Práctica 3.7

Procesamiento del lenguaje natural

Introducción a la creación de chatbots con RASA

En esta práctica tiene por objetivo tener una primera toma de contacto con el entorno de creación de chatbots conocido como RASA.

SESIÓN 1	**1. Instalación de RASA** a. En entornos virtuales (Python) b. En Docker **2. Ejecución y comandos más útiles** **3. Estructura de un proyecto RASA** Ejercicio 1: Primera toma de contacto con RASA Ejercicio 2: Creación de un chatbot (en español)
SESIÓN 2	Ejercicio 3: Manejar desvíos en la conversación **4. Ampliar la funcionalidad del chatbot** **4.1. Acciones** ("Custom actions") Ejercicio 4: Acceso a una API
SESIÓN 3	**4.2. Formularios ("Forms")** **4.2.1. Ranuras ("Slots")** Ejercicio 5: Primera prueba con los "slots" **4.2.2. Uso básico de formularios** Ejercicio 6: Ser profesor del curso de IA **4.2.3. Uso avanzado de formularios** Ejercicio 7: Salir de un formulario **5. Integración web de RASA**

1. Instalación de RASA

La página oficial de RASA indica los pasos a realizar para su instalación dependiendo del sistema operativo en uso. Existen dos formas de crear un entorno para el desarrollo de proyectos en RASA.

a. Mediante entornos virtuales (Python)

En este caso, debes contar con una VM con Linux. En "Aules" dispones de una VM con Ubuntu 20.04 en la que RASA ya está previamente instalado. Otra opción, si dispones de Windows o Mac, es crear el entorno virtual en en el sistema de ficheros del sistema operativo base. En este caos, debes contar con Python. De una forma u otra, la mejor forma de correr RASA es hacerlo en un entorno virtual de Python. Asegúrate antes de realizar los siguientes pasos que tienes acceso a internet.

1) Empezaremos creando el entorno de Python dónde correrá RASA:

```
# sudo mkdir rasa_prueba
# cd rasa_prueba
# python3 -m venv /home/mia/rasa_prueba/          // creamos el entorno …
# source /home/mia/rasa_prueba/bin/activate       // … y lo activamos
```

2) Actualizamos la versión del paquete "pip" e instalamos el de RASA

```
# sudo python -m pip install --upgrade pip rasa
```

Si tuvieras problemas con la instalación anterior, la VM cuenta con un entorno dependiente de la carpeta "rasaprojects" donde puedas ejecutar RASA tal y como se describe a continuación.

b. Mediante Docker

Una forma alternativa de trabajar con RASA es haciendo uso de contenedores Docker. Si no has usado Docker previamente, revisa antes de seguir este breve **tutorial** con lo básico que se puede hacer en Docker.

1) Antes de seguir, instala Docker (las primeras líneas instalan los paquetes necesarios):

```
# sudo apt install apt-transport-https ca-certificates curl software-
properties-common
```

```
# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
# echo "deb [signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
# sudo apt update
# sudo apt install docker-ce docker-ce-cli containerd.io
```

2) Comprueba que Docker funciona correctamente, comprobando, primero, la versión, y después lanzando un contenedor a partir de la imagen "hello-world".

```
# docker –version
# sudo docker run hello-world
```

3) Crea un directorio con el nombre "rasa_docker" y Lanza el siguiente comando para crear un contenedor de RASA con la versión 2.0.

```
# mkdir rasa_docker
# cd rasa_docker
# sudo docker run –u $(id –u) -v $(pwd):/app rasa/rasa:3.0.0-full init --no-prompt
```

¿Qué significa este comando?

