1. **Introduction**

Dans le cadre de notre formation *Python – Niveau Intermédiaire*, il nous a été demandé de réaliser un projet final complet afin de mettre en pratique l’ensemble des connaissances acquises durant cette formation. Après avoir étudié les bases de la programmation orientée objet, la manipulation des fichiers et des données, la mise en place de tests unitaires ainsi que les bonnes pratiques de versionnement avec Git et GitHub, nous avons choisi de travailler sur un thème concret et proche des besoins des entreprises : **la gestion d’inventaire**.

Ce projet a été conçu comme une mise en situation réelle. Il s’agit non seulement de développer un programme fonctionnel en Python, mais aussi d’adopter une méthodologie complète de gestion de projet logiciel.

1. **Présentation du projet et des objectifs**

L’objectif général est de concevoir une application Python capable de gérer de manière efficace un inventaire de produits avec leurs fournisseurs et catégories. Plus précisément, on souhaite que l’application permette :

* De créer, consulter, modifier et supprimer (CRUD) des produits , des catégories et des fournisseurs ;
* De gérer les stocks et de suivre les ventes ;
* D’attribuer des identifiants uniques aux enregistrements via la bibliothèque uuid ;
* De générer des rapports d’analyse (Consultation par catégorie et fournisseur, Alerte de seuil, valorisation du stock, calcul de la marge bénéficiaire) ;
* D’exporter ces rapports en différents formats (CSV et JSON) pour faciliter leur exploitation ;
* De fournir une documentation technique et un guide utilisateur clairs pour accompagner la prise en main du logiciel.

À travers ce projet, on veut aboutir à une solution simple, automatique et évolutive qui illustre la manière dont Python peut répondre à des problématiques de gestion courantes.

1. **Contexte et enjeux de la gestion d’inventaire**

La gestion d’inventaire est un élément central dans toute entreprise manipulant des biens physiques. Sans système informatisé, la gestion manuelle des stocks entraîne souvent des erreurs, un manque de traçabilité et des décisions mal informées. Dans un environnement concurrentiel, disposer d’un outil fiable pour gérer les stocks est devenu indispensable.

En réalisant ce projet, on se place dans la peau d’une équipe de développeurs chargée de concevoir un logiciel interne pour une petite ou moyenne entreprise. L’outil doit aider à :

* **Réduire les erreurs** grâce à l’automatisation des opérations ;
* **Améliorer la traçabilité** des produits, des catégories et des fournisseurs ;
* **Gagner du temps** sur les tâches répétitives ;
* **Produire des indicateurs fiables** pour soutenir la prise de décision.

Ce projet est donc non seulement un exercice pédagogique mais aussi une illustration concrète d’un besoin professionnel réel.

1. **Méthodologie adoptée**

Pour mener à bien ce projet, nous avons suivi une démarche progressive et structurée, inspirée des principes agiles :

1. **Analyse des besoins** : définition précise du périmètre du projet et identification des fonctionnalités essentielles.
2. **Modélisation UML** : élaboration de diagrammes de classes et de cas d’utilisation pour représenter les entités principales (Produit, Fournisseur, Rapport) et leurs interactions.
3. **Architecture logicielle modulaire** : séparation claire des responsabilités au sein de modules Python distincts (produits, fournisseurs, rapports).
4. **Développement itératif** : intégration progressive des fonctionnalités, mise en place de la gestion des exceptions et adoption de la bibliothèque uuid pour la génération d’identifiants uniques.
5. **Qualité et tests** : utilisation de Pytest, coverage et pylint afin de garantir la robustesse, la maintenabilité et la conformité du code.
6. **Gestion de version et collaboration** : création d’un dépôt GitHub, réalisation de commits réguliers et suivi rigoureux des évolutions.
7. **Documentation complète** : rédaction d’un README technique, d’un guide utilisateur et d’un rapport détaillé du projet.

Cette méthodologie nous a permis de structurer efficacement le développement, de détecter rapidement les anomalies et d’aboutir à une application fonctionnelle, conforme aux bonnes pratiques.

