RAPPORT - MIF01
ELIZAGPT

Illustration réalisée à l'aide de *Dall-E*

SIONI Farès

THINOT Edouard

17/12/2023

MIF01 - Rapport de projet - ElizaGPT

Présentation du projet

Ce projet s'inscrit dans l'UE Gestion de Projet et Génie Logiciel du M1 informatique de l'UCBL, mettant l'accent sur la gestion de projets informatiques, les méthodes agiles, la spécification logicielle, la conception avec les *design patterns*[1], et la mise en place de tests.

Ce projet s'inscrit dans une approche moderne du langage Java, mettant en oeuvre une base de code structurée nécessitant un découpage en plusieurs fichiers selon des rôles spécifiques. La conception de cette architecture repose sur l'application du *pattern MVC*[2], conforme aux exigences du projet.

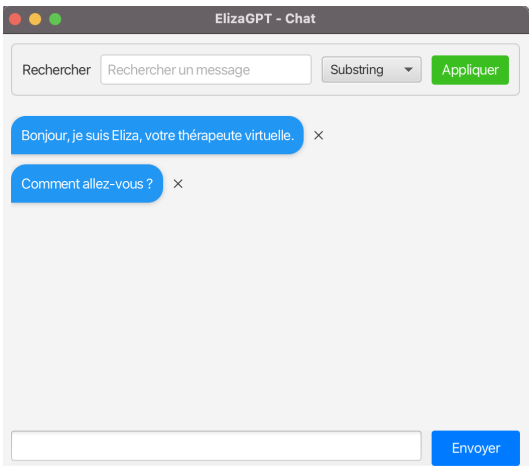
Les outils utilisés dans le cadre de ce projet comprennent :

- **Maven** : Employé pour la gestion des dépendances ainsi que pour orchestrer les différents cycles de vie du projet, notamment la compilation et les tests.
- **JUnit/Hamcrest** : Ces frameworks sont intégrés pour simplifier l'implémentation des tests, assurant ainsi la robustesse et la fiabilité du code développé.
- **Checkstyle** : Utilisé pour émettre des avertissements en accord avec des règles de formatage du code. Cette approche garantit une cohérence et une lisibilité accrues de l'ensemble du code source.
- **JavaFX** : Pour la partie visuelle du projet, il offre une solution moderne pour le développement d'interfaces graphiques interactives, contribuant ainsi à une expérience utilisateur optimale.

L'ensemble de ces choix technologiques reflète une démarche résolument orientée vers les bonnes pratiques de développement logiciel, la maintenabilité du code et l'optimisation de l'expérience utilisateur.

L'objectif central de ce projet est de mettre en pratique les principes fondamentaux du *design pattern* et de la gestion de projet.

L'utilisation des Merge Requests[3] nous a été très utile pour la collaboration au sein de notre binôme. Elles nous ont aidé à améliorer la qualité du code en discutant des solutions mises en place. Bien que non essentielles dans le cadre de ce projet de taille modeste, cette approche semble indispensable pour des projets plus vastes.



Interface finale de l'application

[1] En français : "patrons de conception"
[2] MVC Design Pattern - GeeksforGeeks
[3] En français : 'demandes de fusion"

Conception de l'application

Patrons de conception

Dans notre projet, outre l'implémentation incontournable du *modèle-vue-contrôleur* (*MVC*) qui assure une architecture robuste et modulaire, nous avons également judicieusement intégré plusieurs autres patrons de conception pour optimiser la structure globale de notre application.

En incorporant ces *design patterns* de manière réfléchie, nous avons pu non seulement répondre aux exigences fonctionnelles du projet, mais aussi élever la qualité de notre code en favorisant la maintenabilité et l'extensibilité. Nous allons aborder quelques patrons de conceptions implémentés dans le projet.

Pattern Factory

Dans le cadre de ce projet, nous avons mis en oeuvre le patron de conception *Factory*[4] dans deux contextes distincts pour améliorer la gestion de la création d'objets.

La première implémentation est la classe *ResearchStrategyFactory*, responsable de la création d'instances de différentes stratégies de recherche, telles que *SubstringResearch*, *RegexpResearch* et *WordResearch*.

