Labo 1 – Architecture 2 tiers (Client/Serveur), Persistance

Cours : **Architecture Logicielle (LOG430)**

Session : **Été 2025**

Date du laboratoire : **Semaine du 19 mai 2025**

Le génie pour l'industrie

1 Contexte

Dans le cadre du cours, ce laboratoire a pour objectif de consolider les concepts fondamentaux liés aux architectures logicielles simples. Vous serez amené à concevoir et développer une application de type **client/serveur à deux niveaux (2-tier)**, dans laquelle le client est une application console locale qui interagit directement avec une base de données locale (le "serveur").

L'application cible est un **système de caisse pour un petit magasin de quartier**, simple, robuste et autonome, qui pourra évoluer dans les futurs laboratoires (ex. : gestion multi-succursales, puis e-commerce distribué).

2 Objectifs d'apprentissage

À la fin de ce laboratoire, vous serez capable de :

— Concevoir une architecture client/serveur à deux couches (2-tier)

— Mettre en œuvre une couche de persistance abstraite (via un Object-Relational Mapping (ORM), par exemple)

— Justifier vos choix d'architecture à l'aide de documents structurés (diagrammes et ADR)

— Évaluer les performances d'un système à travers des tests de charge ciblés.

Remarque importante : Vous devez poursuivre les bonnes pratiques CI/CD mises en place dans le Laboratoire 0 : exécution des tests unitaires, vérification de la qualité du code, et build de l'image Docker avec publication sur Docker Hub.

3 Pré-requis

Avant de commencer, assurez-vous de disposer des compétences suivantes :

— Programmation orientée objet et/ou fonctionnelle

— Être capabl d'appliquer le concept de l'abstraction de la persistance (par exemple, en utilisation d'un ORM comme Diesel, GORM, SQLAlchemy, Hibernate, etc)

— Manipulation d'une base de données relationnelle ou NoSQL (ex. : SQLite, PostgreSQL, MongoDB)

— Concepts de base de l'architecture client/serveur à deux niveaux (2-tier)

— Notions de modélisation UML (cas d'utilisation, classes, séquences)

— Utilisation d'un langage de programmation de votre choix, tel que Java, Rust, Go, Python ou Node.js

4 Contexte pédagogique

Dans le cadre de ce premier laboratoire, vous serez amené à concevoir et développer une application logicielle respectant les principes suivants :

— Architecture **Client/Serveur à deux niveaux (2-tier)**

— Persistance via **RDBMS ou NoSQL**

— L'abstraction de la couche de persistance (suggestion : utilisez un **ORM**)

Vous développerez une solution complète pour un système de point de vente (POS).

5 Partie 1 – Analyse et Conception

Avant de débuter l'implémentation, vous devez analyser les besoins du système et concevoir une architecture adaptée. Cette section sera complétée sous un répertoire **docs/ de votre dépot de projet**.

5.1 Tâches à réaliser

1. **Analyse des besoins** fonctionnels et non-fonctionnels du système.
2. **Proposition d'architecture** sous forme de vues UML selon le modèle **4+1** :
   — Vue logique : diagramme de classes représentant les entités principales (produits, ventes, utilisateurs, etc.)
   — Vue des processus : diagrammes de séquence illustrant les interactions entre l'utilisateur console et la base de données
   — Vue de déploiement : architecture locale illustrant le modèle 2-tier (application console + base de données)
   — Vue d'implémentation : organisation des modules (présentation, logique métier, persistance)
   — Vue des cas d'utilisation : ex. : vente de produit, retour, consultation du stock
3. **Justification des décisions d'architecture (ADR)** : rédigez Architecture Decision Records sur les sujets suivants (au moins 2 ADRs) :
   — Choix de la platform
   — Séparation des responsabilités entre présentation, logique et persistance
   — La stratégie de persistence
   — Choix de mécanisme de base de données (SQL vs NoSQL, local vs serveur)
4. **Choix technologiques** : listez et justifiez les outils et bibliothèques choisis selon les contraintes du projet (simplicité, coût, portabilité, fiabilité...).

6 Partie 2 – Implémentation Technique

Vous devez concevoir et développer une application **console ou desktop simple** permettant de gérer les opérations de base d'un système de caisse pour un petit magasin de quartier

avec 3 caisses qui travail simultanément (avoir des transactions pour garantir la consistance). Continuez à appliquer les bonnes pratiques de CI/CD pour ce lab 1 : vérifications automatiques, tests, et construction reproductible.

6.1 Client (console)

Le cliente est une application console/desktop qui permet à un employé du magasin d'effectuer, par exemple, les opérations suivantes :

— Rechercher un produit (par identifiant, nom ou catégorie)

— Enregistrer une vente (sélection de produits et calcul du total)

— Gérer les retours (annuler une vente)

— Consulter l'état du stock des produits

6.2 Serveur (base de données)

Dans ce laboratoire, le terme **"serveur"** désigne uniquement la base de données (SQLite, PostgreSQL, MongoDB...) sur votre machine virtuelle fourni.

L'application console accède directement à la base de données via une couche de persistance abstraite. Aucun serveur HTTP ou API REST n'est requis dans ce laboratoire.

La base doit permettre :

— La gestion des transactions (ex : enregistrement d'un achat)

— La fiabilité des écritures (ex : cohérence du stock)

7 Livrables attendus

L'ensemble des éléments suivants doit être remis dans votre dépôt GitHub ou GitLab, de façon organisée.

1. **Code source** complet de l'application (client console + modules internes), accompagné de :
   — un `Dockerfile` pour conteneuriser l'application,
   — un `docker-compose.yml` pour orchestrer l'exécution avec une base de données,
   — les instructions d'exécution/test dans un `README.md`.
2. **Documentation technique**, structurée dans un dossier `docs/` à la racine du projet, comprenant :
   — un fichier Markdown `README.md` décrivant :
      — les instructions d'exécution, de test, de compilation, et d'utilisation de l'application,
      — les choix technologiques retenus avec leur justification.
   — vos décisions d'architecture sous forme d'ADRs (`docs/ADR/`),
   — les diagrammes UML du système selon les vues du modèle 4+1 (`docs/UML/`),