**GESTIONNAIRE INTELLIGENT DE STOCKS (TABLEAU DYNAMIQUE + ARBRE AVL + FILE PRIORITAIRE)**

# Introduction

Le projet « Gestionnaire Intelligent de Stocks » vise à développer une application de gestion de produits intégrant trois structures de données avancées :

* **Tableau Dynamique** pour une gestion flexible du stock,
* **Arbre AVL** pour une recherche rapide et un accès optimisé aux produits,
* **File Prioritaire** pour identifier et traiter en priorité les produits en rupture de stock ou sous le seuil critique.

L’objectif est de fournir aux utilisateurs (par exemple, un gestionnaire de stock dans une entreprise) un outil efficace pour surveiller, mettre à jour et réapprovisionner leur inventaire.

# 2. Objectifs du Projet

* **Optimisation des performances :** Utiliser des structures de données avancées afin d’accélérer la recherche, l’insertion et la suppression des produits.
* **Gestion dynamique du stock :** Permettre l’ajustement automatique de la taille du tableau de produits selon les besoins.
* **Traitement prioritaire :** Gérer de manière efficace les produits en pénurie grâce à une file prioritaire, garantissant ainsi un réapprovisionnement en temps utile

# 3. Architecture Générale du Projet

**Modules principaux :**

* **Tableau Dynamique (Dynamic Array) :**
  + Stocke l’ensemble des produits.
  + Permet l’ajout, la modification, la suppression et l’affichage.
  + Bien que Python propose déjà des listes dynamiques, vous pouvez créer votre propre classe pour maîtriser la logique d'extension/réduction.
* **Arbre AVL :**
  + Organise les produits pour assurer une recherche rapide (par identifiant ou nom).
  + Implemente automatiquement l’équilibrage pour garantir une complexité en O(log n) pour les opérations de recherche, insertion et suppression.
  + Fournit des méthodes pour rechercher, mettre à jour et supprimer un produit.
* **File Prioritaire :**
  + Utilisée pour gérer les produits en rupture (quantité < seuil critique).
  + Implémentable par un tas binaire (ou, en bonus, un tas de Fibonacci).
  + Classe les produits en fonction de l'urgence (les stocks les plus bas ayant la priorité la plus haute).

# 4. Fonctionnalités Attendues

## 3.1. Gestion du Stock (Tableau Dynamique)

* **Stockage des produits :** Implémenter un tableau dynamique qui peut évoluer (agrandissement/rétrécissement) en fonction du nombre de produits.
* **Attributs des produits :** Chaque produit sera défini par un identifiant unique, un nom, une quantité en stock, un prix et une date d’expiration.
* **Méthodes de gestion :**
  + **Ajout d’un produit :** Insérer un nouveau produit dans le tableau.
  + **Modification d’un produit :** Mettre à jour les informations (quantité, prix, date, etc.).
  + **Suppression d’un produit :** Retirer un produit du tableau.
  + **Affichage du stock :** Lister l’ensemble des produits présents dans le stock.

**Exemple de classe en Python :**

class Produit:

Points notables :

* **Vérifications des valeurs d'entrée** pour éviter les incohérences (ID négatif, prix négatif, etc.).
* **Encapsulation des validations** via des méthodes statiques.
* **Ajout de la méthode est\_expire()** pour détecter les produits périmés.

class DynamicArray:

Points remarquable et avantages :

### **Optimisation avec un dictionnaire (self.produits)** :

* + Accès plus rapide aux produits par ID.
  + Évite de parcourir toute la liste pour chercher/modifier/supprimer un produit.

### **Contrôles d'entrée et validation** :

* + Vérifie que l’objet ajouté est bien un Produit.
  + Empêche l’ajout de produits avec le même ID.
  + Empêche la modification/suppression d’un produit inexistant.

### **Réutilisation des méthodes de validation de Produit** :

* + Assure la cohérence des données avec \_valider\_nom, \_valider\_quantite, \_valider\_prix, \_valider\_date.

### **Méthode trouver\_produit(id\_produit)** :

* + Permet de récupérer un produit sans erreur si l’ID n’existe pas.
  + Utile pour d'autres fonctionnalités.

## 3.2. Recherche et Organisation (Arbre AVL)

* **Stockage dans l’arbre AVL :** Organiser les produits en fonction de leur identifiant unique ou nom pour une recherche optimisée.
* **Opérations sur l’arbre AVL :**
  + **Recherche d’un produit :** Retrouver rapidement un produit par son identifiant ou son nom.
  + **Mise à jour d’un produit :** Modifier les informations d’un produit dans l’arbre tout en préservant l’équilibre.
  + **Suppression d’un produit :** Supprimer un produit et rééquilibrer l’arbre automatiquement.

Remarques :

✔ **Gestion des erreurs** : Vérification des types et levée d’exceptions.  
✔ **Méthodes de suppression et mise à jour** : Maintien de l’équilibre après modification.  
✔ **Affichage en ordre croissant d’ID** pour une meilleure lisibilité.

