**Design Document**

**1 Introduction**

**1.1 Purpose**

The purpose of this document is to outline the technical design for the claims

transaction processing of **ABC Insurance company** and provide an overview for the

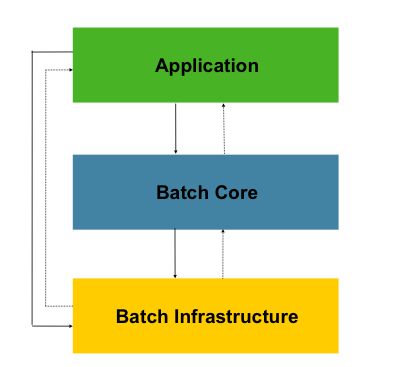
claims transaction processing implementation.

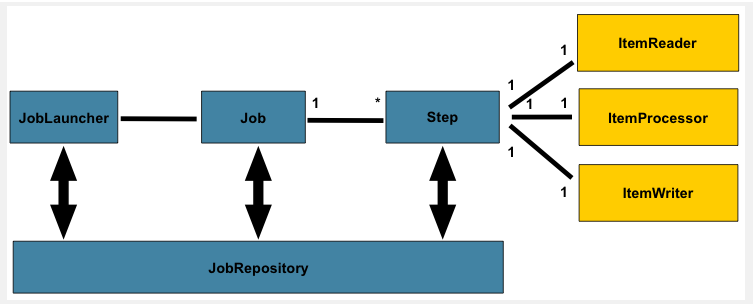
**2. Application Architecture**

Application architecture defines the various components and their interactions in context of a whole system.

**Spring Batch**

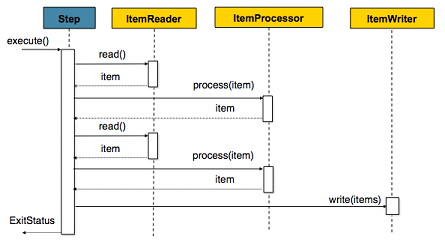
The Spring Batch processing architecture is used. Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications vital for the daily operations of enterprise systems.This layered architecture highlights three major high level components: Application, Core and Infrastructure. The Batch Core contains the core runtime classes necessary to launch and control a batch job. It includes JobLauncher, Job and Step implementations. Both Application and Core are built on top of a common infrastructure. This infrastructure contains common readers and writers





**Chunk-Oriented Processing**

Spring Batch uses a 'Chunk Oriented' processing style within its most common implementation. Chunk oriented processing refers to reading the data one at a time, and creating 'chunks' that will be written out, within a transaction boundary. One item is read in from an ItemReader, handed to an ItemProcessor, and aggregated. Once the number of items read equals the chunk size, the entire chunk is written out via the ItemWriter, and then the transaction is committed.



### **Skip Logic**

There are many scenarios where errors encountered while processing should not result in Step failure, but should be skipped instead. Usually these bad records are logged as well.

**Asynchronous Job**

CompletableFuture is one of the most important enhancements in Java 8. CompletableFuture is used to run the job asynchronously from the GUI.

**Logging**

Logging in the application is done using Logback. Logback is intended as a successor to the popular log4j project. Logback, is faster and has a smaller footprint than all existing logging systems.

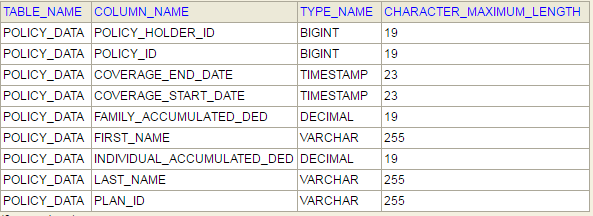
**3. Database Architecture**

H2 is an opensource lightweight Java database. H2 in memory database is used in this apllication.

**3.1 Tables**

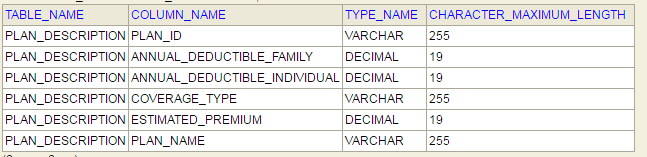
**3.1.1 PolicyData**

This table consists of the policy related data for each policy holder. The table structure is :



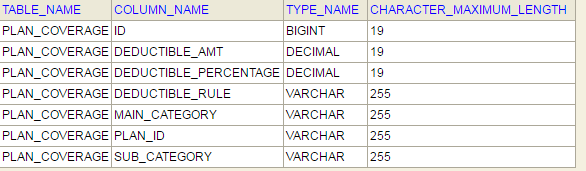
**3.1.2 PolicyDescription**

This table consists the details about the different policy plans. The table structure is :



**3.1.3 PlanCoverage**

This table consists data regarding what policy plans covers, i.e. how much policy holder would pay and how much plan will pay for various services. The table structure is :



Spring batch creates following tables to store details of job and step execution.