1. **Plan du rapport**

Le présent rapport rend compte de l’ensemble de notre démarche. Il est structuré en plusieurs sections :

* **Section 1- Introduction** : présentation du projet et des objectifs , contexte et enjeux de gestion d’inventaire, méthodologie adoptée, plan du rapport .
* **Section 2 – Analyse et Conception** : description détaillée des besoins fonctionnels, modélisation des données à l’aide de diagrammes UML, architecture logicielle et technologies sélectionnées.
* **Section 3 – Implémentation** : explication de la structure du projet, présentation des fonctionnalités implémentées avec des détails techniques et extraits de code commentés.
* **Section 4 – Tests et Qualité** : présentation de la stratégie de test, des outils utilisés et de l’assurance qualité mise en place.
* **Section 5 – Résultats et Évaluation** : exposé des fonctionnalités réalisées, démonstrations des cas d’usage principaux, évaluation des performances et retour d’expérience utilisateur.
* **Section 6 – Conclusion et Perspectives** : bilan du projet, compétences acquises, pistes d’amélioration et perspectives professionnelles.

Cette organisation reflète notre démarche de travail et offre une vision complète du projet, de son analyse initiale à son implémentation finale et son évaluation.

1. **Analyse et Conception**

Cette section présente l’ensemble de la réflexion menée avant de passer au développement effectif du projet. On y expose les besoins fonctionnels détaillés, l’architecture logicielle retenue et les technologies sélectionnées. Cette étape est cruciale pour s’assurer que le système de gestion d’inventaire répondra réellement aux besoins identifiés et sera facilement maintenable.

1. **Analyse des besoins fonctionnels**

Dans le cadre de ce projet, nous avons commencé par dresser la liste des besoins fonctionnels et non fonctionnels. Cette démarche permet de cerner précisément les attentes de l’utilisateur final et les contraintes techniques.

**Besoins fonctionnels**

Le système doit permettre :

* **Gestion des produits**
  + Ajouter de nouveaux produits dans l’inventaire (nom, catégorie, prix, quantité, fournisseur associé).
  + Modifier les informations d’un produit existant.
  + Supprimer un produit de l’inventaire.
  + Consulter la liste complète des produits.
* **Gestion des catégories**
  + Ajouter de nouvelles catégories dans l’inventaire (nom, description).
  + Modifier les informations d’une catégorie existante.
  + Supprimer une catégorie de l’inventaire.
  + Consulter la liste complète des catégories.
* **Gestion des fournisseurs**
  + Ajouter de nouveaux fournisseurs (nom, téléphone, email, adresse).
  + Modifier les informations d’un fournisseur existant.
  + Supprimer un fournisseur.
  + Associer des produits à un fournisseur.
* **Gestion des rapports et analyses**
  + Générer des rapports sur les produits les plus et les moins vendus.
  + Générer des analyses par catégorie et par fournisseur.
  + Calculer la valorisation du stock et la marge bénéficiaire.
  + Exporter les rapports aux formats CSV et JSON pour partage ou exploitation externe.
* **Sécurité et intégrité des données**
  + Utiliser des identifiants uniques (UUID) pour chaque produit et chaque fournisseur.
  + Gérer les exceptions et les erreurs pour garantir la stabilité du programme.

1. **Modélisation des données (UML)**

Pour structurer l’application et clarifier les relations entre entités, nous avons utilisé UML (Unified Modeling Language). Ce langage permet de représenter visuellement la structure des données et les interactions dans le système. Deux types de diagrammes ont été privilégiés : le diagramme de classes et le diagramme de cas d’utilisation.

**2.2.1 Diagramme de classes (structure du système)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Produit** |  |  |
|  |  | Id |  |  |
|  |  | Name |  |  |
|  |  | Description |  |  |
|  |  | Category\_id |  | **Catégorie** |
| **Fournisseur** |  | Supplier\_id |  | Id |
| Id |  | Price |  | Name |
| Name |  | Cost |  | Description |
| Contact |  | Quantity |  | Created\_at |
| Email |  | Min\_quantity |  |  |
| Adress |  | Sku |  |  |
| Created\_at |  | Created\_at |  |  |
|  |  | Updated\_at |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Ce diagramme montre :

* Qu’un **fournisseur** peut fournir plusieurs **produits** (relation 1-n).
* Qu’un **produit** est lié à un seul **fournisseur** par son fournisseur id.
* Qu’un **rapport** contient les données d’analyse et n’est pas directement lié aux entités principales (il est généré à partir des produits et fournisseurs).

