Rapport du Projet TaskManagerMVC

Sami Laouaj, Yacine Le Mouel, Amine Boujemaaoui

1er février 2025

1 Introduction

Le projet **TaskManagerMVC** est une application de gestion de tâches développée en Java en respectant les principes de conception **SOLID** ainsi que l'architecture **Modèle-Vue-Contrôleur (MVC)**. L'objectif principal de ce projet est de fournir une structure propre et évolutive permettant la gestion des tâches, des projets et des missions, tout en appliquant des pratiques avancées de développement logiciel.

2 Objectifs du Projet

L'application vise à :
- Permettre la création, la modification, la suppression et l'affichage des tâches.
- Structurer les tâches en missions et en projets pour une meilleure organisation.
- Assurer une gestion des données en utilisant des fichiers CSV pour la persistance.
- Appliquer les principes SOLID afin de garantir un code maintenable et modulaire.

3 Architecture Générale

L'application repose sur une architecture MVC composée des trois couches suivantes :

3.1 Modèle

La couche **modèle** gère les données et leur persistance. Elle est divisée en deux sous-parties :
- **Entities** : Contient les classes représentant les entités métier telles que `Tache`, `Projet`, et `Mission`.
- **Repositories** : Contient les classes de gestion des entités et d'interaction avec les fichiers, telles que `TacheRepository`, `ProjetRepository`, et `MissionRepository`.

3.2 Vue

La couche **vue** est responsable de l'affichage des données et de l'interaction avec l'utilisateur via la console. Chaque entité dispose d'une vue associée, telles que `TacheView`, `ProjetView` et `MissionView`.

3.3 Contrôleur

Les contrôleurs gèrent la communication entre la vue et le modèle. L'application utilise `MainController` pour orchestrer les interactions, ainsi que des contrôleurs spécifiques pour chaque entité comme `TacheController`, `ProjetController`, et `MissionController`.

4 Fonctionnalités Principales

L'application TaskManagerMVC permet à l'utilisateur de :
- Ajouter, modifier et supprimer des tâches.
- Regrouper les tâches sous forme de missions.
- Organiser les missions dans des projets.
- Afficher la liste des tâches, des missions et des projets avec un affichage formaté en console.
- Sauvegarder et charger les données via des fichiers CSV.

5 Persistance des Données

Les données sont stockées sous format CSV dans le répertoire `resources/` :
- `taches.csv` : Contient les tâches enregistrées.
- `missions.csv` : Contient les missions et les tâches qui leur sont associées.
- `projets.csv` : Contient les projets et leurs missions respectives.

Ces fichiers sont gérés à l'aide de classes utilitaires telles que `TacheFileUtil`, `MissionFileUtil`, et `ProjetFileUtil`.

6 Application des Design Patterns et des Principes SOLID

L'application TaskManagerMVC suit plusieurs principes de conception pour assurer un code modulaire, maintenable et évolutif.

6.1 Principes SOLID

- **Single Responsibility Principle (SRP)** : Chaque classe a une responsabilité unique. Par exemple, `TacheRepository` est uniquement responsable de la gestion des tâches.
- **Open/Closed Principle (OCP)** : L'architecture permet d'ajouter de nouvelles fonctionnalités sans modifier les classes existantes, en utilisant l'héritage et le polymorphisme.
- **Liskov Substitution Principle (LSP)** : Les sous-classes peuvent remplacer les classes parents sans altérer le comportement du programme.
- **Interface Segregation Principle (ISP)** : Les interfaces `Observateur` et `Sujet` assurent une séparation claire des responsabilités.
- **Dependency Inversion Principle (DIP)** : L'application repose sur des abstractions plutôt que sur des implémentations concrètes.

6.2 Design Patterns Utilisés

6.2.1 Modèle-Vue-Contrôleur (MVC)

Le pattern **MVC** est utilisé pour séparer les préoccupations et structurer l'application de manière claire :

- **Modèle** : Gère les données et la logique métier à travers les entités et les repositories.
- **Vue** : Présente les données sous forme lisible pour l'utilisateur en console.
- **Contrôleur** : Gère la communication entre le modèle et la vue.

Ce pattern permet d'améliorer la modularité, de faciliter les tests et d'assurer une meilleure maintenabilité du code.

6.2.2 Observer Pattern

L'**Observer Pattern** est utilisé pour assurer la mise à jour automatique des vues lorsque les données changent. L'interface `Sujet` permet d'ajouter ou de retirer des observateurs (`Observateur`), et dès qu'un changement se produit dans le repository, les observateurs en sont informés automatiquement.

