Project Documentation

Spring Boot Batch Application

---

1. Overview

This is a Spring Boot-based batch processing application designed to read customer data from a text file during application startup and insert it into a database. Additionally, the application provides two RESTful APIs to interact with customer data.

2. Batch Processing
2.1 Batch Execution on Startup

Upon starting the application, the CustomerBatchApplication class initializes and executes a batch job using Spring Batch. This is triggered by a CommandLineRunner bean that launches the job with unique parameters to avoid duplication.

2.2 File Reading Process

The batch job reads customer data from a .txt file located in the resources/input/ directory. Each line in the file corresponds to a single customer's data. Blank lines and improperly formatted data are safely skipped using fault-tolerant configuration.

The file uses a pipe (|) delimiter and contains the following fields:
- accountNumber
- trxAmount
- description
- trxDate
- trxTime
- customerId

This data is mapped into a Customer entity and inserted into the database.

3. Batch Components

The batch job consists of the following key Spring Batch components:

3.1 Reader

Class: FlatFileItemReader
Reads each line from the .txt file and maps it to the Customer entity using a tokenizer and custom field set mapper.

3.2 Processor

Class: CustomerItemProcessor
(Optional) Used to perform transformations or validations before writing to the database.

3.3 Writer

Class: JpaItemWriter
Persists each Customer object into the database using JPA.

3.4 Step Configuration

Step Name: customerStep
A chunk-oriented step that reads, processes, and writes customer records in batches of 10.
Configured to skip parsing errors and blank lines using fault-tolerant settings.

3.5 Job Configuration

Job Name: importCustomerJob
Executes the configured step as part of the job workflow.

4. RESTful APIs

The application exposes the following APIs to interact with the customer records:

4.1 Retrieve Customer Data

Returns customer data based on the provided search criteria.

Endpoint: /api/customers/search
Method: POST
Sample request payload:


```
{
  "description": "FUND TRANSFER",
  "accountNumber": "8872838283",
  "customerID": "222",
  "page": 0,
  "size": 10
}
```


Sample response:


```
{
    "content": [
        {
            "id": 1,
            "accountNumber": "8872838283",
            "trxAmount": 123.00,
            "description": "FUND TRANSFER 2",
            "trxDate": "2019-09-12",
            "trxTime": "11:11:11",
            "customerId": 222,
            "version": 3,
            "createdAt": "2025-08-06T18:04:46",
            "updatedAt": "2025-08-06T18:05:21"
        },
        {
            "id": 3,
            "accountNumber": "8872838283",
            "trxAmount": 1223.00,
            "description": "FUND TRANSFER",
            "trxDate": "2019-10-11",
            "trxTime": "11:11:11",
```

```
    "customerId": 222,
    "version": 1,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-06T18:04:46"
},
{
    "id": 4,
    "accountNumber": "8872838283",
    "trxAmount": 1233.00,
    "description": "3RD PARTY FUND TRANSFER",
    "trxDate": "2019-11-11",
    "trxTime": "11:11:11",
    "customerId": 222,
    "version": 1,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-06T18:04:46"
},
{
    "id": 5,
    "accountNumber": "8872838283",
    "trxAmount": 1243.00,
    "description": "3RD PARTY FUND TRANSFER",
    "trxDate": "2019-08-11",
    "trxTime": "11:11:11",
    "customerId": 222,
    "version": 1,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-06T18:04:46"
},
{
    "id": 6,
    "accountNumber": "8872838283",
    "trxAmount": 12553.00,
    "description": "3RD PARTY FUND TRANSFER",
    "trxDate": "2019-07-11",
    "trxTime": "11:11:11",
    "customerId": 222,
    "version": 1,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-06T18:04:46"
},
{
    "id": 9,
    "accountNumber": "8872838283",
    "trxAmount": 2123.00,
    "description": "FUND TRANSFER",
    "trxDate": "2019-09-11",
    "trxTime": "11:11:11",
    "customerId": 222,
    "version": 1,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-06T18:04:46"
},
{
    "id": 10,
    "accountNumber": "8872838283",
    "trxAmount": 1323.00,
    "description": "FUND TRANSFER",
    "trxDate": "2019-09-11",
    "trxTime": "11:11:11",
    "customerId": 222,
    "version": 1,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-06T18:04:46"
```

```
        },
        {
            "id": 11,
            "accountNumber": "8872838283",
            "trxAmount": 12443.00,
            "description": "FUND TRANSFER",
            "trxDate": "2019-09-11",
            "trxTime": "11:11:11",
            "customerId": 222,
            "version": 1,
            "createdAt": "2025-08-06T18:04:46",
            "updatedAt": "2025-08-06T18:04:46"
        },
        {
            "id": 12,
            "accountNumber": "8872838283",
            "trxAmount": 125553.00,
            "description": "FUND TRANSFER",
            "trxDate": "2019-09-11",
            "trxTime": "11:11:11",
            "customerId": 222,
            "version": 1,
            "createdAt": "2025-08-06T18:04:46",
            "updatedAt": "2025-08-06T18:04:46"
        },
        {
            "id": 13,
            "accountNumber": "8872838283",
            "trxAmount": 126663.00,
            "description": "FUND TRANSFER",
            "trxDate": "2019-09-11",
            "trxTime": "11:11:11",
            "customerId": 222,
            "version": 1,
            "createdAt": "2025-08-06T18:04:46",
            "updatedAt": "2025-08-06T18:04:46"
        }
    ],
    "pageable": {
        "pageNumber": 0,
        "pageSize": 10,
        "sort": {
            "empty": true,
            "sorted": false,
            "unsorted": true
        },
        "offset": 0,
        "paged": true,
        "unpaged": false
    },
    "last": false,
    "totalPages": 2,
    "totalElements": 11,
    "size": 10,
    "number": 0,
    "sort": {
        "empty": true,
        "sorted": false,
        "unsorted": true
    },
    "first": true,
    "numberOfElements": 10,
    "empty": false
}
```

4.2 Update Customer Description

Description:
Updates the description field of the customer record associated with the specified customerId.