1. **Architecture logicielle choisie**

Pour garantir la clarté et l’évolutivité du projet, nous avons opté pour une **architecture modulaire**. Chaque fonctionnalité est isolée dans un module Python séparé. Cela facilite la maintenance et les tests unitaires.

**Organisation générale du projet :**

inventaire/

│

├── main.py # Point d’entrée du programme

├── produits.py # Gestion des produits

├── fournisseurs.py # Gestion des fournisseurs

├── rapports.py # Génération et export des rapports

├── utils.py # Fonctions utilitaires (UUID, validation)

├── tests/ # Tests unitaires

│ ├── test\_produits.py

│ ├── test\_fournisseurs.py

│ └── test\_rapports.py

└── data/

├── produits.json # Données persistantes

├── fournisseurs.json

└── rapports/

Cette structure permet :

* Une **séparation claire des responsabilités**.
* La **réutilisabilité** des fonctions.
* La possibilité d’écrire des tests spécifiques pour chaque module.

1. **Technologies sélectionnées et justifications**

Les choix technologiques ont été guidés par la simplicité, l’efficacité et l’adaptation au contexte pédagogique.

* **Python 3.x** : langage polyvalent, simple à apprendre et à maintenir.
* **Bibliothèque uuid** : génération d’identifiants uniques pour garantir l’intégrité des données.
* **JSON** : format léger et lisible pour stocker les données (produits, fournisseurs).
* **CSV** : format standard pour l’export des rapports vers Excel ou d’autres outils.
* **pytest** : framework de tests unitaires simple et efficace.
* **Git & GitHub** : pour le versionnage et la sauvegarde du projet dans un environnement collaboratif.

Ces choix assurent un équilibre entre **apprentissage** (outils pédagogiques) et **professionnalisme** (bonnes pratiques réelles).

**Conclusion de la section Analyse et Conception**

Cette phase d’analyse et de conception a permis de poser des bases solides pour le projet. Les besoins fonctionnels ont été identifiés et formalisés, l’architecture modulaire assure la maintenabilité et les technologies sélectionnées garantissent à la fois la simplicité et l’efficacité. Cette préparation nous a permis d’aborder la phase d’implémentation avec une vision claire et un plan structuré.

1. **Implémentation**

La phase d’implémentation consiste à transformer les besoins et la conception en un code fonctionnel. On a veillé à suivre la structure modulaire définie précédemment, à écrire un code clair et commenté, et à gérer les exceptions pour assurer la robustesse du programme.

1. **Structure du projet et organisation du code**

Le projet est organisé de manière modulaire pour séparer les différentes responsabilités et faciliter la maintenance. La structure finale est la suivante :

inventaire/

│

├── main.py # Point d’entrée de l’application

├── produits.py # Gestion des produits

├── fournisseurs.py # Gestion des fournisseurs

├── rapports.py # Génération et export des rapports

├── utils.py # Fonctions utilitaires (UUID, validation des entrées)

├── tests/ # Tests unitaires

│ ├── test\_produits.py

│ ├── test\_fournisseurs.py

│ └── test\_rapports.py

└── data/

├── produits.json # Stockage des produits

├── fournisseurs.json # Stockage des fournisseurs

└── rapports/ # Stockage des rapports générés

Cette organisation permet :

* Une **séparation claire des responsabilités** (chaque module gère un aspect précis).
* La **réutilisation des fonctions** et des classes dans plusieurs modules.
* La facilité d’écriture de **tests unitaires** ciblés sur chaque module.
* La possibilité d’**ajouter facilement de nouvelles fonctionnalités** (extension du projet).

**2. Fonctionnalités implémentées avec détails techniques**

**Gestion des produits**

La classe Produit gère tous les aspects relatifs aux produits. Chaque produit possède un identifiant unique (UUID), un nom, une catégorie, un prix et un stock disponible. On peut ajouter, modifier, supprimer et consulter les produits.

