Lambda関数の設計方針

穴埋めレベルの設定

埋めてもらう部分

- バリデーションロジック
- エラーメッセージ
- DynamoDB操作の一部（put_item, get_itemの引数など）
- レスポンスのステータスコード

提供する部分

- 関数の骨格（lambda_handler）
- boto3の初期化
- try-exceptの構造
- CORSヘッダー

Lambda関数1: posts-list（穴埋め版）

```
import json
import boto3
from decimal import Decimal

# DynamoDBクライアント初期化
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('posts')

def lambda_handler(event, context):
    try:
        #【穴埋め1】クエリパラメータからカテゴリを取得してください
        query_params = event.get('queryStringParameters', {})
        category = _________________ if query_params else None

        # DynamoDBスキャン
        if category:
            #【穴埋め2】カテゴリでフィルタリングしてスキャンしてください
            response = table.scan(
                FilterExpression=_________________,
                ExpressionAttributeValues=_________________
            )
        else:
            # 全件取得
            response = table.scan()

        # 取得したアイテム
        items = response.get('Items', [])

        #【穴埋め3】タイムスタンプ降順でソートしてください（新しい順）
        items.sort(key=lambda x: _________________, reverse=_________________)
```

```
        # Decimal型をfloat/intに変換
        items = json.loads(json.dumps(items, default=decimal_default))

        #【穴埋め4】成功時のHTTPステータスコードを記入してください
        return {
            'statusCode': _______________,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Methods': 'GET, OPTIONS',
                'Access-Control-Allow-Headers': 'Content-Type'
            },
            'body': json.dumps({
                'status': 'success',
                'posts': items
            })
        }

    except Exception as e:
        print(f'Error: {str(e)}')
        #【穴埋め5】エラー時のHTTPステータスコードを記入してください
        return {
            'statusCode': _______________,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'error',
                'message': '投稿の取得に失敗しました'
            })
        }

def decimal_default(obj):
    if isinstance(obj, Decimal):
        return float(obj) if obj % 1 else int(obj)
    raise TypeError
```

ヒント

```
【穴埋め1】query_params.get('____')
【穴埋め2】FilterExpression='category = :cat', ExpressionAttributeValues={':cat':
____}
【穴埋め3】x.get('____', 0), reverse=True
【穴埋め4】成功時は200
【穴埋め5】サーバーエラーは500
```

Lambda関数2: posts-get（穴埋め版）

```python
import json
import boto3
from decimal import Decimal

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('posts')

def lambda_handler(event, context):
    try:
        # 【穴埋め1】パスパラメータからpostIdを取得してください
        post_id = event['pathParameters']['_________________']

        # 【穴埋め2】DynamoDBからpostIdでアイテムを取得してください
        response = table.get_item(Key={'postId': _________________})

        # アイテムが存在しない場合
        if 'Item' not in response:
            # 【穴埋め3】見つからない場合のHTTPステータスコードを記入してください
            return {
                'statusCode': _________________,
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({
                    'status': 'error',
                    'message': '投稿が見つかりません'
                })
            }

        item = response['Item']

        # Decimal型をfloat/intに変換
        item = json.loads(json.dumps(item, default=decimal_default))

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'success',
                'post': item
            })
        }

    except KeyError:
        # 【穴埋め4】リクエスト不正のHTTPステータスコードを記入してください
        return {
            'statusCode': _________________,
            'headers': {
```

```
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'error',
                'message': 'postIdが指定されていません'
            })
        }
    except Exception as e:
        print(f'Error: {str(e)}')
        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'error',
                'message': '投稿の取得に失敗しました'
            })
        }

def decimal_default(obj):
    if isinstance(obj, Decimal):
        return float(obj) if obj % 1 else int(obj)
    raise TypeError
```

ヒント

```
【穴埋め1】postId
【穴埋め2】post_id
【穴埋め3】Not Foundは404
【穴埋め4】Bad Requestは400
```

