Système d'Automatisation de Tests Logiciels Basé sur l'IA

Elassouli Nouhayla, Tizgha Nihal, Didi Alaoui Latifa,
Rouita Chaimae, Tantouft Abdelouahed

Ecole marocaine des sciences de l'ingénieur

December 26, 2024

Encadré par : Mohamed Lachgar
Date : December 26, 2024

Abstract

The *AI-based Software Test Automation System* aims to reduce manual intervention in the generation and execution of unit and functional tests. By using artificial intelligence algorithms, this system automates the creation of test scenarios, enhances test coverage, and optimizes code quality. This project addresses the need for more efficient, reliable, and scalable testing solutions in software development. It is particularly relevant for developers, quality assurance professionals, and organizations looking to improve software testing efficiency.

Keywords: *AI, Software Test Automation, Test Scenarios, Code Quality, Test Coverage, Software Development*

Contents

1 Introduction Générale **3**
1.1 Contexte et Problématique 3
1.2 Metadata 3
1.3 Objectifs du Projet 3
1.4 Motivation et Signification 4

2 Architecture Globale du Projet **4**
2.1 Microservices 4
2.2 Eureka & API Gateway 4
2.3 Frontend (ReactJS) 5
2.4 Postman 5
2.5 Communication & Sécurité 5

3 Réalisation **6**
3.1 Outils Utilisés 6
3.2 Captures d'Écran 8

4 Impact **9**

5 Assurance qualité et intégrité du code **9**

6 Conclusion et Perspectives Futures **10**
6.1 Conclusion 10
6.2 Perspectives Futures 10

1 Introduction Générale

1.1 Contexte et Problématique

Dans le contexte actuel du développement logiciel, les tests unitaires et fonctionnels sont essentiels pour garantir la fiabilité et la performance des applications. Cependant, la création et l'exécution manuelles de ces tests peuvent être sujettes à des erreurs humaines et entraîner une couverture de test insuffisante. Cette situation peut nuire à la qualité du logiciel et à la productivité des équipes de développement. Il devient donc crucial de trouver une solution pour automatiser et optimiser la génération et l'exécution des tests afin d'améliorer la qualité et d'accélérer le cycle de développement.

1.2 Metadata

Le tableau suivant contient les métadonnées essentielles relatives au projet. Il inclut des liens vers le projet Overleaf, le dépôt GitHub, ainsi que la documentation README. En plus des informations sur la version du code, les dépendances utilisées et les frameworks adoptés, ces liens permettent aux utilisateurs d'accéder facilement au projet et de consulter la documentation pour plus de détails 1.

Nr.	**Description du Metadata**	**Link**
S1	Permanent link to Overleaf project	https://www.overleaf.com/project/676ab4e3246396661ae5f43e
S2	Permanent link to GitHub repository	https://github.com/Chaimaert/Microservices-Test-App.git
S3	Link to README documentation	https://github.com/Chaimaert/Microservices-Test-App/blob/main/README.md
S4	Code version	0.0.1-SNAPSHOT
S5	Dependencies	Spring Boot, Mysql, Postgresql ,Eureka, Spring Cloud
S6	Frameworks used	Spring Boot, ReactJS, Spring Cloud

Table 1: Metadata

1.3 Objectifs du Projet

l'objectif principal de ce projet est de concevoir et de mettre en œuvre un système d'automatisation des tests unitaires basé sur une architecture microservices. Ce système sera capable de générer, analyser et rapporter automatiquement les résultats des tests unitaires grâce à l'intégration de l'API OpenAI et de plusieurs services dédiés. Le projet se divise en trois services principaux :

- **Génération automatique des tests unitaires**
- **Analyse des résultats des tests**
- **Génération de rapports sur les résultats des tests**

1.4 Motivation et Signification

Ce projet vise à améliorer l'efficacité des tests logiciels et à optimiser le processus de développement en automatisant la génération des tests unitaires, particulièrement dans les environnements Agile et DevOps. L'intégration de l'IA permet de répondre aux défis actuels liés aux tests manuels.

- **Précision des tests** : Génération automatique de tests plus précis, réduisant les erreurs humaines.
- **Gain de temps** : Automatisation des tâches répétitives pour libérer les développeurs.
- **Amélioration de la couverture de test** : L'IA génère des tests mieux adaptés aux besoins du projet.
- **Adaptabilité dans les environnements Agile et DevOps** : Accélération des cycles de développement avec une qualité constante.
- **Réduction des coûts** : Moins d'efforts manuels et réduction des erreurs, menant à des économies de temps et d'argent.