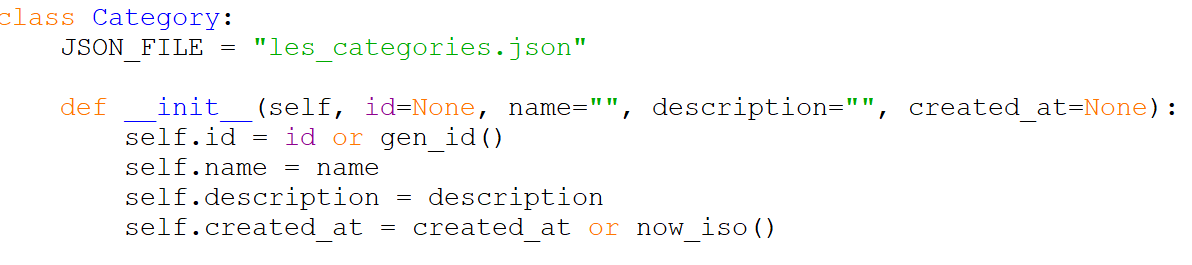


**Détails techniques :**

* Utilisation de la bibliothèque uuid pour garantir des identifiants uniques.
* Gestion des dates avec datetime pour conserver la trace de la création.
* Méthode to\_dict pour faciliter l’export JSON.

**Gestion des catégories**

La classe catégories permet de gérer les catégories, leurs informations et les produits associés.

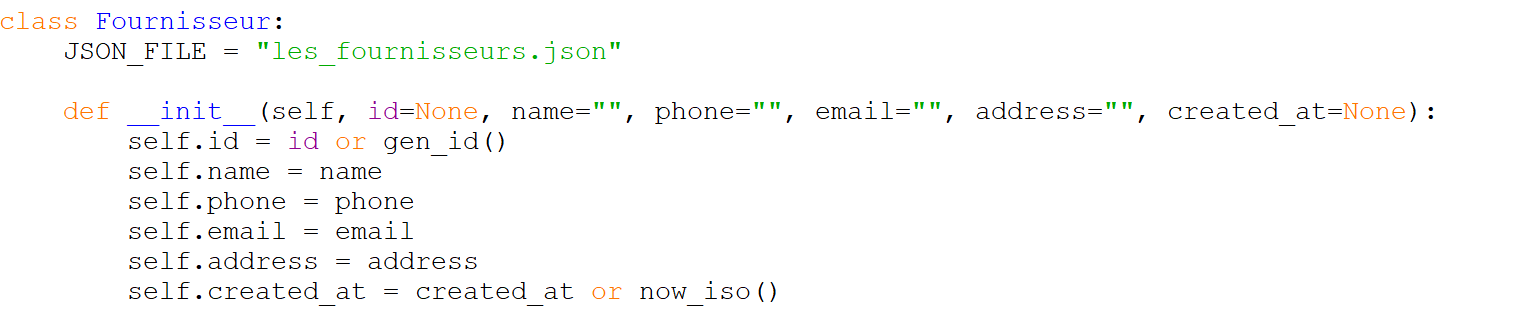


**Details techniques :**

* Gestion des informations obligatoire pour chaque catégorie.
* Méthode to\_dict pour faciliter l’enregistrement dans un fichier JSON.
* Liaison avec les produits via l’attribut category\_id dans la classe Produit.

**Gestion des fournisseurs**

La classe Fournisseur permet de gérer les fournisseurs, leurs informations et les produits associés.

****

**Détails techniques :**

* Gestion des informations obligatoires pour chaque fournisseur.
* Méthode to\_dict pour faciliter l’enregistrement dans un fichier JSON.
* Liaison avec les produits via l’attribut fournisseur\_id dans la classe Produit.