Lambda関数3: posts-create（穴埋め版）

```
import json
import boto3
import uuid
import time

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('posts')

def lambda_handler(event, context):
    try:
        # リクエストBodyをパース
        body = json.loads(event['body'])
```

```
        # 【穴埋め1】bodyから各値を取得してください
        title = body.get('_____', '').strip()
        content = body.get('_____', '').strip()
        author = body.get('_____', '').strip()
        category = body.get('_____', '').strip()

        # バリデーション
        # 【穴埋め2】titleが空の場合のチェックを記述してください
        if _____:
            return error_response(400, 'タイトルを入力してください')

        # 【穴埋め3】titleの長さが100文字を超える場合のチェックを記述してください
        if _____:
            return error_response(400, 'タイトルは100文字以内にしてください')

        # 【穴埋め4】contentが空の場合のチェックを記述してください
        if _____:
            return error_response(400, '本文を入力してください')

        # 【穴埋め5】contentの長さが1000文字を超える場合のチェックを記述してください
        if _____:
            return error_response(400, '本文は1000文字以内にしてください')

        # 【穴埋め6】authorが空の場合のチェックを記述してください
        if _____:
            return error_response(400, '投稿者名を入力してください')

        # 【穴埋め7】categoryが空の場合のチェックを記述してください
        if _____:
            return error_response(400, 'カテゴリを選択してください')

        # 【穴埋め8】categoryが許可された値かチェックしてください
        valid_categories = ['技術', '質問', '雑談', 'お知らせ']
        if category not in _____:
            return error_response(400, f'カテゴリは{", ".join(valid_categories)}から選
択してください')

        # postIdとtimestamp生成
        # 【穴埋め9】UUIDを生成してpostIdに代入してください
        post_id = str(_____)

        # 【穴埋め10】現在時刻のUnixTimeを取得してください
        timestamp = int(_____)

        # DynamoDBに保存するアイテム
        item = {
            'postId': post_id,
            'title': title,
            'content': content,
            'author': author,
            'category': category,
            'timestamp': timestamp
        }
```

```
        # 画像キーがあれば追加
        if 'imageKey' in body and body['imageKey']:
            item['imageKey'] = body['imageKey']

        #【穴埋め11】DynamoDBにアイテムを保存してください
        table._____________(_____________)

        #【穴埋め12】作成成功時のHTTPステータスコードを記入してください
        return {
            'statusCode': _____________,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'success',
                'message': '投稿を作成しました',
                'postId': post_id
            })
        }

    except json.JSONDecodeError:
        return error_response(400, 'リクエストBodyが不正です')
    except Exception as e:
        print(f'Error: {str(e)}')
        return error_response(500, '投稿の作成に失敗しました')

def error_response(status_code, message):
    return {
        'statusCode': status_code,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'status': 'error',
            'message': message
        })
    }
```

ヒント

```
【穴埋め1】'title', 'content', 'author', 'category'
【穴埋め2】not title
【穴埋め3】len(title) > 100
【穴埋め4】not content
【穴埋め5】len(content) > 1000
【穴埋め6】not author
【穴埋め7】not category
【穴埋め8】valid_categories
```

```
【穴埋め9】uuid.uuid4()
【穴埋め10】time.time()
【穴埋め11】put_item(Item=item)
【穴埋め12】Createdは201
```

Lambda関数4: posts-delete（穴埋め版）

```python
import json
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('posts')

def lambda_handler(event, context):
    try:
        # 【穴埋め1】パスパラメータからpostIdを取得してください
        post_id = event['pathParameters']['_________________']

        # 【穴埋め2】投稿が存在するか確認してください
        response = table.get_item(Key={'postId': _________________})

        if 'Item' not in response:
            return {
                'statusCode': 404,
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({
                    'status': 'error',
                    'message': '投稿が見つかりません'
                })
            }

        # 【穴埋め3】DynamoDBから投稿を削除してください
        table._________________(Key={'postId': _________________})

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'success',
                'message': '投稿を削除しました'
            })
        }

    except KeyError:
```

```
        return {
            'statusCode': 400,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'error',
                'message': 'postIdが指定されていません'
            })
        }
    except Exception as e:
        print(f'Error: {str(e)}')
        return {
            'statusCode': 500,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'error',
                'message': '投稿の削除に失敗しました'
            })
        }
```

