**Music box skill – part one**



You tube video: https://youtu.be/eRbn0711hK0

Music box skill – I’ve called it johns music box

References:

Dabble labs

<https://dabblelab.com/tutorials/creating-an-alexa-audio-streaming-skill>

Alexa skills:

https://github.com/alexa/skill-sample-python-audio-player

**Instructions**

Create a new Alexa Hosted skill.

Check / Change the invocation

**Now add audio player interface**. This adds the Pause, Resume intents. We’ll add code for those later.

Click Interface > audio player > switch On. Save interfaces, and build the model

Save model

Change intent json code to:

{

"interactionModel": {

"languageModel": {

"invocationName": "johns music box",

"intents": [

{

"name": "PlayAudio",

"slots": [],

"samples": [

"play my radio",

"start my radio",

"start the audio",

"play the audio",

"start the music",

"play the music",

"play my music"

]

},

{

"name": "AMAZON.PauseIntent",

"samples": []

},

{

"name": "AMAZON.ResumeIntent",

"samples": []

},

{

"name": "AMAZON.HelpIntent",

"samples": [

"how can I listen to my radio",

"how do I play my music",

"tell me how to play",

"how do I listen to my music"

]

},

{

"name": "AMAZON.StopIntent",

"samples": []

},

{

"name": "AMAZON.CancelIntent",

"samples": []

},

{

"name": "AMAZON.StartOverIntent",

"samples": []

},

{

"name": "AMAZON.FallbackIntent",

"samples": []

},

{

"name": "AMAZON.NavigateHomeIntent",

"samples": []

}

],

"types": []