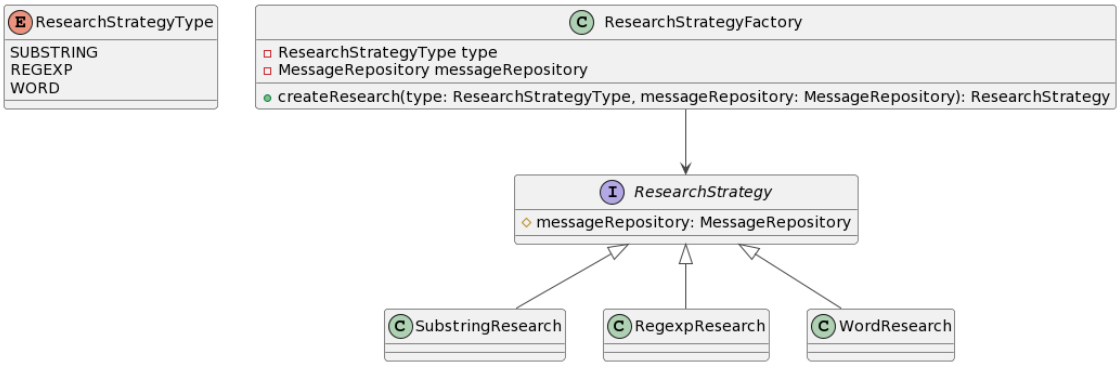


Diagramme de classe : Factory pour les stratégies de recherche

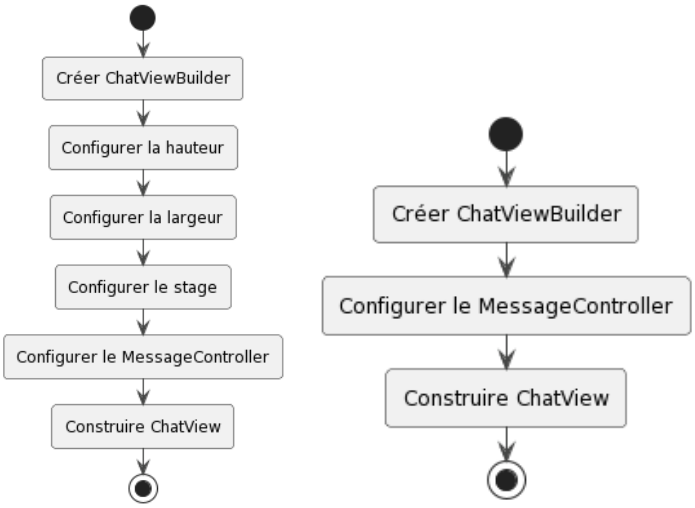
De la même manière, l'autre implémentation, avec la classe *MessageFactory* sert à créer des objets de type *Message* en fonction du texte du message et de l'auteur.

Ces deux utilisations du patron *Factory* contribuent à rendre la logique de création d'objets plus claire. Ce dernier offrant la possibilité de retourner différents types d'objets en fonction des besoins et d'apporter une modularité pour l'ajout d'autres types à l'avenir.

[4] Factory design pattern - GeeksforGeeks

Pattern Builder

Dans le prolongement du patron de conception *Factory*, nous avons également appliqué le patron *Builder*[5] dans la classe *ChatViewBuilder*. Ce dernier se focalise sur la création et la configuration d'objets complexes, notamment ceux aux constructeurs verbeux. Contrairement au patron *Factory*, qui se concentre sur la création d'objets en gérant le polymorphisme, le *Builder* intervient davantage dans la mise en place initiale d'objets en s'abstrayant de leur interface.



Deux scénarios d'utilisation du ChatViewBuilder (valeurs par défaut)

Dans le contexte de la classe *ChatViewBuilder*, notre objectif est de simplifier le processus de construction de l'interface utilisateur de la classe *ChatView*. En permettant la configuration flexible des propriétés telles que la taille de la fenêtre et le contrôleur de messages. Le Builder propose une alternative élégante pour créer des objets. Il offre plus de souplesse au client dans la spécification des paramètres, que ce soit dans leur ordre ou avec la présence de valeurs par défaut. Cette approche s'inscrit dans une logique de gestion des objets complexes.

