UMONS
Université de Mons

Faculté des Sciences

Projet de premier master en sciences informatiques

Rapport

Prototype de chatbot assistant l'écriture de code propre et exploitant les grands modèles de langage

Élève :
Nicolas Gatta

Directeur de projet :
Stéphane Dupont

Année académique 2023 - 2024

Table des matières

1 Introduction

Ce projet en sciences informatiques se déroule dans le cadre de la première année du diplôme de master en sciences informatiques à finalité spécialisée en Artificial Intelligence and Data Analytics. Les objectifs principaux de ce projet sont de permettre d'apprendre à se documenter, de s'informer, d'apprendre de nouveaux concepts et outils ainsi que de faire preuve d'indépendance et de savoir gérer son temps.

Le thème choisi pour ce projet est intitulé "Prototype de chatbot assistant l'écriture de code propre et exploitant les grands modèles de langage". Ce thème a pour but de se concentrer sur le développement d'un logiciel permettant à l'utilisateur de parler à l'aide d'une interface de saisie à l'ordinateur.

Ce thème offrant un éventail vaste de possibilités, il a été décidé que ce projet allait se concentrer sur le développement d'un chatbot complété d'une interface, qui sera dans la capacité d'aider l'utilisateur à corriger son code au niveau de la syntaxe ainsi que de l'aider à appliquer les principes du Clean Code. En plus de cela, il inclura des fonctionnalités standard de chatbot tel que la tenue de conversations.

Les objectifs de ce travail vont être d'une part, d'approfondir les connaissances liées aux concepts d'intelligence artificielle et de chatbot, et d'autres par, d'améliorer les compétences liées à la conception et l'implémentation de code tout en respectant les principes de Clean Code. Ce rapport a pour objectif de fournir un aperçu du projet en permettant au lecteur de découvrir les tenants et aboutissant grâce à la division du rapport en plusieurs sections distinctes.

La section 2 a pour but d'aider le lecteur qui n'a pas les connaissances sur les concepts centraux de ce projet, de les présenter de manière détaillée et simple. La section 3 liste les technologies qui ont été nécessaires pour la réalisation de ce projet tout en y incluant des explications sur leur fonctionnement. La section 4 donne de manière graphique tout le processus de fonctionnement du logiciel ainsi que la structure de la base de données. La section 5.2 aborde la description complète du logiciel en passant par les détails techniques de l'implémentation des différentes technologies jusqu'au manuel de l'utilisateur. La section 6 fournit une description détaillée de la performance du logiciel en utilisant diverses métriques telles que la rapidité de réponse. Enfin, la section 7 conclut le travail réalisé en récapitulant de manière concise les différentes observations et en offrant des pistes permettant une amélioration possible du logiciel.

2 Préambule

L'objectif de cette section est de permettre aux lecteurs de se familiariser avec les concepts qui composent la pierre triangulaire de ce projet ainsi que de servir d'introduction à des éléments souvent négligés qui jouent pourtant un rôle crucial dans le monde du développement informatique. Les trois concepts qui composent la pierre triangulaire sont le Clean Code, la rigueur syntaxique dans le développement de logiciels et les chatbots.

2.1 Le Clean Code : L'art de la programmation propre

2.1.1 Introduction au Clean Code

Le Clean Code est un concept popularisé par l'ingénieur logiciel Robert C. Martin grâce à son livre "Clean Code : A Handbook of Agile Software Craftsmanship"[1]. Ce livre représente bien plus que de simples conventions de programmation, il incarne une philosophie que tout développeur logiciel devrait suivre. Il a pour but de placer la lisibilité, la simplicité et la maintenabilité au cœur de la conception des programmes. A la base de cette approche de la part de Robet C. Martin se trouve la conviction que le code source devrait non seulement permettre à un programme de fonctionner, mais également de permettre aux développeurs de facilement faire transiter l'information dans le présent ou pour le futur. De plus, il aurait pour effet d'améliorer la collaboration au sein des équipes, d'accélérer la détection d'erreurs et de faciliter la maintenance du logiciel tout au long de son cycle de vie.

2.1.2 Principes fondamentaux

Les principes du Clean Code orientent la production de code informatique vers une qualité supérieure et une compréhension plus aisée. En suivant les principes ci-dessous, les développeurs créent un code source qui favorise une communication claire et efficace entre les membres de l'équipe :

1. **Lisibilité**
   Le premier principe du Clean Code est la lisibilité. Le code doit être rédigé de manière à ce que sa compréhension soit immédiate. Cela passe par des noms de variables explicites, une indentation cohérente et une organisation logique du code.

2. **Simplicité et Clarté**
   Le deuxième principe du Clean Code est la simplicité et la clarté. Les solutions simples sont préférées aux solutions complexes. Il est entendu par là que tout un chacun doit pouvoir comprendre le code et non pas seulement l'auteur.

3. **Modularité**
   Le troisième principe du Clean Code est la modularité. Ce principe encourage la division de code en modules distincts. Chaque module doit avoir une responsabilité unique et bien définie qui permet ensuite de faciliter la réutilisation du code et permet une maintenance plus aisée.

