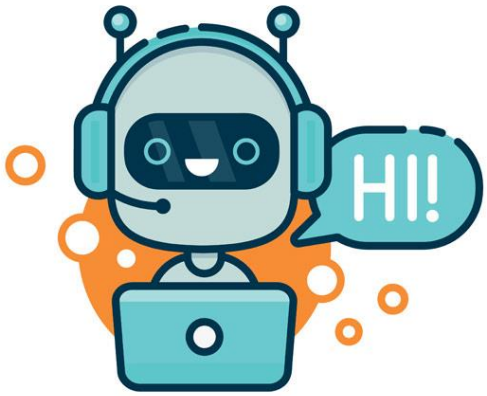


PROGRAMACIÓN EXPLORATORIA

Año 2022

DESCRIPCIÓN BREVE

Implementación de un ChatBot

que nos representa como alumnos,

con el framework de Rasa.

jcuevas@alumnos.exa.unicen.edu.ar

Índice

➢ *Introducción*

Comenzaremos con un poco de información para saber de dónde se originó esta idea de crear ChatBot; La era de la digitalización ha transformado la forma en la que los individuos se comunican e interactúan. Mientras que en el pasado las vías de comunicación eran limitadas y unidireccionales, en la actualidad existen diversos medios que han democratizado los canales de conexión e interlocución. Por ejemplo, la introducción de teléfonos celulares portátiles e inteligentes ha permitido que las personas puedan permanecer conectadas entre sí, desde cualquier lugar. A través de estos dispositivos, las personas pueden conectarse a las diferentes redes sociales que han surgido en la actualidad, como Facebook, Telegram, WhatsApp o Twitter. Dichas herramientas no sólo brindan a las personas la oportunidad de interactuar con amigos y familiares en cualquier momento, sino que han empezado a ser utilizadas por diversas empresas para mejorar algunos servicios de sus negocios. De tal forma, se han empleado herramientas tecnológicas como la Inteligencia Artificial (IA) para poder satisfacerlas. En defecto, la pregunta general de investigación del presente trabajo muestra cómo se diseña e implementa un asistente virtual (chatbot) para ofrecer atención personalizada de nuestro representante como si fuéramos nosotros (lo más humanamente posible), por medio del canal conversacionales de Telegram. Por tanto, el objetivo principal será explicar el proceso de diseño e implementación del proyecto, y en que consiste el desarrollo de nuestro chatbot que nos representa como alumnos. La implementación de este se realizó mediante el Framework de Rasa, el cual hace uso de la inteligencia artificial y nos permite crear un asistente escrito en Pithon. El cual basa su funcionamiento en dos componentes principales, NLU y Core.

- **NLU:** es la parte encargada de tomar el texto, analizarlo y descomponerlo de tal manera que el Bot comprenda el contenido del mensaje.
- **Core:** es la parte encargada de tomar decisiones y dar seguimiento a la conversación según lo que el NLU provee.

➢ Resumen

La modalidad que implemente para el desarrollo de este trabajo de un ChatBot que nos represente como alumnos fue ir agregando poco a poco empezando con las funcionalidades más básicas por ejemplo lograr que el Bot pueda reconocerte y saludarte tanto un saludo de bienvenida como de despedida, esto se modelo asi para llevar un desarrollo más organizado y también minimizar muchos errores posibles; en cambio si hacemos todo de un tirón y tratamos que funcione todo a la primera es evidente que quizás desatemos una cantidad de errores que se nos va a dificultar mucho más el desarrollo del mismo. En el desarrollo siempre se trato de naturalizar lo mas posible al Bot por ejemplo otras interacciones que el mismo puede hacer es preguntarte el estado de ánimo, y en base a que respondas (Feliz, Triste, Enojado) puede darte una atención personalizada siempre tratando de subirte el estado de ánimo y hacerte sentir bien o por lo menos intentarlo.

Lo más interesante del Bot es haber vinculado las APIs de Wikipedia y de Google Calendar; lo que me permite la primera es poder buscar información relacionada a la materia accediendo a la base de datos de Wikipedia y de allí extraer la información que el usuario solicito por mensaje; y la segunda lo que me permite es poder tener organizado la agenda de reuniones en mi cuanta de Google Calendar, asi tener organizada todas las fechas, tener un recordatorio de las mismas y poder cuando quiera ver los 10 próximos eventos que se encuentren agendados.

➢ Desarrollo

A continuación, se muestra todo lo que puede hacer el Bot a detalle, mostrando las capturas del NLU que son las interacciones que puede hacer el usuario con el Bot, mas abajo se va a mostrar las stories con una breve descripción de que hace cada una de ella y por último tenemos 3 secciones mas que son Actions, Rules, Prolog, donde se menciona que es cada cosa y se hace un breve resumen de cada una de ellas.