- **-v $(pwd):/app** monta el directorio de trabajo actual en el directorio de trabajo del contenedor de Docker. Esto significa que los archivos que cree en su computadora serán visibles dentro del contenedor y los archivos creados en el contenedor se sincronizarán de nuevo con su computadora. *rasa/rasa* es el nombre de la imagen de Docker que se va a ejecutar, en estes caso, '2.0.0-full', que especifica la versión y las dependencias.
- la imagen de Docker tiene el comando como punto de entrada, con lo que basta con introducir **# rasa init**

4) Introduce el siguiente comando para interactuar con el chabot:

```
# sudo docker run -it -v $(pwd):/app rasa/rasa:3.0.0-full shell
```

5) Si se modifica algún dato de entrenamiento o se edita el archivo "config.yml", tendrás que volver a entrenar el modelo Rasa.

```
# sudo docker run –u $(id –u) -v $(pwd):/app rasa/rasa:3.0.0-full train --domain domain.yml --data data --out models
```

Esto es lo que está sucediendo en ese comando:

a. **–u $(id –u):** toma como usuario el usuario actual.
b. **-v $(pwd):/app**: monta el directorio del proyecto en el contenedor de Docker para que Rasa pueda entrenar un modelo en los datos de entrenamiento
c. "**rasa/rasa:2.0.0-full":** usa la imagen de Rasa con la etiqueta '2.0.0-full'
d. **train**: ejecuta el comando dentro del contenedor.
e. **--domain domain.yml:** pasa este fichero al contenedor.
f. **--data data:** pasa el contenido del directorio "data" al contenedor.
g. **--out models:** indica qué directorio recibe el fichero con el modelo generado.

2. Ejecución y comando más útiles

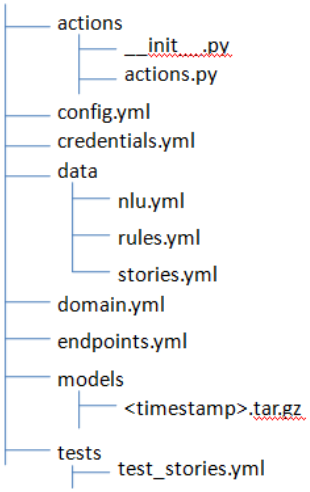
Para ejecutar RASA contamos con el comando "rasa".

1) Ejecuta en el terminal **# rasa – version**, para comprobar que el comando funciona
2) Existen 3 opciones básicas de uso del comando "rasa":

Comando	Cometido	Descripción
# rasa init	**Inicialización del entorno**	Mediante esta opción lanzamos por primera vez el RASA, donde se pregunta, primero, cuál es el directorio con los ficheros de configuración (es decir, el proyecto en sí), y después, si queremos entrenar el sencillo modelo que viene con la instalación de base. Además, se nos pregunta si queremos entrenar el modelo. A ambas preguntas podemos contestar afirmativamente. Tras ello, el entorno nos brindará un ***prompt*** mediante el cual podemos actuar con el chatbot por defecto.
# rasa shell	**Ejecución del entorno**	Tras haber ejecutado un "rasa init" por primera vez, la forma de interactuar con el terminal del chatbot pasa por lanzar "rasa shell".
# rasa train	**Entrenamiento del modelo**	Debes ejecutar este comando después de haber configurado tu proyecto y haber añadido datos de entrenamiento o haber modificado el modelo de entrenamiento para que el modelo de Rasa pueda aprender de estos datos.
# rasa run actions	**Manejo de acciones personalizadas**	Este comando se utiliza para ejecutar el servidor de acciones personalizadas. Si el chatbot tiene acciones personalizadas que necesitan ser manejadas por código Python, este servidor debe estar activo para que el chatbot funcione correctamente.
# rasa interactive	**Entrenamiento interactivo**	Permite interactuar con el chabot para entrenarlo paso a paso dentro de un modo de ejecución interactivo

3. Estructura de un proyecto

Un proyecto en el entorno de RASA cuenta con la siguiente estructura de directorios y ficheros de configuración:



Descripción del propósito de cada fichero:

- **__init__.py: A**rchivo vacío que ayuda a Python a localizar tus acciones.
- **actions.py:** Este archivo se utiliza para crear acciones personalizadas. Si se deseas llamar a un servidor externo u obtener datos de API externos, se pueden definir las acciones asociadas en este fichero.
- **config.yml:** Este archivo define la configuración de la NLU y el modelo principal. Si se está utilizando algún modelo fuera del modelo NLU, se debe definir la canalización aquí.
- **credentials.yml:** Este archivo se utiliza para almacenar credenciales para conectarse a servicios externos como Facebook Messenger, Slack, etc
- **data/nlu.md**: En este archivo definimos nuestras "intenciones" *(intents)*, es decir, lo que indicamos al bot que haga). Estas intenciones se utilizan para entrenar el modelo NLU.
- **data/stories.md:** Las historias son la conversación de muestra entre un usuario y un bot en forma de intenciones, respuestas y acciones. Las historias no son más que datos de entrenamiento que se utilizan para entrenar los modelos de gestión del diálogo de Rasa.