4. **Éviter la redondance**
   Le quatrième principe du Clean Code est d'éviter la redondance. Ce principe va de pair avec le principe de modularité, il faut éviter à tout prix des morceaux de codes répétitifs et si le cas se présentait les encapsuler dans des fonctions ou des classes.

5. **Maintenabilité**
   Le cinquième principe du Clean Code est la maintenabilité. Les développeurs doivent pouvoir comprendre rapidement le fonctionnement du code existant et y apporter si possible des modifications sans introduire de bugs inattendus.

6. **Tests unitaires**
   Le sixième principe du Clean Code est les tests unitaires. Des tests bien écrits garantissent que le code fonctionne comme prévu et permettent de détecter rapidement un problème après une modification.

7. **Élégance**
   Le septième principe du Clean Code est l'élégance. Un code élégant est à la fois fonctionnel et agréable à lire, il montre l'habilité du développeur dans son art.

2.1.3 Exemple : Impacts du Clean Code sur la compréhension du code

Dans l'exemple ci-dessous, on peut constater que les noms des variables **x**, **y**, **result** ne reflètent pas de manière optimale la nature de ces variables dans la partie Listing 1 – Sans Clean Code. De plus, on peut aussi remarquer que le nom de la fonction **fn** n'est pas représentatif de ce que celle-ci effectue.

Pour corriger ce problème, il faut effectuer des changements sur les éléments cités ci-dessus pour respecter les principes du Clean Code et obtenir le résultat qui se trouve dans la partie Listing 2 – Avec Clean Code :

1. **x** → **nombre1**
2. **y** → **nombre2**
3. **result** → **somme**
4. **fn** → **additionner**

```python
def fn(x, y):
result = x + y
return result
```

Listing 1 – Sans Clean Code

```python
def additionner(nombre1, nombre2):
    somme = nombre1 + nombre2
    return somme
```

Listing 2 – Avec Clean Code

2.2 Rigueur syntaxique dans le développement de logiciels

2.2.1 Définition de la syntaxe

Avant de définir ce qu'est la rigueur syntaxique, il est important de définir ce qu'est la syntaxe. La syntaxe est définie comme des séquences et combinaisons de caractères nécessaires pour créer du code correctement structuré. En effet, la syntaxe ne fait que donner une structure et de l'ordre dans les instructions d'un code, mais celui-ci dépend énormément de la sémantique pour donner une signification à ces instructions. Cependant, la syntaxe est importante lors de la création de code et ne pas la respecter conduit à s'exposer à des erreurs de syntaxe qui vont bloquer le programme lors de son exécution.[2]

2.2.2 Définition de la rigueur syntaxique

De par la définition dans la section 2.2.1, la définition de la rigueur syntaxique découle tout naturellement. La rigueur syntaxique dans le contexte du développement logiciel fait référence à la conformité stricte du code à une syntaxe spécifique du langage de programmation actuellement utilisé. Il est question de respecter les règles et conventions établies pour s'assurer de la lisibilité, la maintenabilité et la compréhension du code.

2.2.3 Importance de la rigueur syntaxique

L'importance de la rigueur syntaxique se retrouve très fréquemment associée au Clean Code et pour cause, ils partagent tous deux énormément de points. En conséquence, il est difficile de les dissocier. La liste des éléments composant la rigueur syntaxique est ainsi très similaire à celle que l'on peut retrouver pour le Clean Code :

1. **Lisibilité**
   Une syntaxe claire et cohérente permet une plus grande aisance lors de la lecture du code et donc facilite la collaboration au sein de l'équipe de développement.

2. **Prévention des erreurs**
   La rigueur syntaxique contribue à prévenir de potentiels bugs liés à une mauvaise structure du code ou du moins permet de réduire les risques que cela se produise.

3. **Conformité aux standards**
   La rigueur syntaxique pousse les développeurs à respecter les conventions de codage recommandées par la communauté du langage en question.

4. **Facilitation des modifications**
   La rigueur syntaxique permet d'avoir un code bien formaté, ce qui permet aux développeurs de se repérer plus facilement dans le code pour opérer des modifications.

5. **Maintenabilité**
   La rigueur syntaxique permet une maintenabilité du code plus aisée et donc une réduction des coûts associés lors de l'évolution du programme.

