Projet Mif01

Oussama Benaziz p2007990 - Jules bonhotal p2003042

December 2023

1 Introduction

Ce projet se concentre sur le développement d'un chatbot simple, offrant l'opportunité d'implémenter différents patterns vus en cours. Ces derniers sont présentés en détail dans ce document.

Le code que nous avons écrit est accessible dans le répertoire : "mif01/eliza-gpt/src/main". Les tests automatiques sont disponibles dans : "mif01/eliza-gpt/src/main". Une version non corrigée par ChatGPT de ce rapport peut être consultée ici : "mif01/Rapport/TextOriginal.txt". Le contexte de l'existence de ce document est décris dans la partie 3.3.

2 Design Patterns

2.1 MVC et Observer

Le pattern MVC (cf. figure 1) a été implémenté dans ce projet, conformément aux consignes. Cependant, son adoption offre également l'avantage de mieux séparer les responsabilités des classes, en accord avec le principe de la responsabilité unique (S de SOLID). Dans notre projet, ce pattern repose principalement sur l'implémentation de la classe `MessageStorage`, qui agit en tant qu'observable. Grâce à l'interface `Observer` utilisée par les acteurs nécessitant des informations sur l'état du stockage des messages, cela permet une inversion de dépendance (D de SOLID).

Dans le cadre de ce projet, le pattern Observer a été légèrement modifié en faisant en sorte que l'observable n'implémente pas d'interface observable. Cette modification a été apportée car il ne semblait pas nécessaire d'ajouter de nouveaux objets observables dans le cadre de ce que nous avions prévu d'implémenter.

À l'origine, la classe `Message` était une sous-classe de la classe `MessageStorage`. Cependant, cela posait le problème que toutes les autres classes manipulant des messages devaient donc dépendre de la classe `MessageStorage`, même si elles n'en avaient pas nécessairement besoin. Par conséquent, nous avons décidé de rendre la classe `Message` indépendante, afin de réduire les dépendances (faible couplage) et de mieux respecter le principe de la responsabilité unique.

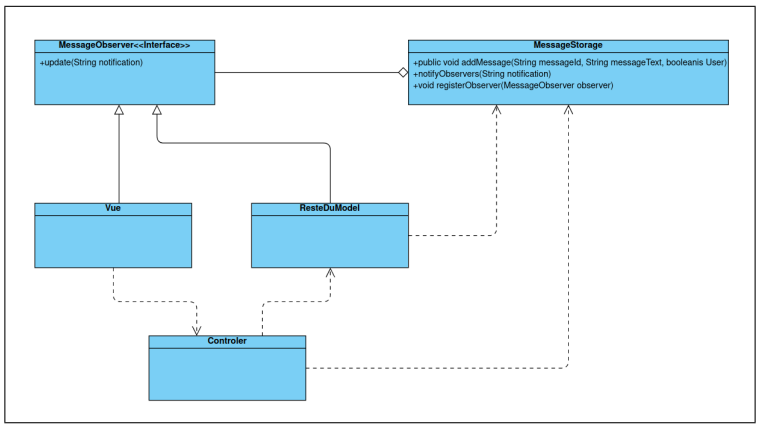

Figure 1: Pattern MVC et Observer.

2.2 DAO (Data Access Object)

Initialement, notre approche pour conjuguer les verbes dans le code reposait sur trois méthodes distinctes. Les verbes du premier groupe nécessitaient l'ajout d'un z à la fin du verbe, qui était initialement conjugué à la première personne du singulier. Pour les verbes du deuxième groupe, l'ajout de sez était requis. Quant aux verbes du troisième groupe, nous utilisions une liste prédéfinie de six verbes avec leur conjugaison. Cependant, afin d'améliorer cette méthode et d'obtenir une liste de verbes plus exhaustive à partir d'un fichier CSV, nous avons décidé d'intégrer le pattern DAO dans notre projet. Cette démarche a conduit la classe VerbDAO à se spécialiser exclusivement dans l'accès aux données des verbes, contribuant ainsi de manière significative à la lisibilité du code. Cette approche renforce de manière notable la modularité du système en isolant efficacement la logique d'accès aux verbes. Les modifications apportées à la source de données n'ont pas d'impact direct sur les autres composants du système. Cette séparation facilite l'entretien du code et prépare le terrain pour des évolutions futures, telles que la possibilité d'ajouter dynamiquement de nouveaux verbes sans altérer le code source principal. Une fonctionnalité particulièrement avantageuse pour une liste de verbes évolutive.

