**Modular file ingestion and rules processing system Evidences**

**Source csv**

**claim\_id,patient\_id,diagnosis\_code,claim\_amount,service\_date,provider\_id,status**  
C101,P201,**I21**,1200.00,2023-02-15,PRV010,Pending  
C102,P202,**C34**,**30.00**,2023-01-10,PRV011,Pending  
C103,P203,J45**,45.00,2022-12-25**,PRV012,Pending  
C104,P204,K21,500.00,2023-03-01,PRV013,Pending  
C105,P205,I10,60.00,**2022-11-30**,PRV014,Pending  
C106,P206,**C34**,200.00,2023-01-05,PRV015,Pending  
C107,P207,N18,**40.00**,**2022-10-20**,PRV016,Pending

**Applying Remote rules**

****

**Output**:

Transformed JSON after applying remote rules:



**Applying local rules (When remote rules is not available)**

****

**Sample csv**

**claim\_id,patient\_id,diagnosis\_code,claim\_amount,service\_date,provider\_id,status**  
C001,P101,J20.9,150.00,2023-01-15,PRV001,Pending  
C002,P102,**I21**,500.00,2023-01-16,PRV002,Pending  
C003,P103,N18.9,1200.50,2023-01-17,PRV003,Pending  
C004,P104,J20.9,75.20,2023-01-18,PRV001,Pending  
C005,P105,K21.9,45.00,2023-01-19,PRV004,Pending  
C006,P106,I10,250.00,2023-01-20,PRV002,Pending  
C007,P107,N18.9,900.00,2023-01-21,PRV003,Pending  
C008,P108,J20.9,200.00,2023-01-22,PRV001,Pending

**Output**:

Transformed JSON after applying local rules:

****

**Invalid claims**

*Incorrect Data*

**Source csv:**

**claim\_id,patient\_id,diagnosis\_code,claim\_amount,service\_date,provider\_id,status**  
C101,P201,I21,**1200A**,2023-02-15,PRV010,Pending – Incorrect claim\_amount  
C102,P202,C34,30.00,2023-01-10,PRV011,Pending  
,P203,J45,45.00,2022-12-25,PRV012,Pending - Missing claim id  
C104,P204,K21,500.00,**2023-03**,PRV013,Pending – Invalid service date  
C105,P205,I10,60.00,2022-11-30,PRV014,Pending  
C106,P206,C34,200.00,2023-01-05,PRV015,Pending  
C107,P207,N18,40.00,2022-10-20,PRV016,Pending

**Output:**

****

**Logs written to a file (logs.json)**

****

**Basic unit tests output**

****

**Alternate Approach using AWS services**

In this scenario, I have coded a Flask-based standalone app to demonstrate the file ingestion and rules processing system. However, we typically use AWS services in our architecture. Here’s the common AWS-based approach in steps:

1. **API Gateway**

* Expose a REST API endpoint /upload to accept file uploads directly.
* Handle request validation, authentication, and throttling.

1. **Lambda Functions**

* **File Processing Lambda**: Triggered by API Gateway on file upload; parses CSV and converts it to JSON.
* **Rule Engine Lambda**: Applies configurable rules to the JSON data, loading rules from environment variables or a remote rules service.

1. **DynamoDB (or another database)**

* Persist logs and processing results for audit and troubleshooting.

1. **CloudWatch Logs**

* Centralized logging for Lambda executions and error tracking.

1. **Optional Remote Rules Service**

* A dedicated Lambda or microservice that provides dynamic rule sets via API calls.

1. **Testing**

* Use Python unit tests with mocking libraries (pytest, moto) to test Lambda functions locally.