2.2.4 Exemple : Impacts de la rigueur syntaxique sur la lisibilité du code

Lorsque la rigueur syntaxique est respectée, le code devient plus lisible et compréhensible pour les développeurs. Grâce aux exemples ci-dessous, la raison pour laquelle la rigueur syntaxique est si importante devient évidente. On peut voir les effets du non-respect de la syntaxe dans Listing 3 – Sans rigueur syntaxique. Il est plus compliqué de lire le code à cause d'une syntaxe un peu hasardeuse alors que dans Listing 4 – Avec rigueur syntaxique, le contenu est plus agréable visuellement.

```
def calculer_resultat(x, y,
    ↪ operation){
    resultat = 0
    if operation == 'add': resultat
        ↪ = x + y
    else:
        if operation == 'sous':
            ↪ resultat = x - y
        else:
            if operation == 'multi':
                ↪ resultat = x * y
            else: resultat = x / y
    return resultat
```

Listing 3 – Sans rigueur syntaxique

```
def calculer_resultat(x, y,
    ↪ operation):

    if operation == 'add':
        return x + y
    if operation == 'sous':
        return x - y
    if operation == 'multi':
        return x * y
    if operation == 'div':
        return x / y

    return None
```

Listing 4 – Avec rigueur syntaxique

2.3 Les chatbots : La révolution des interactions Homme-Machine

2.3.1 Introduction aux chatbots

Un chatbot est un type d'intelligence artificielle qui incarne l'interaction entre l'homme et la machine. Fonctionnant sur le même principe qu'un programme informatique, un chatbot a pour but principal de créer et maintenir des conversations avec son utilisateur par le biais de texte ou encore de la voix. En plus de cela, il fait preuve de la capacité de comprendre et de répondre dans une ou plusieurs langues à l'aide d'algorithme de traitement du langage naturel qui sera abordée plus en détail dans la section 3.1.[3]

2.3.2 La Révolution des interfaces Homme-Machine

L'intégration de l'intelligence artificielle dans notre vie quotidienne a doucement amené à une nouvelle révolution aussi appelée la quatrième révolution industrielle. L'un des points clés de l'entrée dans cette nouvelle révolution est l'apparition des chatbots qui permettent une interaction plus intuitive et facile entre l'utilisateur et la machine en éliminant les barrières techniques qui pouvaient exister. Maintenant, tout un chacun peut se permettre d'utiliser des chatbots sans avoir de compétences techniques dans le domaine. Cela a été rendu possible grâce à deux éléments clés. Le premier élément est l'utilisation de technologie permettant à des programmes informatiques de mieux comprendre et donc répondre aux requêtes provenant du langage humain. Le deuxième élément est des interfaces simples pour permettre à monsieur et madame Tout-le-Monde d'utiliser ces nouvelles technologies.

2.3.3 Exemple : ChatGPT

Un exemple concret illustrant bien l'avènement des chatbots est la création et la mise à disposition de chatGPT. En effet, même si chatGPT est une nouvelle technologie, il reste très simple d'utilisation grâce à son interface plus que basique mais remplissant sa fonction à merveille comme montrer dans la Figure 1 – Interface de chatGPT.

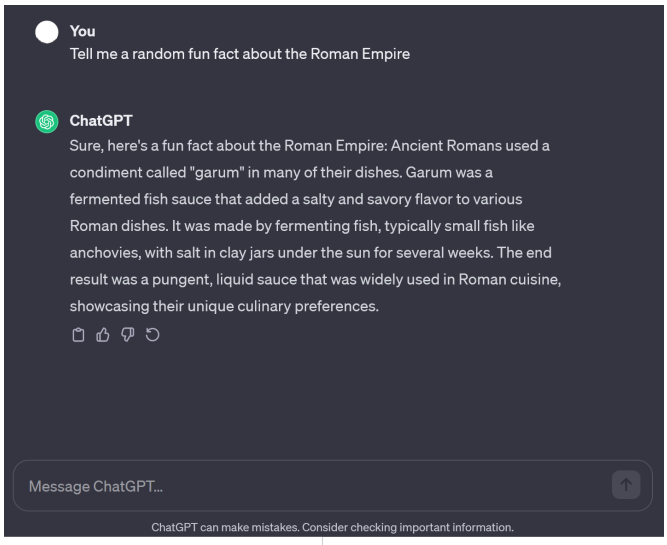

FIGURE 1 – Interface de chatGPT

3 Technologies utilisées

Cette section vise à expliquer les principes de fonctionnements des différents composants exploités tout au long de ce projet. C'est une partie cruciale du document qui permet d'explorer en détails les différentes architectures, technologies et frameworks intégrés dans le projet.

3.1 Natural Language Processing

3.1.1 Introduction

Le Natural Language Processing (NLP) ou Traitement Automatique du Langage Naturel (TALN) en français est une sous-discipline du domaine mêlant intelligence artificielle et linguistique. Cette technologie est dédiée à la compréhension des mots, phrases ou expressions dans les langues humaines par un ordinateur.

Un langage naturel est un langage qui est parlé ou écrit par des êtres humains dans un but de faire transiter des informations entre eux. Ce terme est apparu grâce à l'apparition d'utilisateurs voulant communiquer avec un ordinateur sans pour autant apprendre le langage spécifique de la machine.

