Documentazione Test Backend - gruppo Omninext

Candidato: Francesco Muscatelli

Obiettivo Test:

Implementare delle Rest API per la creazione e visualizzazione del modello user, in particolare dovrai creare i seguenti endpoint con i rispettivi metodi per la scrittura/lettura sul db:

- createUser
- getUserById

Ambiente di sviluppo:

AWS, in particolare l'utilizzo dei seguenti servizi:

- Lambda Function
- DynamoDB
- API Gateway

Linguaggio di programmazione:

Ho deciso di adottare il liguaggio di programmazione "Python", dal momento che è quello utilizzato dall'azienda per lo sviluppo.

Fasi:

1. CREAZIONE DATABASE
   Prima di tutto ho creato una tabella in DynamoDB:
   name = "user-list"
   partition key = "userid"

2. CREAZIONE LAMBDA FUNCTION
   Per creare la lambda function per l'API è necessario prima creare un item role cosi che la funzione lo possa utilizzare per accedere alla tabella creata in DybamoDB.
   Role:
   Use case = "Lambda"
   Policies = "CloudWatchFullAccess" (ci permette di accedere ai logs per le richieste API) e "AmazonDynamoDBFullAccess " (ci permette di accedere a DynamoDB per fare operazioni, come modifiche e creazioni)
   Role name = "serverless-api-role"

Ho poi creato la funzione lambda:
Function name = “serverless-api”
Runtime = “Python 3.9”
Permissions = “Use an existing role: serverless-api-role”

3. CREAZIONE API GATEWAY
   Utilizzato per conettere il tutto, ho costruito un “REST API”:
   API name = “serverless-api”
   Endpoint Type = “Regional”

   Ho poi creato al suo interno delle risorse caratterizzate da dei metodi:
   - /health (per controllare la salute dell’ API) : GET
   - /user (per operazioni sul singolo utente): GET, PATCH, POST, DELETE
   - /users (per operazioni su tutti gli utenti): GET

   Ho poi effettuato il Deploy API (per renderlo attivo) creando un nuovo stage:
   Stage name = “us”

4. SCRITTURA CODICE CRUD API IN PYTHON
   Sono state create i seguenti metodi:
   - lambda_handler(event, context)
   - buildResponse(statusCode, body=None)
   - getUserById(userid)
   - getAllUsers()
   - createUser(requestBody)
   - modifyUser(userid, updateKey, updateValue)
   - deleteUser(userid)

   Dal momento che l’oggetto che otteniamo da DynamoDB è in decimali, che non è supportato dalle impostazioni di default di JSON Encoder, è stato necessario definire un encoder personalizzato per i decimali. Questo è stato fatto nel file “custom_encoder.py”.

Implementazioni:

Oltre ai due metodi richesti sono stati implementati 3 metodi aggiuntivi:

- getAllUsers(): per ottenere tutti gli utenti presenti nella tabella
- modifyUser(userid, updateKey, updateValue): per modificare un singolo utente
- deleteUser(userid): per eliminare un singolo utente

È stato inoltre implementato un custom encoder come descritto nel punto 4 delle fasi.

Test:
Per testare l’operato realizzato è stato utilizzato “Postman”, dove sono state svolte semplici operazioni che hanno dato tutte successo.