- **domain.yml:** Este archivo enumera las diferentes intenciones junto con las respuestas del bot y las acciones que puede realizar.
- **endpoints.yml:** Define los detalles para conectar canales como Slack, Facebook messenger, etc. para almacenar datos de chats en las bases de datos en línea como Redis, etc.
- **models/<timestamps>.tar.gz:** el modelo inicial, todos los modelos entrenados almacenados en la carpeta models. Para volver a entrenar el modelo, usamos el comando "rasa train".

> Ejercicio 1 : Primera toma de contacto con RASA

Realiza los siguientes pasos para practicar la operativa básica con RASA:

1) **Instala RASA** siguiendo uno de los dos métodos descritos en el apartado anterior.
   Si tienes dudas en algún punto, este breve **vídeo** da una idea de cómo debe realizarse el proceso.

2) **Lanza RASA por primera vez con el comando "# rasa init".**
   RASA toma como proyecto de referencia el conjunto de ficheros del directorio desde el cual se lanza el comando. La primera vez que lo lances se te preguntará cuál es tal directorio, además de si se desea entrenar el modelo, a la cual cabe responder afirmativamente. Cuando finalice el arranque de la aplicación, aparecerá un prompt que permite interactuar con el chatbot (puedes salir mediante "/stop").

3) **Modifica el contenido del fichero "domain.yml"** para que en la respuesta que el chatbot da por defecto a un estado de ánimo positivo no sugiera una imagen (es decir, cambia la línea correspondiente a "*image: https://i.imgur.com/nGF1K8f.jpg* por una frase que se te ocurra)

4) **Entrena el modelo** para que registre los cambios mediante el comando "**# rasa train**".

5) **Ejecuta RASA** mediante el comando "**# rasa shell**" y comprueba que el chatbot ha incluido el comportamiento anterior.

6) Finalmente, vamos a **probar el chatbot en modo de entrenamiento interactivo**.
   Esto permite ejecutar el chatbot y comprobar si la predicción de los algoritmos funciona correctamente dada una entrada por parte del usuario/a. Ejecuta para ello el comando **# rasa interactive.** Debes introducir preguntas relacionadas con los cambios que hemos hecho anteriormente (por ejemplo, introduce cosas como "hello" al principio o de "bot", relacionados con

las intenciones ("intent") de "greet" o "bot_challenge", respectivamente). Verás que el chatbot te va preguntando si la asociación que hace de las preguntas a su predicción es correcta. Contesta afirmativa o negativamente según el caso. Para salir y guardar cambios, pulsa CTRL+C.

> Ejercicio 2 :: Creación de una versión en español

En este caso crearemos un chatbot muy similar al anterior, pero que permita interactuar con el usuario en español. Aprovecharemos además para enriquecer la conversación. La creación de un chatbot en español requiere de ciertos cambios en el fichero de configuración **config.yml** donde debemos indicar el modelo y técnicas en el que se basará el chatbot en su funcionamiento.

Comprueba que en tu caso tienes algo similar a la siguiente configuración:

```
# The config recipe.
# https://rasa.com/docs/rasa/model-configuration/
recipe: default.v1

# The assistant project unique identifier
# This default value must be replaced with a unique assistant name within your deployment
assistant_id: 20231116-162944-wide-empowerment

language: es

pipeline:
- name: WhitespaceTokenizer
- name: RegexFeaturizer
- name: LexicalSyntacticFeaturizer
# - name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 50
  constrain_similarities: true

policies:
- name: MemoizationPolicy
- name: TEDPolicy
  max_history: 2
  epochs: 100
  constrain_similarities: true
```

Entre los parámetros anteriores hay uno fundamental para este chatbot en particular, y es la indicación de que el pipeline se aplicara al idioma español (language: "es").

