1. Batch Job

* Pattern: Factory Pattern
* Can handle multiple file formats, use the factory pattern to instantiate different transaction reader (CsvTransactionReader, TxtTransactionReader)
* Pattern: Observer/Listener Pattern
* TransactionJobListener implements JobExecutionListener. It observes the job execution and writes a summary after the job finished.
* It “listens” for events (beforeJob and afterJob) and reacts.
* Pattern: Builder Pattern
* Creating Transaction objects is repetitive. Using a builder makes your code cleaner and more readable.
* @Builder on Transaction class

1. RESTful API (Backend)

* Pattern: Repository Pattern
* Abstacts data access logic, and enabling service layer to interact with database in a clean and flexible way. As well as transaction management.
* Pattern: Optimistic Locking (for concurrent updates)
* Optimistic locking ensures that updates to records prevent overwrite each other by checking version numbers before committing an update. This is useful when handling concurrent updates in RESTful API.
* Pattern: Pagination Strategy
* Using JPA build in pagination to handle large dataset. Prevent overload heap memory.
* Pattern: MVC (Model-View-Controller)
* A standard architectural design pattern to separate concerns between the API routes (controllers), the business logic (services), and data models (entities). It keeps your code modular and easier to maintain.