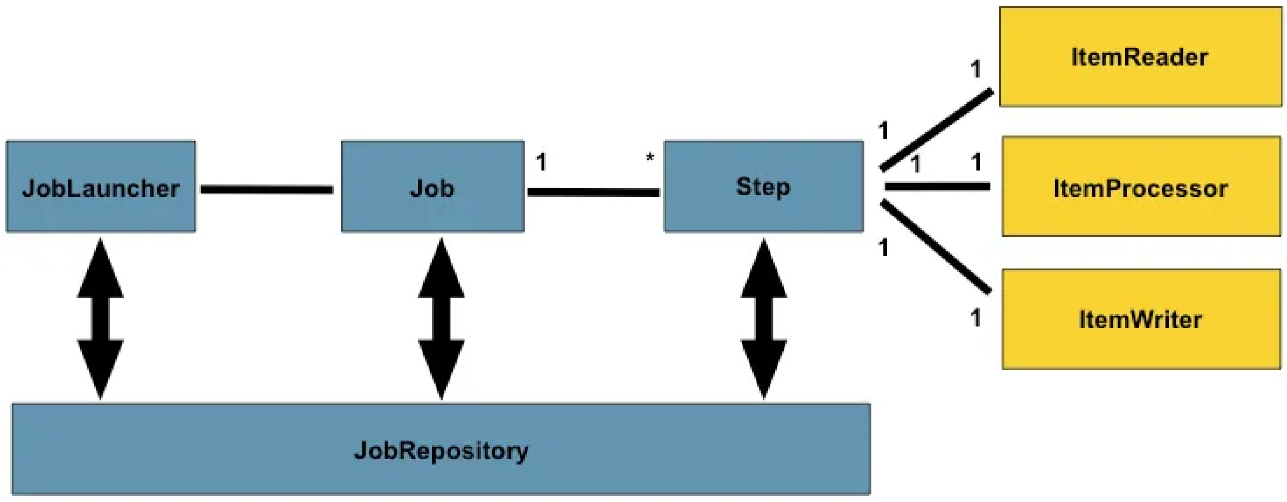
Assignment 1- Description

Batch processing is a common requirement in many business applications where large volumes of data need to be processed periodically or on demand. Batch processing involves reading data from various sources, processing it in a batch, and writing the processed data to other destinations. This type of processing is different from real-time or online processing where data is processed continuously and results are immediately available.



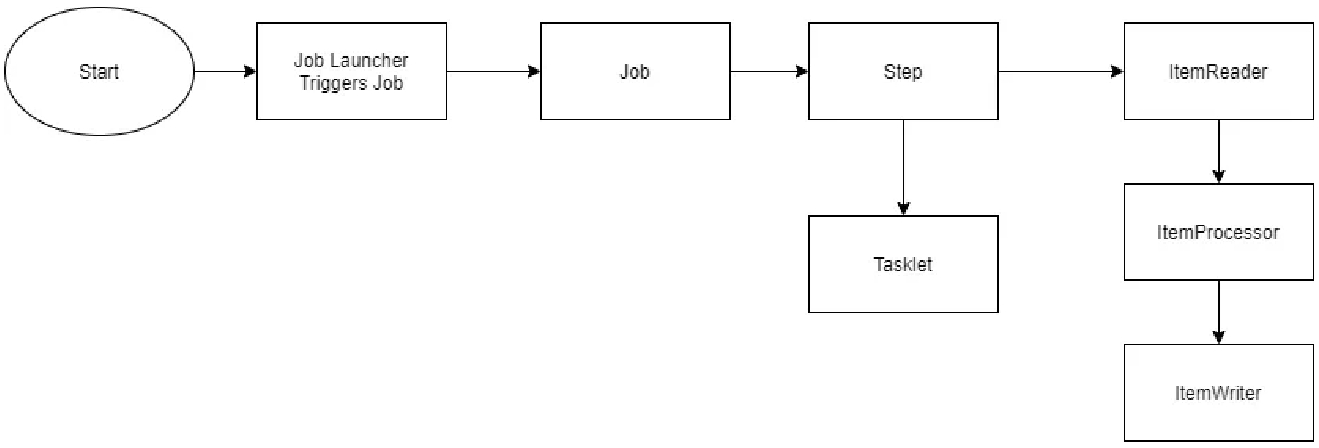
Spring Batch provides a wide range of features that make it easy to develop and manage batch processing applications, such as:

