Name : Atharva Nimbalkar FN27859

Question 1.1

Code :

```python
import datetime

from flask import Flask, render_template

from google.cloud import datastore

datastore_client = datastore.Client()

app = Flask(__name__)

def store_visit(user_id, dt):
    entity = datastore.Entity(key=datastore_client.key("visit"))
    entity.update({"user_id": user_id, "timestamp": dt})

    datastore_client.put(entity)

def fetch_visits(limit):
    query = datastore_client.query(kind="visit")
    query.order = ["-timestamp"]

    visits = list(query.fetch(limit=limit))

    # return [{"user_id": visit["user_id"], "timestamp": visit["timestamp"]} for visit in visits]
    return visits

@app.route("/visit/<user_id>")
def root(user_id):
    # Store the current access time in Datastore.
    timestamp = datetime.datetime.now(tz=datetime.timezone.utc)
    store_visit(user_id, timestamp)

    # Fetch the most recent 10 access times from Datastore.
    visits = fetch_visits(10)

    return render_template("index.html", visits=visits)


if __name__ == "__main__":
    app.run(host="127.0.0.1", port=8080, debug=True)
```

Description of changes made : Original code used to store just times, that was changed to store and entity with both user id and time. Functions store and fetch times were changes to store and fetch visits. Finally the html file was chaged to get a custom output which also included user ids.

Deployed url: https://potent-symbol-455519-n0.uk.r.appspot.com/visit/user

App Engine Link :
https://console.cloud.google.com/appengine?referrer=search&authuser=1&invt=AbtuUA&project=potent-symbol-455519-n0&supportedpurview=project&serviceId=default

Question 1.2

Code :

```python
import datetime

from flask import Flask, render_template

from google.cloud import datastore

'''This code is written to access the google cloud datastore and perform operations on it'''

datastore_client = datastore.Client()

app = Flask(__name__)

def fetch_visits(limit):
    query = datastore_client.query(kind="visit")
    query.order = ["-timestamp"]

    visits = list(query.fetch(limit=limit))

    return visits

def store_visit(user_id, dt):
    entity = datastore.Entity(key=datastore_client.key("visit"))
    entity.update({"user_id": user_id, "timestamp": dt})

    datastore_client.put(entity)

def delete_user_visits(user_id):
    query = datastore_client.query(kind="visit")
    query.add_filter("user_id", "=", user_id)
    visits = query.fetch()

    for visit in visits:
        datastore_client.delete(visit.key)

def search_user_visits(user_id):
    query = datastore_client.query(kind="visit")
    query.add_filter("user_id", "=", user_id)
    query.order = ["-timestamp"]
    visits = list(query.fetch())
    visits_list = [{"user_id": visit["user_id"], "timestamp": visit["timestamp"]} for visit
in visits]
    return visits_list

@app.route("/visit/<user_id>")
def root(user_id):
    timestamp = datetime.datetime.now(datetime.timezone.utc).isoformat()
    store_visit(user_id, timestamp)

    visits = fetch_visits(10)

    return render_template("index.html", visits=visits)

@app.route("/delete/<user_id>")
def delete(user_id):
    delete_user_visits(user_id)
    return f"All visits for user {user_id} have been deleted."

@app.route("/search/<user_id>")
def search(user_id):
    visits = search_user_visits(user_id)
    return f"visits for the {user_id} {visits}"

if __name__ == "__main__":
    app.run(host="127.0.0.1", port=8080, debug=True)

