ECOLE MAROCAINE DES SCIENCE DE L'INGENIEUR

ET

EMSIJourney

Architecture des composants d'entreprise

Realisation test déploiement d'un application de blogs de voyages

January 20, 2024

Rapport de Projet

Amine Frira
Mohamed Amine Elbahaoui
Mohamed Adnane Jammoua

Encadre par:
Mohamed Lachgar, Professeur Universitaire

Contents

1 Introduction

Dans le cadre du module de gestion de la qualité à l'université, nous avons été chargés de développer une application de microservices, spécifiquement un blog de voyages. Ce projet a pour objectif non seulement de concevoir une application fonctionnelle mais aussi de mettre en œuvre et de tester divers aspects de la qualité logicielle. L'application a été conçue en utilisant une architecture de microservices, intégrant divers outils et technologies modernes.

2 Aperçu du Projet

L'application, intitulée "Blog de Voyages", est structurée autour de trois microservices principaux : le service d'authentification, le service de blog et le service social. Chacun de ces services utilise sa propre base de données, gérée par PostgreSQL et dockerisée pour garantir la portabilité et la cohérence de l'environnement.

Pour la découverte des services, nous avons opté pour Eureka Discovery Server. Cela facilite la communication et l'orchestration entre les différents microservices. En outre, un API Gateway est utilisé pour gérer efficacement les requêtes entrantes et diriger celles-ci vers les services appropriés.

Concernant la partie frontale de l'application, Angular a été choisi pour son efficacité et sa capacité à créer des interfaces utilisateur dynamiques et réactives. En outre, l'utilisation de Zipkin pour visualiser la latence des services ajoute une couche supplémentaire de monitoring, cruciale pour l'analyse des performances et la résolution des problèmes.

L'accent de ce projet est mis sur les tests et la qualité du code. Pour cela, plusieurs types de tests ont été implémentés : des tests unitaires, des tests avec Selenium pour l'interface utilisateur et l'intégration des tests avec SonarQube pour une analyse approfondie de la qualité du code. Ces tests sont essentiels pour s'assurer que l'application est non seulement fonctionnelle mais aussi fiable, sécurisée et performante.

Dans les sections suivantes, nous détaillerons davantage chacun de ces aspects, en mettant particulièrement l'accent sur les stratégies de test et leur mise en œuvre dans le cadre de ce projet.

3 Importance de l'Architecture Microservices

- **Scalabilité et Flexibilité** L'architecture microservices permet au blog de voyage de faire évoluer individuellement les composants de l'application. À mesure que la base d'utilisateurs grandit et que de nouvelles fonctionnalités sont requises, il devient plus facile d'allouer des ressources précisément là où elles sont nécessaires, sans avoir à faire évoluer toute l'application.

- **Développement et Déploiement Rapides** En décomposant l'application en services plus petits et gérables, différentes équipes peuvent travailler simultanément sur différentes parties du système. Cela conduit à des temps de développement et de déploiement plus rapides, ce qui est crucial pour maintenir le blog de voyage à jour avec les dernières tendances de voyage et avancées technologiques.

- **Meilleure Isolation des Fautes** Dans une architecture microservices, les problèmes dans un service n'affectent pas nécessairement l'ensemble de l'application. Cette isolation réduit les temps d'arrêt et améliore l'expérience utilisateur globale sur le blog de voyage, garantissant que la plateforme reste opérationnelle même si l'un des services rencontre des problèmes.

- **Flexibilité Technologique** Les microservices permettent l'utilisation de différentes technologies et frameworks mieux adaptés à des fonctionnalités spécifiques. Ce projet tire parti de Spring Boot pour

sa robustesse, sa facilité d'utilisation et son large soutien communautaire, ce qui en fait un choix idéal pour développer des applications de niveau entreprise comme notre blog de voyage.

- **Livraison et Intégration Continues** Les microservices supportent la livraison et l'intégration continues, permettant des mises à jour régulières du blog de voyage avec une perturbation minimale du service. C'est essentiel pour intégrer les retours des utilisateurs, corriger les bugs et ajouter de nouvelles fonctionnalités pour garder la plateforme engageante et actuelle.

En conclusion, l'architecture microservices, renforcée par Spring Boot, forme l'épine dorsale de notre projet de blog de voyage, fournissant une plateforme résiliente, scalable et conviviale pour les passionnés de voyage afin de se connecter et partager leurs expériences.

4 Architecture Microservices

4.1 Architecture

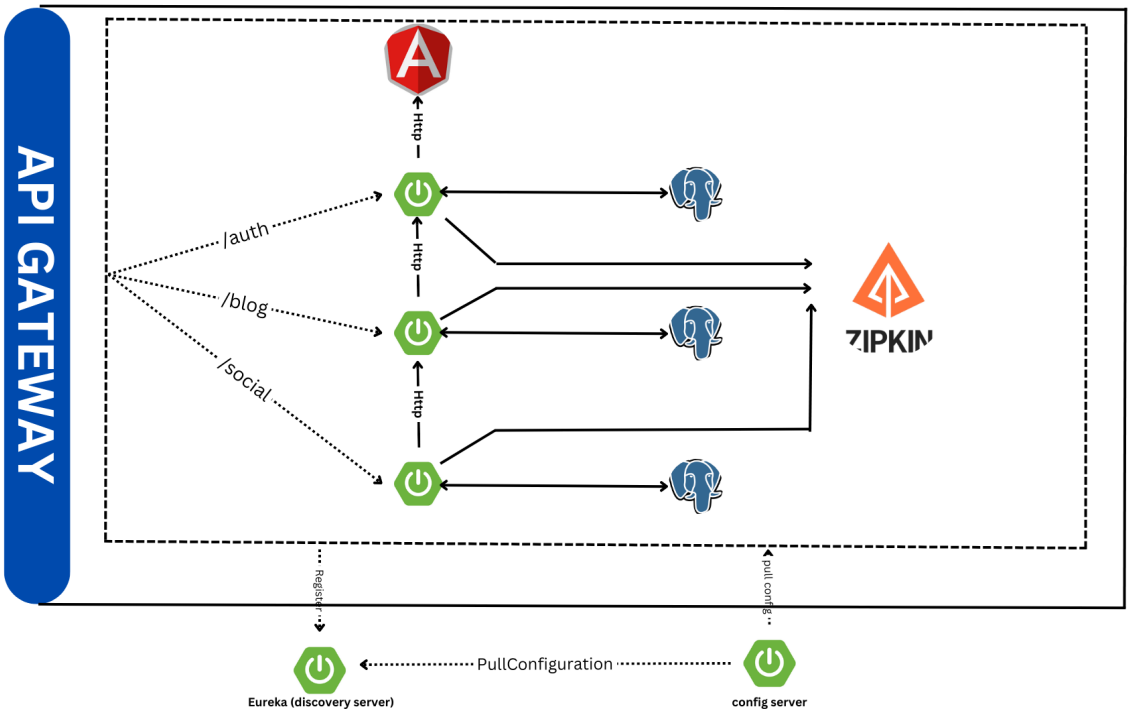

Figure 1: Figure de l'architecture de l'application.

L'application de blog de voyage adopte une architecture microservices, divisée en services distincts mais interconnectés, chacun dédié à une fonctionnalité spécifique. Cette approche offre une modularité accrue, permettant à chaque service de fonctionner de manière indépendante tout en collaborant pour former un système cohérent. L'architecture est soutenue par un serveur de découverte Eureka, qui facilite la localisation et le balancement de charge des différents microservices. En outre, un serveur de configuration centralisé gère toutes les configurations de l'application, tandis qu'une passerelle API agit comme un point d'entrée unique pour les requêtes externes, offrant une couche de sécurité supplémentaire et la gestion du trafic.

4.2 Description des Services

- **Service d'Authentification** Utilisant Spring Security et JWT (JSON Web Token), ce service gère l'authentification et l'autorisation des utilisateurs. Il assure la sécurité des données utilisateur et permet un accès sécurisé

aux autres microservices.