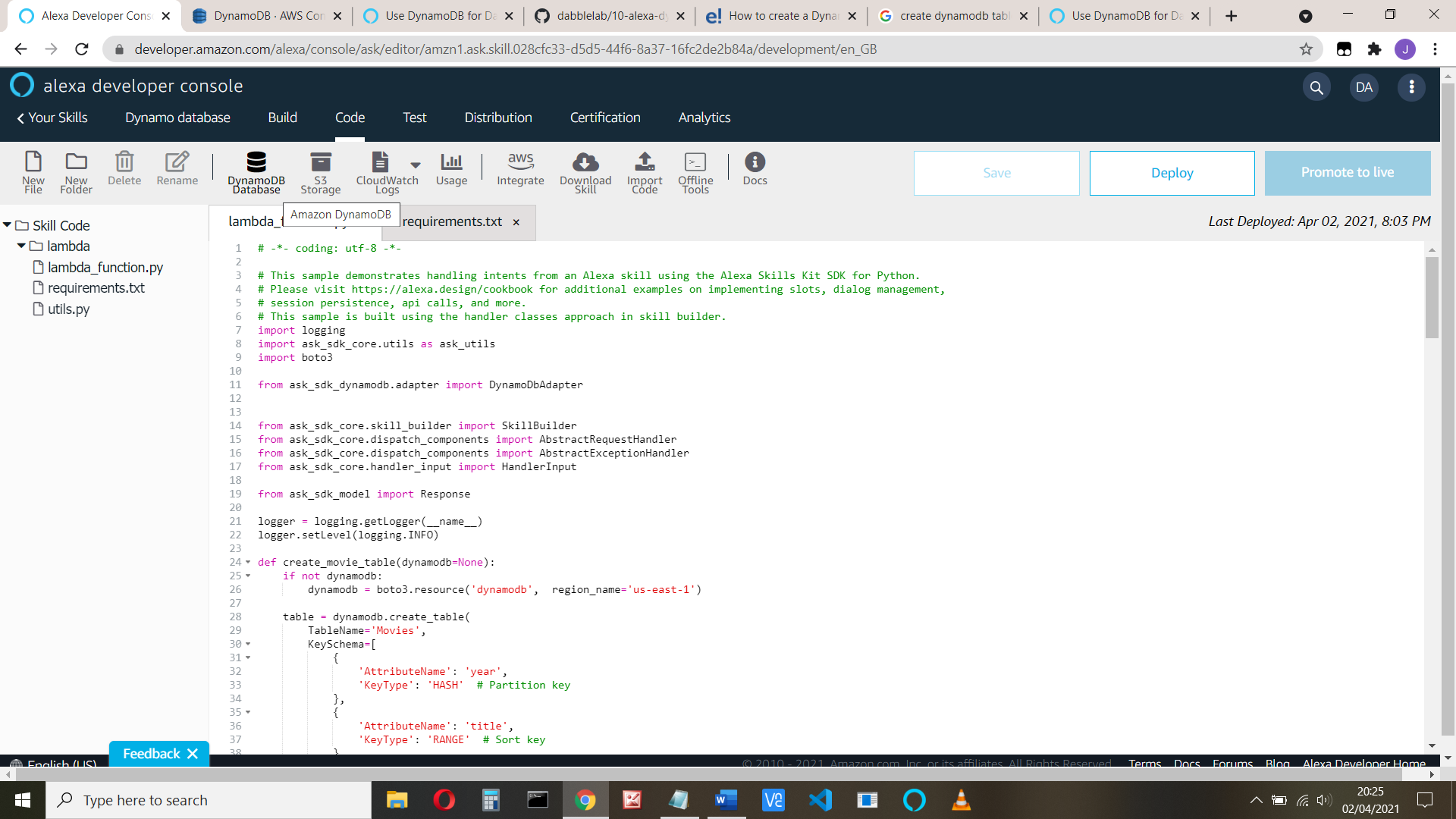
}

}

}

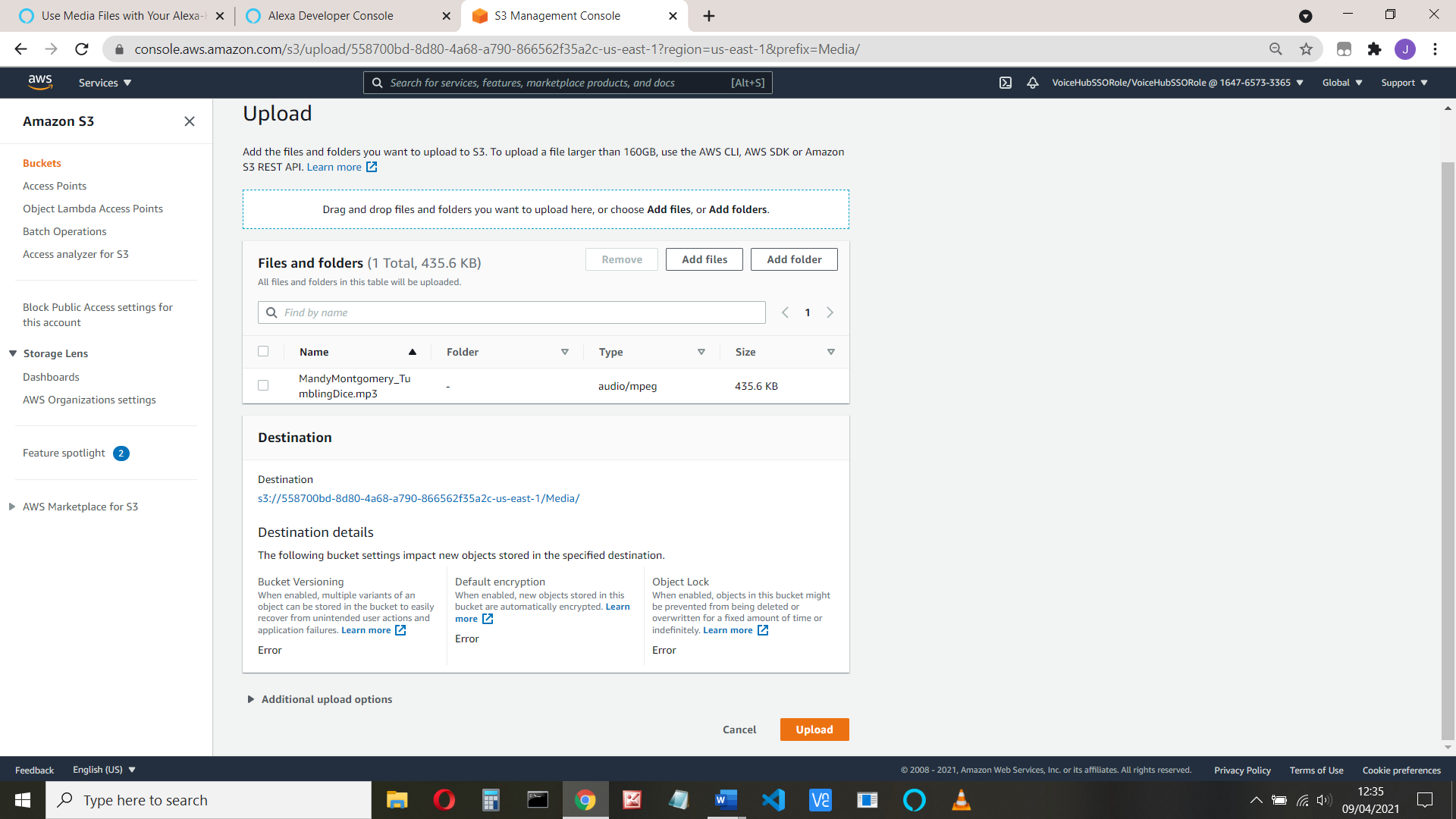
Save and build model

Upload music to S3 Storage



Click the S3 storage tab. It opens in a new window.

Upload your music and images too, if you want them.



Save the end part of the url e.g. Media/RemovalMen\_Borderline.mp3

Or click ‘key’ on the bottom left hand side.

Media/RemovalMen\_Borderline.mp3

Media/waveform108.png

Media/waveform512.png

See <https://developer.amazon.com/es-MX/docs/alexa/hosted-skills/alexa-hosted-skills-media-files.html>

The full code follows, but the important bits are:

1. Launch event – Changed to prompt to play your music.
2. AudioPlayIntentHandler – runs when requested. Executes the PlayDirective. This accesses the music stored in S3 with a presigned url, which gives the program access to the storage. This has a lifetime of 60 seconds, determined in ExpiresIn in utils.py

N.b **The token cannot exceed 1024 characters.**

1. The audioplayer intent handlers – started, nearly finished, stopped etc. All used by the Alexa code. These are added in the skill builder code

Copy the code, replace your url values, deploy and test in developer, enabling skill testing to Development. Check that is does play your music

Next video we’ll add more tracks and a music database.

# -\*- coding: utf-8 -\*-

# This sample demonstrates handling intents from an Alexa skill using the Alexa Skills Kit SDK for Python.

# Please visit https://alexa.design/cookbook for additional examples on implementing slots, dialog management,

# session persistence, api calls, and more.

# This sample is built using the handler classes approach in skill builder.

import logging

import ask\_sdk\_core.utils as ask\_utils

**from utils import create\_presigned\_url**

from ask\_sdk\_core.skill\_builder import SkillBuilder

from ask\_sdk\_core.dispatch\_components import AbstractRequestHandler

from ask\_sdk\_core.dispatch\_components import AbstractExceptionHandler

from ask\_sdk\_core.handler\_input import HandlerInput

from ask\_sdk\_model import Response

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.INFO)