Pattern Observer

Pour compléter l'utilisation des patrons de conception dans notre projet, nous avons intégré le pattern *Observer*[6] pour assurer une communication efficace entre différentes parties de notre application, notamment la gestion des messages dans le chat. Ce *pattern* permet à plusieurs objets souscripteurs d'être informés des changements d'état d'un objet diffuseur, tout en maintenant un faible couplage entre eux.

[5] Builder Design Pattern - GeeksforGeeks
[6] Observer Pattern - GeeksforGeeks

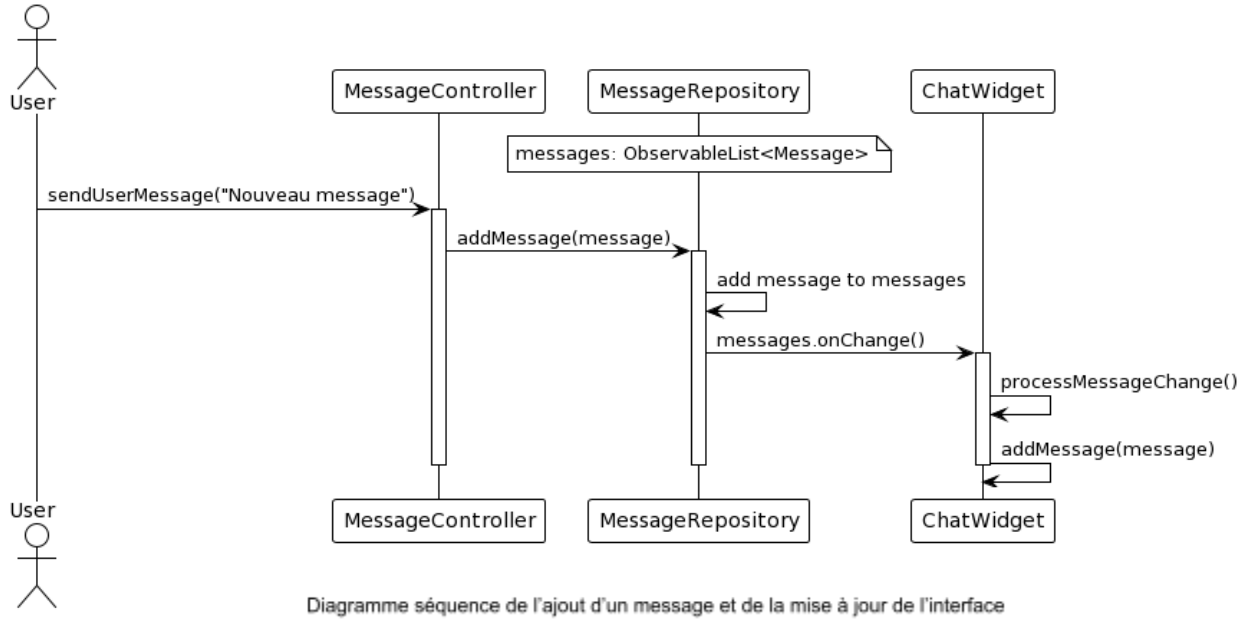


Diagramme séquence de l'ajout d'un message et de la mise à jour de l'interface

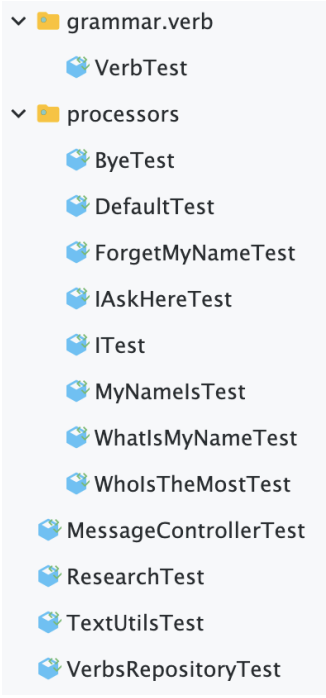
Dans le contexte de notre application, le *MessageRepository* agit en tant que sujet observé, et le *ChatWidget* fonctionne en tant qu'observateur. Lorsqu'un message est ajouté ou supprimé dans le *MessageRepository*, le *ChatWidget* est notifié et met à jour son affichage en conséquence.