- **Service Social** Ce service gère les aspects sociaux du blog, tels que les commentaires, les likes et les upvotes des articles de blog. Il offre aux utilisateurs une plateforme interactive pour engager des discussions et partager des opinions.
- **Service de Blog** Le cœur de l'application, ce service gère la création, la modification, et l'affichage des articles de blog. Il traite les demandes de contenu, maintient la base de données des articles et assure une présentation efficace des informations de voyage.

4.3 Mécanismes de Communication

Pour la communication entre les différents microservices, l'application utilise OpenFeign, un client déclaratif de services web. OpenFeign simplifie le développement des clients HTTP, permettant une communication fluide et efficace entre les services. En encapsulant les appels de service dans des interfaces Java simples, il facilite l'intégration et la maintenance du code. De plus, la combinaison d'Eureka et OpenFeign permet un découplage des services, où les services peuvent communiquer sans connaître les adresses physiques des autres services, renforçant ainsi la résilience et la scalabilité de l'architecture.

5 Conception des Microservices

Approche de Conception pour Chaque Service
L'architecture de l'application de blog de voyage a été soigneusement conçue pour optimiser la performance, la maintenance, et l'évolutivité. Chaque service a été développé avec une approche spécifique, répondant à des besoins distincts tout en contribuant à l'ensemble du système.

5.1 Service d'Authentification

- **Raison de la Séparation** La sécurité est primordiale dans les applications web, surtout lorsqu'il s'agit de données personnelles et de contenu généré par l'utilisateur. Séparer le service d'authentification permet une gestion plus concentrée et sécurisée des informations d'identification des utilisateurs.
- **Avantages** Cette séparation facilite l'intégration de mécanismes de sécurité avancés, comme Spring Security et JWT, sans surcharger les autres services avec des préoccupations de sécurité. Elle permet également des mises à jour et des modifications de la politique de sécurité sans perturber le fonctionnement des autres services.

5.2 Service Social

- **Raison de la Séparation** Les interactions sociales comme les commentaires, les likes et les upvotes nécessitent une gestion dynamique et souvent complexe des données. En isolant ces fonctionnalités, le service social peut être optimisé pour gérer efficacement les interactions en temps réel.
- **Avantages** Cette approche offre une meilleure performance dans la gestion des interactions des utilisateurs et permet une évolutivité indépendante à mesure que la communauté du blog grandit. De plus, cela simplifie l'intégration de nouvelles fonctionnalités sociales à l'avenir.

5.3 Service de Blog

- **Raison de la Séparation** Le cœur de l'application, la gestion des articles de blog, nécessite une attention particulière pour le stockage, la récupération et la présentation des contenus. Séparer ce service permet une gestion plus ciblée et efficace des données de contenu.
- **Avantages** Cela permet une évolution et une maintenance plus aisées du système de gestion de contenu. La séparation garantit également que les mises à jour ou les problèmes liés au contenu du blog n'affectent pas les autres aspects fonctionnels de l'application.

En conclusion, la conception de chaque microservice a été guidée par le principe de responsabilité unique. Cette approche garantit que chaque service est suffisamment indépendant pour fonctionner, évoluer, et être maintenu séparément, tout en contribuant à l'efficacité et à la robustesse globales de l'application de blog de voyage.

6 Conteneurisation avec Docker

6.1 Implémentation

Dans le cadre de notre projet de blog de voyage, nous avons adopté Docker pour la conteneurisation des différents microservices. Docker fournit une plateforme pour empaqueter chaque service dans son propre conteneur, isolant l'environnement d'exécution de chacun. Ce processus commence par la création d'images Docker personnalisées pour chaque microservice, incluant le service d'authentification, le service social et le service de blog. Ces images contiennent le code, les bibliothèques, les dépendances et les variables d'environnement nécessaires à l'exécution de chaque service.

Après la création des images, elles sont déployées dans des conteneurs Docker. Grâce à la nature légère et portable des conteneurs, ils peuvent être facilement déployés sur divers environnements de développement, de test et de production, garantissant une cohérence dans toutes les phases du cycle de vie du logiciel.

6.2 Avantages

- **Isolation et Consistance** Docker garantit que chaque microservice s'exécute dans un environnement isolé, réduisant les conflits entre services et facilitant la gestion des dépendances. Cette isolation assure également une consistance entre les environnements de développement et de production, minimisant les problèmes de "ça marche sur ma machine".
- **Déploiement Rapide et Scalabilité** Les conteneurs Docker peuvent être démarrés, arrêtés et redimensionnés rapidement et efficacement. Cela facilite un déploiement rapide des services et une scalabilité horizontale, essentielle pour répondre aux variations de la demande sur le blog de voyage.
- **Maintenance et Mises à Jour Facilitées** La conteneurisation simplifie la maintenance et les mises à jour des applications. Les changements peuvent être apportés à l'image d'un service et redéployés sans perturber les autres services, rendant le processus de mise à jour plus agile et moins risqué.
- **Portabilité et Intégration Continue** Docker offre une portabilité accrue, permettant à l'application de s'exécuter de manière uniforme sur différentes plates-formes et environnements cloud. Cette caractéristique, combinée à l'intégration continue, facilite les tests automatisés et le déploiement continu, accélérant le cycle de développement.

En intégrant Docker dans l'architecture microservices de notre application de blog de voyage, nous avons créé un écosystème robuste, agile et facilement évolutif, répondant aux exigences modernes de développement de logiciels et offrant une expérience utilisateur optimale.

7 CI/CD avec Jenkins

7.1 Processus

Pour automatiser le développement et le déploiement de notre application de blog de voyage, nous avons mis en place un pipeline CI/CD en utilisant Jenkins. Jenkins, un outil d'automatisation open-source, facilite l'intégration et le déploiement continus en automatisant diverses étapes du cycle de développement.

Le processus commence par la configuration des jobs Jenkins pour chaque microservice de l'application (Service d'Authentification, Service Social, et Service de Blog). Chaque fois qu'un développeur effectue un commit dans le référentiel de code source (par exemple, GitHub ou Bitbucket), Jenkins déclenche un processus automatisé. Ce processus comprend les étapes suivantes :

- **Extraction du Code** Jenkins récupère la dernière version du code source depuis le référentiel.
- **Construction et Tests** Le code est construit, et des tests automatisés sont exécutés pour vérifier son intégrité et sa performance.
- **Analyse de la Qualité du Code** Des outils d'analyse de code tels que SonarQube peuvent être intégrés pour évaluer la qualité du code.
- **Conteneurisation** Si le code passe les tests, Jenkins procède à la conteneurisation en utilisant Docker, créant des images pour chaque service.
- **Déploiement** Les images Docker sont déployées dans l'environnement de test.

7.2 Configuration

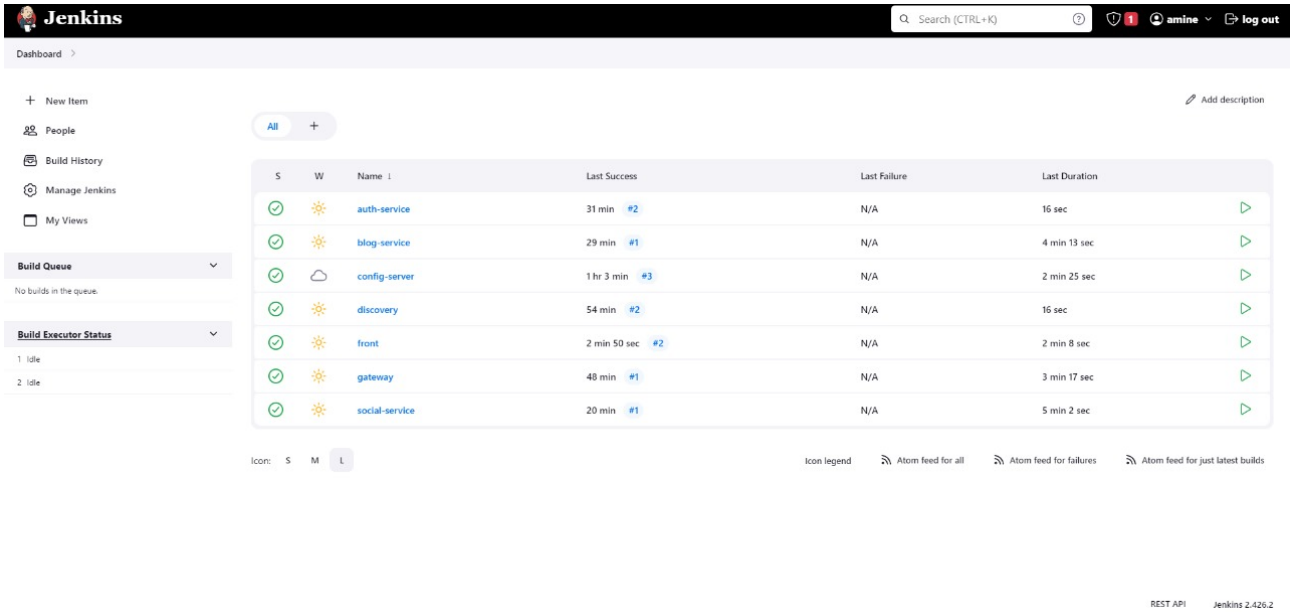

Figure 2: Exemple de probleme de securite.

