TUTUKA TRANSACTION COMPARISION

ACCEPTANCE DOCUMENT

MOTIVE:

To match two transaction report files – one from Tutuka and other from a Client. There could be mismatches due to discrepancies in the KYC process/front-end user entered values and hence calls for a fuzzy cross matching.

Assumptions:

1. Transaction ID is the unique ID and only repeats in case of a reversal. Hence, the composite primary key Tkey = Transaction_ID+TransactionDescription
2. If Transaction ID or TransactionDescription is NULL/ZERO, then the transaction is marked as BAD_TRANSACTION and will not be fuzzy matched with the transactions in the other file
3. If the Tkey is not present in the other file, it is marked as UNMATCHED
4. Only if Tkey matches, it qualifies for a fuzzy matching.
5. Transaction Narrative field is an address field and will have to be fuzzy matched. As almost NONE of the values from the files [TutukaMarkoffFile20140113.csv & ClientMarkoffFile20140113.csv] resolved to a valid location, geo-location fuzzy matching was out of the equation.
6. Also, assumed was that the field will contain maximum noise and hence, resorted to look at this column as a regular string valued column.
7. Client transactions may be logged in with their local time and hence, there might be time zone differences and two very different TransactionDate fields may point the same time. Since there was no information on the client's time zone, assumption was made that the client would be in Africa too and a tolerance level of 180 minutes[largest time zone difference across Africa] was added in the fuzzy logic match
8. For those transactions that have different TxIDs but other transactions do match mostly, are still considered as UNMATCHED as TxID is an unique ID and a null value of which is considered a BAD_TRANSACTION. Reason for this is TxID should be a generated value for every transaction and cannot be the same for two different transactions. It is better NOT TO match and show up in the report as UNMATCHED and end up as a bug on the client side caught late in the cycle than to treat it as a typo

Design Overview:

UML -> https://raw.githubusercontent.com/quas0r/TxFuzzyCompare/master/tutukaUML.png

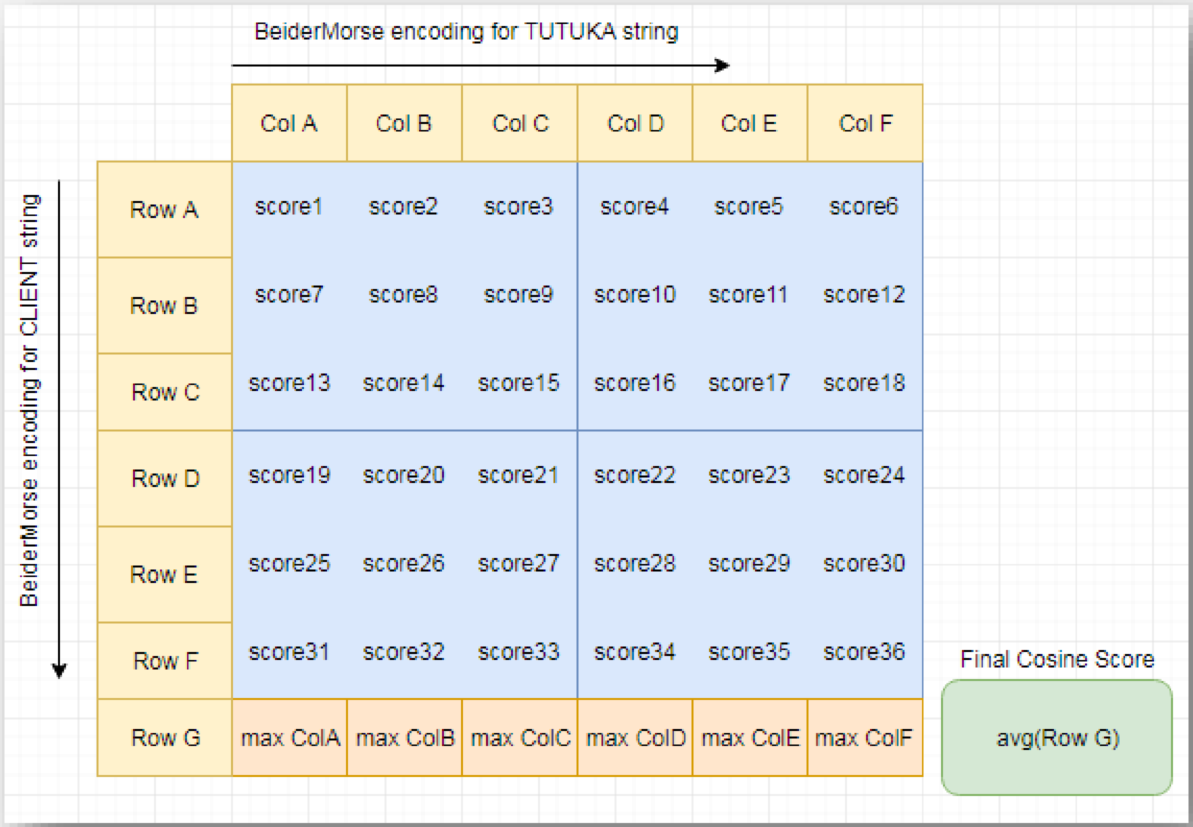
Spring Boot is used with Thymeleaf as the template engine.

