# -\*- 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

import pandas as pd

import requests

import io

import calendar

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)

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 = "Hi, AIML talent sprint participants! Welcome to Hackathon Project Group 15. Would you like to nowyour zodiac sign?"

reprompt\_text = "I was born Nov. 6th, 2014. When were you born?"

return (

handler\_input.response\_builder

.speak(speak\_output)

#.ask(speak\_output) -----------------------------------------------------------------------------

.ask(reprompt\_text)

.response

)

class ZodiacSignIntentHandler(AbstractRequestHandler): #-------------------------------------------

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

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

# type: (HandlerInput) -> bool

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

def filter(self, X):

date = X.split()

month = date[0]

month\_as\_index = list(calendar.month\_abbr).index(month[:3].title())

day = int(date[1])

return (month\_as\_index,day)

def handle(self, handler\_input):

slots = handler\_input.request\_envelope.request.intent.slots

year = slots["year"].value

month= slots["month"].value

day = slots["day"].value

url = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSZ\_08J2ItEG1CSUTIjo6mJBiY8rVCd9tS19n8gS-G5iGiQuCEkZXdbeCRHlTYV3Q/pub?gid=1788107277&single=true&output=csv"

csv\_content = requests.get(url).content

df = pd.read\_csv(io.StringIO(csv\_content.decode('utf-8')))

zodiac = ''

month\_as\_index = list(calendar.month\_abbr).index(month[:3].title())

usr\_dob = (month\_as\_index,int(day))

for index, row in df.iterrows():

if self.filter(row['Start']) <= usr\_dob <= self.filter(row['End']):

zodiac = row['Zodiac']

speak\_output= 'I see you were born on the {day} of {month} {year}, which means that your zodiac sign will be {zodiac}.'.format(month=month, day=day, year=year,zodiac=zodiac)

return (handler\_input.response\_builder

.speak(speak\_output)

.response)

class MovieIntentHandler(AbstractRequestHandler): #-------------------------------------------MovieIntentHandler

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

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

slots = handler\_input.request\_envelope.request.intent.slots

year = slots["year"].value

movie= slots["movie"].value

metascore = slots["metascore"].value

n\_imdb= slots["n\_imdb"].value

url = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSBwGObPGTKAxRdFv0aQv73npIhNwB239jeZwWzw1kREVdPq6pRIaxdFzLEi\_5T6KtRGdULAyjR2kF9/pub?gid=482424460&single=true&output=csv"

csv\_content = requests.get(url).content

df = pd.read\_csv(io.StringIO(csv\_content.decode('utf-8')))

yourMovie = ''

for index, row in df.iterrows():

if (str(year) == str(row['year']) and str(metascore) == str(row['metascore']) and str(n\_imdb) == str(row['n\_imdb']) ):

yourMovie = row['movie']

speak\_output= 'I see you were looking for movie on the {year} with n\_imdb {n\_imdb} and the metascore {metascore}, which means that your movie sign will be {yourMovie}.'.format(year=year, n\_imdb=n\_imdb,metascore=metascore,yourMovie=yourMovie)

return (handler\_input.response\_builder

.speak(speak\_output)

.response)

class BookIntentHandler(AbstractRequestHandler): #-------------------------------------------BookIntentHandler

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

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

# type: (HandlerInput) -> bool

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

def handle(self, handler\_input):

slots = handler\_input.request\_envelope.request.intent.slots

book = slots["book"].value

author= slots["author"].value

#rating = slots["rating"].value

ratings\_count= slots["ratings\_count"].value

year = slots['year'].value

isbn = slots['isbn'].value

book\_id = slots['book\_id'].value

url = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSV7b7gM0lNYIIs8pmdcCVkJg3Db5Nl2CEXKg6HT\_N4PjFv-ZdEw5USlBezH5Okzcm7NkxzAJht6sbB/pub?gid=1808703161&single=true&output=csv"

csv\_content = requests.get(url).content

df = pd.read\_csv(io.StringIO(csv\_content.decode('utf-8')))

yourbook = ''

bookAuthor = ''#str(ratings\_count) == str(row['ratings\_count']) and str(year) == str(row['year']) and # and str(book\_id) == str(row['book\_id'])

bookRating = ''

for index, row in df.iterrows():

if ( str(isbn) == str(row['isbn']) ):#str(author) == str(row['author']) and

yourbook = row['book']

bookAuthor = row['authors']

#bookRating = row['rating']

speak\_output= 'I see you were looking for book id {book\_id} with isbn {isbn} was published by author {bookAuthor} with the ratings\_count {ratings\_count} was published in the year of {year}, base on that your recommended book will be {yourbook} which has rating {bookRating}.'.format( isbn=isbn, bookAuthor=bookAuthor, yourbook=yourbook)

#speak\_output= 'I see you were looking for book id {book\_id} with isbn {isbn} was published by author {bookAuthor} with the ratings\_count {ratings\_count} was published in the year of {year}, base on that your recommended book will be {yourbook} which has rating {bookRating}.'.format(book\_id=book\_id, isbn=isbn, bookAuthor=bookAuthor, ratings\_count=ratings\_count,year=year, yourbook=yourbook, bookRating=bookRating)

return (handler\_input.response\_builder

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

return (

handler\_input.response\_builder

.speak(speak\_output)

#.ask(speak\_output)

.ask(reprompt\_text)

.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 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(ZodiacSignIntentHandler()) #-------------------------------------------

sb.add\_request\_handler(MovieIntentHandler()) #-------------------------------------------MovieIntentHandler

sb.add\_request\_handler(BookIntentHandler()) #-------------------------------------------BookIntentHandler

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

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

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()