Endpoint: api/customers/update/{id}/
Method: POST
Sample request payload:

```
{
  "description": "FUND TRANSFER 1",
  "version": 1
}
```

Sample response:

```
{
    "id": 2,
    "accountNumber": "8872838283",
    "trxAmount": 1123.00,
    "description": "FUND TRANSFER 1",
    "trxDate": "2019-09-11",
    "trxTime": "11:11:11",
    "customerId": 222,
    "version": 2,
    "createdAt": "2025-08-06T18:04:46",
    "updatedAt": "2025-08-07T09:52:13.5508823"
}
```

To handle concurrent updates, the Customer entity uses optimistic locking via a version field. This ensures that simultaneous modifications to the same record do not overwrite each other. If a conflict is detected (i.e., if the version in the database does not match the expected version), the update will fail, preventing data inconsistency.

7. Authentication and Security

The REST APIs in this application are secured using JWT (JSON Web Token) based authentication.

7.1 Login

Description:
Authenticates the user and returns a JWT token. The current implementation uses hardcoded credentials for demonstration purposes:

Endpoint: /api/login
Method: POST

Sample payload request

```
{
  "username": "admin",
  "password": "123456"
}
```

Sample response:

```
{
  "token":
"eyJhbGciOiJIUzI1NiJ9.eyJyb2xlcyI6WyJST0xFX0FETUlOIl0sInN1YiI6ImFkbWluIiwiaWF0IjoxNzU0NTMxMzEzLCJleHAiOjE3NTQ1MzQ5MTN9.Jg8qBMxsNYqMlQ83Ff51UNYAGGQz7i3X9PCaHM4AQW8"
}
```

7.2 Accessing Protected Endpoints

All protected API requests must include the JWT token in the Authorization header:

Header Example:
Authorization: Bearer <JWT_TOKEN>

Without a valid token, the API will reject access with an appropriate HTTP status (typically 401 Unauthorized or 403 Forbidden).

7.3 Token Validation

The JWT token is validated on each request using Spring Security filters.
If the token is expired, invalid, or missing, access to secured endpoints is denied.

6. Database Schema

The application uses a single table named customer under the assement schema. This table is used to store the transactional data parsed from the batch input file. The sql script provided in project file path src\main\resources\table-mysql.sql

6.1 Table: customer

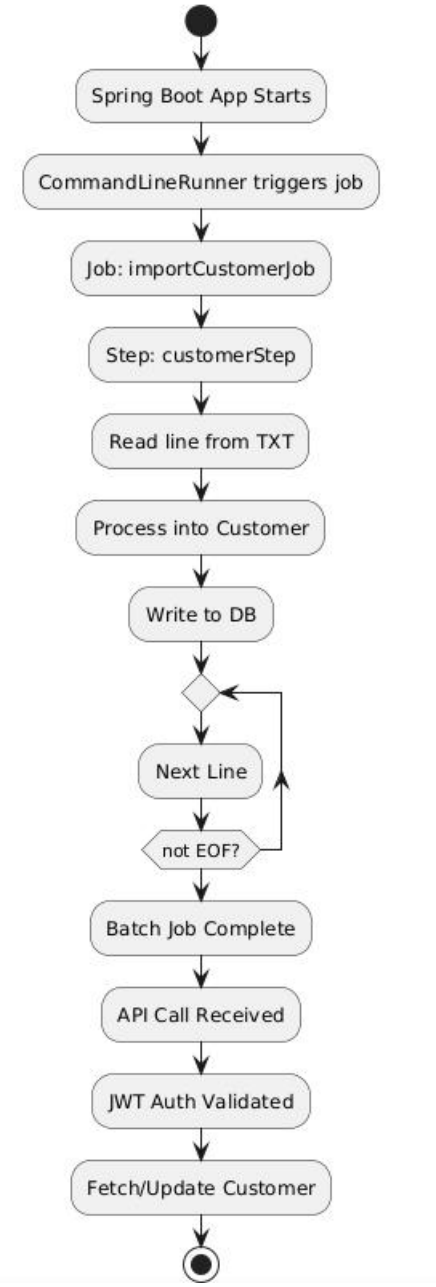
Column Name	Type	Description
id	bigint	Primary key. Auto-incremented unique identifier for each row.
customer_id	bigint	Represents the ID of the customer from the input file.
account_number	varchar(100)	Customer's account number.
description	text	A free-text field describing the transaction or customer details.
created_at	timestamp	The timestamp when the record was first inserted. Defaults to current time.
updated_at	timestamp	The timestamp when the record was last updated. Automatically updated.
trx_amount	decimal(38,2)	The transaction amount.
trx_date	date	The date when the transaction occurred.
trx_time	time(6)	The exact time of the transaction.
version	int	A fixed integer version, defaulted to 1.

6. Error Handling

- Blank or malformed lines in the input file are safely skipped.
- The job is configured to tolerate faults using skip and skipLimit settings in the step configuration.
- Errors during parsing are logged and skipped, ensuring that valid data continues to be processed.

7. **Diagrams**

7.1 Activity Diagram



7.2 Class Diagram