EL OUARRAD MARIAM
ATAMNA MIRYAM

RAPPORT PROJET

décembre 2024

MATTHIEU MOY

mif01

SOMMAIRE

01

INTRO

Introduction

Présentation du projet

Notre projet “Y Microblogging” est une application de microblogging inspirée de plateformes comme Twitter, développée dans le cadre de l’UE Génie Logiciel. L'objectif principal de cette application est de permettre aux utilisateurs de publier, trier et interagir avec des messages dans un environnement orienté objet.

L’application repose sur des principes fondamentaux de la programmation orientée objet tels que l’héritage, l’encapsulation et la composition. Chaque utilisateur peut publier, supprimer des messages sur son profil, interagir avec ceux des autres (par exemple, en les ajoutant à ses favoris), et organiser ses publications selon différents critères. Par ailleurs, pour améliorer l'expérience utilisateur, lorsque celui-ci se déconnecte et revient ultérieurement, toutes ses données, y compris ses messages postés et ses paramètres de tri personnalisés, sont préservées, garantissant une continuité dans l’utilisation de l'application.

Technologies utilisés

- Langage de programmation : Java, utilisé pour développer l'application de manière structurée et modulaire (aussi parce que c’était demandé...).
- Gestion des dépendances : Maven, un outil qui facilite la gestion des bibliothèques nécessaires au projet.

02

DESIGN PATTERN

Design Patterns

Pattern MVC

Pour notre application, il nous a été demandé de mettre en place un pattern MVC.

Premièrement un pattern MVC qu'est ce que c'est ?

MVC qui veut dire Modèle-Vue-Contrôleur, sert à découpler les 3 parties cités.

- Le Modèle doit gérer la logique métier et les données
- La Vue doit gérer l'affichage de l'interface utilisateur
- Le Contrôleur est le fil de communication entre les deux, en tant qu'intermédiaire

Dans notre application, nous avons choisi une architecture de modèle passif. C'est-à-dire que le contrôleur gère les interactions entre la vue et le modèle. Contrairement à un modèle actif, ici ils n'interagissent pas directement entre eux.

Un choix justifié par deux principales raisons :

- Une séparation claire des responsabilités, chaque composante est simple et délimité
- Un meilleur contrôle des interactions, étant donné que tout passe par le contrôleur, on évite les mauvaises surprises en passant par des contrôles nécessaires avant de modifier les données.

Séparation des responsabilités

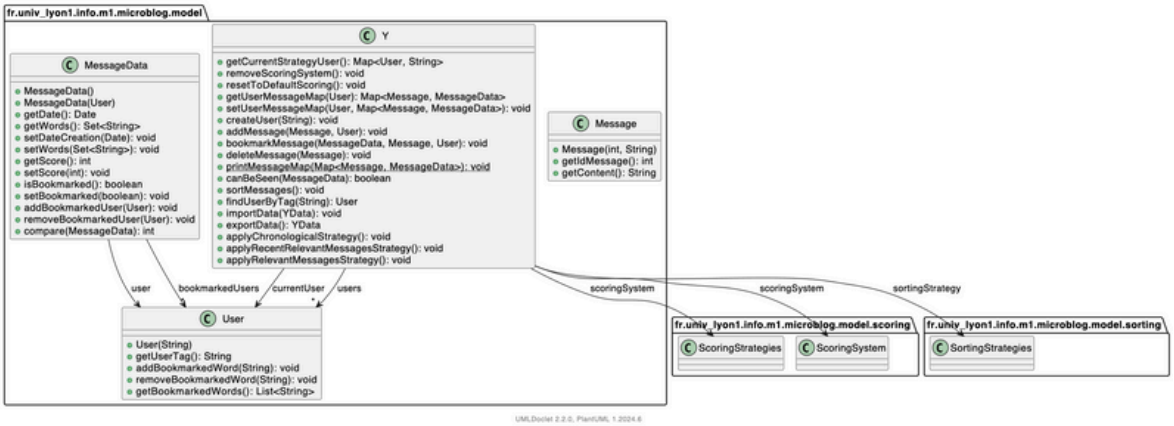
Conformément au principe de responsabilité unique (Single Responsibility Principle - SRP), nous avons séparé les fonctions principales de notre application dans les trois composants MVC, chacun étant dédié à un rôle précis. Pour cela, chaque fonction ont été triées selon ce qu'elles faisaient, pour les placer dans une des trois catégories.

