**Loan File Review Solution – Detailed Technical & Functional Approach (orchestration)**

**1. Introduction & High-Level Overview**

The **Loan File Review** solution automates the ingestion and processing of documents for each client’s (tenant) engagement. Documents come from **SharePoint** folders, are placed in **Azure Blob Storage**, and then orchestrated via **Airflow** for other services run. Key steps:

1. **User/Client** registers an engagement via a **UI** (the FDF platform), providing:
   * **SharePoint site URL** & folder path
   * **Authentication** credentials (username/password or OAuth client secrets)
   * **Industry** (CRE, C&I, etc.) & Engagement details
2. **Document Intake (Ingestion) Service** (Python) pulls docs from SharePoint, places them in Blob, creates a run record in MongoDB.
3. **Airflow DAG** is triggered, calls document extraction and indexing services in parallel, storing final statuses.
4. **Result**: Documents are fully processed (extracted, indexed, optionally LLM-processed), with a final run status in **MongoDB**.

**High-Level Tools**

* **SharePoint**: source of docs; multiple sites possible.
* **Azure Blob**: holds documents post-ingestion.
* **MongoDB**: logs run states, doc statuses.
* **Airflow**: orchestrates doc extraction/indexing.
* **Document Extraction**: external service or microservice.
* **Vector DB or LLM**: for indexing or advanced processing.

**3. 3. User Journey & Detailed Flow for Orchestration**

The pipeline is triggered whenever a user sets up a new loan review or chooses to run an existing engagement.

**3.1. UI Input & Engagement Registration**

1. **User** logs into the FDF platform, clicks “Create Loan Review Engagement.”
2. **Form**: They fill in:
   * **Tenant**: e.g. “CITI,” “AppleBank,” etc.
   * **Engagement ID**: e.g. “ENG\_CITI\_2024\_Q1.” Dynamically create
   * **Industry**: e.g. “CRE” or “C&I.”
   * **SharePoint** site URL\*\*: e.g. https://sharepoint.com/sites/citidocs.
   * **SharePoint** folder\*\*: e.g. /SharedDocuments/data (which might have subfolders).
   * **Auth** method\*\*: either username/password or client\_id/client\_secret (OAuth).
   * **(Optional)** year/quarter\*\* for classification, but **not** used in the blob path

On **Submit**, the UI calls something like:

* POST /api/engagements with JSON

{

"tenant\_id": "CITI",

"engagement\_id": "ENG\_CITI\_2024\_Q1",

"industry\_type": "CRE",

"sharepoint\_site\_url": "https://sharepoint.com/sites/citidocs",

"sharepoint\_folder\_path": "/SharedDocuments/data",

"auth\_method": "username\_password",

"username": "citi\_ingest@myorg.com",

"password\_or\_secret": "...",

"year": 2024,

"quarter": 1

}

The system or the user then **triggers** the run by clicking on submit.

**3.2. SharePoint Recursive Connection & Authentication**

**When** the user triggers a run for the engagement, the system must **connect** to the specified SharePoint site folder:

1. **Auth**:

* If auth\_method=“username\_password”, we do:

ctx = ClientContext(site\_url).with\_credentials(UserCredential(username, password))

* If auth\_method=“client\_secret”, we get a token from Azure AD.

1. **Recursively** list\*\* the folder\*\*: If the user’s folder is /SharedDocuments/data, we might find subfolders like Contracts/, Financials/, etc. We BFS or DFS them:

* For each subfolder, we do folder.files → load → store in a list.
* If it has more subfolders, we recurse until no more.

1. Usually we only handle certain extensions (pdf, docx, xlsx).

**Edge**: If the user tries to connect but the credentials fail or the site/folder doesn’t exist, we log an error in the run doc. The run becomes “failed” quickly. The user must correct the config.

**3.3. Document Intake (Ingestion) Service (intake.py)**

intake.py is the Python microservice or script that:

1. Retrieves engagement info\*\* (tenant, industry, sharepoint, credentials) from the DB or from the request if ephemeral.
2. Authenticates with SharePoint\*\* as above.
3. Recursively enumerates\*\* all docs\*\*.
4. Uploads each doc\*\* to Azure Blob:
   * Container name\*\*: e.g. "{tenant\_id}-loan-docs".
   * Path\*\*: "{industry\_type}/{engagement\_id}/filename"
   * No year/quarter subfolder. Overwrites if doc is the same name. Because we have versioning, older versions remain.
5. Builds a run doc\*\* in Mongo with status="ingestion\_completed" and an array of docs:

{

"run\_id": "RUN\_CITI\_CRE\_2024\_Q1\_001",

"tenant\_id": "CITI",

"industry\_type": "CRE",

"engagement\_id": "ENG\_CITI\_2024\_Q1",

"year": 2024,

"quarter": 1,

"status": "ingestion\_completed",

"docs": [

{

"doc\_id": "DOC12345",

"file\_name": "loan\_abc.pdf",

"blob\_path": "CRE/ENG\_CITI\_2024\_Q1/loan\_abc.pdf",

"status": "Uploaded"

}

// possibly more docs

],

"start\_time": "2024-01-10T10:00:00Z",

"end\_time": null

}

**Triggers** the **Airflow DAG**:

* e.g. POST /api/v1/dags/loan\_review\_dag/dagRuns with {"dag\_run\_id":"RUN\_CITI\_CRE\_2024\_Q1\_001","conf":{"run\_id":"RUN\_CITI\_CRE\_2024\_Q1\_001"}}.

3.6. **Airflow DAG Orchestration & Parallel Calls**

Once the run is “ingestion\_completed,” the **Airflow** DAG named "loan\_review\_dag" is triggered with run\_id.

**DAG Steps** (in code/pseudocode):

1. **prepare\_run**:
   * run\_doc = db.runs.find\_one({ "run\_id": <run\_id> })
   * If run\_doc.status != "ingestion\_completed", fail. Else proceed.
   * Return doc\_list = run\_doc["docs"].
2. **doc\_extraction** (Parallel via Task Mapping):
   * For each doc in doc\_list, create a sub-task that:
     1. **Generate** a SAS URL with container\_name + doc.blob\_path.
     2. **POST** to Document Extraction:
     3. If success => set doc.status="Recognized". If fail => doc.status="Failed" with error\_details.
     4. Update run doc in Mongo accordingly.
3. **doc\_indexing** (Parallel):

* For each doc that is “Recognized,” call an indexing/LLM service:

{ "doc\_id": doc.doc\_id, "text": "extracted\_text" }

* If success => doc=“Indexed.” If fail => doc=“Failed.”
* Update run doc in Mongo.

**Parallel** approach\*\*: Because we do “Task Mapping,” if there are N docs, we get N sub-tasks for extraction (and N for indexing). The system thus can handle multiple docs concurrently. If a doc fails extraction, we do not kill the entire run; we just mark that doc as “Failed.” The DAG’s final step sees partial success.

**Negative point\*\*: If the doc extraction service is slow, it can hamper concurrency. We plan to scale Airflow worker slots if we have many docs.**