Université de Montpellier
Faculté des Scicences
30 Place E.Bataillon, 34095 Montpellier

Année 2023-2024

Les LLM, la génération de code et l'évaluation de la qualité

HAI916I: IA pour le GL

par

Morgan NAVEL

Master Informatique - Génie Logiciel

Responsable du module : M. Abdelhak-Djamel SERIAI

Table des matières

Introduction

Les **LLMs** ou *Large Language Models* représentent aujourd'hui une des technologies les plus avancées dans le domaine de l'intelligence artificielle. Leur potentiel d'**automatisation** de diverses tâches tout au long du cycle de vie d'un logiciel suscite un intérêt croissant. Ces modèles ne se limitent pas à simplifier des tâches répétitives, mais visent également à assister les développeurs dans des étapes complexes du développement logiciel, notamment la génération de code.

Dans ce TP, nous nous concentrons sur l'utilisation des **LLMs** dans le contexte du développement de code. L'objectif est d'évaluer leur capacité à produire des résultats corrects, c'est-à-dire des extraits de code qui non seulement compilent, mais qui fonctionnent comme prévu. De plus, nous utiliserons un outil bien connu du développement logiciel, SonarQube, permettant d'évaluer la qualité du code généré (Code Smell, faille de sécurité, maintenabilité, etc.). Cette analyse permettra de mesurer à quel point ces modèles peuvent se positionner comme des outils fiables et efficaces pour les développeurs.

Génération de code

2.1 Choix technologique

J'ai décidé de développer l'application en Flutter, beaucoup de mes camarades ont choisi de le faire via Java ou Kotlin, j'ai donc voulu proposer un autre technologie afin de pouvoir obtenir des résultats varié et ainsi découvrir s'il y a des différences selon la technologie utilisées.

2.2 Présentation du sujet

L'application que nous développons vise à offrir des cours et des formations en ligne.

Fonctionnalités Principales

L'application permettra :

— De présenter les fonctionnalités principales via différents formats (vidéo, texte, etc.).
— De permettre une inscription selon plusieurs profils d'utilisateurs (parents, élèves).
— De fournir des services adaptés en fonction du profil et du mode de connexion (online, offline).

Modes de Connexion

— **Mode Online :** Accès complet aux cours, exercices et services nécessitant une connexion au serveur.
— **Mode Offline :** Accès limité aux données déjà synchronisées localement.

Inscription et Profils

- **Parent :**
  - Inscription incluant la saisie d'informations personnelles (nom, prénom, email, etc.) et des informations sur les élèves concernés (profil, niveau scolaire, etc.).
  - Choix d'une formule (documents électroniques uniquement ou accompagnement avec un professeur).
  - Gestion des paiements et génération des identifiants pour les élèves.
- **Élève :**
  - Inscription individuelle sans parents, avec accès aux cours, exercices et recommandations adaptées.

2.3 Prompt

Une partie importante pour la génération du code est de faire des prompts bien structuré, cela peut avoir un impact important sur sa génération, cela s'appelle le **Prompt Engineering**. Celon les **LLMs** cette variance est plus ou moins grande, et les méthodes de prompts peuvent être différent. Pour cela, j'ai demandé à un LLM de me créer un **prompt** pour pouvoir bien expliquer son objectif ainsi que ses spécifications.

2.4 Code

Une fois le premier prompt crée, il me suffit juste de le donner à un LLM pour qu'il me génère du code. J'ai donc réussi à générer plusieurs pages : Connexion, Inscriptions, Liste des cours. Par faute de temps, je n'ai pas pu générer plus de fonctionnalités et corriger les erreurs résiduelles. Cependant, j'ai quand même pu ajouter Firebase à mon projet, me permettant ainsi d'enregistrer des utilisateurs, et qu'il puisse s'authentifier.