[Insérez diagrammes ici]

2.3 Strategy

Pour mettre en œuvre les stratégies de recherche, nous avons choisi le pattern Strategy (cf. figure 2). Ce choix s'explique notamment par sa facilité d'ajout de nouvelles stratégies de recherche sans apporter de modifications importantes au reste du code. Il suffit de s'assurer que la nouvelle stratégie implémente l'interface `SearchStrategy` et d'ajouter une instance de la nouvelle stratégie dans la classe utilitaire qui retourne une liste de toutes les stratégies. De plus, il limite le couplage entre les différentes stratégies qui sont donc plus

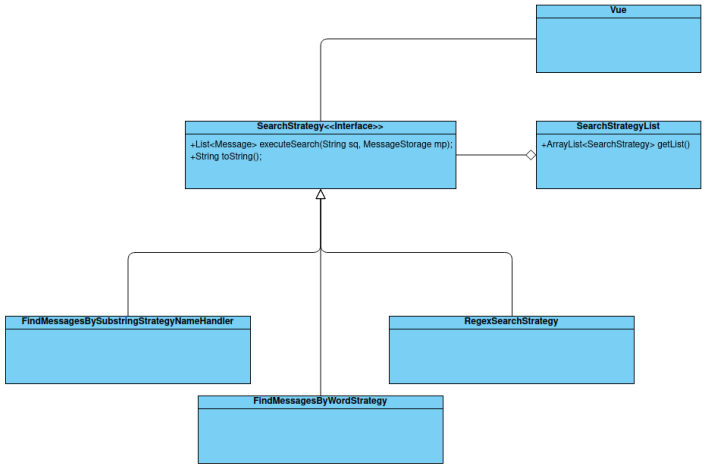

Figure 2: Pattern Strategies.

indépendantes les unes des autres.En d'autres termes, ce pattern permet d'être ouvert à l'extension tout en restant fermé à la modification, car il n'est pas nécessaire de modifier le code existant pour ajouter de nouvelles stratégies (O de SOLID). Nos stratégies sont également contraintes par leur interface à implémenter leur propre version de la fonction `toString()`, héritée de la classe `Object` de Java, qui est utilisée pour l'affichage.

Il est intéressant de noter que des instances de stratégies sont présentes dans la vue, mais leur exécution se réalise dans le modèle via le contrôleur, afin d'éviter une dépendance directe de la vue au contrôleur. Ce choix a été fait pour respecter la consigne de l'énoncé, qui spécifiait que l'affichage des noms des recherches devait se faire à travers la fonction `toString()`. Dans la mesure où l'exécution de la recherche dans la vue ne nécessite pas de dépendance au modèle, cela ne compromet pas le principe du modèle-vue-contrôleur (MVC). Même s'il existait probablement des implémentations ne reposant pas sur `toString()` qui n'auraient pas conduit la vue à posséder une liste d'instances de stratégies.

2.4 Chaine de résponsabiliter

Pour la réimplémentation des règles de réponse fournies par Eliza, nous avons choisi le pattern Chaîne de Responsabilité (et Builder, voir 2.5). Ce choix s'est justifié car il correspondait à la fonctionnalité que nous cherchions à implémenter, tout en permettant d'ajouter facilement de nouvelles règles. De plus, les différentes règles présentent un faible couplage entre elles, ce qui facilite la modification individuelle sans impacter le comportement des autres règles.