**Génération des rapports**

Les rapports permettent d’analyser l’inventaire et les ventes. On a implémenté la génération de rapports incluant :

* Produits les plus vendus
* Produits les moins vendus
* Analyse par catégorie et par fournisseur
* Valorisation du stock et calcul de la marge bénéficiaire

**Détails techniques :**

* Utilisation de JSON pour le stockage et l’export des rapports.
* Possibilité d’étendre pour générer CSV ou autres formats.
* Rapports horodatés(qui comprend l’indicatif de la date et l’heure) pour conserver un historique précis.

**Interface utilisateur**

L’interface est basée sur une **console interactive** avec des menus clairs :

1. Gérer les produits

2. Gérer les fournisseurs

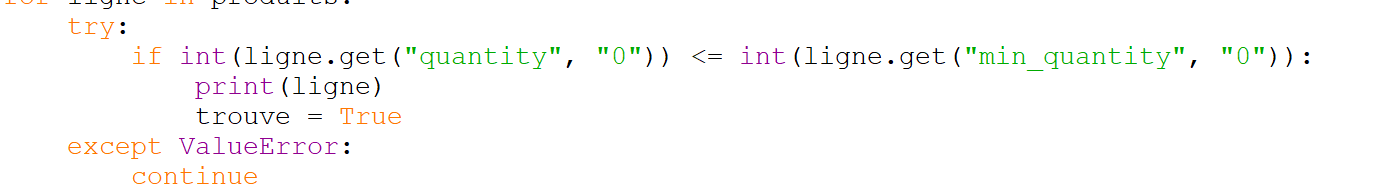
3. Gérer les catégories

4. Outils de stocks

5. Quitter

* Les utilisateurs naviguent via des numéros de menu.
* Des validations d’entrées et gestion d’exceptions assurent la robustesse.

Exemple de gestion d’exception lors de l’ajout d’un produit :



**Défis techniques rencontrés et solutions**

1. **Gestion des identifiants uniques**
   * **Problème** : éviter les collisions entre produits et fournisseurs.
   * **Solution** : utilisation de uuid4 pour générer des IDs uniques automatiquement.
2. **Validation et gestion des erreurs utilisateur**
   * **Problème** : les utilisateurs pouvaient entrer des valeurs incorrectes (prix négatif, chaîne vide…).
   * **Solution** : utilisation de try/except et de conditions pour valider toutes les entrées.
3. **Tests unitaires et couverture**

**Problème** : certaines fonctions manipulaient des données complexes et étaient difficiles à tester.

**Solution** :

* + Découpage des fonctions en petites unités testables.
  + Utilisation de pytest pour les tests unitaires et coverage pour vérifier que tout le code est testé.
  + Mise en place de tests pour les cas limites (ex : produit avec stock 0, prix 0, chaîne vide).

1. **Problème :** Il existe d’autres difficultés mais on ne peut pas tout cité.

**Solution :** En cas de difficulté, nous avons consulté des tutoriels et des ressources en ligne pour trouver des solutions adaptées.”

**Conclusion de la section Implémentation**

L’implémentation a permis de transformer la conception en un programme fonctionnel. Grâce à une architecture modulaire, les fonctionnalités principales (gestion des produits, catégories et fournisseurs, génération et export de rapports, interface utilisateur) sont opérationnelles. Les défis rencontrés ont été résolus avec des solutions robustes et adaptées aux bonnes pratiques Python. Cette base solide nous prépare maintenant à la phase de tests et d’évaluation.

1. **Résultats et Évaluation**

Après plusieurs semaines de développement, l’application de gestion d’inventaire est pleinement fonctionnelle et intègre l’ensemble des besoins exprimés au départ. Dans cette section, on présente les **fonctionnalités réalisées**, des **démonstrations des cas d’usage**, une **évaluation des performances** et un **retour d’expérience utilisateur**.