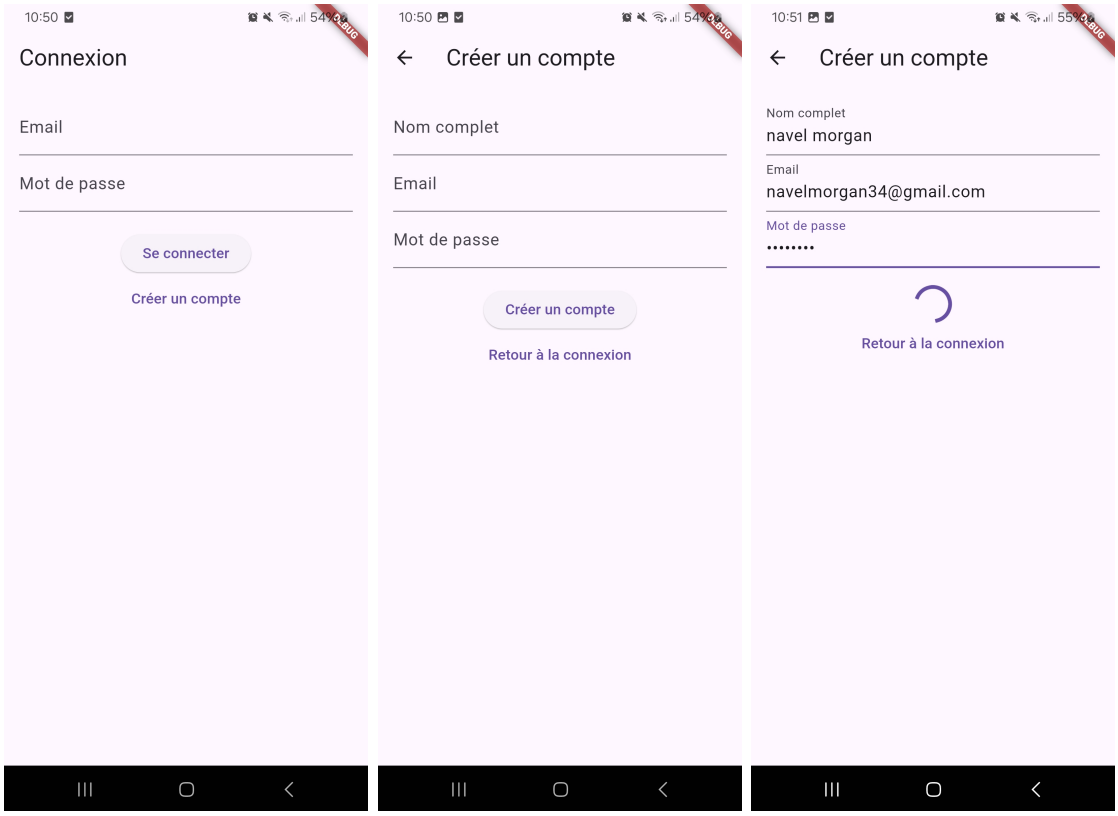

Figure 2.1 – Capture d'écran de l'application

Analyse de la qualité

Dans cette section, nous allons analyser la qualité du code. Pour ce faire, nous devons vérifier que le code respecte les bonnes pratiques, qu'il ne contient pas de bugs, et qu'il est exempt de failles de sécurité. Une analyse exhaustive réalisée manuellement serait idéale, mais cela demanderait beaucoup de temps. C'est pourquoi nous allons utiliser **SonarQube**, un outil d'analyse de code, pour évaluer les **code smells** (mauvaises pratiques de développement), identifier les failles de sécurité (comme les mots de passe ou les clés API écrits en clair), et mesurer la maintenabilité ainsi que la duplication du code.

Dans un premier temps, nous avons configuré le projet pour permettre son analyse. Cela inclut la création d'un fichier de configuration définissant les fichiers à analyser ainsi que les étapes nécessaires pour compiler l'application. De plus, nous avons intégré SonarQube avec **GitHub Actions** afin que l'analyse soit effectuée automatiquement à chaque commit.

On peut voir que le code généré est maintenable, c'est-à-dire qu'il peut évoluer avec le temps plutôt facilement. Cependant, ce code est tout de même sujet à différents problèmes :

- Problème de sécurité (Google API Key présent en clair dans le code).
- Problème sur la constance de l'application à fonctionner correctement même avec des erreurs.
- Couverture est très bas (pas de test)
- Duplication de code

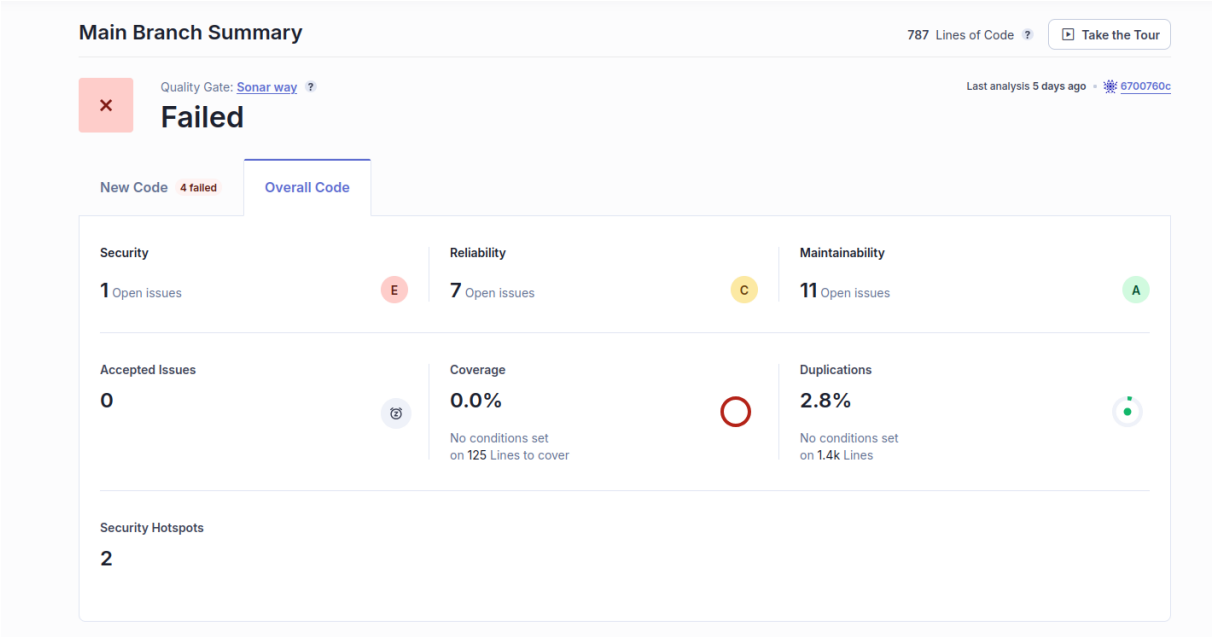

Figure 3.1 – Analyse de code avec SonarQube

Conclusion

En conclusion, le code généré par les **LLMs** est d'une qualité moyenne. Bien qu'il offre une bonne maintenabilité, il présente de nombreuses lacunes en matière de cohérence de l'application et de sécurité. Dans le cadre de ce TP, nous avons développé une petite application dans un temps limité. Cela soulève donc la question suivante : serait-il possible, dans le contexte d'un logiciel de grande envergure et avec davantage de temps, d'obtenir des résultats significativement meilleurs ?