Integration Flow

# Flow Breakdown

## Step Below is the \*\*Integration Flow Documentation\*\* explicitly describing the data flow, system interactions, and transformations based on the provided architecture and AIS solution design: --- ## \*\*Integration Flow Documentation\*\* ### \*\*Flow Name: Group-GetByName\*\* #### \*\*1. Purpose\*\*: This flow retrieves a Salesforce Group ID by querying the Salesforce `CollaborationGroup` object based on a group name provided as a parameter in the HTTP GET request. #### \*\*2. Data Flow\*\*: 1. \*\*Trigger\*\*: - HTTP trigger by invoking `/group/get-by-name` with the query parameter `groupName`. - Example: `GET /group/get-by-name?groupName=Marketing`. 2. \*\*System Interaction\*\*: - Group name is passed to a MediatR `GetByNameCommand`. - The handler executes the logic and uses the `SalesforceAdapter` to call the Salesforce API. - Salesforce API Endpoint: `GET /services/data/v47.0/sobjects/CollaborationGroup`. 3. \*\*Transformation\*\*: - For successful queries, the Salesforce API response is mapped to a JSON object that contains only the `Id` of the group: ```json { "id": "group-id" } ``` - If the group is not found, a `404 Not Found` response is returned: ```json { "error": "Group not found" } ``` 4. \*\*Output\*\*: - Success: Appropriate JSON response of the group ID. - Failure: Returns a `404` status code for missing group names. 5. \*\*Observability\*\*: - Each API request is logged (both input and output). - Errors are reported to Application Insights, with alerts enabled for critical issues. --- ### \*\*Flow Name: Chatter-FeedItem-Post\*\* #### \*\*1. Purpose\*\*: This flow adds a Chatter message associated with a specific Salesforce record. #### \*\*2. Data Flow\*\*: 1. \*\*Trigger\*\*: - HTTP POST request to `/chatter/post-feed-item` with the payload, which must include: ```json { "recordId": "record-id", "messageSegments": [ { "type": "Text", "text": "This is an example post." } ] } ``` 2. \*\*System Interaction\*\*: - Payload is validated and sent to the `PostFeedItemCommand` through MediatR. - The handler prepares the request for the Salesforce API. - Adapter sends the request to Salesforce API Endpoint: `POST /services/data/v47.0/chatter/feed-elements`. 3. \*\*Transformation\*\*: - The payload is transformed to Salesforce's required format: ```json { "body": { "messageSegments": [ { "type": "Text", "text": "This is an example post." } ] }, "feedElementType": "FeedItem", "subjectId": "record-id" } ``` 4. \*\*Output\*\*: - Success: `201` response status code with a message confirming the post creation. - Failure: Errors from Salesforce API (e.g., invalid token) are logged, and appropriate responses (e.g., `401 Unauthorized`) are returned. 5. \*\*Observability\*\*: - Request payload and response status are logged. - Errors are flagged with alerts in Application Insights. --- ### \*\*Flow Name: Chatter-FeedItem-Put\*\* #### \*\*1. Purpose\*\*: Ensures that a Chatter post is added to a specific record in Salesforce only if a similar post does not already exist. #### \*\*2. Data Flow\*\*: 1. \*\*Trigger\*\*: - HTTP PUT request to `/chatter/put-feed-item` with payload: ```json { "recordId": "record-id", "messageSegments": [ { "type": "Text", "text": "This is an example post." } ] } ``` 2. \*\*System Interaction\*\*: - Payload is validated and passed to the MediatR `PutFeedItemCommand`. - Handler queries Salesforce for existing posts using the SalesforceAdapter and API Endpoint: `GET /services/data/v47.0/feed-item`. - If no duplicates are found, the handler prepares the new message payload and calls the `POST /services/data/v47.0/chatter/feed-elements` API to create the Chatter post. 3. \*\*Transformation\*\*: - Existing posts are retrieved and checked for similarity. - If no match is found: ```json { "body": { "messageSegments": [ { "type": "Text", "text": "This is an example post." } ] }, "feedElementType": "FeedItem", "subjectId": "record-id" } ``` 4. \*\*Output\*\*: - Success: `201 Created` status code for newly posted messages. Returns `200 OK` if the post already exists. - Failure: Errors are logged and reported. 5. \*\*Observability\*\*: - Duplicate detection is logged. - Errors and retries (if any) are captured in Application Insights. --- ### \*\*Flow Name: GenerateAccessToken\*\* #### \*\*1. Purpose\*\*: Generates and retrieves Salesforce OAuth access tokens for use in other flows. #### \*\*2. Data Flow\*\*: 1. \*\*Trigger\*\*: - This flow is triggered internally as a sub-flow whenever a token is required or expired. 2. \*\*System Interaction\*\*: - Payload for token generation is prepared: ```json { "grant\_type": "password", "client\_id": "client-id", "client\_secret": "client-secret", "username": "salesforce-username", "password": "password" } ``` - Salesforce API Endpoint: `POST /services/oauth2/token`. 3. \*\*Transformation\*\*: - The response from the Salesforce API (including access token) is parsed and securely stored: ```json { "access\_token": "generated-access-token", "issued\_at": "...", "instance\_url": "https://your-instance.salesforce.com" } ``` 4. \*\*Output\*\*: - The `access\_token` is passed to dependent handlers/functions during execution. 5. \*\*Observability\*\*: - Token generation requests and responses are logged but exclude sensitive data such as credentials. - Errors (e.g., invalid client secrets) are flagged with alerts. --- ### \*\*Generic Observability Features\*\* - \*\*Logging\*\*: Application Insights captures logs for all HTTP requests and responses, including latency, errors, and custom metrics. - \*\*Error Handling\*\*: - Default retry policy for transient errors on Salesforce interactions. - Alerts for specific error codes like `401 Unauthorized` or `500 Internal Server Error` via Azure Monitor. - \*\*Security\*\*: - Credentials and sensitive configuration values (e.g., `Salesforce client secrets`, `Access tokens`) are stored securely in Azure Key Vault. - Role-based access control is enforced for all functions and keys. --- This integration flow documentation provides a clear step-by-step explanation for the data flow, transformations, and system interactions in your AIS implementation. Let me know if you'd like diagrams or additional clarifications!