- *NLU*

En esta sección solo se muestran una lista de capturas de pantalla de las NLUs implementadas en el ChatBot:

- Saludo de bienvenida

```
- intent: greet
  examples: |
    - Hola
    - Hola!
    - Holis
    - Buenas
    - Bueenas
    - Buen dia
    - Buenos dias
    - Buenas tardes
    - Buenas noches
    - hey
    - que onda
```

- Saludo de despedida

```
- intent: Saludo
  examples: |
    - Adios
    - Chau
    - Nos vemos
    - Nos estamos viendo
    - Nos estamos hablando
    - Que tengas un buen dia
    - chau chau
    - despues hablamos
    - hablamos luego
    - muchas gracias
    - gracias
    - saludos
    - nada
    - en nada
```

- Preguntarle ¿quién es?

```
- intent: bot_challenge
  examples: |
    - eres un bot?
    - eres humano?
    - estoy hablando con un bot?
    - estoy hablando con un humano?
    - sos un bot?
    - sos un humano?
    - quien sos?
    - como te llamas?
```

- Comentarle como te sentís

1. *Te sentís bien*

```
- intent: mood_great
  examples: |
    - estoy [bien]{"entity":"mood","value":"happy"}
    - Ando re [zarpado]{"entity":"mood","value":"happy"}
    - [Alegre]{"entity":"mood","value":"happy"}
    - Estoy [alegre]{"entity":"mood","value":"happy"}
    - [bien]{"entity":"mood","value":"happy"}
    - bien [bien]{"entity":"mood","value":"happy"}
    - [bien] aca andamos{"entity":"mood","value":"happy"}
    - [contento]{"entity":"mood","value":"happy"}
    - [feliz]{"entity":"mood","value":"happy"}
```

2. *Te sentís Triste*

```
- intent: mood_unhappy
  examples: |
    - estoy teniendo un dia [horrible]{"entity":"mood","value":"sad"}
    - estoy [triste]{"entity":"mood","value":"sad"}
    - [deprimido]{"entity":"mood","value":"sad"}
    - [masomenos]{"entity":"mood","value":"sad"}
    - [angustiado]{"entity":"mood","value":"sad"}
    - [decepcionado]{"entity":"mood","value":"sad"}
    - [mal]{"entity":"mood","value":"sad"}
    - re [mal]{"entity":"mood","value":"sad"}
    - [he estado mejor]{"entity":"mood","value":"sad"}
    - [no es mi mejor dia]{"entity":"mood","value":"sad"}
```

3. *Te sentís Enojado*

```
- intent: mood_angry
  examples: |
    - estoy [enojado]{"entity":"mood","value":"angry"}
    - [enojado]{"entity":"mood","value":"angry"}
    - [cansado]{"entity":"mood","value":"angry"}
    - estoy [podrido]{"entity":"mood","value":"angry"}
    - estoy [cansado]{"entity":"mood","value":"angry"}
    - muy [enojado]{"entity":"mood","value":"angry"}
    - muy [cansado]{"entity":"mood","value":"angry"}
    - muy [furioso]{"entity":"mood","value":"angry"}
    - estoy re [nervioso]{"entity":"mood","value":"angry"}
```

- Preguntarle ¿cómo esta?

```
- intent: estado
  examples: |
    - vos, como estas?
    - como estas?
    - como te sentis?
    - que tal estas?
```

- Agradecerle

```
- intent: agradecimiento
  examples: |
    - genial
    - buenisimos
    - perfecto
    - espectacular
    - increible
```

- Afirmación

```
- intent: affirm
  examples: |
    - si
    - s
    - claro
    - por supuesto
    - me vendria bien
    - Me ayudaria mucho
    - Seria estupendo
    - seria genial
    - si, por favor
```

- Negación

```
- intent: deny
  examples: |
    - no
    - n
    - no me parece
    - no es necesario
    - de ningun modo
    - na
    - na no lo necesito
    - no lo necesito
    - nono
```

- Materias cursadas

```
- intent: materiasCursadas
  examples: |
    - cursadas
    - que materias cursaste
    - cursaste
    - que cursaste
    - que materias tenes cursadas
    - que has cursado
```

- Materias cursando

```
- intent: materiasCursando
  examples: |
    - que materias estas cursando
    - materias que cursas
    - que cursas este cuatrimestre
    - que cursas este cuatri
    - que estas cursando
```

- Materias optativas

```
- intent: materiasOptativas
  examples: |
    - que optativas hisiste
    - que optativas realizaste
    - optativas
    - estas haciendo alguna optativa
    - cursaste optativas
    - tenes optativas cursadas
    - realizaste optativas
```

- Finales que debe

```
- intent: finales
  examples: |
    - debes finales
    - que finales te quedan dar
    - debes algun final
    - tenes finales para dar
    - finales
    - te faltan finales
```

- Porque decidió estudiar ingeniería

```
- intent: carrera
  examples: |
    - porque decidiste estudiar esta carrera
    - porque elegiste esta carrera
    - porque quisiste estudiar esta carrera
    - porque entraste a sistemas
    - porque entraste a ingenieria de sistemas
    - porque decidiste estudiar sistemas
    - porque elegiste ingenieria de sistemas
    - porque elegiste sistemas
    - ingenieria de sistemas
    - estudiar ingenieria de sistemas
    - ingenieria
    - carrera
```

