Version 1.0

1. Introduction
2. Fraud Detection System as a Java Application Service
3. Goals of the RFDS Service

Fault-Tolerance, Scalable Stateless Service; response time < 10-40 msec

1. Technical Stack
2. Code Style and Best Practices, Unit Test Cases
3. Code Review
4. Details of the Application Components, Repositories, Property/Config/ build Files
5. Description of Each Component , resources Needed , its Implementation, Unit Test Cases, Messaging, Integration, some project-specific links
6. JIRA Tasks for the RFDS Project; Use Cases, Stories and Code Coverage -JaCoCo
7. DEV, DevOps, PROD, DEPLOYMENT, Cloud Foundry Integration
8. Unit Test Cases, e2e
9. Incoming Payment Transactions XML Request: xml Document Instance, xsd Schema  
   Message Types: CT, RFP, RDF
10. XML Response : (“No-Hit”, xsd schema) back to PE engine: Message Types: CT, RFP, RDF
11. XML Response : (“Hit”, score, xsd Response schema) Message Types: CT, RFP, RDF
12. Crafting the XML Request, Response XSD Schemas for each of the (CT, RFP, RDF) message Types
13. Hazelcast distributed Cache Implementation- Scoring Algorithm Implementation
14. MQ-RCV, SEND Configuration for in-flow of PE Transaction Data into RFDS Service
15. Metrics for Evaluation of RFDS system
16. Admin Tools to be developed for Project Files and Dir Setup.  
    Mapping blacklist Lie Items into Hazelcast CHM data  
    Guaranteeing the updated PE Transaction data form th GUI Admin Supervisor is valid  
    before routing them thru RFDS system again. (Updater can make mistakes on “HIT” records; for example, if a $amount got changed by mistake, it will not be trapped in the next RFDS cycle)
17. Swagger and RFDS Open API Interface / Endpoint
18. Acquiring Test Data   
    a) PE Transaction Data for three different Message Types: CT, RFP, RDF  
     How does one know which message type it sent from the PE engine at any time?
19. Gobi Team Information
20. Introduction

Real-Time Payment Processing (RTP) is a large enterprise level system that handles payment from customers and businesses and make settlements at the TCH Clearing House in real-time. The customers and businesses can be large, medium and small businesses.

Any Payment transaction (PT) should be checked for Fraud against a Blacklist of values of selected Meta Data. If the incoming PE transaction matches against any field from the Black List filter, then the payment data is tainted and must be stopped and verified manually against fraud.

**RFDS Technical Stack:**

Development Tools:

JDK 1.8\_172   
Eclipse IDE Neon

Spring-Boot (with actuator), app-engine (bxp3 deployment),   
SMS (db)  
MQ Configuration),    
Hikari CP,    
spring JPA (Please note that only few DAO’s implemented using spring data), pagination.   
JPA Repository, Hibernate, MySql

MQ/Database Config files – SMS - <http://secrets-sms-api.bnymellon.net>

Source Code Repository - <https://git.bnymellon.net/rpx-00>

Jenkins Job Folder - <https://bxp-ci.bnymellon.net/99/job/rpx-development/>

To design any diagram, Please use glify on confluence

Spring boot Program: Liquidity Service Application as an example:

<https://git.bnymellon.net/rpx-00/liquidity-service/tree/feature/CSPTSPDD-5006-fix-sql-injection>

**Coding Style and Best Practices:**

1. Jenkins Job Folder - <https://bxp-ci.bnymellon.net/99/job/rpx-development/>
2. To design any diagram, Please use glify on confluence

* Leverage spring library for injecting resource files – Use Latest library version -                 <http://nexus.bnymellon.net/nexus/>
* No Hard Coded SQL in Java. Use Spring JPA Repository
* Proper Code alignment for readability
* Avoid returning null from a method, instead throw custom exception (Account Not Found Exception)
* Try to leverage Java design principles (Tight Cohesion, Loose coupling) and GOF design patterns wherever applicable.
* Not to re-invent the wheel, if the feature is available in any apache frame-work or other lib, will use that. E.g. StringUtils Empty Check, Joda library for date time conversion, File I/O. 10 lines of code can be written in 1 or 2 lines
* Write Business Logic (E.g. Fraud Score Calculation Logic) in Business Layer (Business Object) not in DAO layer (Only CRUD Operation) or Controller (Only Input Validation or O/P HTTP response object creation) or Service Layer (Only to orchestrate the information/control flow, API invocation to fetch information)
* Don’t create class with static methods/variables to store global constants or to perform file i/o, database read. Try to create enum class. Class & variable names are important which are self-documentation to understand the application.
* Never swallow the exception
* Start Writing Junit Class files to Test locally. Please don’t use System.out call.