Tests

Une grande partie du modèle a été testée, couvrant toutes les classes et la plupart des méthodes. Cependant, la vue n'a pas été testée. Bien que cela soit faisable et utile, bien que complexe, la priorité a été de garantir le bon fonctionnement interne de l'application.

Element	Class, %	Method, %	Line, %
fr.univ_lyon1.info.m1.elizagpt	72% (35/48)	54% (105/191)	51% (276/534)
model	100% (34/34)	86% (99/115)	86% (267/308)
grammar	100% (13/13)	100% (43/43)	98% (104/106)
message	100% (8/8)	77% (17/22)	78% (26/33)
research	100% (8/8)	66% (20/30)	74% (57/77)
utils	100% (4/4)	100% (16/16)	87% (74/85)
Model	100% (1/1)	75% (3/4)	85% (6/7)
controller	100% (1/1)	60% (6/10)	60% (9/15)
MessageController	100% (1/1)	60% (6/10)	60% (9/15)
view	0% (0/12)	0% (0/64)	0% (0/200)

Test coverage du projet



Liste des fichiers de tests

Ces tests se sont avérés utiles. En effet, après le développement d'une fonctionnalité et la validation de son bon fonctionnement (notamment à l'aide de ces mêmes tests), le code en lui-même n'est jamais parfait. C'est pourquoi nous en avons profité pour améliorer sa qualité avec plusieurs itérations de refactoring.

```java
@Test
void testRemoveMessage() {
    // Given
    String userMessage = "Bonjour, Eliza.";
    ObservableList<Message> messages = messageController.getMessages();
    messageController.sendUserMessage(userMessage);
    int nbMessages = messages.size();
    Message message = messages.get(nbMessages - 2);

    // When
    messageController.removeMessage(message);

    // Then
    assertEquals( expected: nbMessages - 1, messages.size());
    assertTrue(messages.stream().noneMatch(msg -> msg.getText().equals(userMessage)));
}
```

Test de la suppression d'un message

L'implémentation des tests suit la structure *Given / When / Then*, ce qui permet de séparer les différentes étapes.

Dans l'exemple donné, nous initialisons d'abord le contexte en envoyant un message. Nous nous assurons de récupérer sa référence ainsi que le nombre de messages présents à cet instant.

Ensuite, nous procédons à l'action à tester, qui est la suppression de ce message.

Enfin, nous évaluons les conséquences de nos actions. Dans ce test particulier, l'objectif est de vérifier que le nombre de messages a bien été décrémenté d'une unité et qu'il n'y a aucun message ayant le même contenu que celui supprimé.

La réalisation de ces tests nous a permis de valider l'implémentation des fonctionnalités tout en agissant comme un garde-fou face à la refactorisation.

Éthique

Introduction

Cette partie a pour but de développer d'autres points de vue que la présentation technique de notre projet. De nos jours, il est crucial de ne pas sous-estimer les considérations éthiques qui entourent un projet.

Problématiques apportées par les IA conversationnelles

Cela fait maintenant plus d'un an que les IA conversationnelles ont vraiment commencé à être démocratisées et populaires. Cette fulgurance s'accompagne encore aujourd'hui de controverses et de zones d'ombre. Cette partie donne une liste non exhaustive des problématiques qui ont émergé en même temps que cette technologie.