Le Modèle

Le modèle regroupe la logique métier et les données. Dans notre application, il est conçu pour être totalement indépendant de l'interface utilisateur, ce qui permet une réutilisation dans différents contextes .

Principales responsabilités du modèle :
- Gestion des utilisateurs
- Gestion des messages
- Stratégies dynamiques
- Importation/Exportation des données

Le modèle encapsule toutes les données et leur manipulation. Il expose ses méthodes publiques pour que le contrôleur puisse les appeler.

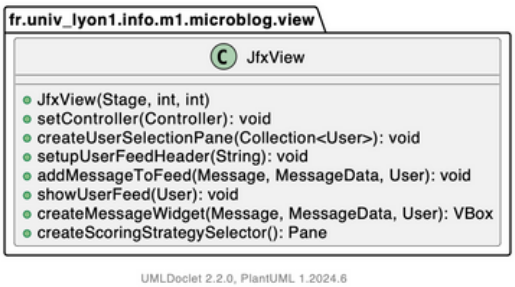


La Vue

La vue est responsable de l'interface utilisateur, implémentée avec JavaFX. Elle traduit les données du modèle en une interface graphique interactive.

Principales responsabilités de la vue
- Affichage des données
- Gestion des interactions utilisateur
- Appels au contrôleur

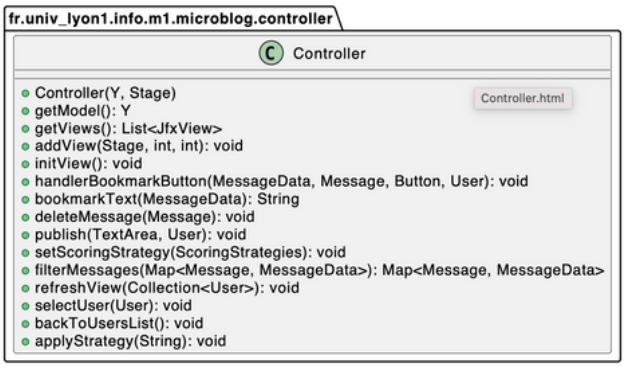
La vue ne contient aucune logique et agit comme une couche d'affichage pure. Toute la logique est délégué par des appels au contrôleur.



Le Contrôleur

Le contrôleur coordonne les interactions entre la Vue et le Modèle. C'est sur lui que repose la communication entre les deux composants.

Principales responsabilités du contrôleur
- Médiateur entre Vue et Modèle
- Gestion des stratégies dynamiques en appelant les méthodes du modèle
- Navigation dans l'application, il gère les changements de contexte

