2024-2025

Rapport

Y Microblog

Etudiants :

AZOUAOU LOUAIFI
GIFU IONUT

Introduction

Le projet que nous avons entrepris consiste à développer une application de microblogging minimaliste, offrant des fonctionnalités de base mais essentielles pour illustrer le fonctionnement d'une telle plateforme. Nous avons choisi de refactorer cette application pour intégrer plusieurs design patterns, notamment afin d'améliorer sa maintenabilité, sa lisibilité et sa modularité. Une des améliorations majeures a été l'implémentation d'un système d'affichage des messages en fonction de divers facteurs, comme la pertinence, la popularité ou encore la récente activité des utilisateurs. En combinant ces éléments, nous avons créé un environnement simplifié mais fonctionnel, qui met en évidence les aspects clés du développement logiciel et de l'éthique associée.

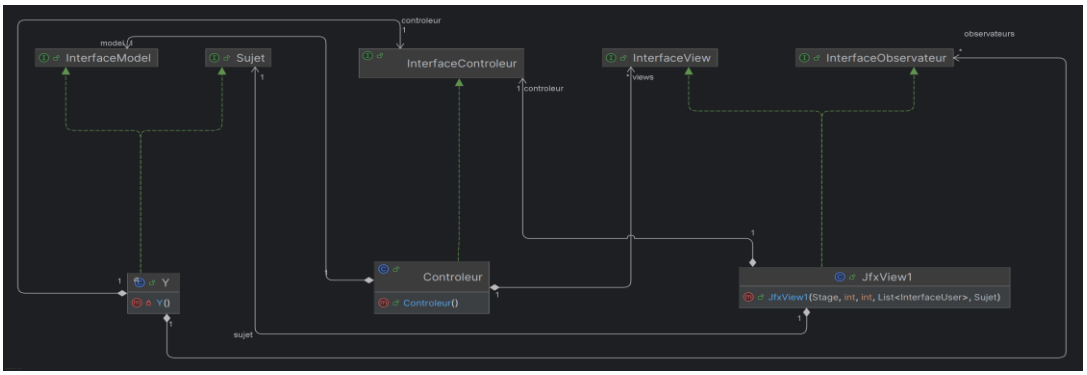
Design patterns

1. MVC

Le pattern MVC a été choisi pour séparer clairement les responsabilités dans l'application de microblogging. Le modèle représenté par la classe Y gère la logique métier et les données (utilisateurs, messages), tandis que la vue (JfxView1 se concentrent uniquement sur l'affichage et l'interaction utilisateur). Le contrôleur agit comme intermédiaire, acheminant les actions utilisateur vers le modèle et assurant la cohérence de l'application. En suivant le pattern MVC, chaque couche (Modèle, Vue, Contrôleur) est découplée des autres, ce qui facilite la maintenance et réduit l'impact des modifications dans une partie de l'application sur les autres parties.

Le pattern **Observateur** a été implémenté via les interfaces Sujet et InterfaceObservateur pour gérer efficacement la mise à jour des vues. Lorsque le modèle est modifié (ajout/suppression de message, changement de stratégie de tri, marquage en favori), la vue est automatiquement notifiée et mise à jour sans avoir à connaître les détails internes du modèle. Cette approche permet une gestion efficace de la synchronisation entre le modèle et les vues. Elle assure un découplage clair et une réactivité maximale entre les composants, tout en offrant une grande flexibilité pour l'ajout et l'extension de fonctionnalités, comme l'intégration de nouvelles vues sans perturber l'existant.

Diagramme de classes qui implemente le MVC et Observateur



2. GRASP

Voici quelques exemples de principes respectés dans notre projet, conformément aux concepts de GRASP :

Faible couplage :

Le principe de faible couplage est respecté grâce à l'utilisation d'interfaces comme `InterfaceView`, `InterfaceModel` et `InterfaceControleur`. Les classes interagissent principalement avec ces interfaces plutôt qu'avec des implémentations concrètes. Cela permet de réduire la dépendance directe entre les composants et facilite l'évolution du système sans impacter les autres parties de l'application.

Forte cohésion :

La cohésion est bien respectée dans les classes `Message` et `MessageData`, qui ont des responsabilités clairement définies. La classe `Message` gère les données relatives au contenu et à la date du message, tandis que `MessageData` s'occupe des métadonnées, telles que le score et l'état de "bookmarked". Chaque classe est responsable d'un aspect précis de la gestion des messages.

Indirection :

Le principe de l'indirection est appliqué via le contrôleur `Controleur`, qui joue le rôle d'intermédiaire entre les vues et le modèle. Le contrôleur permet de réduire le couplage entre le modèle et les vues en orchestrant les interactions entre ces composants sans les lier directement.

