# Bank Transactions Reconciler

# Prototype Document

**Version: 2**

by

Krishna Angeras

July 01, 2017

# Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision** | **Date** | **Author** | **Section/Pages**  **Affected** | **Remarks** |
| Draft 1 | June 30, 2017 | Krishna Angeras | All sections, Initial |  |
| Version 2 | July 01, 2017 | Krishna Angeras | Compilation and edit |  |

# 

# Table of Contents

[Bank Transactions Reconciler 1](#_Toc486839708)

[Prototype Document 1](#_Toc486839709)

[Version: 2 1](#_Toc486839710)

[Change History 2](#_Toc486839711)

[Table of Contents 3](#_Toc486839712)

[1. Introduction 4](#_Toc486839713)

[1.1. Purpose 4](#_Toc486839714)

[1.2. Scope 4](#_Toc486839715)

[2. Prototype Features 4](#_Toc486839716)

[3. Prototype Architecture and Design 7](#_Toc486839717)

[4. Prototype Test Cases and Results 8](#_Toc486839718)

[5. Future Work 10](#_Toc486839719)

[6. Run Instructions 11](#_Toc486839720)

# 1. Introduction

## 1.1. Purpose

This document contains details about the Prototype of Business Transactions Reconciler project for the Finance team. The engineering task was to test the feasibility of a solution to process huge customer transactions and reconcile them. From the work described in this Requirements Specification (Process Customer Transactions - PDF), a prototype was designed with the use of Java based Frameworks like Spring Batch, which is a complete implementation of Java – JSR-352 specification (for Batch jobs).

## 1.2. Scope

This section provides the details on the scope of this Prototype Document and covers briefly, various sections of the document.

Section 2 of this document details the actual status on the implementation of the functionality as in the Requirement Specifications.

Section 3 provides an insight on the High Level Architecture and design used in the prototype.

Section 4 discusses briefly on the Test data and Test results.

Section 5 has a discussion on future work and suggestions that may be needed for the project.

Section 6 has details on how to execute the prototype.

# 2. Prototype Features

This section describes the features of the prototype for Bank Transactions Reconciler, which processes the customer transactions received as CSV files. The requirements are uses cases inherited from the Requirement Specification document. The high level estimation taken before design and code is provided below. PERT was used on the features discussed below for estimation. The total implementation time took around 24 hours approximately almost in-sync with the estimation.