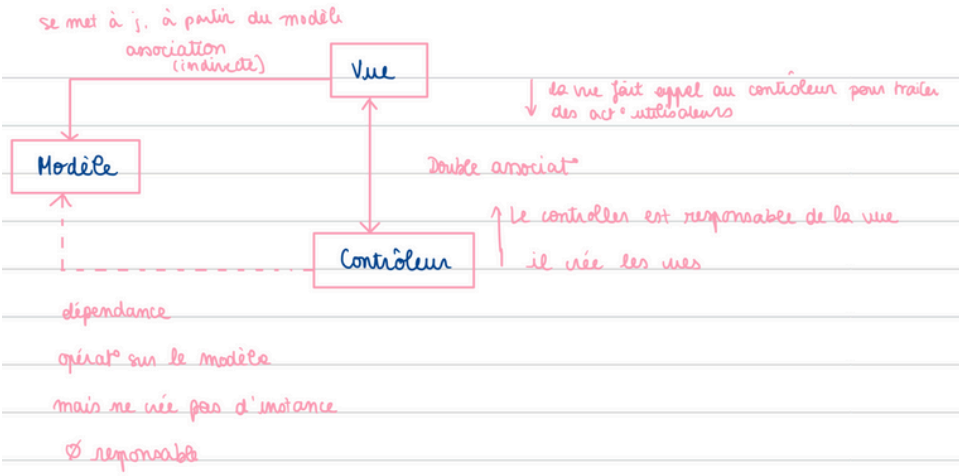


Exemple concret : un utilisateur veut ajouter un message

- Vue vers Contrôleur :
  - L'utilisateur va écrire son message dans l'espace réservé à cet effet.
  - Lorsqu'il appuie sur le bouton publier ou CTRL+Entrée, un évènement est déclenché et la Vue appelle la méthode **publish** du Contrôleur.

- Contrôleur vers Modèle :
  - Après avoir reçu l'événement de la Vue, le Contrôleur il appelle la méthode **addMessage** du Modèle pour appliquer les modifications nécessaires au Modèle,

- Contrôleur vers Vue :
  - Une fois que le Contrôleur a modifié le Modèle, il appelle **refreshView** sur la Vue pour mettre à jour l'affichage . Cela permet de garantir que les modifications dans les données du Modèle sont reflétées dans l'interface utilisateur.

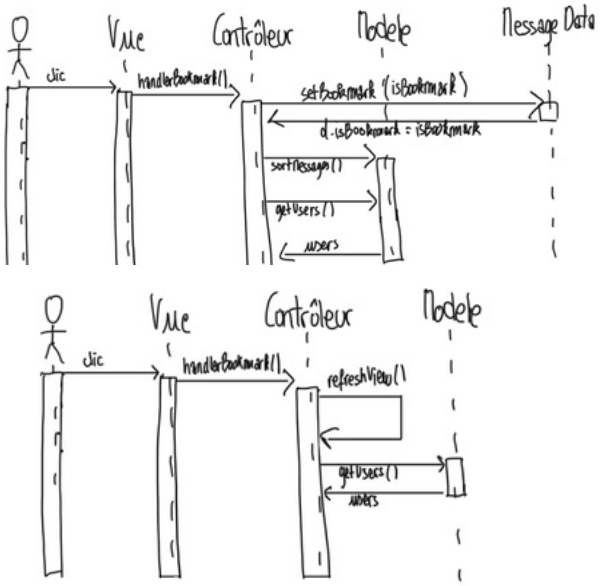


Pattern Controller

Le principe du pattern Controller est de ne pas donner trop de responsabilité au contrôleur qui ne sont pas nécessaires. Le contrôleur doit seulement faire l'intermédiaire entre le modèle et la vue, sans s'occuper de la logique métier ou de la manipulation directe des donnée. Prenons l'exemple de deux méthodes modifiés avec l'implémentation de ce pattern :

```java
/**
 * Handle bookmarks button.
 */
public void handlerBookmarkButton(final MessageData d, final Button bookButton) {
    boolean isBookmarked = d.isBookmarked();
    d.setBookmarked(!isBookmarked);
    bookButton.setText(bookmarkText(d));
    model.sortMessages();
    refreshView(model.getUsers());
}
```

Handler avant implémentation du pattern Controller




```java
/**
 * Handle bookmarks button.
 */
public void handlerBookmarkButton(final MessageData d, final Button bookButton) {
    model.bookmarkMessage(d);
    bookButton.setText(bookmarkText(d));
    refreshView(model.getUsers());
}
```

Handler après implémentation du pattern Controller

La méthode handlerBookmarkButton modifiais les propriétés des messages et mettais à jour l'interface utilisateur en changeant le texte du bouton. Cette logique à été plutôt déléguer au modèle (pour trier les messages) et à la vue (pour changer le texte du bouton), avec le contrôleur simplement relayant les événements.