La configuration de Jenkins est réalisée à travers des fichiers Jenkinsfile, qui définissent les pipelines pour chaque service. Ces fichiers sont stockés dans le référentiel de code source, permettant une configuration en tant que code (Infrastructure as Code, IaC). Ceci assure une grande transparence, une maintenance facile, et une synchronisation parfaite avec les évolutions du code de l'application.

Pour faciliter la communication entre Jenkins et les différents outils et services (comme les serveurs Git, Docker, et les environnements de déploiement), des plugins spécifiques sont installés et configurés dans Jenkins. Cela permet une intégration fluide et une automatisation efficace du pipeline CI/CD.

En résumé, l'implémentation de CI/CD avec Jenkins dans notre projet de blog de voyage offre un processus de développement et de déploiement rapide, fiable et cohérent, réduisant significativement le risque d'erreurs humaines et accélérant le temps de mise sur le marché des nouvelles fonctionnalités et mises à jour.

8 Déploiement Automatique

Utilisation de Google Cloud

Figure 3: Instance du virtual machine dans google cloud .

Pour le déploiement automatique de notre application de blog de voyage, nous avons opté pour l'utilisation des machines virtuelles (VM) de Google Cloud Platform (GCP). Cette approche tire parti de la flexibilité et de la puissance des VM de Google Cloud pour héberger et déployer efficacement notre application.

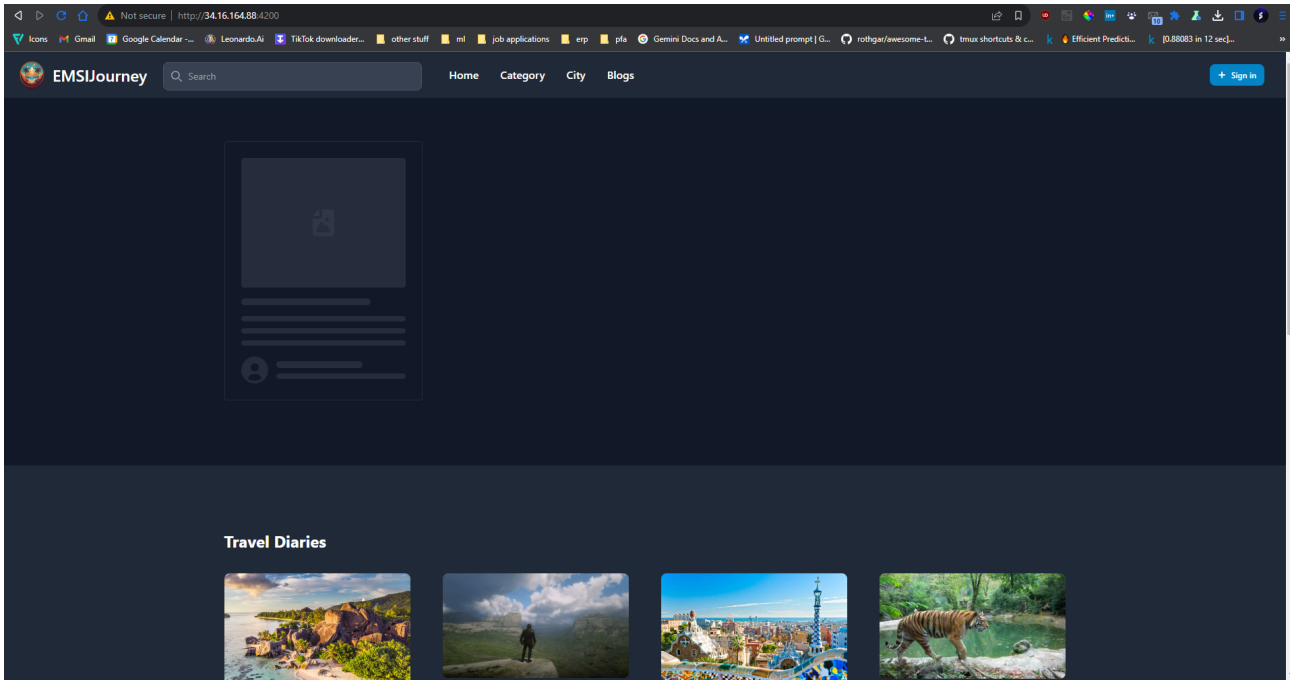

Figure 4: Instance du virtual machine dans google cloud .

- **Configuration des VM sur Google Cloud:** Nous commençons par configurer des VM sur Google Cloud, en choisissant les spécifications adaptées aux besoins de notre application en termes de CPU, de mémoire et de stockage. Ces VM servent d'hôtes pour nos microservices conteneurisés.
- **Préparation de l'Environnement de VM** Chaque VM est préparée avec l'environnement nécessaire, y compris l'installation de Docker pour la gestion des conteneurs et toutes les autres dépendances

requises pour l'exécution de notre application.

- **Contrôle Complet** L'utilisation des VM Google Cloud nous donne un contrôle total sur l'environnement d'exécution, nous permettant de personnaliser l'infrastructure selon nos besoins spécifiques.
- **Flexibilité et Scalabilité** Nous pouvons facilement ajuster les ressources allouées aux VM, permettant une scalabilité efficace en réponse aux changements de demande de l'application.

En déployant notre application de blog de voyage sur des VM Google Cloud, nous bénéficions d'une solution de déploiement robuste, sécurisée et hautement personnalisable, adaptée aux exigences et à la croissance de notre plateforme.

9 Planification du Projet et Méthodologie Agile

9.1 Utilisation de la Méthodologie Scrum

Pour la planification et la gestion de notre projet de blog de voyages, nous avons adopté la méthodologie Scrum, un cadre de travail Agile qui favorise la collaboration, la flexibilité et l'amélioration continue. Cette approche nous a permis de décomposer le projet en sprints, des cycles de développement courts et itératifs, facilitant ainsi une progression constante et une adaptation rapide aux changements.

9.2 Distribution des Tâches au sein de l'Équipe

L'équipe du projet était composée de trois membres : Amine Frira, Mohamed Adnane Jammoua et Mohamed Amine Elbahaoui. Chacun de nous a apporté des compétences uniques et complémentaires, ce qui a été crucial pour le succès du projet.

- **Amine Frira** s'est concentré sur la conception et le développement du service d'authentification. Sa responsabilité incluait la mise en place de la sécurité et la gestion des utilisateurs.
- **Mohamed Adnane Jammoua** a pris en charge le développement du service de blog. Il était responsable de la création des fonctionnalités permettant aux utilisateurs de publier, de modifier et de visualiser des articles de blog.
- **Mohamed Amine Elbahaoui** a travaillé sur le service social de l'application, en développant des fonctionnalités pour la gestion des interactions sociales telles que les commentaires et les likes.

En plus de ces rôles spécifiques, tous les membres de l'équipe ont participé activement à la planification des sprints, aux réunions quotidiennes de Scrum, aux revues de sprint et aux rétrospectives. Cette collaboration a permis une répartition équilibrée des tâches et une communication efficace tout au long du projet.