6.2.3 Singleton Pattern et Gestion des Fichiers

Les classes utilitaires de gestion des fichiers, comme `TacheFileUtil`, utilisent le **Singleton Pattern** pour garantir qu'une seule instance est utilisée dans toute l'application. Cela permet d'éviter les problèmes liés à la manipulation concurrente des fichiers et optimise l'accès aux données.

7 Diagrammes UML et Interprétation

Cette section présente les différents diagrammes UML utilisés pour modéliser l'architecture et les interactions du projet TaskManagerMVC.

7.1 Diagramme de Vue Générale du Projet

Ce diagramme offre une vue d'ensemble des principales classes et de leurs relations. Il illustre les entités, les contrôleurs, les repositories et les vues, facilitant ainsi la compréhension de l'organisation générale du projet.

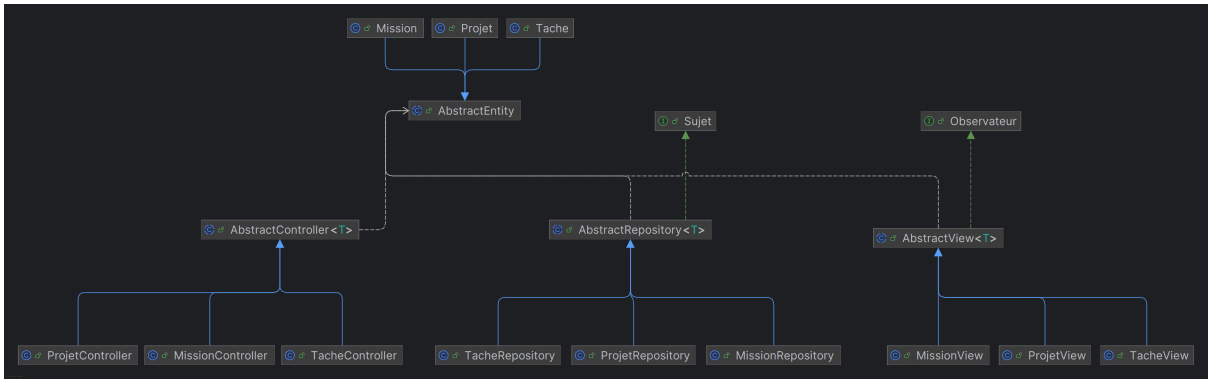

FIGURE 1 – Diagramme de Vue Générale du Projet

7.2 Diagramme des Contrôleurs

Ce diagramme détaille les relations entre les contrôleurs de l'application. Le `MainController` orchestre les interactions entre les `TacheController`, `ProjetController` et `MissionController`, garantissant ainsi une gestion centralisée des différentes entités.