Nous avons préféré la chaîne de responsabilité à l'arbre de responsabilité, car dans le cadre de ce projet, un traitement séquentiel des règles de réponse était suffisant. Cependant, il est intéressant de noter qu'en fonction de l'implémentation

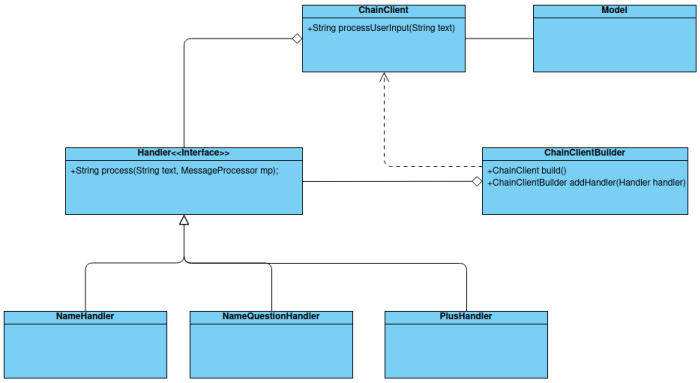

Figure 3: Pattern Chaine de responsabiliter et Builder.

de l'arbre de responsabilité, il aurait pu permettre de placer plusieurs règles au même niveau de priorité ou de réaliser des traitements plus complexes avec des ensembles de règles à valider pour obtenir une réponse spécifique.

2.5 Builder

Malheureusement, le pattern Chaîne de Responsabilité a tendance à avoir une initialisation peu pratique. En effet, avec ce pattern, il aurait fallu créer une chaîne de nouveaux objets comme suit :
`new Chaine1(new Chaine2(new Chaine3(...)))`, ce qui rend l'édition de l'ordre des règles ou leur ajout non idéal. Pour remédier à cela, nous avons décidé d'utiliser le pattern Builder. Celui-ci permet de faciliter la création d'une chaîne de responsabilité.

Grâce au pattern Builder, l'édition d'un objet `ChainClient` devient plus simple, et cela permet également de cacher l'implémentation des chaînes de responsabilité derrière un niveau d'abstraction.

3 Éthique

3.0.1 droite d'auteurs

L'une des problématiques majeures liées à l'intelligence artificielle concerne la question des droits d'auteur, à la fois dans le contexte des productions générées par l'IA et en ce qui concerne les données utilisées pour l'entraînement des IA basées sur des réseaux neuronaux. En effet, de nombreux producteurs de contenus tels que des journalistes ou des auteurs ont vivement protesté contre le développement de ces IA, arguant que celles-ci utilisaient leur travail pour s'entraîner sans qu'ils soient compensés de quelque manière que ce soit. En effet, un humain s'inspirant d'un texte pour en écrire un autre serait tenu de citer

les sources qu'il a utilisées, mais la plupart des IA conversationnelles, telles que ChatGPT, ne sont pas capables de le faire.

Une question légitime se pose également quant à la propriété des droits d'auteur sur le travail collaboratif avec une IA, par example si les droits peuvent revenir en partie au studio finançant l'utilisation de l'IA, voire à l'entreprise qui a développé l'IA en question. C'est en grande partie face à cette problématique que la Guilde des auteurs américains (la WGA) a mené une grève pour obtenir des régulations concernant cette question, entre autres. Ils ont notamment obtenu que les scénarios utilisés pour les séries ou films ne puissent pas être entièrement générés par une IA, ainsi que des restrictions sur l'utilisation de l'IA dans l'industrie du divertissement de manière générale. De plus, ils ont réussi à faire en sorte que le créateur du texte, et non le studio, détienne les droits d'auteur sur un texte qui aurait été produit en partie grâce à une IA, indépendamment du degré d'implication de l'IA.

