Exploitation base de données

Sujet : Système de gestion de stock pour un petit commerce

---

1. Contexte

Vous êtes stagiaire dans une petite entreprise spécialisé dans la vente de maquettes d'avion en papier qui souhaite numériser la gestion de son stock et de ses commandes. Malheureusement, vous n'avez pas de référent technique sur place : l'équipe est réduite et aucune personne n'est dédiée à l'informatique.

L'outil envisagé doit permettre :

1. De gérer un catalogue de produits (références, prix unitaire, quantité en stock…).

2. De classer les produits par catégories.

3. De lier chaque produit à un fournisseur pour identifier son origine (un produit peut avoir plusieurs fournisseurs).

4. De gérer des commandes clients et leurs lignes (produits commandés, quantités, prix unitaire appliqué, etc.).

5. De répertorier les clients qui passent commande (nom, adresse, coordonnées…).

Pour l'instant, le patron de l'entreprise veut que tout le monde dans la boutique puisse accéder à ce système, dans un réseau interne.

La première version (V1) sera donc un prototype (POC) :

Peu ou pas de validation des données, pas de protection spécifique contre les injections SQL…

L'objectif est de démontrer qu'on peut gérer le stock et les commandes de manière basique.

Une fois la V1 terminée, vous ferez un audit pour repérer les failles (ex. injections SQL, manque de validation). Puis, vous réaliserez une V2 prenant en compte les pistes de l'audit.

Lors d’une fête de famille, votre cousin (qui se dit Tech Lead) vous a vaguement soufflé l’idée d’une base de données avec 6 tables principales, et d’une API en JavaScript (Node.js) pour gérer les données. Selon lui, le front pourra être développé plus tard. Au boulot!

---

2. Livrables et étapes

A. Schéma de la base de données

Vous devez :

1. Modéliser la base (type MCD/MLD), en identifiant les tables, les liens entre elles, le clés primaires, clés étrangères, et types de champs adaptés.

2. Prévoir juste ce qu’il faut pour couvrir la gestion du stock, des fournisseurs, des commandes, etc. (n’ajoutez pas de tables bonus si elles ne sont pas essentielles au POC).

B. Scripts SQL

1. Script de création du schéma de la base de données.

2. Script d’insertion de données de test : Bonne quantité de données dans toutes les tables.

C. Développement de l’API JS (V1)

Vous développerez une API (en Node.js + Express, par exemple) qui :

Permet les opérations CRUD (Créer, Lire, Mettre à jour, Supprimer) sur les tables (Produits, Catégories, Fournisseurs, Commandes, Lignes_Commande, Clients).

Gère listing simples : par exemple, lister les commandes avec leurs lignes, tous les clients et leurs commandes…

!Ne comporte pas d’authentification (POC interne).!

Utilise des requêtes SQL directes (ex. via un driver MySQL), sans protection particulière contre l’injection (pour l’instant).

D. Audit de la V1

Une fois la V1 terminée, vous analysez le code et la base de données pour mettre en évidence :

1. Les failles de sécurité

Potentielle injection SQL (concaténation de chaînes pour construire la requête),

Manque de validation sur les champs (quantités négatives, champs vides, etc.).

Vous serez ainsi en mesure d'expliquer à votre patron que vous avez appris énormément, et que la qualité de la V2 que vous allez fournir mériterait une prime.

2. Les améliorations possibles

Requêtes paramétrées ou procédures stockées pour éviter l'injection SQL,

Vérifications métier (contrôler la cohérence du stock, empêcher de commander un produit absent du catalogue, etc.).

Ça devrait lui faire peur.

E. Développement de la V2

Vous implémentez la version 2 en :

1. Corrigeant les failles et incohérences identifiées (requêtes paramétrées, validation des données, etc.).

2. Renforçant la logique métier (contrôle du stock, etc.).

3. (Optionnel) Ajoutant une gestion de rôles ou d'authentification (admin, user…) si vous le jugez nécessaire. (Optionnel)
(Après tout, on ne vous l'a pas demandé)

<u>Fonctions avancées</u>

Pour démontrer l’intérêt de votre API et enrichir la V2, vous implémenterez aussi quelques fonctionnalités avancées :

1. Lister les commandes par année

Ex. GET /commandes?start=2023-01-01&end=2023-12-31.

2. Rechercher les commandes d’un client

Ex. GET /clients/:id/commandes.

3. Lister les commandes qui contiennent un article précis

Ex. GET /produits/:id/commandes.

4. Recherche multi-critères (client, date, statut, produit…)

5. Statistiques simples (produits les plus vendus, total des ventes sur une période…)

6. Gestion fine du stock (décrémentation automatique, blocage si insuffisant…)

7. Notifications de stock faible (ex. GET /produits/stock-faible?seuil=10).

F. Livrable final

Rédigez un document de synthèse :

Présentation du schéma de la base de données (entités, relations, types de champs),

Liste des endpoints de l’API (routes, paramètres, retours JSON), avec des exemples d’appels,

Résumé de l’audit V1, des problèmes détectés et des solutions adoptées en V2 (ex. requêtes paramétrées, contrôle des champs…),

Publiez le code source sur GitHub avec un README expliquant l’installation et un historique de commits (V1 → Audit → V2).

G. (facultatif) Options techniques, axes d’amélioration (facultatif)

Pour aller plus loin, vous pouvez :

1. Utiliser un ORM (Sequelize, TypeORM…) pour automatiser la gestion des tables et des requêtes,

2. Mettre en place de la sécurité (authentification via sessions/cookies ou JWT),

3. Gérer des rôles de façon plus fine (admin, user…),

4. Préparer un frontend (React, Vue, etc.) – mais ce n’est pas obligatoire à ce stade.