- Job execution and scheduling: Spring Batch provides a job launcher that can execute batch jobs on demand or on a predefined schedule.
- Parallel processing: Spring Batch supports parallel processing of batch jobs, allowing you to process large volumes of data in a shorter amount of time.
- Chunk-based processing: Spring Batch allows you to process data in chunks, which means reading a large

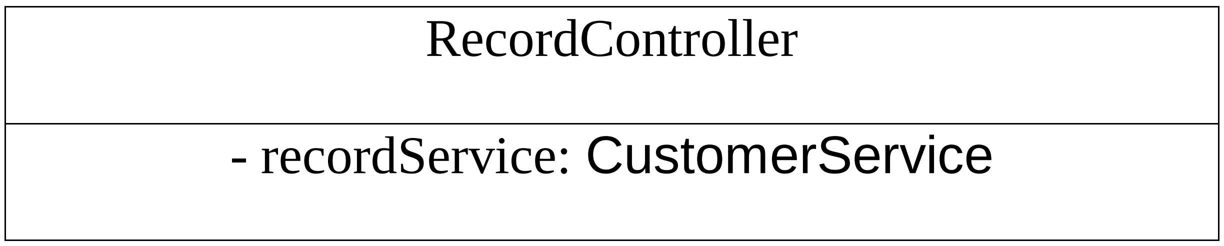
volume of data in smaller chunks and processing them one by one.

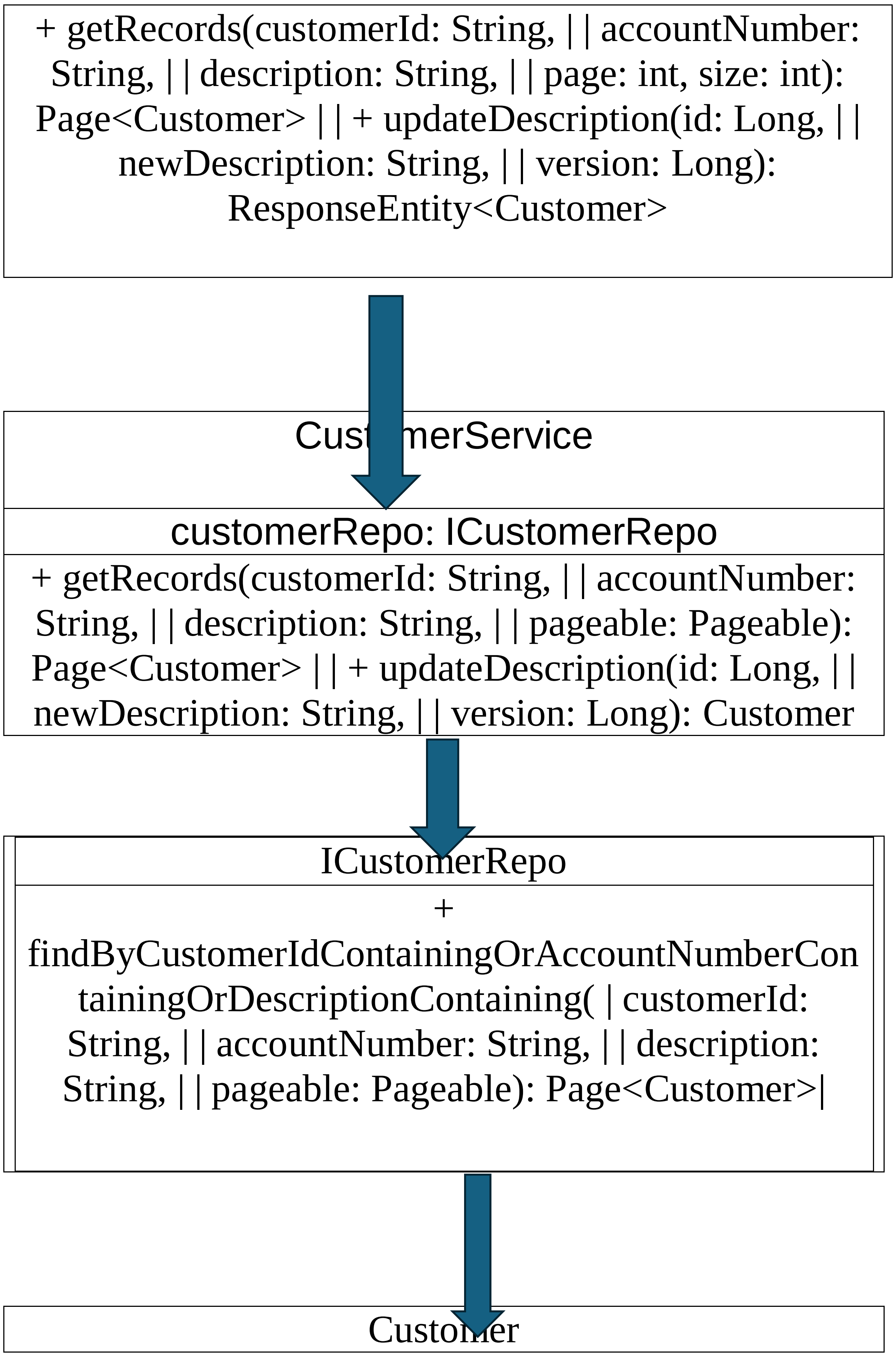
- Item readers and writers: Spring Batch provides a set of predefined item readers and writers that make it easy to read and write data from various sources and destinations.
- Transaction management: Spring Batch provides transaction management features that allow you to ensure data integrity and consistency during batch processing.
- Restartability: Spring Batch supports job restartability, allowing you to restart a failed or interrupted job from the point of failure.