Pour avoir un autre exemple, on peut regarder le code de la méthode addMessage. Cette méthode ajoutait un message à chaque utilisateur, puis triait les messages. Ce type de manipulation des données devrait idéalement se trouver dans le modèle. En effet, le rôle du contrôleur n'est pas de modifier les données se trouvant dans le modèle mais plutôt d'appeler le modèle a le faire.

```java
 * Add message m to all users.
 */
public void addMessage(final Message m, final User u) {
    // on ajoute le message dans la map de chaque user
    for (User user : model.getUsers()) {
        Map<Message, MessageData> userMessages = model.getUserMessageMap(user);
        userMessages.put(m, new MessageData(u));
    }
    model.sortMessages();
    refreshView(model.getUsers());
}
```

addMessage avant implémentation du pattern Controller

```java
/**
 * Add message m to all users.
 */
public void addMessage(final Message m, final User u) {
    // on ajoute le message dans la map de chaque user
    model.addMessage(m, u);
    refreshView(model.getUsers());
}
```

addMessage après implémentation du pattern Controller

Pattern Créateur

Dans le cadre de notre architecture, nous avons appliqué le pattern créateur pour attribuer la responsabilité de la création d'objets. La question était : qui doit être responsable de la création des vues dans notre application ?

Nous avons décidé que ce serait le contrôleur qui serait responsable de la création des vues. En effet, le contrôleur contient une liste de vues, ce qui nous a semblé la solution la plus pratique pour gérer l'ajout de vues qui fonctionnent parallèlement, tout en restant cohérent avec son rôle de médiateur décrit plus tôt.

En ce qui concerne la classe Y, elle est responsable de la création des instances User et Message. Cette classe gère les collections de ces objets et fournit des méthodes spécifiques pour leur instanciation, comme createUser et addMessage. Nous avons considéré la classe Y comme un "expert" dans la création des instances User et Message. Elle garantit la cohérence en s'assurant de l'unicité des noms d'utilisateurs et que le lien des messages à leurs métadonnées (date, auteur, etc.) se fasse bien.

Notre solution suit bien le principe du pattern créateur : la classe Y (et le contrôleur pour les vues) détient la responsabilité de créer des instances des objets qu'elle gère, car elle a accès aux données nécessaires à leur création et maîtrise les règles qui doivent être respectées pour celle-ci.

L'idée de cette approche était de confier la création des objets à une classe qui connaît bien ces objets, ce qui permet de centraliser cette tâche et d'éviter une gestion dispersée. Cela aide aussi à maintenir un faible couplage, ce qui réduit les dépendances et rend le code plus facile à maintenir et à réutiliser.

Pattern Strategy

Nous avons appliqué le pattern Strategy pour rendre la gestion des règles de scoring et de tri des messages flexible et dynamique. Ce pattern permet de désencapsuler les comportements dans des classes spécifiques donc fait sans modifier les classes existantes respectant ainsi le principe de faible couplage. Ceci facilitant la maintenance et l'évolutivité du code.

Nous avons utilisé cette approche pour les deux aspects suivants : **le calcul du score** et **le tri des messages**.

ScoringSystem

Le calcul du score est basé sur différentes stratégies, chacune représentant une règle de scoring distincte. Le ScoringSystem contient un tableau de ScoringStrategies, qui est l'interface commune à toutes les stratégies de scoring.

Chaque stratégie est responsable de calculer le score final d'un message en fonction de critères comme :

- Un bonus pour les messages bookmarkés
- Un malus pour les messages contenant des mots interdits
- Un bonus en fonction de la date de publication des messages

Cela permet d'ajouter facilement de nouvelles stratégies sans modifier le code existant, respectant ainsi le principe ouvert/fermé : le système est ouvert à l'extension (on peut ajouter de nouvelles stratégies) mais fermé à la modification du code déjà existant.

