Real Time Language Translation Agent System for Call Centers

Date: 2023-11-16
By: csr13

Download as PDF

Example use case. There is a call center that receives calls from foreign nationals from all over the world, the call center does not have agents that speak 10 languages, how do you make it so that the operational cost of hiring call center agents that speak 10 languages is not an budget killer?

On solution is to build a system for translating caller origin language to call center agent origin language, and translating call center agent to caller origin language real time, supporting multiple languages.

Building VOIP based systems without a need for something like asterisk can also be built with Python and replacing software PBX systems like Asterisk with Twilio for custom dialplans.

Diagram and Schematics of the System.

For example, the following schematics for this system, please excuse my design skills.

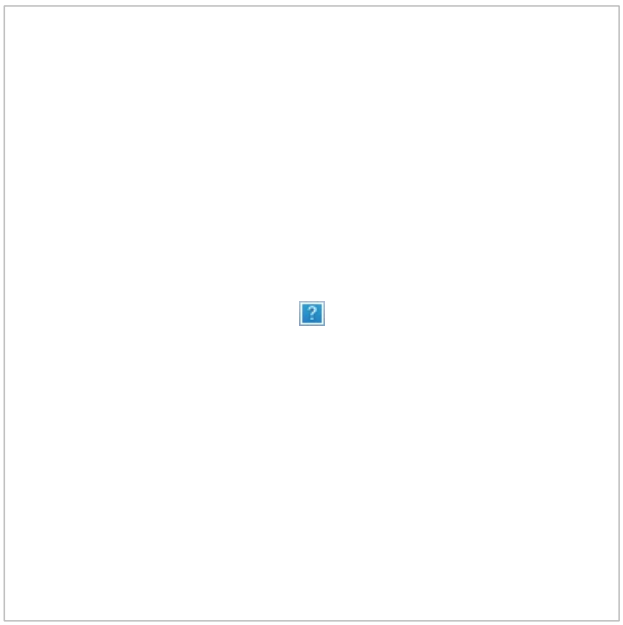

Diagram Breakdown

- 1 Represents the caller, for this instance, the caller speaks chinese, and the agent speaks Arabic.
- 2 Represents the middle broker, Twilio which replaces any VOIP system like Asterisk, so you don't have to program your own PBX.
- 3 Represents the first component of the system which is the relayer component, it's purposes is to pass text converted from speech and translate it from the callers origin language to the agents origin language, back and forward.
- 4 Represents the web application backend the agent is using to type in messages in Arabic and read messages from the caller in Arabic as well, because it was translated from Chinese to Arabic using AWS translation in component 3.
- 5 Is the agent from the call center, that is writing his/her responses in text, and sending them from the web application (4) to the relayer (3) the relayer converts text to the origin caller text, sends to middle broker (2) and finally the caller get's the response in in his/her origin language.

Relayer Breakdown (Component 3)

The relayer and the agent panel use **redis** as the message broker (in memory data storage), and the relayer has a few tables to store call records, and transcripts of calls, which is very important, the secret sauce.

Here are the only tables the relayer use, to store call information, and to store messages from caller and agent, and generate transcript of calls.

```
from django.db import models


class Call(models.Model):
    sid = models.CharField(max_length=100, unique=True)
```

```
    from_number = models.CharField(max_length=100, null=True)
    country = models.CharField(max_length=100, null=True)
    # Call language will be stored as
    #   <full language>|<aws language code>/<twilio language code>
    call_language_code = models.CharField(max_length=255, null=True)
    is_active = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        ordering = ("-created_at",)

    def call_language_twilio_code(self):
        try:
            codes = self.call_language_code.split("|")[1]
            twilio_code = codes.split("/")[1]
        except Exception as error:
            return "Language codes have not been set."
        return twilio_code

    def call_language_aws_code(self):
        try:
            codes = self.call_language_code.split("|")[1]
            aws_code = codes.split("/")[0]
        except Exception as error:
            return "Language codes have not been set."
        return aws_code

    def call_language_natural(self):
        try:
            language = self.call_language_code.split("/")[0]
        except Exception as error:
            return "language has not be set yet .. waiting"
        return language


class Message(models.Model):
    call = models.ForeignKey(Call, on_delete=models.CASCADE)
    language = models.CharField(max_length=100, null=True)
    speech = models.TextField(default='')
    translation = models.TextField(default='')
    from_agent = models.BooleanField(default=False)
    from_caller = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True, null=True)

    class Meta:
        ordering = ("-created_at",)
```