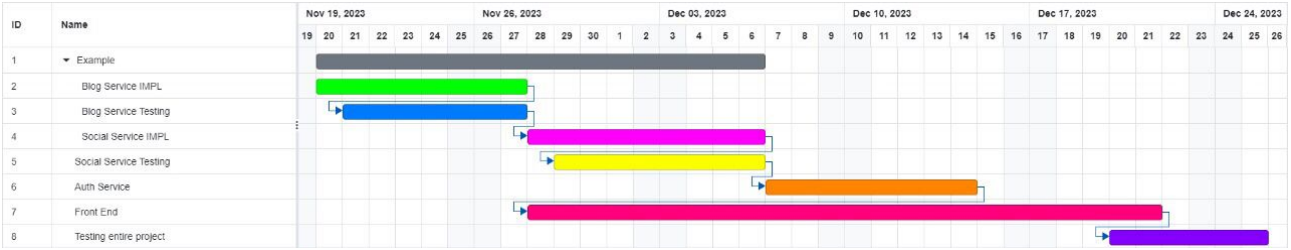

Figure 5: Diagramme de gantt.

10 Outils et Technologies Utilisés

10.1 Introduction

L'efficacité et la qualité de notre application de blog de voyages reposent sur une sélection judicieuse d'outils et de technologies. Cette section met en lumière les outils clés que nous avons utilisés pour le développement, les tests et la conception de l'application.

10.2 Technologies de Base

- **Spring Boot pour Microservices** Nous avons utilisé Spring Boot pour créer trois microservices distincts : authentification, blog et service social. Spring Boot nous a aidé à simplifier le processus de développement grâce à sa configuration automatique et à sa gestion facile des dépendances.
- **PostgreSQL Dockerisé** Chaque microservice utilise sa propre base de données PostgreSQL, dockerisée pour garantir la cohérence et la facilité de déploiement.
- **Eureka Discovery Server** Ce serveur a été utilisé pour la découverte des services, facilitant la communication entre les différents microservices.
- **API Gateway** Un gateway d'API pour gérer les requêtes entrantes et les diriger vers les services appropriés.
- **Angular pour le Front-End** Angular a été choisi pour développer l'interface utilisateur, offrant une expérience riche et interactive.
- **Zipkin** Employé pour visualiser la latence des services, crucial pour le monitoring et l'optimisation des performances.

10.3 Outils de tests

- **Junit/mockito pour les Tests Unitaires** mockito est utilisé pour simuler la base de données lors des tests de la couche de service, permettant des tests isolés et efficaces.
- **Selenium pour les Tests d'Interface Utilisateur** Des tests automatisés avec Selenium ont été réalisés pour s'assurer que l'interface utilisateur fonctionne comme prévu.
- **SonarQube pour les Tests de Qualité du Code** SonarQube a été utilisé pour une analyse approfondie de la qualité du code, s'assurant de sa maintenabilité, de sa sécurité et de son efficacité.
- **Burp Suite pour la Sécurité Web** Employé pour effectuer des tests de sécurité approfondis sur l'application web.

10.4 Dockerisation et design

- **Docker** Utilisé pour la containerisation des microservices et des bases de données, offrant un environnement cohérent et portable.
- **Flowbite et TailwindCSS pour le Design** ont été utilisés pour développer une interface utilisateur attrayante et intuitive.

10.5 Conclusion

L'ensemble de ces outils et technologies a joué un rôle vital dans le succès de notre projet. Chaque choix technologique a été fait avec soin pour s'assurer qu'il répond aux besoins spécifiques de l'application, de sa sécurité, de sa performance et de sa facilité d'utilisation. La combinaison de ces technologies a permis de créer une application robuste, sécurisée et performante.

11 Réalisation

11.1 Démonstration de l'Application

Dans cette section, nous présentons une vidéo de démonstration qui met en lumière les fonctionnalités clés de notre application de blog de voyages. Cette vidéo offre un aperçu complet et interactif de l'expérience utilisateur proposée par l'application. Voici les fonctionnalités démontrées :

- **Inscription et Connexion** L'utilisateur peut facilement s'inscrire et se connecter à l'application. Ce processus est conçu pour être intuitif et sécurisé, offrant une première étape essentielle pour accéder aux fonctionnalités personnalisées.
- **Création de Ville et Catégorie de Blog** Les utilisateurs ont la possibilité de créer de nouvelles villes et catégories pour leurs blogs. Cette fonctionnalité permet une organisation et une classification efficaces des articles de blog, facilitant ainsi la navigation et la recherche par les utilisateurs.
- **Rédaction de Blog en Markdown** L'application offre une fonctionnalité de rédaction de blog en Markdown. Les utilisateurs peuvent écrire leurs articles de blog en utilisant le langage de balisage Markdown, qui est ensuite rendu de manière élégante dans l'interface utilisateur, permettant une présentation claire et professionnelle du contenu.
- **Écriture de Commentaires** Les utilisateurs peuvent interagir avec les articles de blog en écrivant des commentaires. Cette fonctionnalité encourage l'engagement et la discussion au sein de la communauté des utilisateurs.
- **Aimer un Commentaire ou un Post** L'application permet aux utilisateurs d'exprimer leur appréciation pour un commentaire ou un article de blog en cliquant sur 'J'aime'. Cela ajoute un élément social et interactif à l'expérience de l'utilisateur.

Clicker ici pour voir la video de demonstration

11.2 Tests Unitaires

Dans cette partie, nous mettons en lumière les tests unitaires effectués à différents niveaux de notre application, illustrés par trois images spécifiques.

- **Test de Repository avec Base de Données H2** La première image présente un test de repository. Pour ces tests, nous avons utilisé la base de données en mémoire H2, permettant de simuler un environnement de base de données réel pour tester la couche de persistance de l'application. Cela nous a aidé à valider le bon fonctionnement des opérations de base de données comme les requêtes, les insertions et les mises à jour, sans impacter la base de données principale.

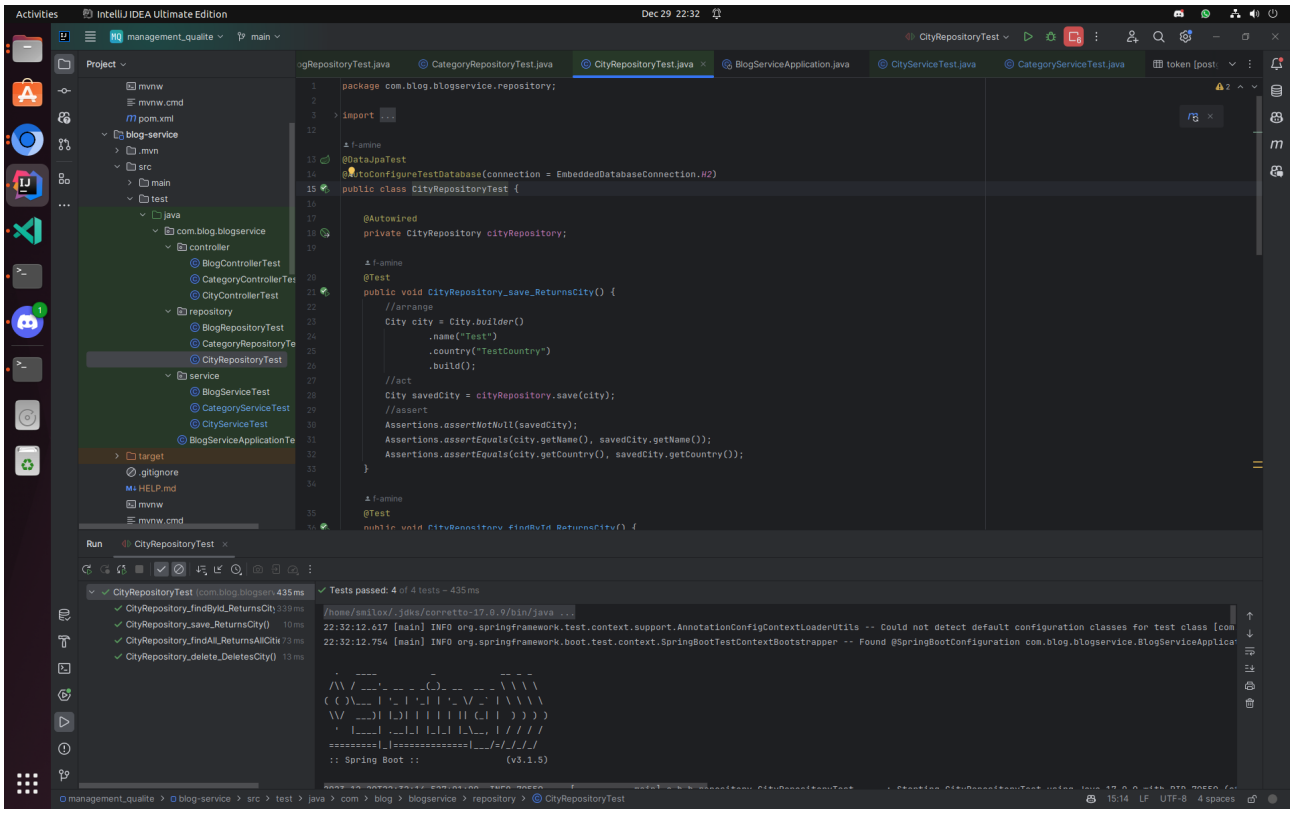

Figure 6: Test Unitaire de couche repository.

