?EquiposF1 .
?Pilotos rdfs:label ?Nombre .
Filter regex(?NomEquiposF1, "Mercedes", "i")
}
```

- Sentencia para consultar las diferentes categorías del automovilismo:

```
select ?Nombre
where
{
skos:Automovilismo skos:narrower ?categorias .
?categorias skos:prefLabel ?Nombre .
}
```

- Sentencia para consultar el nombre de los pilotos campeones de fórmula 1 que actualmente se encuentran compitiendo:

```
Select ?Nombre
where
{
?Pilotos skos:related skos:PilotosF1Campeones .
?Pilotos rdfs:label ?Nombre .
}
```

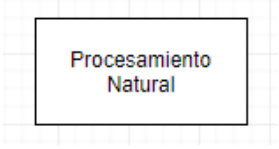
Además de ser la capa encargada de datos, es la capa que se encarga de almacenar los los de consultas y errores que han realizado con el chatbot en las consultas que ha hecho el usuario, estos errores se almacenan en un archivo en formato txt para que el programador o encargado del chatbot pueda incluir las múltiples formas de consulta de un intent y también pueda aumentar la cantidad de datos que pueda contener el Entity y ampliar la forma de realizar consultas dentro de los intents.

```
messageUsr = texto
respuestaBot = str(respuesta)

ahora = time.strftime("%c")

file = open('Conversaciones.txt', 'a')

file.write("\n"+"Usuario ID:\n")
file.write(str(chat_id)+"\n")
file.write("\n"+ahora+": Este es el mensaje del usuario:\n" + messageUsr)
file.write("\n"+ahora +": Este es el mensaje del ChatBot:\n" + respuestaBot+"\n")
file.write("------------Separacion Mensajes-------------")

file.close()
```

En el método anterior podemos encontrar la función para almacenar los logs del usuario, aquí primero declaramos las variables y las normalizamos a string para su almacenamiento, luego abrimos el archivo y comenzamos a ingresar los strings que necesitamos ayudados de un pequeño formateo para su fácil lectura para el diseñador. Aquí almacenamos el Id de la conversación, las preguntas realizadas por el usuario y la respuesta que brinda el chatbot, además de la fecha y hora que se realizó.

4) Procesos intermedios

- Proceso Procesamiento natural

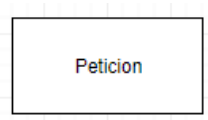


Este proceso es uno de los más importantes debido a que realiza la descomposición y almacenamiento de la información que se va a presentar al usuario, es decir en este proceso en caso de haber realizado una consulta SPARQL va a realizar la descomposición de la respuesta en formato JSON y va a realizar la extracción de la información en lenguaje natural, para asi brindar una respuesta entendible y legible para cualquier persona en otras palabras brinda una respuesta en lenguaje natural. Y en caso de que el usuario no haya realizado la consulta debido a que no se encontró la entidad o el intent dentro de su consulta, este proceso también es el encargado de dar una respuesta de error al usuario.

```
datos2 = []
#datos1 = []
for result2 in results2["results"]["bindings"]:
    datos2.append(result2["Nombre"]["value"])
    #datos1.append(datos.split("example:")) #limpio el example:
print("estos son los datos1")
#print(datos)
return datos2
```

Mediante la funciona append, nosotros recogemos la información que nos es enviada en el JSON hacia los datos encontrados en los campos de Nombre y value, para así poder ser descompuesta en una lista y posteriormente entregada a la capa de presentación.

- Proceso Petición



Este proceso es el más sencillo de todos es el que se encarga de realizar una obtención del texto de consulta del usuario para su procesamiento al igual el enviar la respuesta al usuario por parte del chatbot.

```
try:
    texto = message.json['text']
except AttributeError:
    texto = "N/A"
```

En la imagen anterior podemos encontrar el método para obtención del mensaje o consulta del usuario para el chatbot.

```
bot.send_message(chat_id, respuesta)
```

En el grafico anterior podemos ver el método o función para enviar la respuesta hacia telegram con el mensaje que da el chatbot.

b) Intents

Primero para explicar acerca de los intents que es lo primero que se crea, explicaremos el contenido y función de cada uno de los intents que hemos utilizado para este chatbot, primero contamos con 6 intents:

- Intent_autos
  En este intent hablamos acerca de las preguntas para que el chatbot pueda entrar y buscar la información acerca de los pilotos de fórmula 1 que se encuentran actualmente participando por medio de múltiples preguntas. En este intent utilizaremos la variable de

formula1 como guía para poder seguir al chatbot.