Here is the main code for the relayer component API webhooks that Twilio (the broker -- number 2) talks to and that the agent panel interacts with via **redis**. I will break down the API endpoints in order, order is dicated by the dynamics of the call.

First the system needs to know the language of the caller, so it prompts the caller for it I will include all the imports only on this snippet, on the following ones I won't.

The code is very readable, I added comments.

```
import logging
import time

from django.views import View
from django.http import HttpResponse, JsonResponse
from django.shortcuts import redirect, render
from django.views.decorators.csrf import csrf_exempt
from django.views.decorators.http import require_POST
from django.conf import settings

import boto3
from twilio.twiml.voice_response import Gather, VoiceResponse

from .helpers import translate_text
from .models import Call, Message
from .mappings import LANGUAGE_MAPPINGS
from translator2.settings import redis_connection


logger = logging.getLogger(__name__)


#@require_POST
@csrf_exempt
#@validate_twilio_request
def start(request):
    """
    Entrypoint for the call center dialplan.
    """
    ########################################
    # Create the call in the entry point
```

```
    ######################################

    if request.method == "POST":
        sid = request.POST.get("CallSid")
        from_number = request.POST.get("Caller")
        country = request.POST.get("CallerCountry")
    else:
        sid = request.GET["CallSid"]
        from_number = request.GET["Caller"]
        country = request.GET["CallerCountry"]

    call = Call(
        sid=sid,
        from_number=from_number,
        country=country,
        is_active=True
    )
    call.save()

    ###################################################################
    # Generate response this should go to determine language view
    ###################################################################

    message = "Thanks for calling the call center "
    message += "Please say your native language in english"
    response = VoiceResponse()
    gather = Gather(input='speech', action='/call/determine-language?rc=2&rp=rt')
    response.say(message)
    response.append(gather)
    return HttpResponse(
        response,
        status=200,
        headers={
            "Content-Type": "application/xml"
        }
    )

@require_POST
@csrf_exempt
#@validate_twilio_request
def determine_language(request):
    speech = request.POST.get("SpeechResult")
    speech = speech.strip(".")
    speech = speech.lower()
    sid = request.POST.get("CallSid")
    logger.info("SID => %s is speaking in %s" % (sid, speech))

    ###################################################################
    # Map languages to get the right one.
    ###################################################################

    call_language, language_code = None, None
    for k, v in LANGUAGE_MAPPINGS.items():
        if k == speech:
            call_language = k
            language_code = v

    ###################################################################
    # Handle no language mapped
    ###################################################################

    if call_language is None:
        message = "Invalid language choice, language choices are: "
        for each in LANGUAGE_MAPPINGS:
            message += " %s " % each
        response = VoiceResponse()
        gather = Gather(
            input='speech',
            action='/call/determine-language?rc=2&rp=rt',
        )
        gather.say(message)
        response.append(gather)
        return HttpResponse(
            response,
            status=200,
            headers={
                "Content-Type": "application/xml"
            }
        )

    ###################################################################
    # Get the call and handle call not found.
    ###################################################################

    call = Call.objects.filter(sid=sid, is_active=True)
    if call.exists():
        call = call.first()
    else:
        voice = VoiceResponse()
```

```
        voice.say("Call failed, please try again later.")
        return HttpResponse(voice, status=200, headers={'Content-Type' : ''})

    ###################################################################
    # Store the language codes once they are obtained
    ###################################################################

    call.call_language_code = "%s|%s" % (call_language, language_code)
    call.save()

    ###################################################################
    # Translate the default message and continue with the call
    ###################################################################

    # Messages
    welcome_message = "Your language has been set, please wait for 5 seconds"
    continue_message = "Thank you for waiting, an agent is available, you can talk now."
    # Translations
    welcome_text = translate_text(welcome_message, "en", call)
    continue_text = translate_text(continue_message, "en", call)

    ###################################################################
    # Create response
    ###################################################################

    response = VoiceResponse()
    gather = Gather(
        input='speech',
        action='/call/translate?rc=2&rp=rt',
        language=call.call_language_twilio_code()
    )
    gather.say(welcome_text, language=call.call_language_twilio_code())
    gather.pause(length=7)
    gather.say(continue_text, language=call.call_language_twilio_code())
    response.append(gather)
    return HttpResponse(
        response,
        status=200,
        headers={
            'Content-Type' : 'application/xml'
        }
    )
```