2 Architecture Globale du Projet

Le projet repose sur une architecture microservices, assurant modularité et scalabilité. Chaque fonctionnalité est isolée dans un microservice dédié.

2.1 Microservices

- **Microservice de génération des tests** : Ce service prend en charge la création des tests unitaires en analysant le code source fourni. Il génère les tests nécessaires en fonction des règles définies, permettant une couverture de tests optimale pour le code soumis.
- **Microservice de génération des rapports** :Après l'exécution des tests, ce microservice génère des rapports détaillant les résultats des tests.
- **Microservice d'analyse du code de test** : Ce service est dédié à l'analyse du code généré par le microservice de tests. Il vérifie la qualité du code, s'assure du respect des bonnes pratiques et génère des métriques pour évaluer l'efficacité des tests produits.
- **Microservice d'authentification** : Ce microservice est chargé de la gestion de l'accès au système. Il vérifie les informations d'identification des utilisateurs et émet des tokens JWT pour assurer une communication sécurisée entre les microservices et le frontend.

2.2 Eureka & API Gateway

- **Eureka** : Eureka permet à chaque microservice de s'enregistrer automatiquement dans un registre central à son démarrage. Ainsi, les autres microservices peuvent le découvrir en temps réel, ce qui rend l'architecture flexible et capable de s'adapter à des changements dans l'infrastructure .Grâce à Eureka, les microservices peuvent trouver et communiquer avec d'autres microservices. Cela facilite la communication entre services.
- **API Gateway** : agit comme un intermédiaire centralisé qui gère l'acheminement des requêtes, l'authentification, la sécurité, et la communication entre le frontend et les microservices. Elle simplifie la gestion des interactions avec les services backend tout en offrant une couche supplémentaire de sécurité et de contrôle.

2.3 Frontend (ReactJS)

permet à l'utilisateur de se connecter ou de créer un compte via les pages de login et signup. Après l'authentification, l'utilisateur peut soumettre du code, voir les résultats des tests, consulter les rapports et les télécharger.

2.4 Postman

Utilisé pour tester les APIs des microservices indépendamment.

2.5 Communication & Sécurité

- Le frontend communique avec l'API Gateway, qui redirige vers les microservices.
- Les microservices communiquent entre eux via des API REST, avec une gestion dynamique via Eureka.

La conception de l'architecture repose sur une séparation claire entre le frontend et le backend, avec une communication via API REST. Le backend est développé en Spring Boot, et le frontend est conçu en React.js. L'IA est intégrée pour automatiser la génération des tests et fournir des analyses en temps réel.