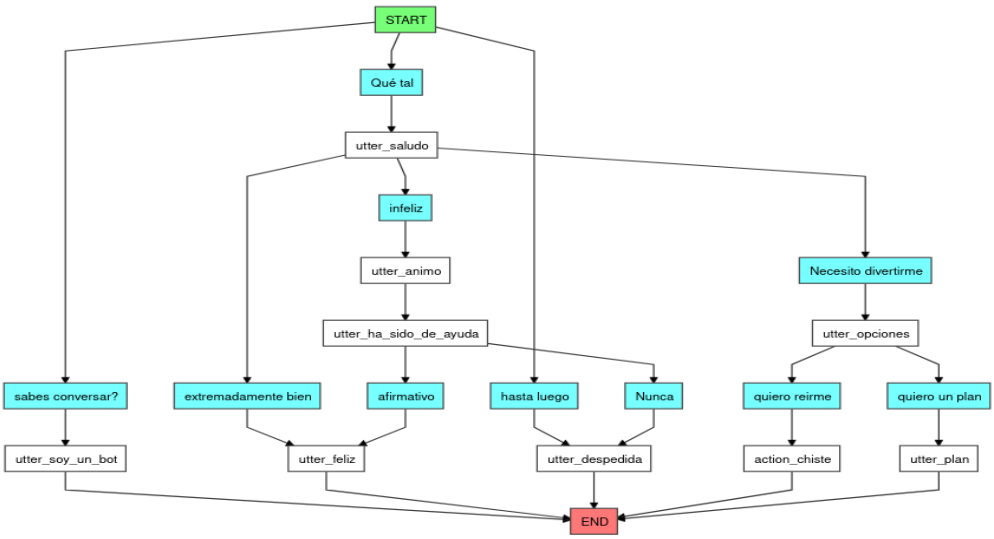
Pautas para el diseño de la conversación:

- el chatbot tendrá los siguientes comportamientos básicos del modelo por defecto: saludar, despedirse, responder ante un estado de ánimo negativo, responder ante un estado de ánimo positivo y responder si es un bot o no.
- además, el chatbot dará dos opciones entre las que el usuario deberá elegir: o bien se le contará un chiste, o bien se le sugerirá un plan para el fin de semana.

A partir de estas indicaciones, crea una carpeta nueva para albergar un nuevo proyecto, que deberás crear, y al que deberás añadir contenido, reemplazando el que tenga los ficheros **domain.yml, config.yml, nlu.md** y **stories.md. Aunque el proyecto sea diferente al que se crea por defecto, puedes usar igualmente el entorno Python creado en un primer momento.**

Recomendaciones para ajustar el comportamiento del chatbot

Cuando hayas añadido el contenido de los archivos, tras entrenar el modelo, puedes visualizar la secuencia que guiará cada flujo de conversación con el comando: **# rasa visualize.** El diagrama resultante debería aproximarse a lo siguiente:



Prueba la secuencia que guía la conversación a través de cada una de las "historias" (**stories***)* definidas. Si en una misma sesión, inicias una nueva conversación de principio a fin y no obtienes el comportamiento esperado, puedes optar por entrenar al chatbot para que integre esas variaciones. Recuerda que puedes hacerlo entrando en modo interactivo (comando **# rasa interactive**) y entrenando expresamente al chatbot.

Finalmente, **si deseas reiniciar una conversación sin tener que salir y volver a lanzar el entorno de conversación en la consola**, bien porque el comportamiento del chatbot no es el esperado, o por cualquier otrá razón, puedes reiniciar la conversación mediante el comando "/restart", para lo cual debes asegurarte de que **RulePolicy** está como política en el fichero **config.xml**.

> Ejercicio 3 : Gestión de desvíos en la conversación

Una forma de hacer más robusto el funcionamiento del chatbot es prepararlo para intervenciones por parte del usuario inesperadas o improcedentes, es decir, cuando el usuario dice algo "que no viene a cuento". RASA de diferentes mecanismos para abordar tales situaciones.