```
Dime los pilotos de [formula1:formula1](formula1)
Dame los pilotos de [formula1:formula1](formula1)
Dime los nombres de los pilotos de [formula1:formula1](formula1)
Dame los nombres de los pilotos de [formula1:formula1](formula1)
Nombres los pilotos de [formula1:formula1](formula1)
Nombre los pilotos de [formula1:formula1](formula1)
Que pilotos son de [formula1:formula1](formula1)
Nombrame los pilotos de [formula1:formula1](formula1)
Cuales son los nombres de los pilotos de [formula1:formula1](formula1)
Que nombres tienen los [formula1:formula1](formula1)
Como se llaman los pilotos de [formula1:formula1](formula1)
Quienes son los pilotos de [formula1:formula1](formula1)
Quienes compiten en la [formula1:formula1](formula1)
Indique los nombres de los pilotos de [formula1:formula1](formula1)
Mencione los nombres de los pilotos de [formula1:formula1](formula1)
Dime los competidores de [formula1:formula1](formula1)
Explicame los pilotos de [formula1:formula1](formula1)
Explicame los nombres de los pilotos de [formula1:formula1](formula1)
Muestrame el nombre de los pilotos de [formula1:formula1](formula1)
Muestrame quienes son los pilotos de [formula1:formula1](formula1)
```

- Intent_campeonesf1
  En este intent hablamos acerca de las preguntas para que el chatbot pueda entrar y buscar la información acerca de los pilotos campeones de fórmula 1 que se encuentran actualmente participando por medio de múltiples preguntas. En este intent utilizaremos la variable de campeonesf1 como guía para poder seguir al chatbot.

```
Dime los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Dame los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Dime los nombres de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Dame los nombres de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Nombres los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Nombre los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Que pilotos son de [campeonesf1:campeonesf1](campeonesf1)
Nombrame los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Cuales son los nombres de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Que nombres tienen los [campeonesf1:campeonesf1](campeonesf1)
Como se llaman los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Quienes son los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Quienes compiten en la [campeonesf1:campeonesf1](campeonesf1)
Indique los nombres de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Mencione los nombres de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Dime los competidores de [campeonesf1:campeonesf1](campeonesf1)
Explicame los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Explicame los nombres de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Muestrame el nombre de los pilotos de [campeonesf1:campeonesf1](campeonesf1)
Muestrame quienes son los pilotos de [campeonesf1:campeonesf1](campeonesf1)
```

- Intent_categorias
  En este intent hablamos acerca de las preguntas para que el chatbot pueda entrar y buscar la información acerca de las diferentes categorías de automovilismo que se encuentran estructuradas, con ayuda de varias preguntas obtendremos la información alojada en virtuoso. En este intent utilizaremos la variable de categorías como guía para poder seguir al chatbot.

```
Dime las [categorias:categorias](categorias) de automovilismo
Dame las [categorias:categorias](categorias) de automovilismo
Dime los nombres de las [categorias:categorias](categorias) de automovilismo
Dame los nombres de las [categorias:categorias](categorias) de automovilismo
Nombres de las [categorias:categorias](categorias) de automovilismo
Nombre las [categorias:categorias](categorias) de automovilismo
Que [categorias:categorias](categorias) hay de automovilismo
Que [categorias:categorias](categorias) existen de automovilismo
Nombrame las [categorias:categorias](categorias) de automovilismo
Cuales son los nombres de las [categorias:categorias](categorias) de automovilismo
Que nombres tienen las [categorias:categorias](categorias) de automovilismo
Como se llaman las [categorias:categorias](categorias) de automovilismo
Indique los nombres de las [categorias:categorias](categorias) de automovilismo
Mencione los nombres de las [categorias:categorias](categorias) de automovilismo
Mencione las [categorias:categorias](categorias) de automovilismo
Explicame las [categorias:categorias](categorias) de automovilismo
Explicame los nombres de las [categorias:categorias](categorias) de automovilismo
Muestrame el nombre de las [categorias:categorias](categorias) de automovilismo
Muestrame cuales son las [categorias:categorias](categorias) de automovilismo
Cuales son las [categorias:categorias](categorias) de automovilismo
```

- Intent_conductorGrupoF1
  En este intent hablamos acerca de las preguntas para que el chatbot pueda entrar y buscar la información acerca de los pilotos que pertenecen a ciertos equipos de fórmula 1 por medio de múltiples preguntas y con ayuda de variables avanzaremos en el programa. En este intent utilizaremos la variable de teamF1 como guía para poder seguir al chatbot y realizar las consultas a virtuoso.