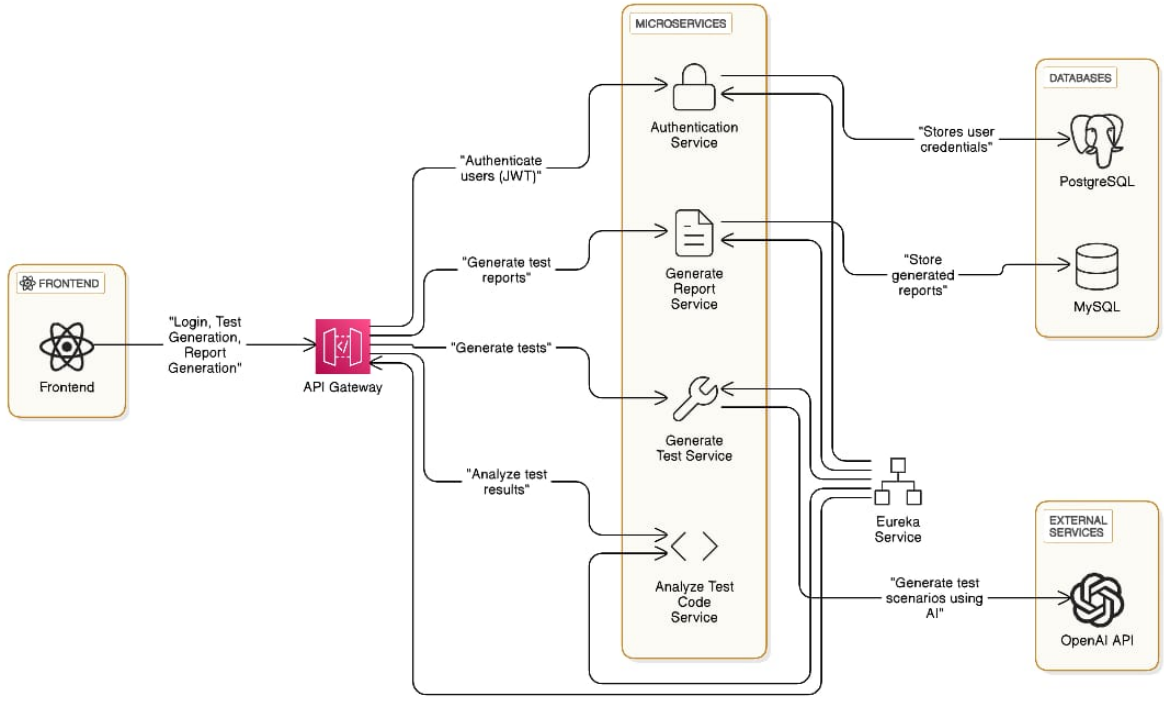


Figure 1: Architecture Globale du Projet

3 Réalisation

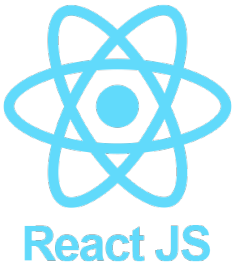
3.1 Outils Utilisés

Dans le cadre du développement de ce projet, plusieurs outils ont été utilisés pour garantir la performance, la scalabilité et la gestion efficace des différentes fonctionnalités. Voici un aperçu des principaux outils et technologies employés dans le développement :

- **Backend: Java (Spring Boot)**

Java est un langage de programmation populaire, utilisé pour la création d'applications backend robustes. Spring Boot est un framework qui simplifie le développement d'applications Java en offrant des configurations par défaut et une architecture flexible permettant de construire des microservices et des applications web performantes.

- **Frontend: React.js**



React.js est une bibliothèque JavaScript développée par Facebook, permettant de créer des interfaces utilisateurs dynamiques et réactives. Elle utilise un modèle basé sur des composants réutilisables pour construire des applications web modernes.

- **Base de Données: MySQL, Postgres**

MySQL est un système de gestion de base de données relationnelle (SGBDR) open-source populaire. Il permet de gérer des données structurées sous forme de tables. Postgres (ou PostgreSQL) est un autre SGBDR, reconnu pour sa stabilité et ses fonctionnalités avancées comme la gestion des données non structurées et la prise en charge des transactions complexes.

Postgres (ou PostgreSQL) est un système de gestion de base de données relationnelle open-source, reconnu pour sa robustesse, sa conformité aux standards SQL et ses fonctionnalités avancées telles que la gestion des données non structurées et des transactions complexes.

- **Outils Supplémentaires: SonarQube**

SonarQube est un outil d'analyse statique de code qui permet d'évaluer la qualité du code source. Il aide à détecter les bugs, les vulnérabilités et les mauvaises pratiques, tout en offrant des rapports détaillés sur la couverture de tests et la complexité du code.

- **Conteneurisation: Docker**

Docker est une plateforme qui permet de créer, déployer et exécuter des applications dans des conteneurs. Cela garantit une gestion uniforme des environnements de développement, de test et de production, et permet une mise à l'échelle rapide et une grande portabilité.

- **Postman**

Postman est un outil de développement d'API qui permet de tester, documenter et simuler des requêtes HTTP. Il est particulièrement utile pour tester des services backend, effectuer des appels API et automatiser les tests de services web.