---  
  
## \*\*Integration Flow Documentation\*\*  
  
### \*\*Flow Name: Group-GetByName\*\*  
  
#### \*\*1. Purpose\*\*:  
This flow retrieves a Salesforce Group ID by querying the Salesforce `CollaborationGroup` object based on a group name provided as a parameter in the HTTP GET request.  
  
#### \*\*2. Data Flow\*\*:  
1. \*\*Trigger\*\*:  
 - HTTP trigger by invoking `/group/get-by-name` with the query parameter `groupName`.  
 - Example: `GET /group/get-by-name?groupName=Marketing`.  
  
2. \*\*System Interaction\*\*:  
 - Group name is passed to a MediatR `GetByNameCommand`.  
 - The handler executes the logic and uses the `SalesforceAdapter` to call the Salesforce API.  
 - Salesforce API Endpoint: `GET /services/data/v47.0/sobjects/CollaborationGroup`.  
  
3. \*\*Transformation\*\*:  
 - For successful queries, the Salesforce API response is mapped to a JSON object that contains only the `Id` of the group:  
 ```json  
 {  
 "id": "group-id"  
 }  
 ```  
 - If the group is not found, a `404 Not Found` response is returned:  
 ```json  
 {  
 "error": "Group not found"  
 }  
 ```  
  
4. \*\*Output\*\*:  
 - Success: Appropriate JSON response of the group ID.  
 - Failure: Returns a `404` status code for missing group names.  
  
5. \*\*Observability\*\*:  
 - Each API request is logged (both input and output).  
 - Errors are reported to Application Insights, with alerts enabled for critical issues.  
  
---  
  
### \*\*Flow Name: Chatter-FeedItem-Post\*\*  
  
#### \*\*1. Purpose\*\*:  
This flow adds a Chatter message associated with a specific Salesforce record.  
  
#### \*\*2. Data Flow\*\*:  
1. \*\*Trigger\*\*:  
 - HTTP POST request to `/chatter/post-feed-item` with the payload, which must include:  
 ```json  
 {  
 "recordId": "record-id",  
 "messageSegments": [  
 {  
 "type": "Text",  
 "text": "This is an example post."  
 }  
 ]  
 }  
 ```  
  
2. \*\*System Interaction\*\*:  
 - Payload is validated and sent to the `PostFeedItemCommand` through MediatR.  
 - The handler prepares the request for the Salesforce API.  
 - Adapter sends the request to Salesforce API Endpoint: `POST /services/data/v47.0/chatter/feed-elements`.  
  