from ask\_sdk\_core.utils import is\_intent\_name

from ask\_sdk\_core.utils import get\_intent\_name

from ask\_sdk\_core.utils import is\_request\_type

from ask\_sdk\_model.ui import StandardCard, Image, SimpleCard

**from ask\_sdk\_model.interfaces.audioplayer import (**

**PlayDirective, PlayBehavior, AudioItem, Stream, AudioItemMetadata,**

**StopDirective)**

from ask\_sdk\_model.interfaces import display

**small\_image\_url = create\_presigned\_url("Media/waveform108.png")**

**large\_image\_url = create\_presigned\_url("Media/waveform512.png")**

audio\_data = {

"card": {

"title": 'My music',

"text": 'I like music',

}

}

card = StandardCard(

title=audio\_data["card"]["title"],

text=audio\_data["card"]["text"],

image=Image(

**small\_image\_url=small\_image\_url,**

**large\_image\_url=large\_image\_url**

)

)

class **LaunchRequestHandler**(AbstractRequestHandler):

"""Handler for Skill Launch."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("LaunchRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Hello, I can play your music, just say play my music"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class HelpIntentHandler(AbstractRequestHandler):

"""Handler for Help Intent."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_intent\_name("AMAZON.HelpIntent")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Just say play my music"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class **AudioPlayIntentHandler**(AbstractRequestHandler):

# Handler for Audioplayer Play Intent

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_intent\_name("PlayAudio")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in AudioPlayIntent")

speech\_text = "Welcome to my audio player"

**audio\_key = "Media/RemovalMen\_Borderline.mp3"**

**audio\_url = create\_presigned\_url(audio\_key)**

directive = **PlayDirective**(

play\_behavior=PlayBehavior.REPLACE\_ALL,

audio\_item=AudioItem(

stream=Stream(

**token=audio\_key,**

**url=audio\_url,**

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.speak(speech\_text).set\_card(card).add\_directive(directive).set\_**should\_end\_session(True)**

logger.info("response")

logger.info(handler\_input.response\_builder.response)

return handler\_input.response\_builder.response

class **AudioStopIntentHandler**(AbstractRequestHandler):

# Handler for Stop – come here on pause or cancel too

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return (is\_intent\_name("AMAZON.CancelIntent")(handler\_input) or

is\_intent\_name("AMAZON.StopIntent")(handler\_input) or

is\_intent\_name("AMAZON.PauseIntent")(handler\_input))

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in AudioStopIntent and request")

logger.info(handler\_input.request\_envelope.request)

speech\_text = "Goodbye"

directive = StopDirective()

handler\_input.response\_builder.speak(speech\_text).add\_directive(

directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

**# ########## AUDIOPLAYER INTERFACE HANDLERS #########################**

# from https://github.com/alexa/skill-sample-python-audio-player

# This section contains handlers related to Audioplayer interface

class PlaybackStartedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackStarted Directive received.

Confirming that the requested audio file began playing.

Do not send any specific response.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackStarted")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackStartedHandler")

return handler\_input.response\_builder.response

class PlaybackFinishedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackFinished Directive received.

Confirming that the requested audio file completed playing.

Do not send any specific response.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackFinished")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackFinishedHandler")

return handler\_input.response\_builder.response

class PlaybackStoppedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackStopped Directive received.

Confirming that the requested audio file stopped playing.

Do not send any specific response.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackStopped")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackStoppedHandler and request is")

logger.info(handler\_input.request\_envelope.request)

return handler\_input.response\_builder.response

class PlaybackNearlyFinishedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackNearlyFinished Directive received.

Replacing queue with the URL again. This should not happen on live streams.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackNearlyFinished")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In PlaybackNearlyFinishedHandler")

audio\_key = "Media/RemovalMen\_Borderline.mp3"

audio\_url = create\_presigned\_url(audio\_key)

directive = PlayDirective(

play\_behavior=PlayBehavior.REPLACE\_ENQUEUED,

audio\_item=AudioItem(

stream=Stream(

token=audio\_key,

url=audio\_url,

offset\_in\_milliseconds=0,

expected\_previous\_token=None),

metadata=None))

handler\_input.response\_builder.set\_card(card).add\_directive(directive).set\_should\_end\_session(True)

return handler\_input.response\_builder.response

class PlaybackFailedHandler(AbstractRequestHandler):

"""AudioPlayer.PlaybackFailed Directive received.