3.2 Captures d'Écran

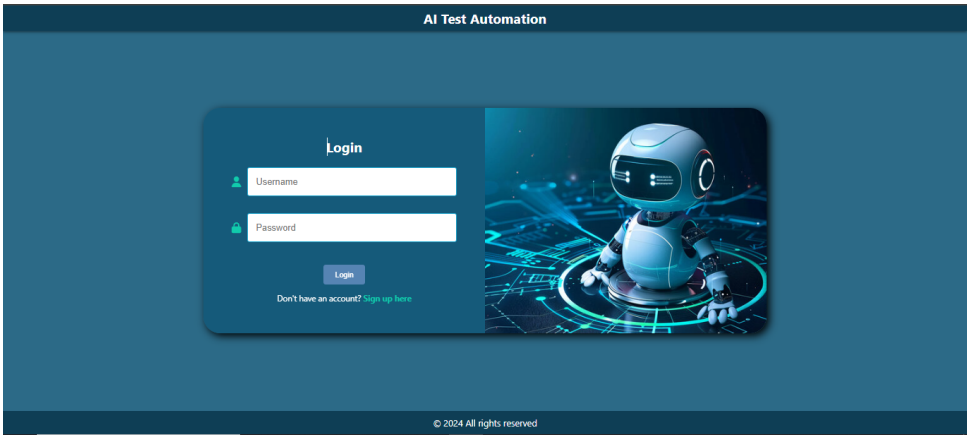


Figure 2: Interface utilisateur " Login "

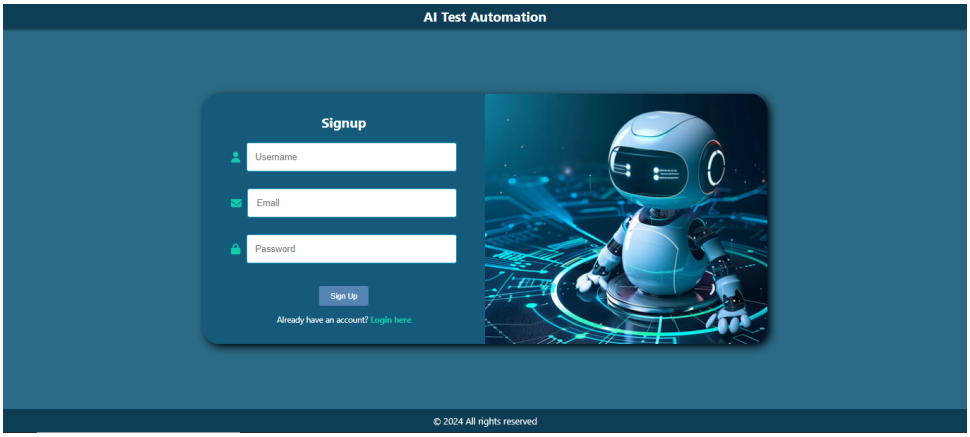


Figure 3: Interface utilisateur " Sign Up "

