Rapport Technique – AI Writing Assistant

Le projet a consisté à concevoir une application web intelligente d'assistance à la rédaction, permettant aux utilisateurs de soumettre un texte en vue d'obtenir une correction grammaticale, une correction orthographique ou une complétion automatique. Cette application exploite les capacités d'un grand modèle de langage (LLM) pour fournir des suggestions précises, tout en proposant une interface claire, réactive et ergonomique accessible depuis un navigateur.

L'application repose sur le framework FastAPI combiné au serveur ASGI Uvicorn pour la gestion des requêtes backend. Côté traitement du langage, une architecture est mise en place pour s'appuyer sur le modèle LLaMA 3.2 via la plateforme Ollama, utilisé exclusivement pour la complétion automatique de texte. La correction orthographique est assurée par l'algorithme SymSpell, réputé pour sa rapidité et son efficacité sur de grands corpus, tandis que la correction grammaticale s'appuie sur l'API externe LanguageTool, spécialisée dans la détection et la suggestion d'erreurs linguistiques complexes. Trois types de services sont ainsi proposés à l'utilisateur : correction grammaticale, correction orthographique et complétion automatique, chacun accessible via un bouton dédié dans l'interface.

La structure frontale est centrée autour de la page index.html, qui constitue la racine du projet. Elle intègre un formulaire de saisie, un affichage dynamique des résultats et une barre latérale permettant de visualiser l'historique des requêtes. Cet historique est géré localement via le navigateur à l'aide de l'API localStorage, garantissant la rapidité d'accès et la confidentialité des données utilisateur. L'ensemble des interactions est enrichi par un script JavaScript assurant l'envoi des données au serveur, la gestion du retour JSON, ainsi que la mise à jour de l'interface sans rechargement de page.

Le backend est centralisé dans un seul fichier Python (app.py) qui définit l'ensemble des routes, traite les requêtes POST, génère les réponses en fonction du mode sélectionné, et gère les identifiants utilisateurs anonymes par cookies. L'interface utilisateur est stylisée à l'aide d'une feuille de style CSS commune à toutes les pages, renforçant la cohérence visuelle de l'ensemble. Le style met l'accent sur la lisibilité, la clarté des composants et l'accessibilité sur différents formats d'écran.

En ce qui concerne le déploiement, une approche par conteneurisation est envisagée. L'infrastructure décrite prévoit l'utilisation de Docker avec un Dockerfile configuré pour intégrer les composants nécessaires (FastAPI, Ollama, Solr).

Plusieurs évolutions sont envisagées afin d'enrichir l'expérience utilisateur : ajout d'un module de synthèse et de reconnaissance vocale pour les utilisateurs malvoyants, possibilité de téléverser des fichiers (PDF, images) afin d'enrichir le contexte de traitement, intégration d'une barre de recherche sur la page d'historique, ainsi qu'une personnalisation complète de l'apparence et du style de réponse du chatbot (mode clair/sombre, ton formel ou amical, ajustement de la taille de police).

Difficulté rencontrée : Gestion de l'historique utilisateur avec cookies

L'un des défis que nous avons rencontrés dans ce projet a été la gestion individualisée de l'historique des textes soumis par chaque utilisateur, en l'absence de tout mécanisme de connexion. Il fallait que chaque utilisateur puisse retrouver ses anciennes saisies ainsi que les résultats associés (textes corrigés ou complétés), sans interférence avec les données d'autres utilisateurs.

Le plus complexe était d'assurer que chaque utilisateur soit identifié de manière unique, même sans compte. L'historique de ses soumissions (entrée et sortie) soit enregistré et consultable ultérieurement. Lors d'un clic sur une phrase précédente, la zone de texte soit automatiquement remplie avec l'ancienne entrée et la zone de résultat affiche la version corrigée correspondante.

Pour cela, nous avons combiné cookie (côté client) et localStorage en JavaScript. Lorsque l'utilisateur soumet un texte pour la première fois, un identifiant unique est généré avec crypto.randomUUID() puis stocké dans un cookie via : setCookie('id_user', id_user, 365); Chaque fois qu'un texte est soumis, les données {input, output} sont sauvegardées localement dans un tableau JSON. Cet historique est ensuite affiché dynamiquement dans une barre latérale sous forme de boutons cliquables. La fonction loadHistory() insère dynamiquement les anciennes entrées dans la barre latérale via des éléments HTML .history-item. Le clic sur un élément de l'historique déclenche un événement JavaScript qui

remplit automatiquement le champ de saisie avec l'entrée d'origine et met à jour la zone de résultats avec la correction correspondante.

Cette solution a donc permis de garantir une expérience fluide et personnalisée pour chaque utilisateur, sans recourir à une base de données distante.