```
Dime los pilotos del [teamF1:teamF1](teamF1)
Dame los pilotos del [teamF1:teamF1](teamF1)
Dime los nombres de los pilotos del [teamF1:teamF1](teamF1)
Dame los nombres de los pilotos del [teamF1:teamF1](teamF1)
Nombres los pilotos del [teamF1:teamF1](teamF1)
Nombre los pilotos del [teamF1:teamF1](teamF1)
Que pilotos del [teamF1:teamF1](teamF1)
Nombrame los pilotos del [teamF1:teamF1](teamF1)
Cuales son los nombres de los pilotos del [teamF1:teamF1](teamF1)
Que nombres tienen los pilotos del [teamF1:teamF1](teamF1)
Como se llaman los pilotos del [teamF1:teamF1](teamF1)
Quienes son los pilotos del [teamF1:teamF1](teamF1)
Indique los nombres de los pilotos del [teamF1:teamF1](teamF1)
Mencione los nombres de los pilotos del [teamF1:teamF1](teamF1)
Mencione los pilotos del [teamF1:teamF1](teamF1)
Explicame los pilotos del [teamF1:teamF1](teamF1)
Explicame los nombres de los pilotos del [teamF1:teamF1](teamF1)
Muestrame el nombre de los pilotos del [teamF1:teamF1](teamF1)
Muestrame cuales son los pilotos del [teamF1:teamF1](teamF1)
Cuales son los pilotos del [teamF1:teamF1](teamF1)
```

- Intent_equiposf1
  En este intent hablamos acerca de las preguntas para que el chatbot pueda entrar y buscar la información acerca de los diferentes equipos de formula 1que se encuentran en la presente edición del campeonato, con ayuda de múltiples preguntas y con ayuda de variables avanzaremos en el programa. En este intent utilizaremos la variable de equiposf1 como guía para poder seguir al chatbot y realizar las consultas a virtuoso.

```
bime los [equiposf1:equiposf1](equiposf1)
Dame los [equiposf1:equiposf1](equiposf1)
Dime los nombres de los [equiposf1:equiposf1](equiposf1)
Dame los nombres de los [equiposf1:equiposf1](equiposf1)
Nombres los [equiposf1:equiposf1](equiposf1)
Nombre los [equiposf1:equiposf1](equiposf1)
Que [equiposf1:equiposf1](equiposf1)
Nombrame los [equiposf1:equiposf1](equiposf1)
Cuales son los nombres de los [equiposf1:equiposf1](equiposf1)
Que nombres tienen los [equiposf1:equiposf1](equiposf1)
Como se llaman los [equiposf1:equiposf1](equiposf1)
Quienes son los [equiposf1:equiposf1](equiposf1)
Indique los nombres de los [equiposf1:equiposf1](equiposf1)
Mencione los nombres de los [equiposf1:equiposf1](equiposf1)
Mencione los [equiposf1:equiposf1](equiposf1)
Explicame los [equiposf1:equiposf1](equiposf1)
Explicame los nombres de los [equiposf1:equiposf1](equiposf1)
Muestrame el nombre de los [equiposf1:equiposf1](equiposf1)
Muestrame cuales son los [equiposf1:equiposf1](equiposf1)
Cuales son los [equiposf1:equiposf1](equiposf1)
```

```
def consulta_teamF1(teamF1):

    sparql4 = SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
    sparql4.setQuery("""
                select distinct ?Nombre where {
                    ?EquiposF1 skos:narrower skos:NombreEquiposF1 .
                    ?EquiposF1 skos:prefLabel ?NomEquiposF1 .
                    ?Pilotos skos:narrower ?EquiposF1 .
                    ?Pilotos rdfs:label ?Nombre .
                    Filter regex(?NomEquiposF1, '"""+teamF1+"""', "i")
                    }
                    """)
    # definition
    sparql4.setReturnFormat(JSON)
    results4 = sparql4.query().convert()

    print(results4)
    datos4 = []
    #datos1 = []
    for result4 in results4["results"]["bindings"]:
        datos4.append(result4["Nombre"]["value"])
        #datos1.append(datos.split("example:")) #limpio el example:
    print("estos son los datos1")
    #print(datos)
    return datos4
```

- Intent_saludo
  Este es el intent más básico que disponemos actualmente, aquí simplemente obtenemos algunas de las formas usuales de saludos por los que una persona puede entrar al chatbot y obtener una respuesta.