- **Test de Service avec Mockito** La deuxième image illustre un test de la couche de service en utilisant Mockito. Cette approche nous a permis de simuler les dépendances externes et de concentrer les tests sur la logique métier. En mockant les réponses des dépendances, nous avons pu vérifier que les services réagissent correctement aux divers scénarios, assurant ainsi la fiabilité et la robustesse de la logique métier.

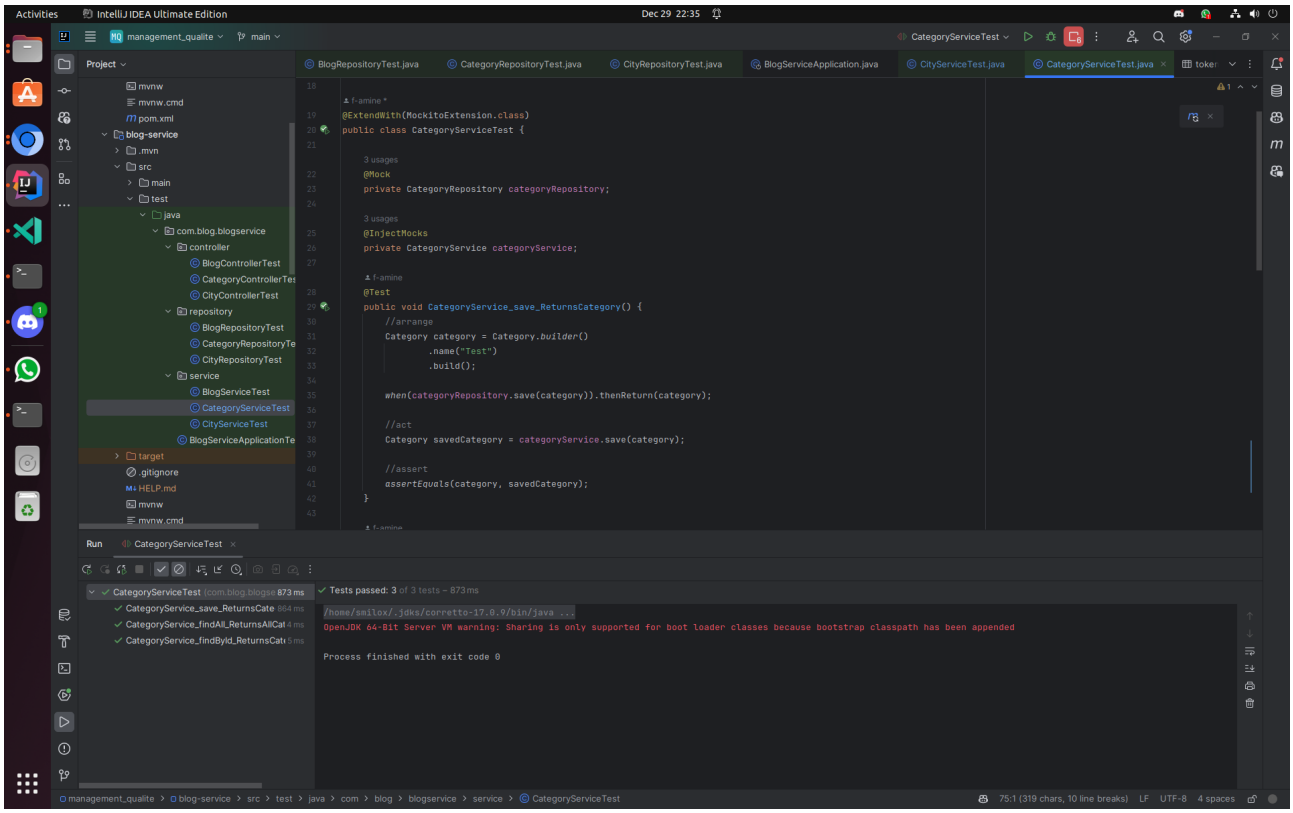

Figure 7: Test Unitaire de couche service.

- **Test de Controller avec MockMvc** La troisième image montre un test de controller réalisé avec MockMvc. Cette technique est cruciale pour tester les points de terminaison de l'API REST, en simulant des requêtes HTTP et en vérifiant les réponses. Cela garantit que les controllers traitent correctement les requêtes et retournent les réponses attendues, conformément aux spécifications de l'API.

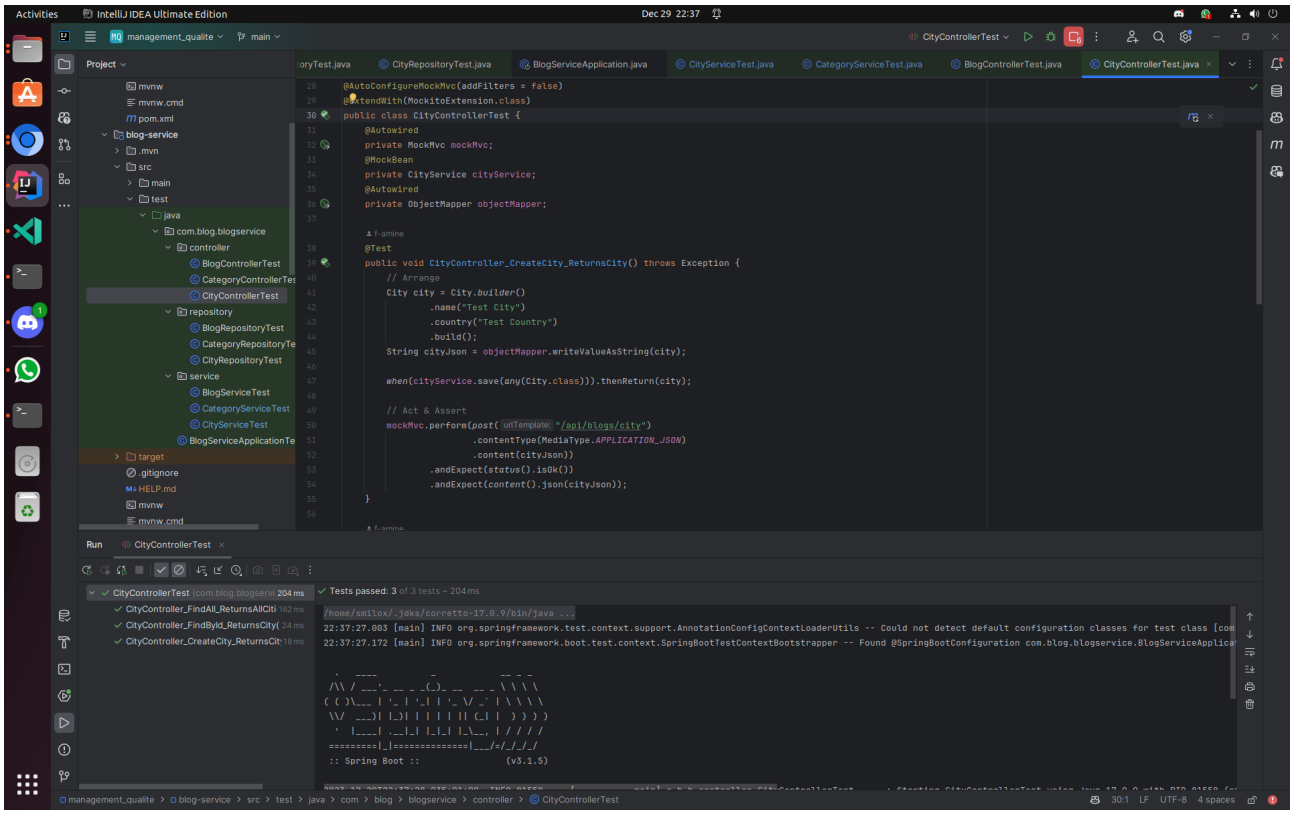

Figure 8: Test Unitaire de couche controlleur.

