import boto3

from botocore.exceptions import ClientError

import bcrypt

from datetime import datetime, timedelta

import jwt # PyJWT library for token management

import os

import logging

# Initialize DynamoDB client

dynamodb = boto3.client('dynamodb')

dynamodb\_resource = boto3.resource("dynamodb")

# Constants for token generation

SECRET\_KEY = os.getenv('JWT\_SECRET', 'your-secret-key') # Store this securely!

TOKEN\_EXPIRATION\_MINUTES = 15 # Shorter expiration for added security

REFRESH\_TOKEN\_EXPIRATION\_DAYS = 7 # Refresh token expiration

# Constants for login

MAX\_LOGIN\_ATTEMPTS = 5

LOCKOUT\_DURATION\_MINUTES = 15

RATE\_LIMIT\_WINDOW\_SECONDS = 60 # 1 minute

RATE\_LIMIT\_MAX\_REQUESTS = 10 # Max 10 requests in 1 minute

# Configure logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger()

def generate\_client\_table\_name(company\_id):

"""Generate the Table name string based on the id."""

return f"{company\_id}Employee"

def generate\_auth\_table\_name(company\_id):

"""Generate the Table name string based on the id."""

return f"{company\_id}Employee\_Auth"

def generate\_token(email, guid, expiration\_minutes):

"""Generate a JWT token for the user."""

expiration = datetime.utcnow() + timedelta(minutes=expiration\_minutes)

payload = {

"email": email,

"guid": guid,

"exp": expiration

}

token = jwt.encode(payload, SECRET\_KEY, algorithm="HS256")

return token

def generate\_refresh\_token(email, guid):

"""Generate a JWT refresh token."""

expiration = datetime.utcnow() + timedelta(days=REFRESH\_TOKEN\_EXPIRATION\_DAYS)

payload = {

"email": email,

"guid": guid,

"exp": expiration

}

refresh\_token = jwt.encode(payload, SECRET\_KEY, algorithm="HS256")

return refresh\_token

def lambda\_handler(event, context):

# Extract email, password, and company\_id (using the correct JSON key "CompanyID")

email = event.get('email')

password = event.get('password')

company\_id = event.get('CompanyID') # Ensure the key matches the payload

# Validate parameters

if not email or not password or not company\_id:

return {

'statusCode': 400,

'body': 'Email, password, and company\_id are required.'

}

auth\_table\_name = generate\_auth\_table\_name(company\_id)

client\_table\_name = generate\_client\_table\_name(company\_id)

# Step 1: Query the auth table for user details

try:

response = dynamodb.get\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}}

)

if 'Item' not in response:

logger.warning(f"User not found: {email}")

return {'statusCode': 404, 'body': 'User not found.'}

auth\_data = response['Item']

guid = auth\_data['GUID']['S']

login\_attempts = int(auth\_data.get('LoginAttempts', {}).get('N', '0'))

blocked\_until = auth\_data.get('BlockedUntil', {}).get('S')

except ClientError as e:

logger.error(f"Error querying auth table: {str(e)}")

return {'statusCode': 500, 'body': f'Error querying auth table: {str(e)}'}

# Check if user is blocked

if blocked\_until:

blocked\_until\_dt = datetime.strptime(blocked\_until, '%Y-%m-%dT%H:%M:%S')

if blocked\_until\_dt > datetime.utcnow():

return {

'statusCode': 403,

'body': f'Account locked. Please try again after {blocked\_until\_dt}.'

}

# Step 2: Verify password

first\_time\_password\_db = auth\_data.get('FirstTimePassword', {}).get('S')

user\_entered\_password\_db = auth\_data.get('UserEnteredPassword', {}).get('S')

password\_correct = False

if first\_time\_password\_db and bcrypt.checkpw(password.encode('utf-8'), first\_time\_password\_db.encode('utf-8')):

password\_correct = True

if not password\_correct and user\_entered\_password\_db and bcrypt.checkpw(password.encode('utf-8'), user\_entered\_password\_db.encode('utf-8')):

password\_correct = True

if not password\_correct:

# Increment login attempts and lock account if necessary

login\_attempts += 1

if login\_attempts >= MAX\_LOGIN\_ATTEMPTS:

blocked\_until\_dt = datetime.utcnow() + timedelta(minutes=LOCKOUT\_DURATION\_MINUTES)

blocked\_until\_str = blocked\_until\_dt.strftime('%Y-%m-%dT%H:%M:%S')

dynamodb.update\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}},

UpdateExpression="SET LoginAttempts = :attempts, BlockedUntil = :blocked\_time",

ExpressionAttributeValues={

':attempts': {'N': str(login\_attempts)},

':blocked\_time': {'S': blocked\_until\_str}

}

)

return {

'statusCode': 403,

'body': f'Too many failed attempts. Account locked until {blocked\_until\_str}.'

}

else:

dynamodb.update\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}},

UpdateExpression="SET LoginAttempts = :attempts",

ExpressionAttributeValues={':attempts': {'N': str(login\_attempts)}}

)

return {'statusCode': 403, 'body': 'Invalid password.'}

# Step 3: Generate a JWT token and refresh token

token = generate\_token(email, guid, TOKEN\_EXPIRATION\_MINUTES)

refresh\_token = generate\_refresh\_token(email, guid)

# Step 4: Fetch user data for the response

try:

user\_data\_response = dynamodb.get\_item(

TableName=client\_table\_name,

Key={'GUID': {'S': guid}}

)

if 'Item' not in user\_data\_response:

return {'statusCode': 404, 'body': 'User data not found.'}

user\_data = user\_data\_response['Item']

user\_data\_cleaned = {

'GUID': user\_data['GUID']['S'],

'FirstName': user\_data.get('FirstName', {}).get('S', 'N/A'),

'LastName': user\_data.get('LastName', {}).get('S', 'N/A'),

'Email': user\_data.get('Email', {}).get('S', 'N/A')

}

except ClientError as e:

logger.error(f"Error fetching user data: {str(e)}")

return {'statusCode': 500, 'body': f'Error fetching user data: {str(e)}'}

return {

'statusCode': 200,

'body': {

'message': 'Login successful.',

'token': token,

'refresh\_token': refresh\_token,

'user\_data': user\_data\_cleaned

}

}