Purpose of the Project

The purpose of this project is to import a 10-column CSV into an SQLite database. Upon calling the POST endpoint, the system inserts all the valid rows into the SQLite database.

The system assumes that the CSV consumed is 10 columns in length. The system tags rows that do not have 10 columns as bad data, and stores them in a separate CSV file for reporting purposes. A brief log file is also generated as a reference, indicating the number of rows processed, rows inserted in the database, and rows tagged as bad data.

Each time the POST API for the data import is called, the system resets the `client` table within the SQLite database.

How to Run the Application

1. Ensure that the local machine is running on JDK 8 since the Spring Boot project uses Java 8.
2. Perform the Maven command `mvn clean install -U` before running the project.
3. Execute the POST API using Postman or cURL command:

   ### POSTMAN Steps:

   1. Fill out the following fields in postman:

      ```
      Request Method: POST
      url: localhost:8080/api/v1/client
      Headers: content-type: application/json
      ```

      For the request body, note the file name of the CSV to be imported as well as the absolute file path.

      ```
      Request Body:
      {
        "fileName": "sampleFileName",
        "filePath": "/Users/user/Documents/sampleFileName.csv"
      }
      ```

   ### cURL Steps:

   1. Execute the following command:

```
curl --location --request POST 'http://localhost:8080/api/v1/client'
--header 'Authorization: '
--header 'Content-Type: application/json'
--data-raw '{
    "fileName": "sampleFileName",
    "filePath": "/Users/user/Document/sampleFileName.csv.csv"
}'
```

4. The output (the CSV File containing the bad data and the log file) will be generated in the project's classpath formatted as follows:

```
sampleFileName-bad.csv #File with bad data
sampleFileName.log #Log file
```

5. To verify how many records have been inserted into the SQLite database, a GET API can be called to output the number of records inserted in the database. To do this, perform the instructions indicated in **Step 3** and replace the request method **POST** to **GET**.

Design Overview

The code observes MVC architecture because this type of architecture is the most suitable design pattern to follow for Spring Boot projects. It promotes a faster development process since it loosely couples each component from another, making the codebase more scalable, extensible, and more robust.

A few useful dependencies, aside from the required dependecies, are also included. Lombok for saving time and lines of code by not having to provide encapsulation methods and constructors because Lombok automatically generates them for you. Java Validation API also is included as a dependency for simple base validation for the models. Java Persistence API is also added to make CRUD database operations easier and also to reduce the total lines of code.

Of course, the code adheres to the 4 basic principles of object oriented programming (Encapsulation, Abstraction, Inheritance, and Polymorphism) by:

- Setting the variables of the model objects to private and generating accessor and mutator methods for them using Lombok.
- Applying polymorphism and abstraction to the service layer methods to provide scalability and reduce future code issues.

The estimated amount of time that it took to complete the code is about 2 full working days for the code to work and half a day for reviewing the code and writing the documentation.