Ces tests unitaires couvrent de manière exhaustive les différentes couches de l'application, s'assurant que chaque composant fonctionne individuellement comme prévu. En combinant ces approches de test, nous avons créé une base solide pour une application fiable et bien structurée.

11.3 Exemples de Tests avec Selenium

Scénarios de Test Automatisés pour les Fonctionnalités Clés
Cette section illustre comment nous avons utilisé Selenium pour automatiser les tests de certaines des fonctionnalités les plus importantes de l'application, avec deux captures d'écran représentatives.

- **Scénario d'authentification** La première capture d'écran montre le scénario de test pour la fonctionnalité de connexion. Dans ce test, Selenium automatise le processus de saisie des identifiants par l'utilisateur et de soumission du formulaire de connexion. L'objectif est de vérifier si l'application permet une connexion réussie avec des identifiants valides et gère correctement les tentatives de connexion erronées. Ce test est essentiel pour assurer que les utilisateurs peuvent accéder à leur compte de manière sécurisée et fiable.

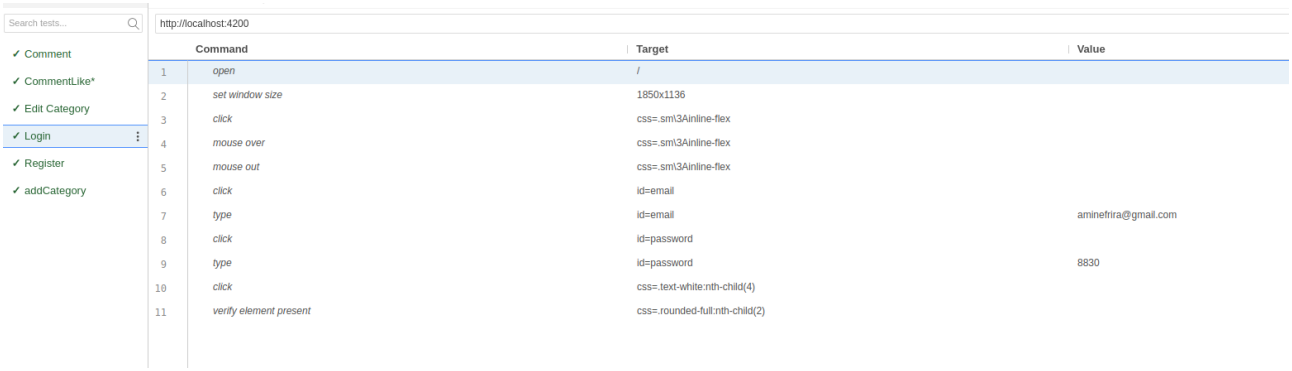

Figure 9: Test Selenium d'authentification.

- **Scénario de Création de Commentaire** En plus des scénarios de connexion et de création de blog, une autre capture d'écran illustre le test automatisé pour la fonctionnalité de création de commentaire. Ce scénario spécifique avec Selenium simule un utilisateur ajoutant un commentaire sur un blog existant.

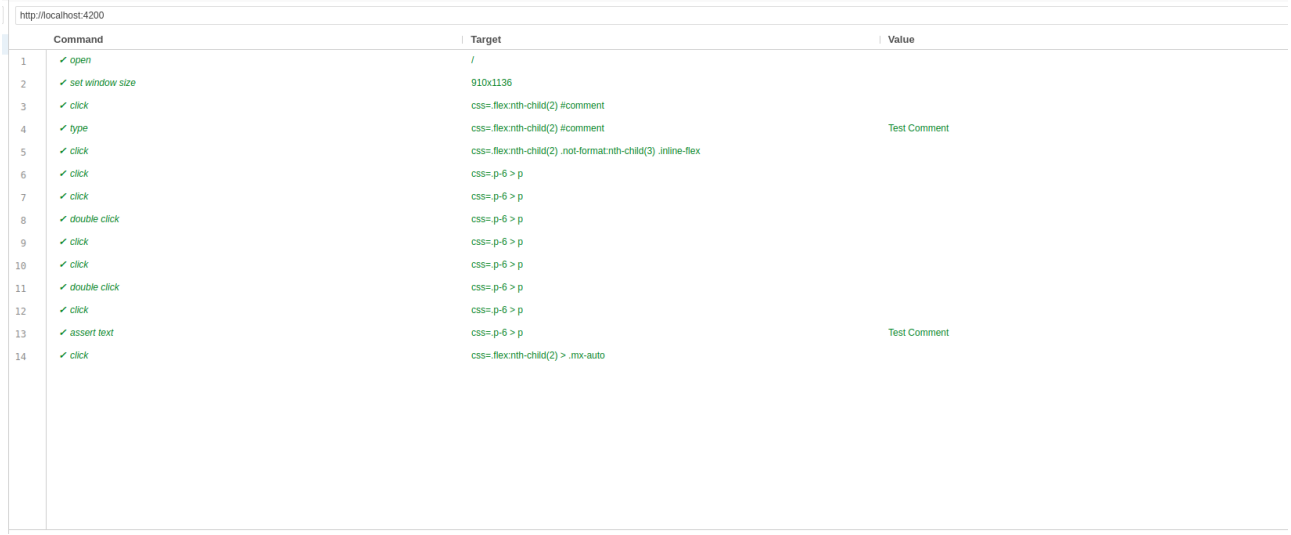

Figure 10: Test Selenium de creation d'un commentaire.

Ces tests automatisés avec Selenium jouent un rôle vital dans l'assurance qualité de notre application. Ils permettent de tester les fonctionnalités de manière exhaustive et répétitive, garantissant une expérience utilisateur cohérente et sans erreur.

11.4 Analyse du Rapport SonarQube

Vue d'ensemble et Exemples Spécifiques des Problèmes Identifiés
Dans cette section, nous explorons les résultats de l'analyse de qualité du code réalisée avec SonarQube, un outil essentiel pour assurer la maintenabilité, la sécurité et la qualité globale de notre code source. Les captures d'écran fournies offrent un aperçu détaillé des problèmes identifiés et des domaines à améliorer.

- **Vue d'Ensemble du Rapport** La première image présente une vue d'ensemble du rapport SonarQube, indiquant 29 problèmes de maintenabilité, 2 problèmes de sécurité et 2 bugs. Cette vue globale donne une indication claire des domaines où le code pourrait être optimisé pour améliorer la qualité et la performance de l'application.