**Code Review:**

Most of the best practices can’t be evaluated in Sonar / Vera code or other analysis tool. So myself, Mayank and Pradeep would be reviewing the application code (from GIT) check-in by developers. Sriram would be reviewing API standards on all RESTful applications.

**RFDS Scoring Algorithm Design and Implementation:**

The inflow of the PE Transaction data will be immutable (means all Java String data types). This will make sur no accidental updates of the input and rejected PE Transactions for manual processing.

Hazelcast (HC) based HM Implementation:

**Features:**

Scalable, fault-tolerant against node failures

**HC Life cycle:**

HC Startup Procedure:

HC Shutdown:

Design Pattern used;  
 Command Query Segreation Responsibiliy Pattern(CQRS)

Comand side of the CQRS patern creates, updaes and deletes CRUD operation on he db, Hazelcast cahce tables. The Query portion of the CQRS object handles queries by executing them against one or more materialized views that are kept up to date by subscribing to the stream of events emitted when data changes in the db due to GUI user updates on the BL data.

Hazelcast Method:  
 Search using Predicate .

“Scoring” Algorithm Implementation Details:

Create Persistent Data and Periodic Modifications(Rule Based):

The following items are variable. They may be hanged, removed, reordered, their individual scores can be changed for the users to run the Fraud Detention system at any time. Any data items that are changed by the GUI/users will persist in the db. In the same transaction, these values will update the Hazelcast cache tables and HMs.

PE Fields and their individual scores assigned to them; PE fields can be added or droped.

PE Transaction Fields Score Table in db

PE Transaction Data that persist in db

One HashMap (HM) for each BlackList (BL) Field Data:

15 HM tables, each table sporting that (key = BL field value, value=1) : Table Name: “Field-value” Table

Diagram where each Meta data field values are stored as (Key, value = 1) in HM data structures.  
(Need to be done using VISIO)

If there are 15 field data types for comparison sorted by highest score first, we have 15 HM tables

We can use ‘newCachedThreadPool” for multithreaded processing and use “CountDownLatch” synchronizer to collect the result of all “callable()” results for final processing.

**Final Result Expected:**

We are comparing the values of relevant fields of incoming PE Transaction data against a set of Blacklisted field data

“NO-HIT”, PT Transaction Data: XML Response Document; xsd scheme  
“HIT”, list of matched field data from BlackList(BL) :: XML Document:: xsd schema

**Gobi Team Information:**

Business Requirements:

Technical Design/Architecture: 07/24:

**Gobi :**

Putting together the “RFDS” Design Document:

**Eric Zhang:**

PE Mesg Type: CT:: Getting XML Request Document, its Request XSD Schema Document from ANAND  
Preparing our XML Response XSD Schema

Gt from Anand:   
CT Request XML Document:: .xml file  
CT Request XSD Schema File: .xsd file

**Deliverables: 07/24/2018:**

**Mesgtype: CT::**XML Response XSD Schema Document File:: .xml file  
XML Response XML Document – an instance of the above schema:: .xsd file

**Surjan:**

Access o all the tools/utilities needed for this project  
Download the JDK 1.8, Eclipse ,Spring Boot download

**Access Rights:**

**Jenkins Access:**

Please approve SRQST0000261761   SCCB request.

SRQST0000261765 -Jenkins access  
**JIRA Access**:: Work with offshore team JIRA raised for APP-Engine

GIT Access:

Extended Access:

**Tools and our Access Rights:**

**RFDS Team Size:**  
**To:** Joshi, Rahul Hemant; Khanna, Mayank; Samarasam, Ramesh; Byram, Pradeep; Hanumantha Reddy, Police; Hegde, Bhagyashree; Gupta, RohitKumar; Kumar, Vivek A; Hosamani, Veeresh; Singh, Anand; Lakshmanan Srinivasan, Gobichettipalayam; Konga, Srujan Kumar; Zhang, Xianfeng