Contrôleur :

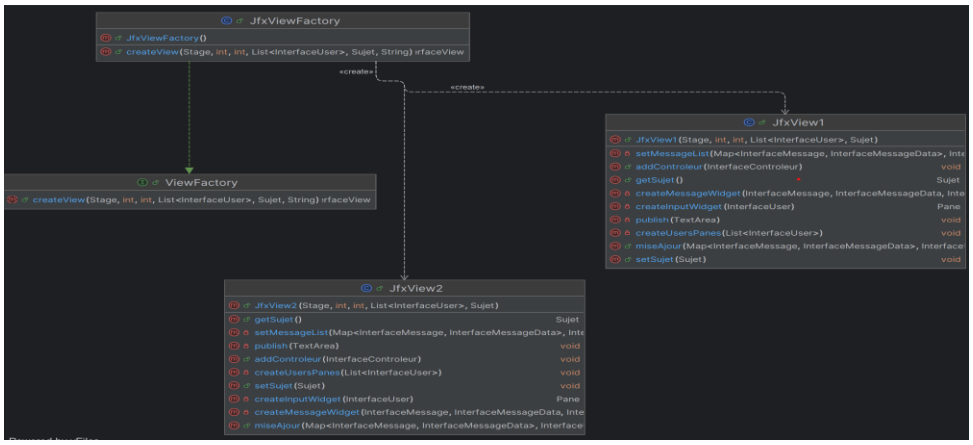
Le contrôleur est bien implémenté dans la classe `Controleur`, qui gère les événements du système. Il coordonne les interactions entre la vue et le modèle, facilitant ainsi la séparation des responsabilités et la gestion de la logique métier sans perturber l'affichage.

3. Singleton et Factory Method :

Le pattern **Singleton** a été appliqué à la classe Y, car nous considérons que le modèle est unique dans l'application et ne doit avoir qu'une seule instance. Cela garantit l'unicité du modèle et assure que toutes les vues partagent le même état. Ce modèle centralisé évite les problèmes de synchronisation des données entre différentes instances et permet une gestion centralisée des messages et des utilisateurs.

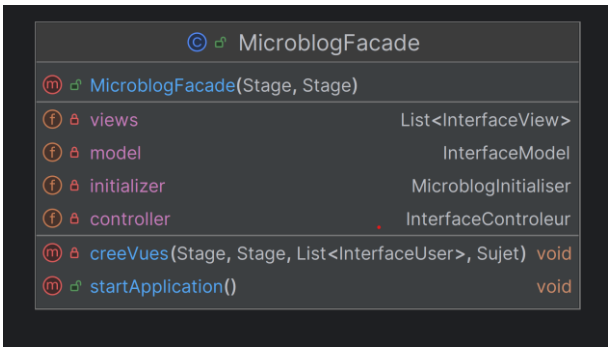
Pour le pattern **Factory Method**, nous l'avons appliqué en utilisant l'interface `ViewFactory` et son implémentation `JfxViewFactory`. Ce design permet de séparer clairement les responsabilités, en isolant la logique de création des vues (comme `JfxView1` et `JfxView2`). Le code client, n'a pas besoin de connaître les détails de la création des vues, ce qui simplifie son utilisation. De plus, cette approche favorise la **flexibilité et l'extensibilité**, car il devient facile d'ajouter de nouvelles vues sans avoir à modifier le code existant, et elle **encapsule la complexité** de la création des vues, permettant ainsi de maintenir un code client plus simple et propre.

Diagramme de classes qui implemente le pattern « Factory Method »



4. Facade

Nous avons implémenté le pattern **Facade** pour simplifier l'interface du système. Ce pattern permet de cacher la complexité interne aux clients, en offrant une interface unifiée et simplifiée. Il favorise le **découplage** en permettant aux clients de ne pas avoir à connaître les détails d'implémentation, réduisant ainsi les dépendances. La façade fournit également un **point d'entrée unique**, centralisant l'accès aux fonctionnalités du système et facilitant son utilisation.



5. Stratégie

Nous avons implémenté le **pattern Stratégie** pour gérer le scoring, l'affichage et le tri des messages. Ce pattern permet un changement dynamique de comportement, offrant aux utilisateurs la possibilité de modifier la façon dont les messages sont triés et affichés sans redémarrer l'application. Il facilite également **l'**ajout de nouvelles stratégies sans modifier le code existant, grâce à une séparation des responsabilités qui isole la logique de scoring et de tri. Chaque stratégie est modulaire et réutilisable, permettant une configuration indépendante pour chaque utilisateur. Enfin, ce pattern offre une **architecture extensible**, adaptée à l'ajout de futures stratégies.