- Cuales son los temas que mas le gustan sobre la carrera

```
- intent: gustos
  examples: |
    - que temas te interezan de la carrera
    - que temas te interezan mas de la carrera
    - cuales son los temas mas relevantes de la carera
    - cuales son los temas que mas te gustan
    - que temas te gustan de la carrera
    - que temas te interezan mas
    - que te gusta de la carrera
    - que es lo que mas te gusta de la carrera
```

- Como se involucró con la programación

```
- intent: involucracionProgramacion
  examples: |
    - cual fue tu primer codigo
    - cual fue tu primera involucracion con la programacion
    - como te involucraste con la programacion
    - como conociste la programacion
    - como te integraste ala programacion
    - como descubriste la programacion
    - como te interesaste por la programacion
```

- Buscar información relacionados a la materia, en la API de Wikipedia

```
- intent: buscar
  examples: |
    - informacion sobre [rasa](busqueda)
    - que sabes sobre [machine learning](busqueda)
    - info [robot](busqueda)
    - [robot](busqueda)

- lookup: busqueda
  examples: |
    - rasa
    - matematicas
    - prolog
    - logica difusa
    - robotica
    - inteligencia artificial
    - pizza
    - boca juniors
    - RoboCup
    - futbol
    - autos autonomos
    - chatbot
```

- Crear un nuevo evento en la API de Google Calendar

```
- intent: nuevoEvento
  examples: |
    - quiero una reunion
    - pactar reunion
    - cuando nos podemos reunir
    - cuando nos juntamos
    - cuando nos podemos juntar
```

1. *Ingresar el día para pactar la reunión*

```
- intent: ingresarDia
  examples: |
    - nos juntamos [mañana](dia)
    - [mañana](dia)
    - podes [mañana](dia)
    - puedo [hoy](dia)
    - si queres el [lunes](dia)
    - [hoy](dia)
    - puedo el [martes](dia)
    - nos juntamos [miercoles](dia)
    - [jueves](dia)
    - podes [sabado](dia)
    - puedo [domingo](dia)
    - si queres el [viernes](dia)
```

2. *Ingresar el horario de reunión*

```
- intent: ingresarHorario
  examples: |
    - te parece a las [10](hora)
    - a las [8](hora)
    - tipo [16](hora)
    - [20](hora)

- regex: hora
  examples: |
   - \b([01][0-9]|2[0-3])\b
```

- Pedir los 10 eventos próximos agendados en Google Calendar

```
- intent: devolverEventos
  examples: |
    - devolver eventos
    - eventos
    - que eventos tenes
    - proximos eventos
    - eventos proximos
```

- ## *Stories*

En esta sección se van a mostrar todas las stories del presente trabajo y explicar brevemente que hace cada una:

- ### Bienvenida

Esta primer Story se ejecuta cuando el usuario ingresa un saludo de bienvenida (greet); El Bot se encarga de saludarlo y verifica si esa persona ya se encontraba agendada, en caso de no estar, se agenda (con el fin de tener todos los contactos agendados y brindar una atención personalizada) para luego preguntarle ¿cómo esta?

```
- story: greeting
  steps:
  - intent: greet
  - action: utter_greet
  - action: action_traer_nombre
  - action: utter_comoestas
  - checkpoint: seguirComoEstas
```

- A continuación, se muestra los 3 tipos personalizados de estado de ánimo que se pueden presentar según lo que el usuario responda (Feliz, Enojado, Triste):

- ### Feliz

En esta Story cuando el usuario nos comenta que se siente “bien”; El Bot le da una respuesta alegre y le pregunta en que lo puede ayudar; como el usuario ya se encuentra con un buen estado de ánimo solo continua la conversación.

- ### Triste

Cuando el usuario nos comenta que se siente “triste”; El Bot le responde con una Imagen (gatito) con el fin de que se sienta mejor, también le pregunta si quiere que le cuente algún chiste para mejorar su estado de ánimo.

- ### *Nuevo chiste*

Tal cual lo muestra el Checkpoint, esta Story se encarga de continuar contando chistes si es que el usuario lo desea.

```
- story: chiste
  steps:
  - checkpoint: seguir
  - intent: affirm
  - action: ActionChistes
```

- Enojado

En cambio, cuando el usuario nos comenta que se siente “enojado”; El Bot le dice que el esta para lo que necesite y al igual que cuando le comentas que te sentís “bien”, te pregunta si te puede ser de ayudar en algo.

Dejo la captura de la story de los 3 estado de ánimo explicada anteriormente

```
- story: happy_Angry_unhappy
  steps:
  - checkpoint: seguirComoEstas
  - or:
    - intent: mood_great
    - intent: mood_angry
    - intent: mood_unhappy
  - or:
    - slot_was_set:
      - mood: happy
    - slot_was_set:
      - mood: angry
    - slot_was_set:
      - mood: sad
  - action: action_mood
  - checkpoint: seguir
```

- Preguntarle ¿Cómo esta?

En esta Story el usuario le pregunta al Bot, su estado ánimo y él siempre le responde que se encuentra bien.

```
- story: preguntarEstado
  steps:
  - intent: estado
  - action: utter_contrarespuestas
```

- Preguntarle ¿Quién es?