ヒント

```
【穴埋め1】postId
【穴埋め2】post_id
【穴埋め3】delete_item, post_id
```

Lambda関数5: upload-url（穴埋め版）

```
import json
import boto3
import uuid

s3_client = boto3.client('s3')
BUCKET_NAME = 'your-images-bucket-name'  # 【注意】実際のバケット名に変更してください

def lambda_handler(event, context):
    try:
        # リクエストBodyをパース
        body = json.loads(event['body'])

        # 【穴埋め1】bodyからfileNameとfileTypeを取得してください
        file_name = body.get('_____________', '')
        file_type = body.get('_____________', '')
```

```
        # バリデーション
        #【穴埋め2】ファイル名が空の場合のチェックを記述してください
        if _________________:
            return error_response(400, 'ファイル名を指定してください')

        #【穴埋め3】ファイルタイプが空の場合のチェックを記述してください
        if _________________:
            return error_response(400, 'ファイルタイプを指定してください')

        # 画像タイプの検証
        allowed_types = ['image/jpeg', 'image/jpg', 'image/png', 'image/gif',
'image/webp']
        #【穴埋め4】fileTypeが許可されたタイプかチェックしてください
        if file_type not in _________________:
            return error_response(400, f'許可されていないファイルタイプです。{",
".join(allowed_types)}のみアップロード可能です')

        # 拡張子取得
        extension = file_name.split('.')[-1] if '.' in file_name else 'jpg'

        #【穴埋め5】UUIDを生成してユニークなIDを作成してください
        unique_id = str(_________________)

        #【穴埋め6】imageKeyを生成してください（形式: images/{unique_id}.{extension}）
        image_key = f'_________________'

        #【穴埋め7】署名付きURLを生成してください（15分間有効）
        presigned_url = s3_client.generate_presigned_url(
            '_________________',
            Params={
                'Bucket': _________________,
                'Key': _________________,
                'ContentType': _________________
            },
            ExpiresIn=_________________  # 秒数
        )

        return {
            'statusCode': 200,
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({
                'status': 'success',
                'uploadUrl': presigned_url,
                'imageKey': image_key
            })
        }

    except json.JSONDecodeError:
        return error_response(400, 'リクエストBodyが不正です')
    except Exception as e:
```

```
        print(f'Error: {str(e)}')
        return error_response(500, '署名付きURLの生成に失敗しました')

def error_response(status_code, message):
    return {
        'statusCode': status_code,
        'headers': {
            'Content-Type': 'application/json',
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps({
            'status': 'error',
            'message': message
        })
    }
```

ヒント

```
【穴埋め1】'fileName', 'fileType'
【穴埋め2】not file_name
【穴埋め3】not file_type
【穴埋め4】allowed_types
【穴埋め5】uuid.uuid4()
【穴埋め6】images/{unique_id}.{extension}
【穴埋め7】'put_object', BUCKET_NAME, image_key, file_type, 900
```

穴埋め演習の進め方（授業での扱い方）

ステップ1：全体説明

- Lambda関数の役割を説明
- boto3の基本的な使い方を説明
- 穴埋め箇所の説明

ステップ2：個別作業

- 各自でコードを読み、穴埋め
- わからない箇所は資料やドキュメント参照
- 完成したらLambdaにデプロイ

ステップ3：テスト実行

- Lambdaコンソールでテストイベント実行
- エラーが出たら修正
- CloudWatch Logsで確認

ステップ4：答え合わせ

- 正解コードを提示
- 各穴埋め箇所の解説
- よくあるエラーとその対処法

配布資料

Lambda穴埋めワークシート

- 各関数の穴埋めコード
- ヒント集
- boto3リファレンス抜粋

HTTPステータスコード一覧表

```
200: OK（成功）
201: Created（作成成功）
400: Bad Request（リクエスト不正）
404: Not Found（見つからない）
500: Internal Server Error（サーバーエラー）
```

boto3 DynamoDB操作リファレンス

```
# 全件取得
table.scan()

# 条件付き取得
table.scan(
    FilterExpression='category = :cat',
    ExpressionAttributeValues={':cat': 'テスト'}
)

# 1件取得
table.get_item(Key={'postId': 'xxx'})

# 保存
table.put_item(Item={...})

# 削除
table.delete_item(Key={'postId': 'xxx'})
```