Diagramme de class pour l'implémentation du pattern « Stratégie » pour le scoring :

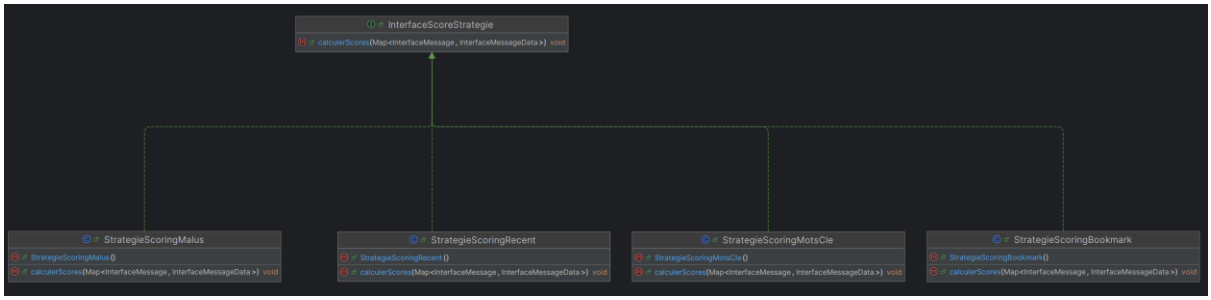
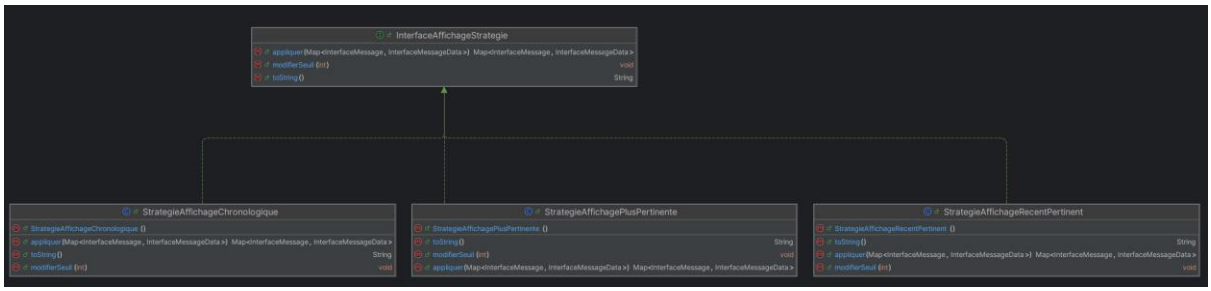


Diagramme de class pour l'implémentation du pattern « Stratégie » pour l'affichage et le trie :

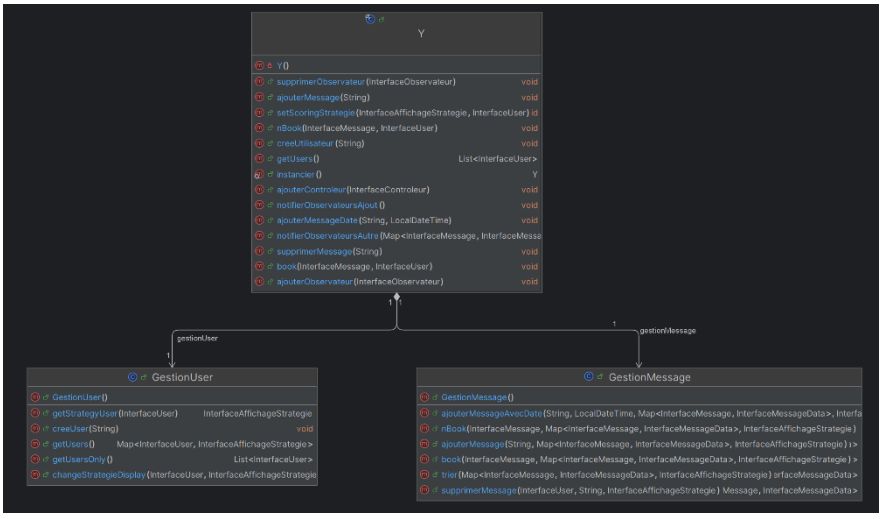


6. SOLID

Single Responsibility Principle (SRP) :

La classe `Y` gère actuellement plusieurs responsabilités (gestion des utilisateurs et des messages), ce qui contrevient au principe de responsabilité unique. Pour améliorer la maintenabilité et respecter ce principe, nous avons décidé de refactorer en deux classes distinctes :

- `GestionUser` : Responsable de la gestion des utilisateurs (création, récupération, etc.)
- `GestionMessage` : Responsable de la gestion des messages (ajout, tri, suppression).
  Pour les autres classes, le principe de responsabilité unique est correctement respecté.



