University of Jijel
Faculty of Exact Sciences and Computer Sciences
Departement of Computer Sciences
Master SIAD - Advanced Software Engineering

Advanced Software Engineering: final project

The "Hanouti" Management System

Project Description

A rapidly growing local retail chain requires a backend system to manage its warehouse operations. Currently, they are using Excel sheets, which has led to data redundancy and errors. They have hired you to build a robust Java application to digitize their inventory.

Functional Requirements

The core of the business revolves around three main entities: the Items sold in the store, the Sections where items are displayed, and the Vendors who supply these items.

- Inventory Organization: Every item in the warehouse must be assigned to a specific section (e.g., Electronics, Groceries, Clothing) to ensure staff can find them easily. An item cannot exist in the system without belonging to a section.
- Supply Chain: The business purchases items from external vendors. It is common for the store to source the exact same item from different vendors depending on availability or seasonal pricing. The system must be able to track which vendors supply which items.
- Item Details: Each item has a unique SKU (Stock Keeping Unit), a display name, and a current stock level.
- Vendor Details: Vendors are identified by a license number and a contact name.
- Section Details: Sections have a unique code and a descriptive label.

Operational Scenarios:

- The manager needs to view all items available in a specific section.
- The procurement officer needs to see a list of all possible vendors for a specific item to compare sourcing options.
- Conversely, the officer needs to see a catalog of all items a specific vendor is capable of supplying.

Project Goals

The objective of this project is not just to make the application "work," but to make it maintainable, scalable, and architecturally sound. Students are expected to demonstrate mastery of the following areas:

- Object-Oriented Analysis: Translating the business reality described above into a correct Class Diagram, specifically inferring the correct multiplicity and directionality of associations.
- Architectural Structuring: Implementing the MVC (Model-View-Controller) pattern to separate business logic, data presentation, and user interaction.
- Refactoring & Clean Code: applying SOLID principles to ensure classes are not tightly coupled and have single responsibilities.
- Design Pattern Application: Identifying problems in the code that can be solved using standard Design Patterns (Creational, Structural, or Behavioral).
- Persistence Layer: Implementing JPA (Java Persistence API) to map the object model to a relational database, handling the specific challenges of the relationships described.

Evaluation Criteria

Students will be graded based on the technical depth of their solution and their adherence to software engineering best practices. The evaluation will cover:

Modeling & OOP (20%)

- Association Mapping: Correct implementation of the relationship between Item and Section (students must determine if this is aggregation or composition and implement it via Java fields).
- Complex Relationships: Correct handling of the relationship between Item and Vendor. Students must implement the underlying collection types (Lists, Sets) correctly to reflect the business logic.
- Encapsulation: Proper use of access modifiers and accessors.

Software Architecture & Principles (30%)

- MVC Implementation: Clear separation of packages (e.g., models, controllers, views, services). Code for UI (Console or GUI) must not be mixed with business logic.
- SOLID Adherence:
  - Single Responsibility: Does the Item class handle database saving?
  - Open/Closed: Can we add a new type of Item (e.g., PerishableItem) without modifying existing logic?
  - Dependency Inversion: Are high-level modules depending on abstractions (Interfaces) rather than concrete implementations?

Design Patterns (30%)

- Required Patterns: Implementation of at least three distinct patterns.

ORM & JPA Implementation (20%)

- Annotation Usage: Correct use of @Entity, @Id, @OneToMany, @ManyToOne, and specifically the @ManyToMany annotation (or the manual creation of a join entity if they choose to model the association explicitly).
- Fetch Types: Justification of EAGER vs LAZY loading for the Item-Vendor relationship.
- DAO Pattern: Implementation of Data Access Objects to abstract the EntityManager logic away from the business service layer.

Submission Guidelines

- Team Structure: This project must be carried out in teams of maximum two students. Individual submissions are also accepted.
- Deadline: The final deadline for the project is Wednesday, December 31, 2025. Late submissions will not be graded.
- Delivery Method: Please zip your source code (NetBeans project structure preferred) along with a short PDF report explaining your architectural choices. Send the files via email to t_boutefara@esi.dz.