Insurer CSV Upload & Processing API – Postman Testing Guide

Overview

Insurers can upload CSV/Excel policy files via the Data Pipeline Service. Files are validated, mapped to a canonical schema, massaged (dates, currency, mobile), and matched to customers using mobile, email, PAN, and DOB.

Prerequisites

- **data-pipeline-service** running on port 8082
- **customer-service** on 8081 (for customer matching)
- **policy-service** on 8085 (for policy creation)
- PostgreSQL (mypolicy_db) and MongoDB (ingestion_db), OR use `local` profile (H2 + MongoDB on localhost:27017)

Start the service (local profile – no external DBs):

```
cd data-pipeline-service mvn spring-boot:run -Dspring-boot.run.profiles=local
```

Base URL for all requests: `http://localhost:8082`

Postman Setup

1. Create Environment (Recommended)

Variable	Initial Value	Current Value
`base_url`	http://localhost:8082	http://localhost:8082
`job_id`	(leave empty)	(filled after upload)

2. Sample CSV Files

Use files from `data-pipeline-service/Datasets/`:

File	insurerId	policyType
`Health_Insurance.csv`	HEALTH_INSURER	HEALTH
`Auto_Insurance.csv`	AUTO_INSURER	MOTOR
`Life_Insurance.csv`	LIFE_INSURER	TERM_LIFE

API 1: Upload CSV/Excel

Purpose & Why We Test

Aspect	Description
What it achieves	Accepts a policy file (CSV/Excel) from an insurer, validates it, stores it on disk, and creates an ingestion job in MongoDB. The job can later be processed to map records to customers and create policies.
Why we test	To verify that file upload works end-to-end: validation (file type, size, schema), storage to `storage/ingestion`, and job creation. Without a valid upload, no processing can occur.
Input	`file` (multipart), `insurerId`, `uploadedBy`, optional `fileType`
Output	`jobId` (UUID), `status: "UPLOADED"` — use the jobId for the next step (Process).

Request

Field	Value
Method	`POST`
URL	`{{base_url}}/api/public/v1/ingestion/upload`
Headers	(None required – Postman sets Content-Type for form-data)

Body (Postman)

1. Go to **Body** tab.
2. Select **form-data** (not x-www-form-urlencoded, not raw).
3. Add these KEY–VALUE pairs:

KEY	TYPE	VALUE	REQUIRED
`file`	**File**	Click "Select Files" and choose a CSV (e.g. `Health_Insurance.csv`)	Yes
`insurerId`	**Text**	`HEALTH_INSURER`	Yes
`uploadedBy`	**Text**	`postman-test`	Yes
`fileType`	**Text**	`normal`	No (default: normal)

InsurerId Values

insurerId	Use With	Sample File
`HEALTH_INSURER`	Health insurance CSVs	`Health_Insurance.csv`
`AUTO_INSURER`	Auto/Motor insurance CSVs	`Auto_Insurance.csv`
`LIFE_INSURER`	Life insurance CSVs	`Life_Insurance.csv`

Successful Response (201 Created)

```
{ "jobId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890", "status": "UPLOADED" }
```

Important: Copy the `jobId` value for the next request.

Error Responses

Status	Example Body
400 Bad Request	`{"timestamp":"...","status":400,"error":"Bad Request","message":"File is empty or missing"}`
400 Bad Request	`{"timestamp":"...","status":400,"error":"Bad Request","message":"Invalid file type. Allowed: .xls, .xlsx, .csv"}`
400 Bad Request	`{"timestamp":"...","status":400,"error":"Bad Request","message":"File size exceeds maximum allowed: 50MB"}`

Postman Screenshot Checklist

- **Body** → **form-data**
- `file` row: type = **File**, value = filename after selection
- `insurerId` row: type = **Text**, value = `HEALTH_INSURER` (or `AUTO_INSURER` / `LIFE_INSURER`)
- `uploadedBy` row: type = **Text**, value = any string (e.g. `postman-test`)