El más sencillo de ellos es el que se denomina "mensajes fuera de contexto", que encontrarás descrito en esta sección del manual oficial de RASA. Sin embargo, resulta más aconsejable por usar otro mecanismo de recuperación para obtener mejores resultados: hablamos del "FallBackClassifier" mediante el cual el chatbot puede tanto identificar entradas que no cumplan un umbral de confianza en su atribución a una

intención de la base de entrenamiento, como ofrecer una respuesta cuando esta aginación no se ha podido realizar.

Básate en el ejemplo que viene en las diapositivas (también puedes consultar el manual oficial) para realizar las modificaciones en tu chatbot que permitan manejar intervenciones del usuario inesperadas.

4. Ampliar la funcionalidad del chatbot

4.1. Acciones ("custom actions")

Para ampliar la funcionalidad del chatbot mediante "acciones" codificadas en Python, recuerda que de forma resumida se deben acometer los siguientes pasos:

a) Incorporar el código al fichero "actions.py", añadiendo las clases necesarias.
b) Añadir una mención a las acciones definidas en el fichero "domain.yml".
c) Activar el servidor de rasa-sdk mediante el comando "**# rasa run actions**".

Puedes repasar estos pasos a través de este breve vídeo en inglés.

> Ejercicio 4: acceder a una API

En este caso, cambiaremos el comportamiento del chatbot que creamos en el ejercicio 2 correspondiente a la parte donde se cuenta un chiste a petición del usuario. Hasta este momento, recordemos, el chatbot cuenta siempre el mismo chiste. Usa una acción para que el chatbot reciba en cada ejecución un chiste distinto de la API abierta publicada en honor al gran actor norteamericano Chuck Norris (**https://api.chucknorris.io/**).

4.2. Formularios ("Custom Forms")

4.2.1. Ranuras ("Slots")

Como hemos visto en clase, RASA proporciona a través de los "slots" la capacidad de guardar información durante la conversación para su uso posterior dentro de la misma. Antes de ver cómo se pueden manejar "slots" de forma más eficiente, haremos una pequeña prueba definiendo uso.

Puedes revisar el ejemplo que viene en este vídeo para refrescar ideas.

> Ejercicio 5: primera prueba con ranuras ("slots")

Modifica el contenido de los ficheros del proyecto en curso, para que el usuario pueda recordar el plan que ha elegido.

4.2.2. Uso básico de formularios

En el ejercicio previo, hemos visto como, mediante el uso de "slots", podemos guardar un valor como parte del historial de la conversación que puede ser recuperado posteriormente durante el transcurso de la misma. Sin embargo, cuando se desean guardar varios valores referidos a un mismo concepto (por ejemplo, características de una persona, de un pedido de un producto de consumo, etc.), se necesita de una estructura de datos que manejen diferentes "slots", llamada formularios ("forms").

> Ejercicio 6: Ser profesor del curso de IA

En este ejercicio haremos una sencilla prueba con formularios. El objetivo será que un usuario introduzca el nombre de una persona (primero su nombre y después su primer apellido). Acto seguido, el chatbot deberá comprobar si la persona con el nombre y apellidos proporcionados es realmente profesor curso, e informará tanto si lo es como si no por pantalla.

Pasos a realizar:

1) Crea un nuevo proyecto de RASA. Recuerda que para ello simplemente tienes que crear una nueva carpeta (puedes llamarla, por ejemplo, "rasa_forms"), activar el entorno virtual de Python que hemos manejado anteriormente (si no estuviera activo) y finalmente ejecutar el comando "# rasa init" para crear la estructura de los ficheros del proyecto.

   Nota: el proyecto se creará en inglés. Si quieres trabajar en español y partir de algún contenido mínimo, puedes copiar las intenciones ("intents") e historias ("stories") del ejercicio 2 referida a lo que puede considerar una interacción básica (por ejemplo, las referidas a saludar, despedirse, comprobar si es el chtabot es humano, etc.).

2) Revisa el ejemplo analizado en clase y añade el contenido necesario en los ficheros "nlu.md", "stories.md" y "domain.yml".

   Nota: En principio podemos pensar que los datos para los "slots" con el "nombre" y el "apellido" sean reconocidos como entidad ("entity"). En ciertos casos, no obstante, los algoritmos de aprendizaje automático que RASA usa para reconocer entidades pueden no funcionar bien. Por ello, se recomienda usar en este el tipo texto ("text") en su lugar al definir el tipo de datos de los "slots".