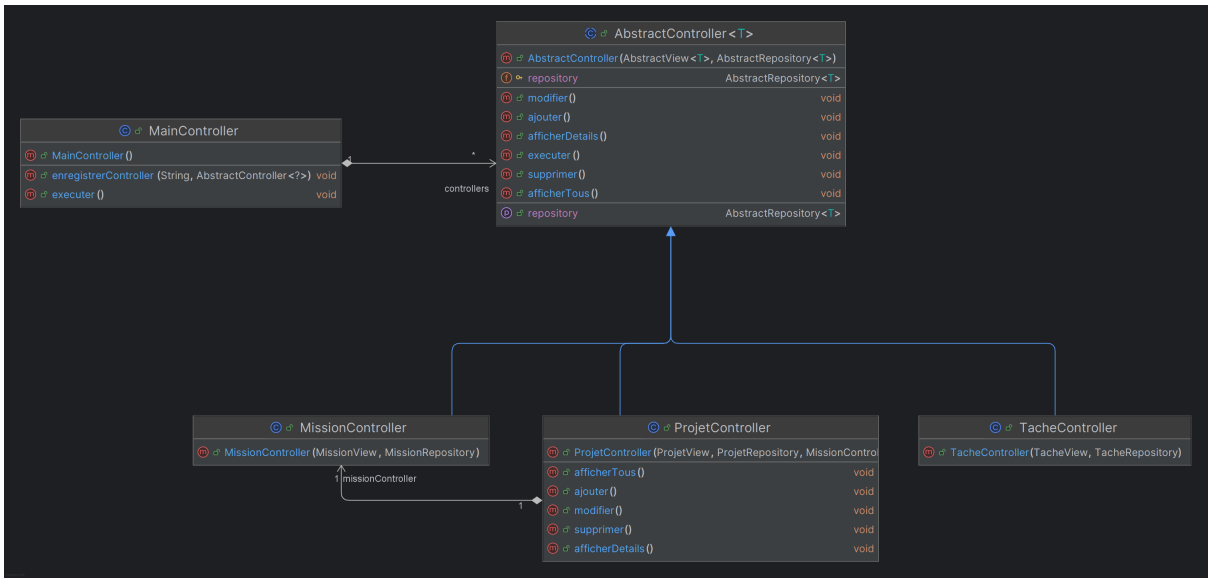

FIGURE 2 – Diagramme des Contrôleurs

7.3 Diagramme des Entités et des Repositories

Ce diagramme met en évidence les entités principales du projet (`Tache`, `Projet`, `Mission`) et leurs repositories respectifs. Il démontre comment ces entités sont gérées et persistées dans l'application.

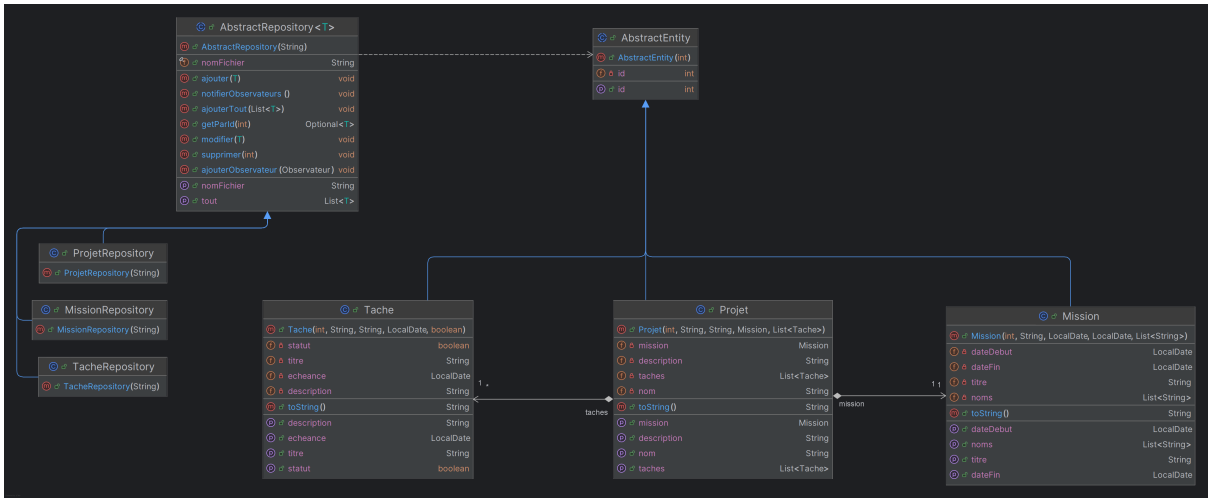

FIGURE 3 – Diagramme des Entités et des Repositories

7.4 Diagramme des Vues

Les classes de vue permettent d'afficher les informations et de capturer les interactions utilisateur. Ce diagramme présente l'organisation de `TacheView`, `ProjetView` et `MissionView`, et leur relation avec les contrôleurs.