Depuis 2019, l'Organisation mondiale de la propriété intellectuelle[7] (*OMPI / WIPO* en anglais) organise des sessions de dialogue sur la propriété intellectuelle et l'intelligence artificielle. De nombreux intervenants à travers le monde viennent enrichir les débats et permettent de parcourir divers sujets reliés à la propriété intellectuelle tels que :

- Les brevets et inventions : comment gérer les inventions impliquant de l'IA dans les processus d'invention ?
- Les droits d'auteur : à qui est-ce que cela appartient ?
- Protection des données et vie privée : questionnement sur la protection des données et de la vie privée.
- Normes et interopérabilité : Mise en place de normes pour garantir l'interopérabilité et la compatibilité entre différents systèmes d'IA.
- Accès et Innovation : Questionnement sur la manière d'équilibrer la protection des droits de propriété intellectuelle avec la promotion de l'innovation et l'accès équitable aux technologies d'IA.

Tous ces sujets sont fréquemment rediscutés en raison de l'évolution constante des technologies.

[7] Dialogue de l'OMPI sur la propriété intellectuelle et l'intelligence artificielle

Risques et bénéfice

Selon *Google Cloud*[8], l'IA conversationnelle met à disposition un certain nombre de services bénéfiques pour les entreprises. On parle entre autres d'une augmentation de la productivité à moindre coût (économique). Ces technologies permettent l'automatisation de certaines tâches (on remplace les tâches pouvant être effectuées par des êtres humains), ainsi cela permet d'éviter les erreurs humaines et de réduire les coûts. Niveau disponibilité, les IA sont imbattables (24h/24, 7j/7) par rapport à un homologue humain.

Comme mentionné plus haut, la notion de propriété intellectuelle soulève de nombreuses questions. De nombreux faits-divers, ont remonté certains problèmes (écologiques / économiques / psychologiques) que pouvaient poser les IA conversationnelles. Nous allons donc parler du parasite que peuvent constituer les *LLM*[9] envers les écosystèmes open-source. Dans *Le Monde informatique*[10], Dries Buytaertles, fondateur de *Drupal*, dit : "nous sommes tous plus « récupérateurs » que « créateurs »." Cette pratique généralisée dans la communauté open-source n'est pas une exception pour les *LLM*. Les modèles de langage sont ainsi principalement "parasitaires", se basant sur les dépôts de code (*GitHub* / *GitLab*), forums (*StackOverflow*) et la littérature. Cette technologie entraîne une baisse d'audience pour les sites qui compilent ces informations. Ainsi, les plateformes comme *StackOverflow* et *Reddit* demandent aux créateurs de *LLM* de payer pour le droit d'utiliser leurs données.

Les IA génératives et la loi

Ce rapport tombe à pic, le 9 décembre 2023, le *Conseil et le Parlement européen*[11], s'est récemment mis d'accord sur une législation à propos de l'IA. Cette première mondiale "harmonise les règles relatives aux systèmes d'IA en veillant à ce qu'ils soient sûrs et respectent les droits fondamentaux et les valeurs de l'UE".

En d'autres termes, la législation parcourt les sujets comme :

- Les définitions et champs d'application.
- La classification de systèmes d'IA (pratiques à haut risque / interdites).
- Les exceptions en matière d'application de la loi (utilisation d'IA à haut risque même sans succès de la procédure d'évaluation de la conformité en cas d'urgence).
- Systèmes d'IA à usage général et modèles de référence (obligations de transparence spécifiques des modèles d'IA génératives avant leur mise sur le marché).
- Une nouvelle architecture de gouvernance (création d'un bureau de l'IA pour superviser / appliquer la législation)

[8] IA conversationnelle pour des expériences plus riches et plus intuitives
[9] En français : "Grand modèle de langage"
[10] ChatGPT, un potentiel parasite pour l'écosystème open source
[11] Conseil européen, Législation sur l'intelligence artificielle

- Sanctions sous forme d'un pourcentage sur le chiffre d'affaires annuel, soit 7% (ou à défaut 35 millions d'euros) pour les violations des applications d'IA interdites, 5 millions d'euros (ou 3 %) pour non-respect de la législation, et enfin, 7,5 millions d'euros (ou 1,5 %) pour la communication d'informations inexactes.
- Transparence et protection des droits fondamentaux (obligation d'utiliser des bases de données de l'UE pour les systèmes d'IA à haut risque)
- Mesures de soutien à l'innovation (promouvoir un apprentissage réglementaire fondé sur des données probantes)