Open/Closed Principle (OCP) :

Ce principe est respecté grâce à l'implémentation du **pattern Stratégie**, permettant d'ajouter de nouvelles stratégies de tri ou de scoring sans modifier le code existant.

Liskov Substitution Principle (LSP) :

Ce principe est bien respecté. Par exemple, les classes concrètes JfxView1 et JfxView2 peuvent être utilisées de manière interchangeable partout où l'interface InterfaceView est attendue, garantissant ainsi la substituabilité.

Interface Segregation Principle (ISP) et Dependency Inversion Principle (DIP) :

Ces principes sont appliqués en utilisant des interfaces pour toutes les classes, assurant un faible couplage, une meilleure modularité, et facilitant les tests et les extensions. Grâce au **Dependency Inversion Principle**, les classes dépendent des abstractions (interfaces) plutôt que des implémentations concrètes, ce qui améliore la flexibilité et l'évolutivité de l'architecture.

Éthique

Les plateformes de microblogging et les réseaux sociaux jouent un rôle essentiel dans la communication et la diffusion d'informations. Toutefois, elles posent des problèmes éthiques majeurs, notamment en ce qui concerne la manipulation de l'information, la protection de la vie privée et l'impact sur les comportements humains. Dans le cadre de notre projet de microblog, nous avons intégré un système de scoring, semblable aux systèmes de recommandation couramment utilisés pour fournir un contenu « pertinent » aux utilisateurs.
Cette section explore les enjeux éthiques associés aux plateformes de microblogging, en mettant l'accent sur les parties prenantes, les avantages et risques des systèmes de recommandation, ainsi que les mesures possibles pour en limiter les dérives.

Selon une étude de *Hootsuite* en 2024, près de 60 % de la population mondiale utilise activement les réseaux sociaux, ce qui représente environ 4,8 milliards d'utilisateurs. Chaque utilisateur passe en moyenne 2 heures et 24 minutes par jour sur ces plateformes.
En France, l'INSEE rapporte que 76 % des adultes consultent régulièrement un réseau social, avec une prédominance de Facebook, Instagram et Twitter. Ces chiffres montrent l'ampleur de l'influence que ces plateformes peuvent avoir sur les comportements et opinions.

Bien que les avantages qu'apportent les algorithmes de recommandation sur les plateformes soient indéniables, les désavantages semblent bien plus nombreux. Par exemple, une analyse de *Veritasium* sur les mécanismes de recommandation de YouTube montre comment ces algorithmes favorisent souvent les contenus sensationnalistes ou polémiques, contribuant à la polarisation des opinions et à la propagation de la désinformation. Une étude publiée dans *Nature Human Behaviour* (2022) met en évidence que les systèmes de recommandation

accentuent les biais préexistants des utilisateurs en privilégiant les contenus confirmant leurs croyances, créant ainsi des « écosystèmes fermés ».

Un exemple frappant est celui du premier scrutin des élections présidentielles roumaines en 2024, annulé après des accusations de manipulation de l'opinion publique par le biais des réseaux sociaux. Les enquêtes ont révélé que des algorithmes de recommandation avaient largement favorisé la promotion d'un candidat en amplifiant ses messages populistes et polémiques. Cette surreprésentation a influencé de manière significative les choix des électeurs, soulevant des questions éthiques sur la responsabilité des plateformes dans le processus démocratique. Les médias roumains l'ont d'ailleurs surnommé « *Le candidat TikTok* », en référence à la large diffusion de ses vidéos sur cette plateforme, où son contenu optimisé pour l'engagement a rencontré un immense succès.

Les préoccupations concernant TikTok ne se limitent pas à la manipulation des algorithmes. Une enquête menée par le Sénat français, publiée en juillet 2023 sous le titre **«** *La tactique TikTok : opacité, addiction et ombres chinoises* **»**, a révélé des problèmes majeurs de transparence et d'éthique autour de cette application populaire chez les plus jeunes. Après quatre mois d'investigations, les sénateurs ont dénoncé une « opacité à plusieurs échelles » concernant les pratiques de la plateforme.

Pour améliorer la situation et réduire les dérives éthiques, plusieurs actions peuvent être envisagées. Tout d'abord, les plateformes devraient être tenues de rendre leurs algorithmes plus transparents en fournissant des explications sur la manière dont les recommandations sont générées. Les gouvernements et régulateurs devraient également s'assurer que les données des utilisateurs sont protégées et que les contenus mis en avant respectent des critères d'éthique et de responsabilité. L'avenir des réseaux sociaux repose sur un équilibre entre innovation et responsabilité sociétale.