SortingStrategy

De même, pour le tri des messages, nous avons appliqué le pattern Strategy à travers un SortingStrategy, avec plusieurs stratégies de tri définies. Ces stratégies permettent à l'utilisateur de trier les messages selon différents critères

- Par pertinence (score)
- Par date

Cela permet à l'utilisateur de choisir dynamiquement la méthode de tri, tout en garantissant que le modèle reste découplé des comportements spécifiques de tri. Par exemple, lorsque l'option ordre chronologique est choisie, les messages sont triés exclusivement selon leur date, et les messages bookmarkés ne sont plus automatiquement affichés en premier, même s'ils sont définis comme "pertinents". Ce comportement a été intentionnellement choisi.

Dans notre application

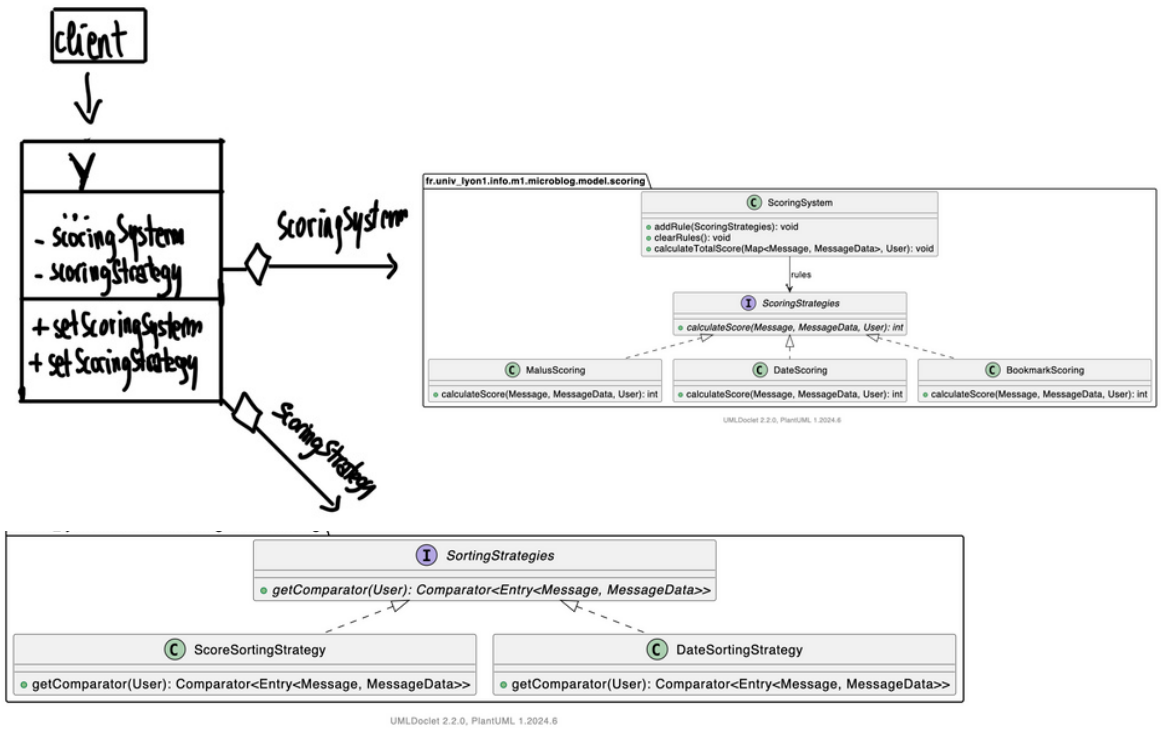
Nous avons combiné tout ça pour gérer dynamiquement les règles de tri et de scoring des messages dans notre application. Cette combinaison permet à chacun des utilisateurs de choisir entre trois options distinctes pour le tri des messages, en fonction des critères souhaités, tout en garantissant une flexibilité et une évolutivité du système.

Les trois options de tri et de scoring sont les suivantes :

- Par ordre chronologique : Applique une stratégie de tri basée uniquement sur la date des messages (DateSortingStrategy).
- Messages les plus pertinents récents : Combine plusieurs stratégies de tri et de scoring, en utilisant ScoreSortingStrategy pour trier les messages par pertinence, et en ajoutant les règles de scoring DateScoring, MalusScoring, et BookmarkScoring.
- Messages les plus pertinents : Utilise également ScoreSortingStrategy, mais avec seulement MalusScoring et BookmarkScoring.