Notre utilisation dans ce projet

Le rapport a été entièrement généré par une IA conversationnelle à l'aide d'un prompt extrêmement bien détaillé de plus de 6 pages. Bien que cette dernière phrase soit fausse, nous avons tout de même utilisé à quelques reprises les IA génératives comme "*ChatGPT*". Ces utilisations nous ont notamment permit d'initier les squelettes de test de certaines classes. On lui donnait alors le code de la classe à tester ainsi que de suivre la spécification *Given / When / Then*. Malgré une base tout à fait correcte, il fallait ajouter une certaine logique d'initialisation et de déroulement des tests. La génération de code nous a aussi permis de nous "mâcher" le travail pour des fonctionnalités comme le *CSS / XML loader / wrapper*. Cependant, nous sommes rapidement arrivés aux limites de ces outils. En effet, en termes de réflexion de conception, il faut avoir une connaissance globale de l'écosystème applicatif (comment nos classes interagissent entre elles).

Au moment où l'on rédige ce rapport, *IntelliJ* vient de sortir son assistant basé sur *GPT-4*. *AI Assistant*[12] vient remédier à la limite exposée ci-dessus. En effet, cette assistance a connaissance du contexte de *l'IDE*. Nous n'avons pu l'expérimenter qu'une journée avant qu'elle ne devienne payante. Cette dernière est particulièrement pratique pour certaines facettes du "refactoring" comme la proposition de nom de variable / constante / fonction plus explicite ou encore la proposition d'optimisation de certaines fonctions. Les points que nous avons trouvés intéressants avec ce type d'assistance ne sont pas tant les fonctionnalités techniques apportées, mais plutôt l'implémentation ergonomique au sein de *l'IDE*.

Ouverture

Après avoir développé les risques et bénéfices "générique", soulevé la problématique de propriété intellectuelle et investigué sur la dualité entre IA et loi, nous allons, pour finir cette partie, nous recentrer sur nous en tant qu'étudiants. Il s'agira alors de formuler deux ouvertures basées sur notre intuition initiale et appuyées avec diverses sources (articles / livres).

Le sentiment de la tâche accomplie pour tout travail, quel qui soit, est gratifiant. Combien d'entre nous (étudiant.es en informatique) avons été fiers de venir à bout d'un bug dans notre code. La

[12] AI Assistant dans les IDE JetBrains

philosophe Nathalie Sarthou-Lajus développe alors dans l'un de ses ouvrages[13] : "la fierté qu'un travail bien fait peut procurer, satisfaction plus proche de l'artisanat que d'une logique de production exclusivement centrée sur le rendement.". Mais alors, avec l'apparition de ces nouveaux outils de développement, sommes-nous encore confrontés à la même intensité de réflexion face à un problème à résoudre. La première ouverture est donc : comment les solutions apportées par les IA génératives impactent-elles le sentiment de satisfaction du travail bien fait ?

Notre deuxième ouverture découle de la première. Il semblerait que le système de récompense[14] soit atteint, car stimulé plus régulièrement et cela pour des tâches qui demandent peu d'implication. Nous pensons alors que cela peut impacter le système d'apprentissage des étudiants qui pourraient rencontrer des difficultés à surmonter les échecs nécessairement présents lorsque l'on cherche une solution à un problème. C'est sur ces réflexions que nous formulons notre deuxième ouverture : les IA génératives, impactent-elles la capacité d'apprentissage ?

Conclusion

Dans sa globalité, l'UE MIF01 : Gestion de Projet et Génie Logiciel, nous a apportée, à travers les différents sujets vu en cours, une nouvelle vision de la conception de projet. Notamment l'importance de travailler proprement tout au long du projet (checkstyle / test), la nécessiter de réflexion autour du squelette de l'application (design pattern / modularité du code / refactoring) et la prise de recul sur les technologies utilisées / mis en place avec un regard éthique.

[13] Un travail, c'est aussi une fierté
[14] Une juste récompense assure une bonne mémoire