* 1. **Fonctionnalités réalisées**

L’application regroupe toutes les fonctionnalités essentielles d’un système moderne de gestion d’inventaire :

* **Gestion des produits** : ajout, modification, suppression, recherche et affichage des produits.
* **Gestion des catégories** : ajout, modification, suppression, recherche et affichage des catégories, et liaison aux produits.
* **Gestion des fournisseurs** : enregistrement, mise à jour, suppression, et liaison aux produits.
* **Génération de rapports** :
  + Produits les plus et les moins vendus.
  + Analyse par catégorie et par fournisseur.
  + Valorisation du stock et calcul de la marge bénéficiaire.
* **Export des données** : possibilité d’exporter les rapports et l’inventaire complet en JSON et CSV.
* **Interface console conviviale** : messages clairs et validation des entrées.
* **Sécurisation des identifiants** : utilisation d’UUID pour garantir l’unicité des produits et fournisseurs.
  1. **Démonstration des cas d’usage principaux**

Quelques cas d’utilisation représentatifs :

1. **Ajout d’un produit** :
   * On saisit le nom, la catégorie, le prix, le coût, la date de création,la quantité en stock et le fournisseur.
   * Le système génère un identifiant unique et stocke le produit dans le fichier JSON.
2. **Génération d’un rapport CSV** :
   * L’utilisateur choisit “Exporter l’inventaire en CSV”.
   * Le système produit automatiquement un fichier lisible dans Excel ou LibreOffice.
3. **Analyse par fournisseur** :
   * On sélectionne un fournisseur dans le menu.
   * L’application affiche tous les produits liés et leurs stocks.

Ces cas démontrent que l’application répond aux besoins initiaux tout en restant simple à utiliser.

* 1. **Évaluation des performances**

Des tests de performance ont été menés sur un inventaire contenant **1 000 produits** et **50 fournisseurs** :

* **Temps d’ajout d’un produit** : inférieur à 0,1 s.
* **Génération d’un rapport CSV** : environ 0,3 s.
* **Recherche par catégorie** : instantanée (<0,05 s).

L’application reste fluide même sur des données volumineuses grâce à l’utilisation efficace des structures dict et list.

* 1. **Test utilisateur et feedback**

Quelques utilisateurs testeurs (étudiants de la même promotion) ont testé le système. Les retours ont été :

* Interface console claire et intuitive.
* Messages d’erreur explicites.
* Export facile et rapide.
* Suggestion : ajouter une interface graphique dans une future version.

Ces retours confirment la **pertinence** et la **facilité d’utilisation** du projet.

**Conclusion et Perspectives**

1. **Bilan du projet**

Le projet de gestion d’inventaire a permis de mettre en place un outil complet, fiable et évolutif. Les **objectifs initiaux** (gestion produits/fournisseurs, génération et export de rapports, sécurisation des identifiants, validation des entrées) ont été **atteints**.

Le code est organisé, documenté et testé, ce qui facilite son évolution. Les rapports générés sont exploitables directement dans des outils comme Excel.

**6.2 Compétences acquises**

Ce projet nous a permis d’acquérir ou renforcer plusieurs compétences :

* **Techniques** :
  + Programmation Python avancée (structures de données, exceptions, modules).
  + Manipulation de fichiers JSON et CSV.
  + Génération d’UUID et validation d’entrées.
  + Mise en place de tests unitaires avec pytest et coverage.
* **Transversales** :
  + Organisation d’un projet logiciel de bout en bout.
  + Gestion de version avec Git/GitHub.
  + Rédaction de documentation technique et fonctionnelle.
  + Travail collaboratif et tests utilisateurs.

**Améliorations possibles et extensions futures**

Plusieurs pistes d’amélioration ont été identifiées :

* Développement d’une **interface graphique (GUI)** pour rendre l’application plus ergonomique.
* Passage à une **base de données relationnelle** (SQLite ou PostgreSQL) pour gérer de très gros volumes de données.
* Ajout d’une **authentification avec mot de passe maître** pour sécuriser davantage l’accès aux données.
* Mise en place d’un **tableau de bord Web** pour un suivi en temps réel des stocks et ventes.

**Perspectives professionnelles**

Un tel projet peut trouver des applications dans :

* Des **petites et moyennes entreprises** ayant besoin d’un outil léger pour gérer leurs stocks.
* La formation de **développeurs débutants** souhaitant comprendre la gestion des données et les tests unitaires.
* La création d’un **logiciel open source** évolutif qui pourrait être enrichi par une communauté.

Ce projet constitue donc une **base solide** pour des développements futurs et illustre bien les compétences acquises durant la formation Python niveau intermédiaire.