Requirements analysis and initial design : 4 hours

Environment and initial code setup : 2 hours

Coding and unit testing : 7 hours

Testing : 4 hours

Logging, Exception handline, bugs, etc : 2 hours

Documentation, JavaDoc : 4 hours

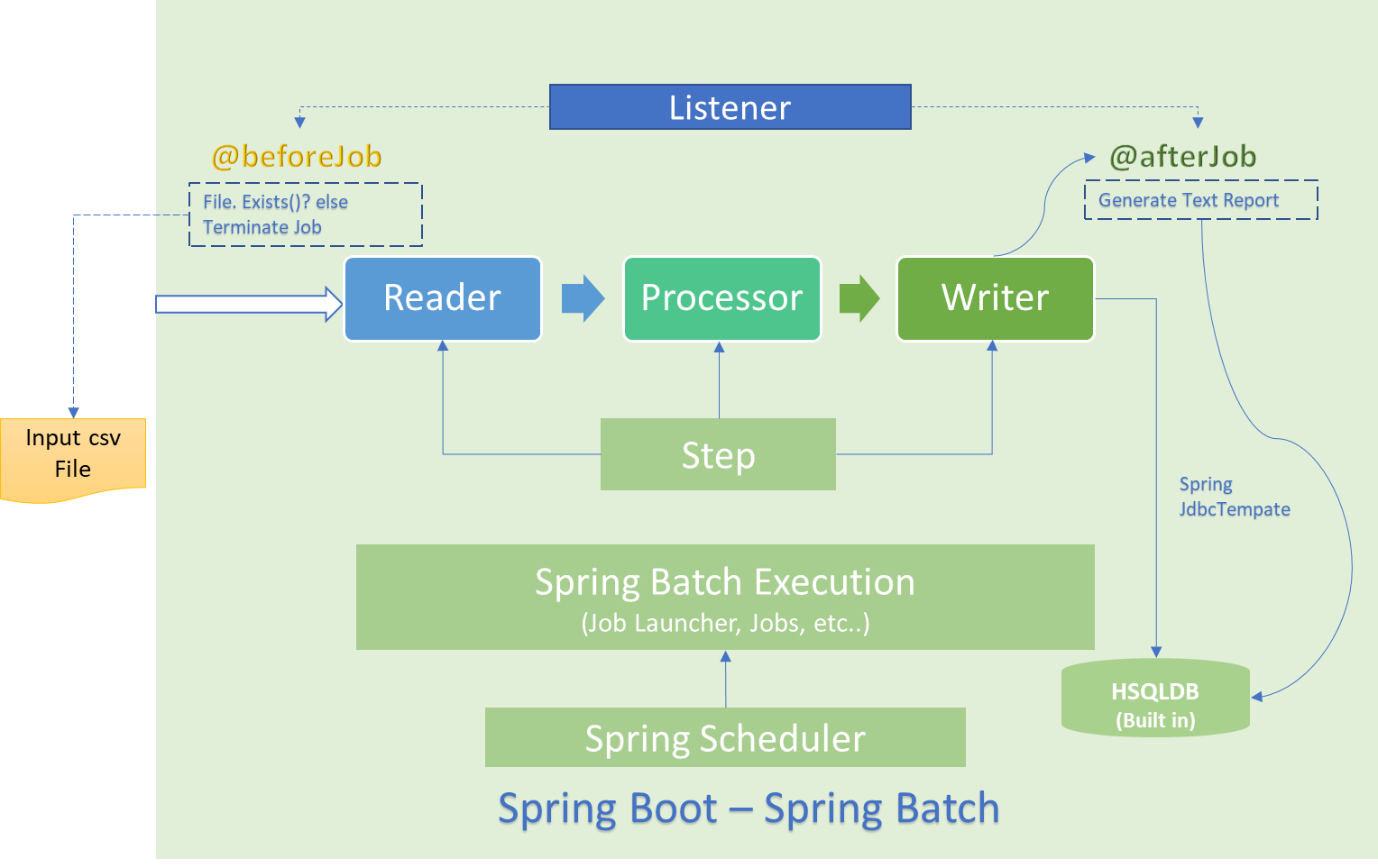
Implemented Features by Reference from Requirements and Design.

