Weekly Report: MD. Ariful Islam Shakil

Week-07 (May 05 – May 09)

1. Digital Resource Management Project

FastAPI Backend Development (Images Upload Features)

In this module, I implemented a FastAPI-based image service that handles uploading, tagging, and retrieving image metadata using AWS S3 and DynamoDB. I integrated the following key features:

- **S3 Upload Integration:** I created a function to upload images to a specific folder in an S3 bucket while preserving metadata like content type. It also handles path management and error logging.
- **Presigned URL Generation:** I implemented a utility that securely generates time-limited presigned URLs for image download access from S3.
- **Image ID Generation:** I developed a helper function to auto-increment unique image IDs (e.g., IMG001, IMG002) based on the existing records in the database.
- **Image Tagging with YOLOv8:** I integrated a custom yolo_api module that processes images on upload to detect objects and generate tags, along with image dimension and file size extraction.
- **Image Metadata Management with DynamoDB:** On every upload, I save the image's metadata (including ID, employee ID, tags, timestamp, size, dimension, and S3 path) into a centralized DynamoDB record under a fixed partition key (Arifs_images).
- **Image Querying by Tags and Employee ID:** I created a search function that returns image metadata if any of the queried tags match, with optional filtering by employee ID. Presigned download links are included in the response.

2. Unit Testing:

Employee Services

In this module, I implemented comprehensive unit tests using **Pytest** and **unittest.mock** to validate the core functionalities of the employee_services backend. My focus was on ensuring correctness, stability, and exception handling for the following operations:

Get Employees:

I tested the get_employees function to retrieve only active employees. I mocked DynamoDB responses and handled edge cases, including missing keys, empty items, and exceptions during database access.

Get Employee by ID:

I verified the ability to fetch a specific employee using their u_id. I wrote test cases for valid matches, unmatched u_ids, missing employee lists, and raised HTTPExceptions when appropriate.

Add Employee:

I validated the add_employee logic by mocking the unique ID generation, retrieving the current state from DynamoDB, and simulating both update and new record insertion scenarios. I also tested invalid name inputs (blank, only digits) to enforce name validation rules.

Update Employee:

I tested employee record updates by checking if a given u_id exists and if the name is valid. My tests covered successful updates, missing employee lists, unmatched u_ids, and invalid name formats.

Delete Employee:

I wrote test cases to soft-delete an employee (by marking active=False) and ensured that only the target employee was updated. I handled situations where the employee was not found, ensuring proper error handling with appropriate status codes.

Overall, this test suite gave me hands-on experience with **mocking AWS DynamoDB interactions**, handling **FastAPI exceptions**, and writing **realistic, production-grade test cases** to ensure high code quality and reliability.

Images Services (Testing)

This module provides comprehensive test coverage for image metadata management functions implemented in images_service.py, which interacts with AWS DynamoDB and S3. It includes the following tested functionalities:

- **Uploading an image to S3 and storing metadata** using add_image_metadata(), including YOLOv8 tag generation, unique image ID creation, and proper DynamoDB structure.
- **Filtering and retrieving image metadata** using get_images_info() based on image tags and optionally emp_id, and generating a presigned URL for download.
- **Supporting utilities** like:
    - upload_to_s3() for S3 integration,
    - generate_presigned_url() for temporary image links,
    - create_unique_img_id() for generating new image IDs based on previously stored data.

  Mocking is used to simulate external dependencies such as boto3 services and YOLO model integration (yolo_api), ensuring the tests are isolated and fast.

  Each test ensures:

    - Proper handling of happy paths (successful metadata creation and retrieval),
    - Graceful handling of exceptions (upload failures, missing data),
    - Accurate filtering logic (by tags and/or emp_id),
    - Validation of presigned URL generation.

3. React Frontend – Employee Image Upload & Query Interface

As part of the **Alpha-AI team project**, I developed a React-based frontend for managing employee-related image uploads and intelligent querying using YOLO-generated tags. The work covers two major components:

UploadImage.jsx – Employee Image Upload Panel

- Integrated **employee data** dynamically from the backend using EmployeeServices.
- Enabled **image file selection and preview** using useState and native browser APIs (URL.createObjectURL()).
- Implemented **image upload to AWS S3** via a backend API, along with **metadata storage** (e.g., image ID, employee ID, size, dimension, tags).
- Showed appropriate user feedback for upload success/failure using loading states and status messages.

- Handled real-time selection of employees from a dropdown populated via backend.

QueriyImages.jsx – Smart Image Search Using Tags

- Built a **search UI** for filtering stored images using YOLOv8 tags and optional employee filtering.
- Parsed user-input tags (comma, dot, dash, underscore, or space-separated) into a clean searchable array.
- Queried image metadata from the backend (DynamoDB) and received **presigned S3 URLs** to display actual image previews.
- Rendered **responsive, styled results** including image ID, employee ID, tags, size, and dimensions.
- Included **robust error handling** and feedback for failed or empty searches.