```
[saludo:saludo](Hola)
[saludo:saludo](Hola) como estas
[saludo:saludo](Hola) que tal
```

c) Entitys

En este apartado se explicará acerca de las entitys que son las referencias que disponemos para que nuestro intent pueda entender múltiples formas de entrar a la búsqueda, aquí se escribirán las posibles formas que el usuario tiene para escribir, además de los sinónimos que se puede tener de cada palabra. Aquí contamos con 1 Entity por cada intent para poder realizar una modificación más sencilla de cada interacción del usuario con el intent.

d) Snips

Ahora hablaremos acerca de lo que es el archivo snips y su contenido, en este archivo podemos encontrar todas las conexiones que dispone nuestro chatbot con virtuoso para realizar las consultas, agrupar las respuestas para que luego puedan ser descompuestas en el chatbot. Aquí ejecutamos las sentencias en lenguaje SPARQL, las cuales son modificadas o seleccionadas dependiendo del intent por el cual estemos ingresando al intent.

En el grafico anterior podemos observar que la sentencia se encuentra dentro de la función de setQuery, posteriormente en la función de SetReturnFormar estamos transformando el resultado de nuestra consulta en virtuoso a formado JSON, para luego con la ayuda del for descomponerlo y almacenarlo en una lista para luego descomponer la lista y obtener la información que necesitamos retornar al usuario.

e) ChatBot

En este archivo tenemos principalmente la obtención de las variables que estamos utilizando en los intents, para luego obtener mediante la librería snips la intención del usuario al entrar al chatbot y poder interpretar la consulta que pide, posteriormente tras una comparación de las intenciones el usuario ira al archivo snips para realizar la consulta y retornar un resultado, este resultado es descompuesto y luego retornado al usuario.

```
if (intencion=="formula1"):
    print("entro a pilotos:")
    print(formula1)
    lista_formula1 = consulta_formula1(formula1)

    respuesta = ""
    lista_nombres_formula1 = []