```

Question 1.3

Code:

```python
from boto3 import resource
from boto3.dynamodb.conditions import Key

'''This code was used to create a access a dynamo table initially'''

def create_table(table_name):
    ''' Create a table and return the table object '''
    dynamodb_resource = resource('dynamodb', region_name='us-east-1')

    existing_tables = [table.name for table in dynamodb_resource.tables.all()]
    if table_name in existing_tables:
        print(f"Table '{table_name}' already exists.")
        return dynamodb_resource.Table(table_name)

    table = dynamodb_resource.create_table(
        TableName=table_name,
        KeySchema=[
            {'AttributeName': 'user_id', 'KeyType': 'HASH'},  # Partition key
            {'AttributeName': 'timestamp', 'KeyType': 'RANGE'}  # Sort key
        ],
        AttributeDefinitions=[
            {'AttributeName': 'user_id', 'AttributeType': 'S'},  # String
            {'AttributeName': 'timestamp', 'AttributeType': 'S'}  # String
        ],
        ProvisionedThroughput={
            'ReadCapacityUnits': 5,
            'WriteCapacityUnits': 5
        }
    )
    print("Waiting for table creation...")
    table.wait_until_exists()
    print(f"Table '{table_name}' created successfully!")

    return table

def get_table(table_name):
    ''' Return the table object if exists, otherwise create and return it '''
    dynamodb_resource = resource('dynamodb', region_name='us-east-1')

    try:
        table = dynamodb_resource.Table(table_name)
        table.load()
        print(f"Table '{table_name}' found.")
    except:
        print(f"Table '{table_name}' not found. Creating new table...")
        table = create_table(table_name)

    return table

def search_table(table, user_id):
    ''' Search for records associated with a given user_id '''
    response = table.query(
        KeyConditionExpression=Key('user_id').eq(user_id)
    )
    return response.get('Items', [])

def add_item(table, user_id, timestamp):
    ''' Add one item (row) to the table '''
    item = {
        'user_id': user_id,
        'timestamp': timestamp
    }
    response = table.put_item(Item=item)
    return response

```

```
66. def delete_item(table, user_id, timestamp):
67.     ''' Delete an item from the table using its primary key '''
68.     response = table.delete_item(
69.         Key={
70.             'user_id': user_id,
71.             'timestamp': timestamp
72.         }
73.     )
74.     return response
75.
76. def main():
77.     table_name = "visit"
78.
79.     table = get_table(table_name)
80.
81.     user_id = "user123"
82.     timestamp = "2025-04-02T12:00:00Z"
83.     print("Adding an item...")
84.     add_item(table, user_id, timestamp)
85.
86.     print(f"Searching records for user_id '{user_id}'...")
87.     records = search_table(table, user_id)
88.     print("Records found:", records)
89.
90.     print(f"Deleting record with user_id '{user_id}' and timestamp '{timestamp}'...")
91.     delete_item(table, user_id, timestamp)
92.
93.     print(f"Searching records for user_id '{user_id}' after deletion...")
94.     records = search_table(table, user_id)
95.     print("Records found:", records)
96.
97. if __name__ == "__main__":
98.     main()
99.
```

Question 1.4

Code :

```
 1. import json
 2. import boto3
 3. from boto3.dynamodb.conditions import Key
 4.
 5. dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
 6. table_name = "visit"
 7. table = dynamodb.Table(table_name)
 8.
 9. def lambda_handler(event, context):
10.     """
11.     AWS Lambda function to handle search, insert, and delete requests for the "visit"
DynamoDB table.
12.     Expects an 'action' key in the event:
13.     - "add": Add a visit record of a user
14.     - "search": Retrieve visits for a user
15.     - "delete": Delete all visit records of a user
16.     """
17.
18.     try:
19.         action = event.get("action")
20.
21.         if action == "add":
22.             user_id = event.get("user_id")
23.             timestamp = event.get("timestamp")
24.             if not user_id or not timestamp:
```

```
                return {"statusCode": 400, "body": json.dumps("Missing 'user_id' or 'timestamp'")}

            response = add_item(user_id, timestamp)
            return {"statusCode": 200, "body": json.dumps(response)}

        elif action == "search":
            user_id = event.get("user_id")
            if not user_id:
                return {"statusCode": 400, "body": json.dumps("Missing 'user_id'")}

            records = search_table(user_id)
            return {"statusCode": 200, "body": json.dumps(records)}

        elif action == "delete":
            user_id = event.get("user_id")
            if not user_id:
                return {"statusCode": 400, "body": json.dumps("Missing 'user_id'")}

            response = delete_item(user_id)
            return {"statusCode": 200, "body": json.dumps(response)}

        else:
            return {"statusCode": 400, "body": json.dumps("Invalid action")}

    except Exception as e:
        return {"statusCode": 500, "body": json.dumps(str(e))}

def add_item(user_id, timestamp):
    """
    To Add an item to the DynamoDB table.
    """
    item = {
        "user_id": user_id,
        "timestamp": timestamp
    }
    table.put_item(Item=item)
    return {"message": "Item added successfully", "item": item}

def search_table(user_id):
    """
    To search for all records of a given user.
    """
    response = table.query(
        KeyConditionExpression=Key('user_id').eq(user_id)
    )
    return response.get('Items', [])

def delete_item(user_id):
    """
    To deletes all records of a user from the DynamoDB table.
    """
    response = table.scan(
        FilterExpression=Key('user_id').eq(user_id)
    )
    for item in response['Items']:
        table.delete_item(Key={'user_id': user_id, 'timestamp': item['timestamp']})
    return {"message": "Item deleted successfully"}