Next, this is the webhook to translate the message from the caller language, to the agents language, the webhooks url are hardcoded in the `Gather(action=<action>, ...)` action parameter of Gather object. One thing that complicates the situation is if the agents takes more than 10 seconds to answer, the call dies, this is solved by a 'duct tape hack' using the request session object, which allows me to store if the agent reponded in time or not, and instead of ending the call on error, this key is used as a boolean flag. This is possible due to the http protocol nature, and it's considered, a 'hack' because twillio does not allow long waiting sessions, so instead direct twilio to forward the call to the same webhook if the agent took long, and request session 'cookies' are used to store boolean value flags to be utilized in the logic of the webhook.

Store last caller message with appended call sid and last agent response with call sid like this

`<call-sid>-latest-caller-message` and `<call-sid>-latest-agent-response` you might encounter them in the snippets bellow.

Once a response for the latest caller message is obtained, then that interaction is saved on the Messages table, related to a Call and the redis key is flushed, waiting for the next interaction, and so on.

```
@require_POST
@csrf_exempt
#@validate_twilio_request
def translate(request):
    ###################################################################
    # Check if the agent was able to respond in time and get the latest
    # speech.
    ###################################################################
    if 'agent_responded_in_time' in request.session.keys():
        agent_responded_in_time = request.session.get(
            'agent_responded_in_time',
        )
        if not agent_responded_in_time:
            speech = request.session.get('caller_latest_text')
        else:
            speech = request.POST.get("SpeechResult")
    else:
        agent_responded_in_time = True
        speech = request.POST.get("SpeechResult")

    sid = request.POST.get("CallSid")
    from_number = request.POST.get("Caller")
    country = request.POST.get("CallerCountry")

    ###################################################
    # Get the call model
    ###################################################
```

```
call = Call.objects.filter(sid=sid, is_active=True)
if call.exists():
    call = call.first()
else:
    call = Call(
        sid=sid,
        from_number=from_number,
        country=country
    )
    call.save()

############################################################
# Only translate if the agent was able to respond in time.
# because the speech was already translated.
############################################################

redis = redis_connection()
if redis is not None:
    logger.info("Redis connection established for sid session %s" % sid)
else:
    raise NotImplementedError()


if agent_responded_in_time:
    translate = boto3.client(
        service_name='translate',
        region_name='eu-west-1',
        aws_access_key_id=settings.AWS_ACCESS_KEY_ID,
        aws_secret_access_key=settings.AWS_SECRET_ACCESS_KEY
    )

    ############################################################
    # Translate speech from the caller to arabic
    ############################################################

    logger.info("Translating %s" % speech)
    result = translate.translate_text(
        Text=speech,
        SourceLanguageCode=call.call_language_aws_code(),
        TargetLanguageCode="en"
    )
    outputText = result.get('TranslatedText')

    ############################################################################
    # Store the message from the caller for transcription.
    ############################################################################

    message = Message(
        call=call,
        speech=speech,
        translation=outputText,
        language=call.call_language_aws_code(),
        from_caller=True,
        from_agent=False,
    )
    message.save()

    ############################################################################
    # Set the latest message from the caller on this call for the agent
    # to read from cache.
    ############################################################################

    logger.info("Setting last caller message for sid %s as => %s" % (
            sid, outputText
        )
    )
    redis.set('%s-latest-caller-message' % sid, outputText)


############################################################################