    for i in lista_formula1:
        lista_nombres_formula1.append(i)
        respuesta = respuesta + i
```

En el grafico anterior, podemos ver como mediante la comparación de la intención con una cadena el usuario accede y envía una petición al archivo snips para que retorne el resultado de la consulta, esta respuesta es almacenada en una lista para luego ser descompuesta y así pueda ser presentada y retornada al usuario, aquí también en caso de que el chatbot no reconozca la

pregunta el usuario obtendrá un resultado que la pregunta no ha sido entendida explicando así que no tiene información sobre el tema.

f) API Telegram

Con ayuda de esta API gratuita nosotros realizamos la conexión de nuestro chatbot con la red social de mensajería Telegram, esta nos permitirá con la creación de un chatbot realizar las interacciones del usuario con nuestro chatbot para realizar las consultas de información.

7. Conclusiones

- Snips nlu es una herramienta muy potente que permite al desarrollador librarse de implementar múltiples métodos para cumplir con cada una de las ramas del chatbot, gracias a esta herramienta el programador debe de dedicarse a las tareas de establecimiento del chatbot aprovechando todas las funciones que snips ofrece, con eso el desarrollador reduce en gran cantidad las horas necesitadas para establecer un chatbot
- Telegram junto a snips es un arma muy fuerte para establecer un chatbot practico sin una dificultas muy elevada, bajo conocimientos moderados de Python, y funciones generales el programador puede implementar sin mayor dificultad un chatbot.
- Snips junto a virtuoso forman un conjunto avanzado de recursos para el almacenamiento de información, para su consulta por medio de sparql y el retorno al chatbot de información.
- Para implementar un software como un chatbot, se necesita un conocimiento moderado del lenguaje de Python, ya que la librería principal utilizada aquí (snips) se encuentra codificada bajo Python, el comprender esta herramienta, al igual que el llamado de funciones y programación es primordial para utilizar y aprovechar las funciones que nos da esta herramienta.
- Las funcionalidades de un chatbot son infinitas y gracias al alcance que nos aporta una herramienta como telegram en la mensajería el esparcimiento de la información almacenada en virtuoso no tendría un límite establecido, el permitir a las personas obtener información de temas concretos que se encuentran en la dbpedia o en un lenguaje RDF facilita la localización de la información crítica en un grafo, permitiendo consultas específicas y retorno completo de la información requerida por el usuario.
- Link en GitTaw para descargar el repositorio: https://git.taw.utpl.edu.ec/dppenarreta/ChatBot_Automovilismo_con_Linked_Data

V. REFERENCIAS

[1] A. Yanes, «Datosconinteligencia.blogspot.com,» [En línea]. Available: http://datosconinteligencia.blogspot.com/2010/08/que-es-skos-simple-knowledge.html.

[2] A. Tienda, «Taxonomías en el Open Data: SKOS ¿Y eso qué es?????,» Blog de Alfonso Tienda, [En línea]. Available: https://viviendo20.wordpress.com/2015/10/27/taxonomias-en-el-open-data-skos-y-eso-que-es/.

[3] W3, «W3.org,» SKOS Simple Knowledge Organization System Primer, [En línea]. Available: https://www.w3.org/TR/skos-primer/.

[4] S. A. C. Bukhari, «Ahmad Chan,» 19 Febrero 2014. [En línea]. Available: https://ahmedchan.wordpress.com/2014/02/19/virtuoso-servers-installation-setup-data-upload-and-querying-on-windows/.

[5] K. Idehen, «Introducing OpenLink Virtuoso,» *OpenLink Software*, p. 35, 2018.

[6] J. Dureau, «Planeta ChatBot,» Planeta ChatBot, 28 Marzo 2018. [En línea]. Available: https://planetachatbot.com/alternativa-de-codigo-abierto-privada-por-diseno-para-dialogflow-amazon-lex-y-otros-e561c28bb9cb.

[7] J. Dureau, «chatbots magazine,» 28 Marzo 2018. [En línea]. Available: https://chatbotsmagazine.com/snips-nlu-is-an-open-source-private-by-design-alternative-to-dialogflow-amazon-lex-and-other-nlu-af12933b579d.

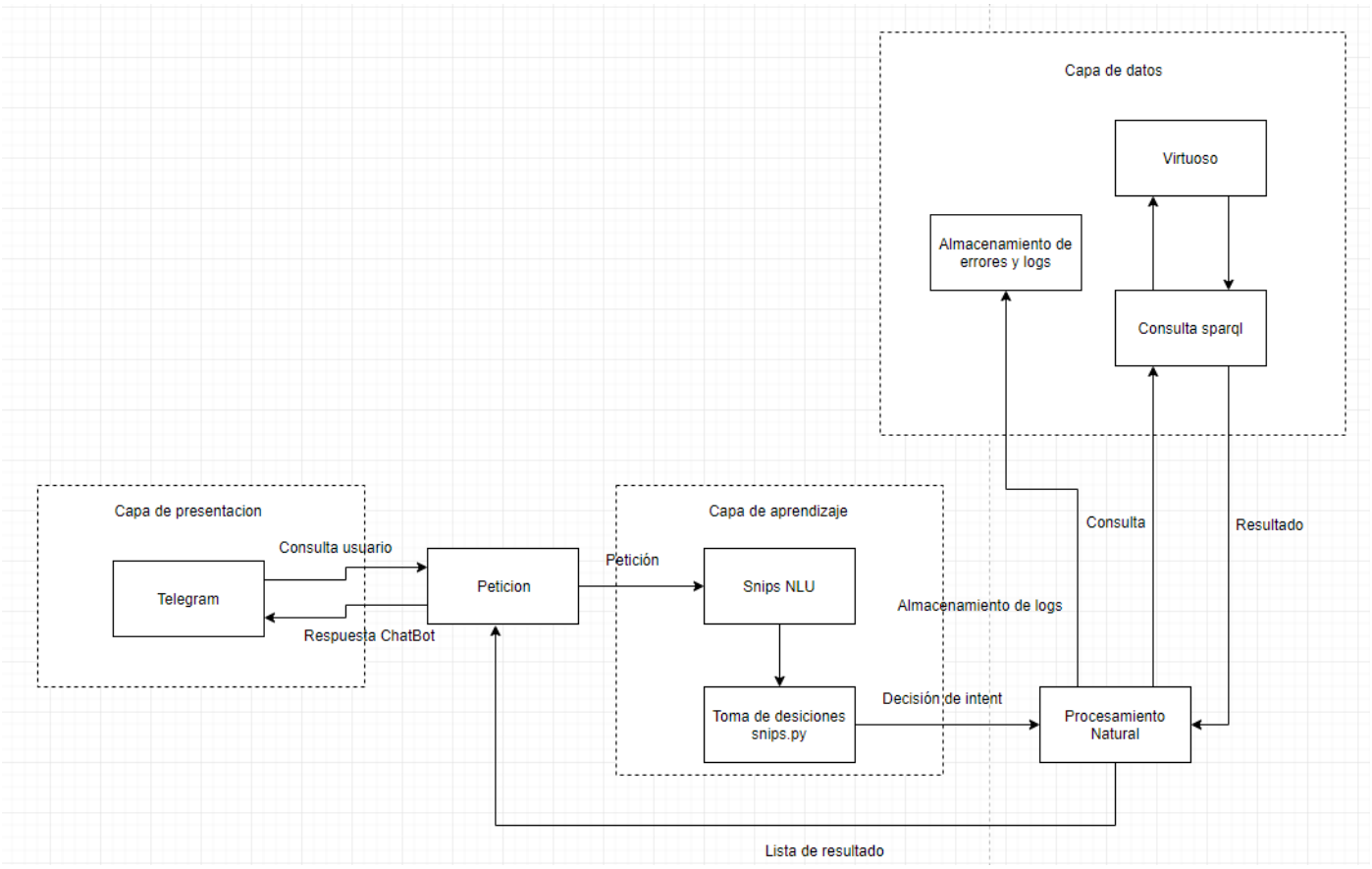
[8] A. Vidhya, «analyticsvidhya,» 01 2018. [En línea]. Available: https://www.analyticsvidhya.com/blog/2018/01/faq-chatbots-the-future-of-information-searching/.

[9] S. M. Talim, «Creating a Bot using the Telegram Bot API,» Bot Tutorials, 14 septiembre 2016. [En línea]. Available: https://tutorials.botsfloor.com/creating-a-bot-using-the-telegram-bot-api-5d3caed3266d.

[10] R. y. D. Secretaría de Cultura, «Secretaría de Cultura, Recreación y Deporte,» Automovilismo, [En línea]. Available: https://www.culturarecreacionydeporte.gov.co/es/bogotanitos/recreaccion/automovilismo.

ANEXOS

- **Arquitectura del chatbot:**



- **Código del chatbot:**

autobot.py