3. \*\*Transformation\*\*:  
 - The payload is transformed to Salesforce's required format:  
 ```json  
 {  
 "body": {  
 "messageSegments": [  
 {  
 "type": "Text",  
 "text": "This is an example post."  
 }  
 ]  
 },  
 "feedElementType": "FeedItem",  
 "subjectId": "record-id"  
 }  
 ```  
  
4. \*\*Output\*\*:  
 - Success: `201` response status code with a message confirming the post creation.  
 - Failure: Errors from Salesforce API (e.g., invalid token) are logged, and appropriate responses (e.g., `401 Unauthorized`) are returned.  
  
5. \*\*Observability\*\*:  
 - Request payload and response status are logged.  
 - Errors are flagged with alerts in Application Insights.  
  
---  
  
### \*\*Flow Name: Chatter-FeedItem-Put\*\*  
  
#### \*\*1. Purpose\*\*:  
Ensures that a Chatter post is added to a specific record in Salesforce only if a similar post does not already exist.  
  
#### \*\*2. Data Flow\*\*:  
1. \*\*Trigger\*\*:  
 - HTTP PUT request to `/chatter/put-feed-item` with payload:  
 ```json  
 {  
 "recordId": "record-id",  
 "messageSegments": [  
 {  
 "type": "Text",  
 "text": "This is an example post."  
 }  
 ]  
 }  
 ```  
  
2. \*\*System Interaction\*\*:  
 - Payload is validated and passed to the MediatR `PutFeedItemCommand`.  
 - Handler queries Salesforce for existing posts using the SalesforceAdapter and API Endpoint: `GET /services/data/v47.0/feed-item`.  
 - If no duplicates are found, the handler prepares the new message payload and calls the `POST /services/data/v47.0/chatter/feed-elements` API to create the Chatter post.  
  
3. \*\*Transformation\*\*:  
 - Existing posts are retrieved and checked for similarity.  
 - If no match is found:  
 ```json  
 {  
 "body": {  
 "messageSegments": [  
 {  
 "type": "Text",  
 "text": "This is an example post."  
 }  
 ]  
 },  
 "feedElementType": "FeedItem",  
 "subjectId": "record-id"  
 }  
 ```  
  
4. \*\*Output\*\*:  
 - Success: `201 Created` status code for newly posted messages. Returns `200 OK` if the post already exists.  
 - Failure: Errors are logged and reported.  
  
5. \*\*Observability\*\*:  
 - Duplicate detection is logged.  
 - Errors and retries (if any) are captured in Application Insights.  
  
---  
  
### \*\*Flow Name: GenerateAccessToken\*\*  
  
#### \*\*1. Purpose\*\*:  
Generates and retrieves Salesforce OAuth access tokens for use in other flows.  
  
#### \*\*2. Data Flow\*\*:  
1. \*\*Trigger\*\*:  
 - This flow is triggered internally as a sub-flow whenever a token is required or expired.  
  
2. \*\*System Interaction\*\*:  
 - Payload for token generation is prepared:  
 ```json  
 {  
 "grant\_type": "password",  
 "client\_id": "client-id",  
 "client\_secret": "client-secret",  
 "username": "salesforce-username",  
 "password": "password"  
 }  
 ```  
 - Salesforce API Endpoint: `POST /services/oauth2/token`.  
  
3. \*\*Transformation\*\*:  
 - The response from the Salesforce API (including access token) is parsed and securely stored:  
 ```json  
 {  
 "access\_token": "generated-access-token",  
 "issued\_at": "...",  
 "instance\_url": "https://your-instance.salesforce.com"  
 }  
 ```  
  
4. \*\*Output\*\*:  
 - The `access\_token` is passed to dependent handlers/functions during execution.  
  
5. \*\*Observability\*\*:  
 - Token generation requests and responses are logged but exclude sensitive data such as credentials.  
 - Errors (e.g., invalid client secrets) are flagged with alerts.  
  
---  
  
### \*\*Generic Observability Features\*\*  
- \*\*Logging\*\*: Application Insights captures logs for all HTTP requests and responses, including latency, errors, and custom metrics.  
- \*\*Error Handling\*\*:  
 - Default retry policy for transient errors on Salesforce interactions.  
 - Alerts for specific error codes like `401 Unauthorized` or `500 Internal Server Error` via Azure Monitor.  
- \*\*Security\*\*:  
 - Credentials and sensitive configuration values (e.g., `Salesforce client secrets`, `Access tokens`) are stored securely in Azure Key Vault.  
 - Role-based access control is enforced for all functions and keys.  
  
---  
  
This integration flow documentation provides a clear step-by-step explanation for the data flow, transformations, and system interactions in your AIS implementation. Let me know if you'd like diagrams or additional clarifications!