# We are waiting for the agent to respond on other backend this is the
# checkpoint for that.
############################################################################

agent_response = None
for i in range(1, 11):
    logger.info("Trying to get the latest agent response for %s" % sid)
    response = redis.get('%s-latest-agent-response' % sid)
    logger.info("Latest agent response for sid %s => %s" % (sid, response))
    if response is not None:
        ############################################################
        # Once we get the response, we need to set it back to empty.
        ############################################################
        agent_response = response
        logger.info("%s got latest agent response => %s" % (sid, response))
        redis.delete('%s-latest-agent-response' % sid)
        break
    logger.info(
```

```
            "%s waiting for agent response: seconds %s of %s" % (
                sid,
                i,
                11 - i
            )
        )
        time.sleep(1)

    ###########################################################################
    # Handle if the no agent picked up the phone call on their end, rude
    ###########################################################################

    if agent_response is None:
        logger.info("%s No response from the agent." % sid)
        response = VoiceResponse()
        response.redirect("/call/translate?rc=2&rp=rt")
        http_response = HttpResponse(
            response,
            status=200,
            headers={
                "Content-Type": "application/xml"
            }
        )
        request.session['agent_responded_in_time'] = False
        request.session['caller_latest_text'] = speech
        return http_response

    ###########################################################################
    # Translate back to caller in caller's origin language
    ###########################################################################

    # Flush the request, because we did obtain, and we reset.
    logger.info("%s Flushing the request session" % sid)
    request.session.flush()
    agent_response = agent_response.decode("utf-8")
    result = translate_text(agent_response, "en", call)
    if result is None:
        result = 'unable to translate'

    ###########################################################################
    # Create a new message, this time, the message comes from the agent, and it
    # needs to be translated to the callers language.
    ###########################################################################

    agent_message = Message(
        call=call,
        speech=agent_response,
        translation=result,
        language=call.call_language_natural(),
        from_caller=False,
        from_agent=True,
    )
    agent_message.save()

    ###############################################################
    # Create the final response
    ###############################################################

    response = VoiceResponse()
    gather = Gather(
        input='speech',
        action='/call/translate?rc=2&rp=rt',
        language=call.call_language_twilio_code(),
    )
    gather.say(result, language=call.call_language_twilio_code())
    gather.pause(length=1)
    response.append(gather)
    return HttpResponse(
        response,
        status=200,
        headers={
            "Content-Type" : "application/xml"
        }
    )
```

As you can see, the usage of redis is extensive, as it is the only way for the call center agent panel to 'pick up calls' and read messages.

This last webhook is the error webhook, takes care of flagging the call as not active, so call center agents are not able to see this call as active.

```
@require_POST
@csrf_exempt
#@validate_twilio_request
def error_status(request):
    """
    This view should take care of cleanups
     - Modifying call statuses marking them as not active.
```

```
     - Other administrative tasks.
    """
    response = VoiceResponse()
    response.say("Goodbye")
    sid = request.POST.get("CallSid")
    call = Call.objects.filter(sid=sid)
    if call.exists():
        call = call.first()
        call.is_active = False
        call.save()
    return HttpResponse(
        response,
        status=200,
        headers={
            "Content-Type": "application/xml"
        }
    )
```

So far, the relayer takes care of message translation and storing in memory (redis) for the agent panel to be able to query, this message only if the call is active and via redis storage.