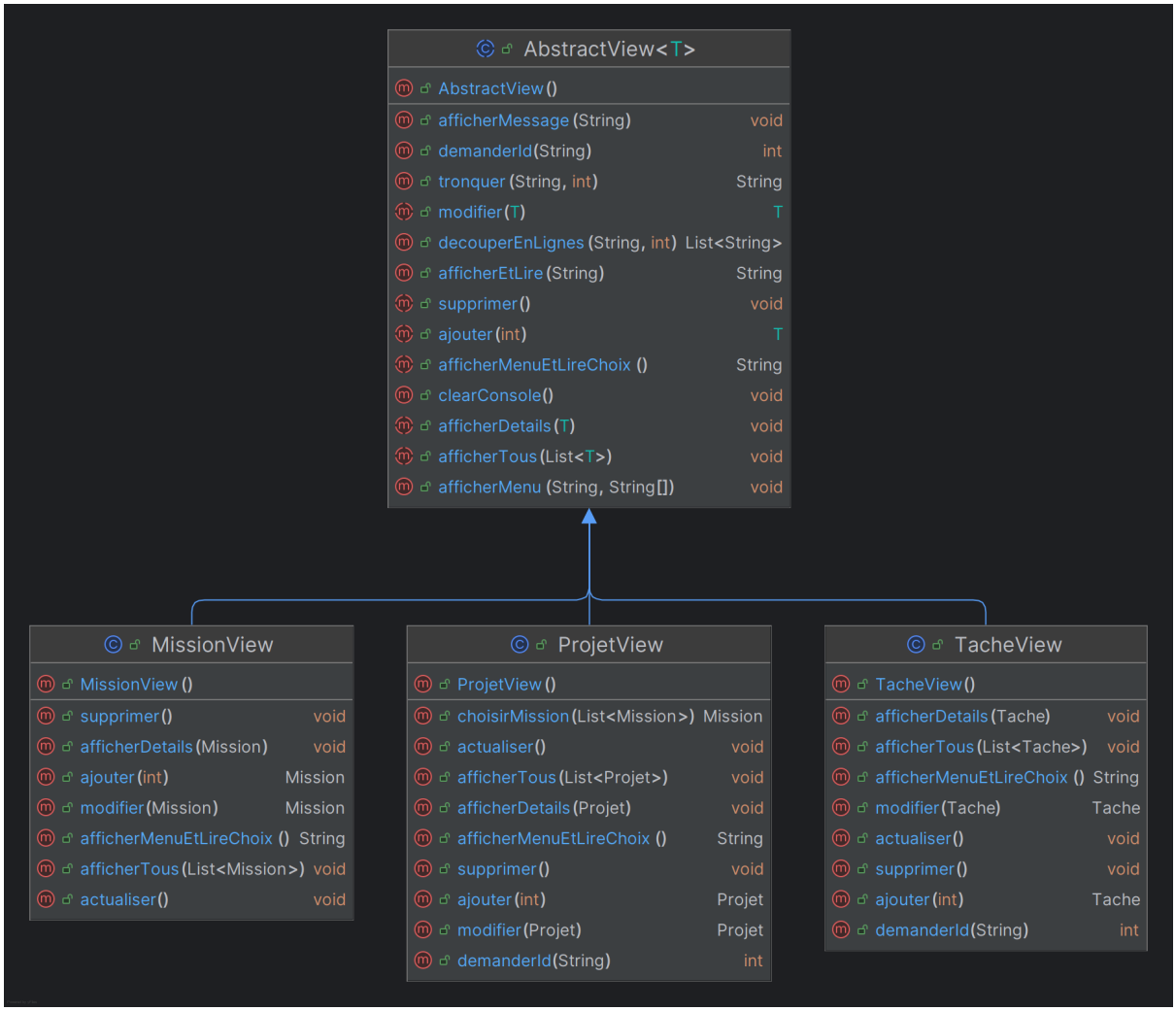

FIGURE 4 – Diagramme des Vues

7.5 Diagramme des Relations entre les Repositories et les FileUtils

Les fichiers de persistance sont gérés via des classes utilitaires spécifiques (`TacheFileUtil`, `ProjetFileUtil`, `MissionFileUtil`). Ce diagramme illustre comment les repositories interagissent avec ces classes pour charger et sauvegarder les données.

