This implementation follows a serverless microservices approach using AWS Lambda, API Gateway, S3, DynamoDB, OpenSearch, SQS, and Cognito.

# 1. SYSTEM OVERVIEW

I. **Document Upload (S3, Lambda functions)**:

Users upload documents to Amazon S3 storage using Lambda functions for additional flexibility and logic. These documents could be invoices, medical records, or any other scanned content.

II. **OCR Text Extraction (Amazon Textract)**:

When a document is uploaded to S3, an automated process triggers Amazon Textract, a fully managed OCR service.

Textract extracts text, tables, forms, and other relevant metadata from scanned documents

The output of the extraction process is a JSON structure containing the extracted text and metadata (e.g., page numbers, text blocks).

III. **Document Indexing (OpenSearch)**:

The extracted text and relevant metadata (document type, date, document ID) are indexed in Amazon OpenSearch.

OpenSearch allows for fast and flexible querying based on keywords, document type, metadata filters, and other search criteria.

The system can store links to the original S3 document in OpenSearch, enabling users to access the original document from search results.

IV. **Search & Query (OpenSearch)**:

Users can search for documents using a web interface (via OpenSearch Dashboards) or through an API.

Full-text search allows users to search across extracted text.

Filters and advanced search features help narrow down results based on metadata (e.g., document date, type).

V. **Document Retrieval**:

From OpenSearch search results, users can retrieve the original document stored in Amazon S3 by clicking on the link to the S3 object.

This provides seamless access to both the extracted text in OpenSearch and the original document in S3.

## VI. Services

| Service | Responsibilities | AWS Services Used | Lambda Functions |
|---|---|---|---|
| **Document Service** | Handle document CRUD, versioning, and folder management | S3, DynamoDB | createDocument, updateDocument, deleteDocument, getDocumentVersion |
| **OCR Service** | Handle OCR jobs, store extracted text, and index results | S3, Textract, OpenSearch | startOCRJob, getOCRResults, indexOCRData |
| **Permissions Service** | Manage role-based access and document permissions | Cognito/DynamoDB | assignUserRole, checkDocumentAccess |
| **Project Service** | Handle project creation, membership, and settings | DynamoDB | createProject, addMemberToProject |
| **Audit Log Service** | Track all actions for compliance and debugging | DynamoDB, CloudWatch | logAction, getAuditLogs |

# 2. DESIGN CONSIDERATIONS

**I.** Serverless

The solution leverages a serverless architecture using AWS services Lambda, API Gateway, S3, and DynamoDB. This allows the system to automatically scale based on demand, without the need for managing servers or infrastructure manually.

Reduced Operational Overhead: No need to provision or manage servers to handle incoming requests. AWS automatically handles resource scaling based on the number of incoming requests.

Cost Efficiency: Lambda and other serverless services like S3 use a pay-per-use model. This means you only pay for the compute power and storage you actually use, avoiding the costs associated with idle server time.

Flexibility: With Lambda, it's easier to iterate on specific business logic and scale functions independently based on individual demands (like OCR processing, document search, etc.), without worrying about system-wide scaling.

Event-Driven Architecture: Services S3 (document uploads) and SQS (OCR processing) are event-driven, which naturally integrates with Lambda to trigger workflows when certain events occur.

**II.** Scalability

Lambda: AWS Lambda functions can automatically scale up to handle many simultaneous requests. For example, when many users upload documents or trigger OCR jobs simultaneously, Lambda can handle these in parallel, ensuring there is no bottleneck.

DynamoDB: DynamoDB scales horizontally by automatically partitioning data across multiple servers. As the number of documents and metadata grows, DynamoDB can scale seamlessly without manual intervention, allowing for fast metadata storage and querying.

OpenSearch: OpenSearch is designed for fast, scalable search operations. It can handle a growing number of documents and search queries by distributing indexing across multiple nodes.

As the user base grows, the system will dynamically scale its Lambda functions to handle an increased number of document uploads, metadata indexing, and search queries. If the demand on OpenSearch grows (due to frequent searches), additional OpenSearch nodes will be automatically provisioned.

**III.** Fault Tolerance

Lambda Retries: Lambda functions have built-in retry mechanisms for transient errors. For example, if a Lambda function fails while processing a document (e.g., due to temporary unavailability of an external service), AWS will automatically retry the function a set number of times before marking it as failed.

SQS Dead Letter Queues (DLQs): If a document processing event (like OCR) fails even after retries, the failure is captured in a DLQ, ensuring the document does not get permanently lost or skipped. This allows for manual intervention or reprocessing.

DynamoDB and S3 Redundancy: Both DynamoDB and S3 are designed to be highly available. DynamoDB automatically replicates data across multiple availability zones, while S3 provides redundancy by storing copies of objects across multiple physical locations. If one availability zone experiences a failure, the service will still be available.

If OCR processing via Textract fails due to a bad document format, the system logs this failure and can retry or notify the user via a monitoring service, ensuring that the document is eventually processed.

## **IV.** Performance Considerations

Cold Starts: Lambda cold starts can introduce latency. To minimize this impact, the solution can optimize the size and efficiency of Lambda functions and also ensure that Lambda functions used for frequent tasks (like document uploads) are designed to minimize initialization time.

Caching Mechanisms: In some cases, for frequent queries or searches, results can be cached in-memory (using services like DynamoDB Accelerator (DAX)) to speed up retrieval. Out of scope in current design.

Parallel Processing: Lambda allows for parallel processing, such as when multiple OCR jobs are triggered simultaneously or when documents are uploaded in bulk. This helps minimize delays and improves processing time.