En esta Story el usuario le puede preguntar al Bot ¿Quién es?, el cual responde que es el representante de ‘Juani’.

```
- story: quienEres
  steps:
  - intent: bot_challenge
  - action: utter_iamabot
```

- Saludo de despedida

  Esta Story es para cuando el usuario realiza el saludo de despedida, el Bot envía un mensaje personalizado de despedida con su nombre de usuario.

```
- story: saludo
  steps:
  - intent: Saludo
  - action: action_saludo
```

- Agradecimiento

  En esta Story el usuario puede agradecerle al Bot por la información que le brindo.

```
- story: agradecimiento
  steps:
  - intent: agradecimiento
  - action: utter_respuesta
```

- Buscar información sobre la materia en Wikipedia

  En esta Story el usuario puede pedir cualquier información que se encuentra en la lista de búsqueda mostrada anteriormente en la sección de NLU (Buscar información relacionados a la materia, en la API de Wikipedia) y el Bot se encarga de buscar la información pedida en Wikipedia dándonos el título y un breve resumen de ella.

```
- story: buscarWikipedia
  steps:
  - intent: buscar
    entities:
    - busqueda
  - action: action_wikipedia_api
  - checkpoint: continuar
```

Esta Story es para que no rompa cuando se busca información que no se encuentra en la lista antes mencionada y devuelve un mensaje que no se encontró resultados para dicha búsqueda.

```
- story: NobuscaWikipedia
  steps:
  - intent: buscar
  - slot_was_set:
    - busqueda: None
  - action: utter_NoEncontroBusqueda
```

- *Obtener link de la búsqueda*

Continuando de la Story anterior donde obtenemos un breve resumen de la información que deseamos, también podemos pedir el link de la página encontrada para poder profundizar un poco mas sobre el tema, si así se quiere.

```
- story: darLink
  steps:
  - checkpoint: continuar
  - intent: affirm
  - action: action_wikipedia_link

- story: nodarLink
  steps:
  - checkpoint: continuar
  - intent: deny
  - action: action_wikipedia_linkNegado
```

- Materias cursando

En esta Story el usuario le pide las materias que está cursando y el Bot se encarga de irlas a buscar al Prolog para luego imprimirlas por pantalla.

```
- story: carreraCursando
  steps:
  - intent: materiasCursando
  - action: action_prolog_cursando
```

- Materias cursadas

En esta Story el usuario le pide las materias que ya curso en lo que va de la carrera y el Bot se encarga de irlas a buscar al Prolog para luego imprimirlas por pantalla.

```
- story: carreraCursadas
  steps:
  - intent: materiasCursadas
  - action: action_prolog_cursadas
```

- Materias optativas

  En esta Story el usuario le pide las optativas que ya curso y el Bot le comenta que aun no realizo ninguna optativa.

```
- story: carreraOptativas
  steps:
  - intent: materiasOptativas
  - action: utter_optativas
```

- Finales que debe

  En esta Story el usuario le pide los finales que debe y el Bot se encarga de irlas a buscar al Prolog para luego imprimirlas por pantalla.

```
- story: carreraFinales
  steps:
  - intent: finales
  - action: action_prolog_finales
```

- Porque decidió estudiar ingeniería

  En esta Story el usuario le puede preguntar cual fue el motivo de decidir estudiar ingeniería de sistemas obteniendo una opinión personal de la carrera.

```
- story: carreraOpinion
  steps:
  - intent: carrera
  - action: utter_carrera
```

- Cuales son los temas que más le gusta de la carrera

  En esta Story el usuario puede preguntarle cuales son los temas que más le interesan al Bot, relacionado con la carrera.

```
- story: carreraGustos
  steps:
  - intent: gustos
  - action: utter_gustopersonal
```

- Como se involucra con la programación

En esta Story el usuario puede preguntarle al Bot como es que conoció la programación y como se involucró en ella.

```
- story: carreraProgramacion
  steps:
  - intent: involucracionProgramacion
  - action: utter_primerasProgramacion
```

- Realizar reunión con la API de Google Calendar

En esta Story el usuario puede pedir una reunión al Bot, para eso el Bot se encarga de pedirle el día que quiere la reunión y en que horario le parece que sea; así el mismo se encarga de crear un evento en la cuenta de Google Calendar que se encuentre vinculada y agenda la nueva reunión en esa fecha dada y con la persona que le solicito dicha reunión; para el caso de una reunión en un grupo, el Bot deja el link de invitación a la reunión para que cualquier integrante del grupo pueda unirse a ella.

```
- story: reunion
  steps:
  - intent: nuevoEvento
  - action: utter_dameDia
  - intent: ingresarDia
  - action: action_guardarMensaje
  - action: utter_dameHorario
  - intent: ingresarHorario
  - action: action_crear_evento
  - action: actionResponder
```

- Pedir los 10 eventos más próximos a la API de Calendar

En esta Story el usuario puede listar los 10 eventos más próximos que se encuentran agendados en la cuenta de Google Calendar vinculada.

```
- story: devolverEventos
  steps:
  - intent: devolverEventos
  - action: action_get_eventos
```