|  |  |  |  |
| --- | --- | --- | --- |
| **Req #** | **Module** | **Requirements** | **Implementation Details** |
| R.1 | Handle Input | Customer transactions are received as a File daily once at 06:00 hours once at 21:00 hours When a file is received, processing of the File should commence within 5minutes.  Files are received in a directory located at $TRANSACTION\_PROCESSING/pending.  where $TRANSACTION\_PROCESSING is an environment variable referencing a arbitrary directory on each machine. The program will execute the format of the file finance\_customer\_transactions-${datetime}.csv, where ${datetime} represents the time the file was written to the filesystem. | **Implementation:**  Handled using Spring Scheduler and Java File handling functionalities. If the at TRANSACTION\_PROCESSING is not found or file doesn’t exist, job will be terminated. **Assumptions:** It is assumed that the files will be completely written and available in the path exactly at 6 AM and 9 PM daily. The job retrieves the file at 6 AM and 9 PM. There will be only one input file per batch. Only one file (randomly picked) will be processed and rest will be ignored.  If the file is not available on the given times; the job will be terminated which can be processed manually.  **Limitations:**  The Prototype does not check if the file is completely written and available for reading. Also the prototype does not care if there are multiple files. The prototype works as the above mentioned assumption. |
| R.2 | Read CSV Input | File is received in csv format; example format provided in Requirements Each file can contain up to 500K customer payments Each line in the file represents a customer transaction. Customer accounts numbers contain numerical characters only. however some lines are encountered where the account number contains non numerical characters - these lines are considered corrupt. Corrupt lines should not be processed but skipped | **Implementation:**  Implemented using Spring Batch - Item Reader interface - Chunk Reading (1000) for faster reading - SkipPolicy to skip corrupt lines. **Assumptions:** Apart from customer Account number, same non numerical validation has been placed on transaction amount as well.  Both Account Number is restricted to 10 digits and Transaction amount is restricted to 10 digits and 2 decimals. Records that dont pass this validations will also be skipped.  **Limitations:** N/A |
| R.3 | Process Input | Files must only be processed once The same customer account number can appear multiple times in a customer payment file | **Implementation:**  Implemented using Spring Batch - Item Reader and ItemWriter Interface. File is read once and inserted into a db-tables, which will be used for reporting. **Assumptions:** N/A **Limitations:** N/A |
| R.4 | Reporting | Initially the Finance stakeholders would like to see the program process each file and produce a text report detailing name of the file processed the number of accounts processed total credit amount total debit amount number of skipped lines The report should be written to the directory $TRANSACTION\_PROCESSING/reports report file name should be of the form finance\_customer\_transactions\_report-${datetime}.txt | **Implementation:** Implemented using Spring Batch - JobExecutionListener. After the ItemReader, Processor and Writer are executed, the database will be queried to generate the text report. **Assumptions:** Number of Accounts Processed - is the count of the unique account processed.  Total Credit is the total of positive numbers in the file and Total Debit is the total of all negative numbers in the file (no grouping with customer numbers). 'reports' path will be created in 'TRANSACTION\_PROCESSING' if it doesn’t exist. **Limitations:** N/A |
| R.5 | Post Processing | Processed files should be placed in $TRANSACTION\_PROCESSING/processed | **Implementation:**  Java File handling  **Assumptions:** 'processed' path will be created in 'TRANSACTION\_PROCESSING' if it doesn’t exist. **Limitations:** N/A |

# 3. Prototype Architecture and Design

This section provides a high level insight on the architecture, tools and design used for developing the prototype. The architecture and design is arrived after considering the uses cases in the requirements and their corresponding assumptions.

**High Level Architecture/ Process Flow**



* Reader: FlatFileItemReader is used to read contents of CSV file, then maps the read data to fields of DTO.
* Processor: Pass on the DTO object to writer as it is.
* Writer: writes batch of records to PostgreSQL Database using DAO.
* Listener: handles beforeStep and afterStep tasks, Pre-validation and reporting respectively.
* Spring JdbcTemplate is used to execute SQL Queries.
* Cron, SQL Queries, and other properties are defined in Properties file (application.properties located at src/main/resources)
* Log file (application.log) is generated in one step above the root directory.

**Technologies and Frameworks used:**

Language : Java (version 8)

Frameworks : Spring Boot (1.5.2), Spring Boot-Batch (1.5.2),

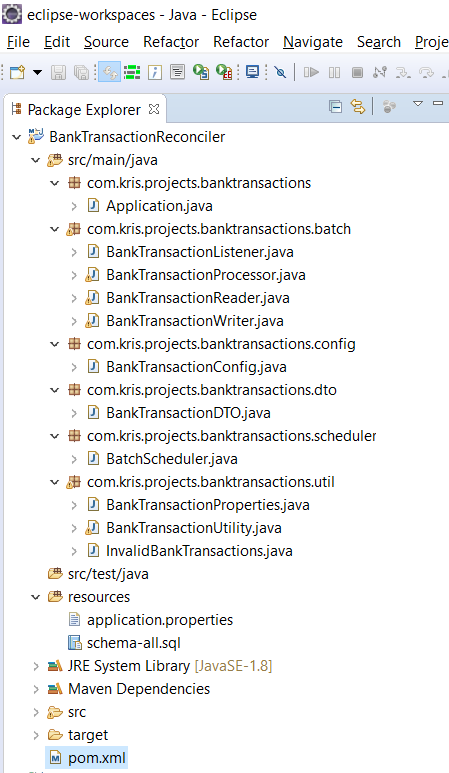
Database : Built in HSQLDB (can be ported to other DataSources)

Dependencies : Maven

Batch Mechanism : Cron using Spring Scheduler (configurable in application.properties)

For more insight on classes and methods created, Javadoc can be referred.