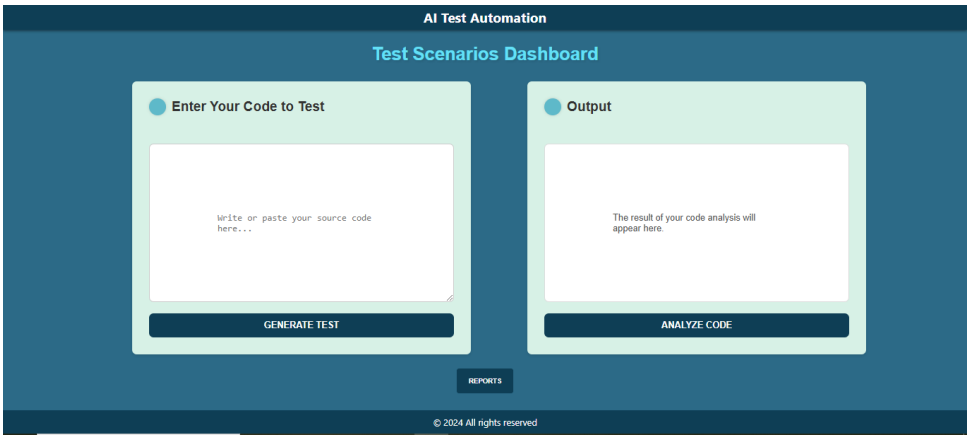


Figure 4: Interface Test Scenarios Dashboard

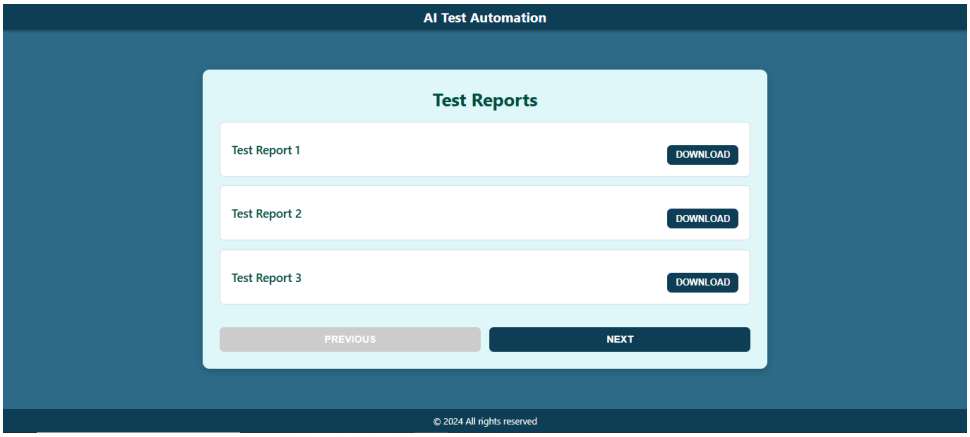


Figure 5: Test Reports Interface

4 Impact

Cette section explore en les effets positifs du projet, en soulignant ses avantages dans différents domaines comme l'efficacité, la qualité du travail, et l'apport d'innovations dans le développement logiciel.:

- Amélioration de l'efficacité : Réduction des efforts grâce à l'automatisation des tests unitaires.
- Renforcement de la qualité : Détection rapide des anomalies pour un code plus robuste.
- Optimisation des ressources : Réduction des coûts et des délais de développement.
- Innovation technologique : Intégration de l'IA pour repousser les limites des méthodologies de test traditionnelles.

5 Assurance qualité et intégrité du code

Dans ce projet, l'assurance qualité et l'intégrité du code reposent sur l'utilisation ciblée de deux outils principaux :

- SonarQube : Utilisé pour analyser le code source de manière statique, SonarQube détecte les bugs, vulnérabilités et problèmes de maintenabilité. Il a été configuré pour générer des rapports réguliers permettant de suivre l'évolution de la qualité du code et de garantir son alignement avec les standards du projet.
- Docker : Docker est exploité pour conteneuriser l'ensemble des services du projet, y compris l'environnement d'exécution de SonarQube. Cette approche garantit la reproductibilité des analyses de code dans des environnements uniformes et simplifie le déploiement des services dans différents contextes.

Grâce à ces outils, le projet assure un suivi rigoureux de la qualité du code, tout en facilitant la gestion des environnements et des dépendances.

6 Conclusion et Perspectives Futures

Le Système d'Automatisation de Tests Logiciels Basé sur l'IA offre une solution innovante pour l'automatisation des tests logiciels. Il améliore la précision des tests et permet de gagner du temps dans les cycles de développement, ce qui est essentiel dans des environnements de développement rapide.

6.1 Conclusion

Ce projet a permis de développer un système d'automatisation des tests unitaires efficace et adaptable, en utilisant des microservices et des outils d'intelligence artificielle. Il a démontré l'importance d'améliorer la couverture des tests et de réduire les efforts manuels dans le processus de développement. Grâce à l'utilisation de technologies modernes comme SonarQube et Docker, l'intégrité et la qualité du code sont garanties, tout en offrant une base solide pour des améliorations futures.

6.2 Perspectives Futures

Les perspectives futures de ce projet se concentrent sur plusieurs axes de développement afin d'enrichir ses fonctionnalités et d'améliorer ses performances. Tout d'abord, l'extension des capacités de l'intelligence artificielle sera envisagée pour permettre la prédiction des résultats des tests, ce qui offrira une approche plus proactive dans l'identification des problèmes. Par ailleurs, une amélioration de l'interface utilisateur est prévue, avec l'ajout d'outils de visualisation avancés pour offrir une expérience utilisateur plus intuitive et interactive. Enfin, l'intégration de tests distribués sur des plateformes cloud devrait être mise en place pour améliorer l'évolutivité et permettre de traiter des volumes de données encore plus importants tout en optimisant les performances globales du système.

References

[1] W3Schools, *Java*, https://www.w3schools.com/java/

[2] W3Schools, *React.js*, https://www.w3schools.com/react/

[3] W3Schools, *MySQL*, https://www.w3schools.com/sql/

[4] PostgreSQL, *PostgreSQL Documentation*, https://www.postgresql.org/docs/

[5] SonarQube, *SonarQube Features*, https://www.sonarqube.org/features/

[6] Docker, *What is a Container?*, https://www.docker.com/resources/what-container

[7] Postman,*API Documentation Tool*, https://www.postman.com/api-documentation-tool