Except for certain 3rd party libraries like guava, apache commons etc, adding in dead code has been avoided. Reading CSV to Bean classes has been done through Java 8 streams rather than OpenCSV. The only advantage OpenCSV had was the ability to couple header names with bean variable so that the order of header columns wouldn't matter but had a huge performance overhead.

A CSV having 100,000 transactions took ~ 3000 ms while Java 8 stream took ~ 700ms. MappedByteBuffer took around the same time as Java 8 streams. So finally settled with Java8 stream and implemented a simple solution for loose coupling on header column order.

For the fuzzy matching of TransactionNarrative field, tried with different matching algorithms from Levenshtein distance to a combination of edit distance and phonetic matching algorithms like double metaphones. None were accurate enough. Finally, Beider-Morse algorithm generated encodes and the average cosine similarity between the two arrays of encodes did the trick albeit with minor discrepancies discussed later.

Cosine Similarity Score Calculation: Phonetic Matrix Method


BeiderMorse encoding for TUTUKA string
BeiderMorse encoding for CLIENT string
Final Cosine Score
avg(Row G)

	Col A	Col B	Col C	Col D	Col E	Col F
Row A	score1	score2	score3	score4	score5	score6
Row B	score7	score8	score9	score10	score11	score12
Row C	score13	score14	score15	score16	score17	score18
Row D	score19	score20	score21	score22	score23	score24
Row E	score25	score26	score27	score28	score29	score30
Row F	score31	score32	score33	score34	score35	score36
Row G	max ColA	max ColB	max ColC	max ColD	max ColE	max ColF

Initial plan was to construct a Phonetic Matrix as above, with scores representing cosine similarity scores with left and right vectors as Beider Morse encoding of Tutuka and Client TxNarrative strings. But as we can see in the report generated here at CosineSimilarityReport, just the linear average did the trick. But if need be in the near future to add additional fields like Name, the above method helps.

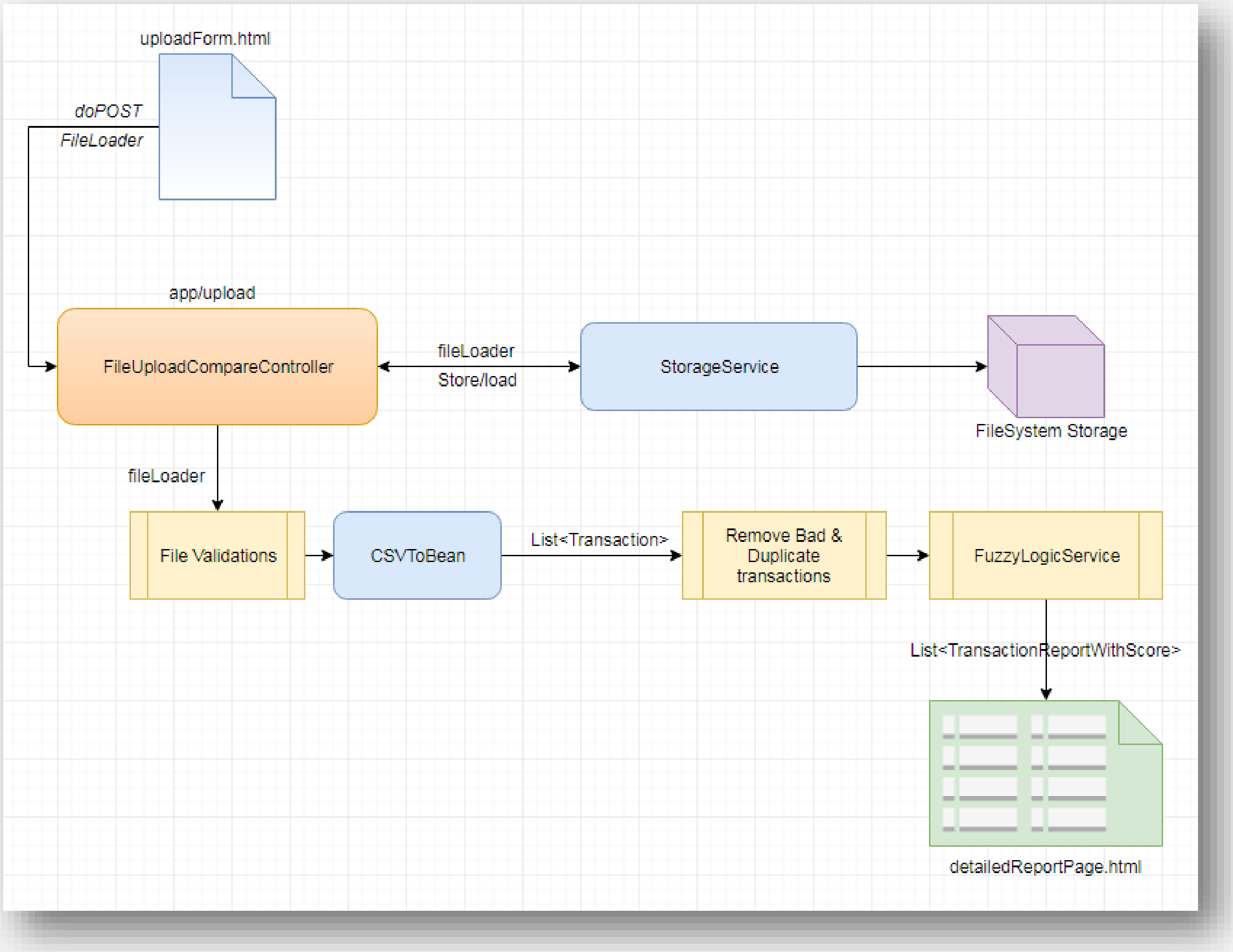
Note: Incase name field need be added in the future, it should be noted that Name Fuzzy Matching is different completely and cannot be treated as simple strings.

