# Introduction

The *multi-app* application is a brief demonstration on how to detect suspect transactions and report them.

##### **Design**

The application is a collection of 3 separate components, all of them working in tandem. All the 3 components are independent spring boot applications communicating among themselves using REST APIs or MQs.

##### **Components**

1. **fraud-detection:**

This component is responsible for reading the batch files, retrieving the trade-transaction details, process them and check for suspect behavior and alert the regulators if any suspect behavior is found. This component is made up of the below sub-components –

* **PollFolder.java**: This class is responsible for reading all available batch files from the input folder. It retrieves all trade-transactions from the files, collects them and forwards the records for processing. At the end of processing, it deletes the handled batch files.
* **ProcessTransactions.java**: This class analyses the trade-transaction records. It checks the local cache to determine if the trade is suspicious or not.
* **TransactionsCache.java**: This is a TTL-expiry based local cache created using ConcurentHashmap. It stores all trade-transaction records for a specified TTL based on key created from a combination of TraderID and StockID. An internal thread running at specified intervals cleans stale records which have outlived TTL.
* **PrepareReport.java**: This class is responsible for storing the suspicious Trader details in the database. It also prepares the data to be sent to regulatory authorities for suspicious trades.
* **SendToRegulator.java**: This class is responsible for setting up the REST calls to the regulatory authorities. This class also forks out the external client call as an asynchronous process, so as to not block the actual transaction record processing.
* **RegulatorClient.java**: This class is responsible for making the actual REST calls to the regulatory authorities. It uses Spring Retryable/Recover functionality to retry the REST API calls multiple times in case of failures.
* **H2 database**: This is used to store the data for all suspicious traders.

1. **regulatory-clients**:

This component exposes 2 REST APIs, each representing a single regulatory authority. The received REST messages are in turn forwarded to ActiveMQ queues. This component is made up of the below sub-components –

* **RegulatoryController.java**: This is the Controller class which is exposing the REST endpoints. It is also responsible for forwarding the received request message to ActiveMQ queues.

1. **jms-service**:

This component is responsible for listening to the 2 ActiveMQ queues – one for each regulatory authority. Any received message is printed to the logs in Json format. This component is made up of the below sub-components –

* **Receiver.java**: This is the main part of this component. It is responsible for both listening on the ActiveMQ queues and then printing the received messages on the logs.

##### **Application Flow**

The *input folder* contains one or more batch files, each comprising of finite number of trade-transaction records. A single batch file is made up of multiple lines where each line represents a single transaction record. Each record consists of fields following the below format –   
First\_name;Last\_Name;Nationality;Country\_of\_Residence;date\_of\_birth;unique\_Trader\_id;Amount;currency;unique\_Stock\_ID;Buy\_or\_Sell

The *fraud-detection* service polls the input folder at regular intervals. If one or more file is found existing, these are read and transaction records are fetched from all existing files. These records are then checked against a local cache to compare with older trade-transactions. If upon analyzing the records, a trader is found to have handled a specific stock more than 5 times in the last 10 minutes, both the trader and the transaction are flagged.

The local cache stores old records for a specific duration and expires the records on the expiry of the duration. This helps in current trade details with older trades, thus helping verify if any trade is suspicious or not.

Details about the flagged suspicious trader is stored to database for future reference. The flagged suspicious transaction details are remodeled and forwarded to REST API services exposed by the *regulatory-client* service.

The data send to the external REST services follow the below format -

{

"traderDetails": {

"traderId": "",

"firstName": "",

"lastName": "",

"nationality": "",

"countrOfResidence": "",

"dateOfBirth": ""

},

"detectionDetails": {

"stockId": "",

"detectionTime": ""

}

}

The *regulatory-client* service receives the incoming request and forwards it as-is to externally set ActiveMQ queue. The messages are transmitted to 2 queues - *first\_queue* and *second\_queue*.

The *jms-service* application is responsible for listening on the ActiveMQ queues. On receiving a message, it logs the JSON message to a log file.