Product Requirements Document (PRD)

Project: Cloud Image Upload and Management Service

Prepared for: Lead Platform Engineer Coding Exercise

Language: Python 3.7+ | Cloud Stack: AWS (API Gateway, Lambda, S3, DynamoDB) | Local Development: LocalStack + Docker Compose

1. Overview

This project defines a scalable cloud-native image upload and management service, similar in functionality to the image management layer of Instagram. The service enables users to upload, store, retrieve, list, and delete images. Metadata associated with each image is persisted in a NoSQL store for efficient querying and filtering.

2. Objectives

Develop a robust backend that supports concurrent uploads from multiple users.
Ensure scalability, reliability, and easy deployment on AWS using Lambda, S3, and DynamoDB.
Enable local development using LocalStack for AWS service simulation.
Provide full API documentation and unit test coverage.

3. Core Features and Requirements

Feature	Endpoint	Description
Image Upload API	POST /images/upload	Upload an image with metadata to S3 and DynamoDB.
List Images API	GET /images	Retrieve all images with filtering options (user_id, tag, date_range).
View/Download Image API	GET /images/{image_id}	Retrieve metadata and presigned S3 URL for download.
Delete Image API	DELETE /images/{image_id}	Delete image object and metadata record.

4. Architecture

Components include API Gateway (for routing), AWS Lambda (for business logic), S3 (for image storage), and DynamoDB (for metadata persistence).

DynamoDB Data Model

Attribute	Type	Description
image_id	String (PK)	Unique identifier
user_id	String	Owner of the image
title	String	Title of image
description	String	Optional description
tags	List	Image tags

upload_time	String (ISO 8601)	Upload timestamp
s3_url	String	S3 object URL

5. Local Development Setup

Use LocalStack with Docker Compose to simulate AWS services (API Gateway, Lambda, S3, DynamoDB). Configure AWS CLI locally with test credentials.

6. Testing

Write unit tests using pytest/unittest for all Lambda functions. Mock AWS services using moto and pytest-mock to simulate S3 and DynamoDB.

7. Scalability Considerations

Lambda concurrency allows multi-user uploads, S3 scales automatically, and DynamoDB supports parallel reads/writes with GSI for filtering.

8. Deliverables

Python source code for Lambda functions.
docker-compose.yml for LocalStack setup.
Unit tests and coverage report.
Swagger/Postman API documentation.
README with setup and usage instructions.