Agent Panel Logic

The following API endpoints are from the agent panel, it has only a few endpoints, needed to code a frontend capable of having call sessions via chat for available calls, see transcripts, see active calls, and see inactive calls.

The main endpoints to look for are

- LatestCallerMessae
- AnswerCallerMessage

Self explanatory, but both of the use **redis** connection to read from the keys done in step 3 component on the above part.

```
import logging

import requests
from django.conf import settings
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView

from api.serializers import AnswerCallerSerializer
from config.settings import redis_connection


logger = logging.getLogger(__name__)


class GetActiveCalls(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):
        resp = requests.get(
            "{}/call/calls?call-status=active".format(
                settings.RELAYER_URL
            )
        )
        if resp.status_code != 200:
            return Response(
                data=dict(error="Unable to get active calls"),
                status=400
            )
        calls = resp.json()
        return Response(data=calls, status=200)


class GetInactiveCalls(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):
        resp = requests.get(
            "{}/call/calls?call-status=inactive".format(
                settings.RELAYER_URL
            )
        )
        if resp.status_code != 200:
            return Response(
                data=dict(error="Unable to get inactive calls"),
                status=400
            )
        calls = resp.json()
        return Response(data=calls, status=200)


class LatestCallerMessage(APIView):
```

```
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):
        sid = kwargs["sid"]

        ############################################################
        # Get the latest message, and clear it from the cache
        ############################################################

        redis = redis_connection()
        message = redis.get('%s-latest-caller-message' % sid)
        if message is None:
            return Response(
                data=dict(message="not-ready"),
                status=400
            )
        message = message.decode("utf-8")
        redis.delete("%s-latest-caller-message" % sid)

        return Response(data=dict(message=message), status=200)


class AnswerCallerMessage(APIView):
    permission_classes = []

    def post(self, request, *args, **kwargs):
        serializer = AnswerCallerSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(data=dict(error=serializer.errors, status=400))

        sid = serializer.validated_data["sid"]
        message = serializer.validated_data["message"]

        ############################################################
        # Set the latest message, and ensure there is no message
        # set before doing so.
        ############################################################

        redis = redis_connection()
        temp_message = redis.get('%s-latest-agent-response' % sid)
        if temp_message is not None:
            redis.delete('%s-latest-agent-response' % sid)

        redis.set("%s-latest-agent-response" % sid, message)
        redis.delete("%s-latest-caller-message")

        return Response(
            data=dict(message="Message sent ... waiting for response"),
            status=200
        )

class GetCallDetails(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, *args, **kwargs):
        resp = requests.get("{}/call/messages/{}".format(
                settings.RELAYER_URL,
                kwargs["sid"]
            )
        )
        if resp.status_code != 200:
            return Response(
                data=dict(error="Unable to get call messages"),
                status=200
            )
        return Response(data=resp.json(), status=200)
```

As far as frontend, I won't get into it, but the way it was done, is a simple dashboard with login for agents, agents can pick up calls, have intervals of requests made to these API endpoints every second (normal traffic in http) and when in a call, having another interval to check for the latest caller message, and answering back and forth, the agent sees the translated text from the caller on this chat window, and answers in his language of origin, the relayer converts back to caller language.

If you are interested in VOIP custom call centers contact me, or simple business call logic checks, like bank menus, or hotel menus, also contact me for consulting and implementation.

As always the source code for this project is on my github, will add the link later in the week.

Thanks for reading.

Related Notes

[1) Using ec2 instances as sneaker bid bots pt 2.]
[Download PDF]
Date published: 2023-11-27
bots python aws

2) Whatsapp chatbot with Python and Twilio
Download PDF
Date published: 2023-11-15
bots python whatsapp business

3) Backend Celery task manager dashboard via Flower
Download PDF
Date published: 2023-10-29
backend tasks python

4) Using ec2 instances as sneaker bid bots pt 1.
Download PDF
Date published: 2023-11-21
bots python aws