# -\*- 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 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)

import smtplib, ssl

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

import credentials

import os

import boto3

class LaunchRequestHandler(AbstractRequestHandler):

#Handler for Skill Launch.

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "I've sent your email"

port = 465 # For SSL

smtp\_server = "smtp.gmail.com"

# don't forget to import credentials

sender\_email = credentials.key['sender\_email'] # Enter your address

receiver\_email = credentials.key['receiver\_email'] # Enter receiver address

password = credentials.key['password'] # "given gmail password"

# Send simple text.

#message = "This message is sent using smtplib from Python."

# send multipart message

message = MIMEMultipart("alternative")

message["Subject"] = "TUMBOT YOU HAVE ORDERS-Alexa"

message["From"] = sender\_email

message["To"] = receiver\_email

# add plain text and HTML versions

text = """\

You can find perkin at tinyurl.com/y8dk523j

"""

html = """\

<html>

<body>

<p>Hi,<br>

<a>Go to evans hall</a>

</p>

</body>

</html>

"""

# Convert to plain and html MIMEText objects

plainMIME = MIMEText(text, "plain")

HTMLMIME = MIMEText(html, "html")

# Add the HTML and plain-text to the MIME message

# The email client renders the last part first

message.attach(plainMIME)

message.attach(HTMLMIME)

# and send message as string()

context = ssl.create\_default\_context()

with smtplib.SMTP\_SSL("smtp.gmail.com", 465, context=context) as server:

server.login(sender\_email, password)

server.sendmail(sender\_email, receiver\_email, message.as\_string())

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

# type: (HandlerInput) -> Response

speak\_output = "Welcome, you can say Hello or Help. Which would you like to try?"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class EquipmentInfoRequestHandler(AbstractRequestHandler):

"""Handler for Equipment Info Request."""

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "There are a variety of tools and equipment in the Maker Space such as 3D printers, Oscilloscopes, multimeters, and soldering irons."

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 LocInfoRequestHandler(AbstractRequestHandler):

"""Handler for Location Info Request."""

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Hello World!"

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 HelloWorldIntentHandler(AbstractRequestHandler):

"""Handler for Hello World Intent."""

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Hello World!"

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 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 = "You can say hello to me! How can I help?"

return (

handler\_input.response\_builder

.speak(speak\_output)

.ask(speak\_output)

.response

)

class CancelOrStopIntentHandler(AbstractRequestHandler):

"""Single handler for Cancel and Stop Intent."""

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

# type: (HandlerInput) -> bool

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

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

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

speak\_output = "Goodbye!"

return (

handler\_input.response\_builder

.speak(speak\_output)

.response

)

class FallbackIntentHandler(AbstractRequestHandler):

"""Single handler for Fallback Intent."""

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

# type: (HandlerInput) -> Response

logger.info("In FallbackIntentHandler")

speech = "Hmm, I'm not sure. You can say Hello or Help. What would you like to do?"

reprompt = "I didn't catch that. What can I help you with?"

return handler\_input.response\_builder.speak(speech).ask(reprompt).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

# 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 ask\_utils.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)

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(HelloWorldIntentHandler())

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

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

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

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()