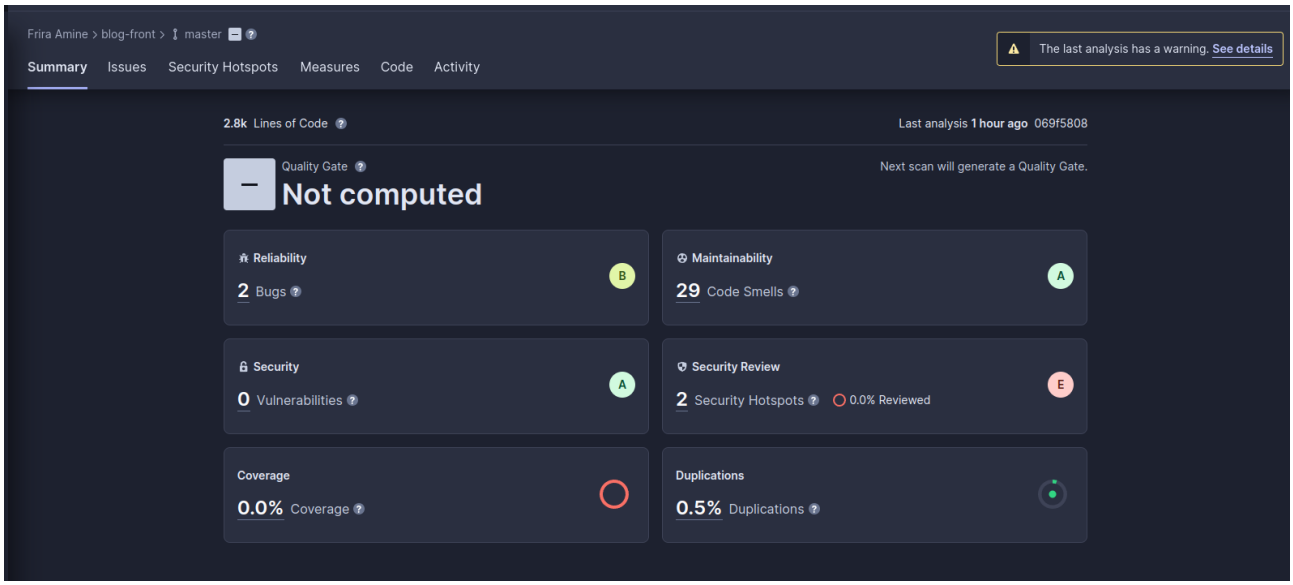

Figure 11: Rapport sonarqube.

- **Exemple de Problème de Maintenabilité** Une capture d'écran montre un exemple spécifique de problème de maintenabilité identifié par SonarQube. Ces problèmes peuvent inclure des codes complexes, des blocs de code mal structurés ou des pratiques de codage qui rendent le code difficile à lire et à maintenir. En résolvant ces problèmes, nous pouvons améliorer la facilité de maintenance et d'évolution de l'application.

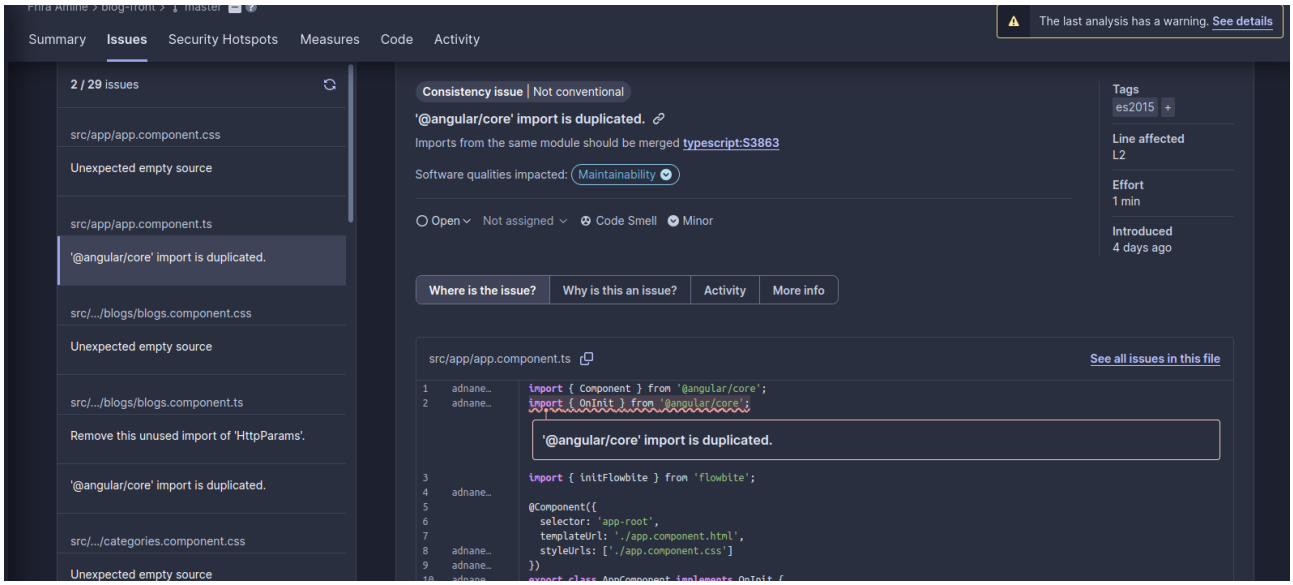

Figure 12: Exemple de probleme de maintenabilite.

- **Exemple de Duplication de Code** Une autre capture d'écran illustre un cas de duplication de code. La duplication est un problème courant qui peut rendre le code plus difficile à gérer et plus sujet aux erreurs. En identifiant et en résolvant ces duplications, nous augmentons l'efficacité et la clarté du code.

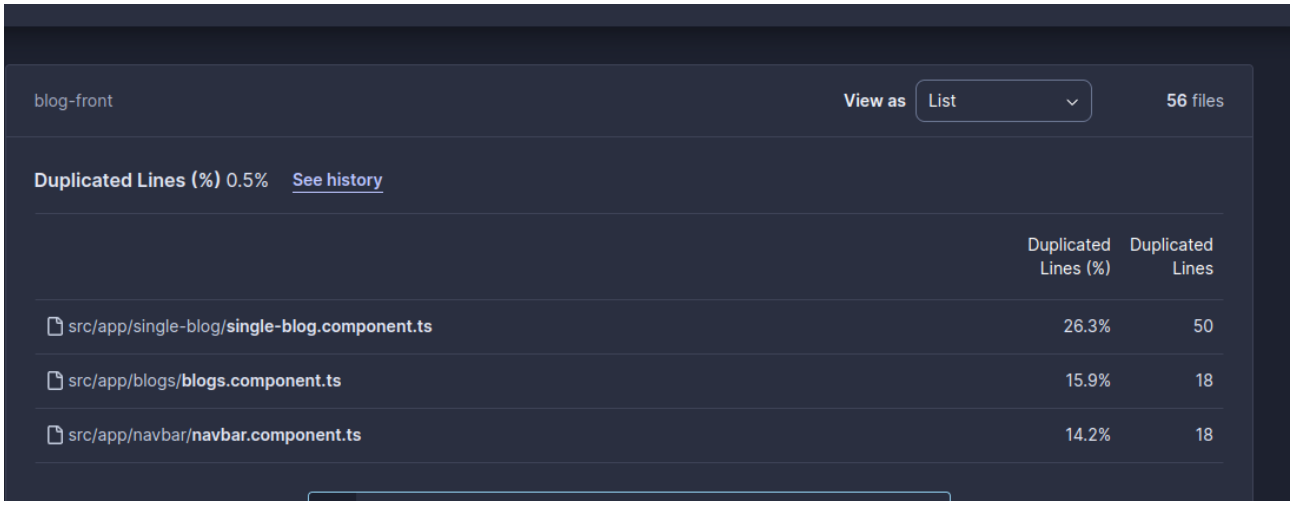

Figure 13: Exemple de probleme de duplication de code.

- **Problème de Sécurité** Enfin, une image montre un exemple de problème de sécurité détecté. Ces problèmes sont cruciaux car ils peuvent rendre l'application vulnérable à des attaques ou à des fuites de données. Identifier et corriger ces vulnérabilités est essentiel pour garantir la sécurité de l'application et la protection des données des utilisateurs.