➢ *Actions*

En esta sección se van a mostrar algunas de las Actions mas relevantes para no hacer tan extenso el informe:

- Saludo de despedida

Esta Action lo que hace es realizar un saludo de despedida personalizado con el nombre del usuario quien ingreso el saludo; el nombre se coloca en donde dice “name”, en caso de no encontrar el nombre del usuario, se realiza igual el saludo, pero sin colocar el nombre de la persona.

```python
class ActionSaludo(Action):

    def name(self) -> Text:
        return "action_saludo"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        input_data=tracker.latest_message
        from_data=input_data["metadata"]["message"]["from"]

        name= from_data["first_name"]
        if name is None:
            dispatcher.utter_message("Nos vemos! Que sigas bien!")
        else:
            dispatcher.utter_message("Nos vemos  "+ name +"! Que sigas bien!".format(name))
        return[]
```

- Operar archivos ‘.json’

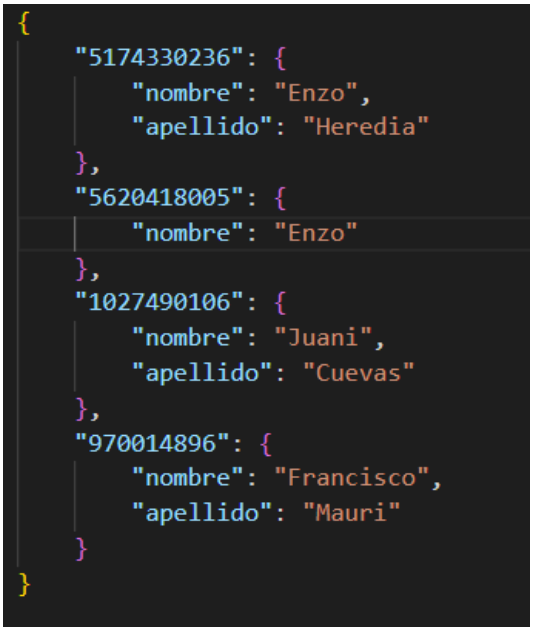
Esta Actions es simplemente para poder abrir los archivos de diccionario ‘.json’; y a su vez poder guardar mas valores y extraer de allí la información requerida.

```python
class OperarArchivo():

    @staticmethod
    def guardar(AGuardar):
        with open(".\\actions\\datos.json","w") as archivo_descarga:
            json.dump(AGuardar, archivo_descarga, indent=4)
        archivo_descarga.close()

    @staticmethod
    def cargarArchivo():
        if os.path.isfile(".\\actions\\datos.json"):
            with open(".\\actions\\datos.json","r") as archivo_carga:
                retorno=json.load(archivo_carga)
                archivo_carga.close()
        else:
            retorno={}
        return retorno

class OperarAchDias():

    @staticmethod
    def cargarArchivo():
        if os.path.isfile(".\\actions\\dias.json"):
            with open(".\\actions\\dias.json","r") as archivo_carga:
                retorno=json.load(archivo_carga)
                archivo_carga.close()
        else:
            retorno={}
        return retorno
```

A continuación se muestran los 2 únicos archivos.json que se utilizaron para el desarrollo del trabajo:

- *datos.json*

En este archivo se guardar los nombres de usuarios con quien interactúa el Bot extrayendo estos datos desde la metadata de Telegram; Guardándolos en este diccionario por su numero de clave (key) y dentro su nombre y apellido si es que tiene. Como vemos en la imagen son algunas pruebas que realice con mis compañeros donde colocamos los ChatBot en un grupo de Telegram y se puede ver que guardo los datos de las personas que interactuaron con él, esto se hace para tener agendadas las personas "conocidas" y así el Bot les brinda una atención más personalizada.



- *días.json*

Este diccionario se realizo para poder almacenar la diferencia de días que hay desde la fecha actual hasta el día que ingresa por mensaje el usuario que quiere pactar una reunión; es simplemente para poder hacer uso de la API de Google Calendar y que el usuario pueda tener una interacción con el Bot más natural; es decir, el usuario puede pactar una reunión solo diciendo el día (por ejemplo: Martes) y este diccionario se encarga de encontrar el martes más próximo a la fecha actual, en ves de que tenga que ingresar en este formato: reunión para el día 25/10/2022 por ejemplo.

```
{
    "Monday": {
        "lunes": 7,
        "martes": 8,
        "miercoles": 2,
        "jueves": 3,
        "viernes": 4,
        "sabado": 5,
        "domingo": 6
    },
    "Tuesday": {
        "lunes": 6,
        "martes": 7,
        "miercoles": 8,
        "jueves": 2,
        "viernes": 3,
        "sabado": 4,
        "domingo": 5
    },
    "Wednesday": {
        "lunes": 5,
        "martes": 6,
        "miercoles": 7,
        "jueves": 8,
        "viernes": 2,
        "sabado": 3,
        "domingo": 4
    },
    "Thursday": {
        "lunes": 4,
        "martes": 5,
        "miercoles": 6,
        "jueves": 7,
        "viernes": 8,
        "sabado": 2,
        "domingo": 3
    },
    "Friday": {
        "lunes": 3,
        "martes": 4,
        "miercoles": 5,
        "jueves": 6,
        "viernes": 7,
        "sabado": 8,
        "domingo": 2
    },
    "Saturday": {
        "lunes": 2,
        "martes": 3,
        "miercoles": 4,
        "jueves": 5,
        "viernes": 6,
        "sabado": 7,
        "domingo": 8
    },
    "Sunday": {
        "lunes": 8,
        "martes": 2,
        "miercoles": 3,
        "jueves": 4,
        "viernes": 5,
        "sabado": 6,
        "domingo": 7
    }
```