```
'''

https://www.youtube.com/watch?v=4fcDku71LLY
https://github.com/eternnoir/pyTelegramBotAPI

'''

import json

from snips_prueba import pregunta, consulta_formula1,
consulta_EquiposF1, consulta_CampeonesF1, consulta_Categorias,
consulta_teamF1

import telebot
import random

bot =
```

```
telebot.TeleBot("613371410:AAGlohX_39XHYp1oifk4WxGgaLeDUczdYig")

@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
    print(message)
    bot.reply_to(message, "Howdy, how are you doing?")

@bot.message_handler(func=lambda message: True)
def echo_all(message):
    print("message")
    print(message)
    #bot.reply_to(message, message.text)

    chat_id = message.chat.id
    try:
        texto = message.json['text']
    except AttributeError:
        texto = "N/A"

    print("Este es el texto")
    print(texto)
    lista = ["No entendi la pregunta", "Puedes repetir la
pregunta", "Indicame la pregunta nuevamente", "Dime de nuevo lo
que pides", "Vuelve a ingresar tu pregunta porfavor"]
    listaError = ["Valor ingresado invalido", "El valor que
ingresaste no existe", "Ingresa un valor valido por favor",
"Intenta nuevamente", "No entendi el valor"]
    listaSaludo = ["Hola, como puedo ayudarte", "Hola como estas,
como puedo ayudarte", "Buenos dias, como puedo ayudarte", "Hola,
como puedo ayudarte", "Saludos amigo, como puedo ayudarte"]
    listaDespedida = ["Nos vemos", "Adios", "Hasta otra ocacion",
"Bye, hasta otro rato", "Nos vemos amigo", "hasta otro momento"]
    respuesta_json = pregunta(texto)
    respuesta_json = json.loads(respuesta_json)