Logging the error and stoprestarting playing with no output speech and card.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("AudioPlayer.PlaybackFailed")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

request = handler\_input.request\_envelope.request

logger.info("Playback failed: {}".format(request.error))

return handler\_input.response\_builder.response

class ExceptionEncounteredHandler(AbstractRequestHandler):

"""Handler to handle exceptions from responses sent by AudioPlayer

request.

"""

def can\_handle(self, handler\_input):

# type; (HandlerInput) -> bool

return is\_request\_type("System.ExceptionEncountered")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EXCEPTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

logger.info(handler\_input.request\_envelope)

return handler\_input.response\_builder.response

# ###################################################################

class SessionEndedRequestHandler(AbstractRequestHandler):

"""Handler for Session End."""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return ask\_utils.is\_request\_type("SessionEndedRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("in SessionEnded")

# Any cleanup logic goes here.

return handler\_input.response\_builder.response

class IntentReflectorHandler(AbstractRequestHandler):

"""The intent reflector is used for interaction model testing and debugging.

It will simply repeat the intent the user said. You can create custom handlers

for your intents by defining them above, then also adding them to the request

handler chain below.

"""

def can\_handle(self, handler\_input):

# type: (HandlerInput) -> bool

return is\_request\_type("IntentRequest")(handler\_input)

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

intent\_name = ask\_utils.get\_intent\_name(handler\_input)

speak\_output = "You just triggered " + intent\_name + "."

return (

handler\_input.response\_builder

.speak(speak\_output)

# .ask("add a reprompt if you want to keep the session open for the user to respond")

.response

)

class CatchAllExceptionHandler(AbstractExceptionHandler):

"""Generic error handling to capture any syntax or routing errors. If you receive an error

stating the request handler chain is not found, you have not implemented a handler for

the intent being invoked or included it in the skill builder below.

"""

def can\_handle(self, handler\_input, exception):

# type: (HandlerInput, Exception) -> bool

return True

def handle(self, handler\_input, exception):

# type: (HandlerInput, Exception) -> Response

logger.error(exception, exc\_info=True)

logger.info("exception")

logger.info(exception)

#logger.info(ask\_utils.get\_intent\_name(handler\_input))

#The provided request is not an IntentRequest

logger.info(ask\_utils.get\_request\_type(handler\_input))

speak\_output = "Sorry, I had trouble doing what you asked. Please try again."

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

# The SkillBuilder object acts as the entry point for your skill, routing all request and response

# payloads to the handlers above. Make sure any new handlers or interceptors you've

# defined are included below. The order matters - they're processed top to bottom.

sb = SkillBuilder()

sb.add\_request\_handler(LaunchRequestHandler())

sb.add\_request\_handler(HelpIntentHandler())

sb.add\_request\_handler(AudioPlayIntentHandler())

sb.add\_request\_handler(AudioStopIntentHandler())

# ########## AUDIOPLAYER INTERFACE HANDLERS #########################

sb.add\_request\_handler(PlaybackStartedHandler())

sb.add\_request\_handler(PlaybackFinishedHandler())

sb.add\_request\_handler(PlaybackStoppedHandler())

sb.add\_request\_handler(PlaybackNearlyFinishedHandler())

sb.add\_request\_handler(PlaybackFailedHandler())

sb.add\_request\_handler(ExceptionEncounteredHandler())

sb.add\_request\_handler(SessionEndedRequestHandler())

sb.add\_request\_handler(IntentReflectorHandler()) # make sure IntentReflectorHandler is last so it doesn't override your custom intent handlers

sb.add\_exception\_handler(CatchAllExceptionHandler())

lambda\_handler = sb.lambda\_handler()

If you get an error look at the cloudwatch logs. Note if you are in the UK, click the down arrow and choose EU.