## 3.3. Gestion des Produits en Rupture (File Prioritaire) « class PriorityQueue: »

* **File de priorité :** Intégrer une file de priorité (implémentée par un tas binaire ou tas de Fibonacci) pour gérer les produits dont la quantité est inférieure à un seuil critique.
* **Opérations sur la file :**
  + **Insertion en cas de pénurie :** Ajouter un produit à la file lorsque son stock devient critique.
  + **Extraction prioritaire :** Extraire les produits ayant le niveau de stock le plus bas pour déclencher un réapprovisionnement immédiat.

## ****Explication des améliorations**** :

### **1 Validation des entrées**

🔹 Vérification que produit est bien un **objet Produit**.  
🔹 Vérification que seuil\_critique est bien un **nombre positif**.

### **2 Gestion des erreurs**

🔹 Évite les erreurs de heapq en s’assurant que seuls des objets valides sont ajoutés.  
🔹 Empêche l’extraction d’un élément si la file est vide.

### **3 Ajout d'une flexibilité sur la priorité**

🔹 Permet d’utiliser une fonction personnalisée priorite\_personnalisee pour adapter la logique de priorité.

* Exemple : Prioriser les produits par **date d'expiration** au lieu du stock.

### **4 Optimisation et nouvelles fonctionnalités**

🔹 **Méthode est\_vide()** : Vérifie si la file est vide avant d’essayer d’extraire.  
🔹 **Méthode taille()** : Retourne le nombre d’éléments dans la file.  
🔹 **Affichage trié** : Affiche les produits dans l’ordre de priorité.

# 5 Intégration et Interface (CLI) «main.py »

L’interface en ligne de commande pourra proposer des menus pour :

* Ajouter, modifier, supprimer un produit (qui interagira avec le tableau dynamique et mettra à jour l’arbre AVL en conséquence).
* Rechercher un produit via l’arbre AVL.
* Vérifier la file prioritaire et extraire les produits les plus urgents pour réapprovisionnement.
* Afficher l’ensemble des produits du stock.

Explications :

# ****6 Explication et améliorations**** :

## ****6.1 Vérification et validation des entrées utilisateur****

✅ **ID du produit** : Doit être un nombre entier positif.  
✅ **Nom du produit** : Enlève les espaces inutiles.  
✅ **Quantité** : Doit être un entier positif.  
✅ **Prix** : Doit être un nombre positif.  
✅ **Date d'expiration** : Vérifie le format YYYY-MM-DD avec datetime.strptime().

## ****6.2 Meilleure gestion des erreurs****

✅ **Empêche d'ajouter un produit avec un ID déjà existant** (évite les doublons).  
✅ **Empêche de modifier ou supprimer un produit inexistant**.  
✅ **Empêche les entrées invalides pour éviter les crashs**.

## ****6.3 Mise à jour correcte dans toutes les structures****

✅ **Lorsqu’un produit est modifié**, il est mis à jour dans :

* DynamicArray
* AVLTree
* PriorityQueue ✅ **Lorsqu’un produit est supprimé**, il est retiré de :
* DynamicArray
* AVLTree
* (*PriorityQueue* est mis à jour automatiquement lors de l'extraction)

## ****6.4 Meilleure expérience utilisateur****

✅ **Affichage plus structuré** avec des messages explicites.  
✅ **Indications claires sur les erreurs** pour éviter les saisies invalides.  
✅ **Ajout d’icônes (✅, ❌, 👋) pour une meilleure lisibilité**.

# 7 Architecture et Choix Technologiques

## 7.1. Structure Modulaire

Le projet sera organisé de manière modulaire afin de faciliter la maintenance et les évolutions futures. Chaque module correspondra à une des fonctionnalités principales :

* **Module Stock :** Gestion du tableau dynamique et des opérations associées.
* **Module Recherche :** Implémentation de l’arbre AVL pour la recherche et l’organisation.
* **Module Priorisation :** Gestion de la file prioritaire pour le traitement des produits en rupture.

## 7.2. Langage de Programmation

Le projet est implémenté en **Python**. Puisque en Python, l'utilisation de classes et de modules permet de structurer efficacement le code tout en facilitant les tests unitaires.

## 7.3. Interface Utilisateur

* **Interface en Ligne de Commande (CLI) :** Fournir une interface textuelle permettant d’interagir avec l’application (ajout, suppression, modification, recherche, etc.).

# **Interface Graphique (Bonus) :** Développer une interface simple avec des bibliothèques telles que Tkinter (pour Python) afin d’améliorer l’expérience utilisateur.

# 8. Conclusion

Ce projet permet de mettre en œuvre plusieurs structures de données avancées pour répondre à un besoin concret de gestion de stocks. La combinaison d’un tableau dynamique, d’un arbre AVL et d’une file prioritaire offre une solution robuste, efficace et modulable pour une gestion intelligente des inventaires. Le respect de la modularité et la versioning via GitHub garantissent une maintenance aisée et des évolutions futures possibles.