OpenSearch Optimization: OpenSearch allows for efficient indexing and querying. To maintain performance, the system can apply indexing strategies that optimize search queries, such as full-text search and metadata filtering.

For large sets of documents uploaded at once, Lambda functions will process them in parallel to ensure that the system can handle large volumes without affecting performance. Additionally, indexing in OpenSearch will be optimized to allow for quick searches across millions of documents.

## **V.** Security

Authentication with Cognito: Amazon Cognito handles authentication, allowing users to log in and access the system securely. The solution uses role-based access control (RBAC) to ensure that users can only access documents or perform actions they are authorized to do (e.g., only admins can delete categories).

Encryption: All documents stored in S3 are encrypted at rest using AWS-managed keys (or custom keys via AWS KMS). Similarly, DynamoDB data can be encrypted using DynamoDB's encryption feature.

Access Control: Only authorized users with valid JWT tokens (via Cognito) can access the document upload, search, and retrieval features. API Gateway uses these tokens to ensure that users can only access the appropriate resources.

Only authorized users can upload documents or access search results. Users with the correct role (admin) can manage categories, while regular users can upload documents, search, and view their own documents. Security is also reinforced through encryption of documents both at rest and in transit.

**VI.** Cost Optimization

Pay-per-Use Model: Since most services like Lambda, API Gateway, S3, and DynamoDB are pay-per-use, the system scales in cost with the volume of operations. For example, if only 100 documents are uploaded in a month, the costs will reflect that, and if 50,000 are uploaded, the system will automatically scale and cost more accordingly.

# 3. MVP SCENARIO

**I. Ability to Create/Edit/Delete Projects**

**Services Used:**

- **API Gateway** - Routes requests to appropriate Lambda functions.
- **Lambda Functions** - Handles CRUD operations for projects.
- **DynamoDB** - Stores project details.
- **Cognito** - Manages authentication and authorization.

**User Flow:**

1. User sends API request via frontend:
    - POST /projects - Creates a new project.
    - PUT /projects/{id} - Updates project details.
    - DELETE /projects/{id} - Deletes a project.
2. API Gateway routes the request to the respective Lambda function.
3. The Lambda function updates the DynamoDB "Project" table.
4. Cognito ensures only authorized users perform actions.
5. Upon success, the Lambda returns a confirmation response to the user.

**II. Ability to Create/Edit/Delete Categories (Admin Only Feature)**

**Services Used:**

- **API Gateway** - Handles requests.
- **Lambda Functions** - Processes category CRUD operations.
- **DynamoDB** - Stores categories.
- **Cognito Role Restriction** - Only **admins** can modify categories.

**User Flow:**

1. Admin sends API request:
    - POST /categories - Creates a category.
    - PUT /categories/{id} - Updates category details.
    - DELETE /categories/{id} - Deletes a category.
2. API Gateway routes the request to Lambda.
3. The Lambda function validates the user role (only admins can modify categories via Cognito roles).
4. The Lambda function updates the "DocumentCategory" table in DynamoDB.
5. The system returns a success response to the admin.

**III. Ability to Upload Project Documents & Set Categories**

**Services Used:**

- **S3 Bucket** - Stores document files.
- **API Gateway + Lambda** - Generates pre-signed URLs for document upload.
- **DynamoDB**- Stores document metadata.
- **Cognito Policies** - Restricts document access.

**User Flow:**

1. User selects a file to upload & assigns a category.
2. Frontend requests an S3 pre-signed URL from API Gateway (POST /documents/upload).
3. API Gateway calls a Lambda function to generate the pre-signed URL.
4. User uploads the file to Amazon S3 using the pre-signed URL.
5. After successful upload, an S3 event triggers another Lambda to:
    - Store metadata (file name, project, category, upload time) in DynamoDB.

- Send an SQS message for OCR processing (if applicable).

**IV. Ability to Search Documents by Project, Categories, Notes, and Upload Date**

**Services Used:**

- **Amazon OpenSearch (Elasticsearch)** - Indexes & searches documents.
- **API Gateway + Lambda** - Processes search queries.
- **DynamoDB** - Stores metadata for filtering.
- **S3 Signed URLs** - Provides access to documents.

**User Flow:**

1. User sends a search request:
    - Example: GET /documents/search?project_id=123&category_id=987&date_range=last30days&query=red
2. API Gateway routes the request to a Lambda search function.
3. The Lambda function:
    - Queries OpenSearch for text-based searches.
    - Queries DynamoDB for category & date-based filtering.
4. The Lambda function returns matching documents with metadata.
5. The user receives document links (S3 pre-signed URLs) to download/view files.

# 4. Storage calculations for 500 users

Each document will be stored, and OCR will process the text from the scanned document to make it searchable. The amount of storage will depend on several factors:

We Assume:

- Assume an average document size of 1MB.
- Each user could upload around 100 documents per month.
- OCR Data Storage: OCR will generate metadata and text from the document. Let's assume the OCR data for each document is an additional 0.2MB (for OCR text output).

*Monthly Storage Calculation:*

- Documents per user per month: 100 documents
- Total Documents per month for 500 users: 100 * 500 = 50,000 documents

- Document Size: 1MB per document
- OCR Output Size per Document: 0.2MB

So, the storage requirements for documents and OCR output can be calculated as:

- Total document storage per month: 50,000 * 1MB = 50,000MB = 50GB per month
- Total OCR output storage per month: 50,000 * 0.2MB = 10,000MB = 10GB per month

Thus, the total monthly storage requirement would be:

- 50GB (document storage) + 10GB (OCR output) = 60GB per month