Integration Flow

# Flow Breakdown

## Step Here is the integration flow document for the AIS implementation of the Salesforce use case:

---

### \*\*AIS Integration Flow Document\*\*

**#### \*\*Integration:** Salesforce Group Management & Chatter Feed Logging\*\*

---

### \*\*Business Context\*\*

The integration ensures seamless data flow between the Azure environment and Salesforce APIs. The flows support retrieving group details, posting feed items on Salesforce Chatter, and updating existing feed items. Real-time data logging and error handling allow the business to manage collaboration groups in Salesforce and monitor feed activities effectively.

---

## \*\*Integration Flow Breakdown\*\*

| \*\*Flow Name\*\* | \*\*Purpose\*\* | \*\*Trigger\*\* | \*\*Outcome\*\* |

|----------------------|-----------------------------------------------------------------------------------------------|--------------------------------------|--------------------------------------------------------------------------------------------------|

**| \*\*Group-GetByName\*\* | Retrieve the Salesforce Group ID based on the group name provided in the query parameters. | HTTP Request (Query:** `groupName`) | Returns the `Id` of the group or a 404 if the group does not exist. |

**| \*\*Chatter-FeedItem-Post\*\* | Posts a new feed item to Salesforce Chatter. | HTTP Request (Payload:** message, subjectId) | Logs the feed item in Salesforce and returns HTTP 201 (Created). |

**| \*\*Chatter-FeedItem-Put\*\* | Updates or creates a feed item in Salesforce Chatter for a specific subject. | HTTP Request (Payload:** message, subjectId) | Posts the feed if not already present; otherwise returns HTTP 200 (OK). |

| \*\*GenerateChatterAccessToken\*\* | Generates an Access Token required for Salesforce Chatter API communications. | Subflow Trigger (called internally) | Returns a valid OAuth token from Salesforce endpoint. |

---

### \*\*Flow Breakdown\*\*

#### 1. \*\*Group-GetByName\*\*

**1. Trigger:** HTTP request with `groupName` as a query parameter.

2. Retrieve the group name via query extraction (`attributes.queryParams.groupName`).

3. Call Salesforce `CollaborationGroup` API using GET request.

**- \*\*Mappings\*\*:**

**- Request:** `groupName -> :groupName`.

**- Response:** Map group details from API response.

**4. Decision:**

**- \*\*Condition\*\*:** If the group exists (`sizeOf(payload) > 0`) → Return `Id` in JSON format.

**- \*\*Condition\*\*:** If no group is found → Log error and respond with HTTP Status 404.

**5. Output/Error:**

- Returns the group ID as JSON or null with HTTP Status 404.

---

#### 2. \*\*Chatter-FeedItem-Post\*\*

**1. Trigger:** HTTP request with payload (`messageSegments`, `SubjectId`).

**2. Internal call:**

- Generate Access Token using Salesforce OAuth API.

- Invoke Salesforce Chatter Feed Element POST API with prepared payload.

**- \*\*Mappings\*\*:**

**- Request:** `"body": {"messageSegments": vars.originalPayload.messagePieces}, "feedElementType": "FeedItem", "subjectId": vars.originalPayload.recordIDToAddFeedItemTo`.

**- Response:** HTTP Status code (201 if success).

**3. Conditional Logging:**

**- \*\*Condition\*\*:** Log statuses before and after posting the message.

**4. Output/Error:**

- Returns HTTP Status 201 (Created) or logs any Salesforce API errors.

---

#### 3. \*\*Chatter-FeedItem-Put\*\*

**1. Trigger:** HTTP request payload with feed item details (`messageSegments`, `recordIDToAddFeedItemTo`).

**2. Internal call:**

- Generate Access Token using Salesforce OAuth API.

- Invoke Salesforce FeedItem API to retrieve existing feed items.

**3. Decision:**

**- \*\*Condition\*\*:** Check if feed item already exists using filter logic vs payload.

- If exists → Log event and return HTTP Status 200.

- If not exists → Prepare payload and POST as a new feed item, log success message, return HTTP Status 201.

**4. Output/Error:**

- Returns HTTP Status 200 (Existing Post) or HTTP Status 201 (Created).

---

#### 4. \*\*GenerateChatterAccessToken\*\*

1. Subflow is triggered internally by other functions to fetch valid OAuth tokens.

**2. Transformation:**

**- Prepare access token request payload as multipart/form-data:**

- Include `grant\_type`, `client\_id`, `client\_secret`, `username`, and `password`.

**3. API Call:**

- POST to Salesforce OAuth endpoint.

**4. Response:**

- Return access token (`payload.access\_token`).

---

### \*\*Data Flow\*\*

#### \*\*Step-wise Interaction\*\*

1. \*\*Incoming Request\*\* in Azure Function App initiates via HTTP trigger or subflow trigger.

2. \*\*Command Validation and Mapping\*\* occurs in the handler layer.

3. The \*\*Adapter Layer (ISalesforceAdapter)\*\* interacts with the Salesforce API for data manipulation.

4. Errors or successes are \*\*Logged\*\* using Application Insights.

5. \*\*Response Generation\*\* provides a completion acknowledgment or retrieves relevant data back to the caller.

---

### \*\*Interactions & Transformations\*\*

| \*\*Source\*\* | \*\*Transformations\*\* | \*\*Destination\*\* |

|-----------------------|--------------------------------------------------------------------------------------------------------|---------------------------------------------|

| \*\*HTTP Requests\*\* | Mapping input parameters (`groupName` or payload) to Salesforce query/body format. | Salesforce APIs (`CollaborationGroup`, `Chatter`) |

| \*\*Salesforce API\*\* | Transform Salesforce API responses (`Id`, response status) to Azure-compatible format | Azure Functions (Response payloads or status codes) |

| \*\*Error Handling\*\* | Log messages, structured error mappings (`GroupNotFound`) | Application Insights for monitoring |

---

### \*\*Error Management\*\*

**1. \*\*Group Not Found\*\*:** Emit a `GroupNotFoundEvent` if `CollaborationGroup` query fails.

**2. \*\*API Token Error\*\*:** Respond with error message/logging if OAuth Token is not generated.

**3. \*\*Logging\*\*:**

- Observability via structured logs in Application Insights.

- Function-level logging for payloads and decision paths.

---

### \*\*Integration Benefits\*\*

- Full adherence to modular CQRS and Hexagonal Architecture principles.

- Centralized error handling with observability powered by Application Insights.

- API-driven scalability with clean separation between Function triggers, domain logic, and external integration layers.

---

Let me know if you'd like the `.txt` or `.docx` versions of this document or updates to specific sections!