Version 1.1

1. Introduction
2. Fraud Detection System as a Java Application Service
3. Goals of the RFDS Service
4. Technical Stack
5. Code Style and Best Practices, Unit Test Cases
6. Code Review
7. Building RFDS Project around Spring Boot-MVC System
8. Hazelcast distributed Cache for “Scoring” Algorithm Implementation
9. MQ-RCV, MQ-SEND Configuration for in-flow of PE Transaction Data into RFDS Service
10. Details of the Application Components, Repositories, Property/Config/ build Files
11. Components, SubTasks and their Allocation to Group Members
12. Description of Each Component , resources Needed , its Implementation, Unit Test Cases, Messaging, Integration, some project-specific links
13. JIRA Tasks for the RFDS Project; Use Cases, Stories and Code Coverage -JaCoCo
14. DEV, DevOps, PROD, DEPLOYMENT, Cloud Foundry Integration
15. Unit Test Cases, e2e
16. Incoming Payment Transactions XML Request: xml Document Instance, xsd Schema  
    Message Types: CT, RFP, RDF
17. XML Response : (“No-Hit”, xsd schema) back to PE engine: Message Types: CT, RFP, RDF
18. XML Response : (“Hit”, score, xsd Response schema) Message Types: CT, RFP, RDF
19. Crafting the XML Request, Response XSD Schemas for each of the (CT, RFP, RDF) message Types
20. Metrics for Evaluation of RFDS system
21. Admin Tools to be developed for Project Files and Dir Setup.  
    AngularJS GUI Front-End Mapping blacklist Lie Items into Hazelcast HM data structure  
    Guaranteeing the updated PE Transaction data format is valid by GUI Admin Supervisor   
    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)
22. Swagger Use and RFDS Open API Interface / Endpoint
23. Acquiring Test Data for Each Component  
    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?
24. Gobi Team Information(1+3)
25. Offshore Team Information – WIPRO, Joshi Team (1+5)

25. Access to all Resources required to execute this Project

1. 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 fields matche against corresponding field from the Black List filter, then the payment data is tainted and must be stopped and verified manually against fraud.

1. RFDS as a Service (PaaS):  
   MQ – Inbound service; MQ-Outbound service  
   Pinging a public API URL or check of the Health of the Service

**3.Goals of the RFDS Service**

**4.RFDS Technical Stack:**

Development Tools:

JDK 1.8\_172 + Tomcat server 8?  
Eclipse IDE Neon

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

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

**5.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.

**6.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.

**Major Components in RPT-FDS, Layers, Packages, SubTasks in the Project:**

Definition of Component?

Assembling systems using component Assembly model

**Tasks List:**

**-1) Use Cases:**

1) Straight Thru case: “NO-HIT”, gets enqueued as response; all received XML requests are logged.

2) Sending an invalid Request: returns an EXCEPTION Error message to PE engine and let the GUI Admin know how to handle it.

3) Handling valid, well-formed XML Request messages for Fraud Detection processing.

1. Integrating pinging the FraudD Service in our project to query he health and other monitoring info from the “Spring Boot” Actuator (“Heartbeat”)

Background Scheduled Task: to check the health of the FRDS link and restart if the link has become stale/dead links

1. **MQ Manager:**

**Layer: com.**bnym.rpx.fraudD.mq.service.\*

Configure MQ Manager for our Fraud detection Queues.

Queue manager Name:

Channel:

RCV Queue Name:

SEND Queue Name: Command Channel?

No. of concurrent connections: 100 (Rahul)  
  
Listeners:

Manually insert PE Transaction messages in the Queue for now: one of the approved message type and another one with an unknown Message type.

Input: XML Request Message : CT, RFR,   
If the message type is not one of the above, log the input PE Transaction and enqueue an Error response in MQ SENDER q.

Input: XML Request message failed validation or corrupted.

1. XML Document Validation and XML-Java Bean JAXB or Castor Conversion:= from MQ-RCV  
   Java Bean to XML Response Conversion back to MQ Sender  
   NO XML Parsing here   
   All field data are String data types for immutability

**Layer: com.**bnym.rpx.fraudD.xml.castor.\*

1. MQ Response Messages: “HIT”, “NO-HIT”, “ERROR|EXCEPTION|OTHER” status messages enqueued in MQ-SEND q.
2. Come up with Exception cases, status codes to cover both MQ and Open Source APIs(later)

**3) Hazelcast Dist. Cache HMaps: Layer :** com.bnyn,rpx.fraudD.hazelcast.\*

- a) Setting up the Hazelcast runtime in our address space.

a) Reading the BL list data into HM maps

b) Matching tagged field name values against BL data in each of the HM maps for matches  
 c) Converting input Address data set into a canonical form for address matches or   
 mismatches among its data fields. (Pattern matching using regular expressions)  
 d) Generate new XML response message with “HIT|NO-HIT|ERROR|EXCEPTION” status codes  
 e) Write both the REQ and RESP messages in the db for auditing purposes(in what format to be written is to be decided  
 f) Enqueue the XML RESPONSE in the MQ\_SEND Queue and this worker thread dies if no more work. (newCachedThreadPool())

**4) Spring Boot Application:: MVC, Rest Controller, Actuator for monitoring, Orderly Startup, shutdown**

**a)** Spring Boot web application connects us with other systems using MQ message broker, AngularJS GUI, REST API . It is like an application container running in a JVM

We had a basic “spring\_web\_starter” module to be able to run under Tomcat 8.0. Adding other dependency libraries as we add additional layers of functionality on top of the Spring Boot. As they say “Convention over Configuration”.  
Do we need additional plug-ins for the Tomcat server to support MQ, Hazelcast, etc.