**Package Structure**

****

# 4. Prototype Test Cases and Results

This section contains some (not all) of the Unit Testing, System Testing and Performance testing scenarios and corresponding results. A sample test data is also provided as an attachment in this section.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Test Case** | **Expected Result** | **Status** |
| 1 | $TRANSACTION\_PROCESSING Path is not set/ Incorrectly set | Batch Job should be terminated | Pass |
| 2 | File Format should be  finance\_customer\_transactions-${datetime}.csv | Read only the file with same file format. Other files should be ignored | Pass |
| 3 | In case of Multiple files, read only one file (as per the assumption) | Read only one file (randomly) (as per the assumption) | Pass |
| 4 | Processed files should be placed in ./processed | File will be moved from ./pending to ./processed Create ./processed if not found. Additionally if the same file already exists, rename it to filename\_backup and copy the new file | Pass |
| 5 | Same customer account# can appear multiple times. Reporting to be done for unique customer accounts. | Text report field Total Accounts, to contain distinct customer accounts | Pass |
| 6 | When a file is received at 6 AM and 9 PM, processing of the File should commence within 5 minutes | Batch Process to start within 5 minutes. | Pass |
| 7 | Both account number and amount should have 10 digits. Additionally amount can have 2 decimals. (as per assumption). | To be generated as per the assumption | Pass |
| 8 | Validate for any alphanumeric or special characters in Account Numer, Amount | Lines that have Alphanumeric and Special characters will be skipped and skipped records will be counted for reporting | Pass |
| 9 | Report should be generated in ./reports | Report file should be seen in ./reports directory. If ./reports directory doesn’t exist, create new | Pass |
| 10 | Report to be generated as the sample provided in the requirement.  Positive numbers represent credit and negative numbers represent debit  It is assumed that Total Debit and Total Credit is the sum of all the debits and credits in the customer file and not grouped by accounts. | File Processed to have the file name. Total Accounts will have the count of distinct account numbers Total Credits will have the total sum of (+) amounts in the file. Total Debits will have the total sum of (-) amounts in the file  (as per assumptions) number of lines skipped will have the count of corrupt lines in the csv | Pass |
| 11 | Process 500k records with a decent Performance | Performance to be targetted <1 minute for 500K. (test with 1 million records) | Pass |

**Sample Test Data**

****

# 5. Future Work

The prototype demonstrates the functionality of reconciling the customer Transaction by processing the inputs and generating the report in a batch system.

More additions can be accommodated in future when the prototype is approved to transform into a project. Some future suggestions include:

Process multiple csv files at input instead of a single file

– Spring batch can be configured to a multi threaded mode to achieve high performance.

Since this is a prototype, HSQLDB which is built in db for spring batch is used. More powerful databases or NoSQLs can be used for data processing and reporting

Provide more robust validations (for eg: a person can deposit only ‘n’ amount on a given day,etc..) - Spring validator framework can be extensively used for this.

Develop the application for a web container and create UI and reporting.

Scheduling – Currently the prototype is a stand alone batch job scheduler. The prototype uses only basic scheduling mechanism. A more robust scheduler framework (like quartz, etc.. ) can be integrated with spring to monitor and operate the batch jobs.

Further data analysis can be included to analyze customer data - to find his/her spending trend, etc..

# 6. Run Instructions

**Pre-requisite:**

The environment variable “TRANSACTION\_PROCESSING” should be set to a valid folder. (preferably as System variable; User also works, but only for the current logged in). Java 8 environment should be available.

**From Eclipse:**

Developed and Tested with Neon.

Import the code to your workspace. Navigate to and execute Application.java.

**From command prompt/ shell:**

Go to the target folder >>Java -jar BankTransactionReconciler-0.0.1-SNAPSHOT.jar

**As a Service: (untested)**

From Windows – use a service wrapper like winsw and configure the service.

In Unix/ Linux – Write a custom script and wire it with systemmd/ initd services.

Can also be executed using scheduling frameworks like quartz/ controlm etc..

===========================Thank you=============================