- Prolog materias cursando

Esta Action es una de las 3 que se encuentran en este trabajo para el manejo del archivo Prolog, me pareció irrelevante que estén todas ya que son muy similares. Como podemos apreciar se puede ver como se accede al archivo Prolog con su Path y como se va modelando para obtener solo la información que necesitamos ver. En esta captura se obtiene las materias que actualmente me encuentro cursando.

```python
class action_prolog_cursando(Action):
    def name(self) -> Text:
        return "action_prolog_cursando"

    def run(self, dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        with PrologMQI(port=8000) as mqi:
            with mqi.create_thread() as prolog_thread:
                direccion= create_posix_path('C:/Users/54228/Documents/rasa_projects/new_rasa_project/Bot_Personal/actions/Materias(Sistemas).pl'
                prolog_thread.query_async(f'consult("{direccion}").', find_all=False)
                prolog_thread.query_async_result()
                prolog_thread.query_async(f"cursando(X).", find_all=True)
                sleep(0.1)
                result=prolog_thread.query_async_result()
                print(str(result))
                if result:
                    mensaje="Actualmente estoy cursando: " + "\n "
                    for iterator in result:
                        mensaje= mensaje + iterator['X'] + "\n "
                    dispatcher.utter_message(text=mensaje)
        return []
```

- Obtener datos de la metadata (Telegram)

Esta Action se activa cuando el usuario realiza un saludo de bienvenida al Bot; El se encarga de acceder a la metadata de ese mensaje y ahí modelar para obtener el 'id' del usuario para poder obtener de allí su nombre y apellido si es que existe. Estos datos son guardados en 'datos.json' mostrado anteriormente, donde allí se lleva un registro de todas las personas que interactúen con el Bot, para una atención mas personalizada.

```python
class traer_nombre(Action):
    def name(self) -> Text:
       return "action_traer_nombre"

    def run(self, dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        input_data=tracker.latest_message
        from_data=input_data["metadata"]["message"]["from"]
        llave_id=from_data["id"]

        name= from_data["first_name"]
        surname= from_data.get("last_name")
        persona= OperarArchivo.cargarArchivo()

        if not (str(llave_id)) in persona:
            mensaje="Un gusto "+name+", no te conocia"
            persona[llave_id]= {}
            persona[llave_id]['nombre']= name
            if surname:
                persona[llave_id]['apellido']= surname
            OperarArchivo.guardar(persona)
            dispatcher.utter_message(text=str(mensaje))

        print(from_data)
        return []
```

- API de Wikipedia

En esta Action lo que hace es utilizar la API de Wikipedia para encontrar información relacionada a la materia, por una cuestión obvia solo se puede buscar información que se encuentra en la lista del "lookup: búsqueda" que se mostro anteriormente en la sección de NLU; esto se modelo de esa manera para que el usuario no pueda buscar información sobre cualquier tema sino solo relacionado con la materia. En la búsqueda se obtiene el titulo de la pagina encontrada y un breve resumen de ella.

```python
class wikipediaAPI(Action):
    def name(self) -> Text:
       return "action_wikipedia_api"

    def run(self, dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        wiki_wiki = wikipediaapi.Wikipedia('es')
        buscar = tracker.get_slot("busqueda")
        cond = tracker.get_slot("respBuscar")
        print(buscar)
        if buscar:
            page_py = wiki_wiki.page(buscar.replace(" ", "_"))
            print("Page - Exists: %s" % page_py.exists())
            # Page - Exists: True

            respuesta= "Te dejo un breve resumen de lo que encontre sobre '"
            link= "Queres que te pase el link para profundizar?"

            if page_py.exists() and (not cond): # Extraer resumen de pagina
                print("Page - Title: %s" % page_py.title)
                # Page - Title: Python (programming language)

                mensaje= respuesta + page_py.title +"' \n "
                mensaje1= page_py.summary
                dispatcher.utter_message(text=str(mensaje))
                dispatcher.utter_message(text=str(mensaje1))
                dispatcher.utter_message(text=str(link))
                SlotSet("respBuscar", True)

        return [SlotSet("respondido", False)]
```