* BATCH\_JOB\_EXECUTION
* BATCH\_JOB\_EXECUTION\_CONTEXT
* BATCH\_JOB\_EXECUTION\_PARAMS
* BATCH\_JOB\_INSTANCE
* BATCH\_STEP\_EXECUTION
* BATCH\_STEP\_EXECUTION\_CONTEXT

**4 Software Components**

The System is a data driven Application which uses the following :

* Spring Boot
* H2 Database
* Thymeleaf
* Java 1.8
* JPA
* STS IDE
* Apache Tomcat server
* Junit

**5 Application Overview**

The application can be run through the GUI, command prompt and curl.

**Input :**

Input consists of the following :

* Location of the input transaction file
* Location of processed output file

**Output :**

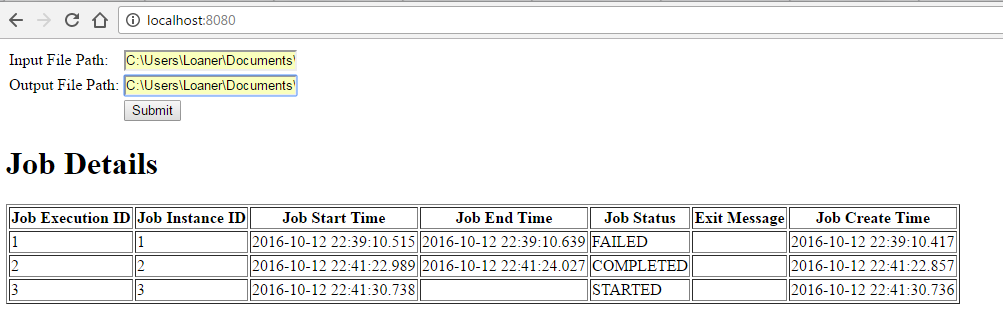
The output is a .csv file with the policy details such as amount paid by the policy holder, amount paid by plan, error information etc.

**GUI**

The GUI consists of 2 input fields :

* Location of the input transaction file
* Location of processed output file

It displays all the job execution details in the tabular format. The jobs are called run asynchronously. The processed output file will be written in the specified output location path.



**Command Prompt**

The application can be run through command prompt by providing the following inputs :

* Location of the input transaction file
* Location of processed output file

The processed output file will be written in the specified output location path.

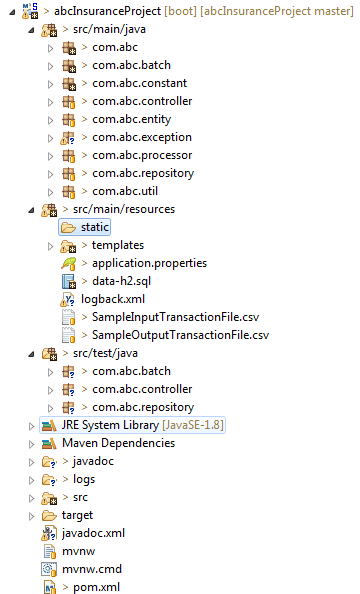
**Curl command**

Real time web services calls can be simulated using curl and/or through browser. In the request specify the following input values :

* Policy ID
* Policy Holder ID
* Date Of Service
* Coverage Main Category
* Coverage Sub Category
* Billed Amount.

curl "<http://localhost:8080/ABCInsurance/processPolicy?policyId=100001&policyHolderId=1000011&dateOfService=5/6/2016&coverageMainCategory=Inpatient%20Hospital%20Care&coverageSubCategory=ROOM%20AND%20BOARD&billedAmount=1000>”

**6. Project Structure**



**7. Assumptions and constraints**

* All the amounts values in the database and displayed in the input/output files are considered in currency USD by default.
* H2 in memory database is used, hence the database content is lost when the server is stopped.
* If the Policy transaction “Date of Service” is after the “Coverage End Date”, then error message will be displayed in the output file.
* Spring batch chuck processing is used. The chunk value is specified as 10.
* Skip Policy processing is while reading. The “FlatFileParseException” is skipped for 5 records. The application will throw exception if there are more than 5 file parsing errors in the input file.