De par cela, on peut conclure que la technologie du NPL est apparue pour aider les personnes qui n'ont pas les connaissances qui permettent l'interaction avec un ordinateur en utilisant le langage machine, mais qui possèdent des connaissances assez poussées dans un langage humain que pour communiquer avec une autre personne, qui est dans ce cas remplacée par un ordinateur. [4]

3.1.2 Les niveaux de langage dans le Natural Language Processing

La compréhension Natural Language Processing peut être ardue de par la complexité de ce système, il est donc très commun d'utiliser l'approche par niveaux de langage pour l'expliquer. Cette méthode de décomposition par niveaux comparer à la méthode séquentielle permet un traitement plus dynamique du langage et une interaction plus facile entre les différents niveaux.

Cette manière de procéder provient des recherches réalisées dans le domaine de la psycholinguistique[1] qui suggèrent que le traitement du langage se fait de manière dynamique. L'un des exemples les plus parlants pour illustrer le dynamisme d'une langue réside dans la présence de mots aux sens multiples. Grâce à l'utilisation des différents niveaux de langage, un sens peut être privilégié par rapport aux autres. Passons maintenant à l'explication des différents niveaux de langages utilisés par la technologie NLP. Il est important de noter que le premier niveau, la phonologie, n'est utilisée que dans le cas d'une utilisation vocale du NLP. [5]

1. Psycholinguistique : la psycholinguistique est un domaine de recherche relativement récent qui se donne pour objectif de mettre au jour les mécanismes impliqués dans l'utilisation du langage, plus spécifiquement dans la production, la compréhension et l'acquisition du langage. [6]

1. **Phonologie**
   Ce premier niveau concerne l'interprétation des sons provenant de la parole telle que les mots. Il utilise trois règles qui sont respectivement la phonétique[1], le phonème[2] et la prosodie[3].

2. **Morphologie**
   Ce deuxième niveau traite la nature des mots composés. Par exemple, le mot pré-enregistrement qui peut être analysé en deux parties distinctes : la première qui est "pré" qui signifie "avant" et la deuxième qui est "enregistrement" qui est "l'action d'enregistrer"

3. **Lexique**
   Ce troisième niveau se base sur l'analyse des mots de manières individuels ce qui aide à attribuer un seul sens à un mot et ainsi permettre de remplacer ce mot par une représentation.

4. **Syntaxique**
   Ce quatrième niveau se concentre sur l'analyse des mots qui composent une phrase pour en comprendre la structure grammaticale. Cela résulte en une représentation de la phrase qui met en évidence les relations entre les mots.

5. **Sémantique**
   Ce cinquième niveau permet de déterminer les significations possibles d'une phrase en se concentrant sur les interactions entre les significations des mots dans la phrase. Cela permet de désambiguïser[4] le sens de certains mots grâce au contexte de la phrase et permettre de fixer un sens au mot qui semble logique pour la représentation de la phrase.

6. **Discours**
   Ce sixième niveau comparé au niveau syntaxique et sémantique œuvre sur des textes plus longs qu'une phrase. Il se concentre sur les propriétés du texte dans son ensemble et le sens que celui-ci essaie de transmettre.

7. **Pragmatique**
   Ce septième niveau se concentre sur l'utilisation intentionnelle de certains types de langage dans des situations bien précises et utilise le contexte global du contenu pour établir une meilleure compréhension du texte.

1. Phonétique : partie de la linguistique qui étudie les sons de la parole.
2. Phonème : unité distinctive de prononciation dans une langue.
3. Prosodie : ensemble des traits oraux d'une expression verbale d'un locuteur.
4. Désambiguïser : faire disparaître l'ambiguïté.

3.1.3 Comment fonctionne le Natural Language Processing

La technologie du Natural language Processing utilise trois grands processus qui sont respectivement le prétraitement des données, l'extraction de caractéristiques et la modélisation.

La première étape est le prétraitement des données. Avant qu'un modèle ne traite le texte pour une tâche spécifique, il est nécessaire de prétraiter les données pour améliorer les performances du modèle. On retrouve une variété de techniques diverses qui peuvent être utilisées lors de ce prétraitement tel que l'analyse syntaxique, la racinisation, la sémantique, la suppression des mots vides et la tokenisation.

1. **Analyse syntaxique**
   L'analyse syntaxique est un processus qui a pour but de déterminer le début et la fin de chaque phrase dans le document.

2. **Racinisation**
   La racinisation est un processus qui permet de convertir des mots en leurs formes de bases. On peut par exemple ramener les mots "chanteur", "chanteuses" et "chantent" à la base "chant".

3. **Sémantique**
   La sémantique est un processus qui permet de comprendre le rôle de chaque mot dans une phrase. Ce processus va marquer chaque mot avec une étiquette telle que nom, adverbe, etc.

4. **Suppression des mots vides**
   La suppression des mots vides est un processus qui permet d'enlever les mots qui n'apportent que peu d'informations sur le texte telles que "un", "le", "une", etc.

5. **Tokenisation**
   La tokenisation est un processus qui consiste à diviser le texte en mots individuels puis à attribuer à chaque mot un numéro spécial comme une sorte de code d'identification pour ensuite reconstruire la phrase grâce au code d'identification des mots comme dans la Figure 2 – Fonctionnement de la tokenisation.

