Here are the structured use cases for system involving credit card transactions and claims processing:

**Happy Path Use Cases**

**1. Transaction Processing**

* **Objective**: Process a user's credit card transaction successfully.
* **Steps**:
  1. User submits a credit card transaction.
  2. The system validates the expiration date of credit card and approves it.
  3. Validate the credit card number format. 1111-2222-3333-4444
  4. Transaction details are stored in the database.
* **Outcome**: Transaction is successfully completed and recorded.

Transaction Payload Request:

{

"User Name": "asdf",

“Card\_ID”: “2345”

"amount": "120.00",

"item\_name": "apple",

"vendor": "vendor\_name",

"credit\_card": "credit\_card",

"exp\_date": "date",

"CVC": "123",

}

Transaction Payload Successful Response:

{

“transaction\_id”: “uuid”,

“timestamp”: “timestamp”,

“message”: “transaction successful”,

“result”: “approved”,

“status\_code”: “200”

}

**2. Claim Processing**

* **Objective**: Handle a claim made by a user based on a previous transaction.
* **Steps**:
  1. User submits a claim related to a specific transaction.
  2. The system verifies the claim details and processes it successfully.
  3. Claim details are stored in the database.
* **Outcome**: Claim is successfully processed and recorded.

Claims Payload Request

{

“transaction\_id”: “uuid”,

“credit card”: "credit\_card",

“amount”: “amount”,

“vendor”: {“vendor\_name”,

“exp date”: “date”,

“claim\_reason”: “reason for claim”

}

}

Claims Payload Response

{

“transaction\_id”: “uuid”,

“claim\_number”: “claim\_number”,

“result”: “pass”,

“timestamp”: “timestamp”,

“status\_code”: “200”

“message”: “claims successful”,

}

**Unhappy Path Use Cases**

**1. Transaction Rejection Due to Expired Credit Card**

* **Objective**: Handle the rejection of a transaction when the user's credit card is expired.
* **Steps**:
  1. User submits a credit card transaction.
  2. The system checks the credit card details and identifies that the card is expired.
  3. Transaction is rejected.
  4. Details of the rejected transaction are stored in the database.
  5. User is notified of the rejection due to the expired credit card.
* **Outcome**: Transaction is rejected, and user is informed of the reason.

Transaction Payload Request Expired CC:

{

"User\_Name": "Oya Mainib",

"amount": "120.00",

"item\_name": "Tablet",

"vendor": "Techs-R-Us",

"credit\_card": "1111-2222-3333-4444",

"exp\_date": "05/22",

"CVC": "123",

}

Transaction Payload Unsuccessful Response:

{

“transaction\_id”: “uuid”,

“timestamp”: “timestamp”,

“message”: “transaction unsuccessful, credit card expired”,

“result”: “declined”,

“status\_code”: “200”

}

**2. Claim Rejection Due to Various Rule Failures**

**def lambda\_handler(event, context):**

**validation\_status = "valid"**

**message = "All checks passed."**

**# Check if transaction ID is valid UUID and exists**

**if not is\_valid\_uuid(event['transaction\_id']):**

**validation\_status = "invalid"**

**message = "Invalid transaction ID."**

**else:**

**transaction = transaction\_table.get\_item(Key={'transaction\_id': event['transaction\_id']})**

**if 'Item' not in transaction:**

**validation\_status = "invalid"**

**message = "Transaction ID does not exist."**

**else:**

**transaction = transaction['Item']**

**# Additional validations**

**if transaction['amount'] != event['ClaimAmount']:**

**validation\_status = "invalid"**

**message = "Claim amount does not match transaction amount."**

**# Add more validations as needed**

**validation\_record = {**

**'transaction\_id': event['transaction\_id'],**

**'validation\_status': validation\_status,**

**'message': message,**

**'timestamp': datetime.utcnow().isoformat()**

**}**

**validation\_table.put\_item(Item=validation\_record)**

**return {**

**'statusCode': 200,**

**'body': json.dumps({**

**'validation\_status': validation\_status,**

**'message': message**

**})**

**}**

**def is\_valid\_uuid(uuid\_str):**

**try:**

**uuid\_obj = uuid.UUID(uuid\_str, version=4)**

**except ValueError:**

**return False**

**return str(uuid\_obj) == uuid\_str**

**Rule 1 Failure**

* **Objective**: Handle claim rejection based on various validation errors.
* **Steps**:
  1. User submits a claim.
  2. The system checks the claim and finds issues (invalid transaction ID, incorrect amount, etc.).
  3. Claim is rejected.
  4. Rejection details are stored in DynamoDB.
  5. User is notified of the specific reasons for rejection.
* **Outcome**: Claim is rejected and detailed reasons are provided to the user.

**Rule 2 Failure**

* **Objective**: Handle claim rejection when one or more specified rules fail.
* **Steps**:
  1. User submits a claim.
  2. The system evaluates the claim against specified rules.
  3. Claim fails to meet one or more rules.
  4. Claim is rejected.
  5. Rejection details are stored in DynamoDB.
  6. User receives a response explaining which rules were violated.
* **Outcome**: Claim is rejected with explanations provided for rule failures.

**Rule 3 Failure**

* **Objective**: Reject a claim due to exceeding the permissible number of claims within a set period.
* **Steps**:
  1. User submits a claim.
  2. The system detects that the user has exceeded the number of allowable claims in 30 days.
  3. Claim is rejected.
  4. Rejection details are stored in the database.
  5. User is notified of the rejection due to exceeding the claim limit.
* **Outcome**: Claim is rejected with a notification regarding the limitation.

**Rule 4 Failure**

* **Objective**: Reject a claim due to excessive claims filed against a particular vendor.
* **Steps**:
  1. User submits a claim related to a transaction with a vendor.
  2. The system finds that the vendor has had over 10 claims in the past 30 days.
  3. Claim is rejected.
  4. Rejection details are stored in DynamoDB.
  5. User receives a response detailing the rule failure.
* **Outcome**: Claim is rejected with a specific explanation regarding vendor claim frequency.

{

“transaction\_id”: “uuid”,

“credit card”: "credit\_card",

“amount”: “amount”,

“vendor”: {“vendor\_name”,

“exp date”: “date”,

“claim\_reason”: “reason for claim”

}

}

These use cases provide a comprehensive framework for handling both successful and unsuccessful transactions and claims, ensuring clear procedures are in place for various scenarios.