Nous avons choisi de n'avoir qu'une seule méthode de tri par stratégie, contrairement au scoring où il peut y avoir plusieurs règles appliquées simultanément. Cela est logique pour nous, car le tri des messages ne nécessite qu'un seul critère de classement à la fois, contrairement au système de scoring où plusieurs règles peuvent s'appliquer et interagir ensemble.

À travers cette approche, nous avons permis à notre application de proposer des critères de tri flexibles, sans modifier les classes existantes. Chaque option est gérée par une méthode dans le modèle, qui configure la stratégie de tri et les stratégies de scoring associées, pour l'utilisateur connecté actuellement sur l'application.



03

TESTS

Tests

Des tests manuels ont dû être réalisés pour valider les fonctionnalités principales, en particulier toutes celles de la vue, non couvertes par des tests automatisés, mais aussi certaines du contrôleur. Ces tests ont pour but d'assurer que l'application fonctionne correctement du point de vue de l'utilisateur final, en veillant à ce que les comportements attendus soient respectés.

Vue

- Affichage des utilisateurs : Vérification de l'affichage des utilisateurs sous forme de boutons cliquables, permettant de sélectionner un utilisateur et de charger son fil d'actualités.
- Interaction avec le fil d'actualité : Confirmation que le fil d'actualité se charge correctement, que l'ajout de messages est possible via un champ de saisie et que la vue se met à jour après l'envoi d'un message.
- Fonctionnalité "Bookmark" et "Delete" : Vérification du bon fonctionnement des boutons "Bookmark" et "Delete", avec une mise à jour immédiate de l'interface.
- Gestion des préférences de tri : Test pour s'assurer que les messages sont triés selon le critère sélectionné par l'utilisateur, et que la vue se met à jour en conséquence.
- Déconnexion et retour à la liste des utilisateurs : Vérification du bouton "Logout" permettant de se déconnecter fait bien revenir à la liste des utilisateurs.

Contrôleur

- Méthode refreshView : Test de mise à jour correcte de la vue selon l'utilisateur sélectionné, affichant ses messages ou la liste des utilisateurs si aucun n'est sélectionné.
- Méthode filterMessages : Vérification que seuls les messages visibles (ceux qui passent la méthode canBeSeen du modèle) sont affichés.
- Méthode selectUser : Validation que l'utilisateur sélectionné est correctement mis à jour et son fil d'actualité affiché.
- Méthode backToUsersList : Confirmation que l'utilisateur courant est réinitialisé à null et que la liste des utilisateurs est affichée après une déconnexion.

Ces tests manuels couvrent les fonctionnalités non testées automatiquement, en vérifiant le bon fonctionnement de l'interface et des interactions.

04

ÉTHIQUE

Éthique

Nous vivons dans un monde où avoir un téléphone est devenu une norme, jeunes, vieux, femmes, hommes, le monde entier est donc concerné par les algorithmes de recommandations. Allant de la simple recommandation de vidéos à la personnalisation de publicité en passant par la suggestion d'amis à suivre, ces algorithmes influencent subtilement nos choix, nos goûts, nos fréquentations et dans certains cas vont même jusqu'à entraver nos valeurs.
Mais, à qui profitent réellement ces algorithmes ? Quels sont les avantages et les risques de ceux-ci ? Et enfin comment limiter les risques et quels méthodes avons-nous utilisées dans notre projet pour se faire ?