```

Question 2.1

Code:

```
import datetime
```

```
 2. import requests
 3.
 4. from flask import Flask, render_template
 5.
 6. from google.cloud import datastore
 7.
 8. '''This code was written to access google datastore and amazon DynamoDb and perform
operations on them simultaneously and is deployed on the app engine'''
 9.
10. datastore_client = datastore.Client()
11.
12. app = Flask(__name__)
13.
14. AWS_LAMBDA_API_URL = "https://yo4fu6xdvc.execute-api.us-east-
1.amazonaws.com/dev/DynamoDBHandler"
15.
16. def store_visit(user_id, dt):
17.     entity = datastore.Entity(key=datastore_client.key("visit"))
18.     entity.update({"user_id": user_id, "timestamp": dt})
19.
20.     datastore_client.put(entity)
21.
22. def store_in_dynamodb(user_id, timestamp):
23.     payload = {
24.         "action": "add",
25.         "user_id": user_id,
26.         "timestamp": timestamp
27.     }
28.     response = requests.post(AWS_LAMBDA_API_URL, json=payload)
29.     return response.json()
30.
31. def fetch_visits(limit):
32.     query = datastore_client.query(kind="visit")
33.     query.order = ["-timestamp"]
34.
35.     visits = list(query.fetch(limit=limit))
36.
37.     return visits
38.
39. def delete_user_visits(user_id):
40.     query = datastore_client.query(kind="visit")
41.     query.add_filter("user_id", "=", user_id)
42.     visits = query.fetch()
43.
44.     for visit in visits:
45.         datastore_client.delete(visit.key)
46.
47. def delete_from_dynamodb(user_id):
48.     payload = {
49.         "action": "delete",
50.         "user_id": user_id
51.     }
52.     response = requests.post(AWS_LAMBDA_API_URL, json=payload)
53.     return response.json()
54.
55. def search_user_visits(user_id):
56.     query = datastore_client.query(kind="visit")
57.     query.add_filter("user_id", "=", user_id)
58.     query.order = ["-timestamp"]
59.     visits = list(query.fetch())
60.     visits_list = [{"user_id": visit["user_id"], "timestamp": visit["timestamp"]} for visit
in visits]
61.     return visits_list
62.
63. def search_in_dynamo(user_id):
64.     payload = {
65.         "action": "search",
66.         "user_id": user_id
67.     }
68.     response = requests.post(AWS_LAMBDA_API_URL, json=payload)
```

```
    return response.json()

@app.route("/visit/<user_id>")
def root(user_id):
    timestamp = datetime.datetime.now(datetime.timezone.utc).isoformat()
    store_visit(user_id, timestamp)
    store_in_dynamodb(user_id, timestamp)

    visits = fetch_visits(10)

    return render_template("index.html", visits=visits)

@app.route("/delete/<user_id>")
def delete(user_id):
    delete_user_visits(user_id)
    delete_from_dynamodb(user_id)
    return f"All visits for user {user_id} have been deleted."

@app.route("/search/<user_id>")
def search(user_id):
    visits = search_user_visits(user_id)
    visits1 = search_in_dynamo(user_id)
    return f"Visits for {user_id} from datastore:<br>{visits}<br><br>Visits from
DynamoDB:<br>{visits1['body']}"



if __name__ == "__main__":
    app.run(host="127.0.0.1", port=8080, debug=True)

```