Table of Contents

* Technologies in Scope
* PoC Description
  + Payment Processing System (PPS)
  + Broker System (BS)
  + Fraud Check System (FCS)
* Payment Payload
* Remarks
* Structure of the Demo

Technologies in Scope

* Java
* Design Patterns
* Apache Camel (or other alternative open source integration framework)
* XML and JSON creation, validation and parsing
* REST APIs
* JMS API
* ActiveMQ (or similar open source JMS provider)
* (nice to have) OpenShift (or other alternative container/cloud platform)

PoC Description

The PoC is based on a solution that needs to perform fraud check before honoring the payment for a bank.

The systems that need to be mocked in the PoC are:

1. Payment Processing System (PPS)
2. Fraud Check System (FCS)
3. Broker System (BS) insulating the PPS and FCS

You need to develop 2 solutions:

1. for solution 1 all interfaces between the PPS and BS is based on messaging and JSON.
2. for solution 2 all interfaces between the PPS and BS is based on REST APIs and JSON.
3. for both solutions all interfaces between the BS and FCS is based on messaging and XML.
4. for both solutions the preferable communication between components inside each system should be messaging.

Payment Processing System (PPS)

1. receives a payment (in JSON) for processing.
2. performs basic validation on a payment (e.g. valid ISO country code, valid ISO currency code).
3. invokes a broker system in the BS for a fraud check on the payment.
4. processes the payment after the fraud check based on a approval or rejection response from the BS.

Broker System (BS)

1. receives a fraud check request for a payment (in JSON) from the PPS and converts to XML.
2. sends the payment (in XML) to the FCS for executing a fraud check.
3. receives the fraud check result (in XML) from the FCS.
4. converts the fraud check result to JSON and sends it to the PPS.

Fraud Check System (FCS)

1. receives a fraud check request for a payment (in XML) from the BS.
2. checks the payer and payee details (name, country, bank) and the payment instruction.
3. approves or rejects the payment based on the checks.
4. sends the results of the fraud check (e.g. approval, rejection) to the BS.

The system should perform the following checks based on blacklists:

|  |  |
| --- | --- |
| Field | Blacklist |
| payer and payee name | “Mark Imaginary”, “Govind Real”, “Shakil Maybe”, “Chang Imagine” |
| payer and payee country | CUB, IRQ, IRN, PRK, SDN, SYR |
| payer and payee bank | “BANK OF KUNLUN”, “KARAMAY CITY COMMERCIAL BANK” |
| payment instruction | “Artillery Procurement”, “Lethal Chemicals payment” |

If any of the matches is found then reject the payment with message “Suspicious payment”, otherwise approve the payment with message “Nothing found, all okay”.

Payment Payload

|  |  |  |
| --- | --- | --- |
| Field Name | Field Category | Field Value |
| Transaction ID | Mandatory | Transaction ID in UUID format (<https://en.wikipedia.org/wiki/Universally_unique_identifier>)  Must be unique across the system and E2E traceable |
| Payer Name | Mandatory | Payer's first and last name, for example: "Munster Muller" |
| Payer Bank | Mandatory | Name of the payer's bank, for example: "Bank of America" |
| Payer Country Code | Mandatory | ISO alpha-3 country code, for example: DEU, GBR,USA |
| Payer Account | Mandatory | Payer's account number |
| Payee Name | Mandatory | Payee's first and last name, for example: "Munster Muller" |
| Payee Bank | Mandatory | Name of the payee's bank, for example: "BNP Paribas" |
| Payee Country Code | Mandatory | ISO alpha-3 country code, for example: DEU, GBR, USA |
| Payee Account | Mandatory | Payee's account  number |
| Payment Instruction | Optional | Free text, for example: "Loan Repayment", "Tax Reimbursements", etc |
| Execution Date | Mandatory | ISO 8601 date format YYYY-MM-DD, for example: 2020-02-21 |
| Amount | Mandatory | 2 decimal places must be supplied, for example: 17.45 |
| Currency | Mandatory | ISO 4217 currency code, for example: EUR, GBP |
| Creation Timestamp | Mandatory | ISO 8601 UTC timestamp format YYYY-MM-DDThh:mm:ssZ, for example: 2004-02-21T17:00:00Z |

Remarks

1. Define your own XML and JSON formats for the solutions.
2. Define your own payload for the fraud check result.
3. Use Apache AMQ (or any other open source) JMS provider. Please develop the solution using Java and Apache Camel (or other alternative open source integration framework).
4. Deploy on (nice to have) OpenShift (or other alternative container/cloud platform) with tools of your choice that you want to showcase to log or monitor the flows.
5. Define the REST APIs based on REST API best practices (e.g. CRUD).
6. Proper Audit logs need to be maintained in all components of the solutions.
7. Re-use as many as possible components and source code for the solution 1 and solution 2.

Structure of the Demo

1. Be able present the PoC (max 90 minutes).
2. The source code should also be shared in the demo. Questions regarding the source code will be raised.
3. A UML component diagram needs to be created and the solution should be explained with use of the diagram (the solution and design should be well founded).
4. The solutions must be live executed and explained (no powerpoint demo).