Backend Developer Technical Assessment

Introduction

This technical assessment evaluates your proficiency with MongoDB, FastAPI, and AWS services (Lambda, Step Functions, S3), along with your problem-solving skills and ability to design effective database schemas.

Part 1: MongoDB Schema Design (45 minutes)

Scenario

You're building an e-commerce platform that needs to store product information, customer data, orders, and inventory tracking.

Task

1. Design a MongoDB schema that follows best practices for this scenario
2. Include considerations for:
    - Data relationships
    - Indexing strategy
    - Schema validation
    - Query optimization
3. Provide sample JSON documents for each collection
4. Explain your design decisions and any tradeoffs considered

Part 2: FastAPI Implementation (1 hour)

Scenario

Using your MongoDB schema from Part 1, implement a basic FastAPI application with the following endpoints:

- `GET /products`: List all products with pagination and filtering
- `GET /products/{id}`: Get a specific product by ID
- `POST /orders`: Create a new order
- `GET /customers/{id}/orders`: Get all orders for a specific customer

Task

1. Implement the FastAPI application with proper error handling
2. Include Pydantic models for data validation

3. Implement proper dependency injection for MongoDB connections
4. Include appropriate documentation using FastAPI's built-in Swagger UI
5. Implement at least one unit test for each endpoint

Part 3: AWS Architecture (45 minutes)

Scenario

You need to implement a product import process that:

1. Monitors an S3 bucket for new CSV files
2. Processes each file (validates data, transforms it)
3. Imports valid products into MongoDB
4. Generates a report of successful/failed imports
5. Notifies appropriate teams via email

Task

1. Design an AWS architecture using Lambda, Step Functions, and S3
2. Create a Step Functions workflow diagram showing the complete process
3. Provide a code outline for at least two Lambda functions
4. Explain how you would handle errors and retries
5. Discuss scaling considerations for large files

Part 4: Problem-Solving Challenge (1 hour)

Scenario

The e-commerce platform is experiencing performance issues:

1. The product search is slow, especially with many filters
2. Order processing sometimes fails under high load
3. Some MongoDB queries are timing out

Task

1. Analyze the provided sample code and database queries
2. Identify potential bottlenecks and inefficiencies
3. Suggest specific improvements with code examples
4. Explain your reasoning for each suggestion
5. Discuss how you would monitor your solution to ensure it resolves the issues

Part 5: Learning & Adaptability (30 minutes)

Scenario

You need to design an efficient ETL pipeline for processing large amounts of product data with complex transformations between various data sources and your MongoDB database.

Task

1. Design an algorithm to efficiently detect and merge duplicate product records across multiple data sources
2. Implement a data structure that optimizes memory usage when processing millions of product records
3. Create a workflow diagram for your ETL pipeline, including:
   - Data extraction from multiple sources (API, CSV, database)
   - Transformation logic with validation rules
   - Loading strategy with error handling
4. Write pseudocode for a key component that uses an appropriate algorithm (e.g., parallel processing, map-reduce)
5. Explain how your solution handles:
   - Data quality issues
   - Processing failures
   - Incremental updates vs. full refreshes

Submission Guidelines

1. Provide all code as functioning files, not just snippets
2. Include a README.md with:
   - Setup instructions
   - Explanations of your approach
   - Any assumptions made
3. Submit via [your preferred submission method]
4. Be prepared to discuss your solution in a follow-up interview

Evaluation Criteria

Candidates will be evaluated on:

1. MongoDB schema design best practices
2. Code quality and organization

3. API design and implementation
4. AWS architecture knowledge
5. Problem-solving approach
6. Ability to learn and adapt
7. Communication and documentation

Sample Code and Resources

MongoDB Connection Helper (Python)

```python
from motor.motor_asyncio import AsyncIOMotorClient
from typing import Dict, Any
import os
from contextlib import asynccontextmanager

class MongoDBConnection:
    client: AsyncIOMotorClient = None

    @classmethod
    async def connect_to_mongodb(cls):
        cls.client = AsyncIOMotorClient(os.getenv("MONGODB_URL"))
        return cls.client

    @classmethod
    async def close_mongodb_connection(cls):
        if cls.client:
            cls.client.close()

    @classmethod
    @asynccontextmanager
    async def get_collection(cls, db_name: str, collection_name: str):
        if not cls.client:
            await cls.connect_to_mongodb()
        collection = cls.client[db_name][collection_name]
        try:
            yield collection
        finally:
            pass
```

Sample FastAPI Endpoint

```python
from fastapi import APIRouter, Depends, HTTPException, Query
from typing import List, Optional
from pydantic import BaseModel
from .mongodb import MongoDBConnection
import pymongo

router = APIRouter()

class Product(BaseModel):
    name: str
    price: float
    description: str
    category: str
    inventory_count: int

@router.get("/products")
async def get_products(
    skip: int = Query(0, ge=0),
    limit: int = Query(20, ge=1, le=100),
    category: Optional[str] = None
):
    query = {}
    if category:
        query["category"] = category

    async with MongoDBConnection.get_collection("ecommerce", "products") as collection
        cursor = collection.find(query).skip(skip).limit(limit)
        cursor.sort("name", pymongo.ASCENDING)
        products = await cursor.to_list(length=limit)

    return {"total": await collection.count_documents(query), "products": products}
```

AWS Lambda Function Example

```
python
```

```python
import json
import boto3
import pandas as pd
import pymongo
from io import StringIO

s3_client = boto3.client('s3')
mongo_client = pymongo.MongoClient('mongodb://your-connection-string')
db = mongo_client['ecommerce']
collection = db['products']

def lambda_handler(event, context):
    # Get the S3 bucket and key from the event
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    try:
        # Get the file from S3
        response = s3_client.get_object(Bucket=bucket, Key=key)
        csv_content = response['Body'].read().decode('utf-8')

        # Process the CSV
        df = pd.read_csv(StringIO(csv_content))

        # Validate and transform data
        # ... validation and transformation code ...

        # Import to MongoDB
        records = df.to_dict('records')
        result = collection.insert_many(records)

        return {
            'statusCode': 200,
            'body': json.dumps({
                'message': f'Successfully processed {len(records)} records',
                'imported_count': len(result.inserted_ids)
            })
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': json.dumps({
                'message': f'Error processing file: {str(e)}'
            })
        }
```

Additional Notes for Candidates

- You may use any Python libraries that help you complete the tasks efficiently
- Focus on clean, maintainable code rather than premature optimization
- Document your code well and explain your design decisions
- Consider edge cases and error scenarios
- Feel free to ask clarifying questions before or during the assessment