Tout d'abord tout n'est pas à prendre ou à laisser, car les premières personnes à bénéficier de ces algorithmes sont les utilisateurs eux-mêmes. En effet, bien que ces algorithmes aient des buts variés (commerciaux, politique, etc...), leur but principal reste la personnalisation des contenus afin de convenir au plus de monde. Ainsi l'expérience utilisateur est plus agréable et plus personnelle.
Il y a toujours une face cachée à l'iceberg, ces algorithmes ne font pas défaut à la règle, et ne profitent pas uniquement aux utilisateurs. Les personnes ayant accès à ce genre d'algorithmes ont un grand pouvoir entre leurs mains, car ils ont le pouvoir d'influencer un nombre très important de personnes à leur causes.
C'est quelque chose que nous avons pu constater cet été pendant les élections législatives. En effet, les personnes n'ayant pas de contenus politisés, par exemple sur la plateforme TiKTok, se voyaient recommander des comptes officiels de candidats. Bloquant ainsi l'utilisateur dans une bulle de filtre sans forcement s'en rendre compte, tout en favorisant certaines voix et en manipulant subtilement les opinions.

Cela soulève donc la question des avantages, mais aussi des risques de ces algorithmes. Si, d'un côté, ils apportent des bénéfices considérables à leurs utilisateurs, ils peuvent aussi être détournés à des fins qui ne servent pas toujours le bien-être de l'ensemble de la société, créant ainsi des dangers qu'il est important d'examiner.

Ces systèmes présentent des bénéfices indéniables. Ils permettent, par exemple, de gagner du temps : une utilisatrice végane cherchant une recette obtiendra rapidement des résultats adaptés à ses préférences alimentaires.
La personnalisation de contenu est bien sur très utile aussi pour les entreprises afin que l'utilisateur reste le plus de temps sur son application. De leur cotés les utilisateurs peuvent, à l'aide de ces algorithmes, se sentir parfois compris, intégrés dans une communauté, ce qui peut réduire leur sentiment de solitude.
D'autre part, ce sont des algorithmes, donc des machines, il est donc très facile pour des personnes issues des générations vivant avec Internet depuis toujours, de comprendre cet algorithme et de l'utiliser à leurs avantages. C'est d'ailleurs ainsi que ce sont créés les nouveaux métiers de créateur de contenus etc.

Cependant de nombreux avantages sont à nuancer. En effet, la personnalisation de contenu peut paraitre attirante mais cela enferme les individus dans des cases qu'ils semblent avoir choisies. Nous savons que l'humain est en constante évolution, nous ne sommes pas fait pour faire les même activités, aimer les mêmes choses, parler aux mêmes personnes tout le temps.
Ce besoin de changement n'est pas pris en compte par les algorithmes. Des témoignages venant dune étude menée par Amnesty international sur les dangers de l'application TikTok ont prouvé que des personnes dépressifs des besoins différents chaque jour. Par exemple, il y aura des jours ou cette personne aura besoin de réconfort, d'autres de motivation, d'autres de conseils etc.
Cependant l'algorithme ayant ciblé sa maladie et donc son état mental va le bloquer dans une spirale de vidéos maussade de personnes broyant du noir qui pourraient le pousser à commettre l'irréparable. Cela s'est vu dans la tendance qu'il y avait à une époque récente de se taillader les veines et de le filmer, ou encore le jeu de la tomate qui consistait à ne plus respirer jusqu'à devenir rouge. Ces vidéos n'étaient recommandé qu'aux personnes souffrant de troubles psychologiques et/ou suicidaires.
Moins grave mais il est important de ne pas le sous estimer, est le problème de perte de curiosité et d'ouverture d'esprit. Évidemment ce problème coule de source lorsqu' nous parlons de bulles de filtres.
Ne pas être exposé à des personnes différentes, pensant, agissant, ou vivant différemment de nous et que toute la communauté que ces algorithmes ont construit autour de nous, ne nous poussent plus à questionner, plus rien découvrir, plus rien... et se crée ce sentiment de ne pas vouloir découvrir. Ce qui peut découler bien sûr de racisme ou bien juste de baisse de prise de risques, une constante envie de rester dans sa zone de confort.

Enfin, face à ces risques, il est nécessaire de chercher des moyens de les limiter tout en préservant les avantages de ces technologies.