Component of Spring Batch: Step, Job, Tasklet Job Launcher.



Second Assignment.





+ getRecords(customerId: String, | | accountNumber: String, | | description: String, | | page: int, size: int): Page<Customer> | | + updateDescription(id: Long, | | newDescription: String, | | version: Long): ResponseEntity<Customer>

CustomerService

customerRepo: ICustomerRepo

+ getRecords(customerId: String, | | accountNumber: String, | | description: String, | | pageable: Pageable): Page<Customer> | | + updateDescription(id: Long, | | newDescription: String, | | version: Long): Customer

ICustomerRepo

+ findByCustomerIdContainingOrAccountNumberContainingOrDescriptionContaining(| customerId: String, | | accountNumber: String, | | description: String, | | pageable: Pageable): Page<Customer>|

Customer

- id: Long | | - customerId: String | | - accountNumber: String | | - description: String | | - version: Long (Optimistic Locking)

SecurityConfig

InMemoryUserDetailsManager
+ securityFilterChain(http: HttpSecurity): SecurityFilterChain |

RecordController: The REST controller responsible for handling the API requests. It uses RecordService to process the business logic.

- Methods:
    - getRecords(...): Handles the retrieval of records with pagination and search.
    - updateDescription(...): Handles the update of a record’s description with optimistic locking.

CustomerService: The service layer that interacts with the repository and contains the business logic.

- Methods:
    - getRecords(...): Retrieves records from the repository based on search criteria and pagination.
    - updateDescription(...): Updates a record’s description with optimistic locking to handle concurrency.

ICustomerRepo: The repository interface extends JpaRepository and is responsible for querying the database.

- Method:
- findByCustomerIdContainingOrAccountNumberContainingOrDescriptionContaining(...): A custom query method for searching records by customerId, accountNumber, or description with pagination.

Customer: The entity class representing the record in the database, with fields for id, customerId, accountNumber, description, and version (used for optimistic locking).

- Attributes:
  - id: The unique identifier of the record.
  - customerId, accountNumber: Fields used for searching records.
  - description: The description of the record.
  - version: The version field used for optimistic locking.

SecurityConfig: The Spring Security configuration class. It defines security rules using SecurityFilterChain, such as restricting access to the API endpoints and configuring basic authentication with roles.

- Methods:
  - securityFilterChain(...): Configures the HTTP security rules (authentication and authorization).

Get Request

GET
http://localhost:8080/api/records?customerId=123&page=0&size=10

Post Request
PUT http://localhost:8080/api/records/1
Content-Type: application/json
If-Match: 1 (Version from the Record entity)
Body: "Updated Description"