Sources www.snac.fr/alerter-intelligence-artificielle
www.theguardian.com/culture/hollywood-writers-strike

3.1 Interaction Avec des Personnes Vulnérables

La plupart des IA conversationnelles possèdent des comportements prédéfinis visant notamment à maintenir la conversation dans le domaine de la bienséance. Malheureusement, ces protections sont souvent faciles à contourner avec des prompts spécifiquement conçus à cet effet, voire parfois en répétant une idée à plusieurs reprises. C'est ce qui est arrivé à un homme belge nommé Pierre, qui conversait avec le chatbot "Eliza" disponible sur l'application Chai. Pierre partageait ses pensées suicidaires avec le bot, et bien que le bot ait initialement tenté de le dissuader, il a fini par l'encourager au suicide lorsque Pierre a continué à parler de ses idées. Bien qu'il soit difficile de déterminer si cette conversation a eu un impact sur la décision de Pierre de se suicider, cet incident a soulevé de nombreuses questions concernant les interactions des chatbots avec des personnes fragiles au sein de l'entreprise qui avait développé ce bot.

Il est compliqué pour les développeurs d'IA de contrôler la réaction de leurs modèles face à toutes les situations, et il sera sans doute jamais possible d'empêcher complètement une IA de générer des contenus pouvant nuire à des personnes fragiles. Cependant, la plupart des entreprises d'IA prennent des mesures pour éviter ce genre de situations. Par exemple, l'entreprise développant Elisa a mis en place une surcouche de vérification qui offre un lien vers des sites de soutien pour les personnes suicidaires si le sujet est abordé.

La prévention de la production de contenu dangereux ou immoral demeure, et restera sans doute, une problématique importante du domaine dans le futur, car il ne semble pas y avoir de solution définitive à ce problème pour l'instant.

Sources www.vice.com/suicide-talking-with-ai

3.2 Éthique dans notre projet

Aucune mesure de prévention n'a été mise en place dans notre projet, car il n'est pas destiné à une utilisation publique. Cependant, certaines de ces mesures de prévention pourraient être pertinentes si nous venions à reprendre le développement de ce projet dans le but de le publier.

Notamment, l'implémentation de messages prédéfinis autour de sujets sensibles tels que la dépression ou les idées suicidaires pourrait être envisagée. Notre projet pourrait chercher à détecter ces sujets et fournir des réponses prédéfinies accompagnées de ressources pour aider les utilisateurs dans ces situations.

3.3 Notes sur l'utilisation de l'IA dans ce rapport

Pour cette section, tout comme le reste du rapport, ChatGPT a été utilisé pour corriger et reformuler du texte brut que nous lui fournissions. Grâce à cette méthode de travail, j'ai (Jules Bonhotal) pu compenser des défauts de clarté et d'orthographe, notamment causés par ma dyslexie.

Les prompts utilisés étaient du format : "Tu peux corriger et reformuler ce texte qui fait partie d'un document LaTeX ?

Le texte : 'le texte en question.'"

Si vous voulez voir les textes originaux sans correction, vous pourrez les retrouver dans le fichier Rapport/TextOriginal.txt (mais attention, ça pique les yeux).

4 Tests

4.1 Tests Manuels

Pour les tests manuels, nous avons généralement effectué des clics sur tous les boutons de l'interface pour vérifier s'ils provoquaient une réaction, et si celle-ci était conforme aux attentes.

Pour évaluer les réponses, nous avons cherché à obtenir toutes les réponses possibles en saisissant successivement des entrées censées générer des réponses spécifiques. Pour tester les fonctionnalités de recherche, nous avons entré des requêtes de base, suivies d'autres qui ne devraient fonctionner qu'avec certains types de recherche.

Par exemple, en générant des réponses aléatoires du bot (en envoyant des messages vides de manière répétée), "beau" devrait fonctionner avec toutes les recherches, "bea" ne devrait fonctionner qu'avec SubString et Regex, et enfin, "be." ne devrait fonctionner qu'avec Regex.

4.2 Tests Automatiques

Les tests automatiques peuvent être exécutés via la commande "mvn test". Ils évaluent principalement les réponses du bot à des entrées prédéfinies.