### Required Libraries ###

from datetime import datetime

from dateutil.relativedelta import relativedelta

### Functionality Helper Functions ###

def parse\_int(n):

"""

Securely converts a non-integer value to integer.

"""

try:

return int(n)

except ValueError:

return float("nan")

def build\_validation\_result(is\_valid, violated\_slot, message\_content):

"""

Define a result message structured as Lex response.

"""

if message\_content is None:

return {"isValid": is\_valid, "violatedSlot": violated\_slot}

return {

"isValid": is\_valid,

"violatedSlot": violated\_slot,

"message": {"contentType": "PlainText", "content": message\_content},

}

### Dialog Actions Helper Functions ###

def get\_slots(intent\_request):

"""

Fetch all the slots and their values from the current intent.

"""

return intent\_request["currentIntent"]["slots"]

def elicit\_slot(session\_attributes, intent\_name, slots, slot\_to\_elicit, message):

"""

Defines an elicit slot type response.

"""

return {

"sessionAttributes": session\_attributes,

"dialogAction": {

"type": "ElicitSlot",

"intentName": intent\_name,

"slots": slots,

"slotToElicit": slot\_to\_elicit,

"message": message,

},

}

def delegate(session\_attributes, slots):

"""

Defines a delegate slot type response.

"""

return {

"sessionAttributes": session\_attributes,

"dialogAction": {"type": "Delegate", "slots": slots},

}

def close(session\_attributes, fulfillment\_state, message):

"""

Defines a close slot type response.

"""

response = {

"sessionAttributes": session\_attributes,

"dialogAction": {

"type": "Close",

"fulfillmentState": fulfillment\_state,

"message": message,

},

}

return response

"""

Step 3: Enhance the Robo Advisor with an Amazon Lambda Function

In this section, you will create an Amazon Lambda function that will validate the data provided by the user on the Robo Advisor.

1. Start by creating a new Lambda function from scratch and name it `recommendPortfolio`. Select Python 3.7 as runtime.

2. In the Lambda function code editor, continue by deleting the AWS generated default lines of code, then paste in the starter code provided in `lambda\_function.py`.

3. Complete the `recommend\_portfolio()` function by adding these validation rules:

\* The `age` should be greater than zero and less than 65.

\* The `investment\_amount` should be equal to or greater than 5000.

4. Once the intent is fulfilled, the bot should respond with an investment recommendation based on the selected risk level as follows:

\* \*\*none:\*\* "100% bonds (AGG), 0% equities (SPY)"

\* \*\*low:\*\* "60% bonds (AGG), 40% equities (SPY)"

\* \*\*medium:\*\* "40% bonds (AGG), 60% equities (SPY)"

\* \*\*high:\*\* "20% bonds (AGG), 80% equities (SPY)"

> \*\*Hint:\*\* Be creative while coding your solution, you can have all the code on the `recommend\_portfolio()` function, or you can split the functionality across different functions, put your Python coding skills in action!

5. Once you finish coding your Lambda function, test it using the sample test events provided for this Challenge.

6. After successfully testing your code, open the Amazon Lex Console and navigate to the `recommendPortfolio` bot configuration, integrate your new Lambda function by selecting it in the “Lambda initialization and validation” and “Fulfillment” sections.

7. Build your bot, and test it with valid and invalid data for the slots.

"""

### investment recommendation

### Intents Handlers ###

def recommend\_portfolio(intent\_request):

"""

Performs dialog management and fulfillment for recommending a portfolio.

"""

first\_name = get\_slots(intent\_request)["firstName"]

age = get\_slots(intent\_request)["age"]

investment\_amount = get\_slots(intent\_request)["investmentAmount"]

risk\_level = get\_slots(intent\_request)["riskLevel"]

source = intent\_request["invocationSource"]

# YOUR CODE GOES HERE!

### Validation rules for age and investment amount , age > 0 and less than 65 and Investment amount greater than or equal to $5,000.

if source == "DialogCodeHook":

# Perform basic validation on the supplied input slots.

# Use the elicitSlot dialog action to re-prompt

# for the first violation detected.

# Get the input the slots from the intent request

slots = get\_slots(intent\_request)

# Validates user's input using the three validate\_data function

validation\_result = validate\_data(age, investment\_amount, risk\_level)

# If the age, investement amount and risk-level is in not valid,

# the elicitSlot dialog action is used to re-prompt for the first violation detected.

if not validation\_result["isValid"]:

slots[validation\_result["violatedSlot"]] = None # Cleans invalid slot

# Returns an elicitSlot dialog to request new data for the invalid slot

return elicit\_slot(

intent\_request["sessionAttributes"],

intent\_request["currentIntent"]["name"],

slots,

validation\_result["violatedSlot"],

validation\_result["message"],

)

# Fetch current session attibutes

output\_session\_attributes = intent\_request["sessionAttributes"]

#based on the valid slots , the return will be pciked to the next step.

return delegate(output\_session\_attributes, get\_slots(intent\_request))

#Investement recommendations

content = "Thank you, " + first\_name + ", for your information. "

if risk\_level == "none":

content += "100% bonds (AGG), 0% equities (SPY)"

elif risk\_level == "low":

content += "60% bonds (AGG), 40% equities (SPY)"

elif risk\_level == "medium":

content += "40% bonds (AGG), 60% equities (SPY)"

elif risk\_level == "high":

content += "20% bonds (AGG), 80% equities (SPY)"

# Return a message with conversion's result.

return close(

intent\_request["sessionAttributes"],

"Fulfilled",

{

"contentType": "PlainText",

"content": content,

},

)

# Validation functions for age(greater than 0 & leass than 65, Investment amount( greater than 5,000) and risk level("None", "Low", "Medium", "High")

def validate\_data(age, investment\_amount, risk\_level ):

# Validate the age, it should be age <= 0 or age >= 65

if age is not None:

age = parse\_int(

age

) # Since parameters are strings it's important to cast values

if age <= 0 or age >= 65:

return build\_validation\_result(

False,

"age",

"The age should be greater than 0 and less than 65. "

"please provide a correct age.",

)

# Validate the investment amount, it should be begreater than 5000.

if investment\_amount is not None:

investment\_amount = parse\_int(

investment\_amount

) # Since parameters are strings it's important to cast values

if investment\_amount < 5000 :

return build\_validation\_result(

False,

"investmentAmount",

"The investment amount should be greater than 5000, "

"please provide a correct amount.",

)

# Validate the risk\_level, case insensitive: "None", "Low", "Medium", "High".

if risk\_level is not None:

risk\_level = parse\_str( risk\_level )

if risk\_level not in ["none", "low", "medium", "high"] :

return build\_validation\_result(

False,

"riskLevel",

"risk level should be one of the foloowing keywords (case insensitive) : \"None\", \"Low\", \"Medium\", \"High\". "

"please provide a risk\_level.",

)

# The expected results is returned if age , amount and risk\_level are valid

return build\_validation\_result(True, None, None)

### Intents Dispatcher ###

def dispatch(intent\_request):

"""

Called when the user specifies an intent for this bot.

"""

intent\_name = intent\_request["currentIntent"]["name"]

# Dispatch to bot's intent handlers

if intent\_name == "recommendPortfolio":

return recommend\_portfolio(intent\_request)

raise Exception("Intent with name " + intent\_name + " not supported")

### Main Handler ###

def lambda\_handler(event, context):

"""

Route the incoming request based on intent.

The JSON body of the request is provided in the event slot.

"""

return dispatch(event)

##end of code