**5) Oracle JPA Repository: Layer:**

DB Isolation Level: Serializable. Reads wait for writes to complete.

**6) MVC Component: Sudhir K.**

The AngularJS4.0 GUI/SPA should be able to perform CRUD operation on the BL list of tables persistent store in Oracle db using Spring JPA Repository/DAO interface.

Oracle Schema Name: User/DBA Control: UserId: rtp-fradD Password:

Tables:

Field Names, score:: Field Names and scores allocated will vary over time, What-if-Analysis scenarios.  
These tables are audited.

Black List Line Items Table

**7**) Logger: Logging messages to console during DEV and move to the cloud later

Maven dependency……, log4j.xml

8) Junit, Mockito, Behavior Testing: QA Team

We have to make sure that we allocate time for haiving meaningful unit tests, testing critical methods first, several integration tests beween end-to end and from SB-MQ, SB to Oracle, Mq-Req to MQ-Resp tests.

9) Jenkins Build Component details:

10) Production Env : Do,s and Don’ts

12) Verifying the metrics of the RTP-Fraud Detection system

11) Arching the Design Doc under SharePoint-Confluence

**7.Software Components in RTP Fraud Detection System:**

**1) MQ Manager**Input : XML Req Message

**2) XML Document Validation and XML-Java Bean JAXB or Castor Conversion:= from MQ-RCV  
Java Bean to XML Response Conversion back to MQ Sender**

**3) Hazelcast Dist. Cache HMaps**

**4) Spring Boot Application:: MVC, RestController, Actuator for monitoring, Orderly Startup, shutdown**

**5) Oracle JPA Repository**

DB Isolation Level: Serializable. Reads wait for writes to complete.

**6) MVC Component:**

The AngularJS4.0 GUI/SPA should be able to perform CRUD operation on the BL list of tables persistent store in Oracle db using Spring JPA Repository/DAO interface.

Oracle Schema Name: User/DBA Control: UserId: rtp-fradD Password:  
Tables:

Field Names, score :: Field Names and scores allocated will vary over time, What-if-Analysis scenarios.  
These tables are audited.

Black List Line Items Table

**7) Logger:**

**8) Junit, Mockito, Behavior Testing:**

**7. Building RFDS Project Around the Spring Boot-MVC – System**

**MQ Manager:**

**Hazelcast:**Config file loading  
 Reading SQL store and loading the HMs  
 Address structure Pattern Matching-canonical structure for “Score” calculation  
 Scoring Algorithm => “HIT” or “NO-HIT”, score value

“Search using Predicate”

**UI-SQL Schema Creation and maintenance**

Life Cycle

Spring Boot Web Application with MVC  
Do we plan to use the REST API for local method calls for CRUD operations on Hazelcast cache and SQL Tables to manage BL data?

Maven based, Project Structure, Directories for the given Project

Setup:

Init() {

Read Properties files alod load into memeory  
 enable MQ-RCV set up operations Mq Manager; failure : ?  
 Enable Hazelcast ? failure =>   
 Create db table data first time  
 update Hazelcast cache tables from db every time from startup and every update   
 after GUI BL updates : db engine is not available ?

}

Loop: MQ-RCV (queue name) :: 10 Listeners {  
 read ten messages from Queue => ten threads  
 thread.run() {  
 Each thread runs “Scoring” algorithm  
 Build XML Responsed document for gven Message Type  
 MQ-SEND the RESPonse XML Message; AUDIT info, each thread terminate

}  
 continue

}

Shutdown() {

Shutdown procedure for all components

}

**8. RFDS Scoring Algorithm Design and Implementation, Hazelcast Cache, Sql DB:**

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.  
(PE => Payment Engine).

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 Segregation Responsibility Pattern (CQRS)

Command side of the CQRS pattern creates, updates and deletes CRUD operation on the db, Hazelcast cache 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 dropped.

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

9.MQ-RCV, MQ-SEND Configuration for in-flow of PE Transaction Data into RFDS Service

**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:**

**Gobi:  
Desing Documentation for Fraud Detecti**

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

**Surjan:**

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

**Access Rights to Resources:**

**Jenkins Access:**Please approve SRQST0000261761   SCCB request.SRQST0000261765 -Jenkins access

**JIRA Access**:: Work with offshore team JIRA raised for APP-Engine

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

**Extended Access:**

**Confluence 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