3) La comprobación de la identidad del usuario la deberás resolver desde "actions.py". Allí, puedes crear una lista con los nombres de los profes y comprobar si el nombre proporcionado en la conversación se encuentra en esa lista.

   Este sería un ejemplo de la conversación que debes reproducir:

```
Your input -> hola
¡Hola! ¿Cómo estás?
```

```
Your input ->  quiero saber si una persona es profesor en este instituto
De acuerdo, hagamos la comprobación. ¿Cuál es su nombre?
Your input ->  Toni
Perfecto, su nombre es Toni, ¿y su apellido?
Your input ->  Cambra
Toni Cambra es profesor del curso
Your input ->  /stop
```

Si tienes dudas de proceder, puedes revisar un ejemplo del mismo estilo en este vídeo.

4.2.3. Uso avanzado de formularios

RASA dispone de diferentes mecanismos para hacer frente a las contingencias que pueden darse en el curso de la conversación cuando se está dentro del contexto de un formulario. Por regla general, se recomienda usar reglas siempre que queramos cubrir un único caso de desvío en la conversación, mientras que se hace más recomendable definir varias historias siempre que se quieran contemplar casos como, por ejemplo, aquellos en los que el usuario interrumpe o quiere abortar la conversación.

> Ejercicio 7: Salir de un formulario

Modifica la configuración del ejercicio 6 añadiendo más de una historia que permita al usuario cancelar la conversación si así lo desea.

5. Integración web de RASA

RASA puede ser integrado en cualquier entorno web. En este caso, a modo de ejemplo, puedes probar a través de una página HTML cómo se podría ineractuar con el chatbot desde el navegador.

Pasos:

1. Crea un fichero HTML (ponle como nombre, por ejemplo, "webchat.html") y lleva el siguiente código:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Chatbot con Rasa</title>
  <style>
    #chat-container {
      max-width: 400px;
      margin: 50px auto;
      border: 1px solid #ccc;
      padding: 10px;
    }

    #chat-display {
      height: 300px;
      overflow-y: scroll;
      border-bottom: 1px solid #ccc;
      margin-bottom: 10px;
    }

    #user-input {
      display: flex;
    }

    #user-message {
      flex: 1;
      padding: 5px;
    }

    button {
      padding: 5px;
    }
```

```
  </style>
</head>
<body>
  <div id="chat-container">
    <div id="chat-display"></div>
    <div id="user-input">
      <input type="text" id="user-message" placeholder="Escribe tu mensaje...">
      <button onclick="sendMessage()">Enviar</button>
    </div>
  </div>

  <script>
    function sendMessage() {
      var userMessage = document.getElementById('user-message').value;
      var chatDisplay = document.getElementById('chat-display');

      // Mostrar el mensaje del usuario en el chat display
      chatDisplay.innerHTML += '<p><strong>Tú:</strong> ' + userMessage + '</p>';

      // Enviar mensaje al servidor de Rasa y obtener la respuesta
      fetch('http://localhost:5005/webhooks/rest/webhook', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ message: userMessage }),
      })
      .then(response => response.json())
      .then(data => {
        // Mostrar la respuesta de Rasa en el chat display
        chatDisplay.innerHTML += '<p><strong>Chatbot:</strong> ' + data[0].text + '</p>';
      });

      // Limpiar el input de usuario
      document.getElementById('user-message').value = '';
    }
  </script>
</body>
</html>
```

2. Inicia al servidor de RASA.

```
rasa run -m models --enable-api --cors "*"
```

3. Abre el archivo en el navegador y prueba el chatbot.

Fuentes para consulta

- Tutoriales generales:
    - Rasa 3.x Tutorials (short videos)
    - Conversational AI with Rasa Open Source 3.x
    - Rasa implementation & concepts
- Manejar desvíos o situaciones imprevistas en la conversación:
    - Fallback and Human Handoff
    - Handling Unexpected Input
- Acceso a recursos externos:
    - Rasa Livecoding: Connecting your Rasa Assistant to a Database
    - Enable RESTful Web API for RASA Chatbot