Eg) John and Jonathan may be the same person with different nicknames

The matched/unmatched transactions pairs are categorized as denoted in the Enumeration, Result.java as

```
No Review:
        PERFECT_MATCH,
        PERMISSIBLE_MATCH,
Needs Review:
        PROBABLE_MATCH,
        PROBABLE_MISMATCH,
        PERFECT_MISMATCH,
        BAD_TRANSACTION,
        DUPLICATE,
        UNMATCHED
```

Flow Diagram:



TxNarrative field goes through the following validation/transformations before fuzzy matching

Null Validations -> Trim non-ASCII value -> Normalization -> Stop Word Replacement ->Removal of Vowels

Normalization	:	záýn street	-> zayn street
Stop Word Replacement	:	First road	-> 1ST RD

Improvements that could be done:

- With the help of more transaction data, machine learning algorithms like Decision Trees could be employed. With around 305 transactions and 6 categories to be classified into *[apart from Bad and Duplicate transactions]* the generated hypothesis would be severely affected by over-fitting.
- Could add an additional column in the report to get the agent's input on final STATUS after his/her review and persist in a database. If the status is different than the one predicted, it could be added to the 'delta' during the next run of ML algorithm re-train.
- Could have put more thought into the UI for report display. Added the hover option to help not lose track on the row while scrolling but horizontal scrolling is NOT a good design. May be persist the report as a CSV and provide an option to download on the same page.

Though this works for the most part for the given problem set, in case of adding additional fields like Name, Memo, etc. will need feature extraction, running a machine learning model[multivariate logistic regression] on it and arriving at respective 'weightages' for features. I'd recommend Apache Spark that has built in support for most of the machine learning algorithms [MLlib] or something simpler like the one here in the Jupyter notebook I did for Cross Matching AT20GBSS and Super COSMOS Sky Surveys Catalogues using Nearest neighbor search Algorithm and KD-Tree data structure in O(n*log(n)).

Special Case Scenarios:

- When Tx Narrative in
  Tutuka File = ' MILO MALL BW SEROWE '
  and Client = ' SEROWE MILO MALL BW '

And all other fields do match, this would be reported as PROBABLE_MISMATCH for review but should have been marked as PROBABLE_MATCH at the least.

- When Tx Narrative in
  Tutuka File = ' MILO MALL BW SEROWE '
  and Client = ' MALO MALL BW SEROWE ' and are very different VALID addresses.

Since vowels are removed before the Beider-Morse encoding, both MILO and MALU would resolve to same encoding and report as a perfect 100% match on the cosine similarity score generation. But two transactions from 2 different malls end up having the same transaction Ids and other fields is very rare, but a case to be noted nevertheless.