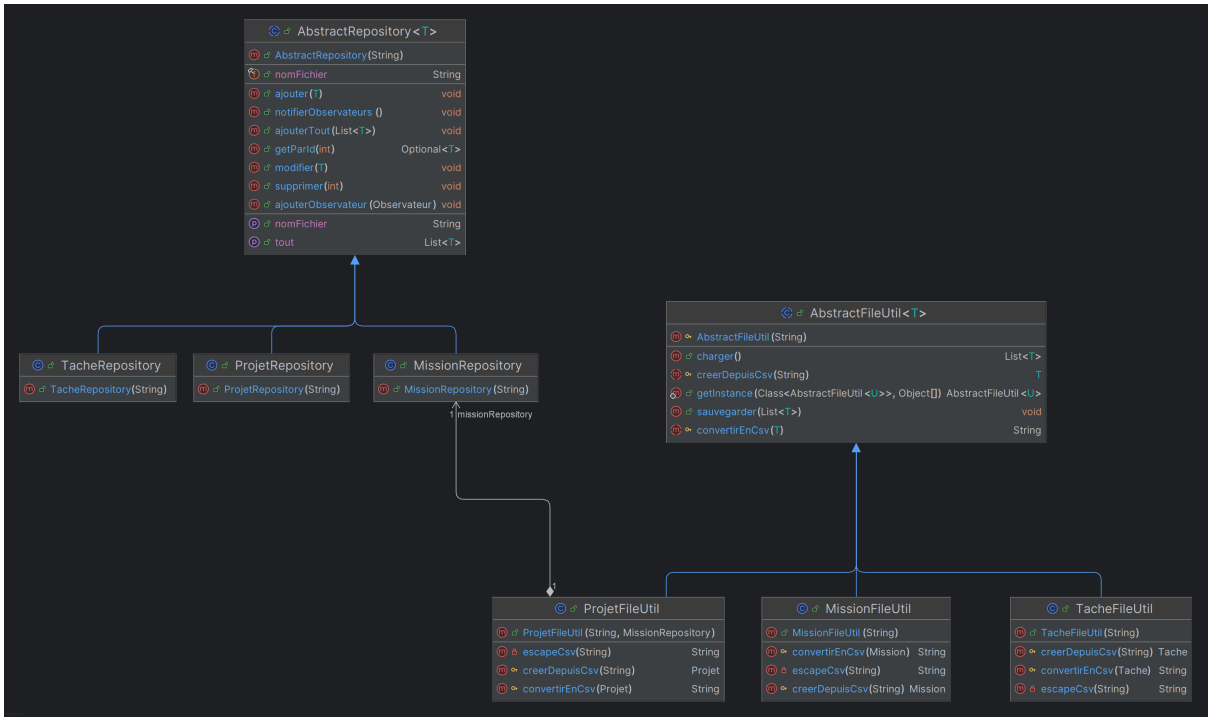

FIGURE 5 – Diagramme des Relations entre les Repositories et les FileUtils

7.6 Diagramme Général des Classes Abstraites et Interfaces

Ce diagramme met en avant les principales classes abstraites et interfaces du projet, telles que `AbstractRepository`, `AbstractEntity`, `AbstractView`, ainsi que les interfaces `Sujet` et `Observateur`. Il démontre l'implémentation des principes SOLID et favorise la réutilisation du code.

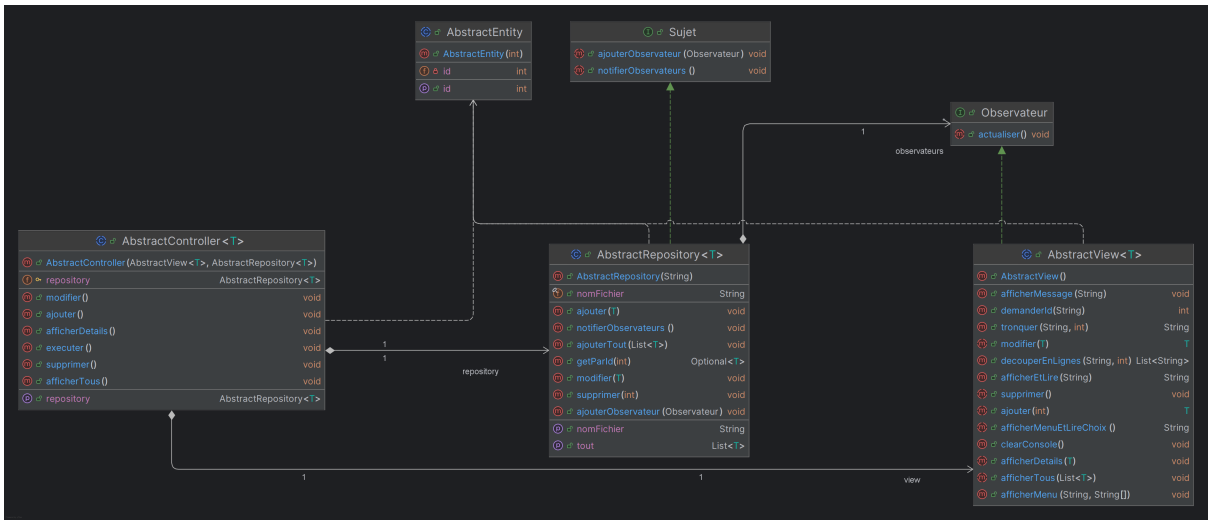

FIGURE 6 – Diagramme Général des Classes Abstraites et Interfaces

8 Conclusion

Le projet TaskManagerMVC a été conçu en respectant les bonnes pratiques de développement, notamment à travers l'utilisation des design patterns et des principes SOLID. La modularité offerte par l'architecture MVC, couplée aux modèles de conception comme Observer et Singleton, garantit une application flexible et évolutive. Grâce à ces choix architecturaux, TaskManagerMVC peut facilement être étendu pour intégrer de nouvelles fonctionnalités et améliorer l'expérience utilisateur.