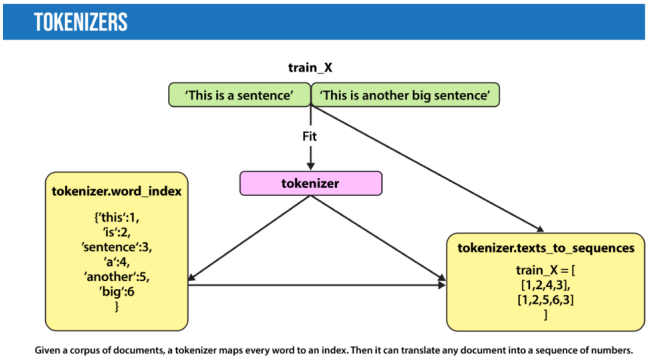

FIGURE 2 – Fonctionnement de la tokenisation [7]

La deuxième étape est l'extraction de caractéristiques. Elle s'effectue à la suite de la tokenisation du texte pour obtenir les séries de chiffres qui décrivent le texte. Il est ensuite possible de trouver des caractéristiques en utilisant différentes méthodes telles que le sac de mots ou le TF-IDF.

1. **Sac de mots**
   La méthode du sac de mots consiste à compter le nombre d'occurrences d'un mot ou d'un groupe de mots dans un document. Pour cela, la tokenisation doit être réalisée au préalable, comme cela devrait déjà être effectué dans la partie prétraitement. Ensuite, on procède au décompte du nombre d'apparitions de chaque mot dans la phrase comme illustré dans la Figure 3 – Fonctionnement du sac de mots.

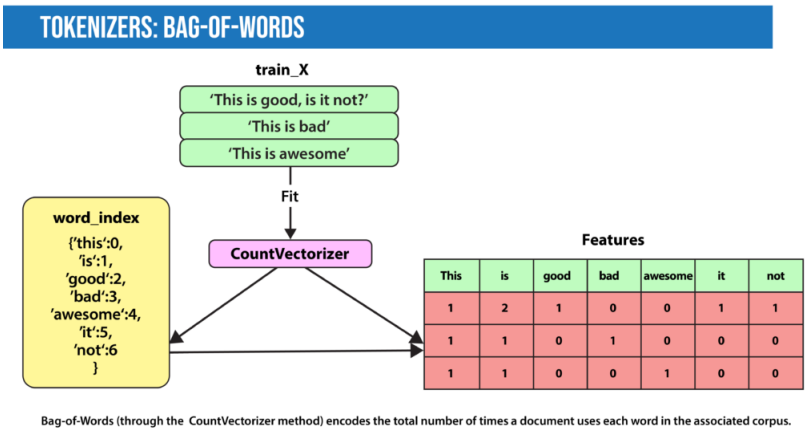

FIGURE 3 – Fonctionnement du sac de mots [7]

2. **TF-IDF**
   La méthode du TF-IDF fonctionne de la même manière que le sac de mots à la différence qu'elle ajoute une pondération à chaque mot en fonction du nombre de fois qu'il apparaît par rapport au nombre total de mots comme illustré dans la Figure 4 – Fonctionnement du TF-IDF.

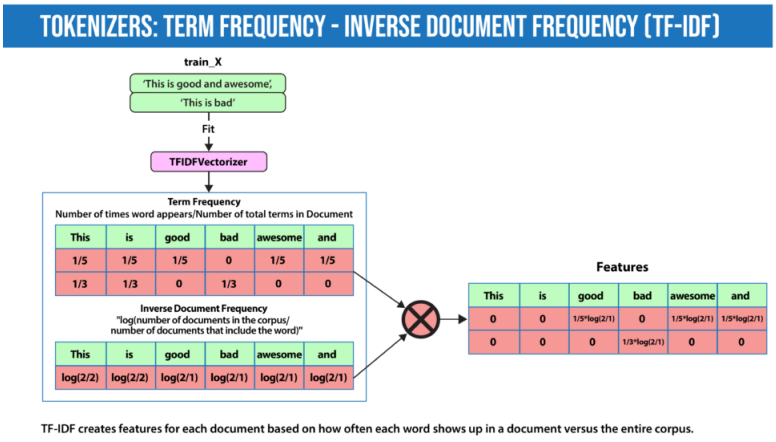

FIGURE 4 – Fonctionnement du TF-IDF [7]

La troisième et dernière étape concerne l'application de modèles d'apprentissages tels que les réseaux neuronaux, les SVM[1] ou des méthodes utilisant des statistiques pour permettre la classification, la traduction automatique ou encore la génération de texte. [7] [8]

1. **Réseau de neurones artificiels**
   Le réseau de neurones artificiel est une structure adaptative inspirée de la nature, plus précisément du fonctionnement du cerveau humain. Composé d'éléments appelés neurones[2], il utilise la plasticité[3] de manière similaire à sa contrepartie humaine ce qui permet la modification de l'agencement des connexions pour se spécialiser dans une tâche précise. Le réseau de neurones fonctionne en traitant des entrées qui passent successivement à travers plusieurs couches de neurones avant de produire une sortie, comme illustré dans la Figure 5 – Fonctionnement du réseau de neurones. [9]