API 2: Trigger Processing

Purpose & Why We Test

Aspect	Description

What it achieves	Starts the processing pipeline for an uploaded file: parses CSV/Excel, applies metadata-driven field mappings, massages data (dates, currency, mobile, status), matches each row to a customer (mobile → PAN → email), and creates policies via the policy-service.
Why we test	To confirm the full transformation and matching flow runs. This is where data is mapped, normalized, and linked to the Customer Master and Policy Master.
Input	`jobId` (path), optional `policyType` (query) — must match a job in UPLOADED state.
Output	`jobId`, `message: "Processing started"`, `status: "PROCESSING"` — processing runs asynchronously; use API 3 to check completion.

Request

Field	Value
Method	`POST`
URL	`{{base_url}}/api/public/v1/ingestion/process/{{job_id}}`
Params	Optional: `policyType`

URL with Path Variable

- Path: `/api/public/v1/ingestion/process/{jobId}`
- Replace `{jobId}` with the `jobId` from the upload response (e.g. `a1b2c3d4-e5f6-7890-abcd-ef1234567890`)

Full URL example:
`http://localhost:8082/api/public/v1/ingestion/process/a1b2c3d4-e5f6-7890-abcd-ef1234567890`

Query Params (Optional)

KEY	VALUE	Description
`policyType`	`HEALTH`	For Health_Insurance.csv
`policyType`	`MOTOR`	For Auto_Insurance.csv
`policyType`	`TERM_LIFE`	For Life_Insurance.csv

With query param:
`http://localhost:8082/api/public/v1/ingestion/process/a1b2c3d4-e5f6-7890-abcd-ef1234567890?policyType=HEALTH`

Body

- **None** – leave Body empty.

Successful Response (202 Accepted)

```
{ "jobId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890", "message": "Processing started", "status":
"PROCESSING" }
```

Error Responses

Status	Condition	Example Body
400 Bad Request	Job not in UPLOADED state	`{"error":"Job must be in UPLOADED state","currentStatus":"PROCESSING"}`
404 Not Found	Invalid jobId	Job not found error
500 Internal Server Error	Processing failure	`{"error":"Processing failed","detail":"..."}`

API 3: Get Job Status

Purpose & Why We Test

Aspect	Description
What it achieves	Returns the current state of an ingestion job: status (UPLOADED, PROCESSING, COMPLETED, FAILED), record counts (total vs processed), file path, insurer, and timestamps.
Why we test	To verify processing completed successfully and to see how many records were processed. After triggering processing (API 2), this confirms whether the job finished or failed.
Input	`jobId` (path)
Output	`jobId`, `status`, `processedRecords`, `totalRecords`, `filePath`, `insurerId`, `fileType`, `createdAt`, `updatedAt`, and `failureReason` (if FAILED).

Request

Field	Value
Method	`GET`
URL	`{{base_url}}/api/public/v1/ingestion/status/{{job_id}}`

Full URL example:
`http://localhost:8082/api/public/v1/ingestion/status/a1b2c3d4-e5f6-7890-abcd-ef1234567890`

Body

- **None** – GET requests have no body.

Successful Response (200 OK)

```
{ "jobId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890", "status": "COMPLETED", "processedRecords":
100, "totalRecords": 100, "filePath":
"C:/path/to/storage/ingestion/a1b2c3d4-e5f6-7890-abcd-ef1234567890.csv", "insurerId":
"HEALTH_INSURER", "fileType": "normal", "createdAt": "2026-02-26T12:00:00", "updatedAt":
"2026-02-26T12:01:30" }
```

Status Values

status	Description
`UPLOADED`	File uploaded, ready for processing
`PROCESSING`	Processing in progress
`COMPLETED`	Successfully completed
`FAILED`	Processing failed (check `failureReason` if present)

Error Response (404)

```
{ "timestamp": "...", "status": 404, "error": "Not Found", "message": "Job not found:
invalid-job-id" }
```

End-to-End Postman Test Flow

Step 1: Upload

1. Create request: **POST** `http://localhost:8082/api/public/v1/ingestion/upload`
2. **Body** → **form-data**:
3. `file` (File): select `Datasets/Health_Insurance.csv`
4. `insurerId` (Text): `HEALTH_INSURER`
5. `uploadedBy` (Text): `postman-test`
6. Click **Send**.
7. Copy `jobId` from the response.

