**Epic 1: Create a CC Transaction Processor**

**Description:** As a product owner, I want to Develop a Credit card transaction processor application with following scope and acceptance criteria:

**Scope:**

1. Create a spring boot application
2. You will get a directory with following folders
   1. new, processing, done, error, garbage
   2. new will have csv files with following naming patterns
      1. transactionXXXX.csv, XXXX denotes an integer
3. The application will read from data from above csv files
   1. It will be credit card transactions data having data for following entities
      1. Transactions
   2. Transaction data will be present in transactionXXXX.csv file
   3. There could be multiple transactionXXXX.csv files and can be processed in parallel
   4. Transaction data files could be very big and may contain millions of entries
4. Schedule the application/process to run as soon as a file arrives
   1. Maintain status of each run and transactions that it have processed
5. Error handling
   1. Do not process any file that does not correspond to naming convention
      1. Move them to garbage folder
   2. New files will be received in new folder
   3. Move the file to processing folder before start processing
   4. If you get any schema error do not process the record and put it in an error file with same name as original file with error appended to it
      1. e.g. transactions0123.csv will have transactions0123-error.csv
   5. Move a processed file to done folder
6. Publish the processes records into a separate Kafka topic
   1. One topic for transactions
7. Security
   1. Communication with kafka should be done over secure channel,
      1. Preferably mTLS
8. Performance
   1. System should be capable of handling 10k records per second
9. Logging
   1. Every processed records should be logged in log file
   2. Every processing of record should have a unique id across different runs of application
   3. Do not log following information in logs
10. Customer name and address
11. Card no and expiry date
12. Merchant name and address

**Acceptance Criteria :**

1. Code quality
   1. 100% unit test code coverage for business logic
   2. Integration test cases
   3. Performance, Load and security testing to validate NFRs
   4. Should not be able to push code if coverage is less than 100%
      1. Lines and branch
   5. Github message should be like
      1. CCT-XX:
      2. Should not be able to push code if the condition is not satisfied
2. Source code repository and version control
3. Containerise the application
4. Automate the build and deployment process
   1. Integration with Jenkins for CI/CD

**Epic 2: Credit card transaction handler (Depends on Epic 3 for DB design)**

**Description :** As a product owner I want to build an application with following scope and Acceptance Criteria :

**Scope:**

1. Listen to kafka topics as created by credit card transaction processor (Can try some kafka connector also)
   1. Topic listener for transactions
2. Read every record in topic and persist it in database created as part of Story 1
   1. Please take meaningful defaults for column sizes for various fields
3. Security
   1. Communication with kafka should be done over secure channel,
      1. Preferably mTLS
4. Performance
   1. System should be capable of handling 10k records per second
5. Logging
   1. Every processed records should be logged in log file
   2. Every processing of record should have a unique id across different runs of application
   3. Do not log following information
      1. Customer name and address
      2. Card no and expiry date
      3. Merchant name and address

**Epic 3: Credit card Appplication**

**Description :** As a product owner I want to build an application with following scope and Acceptance Criteria :

**Scope:**

1. Create a spring boot application
2. Create application for front end (React App)
3. Create following tables with columns :
   1. customer id (PK) name account\_no
   2. credit\_card card\_no (PK) card\_type (FKey ref\_card.card\_type) expiry\_date customer\_id (FKey customer.id)
   3. ref\_card id (PK) card\_type
   4. transaction id (PK) card\_no (FKey credit\_card.card\_no) merchant\_id (FKey merchant.id) transaction\_type (FKey transaction\_type.id)
   5. ref\_transaction\_type id (PK) transaction\_type
   6. merchant id (PK) account\_no name
   7. Create Liquibase/Flyway script as database scm
4. Create following rest apis
   1. Admin apis
      1. Customer apis
         1. Fetch customer details
         2. Create a customer with appropriate details
         3. Update customer details
         4. Remove customer (soft removal)
      2. Merchant apis
         1. Fetch merchant details
         2. Create a merchant with appropriate details
         3. Update merchant details
         4. Remove merchant (soft removal)
      3. Credit card apis
         1. Fetch credit card details
         2. Add a credit card to a customer
         3. Remove a credit card (soft removal)
         4. Update credit card details
      4. Fetch transactions by credit card type
         1. Sortable by date of trx, customer country, merchant country
         2. Should be able to fetch trx pagewise, should be able to specify page size as well
      5. Fetch transaction for a date range, last month, current month
         1. Sortable by date of trx, customer country, merchant country
         2. Should be able to fetch trx pagewise, should be able to specify page size as well
      6. Fetch transaction for a customer or merchant for a date range, last month, current month
         1. Sortable by date of trx, customer country, merchant country
         2. should be able to fetch trx pagewise, should be able to specify page size as well
      7. Default page size is 20 and page no 1
   2. User apis
      1. Customer apis
         1. Fetch customer details
         2. Update customer details e.g. address
      2. Credit card apis
         1. Fetch credit card details
         2. Update credit card details
      3. Fetch transactions for a credit card (customer can have many)
         1. Sortable by date of trx, customer country, merchant country
         2. Should be able to fetch trx pagewise, should be able to specify page size as well
      4. Fetch transaction for a date range, last month, current month
         1. Sortable by date of trx, customer country, merchant country
         2. Should be able to fetch trx pagewise, should be able to specify page size as well
      5. Default page size is 20 and page no 1
5. Create following screens
6. Input handling
   1. Validate input fields, choose meaningful formats default values
   2. Handle sql injection issues
7. Security
   1. Place appropriate authentication and authorization mechanisms e.g.
      1. Admin should be able to call apis
      2. Customer should be able to fetch their own data only
   2. Rest communication must be over secure channel
8. Performance
   1. Any api should be able to process a request within 2 seconds
   2. Any page should load within 5 seconds
9. Output handling
   1. Client must never receive an exception stack trace, send in meaningful messages
      1. E.g. If no record exist for a search criteria then "No data exists for your search criteria" should be returned along with an empty object.
10. Logging
    1. Every processed records should be logged in log file
    2. Every processing of record should have a unique id across different runs of application
    3. Do not log following information
       1. Customer name and address
       2. Card no and expiry date
       3. Merchant name and address

**Acceptance Criteria :**

1. Code quality
   1. 100% unit test code coverage for business logic
   2. Integration test cases
   3. Performance, Load and security testing to validate NFRs
   4. Should not be able to push code if coverage is less than 100%
      1. Lines and branch
   5. Github message should be like
      1. CCT-XX:
      2. Should not be able to push code if the condition is not satisfied
2. Source code repository and version control
3. Containerise the application
4. Automate the build and deployment process
   1. Integration with Jenkins for CI/CD