    #print(type(respuesta_json))
    print(respuesta_json)
    try:
        if (len(respuesta_json['slots'][0]['entity'])!=1):
            intencion = respuesta_json['slots'][0]['entity']
#pilotos
            formula1 =
respuesta_json['slots'][0]['value']['value']
            equiposf1 =
respuesta_json['slots'][0]['value']['value']
            campeonesf1 =
```

```
respuesta_json['slots'][0]['value']['value']
            categorias =
respuesta_json['slots'][0]['value']['value']
            teamF1 = respuesta_json['slots'][0]['value']['value']
            print("esta es la intencion:")
            print(intencion)

            if (intencion=="formula1"):
                print("entro a pilotos:")
                print(formula1)

                lista_formula1 = consulta_formula1(formula1)
                if(len(lista_formula1)!=0 or len(formula1)!=0) :

                    respuesta = ""
                    lista_nombres_formula1 = []

                    for i in lista_formula1:
                        lista_nombres_formula1.append(i)
                        respuesta = respuesta + i + '\n'
                else:
                    respuesta = [random.choice(listaError)]

            elif (intencion=="equiposf1"):
                print("entro a equipos:")
                print(equiposf1)

                lista_equiposf1 = consulta_EquiposF1(equiposf1)
                if(len(lista_equiposf1)!=0 or len(equiposf1)!=0) :
                    respuesta = ""
                    lista_nombres_campeonesf1 = []
                    respuesta = ""
                    lista_nombres_equiposf1 = []

                    for i in lista_equiposf1:
                        lista_nombres_equiposf1.append(i)
                        respuesta = respuesta + i  + '\n'
                else:
                    respuesta = [random.choice(listaError)]

            elif (intencion=="campeonesf1"):
                print("entro a campeones:")
                print(campeonesf1)

                lista_campeonesf1 =
consulta_CampeonesF1(campeonesf1)
```

```
                if(len(lista_campeonesf1)!=0 or
len(campeonesf1)!=0) :

                    respuesta = ""
                    lista_nombres_campeonesf1 = []

                    for i in lista_campeonesf1:
                        lista_nombres_campeonesf1.append(i)
                        respuesta = respuesta + i + '\n'

                else:
                    respuesta = [random.choice(listaError)]

            elif (intencion=="categorias"):
                print("entro a categorias:")
                print(categorias)

                lista_categorias = consulta_Categorias(categorias)
                if(len(lista_categorias)!=0 or len(categorias)!=0)
:
                    respuesta = ""
                    lista_nombres_categorias = []

                    for i in lista_categorias:
                        lista_nombres_categorias.append(i)
                        respuesta = respuesta + i + '\n'
                else:
                    respuesta = [random.choice(listaError)]

            elif (intencion=="teamF1"):
                print("entro a teamF1:")
                print(teamF1)

                lista_teamF1 = consulta_teamF1(teamF1)
                if(len(lista_teamF1)!=0 or len(teamF1)!=0) :
                    respuesta = ""
                    lista_nombres_teamF1 = []

                    for i in lista_teamF1:
                        lista_nombres_teamF1.append(i)
                        respuesta = respuesta + i + '\n'
                else:
                    respuesta = [random.choice(listaError)]

            elif (intencion=="saludo"):
```

```
                respuesta = [random.choice(listaSaludo)]

            elif (intencion=="despedida"):

                respuesta = [random.choice(listaDespedida)]

            else:
                respuesta = [random.choice(lista)]
        else:
            respuesta = [random.choice(lista)]
            bot.send_message(chat_id, respuesta)
            #send_message(chat_id, text)

    except IndexError:
        respuesta = [random.choice(lista)]

    bot.send_message(chat_id, respuesta)
    #send_message(chat_id, text)