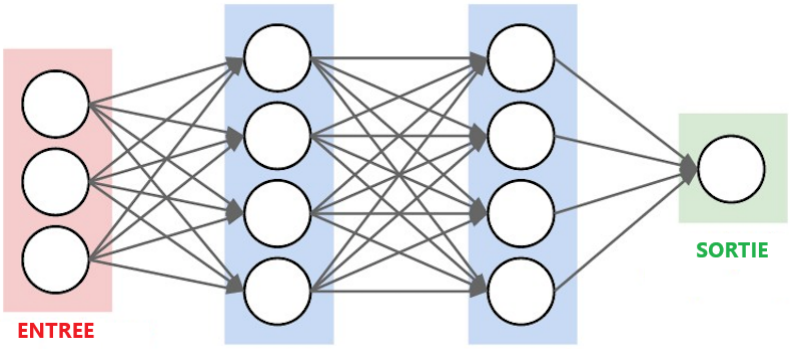

FIGURE 5 – Fonctionnement du réseau de neurones [10]

2. **SVM**
   Les SVM sont des algorithmes d'apprentissage qui sont utilisés à des fins de classifications. Ils fonctionnement grâce à l'utilisation d'un hyperplan[4] qui est utilisé pour séparer les différentes classes d'objets de manière optimale, comme illustré dans la Figure 6 – Fonctionnement de l'hyperplan. [11]

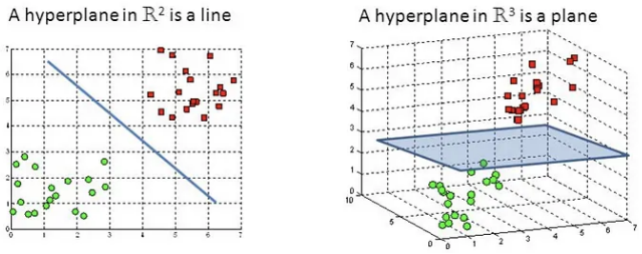

FIGURE 6 – Fonctionnement de l'hyperplan [11]

1. SVM ou Support Vector Machine : Algorithme de classification utilisant les vecteurs.
2. Neurone : Élément spécialisé dans le traitement, la réception et la transmission d'informations.
3. Plasticité : Capacité du cerveau à changer, se remodeler, se réorganiser, dans le but de s'adapter à de nouvelles situations. [12]
4. Hyperplan : Objet qui permet de séparer un espace en deux parties distinctes.

3.2 Analyseur syntaxique

3.2.1 Introduction

Dans le domaine de l'informatique, le parseur également connu sous le nom d'analyseur syntaxique est un composant essentiel présent dans la plupart des compilateurs.[1] Sa fonction principale est d'analyser les données en entrée qui sont sous la forme d'instructions séquentielles provenant du code source puis, de repérer les erreurs potentielles à l'aide de divers mécanismes.[13]

3.2.2 Comment fonctionne un analyseur syntaxique

L'analyseur syntaxique se compose de trois éléments, chacun gérant une étape différente du processus d'analyse. Ces trois étapes sont les suivantes :

1. **L'analyse lexicale**
   L'analyse lexicale est un composant essentiel dans le processus de compilation d'un langage de programmation. Il a pour but de segmenter le code source en petits morceaux appelés jetons ou tokens et en plus de permettre une classification de chaque élément. Chaque jeton est une brique élémentaire de la grammaire du langage comme les mots clés, les identifiants, les opérateurs et les littéraux. Ceux-ci sont essentiel pour que le compilateur comprenne et traite le code de manière appropriée comme illustré dans la Figure 7 – Classification des tokens. En plus de cela, l'analyse lexicale va supprimer tous les éléments non essentiels, les commentaires et les caractères d'espacement.

FIGURE 7 – Classification des tokens [13]

1. Compilateur : Un programme qui traite les instructions écrites dans un langage de programmation donné pour les traduire en langage machine

2. **L'analyse syntaxique**
   L'analyse syntaxique à pour but de validé si la séquence de jetons respect les règles syntaxiques du langage. Pour cela, l'utilisation d'un arbre syntaxique est essentiel et permet la représentation de la structure du code comme présenter dans la Figure 8 – Arbre syntaxique. Cet arbre permet une visualisation des différentes relations entre les jetons tout en respectant les règles grammaticales du langage. Si le code source viole ces règles, des erreurs de syntaxe sont signalées pour indiquer les parties du code qui doivent être corrigées.