- API de Wikipedia obtener Link

Esta Action le da un plus a la Action anterior, lo que nos permite obtener el link de la pagina encontrada anteriormente; esto es opcional para el usuario, si le gusto la información obtenida y quiere seguir profundizando sobre el tema puede pedir el link para continuar leyendo a detalle de la misma.

```python
class action_wikipedia_link(Action):
    def name(self) -> Text:
       return "action_wikipedia_link"

    def run(self, dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        respondido = tracker.get_slot("respondido")

        if not respondido:
            wiki_wiki = wikipediaapi.Wikipedia('es')
            buscar = tracker.get_slot("busqueda")
            print(buscar)
            page_py = wiki_wiki.page(buscar.replace(" ", "_"))
            print(page_py.canonicalurl)
            if page_py.exists(): # Extraer resumen de pagina
                messaje= page_py.canonicalurl
                dispatcher.utter_message(text=str(messaje))
        return [SlotSet("busqueda", None), SlotSet("respondido", True)]

class action_wikipedia_linkNegado(Action):
    def name(self) -> Text:
       return "action_wikipedia_linkNegado"

    def run(self, dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        messaje= "Perfecto, en que mas te puedo ayudar?"
        dispatcher.utter_message(text=str(messaje))
        return [SlotSet("busqueda", None)]
```

- Estado de animo

En esta Action el usuario obtiene mensajes personalizados según su estado de ánimo; Por ejemplo, si se encuentra triste “sad” el Bot lo que hace es enviarle una imagen de un gatito a modo de mejorar su estado de ánimo, y se llama a la siguiente Action mostrada a continuación.

```python
class ActionMood(Action):
    def name(self) -> Text:
        return "action_mood"
    def run(self,dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        tema = tracker.get_slot("mood")

        input_data=tracker.latest_message
        chat_id=input_data["metadata"]["message"]["chat"]["id"]
        file_image= 'C:/Users/54228/Documents/rasa_projects/new_rasa_project/motivadora.jpg'
        photo=open(file_image,'rb')

        if tema == "happy":
            mensaje= "Me alegro mucho!"
            mensaje1= "preguntame lo que quieras"
        elif tema == "sad":
            mensaje= "Espero que te haga sentir mejor!"
            mensaje1= "Queres que te cuente un chiste?"
            bot.send_photo(chat_id, photo)
        elif tema == "angry":
            mensaje= "Estoy aqui para lo que necesites"
            mensaje1= "En que puedo ayudarte?"
        else:
            mensaje= "jaja saludos"
            mensaje1= "Queres que te cuente un chiste?"

        dispatcher.utter_message(text=str(mensaje))
        dispatcher.utter_message(text=str(mensaje1))
        return[]
```

- Generar chistes

Es esta Action lo que se hace es general chistes aleatorios sobre la informática con el mismo fin, obtener una mejoría en el estado de animo de la persona.

```python
class ActionChistes(Action):

    def name(self) -> Text:
        return "ActionChistes"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        joke1 = pyjokes.get_joke(language='es', category= 'all')
        message= "Queres otro?"
        #display the joke
        dispatcher.utter_message(text=str(joke1))
        dispatcher.utter_message(text=str(message))
        return[]
```

- API de Google Calendar

En esta Action como podemos apreciar es la encargada de crear un nuevo evento en la cuenta de Google calendar vinculada previamente, lo que primero tenemos que hacer antes de llamar a esta Action es conseguir la hora y el día que se quiere realizar la reunión, así también quien va a participar en ella; como el día al usuario se lo pide como

lunes, martes, miércoles, etc. Lo que hacemos aca es pararnos en la fecha actual y encontrar la primera fecha que ingreso en usuario transformándola a formato ISO, que es el formato que utiliza Google calendar para poder crear un nuevo evento en la plataforma. Así se consigue tener agendada todas las reuniones en nuestro calendario con un recordatorio de 30min antes de que empiece cada reunión.

```python
class Action_crear_evento(Action):
    def name(self) -> Text:
        return "action_crear_evento"
    def run(self,dispatcher: CollectingDispatcher,tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        archDias= OperarAchDias.cargarArchivo()
        fechaActual= datetime.datetime.now()
        fecha= tracker.get_slot("dia")
        hora= int(tracker.get_slot("hora"))
        if not fecha:
            dispatcher.utter_message(text=str("no te entiendo"))
            return[]
        if fecha == "hoy":
            fechaInicial= datetime.datetime(fechaActual.year,fechaActual.month,fechaActual.day,hora)
            fechaFinal= fechaInicial+ datetime.timedelta(hours=1)
        elif fecha == "mañana":
            sumarDias= datetime.timedelta(1)
            DiaReunion= sumarDias + fechaActual
            fechaInicial= datetime.datetime(DiaReunion.year,DiaReunion.month,DiaReunion.day,hora)
            fechaFinal= fechaInicial+ datetime.timedelta(hours=1)
        else:
            diaActual= datetime.datetime.strftime(fechaActual,'%A')
            diferencia= archDias[str(diaActual)][str(fecha)]
            sumarDias= datetime.timedelta(diferencia)
            DiaReunion= sumarDias + fechaActual
            fechaInicial= datetime.datetime(DiaReunion.year,DiaReunion.month,DiaReunion.day,hora)
            fechaFinal= fechaInicial+ datetime.timedelta(hours=1)

        persona= OperarArchivo.cargarArchivo()
        input_data=tracker.latest_message
        persona_id= input_data["metadata"]["message"]["from"]["id"]

        if (str(persona_id)) in persona:
            nom_persona= persona[str(persona_id)]["nombre"]
            apell_persona= persona[str(persona_id)].get("apellido")
            if apell_persona:
                nom_persona= nom_persona +" "+ apell_persona
            print("nombre",nom_persona)
            #event(nom_persona, fechaInicial.isoformat(), fechaFinal.isoformat())
            evento= event(nom_persona, fechaInicial.isoformat(), fechaFinal.isoformat())

            input_data=tracker.latest_message
            grupo=input_data["metadata"]["message"]["chat"]["type"]

            if grupo == "supergroup" or grupo == "group":
                message= "Perfecto, les dejo el link de la reunion por si alguien mas se quiere unir"
                message1= evento.get('htmlLink')
                dispatcher.utter_message(text=str(message))
                dispatcher.utter_message(text=str(message1))
            return[FollowupAction("actionResponder")]
        return[]
```