print "Bot Iniciado =)"
bot.polling()
```

snips_prueba.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

import sys

reload(sys)
sys.setdefaultencoding('utf8')
import snips_nlu

snips_nlu.load_resources("es")


'''
1)
(venv) calosh@chigo ~/PycharmProjects/SBC/chatbot $ snips-nlu
generate-dataset es intent_enfermedad.txt intent_saludo.txt
entity_enfermedad.txt > dataset.json

2) Ejecutar Entrenamiento
```

```
'''

import io
import json
from snips_nlu import SnipsNLUEngine, load_resources


with io.open("trained.json") as f:
    engine_dict = json.load(f)

engine = SnipsNLUEngine.from_dict(engine_dict)

#phrase = raw_input("Pregunta: ")


def pregunta(phrase):
    r = engine.parse(unicode(phrase))
    return json.dumps(r, indent=2)
    #print(json.dumps(r, indent=2))



from SPARQLWrapper import SPARQLWrapper, JSON


def consulta_formula1(formula1):

    sparql =
SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
    sparql.setQuery("""
                select distinct ?Nombre where {
                    ?Pilotos skos:narrower skos:PilotosF1 .
                    ?Pilotos rdfs:label ?Nombre .
                    }
                    """)
    # definition
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    print(results)
    datos = []
    #datos1 = []
    for result in results["results"]["bindings"]:
        datos.append(result["Nombre"]["value"])
        #datos1.append(datos.split("example:")) #limpio el
example:
```

```
    print("estos son los datos")
    #print(datos)
    return datos

def consulta_EquiposF1(equiposf1):

    sparql1 =
SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
    sparql1.setQuery("""
                select distinct ?Nombre where {
                    ?EquiposF1 skos:narrower skos:NombreEquiposF1
.
                    ?EquiposF1 rdfs:label ?Nombre .
                    }
                    """)
    # definition
    sparql1.setReturnFormat(JSON)
    results1 = sparql1.query().convert()

    print(results1)
    datos1 = []
    #datos1 = []
    for result1 in results1["results"]["bindings"]:
        datos1.append(result1["Nombre"]["value"])
        #datos1.append(datos.split("example:")) #limpio el
example:
    print("estos son los datos1")
    #print(datos)
    return datos1

def consulta_CampeonesF1(campeonesf1):

    sparql2 =
SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
    sparql2.setQuery("""
                select distinct ?Nombre where {
                    ?Pilotos skos:related skos:PilotosF1Campeones
.
                    ?Pilotos rdfs:label ?Nombre .
                    }
                    """)
    # definition
    sparql2.setReturnFormat(JSON)
    results2 = sparql2.query().convert()

    print(results2)
```

```
    datos2 = []
    #datos1 = []
    for result2 in results2["results"]["bindings"]:
        datos2.append(result2["Nombre"]["value"])
        #datos1.append(datos.split("example:")) #limpio el
example:
    print("estos son los datos1")
    #print(datos)
    return datos2

def consulta_Categorias(categorias):

    sparql3 =
SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
    sparql3.setQuery("""
                select distinct ?Nombre where {
                    skos:Automovilismo skos:narrower ?categorias .
                    ?categorias skos:prefLabel ?Nombre .
                    }
                    """)
    # definition
    sparql3.setReturnFormat(JSON)
    results3 = sparql3.query().convert()

    print(results3)
    datos3 = []
    #datos1 = []
    for result3 in results3["results"]["bindings"]:
        datos3.append(result3["Nombre"]["value"])
        #datos1.append(datos.split("example:")) #limpio el
example:
    print("estos son los datos1")
    #print(datos)
    return datos3

def consulta_teamF1(teamF1):

    sparql4 =
SPARQLWrapper("http://localhost:8890/sparql/AutomovilismoNuevo")
    sparql4.setQuery("""
                select distinct ?Nombre where {
                    ?EquiposF1 skos:narrower skos:NombreEquiposF1
.
                    ?EquiposF1 skos:prefLabel ?NomEquiposF1 .
                    ?Pilotos skos:narrower ?EquiposF1 .
                    ?Pilotos rdfs:label ?Nombre .
```

```
                    Filter regex(?NomEquiposF1, '"""+teamF1+"""',
"i")
                    }
                    """)
    # definition
    sparql4.setReturnFormat(JSON)
    results4 = sparql4.query().convert()

    print(results4)
    datos4 = []
    #datos1 = []
    for result4 in results4["results"]["bindings"]:
        datos4.append(result4["Nombre"]["value"])
        #datos1.append(datos.split("example:")) #limpio el
example:
    print("estos son los datos1")
    #print(datos)
    return datos4
```

entrenamiento.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

import sys

reload(sys)
sys.setdefaultencoding('utf8')


import json
from snips_nlu import load_resources, SnipsNLUEngine
from snips_nlu.default_configs import CONFIG_ES

import io

load_resources("es")

with io.open("dataset.json") as f:
    dataset = json.load(f)

engine = SnipsNLUEngine(config=CONFIG_ES)
```

```
engine.fit(dataset)

engine_json = json.dumps(engine.to_dict())
with io.open("trained.json", mode="w") as f:
    f.write(unicode(engine_json))
```