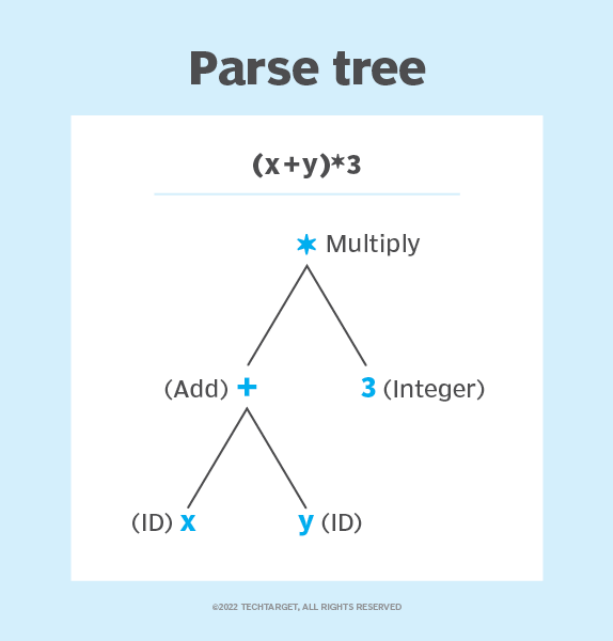

FIGURE 8 – Arbre syntaxique [13]

3. **L'analyse sémantique**
   L'analyse sémantique vérifie la cohérence de l'arbre de dérivation avec la sémantique du langage en effectuant des tâches telles que la vérification des types, la résolution des variables et la détection d'erreurs sémantiques. Elle garantit que les variables sont déclarées avant utilisation et que les fonctions sont appelées correctement. Au final, Son objectif est de détecter les erreurs non capturées par l'analyse syntaxique et qui empêcherait le programme de s'exécuter correctement.

3.2.3 Quels sont les différents types d'analyseur syntaxique ?

Pour accomplir les différentes tâches de l'analyseur syntaxique, il existe différentes méthodes qui vérifies les étapes dans différents ordres. Il existe deux principaux types d'analyseur syntaxique :

1. **L'analyseur descendant (top-down) :**
   Les analyseurs descendant commencent le processus en partant de la règle la plus générale dans la grammaire du langage de programmation et descendent jusqu'aux symboles terminaux.
2. **L'analyseur ascendant (bottom-up) :**
   Les analyseurs ascendant commencent le processus en partant des symboles terminaux dans la grammaire du langage de programmation et remontent jusqu'à la règle la plus générale.

En plus de cela, il existe deux types de variation possible pour les analyseurs syntaxiques :

1. **L'analyseur LL (Left-to-Right, Leftmost Derivation) :**
   L'analyseur LL parcourt l'entrée de gauche à droite en utilisant une dérivation la plus à gauche. Cela signifie qu'à chaque étape de l'analyse, l'analyseur choisit la règle la plus à gauche pour étendre l'arbre de dérivation.
2. **L'analyseur LR (Left-to-Right, Rightmost Derivation) :**
   L'analyseur LR analyse également l'entrée de gauche à droite, mais en utilisant une dérivation la plus à droite. Cela signifie qu'il choisit toujours la règle la plus à droite lorsqu'il étend l'arbre de dérivation.

4 Modélisation conceptuelle

La section sur la modélisation conceptuelle est une étape cruciale dans le développement de logiciel informatique. Elle vise à représenter de manière compréhensible les différentes étapes de conceptions d'un logiciel qui seront ensuite présentées à d'autres personnes qui ne sont pas nécessairement sensibles à la technologie.

4.1 BPMN (Le Business Process Model and Notation)

4.1.1 Qu'est-ce qu'un Business Process Model and Notation ?

Le BPMN est une norme de modélisation des processus métier[1] qui fournit une base graphique commune à toute une équipe. Cela permet aux différentes personnes composant l'équipe d'avoir une base commune pour comprendre, documenter et optimiser le processus métier. [14]