De nombreuses loi sont entrées en vigueur dans de nombreux pays pout limiter ces risques là mais rien ne semble changer aux yeux de tous, les enjeux économiques sociaux et politiques sont trop grands.
Dans notre projet, de manière ultra simpliste nous avons implémenté une méthode de filtrage basique commune à tous. Afin que peu importe les recommandations de chacun, certains sujets ne soient pas discutés car, non, tout n'est pas bon à dire par n'importe qui. Ne pas limiter l'usage de certains mots pourraient conduire à de la désinformation ou à des propos injurieux, racistes, misogynes etc. La liberté des uns s'arrêtent la ou commencent celles des autres, et cela est d'autant vrai quant à la liberté d'expression. Le système de scoring, quant a lui, a été implémenté afin de garder l'avantage des algorithmes de recommandations étant, la personnalisation de contenus.

04

CONCLUSION

Conclusion

Bilan et enseignements

Ce projet nous a permis de développer de nombreuses compétences, sur le plan technique et éthique, tout en nous confrontant à plusieurs défis.

D'un point de vue technique, une des grandes découvertes dans ce projet a été l'utilisation des tests automatisés avec JUnit 5, Hamcrest et TestFX. Ces tests nous ont offert un retour rapide sur nos choix de développement, nous permettant ainsi de détecter si il y avait des dysfonctionnements, et tester les cas un peu plus limites. C'est vraiment cela nous a ainsi permis de garantir la stabilité du code au fur et à mesure.
D'autre part le travail de refactorisation du code a été particulièrement formateur : partir d'un squelette pour l'adapter à l'architecture MVC nous a appris à mieux organiser notre travail et à appliquer des principes solides de programmation orientée objet.
L'implémentation d'une interface de stratégie nous a permis de rendre l'application plus flexible, tout en nous forçant à maintenir un code propre et bien structuré.
Enfin, ce projet nous a aussi amenés à réfléchir aux implications éthiques des systèmes de recommandation, ce qui a enrichi notre vision du développement logiciel.

Cependant, ce n'était pas sans difficultés. La refactorisation pour respecter l'architecture MVC a parfois été complexe, surtout lorsqu'il s'agissait d'ajouter de nouvelles fonctionnalités sans perturber l'existant. Il a fallu faire preuve de rigueur pour maintenir un code propre et lisible, éviter les duplications et garantir une bonne modularité. Cela a été un défi constant, mais essentiel pour garder une cohérence dans le projet.

En conclusion, ce projet a consolidé nos compétences en développement tout en nous sensibilisant aux enjeux éthiques du numérique. Nous avons appris à appliquer des principes de conception solides, à mettre en place des tests automatisés efficaces et à réfléchir à l'impact de nos choix techniques, tant sur l'expérience utilisateur que sur la société.

Sources

- Amnesty International. Poussés vers les ténèbres : Rapport sur les dangers de l'application TikTok. Disponible en ligne : https://amnestyfr.cdn.prismic.io/amnestyfr/f7a52b32-793f-4e0a-a83a-737436f415f3_Pouss%C3%A9%C2%B7e%C2%B7s+vers+les+t%C3%A9n%C3%A8bres_rapport_Tiktok_Amnesty_International_FR.pdf

- Odil. Quel rôle jouent les algorithmes de recommandation dans notre écosystème informationnel ? Disponible en ligne : https://odil.org/quel-role-jouent-les-algorithmes-de-recommandation-dans-notre-ecosysteme-informationnel/

- Le Monde. Soft power, espionnage, données personnelles : que reproche-t-on à TikTok ? Publié le 4 mars 2023. Disponible en ligne : https://www.lemonde.fr/pixels/article/2023/03/04/soft-power-espionnage-donnees-personnelles-que-reproche-t-on-a-tiktok_6164148_4408996.html

- Amnesty International France. Vidéo sur les algorithmes de TikTok (YouTube). Disponible en ligne : https://youtu.be/-pOWdBpVR3s?si=Qx5oScNqayoyB5F4