TOUATI Damien
CAMARA William

Rapport Projet Java : Xplain

Sommaire

1 - Documentation Utilisateur

introduction à l’interface graphique et au serveur

L’application met en œuvre un serveur web exposant une API REST que les utilisateurs peuvent interroger. Cette API prend en entrée des fichiers class Java et fournit un message de compilation avec des suggestions pour améliorer le code.

Installation du serveur

1 . Téléchargement et installation :

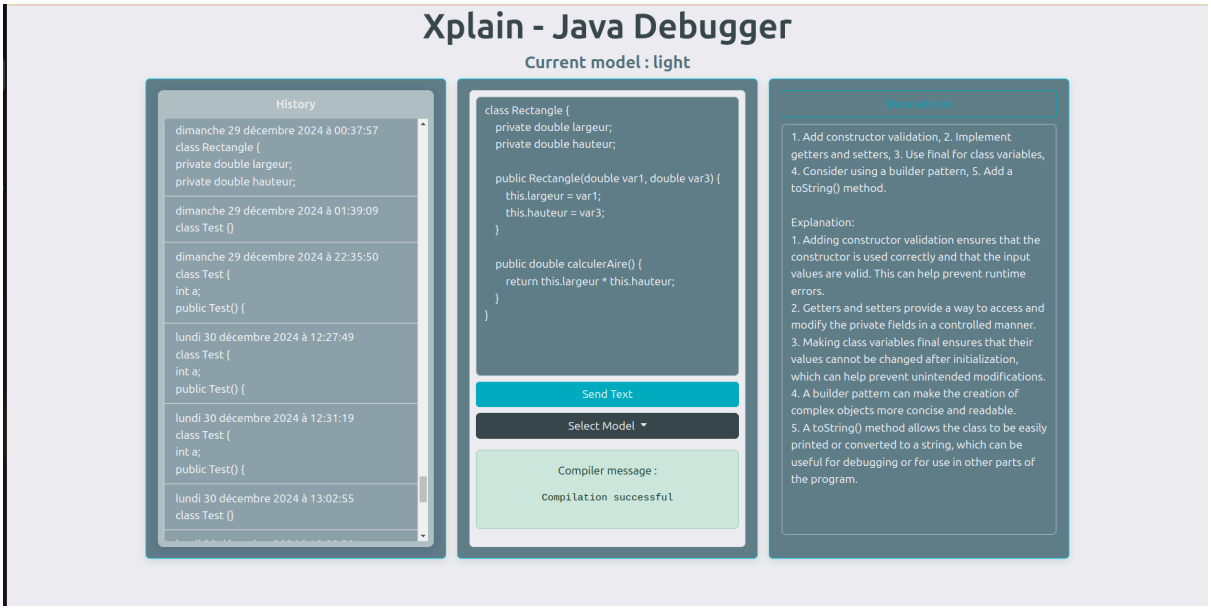
git clone https://gitlab.com/touati_camara/java-project-m1.git

mvn clean

mvn quarkus:dev

Le serveur démarre sur http://localhost:8081/

Utilisation de l’interface



L'interface de l’application est divisé en trois zones :

1. Historique (panneau gauche) :
    - Affiche l'historique chronologique des modifications du code
    - Chaque entrée est daté
    - Permet de revenir à une ancienne version du code
2. Éditeur de Code (panneau central)
    - Zone principale de saisie et d'édition du code Java
    - Inclut un bouton "Send Text" pour compiler le code
    - Le sélecteur de modèle ("Select Model") permet de choisir le modèle light, medium et heavy
    - Affiche les messages du compilateur en bas (ex: "Compilation successful")
3. Suggestions (panneau droit)
    - Fournit des conseils d'amélioration du code
    - Propose des explications détaillées pour chaque suggestion

Utilisation

- Écrivez votre code Java dans l'éditeur central
- Cliquez sur "Send Text" pour compiler
- Consultez les messages de compilation
- Suivez les suggestions d'amélioration à droite après avoir appuyer sur show advices
- Utilisez l'historique à gauche pour revenir aux versions précédentes

Par défaut, le modèle sélectionné est le modèle light.

2 - Documentation Developpeur

Description des packages

Nous avons découpé nos packages pour les regrouper par fonctionnalité et facilité la compréhension du code.

fr.uge.database : contient le code pour gérer la base de données de l’application.

fr.uge.model : contient le code qui permet de charger et d’utiliser les différents model de LLM.

fr.uge.utilities : contient tout le code pour grouper ou emboîter les données utiliser dans les autres package.

fr.uge.xplain : est le package principale du backend de l'application il est le point de communication avec le frontend, c'est le package où les fonctionnalités majeures sont implémentées.

Choix technique

1. Sélection des LLMs :
    - Mistral-7B-Instruct-v0.3.IQ2_XS.gguf
      Car il est léger il fonctionne correctement même avec peu de ressource, il répond vite mais avec moins de précision
    - Mistral-7b-instruct-v0.2.Q4_K_S.gguf
      Bon compris entre le light et le heavy
    - Mistral-7B-Instruct-v0.3.Q6_K.gguf
      il s'agit du même model que le light mais avec meilleur équilibre entre performance et précision

   Lors du choix des LLM nous avons choisi le modéle Mistral-7b car nous étions sur de sa compatibilité avec le binding java de lama.cpp que nous utilisons.

2. Technologie de streaming
   Nous avons utilisé les Server Sent Event pour envoyer les réponses du LLM token par token. Car il était simple à implémenter avec Quarkus.

   Toutefois, nous avons eu du mal à réaliser cette fonctionnalité de l'application car nous avons rencontré divers problèmes.
    - A la réception des Tokens les observateurs ne se déclenchaient pas alors que les données avaient bien été réceptionnées.
      Pour résoudre ce problème nous avons renoncer à utiliser la surcouche UseEventSource de vuejs et nous gérons les SSE avec l'instance EventSource de JavaScript
3. Interface utilisateur

   Nous avons décidé de découper en trois zones l'application afin de mettre à disposition pour l'utilisateur tous les composants de l'application.

   Cependant ce choix nous a contraint à restreindre certaines fonctionnalités lors de la génération d'une réponse, pour éviter des comportement non voulu. En effet, lors de la génération de la réponse il n'est plus possible de soumettre une nouvelle class à l'application, de revenir à une version antérieure ou de changer de modèle.

   Dans l'interface de l'application nous utilisons une animation du dépot :
   https://github.com/greyby/vue-spinner/blob/master/src/BeatLoader.vue#L100
   pour indiquer le début de la génération d'une réponse du LLM.

4. Choix d'implémentation

Nous avons choisi de charger les poids de tous les modèles afin de faciliter le changement de modèle.

Pour la sauvegarde l'historique dans la base de données nous effectuons l'insertion de l'output du compilateur, de la class soumise dans un premier temps puis nous mettrons à jour l'insertion une fois la génération de la réponse complétée dans le but de limiter les accès à la base données et de pas ralentir l'application.

Limite de l'application

Lors de la première utilisation, les modèles doivent d'abord tous s'installer avant que l'application fonctionne.

Le choix de charger les poids des modèles dès le lancement de l'application la rend très gourmande en mémoire.

Il n'est pas possible de récupérer la réponse du LLM si elle a été interrompu.