4.1.2 Représentation du projet en schéma BPMN

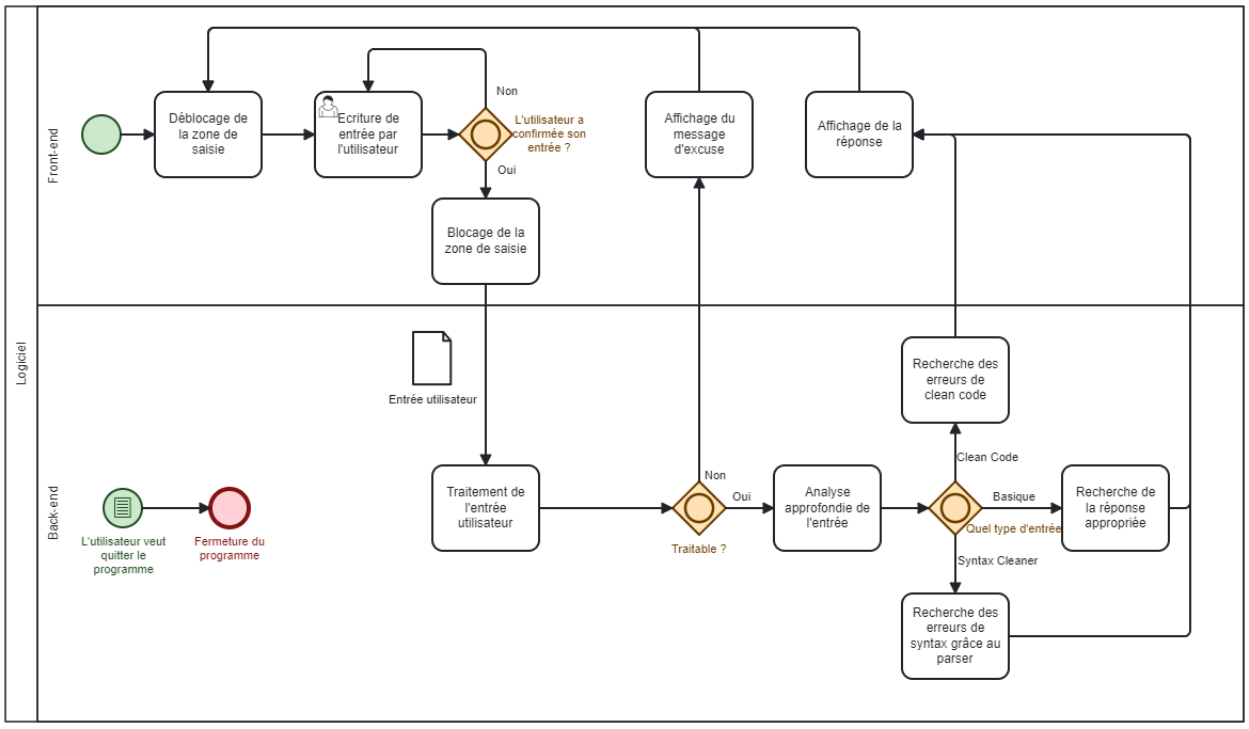

FIGURE 9 – Schéma BPMN du logiciel

1. Processus métier : une activité ou un ensemble d'activités mis en place pour réaliser une tâche, un projet ou atteindre un objectif [15]

4.2 Schéma entité-association

4.2.1 Qu'est-ce qu'un schéma entité-association ?

Le schéma entité-association est une technique de modélisation de données largement utilisée dans le domaine de la conception de bases de données. Il offre une représentation visuelle des entités, des relations et des attributs qui composent un système d'information de manière simple et claire.

4.2.2 Représentation de la base de données en schéma entité-association

5 Structure du logiciel

5.1 Fonctionnalités

5.2 Implémentations

6 Évaluation

7 Conclusion

8 Références

[1] Robert C Martin. *Clean code : a handbook of agile software craftsmanship*. Pearson Education, 2009.

[2] MozDevNet. Syntaxe - glossaire mdn : Définitions des termes du web : Mdn. `https://developer.mozilla.org/fr/docs/Glossary/Syntax`.

[3] Eleni Adamopoulou and Lefteris Moussiades. An overview of chatbot technology. In *IFIP international conference on artificial intelligence applications and innovations*, pages 373–383. Springer, 2020.

[4] Abhimanyu Chopra, Abhinav Prashar, and Chandresh Sain. Natural language processing. *International journal of technology enhancements and emerging engineering research*, 1(4) :131–134, 2013.

[5] Elizabeth D Liddy. Natural language processing. 2001.

[6] Marie Labelle. Trente ans de psycholinguistique. *Revue québécoise de linguistique*, 30(1) :155–176, 2001.

[7] deeplearning.ai. Natural language processing (nlp) - a complete guide. `https://www.deeplearning.ai/resources/natural-language-processing/`.

[8] Vineet Raina, Srinath Krishnamurthy, Vineet Raina, and Srinath Krishnamurthy. Natural language processing. *Building an Effective Data Science Practice : A Framework to Bootstrap and Manage a Successful Data Science Practice*, pages 63–73, 2022.

[9] Enzo Grossi and Massimo Buscema. Introduction to artificial neural networks. *European journal of gastroenterology & hepatology*, 19(12) :1046–1054, 2007.

[10] Imagerie vétérinaire – metron software. `https://www.eponamind.com/fr/neural-networks-deep-learning/`.

[11] Rohith Gandhi. Support vector machine - introduction to machine learning algorithms. `https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47`, Jul 2018.

[12] Amélie Dindouve and Yves Vandermeeren. Apprentissage moteur et plasticité cérébrale chez le patient hémiparétique après avc : L'apprentissage bimanuel est-il amélioré par dual-tdcs, comparé à une stimulation placebo ?

[13] Ben Lutkevich. What is a parser ? definition, types and examples, Jul 2022.

[14] Mark von Rosing, Stephen White, Fred Cummins, and Henk de Man. Business process model and notation-bpmn. `http://www.omg.org/news/whitepapers/Business_Process_Model_and_Notation.pdf`, 2015.

[15] NeoLedge. Définition : Processus métier. `https://www.neoledge.com/fr/a-propos/glossaire/definition-processus-metier/`, May 2019.

Table des figures

Liste des tableaux