Step 2: Process

1. Create request: **POST** `http://localhost:8082/api/public/v1/ingestion/process/{paste-jobId-here}`
2. Add query param: `policyType` = `HEALTH` (optional).
3. No body.
4. Click **Send**.

Step 3: Check Status

1. Create request: **GET** `http://localhost:8082/api/public/v1/ingestion/status/{paste-jobId-here}`
2. No body, no params.
3. Click **Send**.
4. Verify `status` is `COMPLETED` and `processedRecords` equals `totalRecords`.

CSV Column Requirements (Reference)

Health_Insurance.csv (insurerId: HEALTH_INSURER)

Column Name	Required	Example
Policy Number	Yes	HEPOL100000
Customer Name	Yes	Prakash Sharma
DOB	Yes	19720129
PAN	Yes	ONREL7123Y
Mobile	Yes	917456775329
Email	Yes	prakash.sharma0@gmail.com
Insurer	Yes	Bajaj Allianz
Coverage Amount	Yes	300000
Annual Premium	Yes	22630
Policy Start Date	Yes	20240320
Policy End Date	Yes	20320220
City	Yes	PUNE
Gender	No	M
Plan Name	No	Standard Plan
Policy Type	No	Family Floater

Auto_Insurance.csv (insurerId: AUTO_INSURER)

Column Name	Required	Example
PolicyNumber	Yes	AUPOL100000

CustomerName	Yes	Nitin Sharma
DOB	Yes	19881220
PAN	Yes	IFAZG4101U
Mobile	Yes	919839831702
Email	Yes	nitin.sharma0@gmail.com
Insurer	Yes	HDFC Life
IDV	Yes	742536
AnnualPremium	Yes	37512
PolicyStartDate	Yes	20240717
PolicyEndDate	Yes	20310606
City	Yes	PUNE
VehicleType	No	Bike
VehicleRegNo	No	MH12AB1471

Life_Insurance.csv (insurerId: LIFE_INSURER)

Column Name	Required	Example
PolicyNum	Yes	LIPOL1000
CustomerName	Yes	Rahul Desai
DOB	Yes	19661212
PAN	Yes	MAUKT368
Mobile	Yes	91724516
Email	Yes	rahul.desai
Insurer	Yes	SBI Life
SumAssured	Yes	500000
AnnualPrem	Yes	39221
PolicyStart	Yes	20220824
PolicyEnd	Yes	20260527
City	Yes	HYDERABAD

Gender	No	M
PlanName	No	Standard P
PolicyTerm	No	30
NomineeName	No	Spouse
NomineeRelation	No	Spouse

Troubleshooting

Issue	Cause	Fix
"File is empty or missing"	File not selected or wrong key name	Use key `file` (lowercase) and select a file
"Invalid file type"	Wrong extension	Use .csv, .xls, or .xlsx only
"Job not found"	Invalid or expired jobId	Run Upload again and use new jobId
"Job must be in UPLOADED state"	Processing already started	Use a new jobId from a fresh upload
Connection refused	Service not running	Start data-pipeline-service on port 8082
Feign errors on process	customer/policy services down	Start customer-service (8081) and policy-service (8085)

Customer Matching & Data Massaging

- **Matching order:** Mobile → PAN → Email, with name/DOB verification when available.
- **Dates:** Normalized to ISO `yyyy-MM-dd` (accepts YYYYMMDD, dd/MM/yyyy).
- **Currency:** Stripped of symbols and commas.
- **Mobile:** Digits only; 91 prefix added for 10-digit Indian numbers.
- **Status:** Mapped to ACTIVE, LAPSED, CANCELLED, PENDING.