También existe otra Action que se encarga de listar las 10 reuniones más próximas a la fecha actual, esa Action lo que haces es ir al calendario de Google y extraer de allí todos los eventos que se encuentra agendados. Para que no sea tan extenso este informe no adjunto la imagen de la Actions ya que no me parece tan relevante.

➢ Rules

La única rule que fue implementada, tiene el fin de manejar la conversación arrojando un mensaje de que no entendió lo que el usuario ingreso por mensaje, esto lo hace mucho más interactivo al Bot y evita muchos problemas de ruptura; por ejemplo, si el usuario se equivoca al escribir el Bot simplemente le va a decir que no entendió lo que quiso decir al no encontrar ningún NLU que coincida o se asemeje a lo que el usuario ingreso.

```
- rule: Manejar el fallback
  steps:
  - intent: nlu_fallback
  - action: utter_NoEntiendo
```

➢ Prolog

Antes de mostrar para que fue empleado en nuestro proyecto debemos sabes que Prolog es un lenguaje de programación basado en el paradigma lógico, este lenguaje es utilizado principalmente para aplicaciones de inteligencia artificial. En este proyecto Prolog fue utilizado para modelar el listado de materias del plan de estudio extraído del SIU Guaraní; en el cual se listo todas las materias ya cursadas de la carrera con el formato “materia(número de legajo, nombre de la materia)” y también se listo las materias que actualmente estoy cursando con el formato “cursand(número de legajo, nombre de la materia)”. Así también se referencio a las materias que aun debo realizar el final con el formato “debefinales(número de legajo)”. Esto facilito mucho al ChatBot a la hora de modelar el plan de estudio ya que es muy fácil hacer un listado de todas las materias y mostrarlas por mensaje al usuario que las solicite; En la sección de ‘Actions’ se muestra cómo se interactúa con el archivo de Prolog (Materias.pl) para mostrar todas las materias encontradas según lo que le solicite el usuario.

```
  materia(6111,"introducion a la programacion 1").
  materia(6112,"analisis matematico 1").
  materia(6113,"algebra 1").
  materia(6114,"quimica").
  materia(6121,"ciencias de la Computacion 1").
  materia(6122,"introduccion a la programacion 2").
  materia(6123,"algebra lineal").
  materia(6124,"fisica General").
  materia(6125,"matematica discreta").

  materia(6211,"ciencias de la Computacion II").
  materia(6212,"analisis y Disenio_de_Algoritmos I").
  materia(6213,"introduccion a la Arquitectura de Sistemas").
  materia(6214,"analisis Matematico II").
  materia(6215,"electricidad y Magnetismo").
  materia(6221,"analisis y Disenio de Algoritmos II").
  materia(6222,"comunicacion de Datos I").
  materia(6223,"probabilidades y Estadistica").
  materia(6224,"electronica Digital").
  cursand(0,"ingles").

  cursand(6311,"programacion Orientada a Objetos").
  materia(6312,"estructuras de Almacenamiento de Datos").
  cursand(6321,"programacion Exploratoria").
  cursand(6325,"investigacion Operativa I").

  debefinales(6212).
  debefinales(6214).
  debefinales(6221).
  debefinales(6223).
  debefinales(6224).
  debefinales(6312).

cursadas(X):- materia(_,X).

cursando(X):- cursand(_,X).

finales(X):- debefinales(Y),materia(Y,X).
```

➢ ***Conclusión***

El presente trabajo permitió explicar el proceso de diseño e implementación del representante como alumno, que consistió en el desarrollo de un asistente virtual (chatbot) para ofrecer atención personalizada en el canal conversacional de Telegram.

Este proyecto de la materia me pareció muy divertido ya que nos permitió jugar con la creación de un ChatBot pudiendo asi tener libertar de agregarle diferentes cosas como; por ejemplo, en mi caso le agregar la API de Wikipedia y la API de Google Calendar, esto me pareció muy divertido de hacer, ya que tuvimos muchas risas con mis compañeros cuando agregamos varios Bot en un mismo grupo, esto nos sirvió muchísimo para aprender y solucionar errores que iban surgiendo. Asi pudimos naturalizar bastante al Bot y evitar que se colapsen cuando hablaban todos al mismo tiempo.