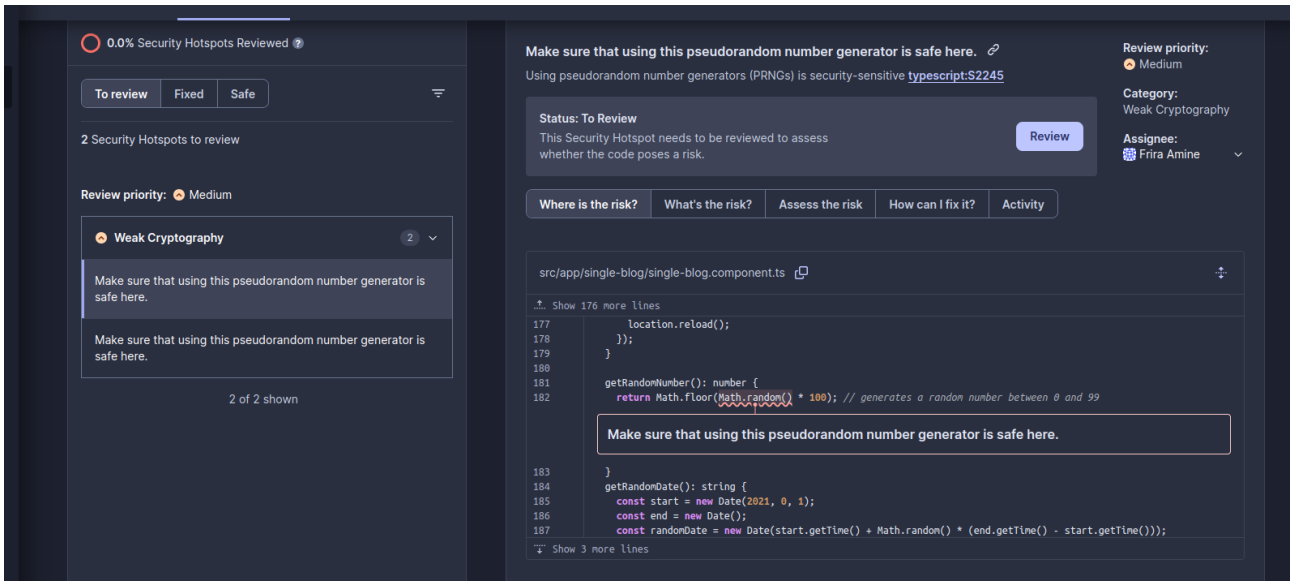

Figure 14: Exemple de probleme de securite.

L'analyse approfondie fournie par SonarQube est indispensable pour maintenir un haut niveau de qualité du code. Elle nous aide à identifier et à corriger proactivement les problèmes potentiels, assurant ainsi une application plus robuste, sécurisée et facile à maintenir.

11.5 Tests de Vulnérabilités de Sécurité avec Burp Suite

11.5.1 Exploitation d'une Faille lors d'authentification

Dans cette partie de notre rapport, nous présentons un test de sécurité réalisé à l'aide de Burp Suite, un outil puissant pour tester la sécurité des applications web. Un vidéo accompagne cette description, illustrant la démarche et la découverte de la faille.

- **Interception de Requêtes d'Authentification** Le premier test a impliqué l'utilisation de Burp Suite comme proxy pour intercepter et analyser les requêtes envoyées pendant le processus d'authentification à l'application. Lors de cette interception, nous avons remarqué que les API exposaient les identifiants (ID) des utilisateurs, un détail qui ne devrait normalement pas être accessible.

- **Exploitation de la Faille** En approfondissant cette découverte, nous avons testé la sécurité en envoyant une requête avec l'identifiant d'un utilisateur authentifié (dans ce cas, moi-même) mais en modifiant l'ID de l'utilisateur dans la requête. À notre surprise, cette action a permis d'accéder aux informations d'autres utilisateurs, révélant une faille significative dans le contrôle d'accès.

- **Origine du Problème et Solution** Après analyse, il s'est avéré que la cause de cette vulnérabilité résidait dans l'implémentation des règles de contrôle d'accès. Initialement, ces règles étaient mises en place uniquement côté front-end de l'application, alors qu'elles auraient dû être implémentées au niveau du service de gateway. Cette erreur a permis aux utilisateurs authentifiés d'accéder aux données d'autres utilisateurs, un problème majeur en termes de sécurité et de confidentialité.

Clicker ici pour visualiser le procesus de test securite d'authentification

Cette démonstration souligne l'importance de mettre en œuvre des contrôles de sécurité adéquats et de tester minutieusement les applications pour détecter de telles vulnérabilités. Elle met également en évidence la nécessité d'appliquer des règles de contrôle d'accès non seulement au niveau de l'interface utilisateur

mais aussi au niveau des services backend et des API, assurant une sécurité robuste à tous les niveaux de l'application.

11.5.2 Exploitation d'une Faille lors de la Création d'un Blog

Cette section de notre rapport se concentre sur une deuxième vidéo de démonstration, utilisant Burp Suite pour révéler une faille de sécurité lors de la création d'un blog. Le processus et les résultats sont décrits ci-dessous.

- **Interception d'une Requête POST lors de la Création d'un Blog** Dans ce scénario, nous avons utilisé Burp Suite pour intercepter une requête POST effectuée lors de la soumission d'un nouveau blog. L'objectif était d'examiner les données envoyées dans la requête, notamment les champs qui ne sont pas visibles côté client mais qui apparaissent dans la requête.
- **Modification du Champ Auteur** Au cours de l'analyse, nous avons découvert que le champ de l'auteur, bien que caché dans l'interface utilisateur du client, était visible et modifiable dans la requête. En exploitant cette vulnérabilité, nous avons changé l'ID de l'auteur dans la requête pour celui d'un autre utilisateur.
- **Création de Blog au Nom d'un Autre Utilisateur** Cette manipulation nous a permis de créer un blog apparemment publié par un autre utilisateur, sans avoir besoin de ses détails d'authentification. Cela révèle une faille critique dans la validation des données côté serveur, où des utilisateurs malveillants pourraient potentiellement publier du contenu en se faisant passer pour d'autres utilisateurs.

Clicker ici pour visualiser le procesus de test securite de creation de blog

Cette découverte met en évidence l'importance cruciale de la validation et de la vérification des données côté serveur. Même si certains champs ne sont pas accessibles ou visibles dans l'interface client, ils peuvent toujours être manipulés dans les requêtes HTTP si des mesures de sécurité adéquates ne sont pas mises en place. Il est donc essentiel de s'assurer que toutes les données envoyées au serveur sont rigoureusement validées et que les permissions sont correctement vérifiées pour prévenir ce type de vulnérabilité.

12 Conclusion

12.1 Résumé des Accomplissements

Au terme de ce projet de blog de voyage, nous avons réalisé une application robuste et évolutif, utilisant une architecture microservices moderne et des technologies de pointe. Les principaux accomplissements incluent :

- **Mise en Place d'une Architecture Microservices Efficace:**Notre choix d'architecture a permis une scalabilité, une maintenance et une mise à jour indépendantes pour chaque service, améliorant ainsi la performance globale de l'application.
- **Conteneurisation avec Docker** L'adoption de Docker a assuré une isolation et une consistance des environnements de développement et de production, facilitant le déploiement et la gestion des services.
- **Intégration et Déploiement Continus avec Jenkins** L'implémentation de CI/CD avec Jenkins a automatisé le processus de développement, réduisant les erreurs humaines et accélérant le déploiement des nouvelles fonctionnalités.

- **Déploiement Automatique sur Google Cloud VM** Cette approche a offert une plateforme stable et flexible pour le déploiement et la gestion de l'application, assurant une haute disponibilité et performance.
- **Intégration de SonarQube pour la Qualité du Code** L'utilisation de SonarQube a permis de maintenir des normes élevées de qualité de code, contribuant à la fiabilité et à la maintenabilité de l'application.

12.2 Perspectives Futures

En regardant vers l'avenir, plusieurs améliorations et expansions sont envisagées pour le projet :

- **Adoption de Technologies d'Intelligence Artificielle et d'Analyse de Données**Pour enrichir l'expérience utilisateur, nous envisageons d'intégrer des fonctionnalités basées sur l'IA, comme la recommandation personnalisée de contenus et l'analyse des tendances de voyage.
- **Expansion des Fonctionnalités Sociales** Pour renforcer l'engagement communautaire, le développement de nouvelles fonctionnalités sociales telles que les groupes de discussion et les événements en direct est prévu.
- **Amélioration de la Sécurité et de la Conformité RGPD** Nous prévoyons de renforcer les mesures de sécurité et d'assurer une conformité totale avec les réglementations sur la protection des données, telles que le RGPD.
- **Scalabilité Cloud et Optimisation des Coûts** L'évaluation continue de l'architecture cloud et l'optimisation des coûts resteront une priorité, garantissant que l'application reste à la fois performante et économiquement viable.

En résumé, ce projet a non seulement atteint ses objectifs initiaux mais a également posé les bases pour une croissance et une amélioration continues. L'application de blog de voyage est bien positionnée pour évoluer en réponse aux besoins changeants de ses utilisateurs et aux avancées technologiques dans l'industrie.