Tour guide

Documentation fonctionnelle et technique

Sommaire

1. Présentation du projet

Tour Guide est une application Spring Boot dédiée à aider les utilisateurs à découvrir des attractions touristiques à proximité, tout en leur offrant des réductions sur les séjours à l'hôtel et les billets pour divers spectacles.

Le public cible comprend les voyageurs et les touristes à la recherche de recommandations personnalisées pour leurs loisirs.

Les avantages commerciaux de la solution incluent l'amélioration de l'engagement des utilisateurs grâce à des recommandations adaptées et la génération de revenus additionnels par le biais de partenariats avec des fournisseurs de services touristiques et ou de la publicité

1.1 Objectifs du projet

Les objectifs du projet sont de résoudre les problèmes de performance de l'application Tour Guide, qui est devenue trop lente en raison de l'explosion du nombre d'utilisateurs. Le projet vise également à améliorer la qualité des recommandations d'attractions touristiques et à corriger les bugs existants, afin de garantir une expérience utilisateur fluide et efficace, même en cas de forte demande. Il est demandé la mise en place d'une intégration continue avec une livraison continue.

1.2 Hors du champ d'application

Le projet n'inclut pas la refonte complète de l'interface utilisateur de Tour Guide, ni le développement de nouvelles fonctionnalités au-delà des améliorations de performance et des corrections de bugs. L'intégration de nouvelles fonctionnalités telles que des services supplémentaires ou des outils de gestion avancés est également hors du champ d'application actuel.

Un refactoring complet n'est pas attendu dans un premier temps mais pourra l'être facilement mis en place par la suite grâce au CI / CD.

1.3 Mesures du projet

Le succès du projet sera mesuré par plusieurs indicateurs clés :

- Temps de réponse amélioré : Le temps de réponse des services gpsUtil et RewardsCentral sera évalué en utilisant le fichier Excel fourni pour comparer les temps de réponse actuels et futurs.

- Stabilité et performance : Les tests unitaires et les simulations de charge seront effectués pour garantir que les améliorations apportées résolvent les problèmes de lenteur et de bugs signalés.

- Satisfaction du client : Les retours d'expérience du client seront collectés pour s'assurer que les nouvelles recommandations sont pertinentes et que les bugs ont été corrigés.

2. Spécifications fonctionnelles

Recommandations d'Attractions

- Calcul des Recommandations : Fournit des recommandations d'attractions touristiques basées sur l'emplacement actuel de l'utilisateur.
  Affichage des Recommandations : Présente les attractions recommandées sur l'interface utilisateur.
  Filtrage par Proximité : Affiche les 5 attractions les plus proches par rapport au dernier emplacement de l'utilisateur, sans tenir compte de la distance.

- Service GPS Util

  Collecte des Emplacements : Collecte les données de localisation des utilisateurs en temps réel.
  Amélioration des Performances : Optimisation pour traiter efficacement jusqu'à 100 000 emplacements d'utilisateurs en moins de 15 minutes.

Gestion des Récompenses

- Obtention des Récompenses : Permet de récupérer des informations sur les récompenses disponibles pour chaque attraction.

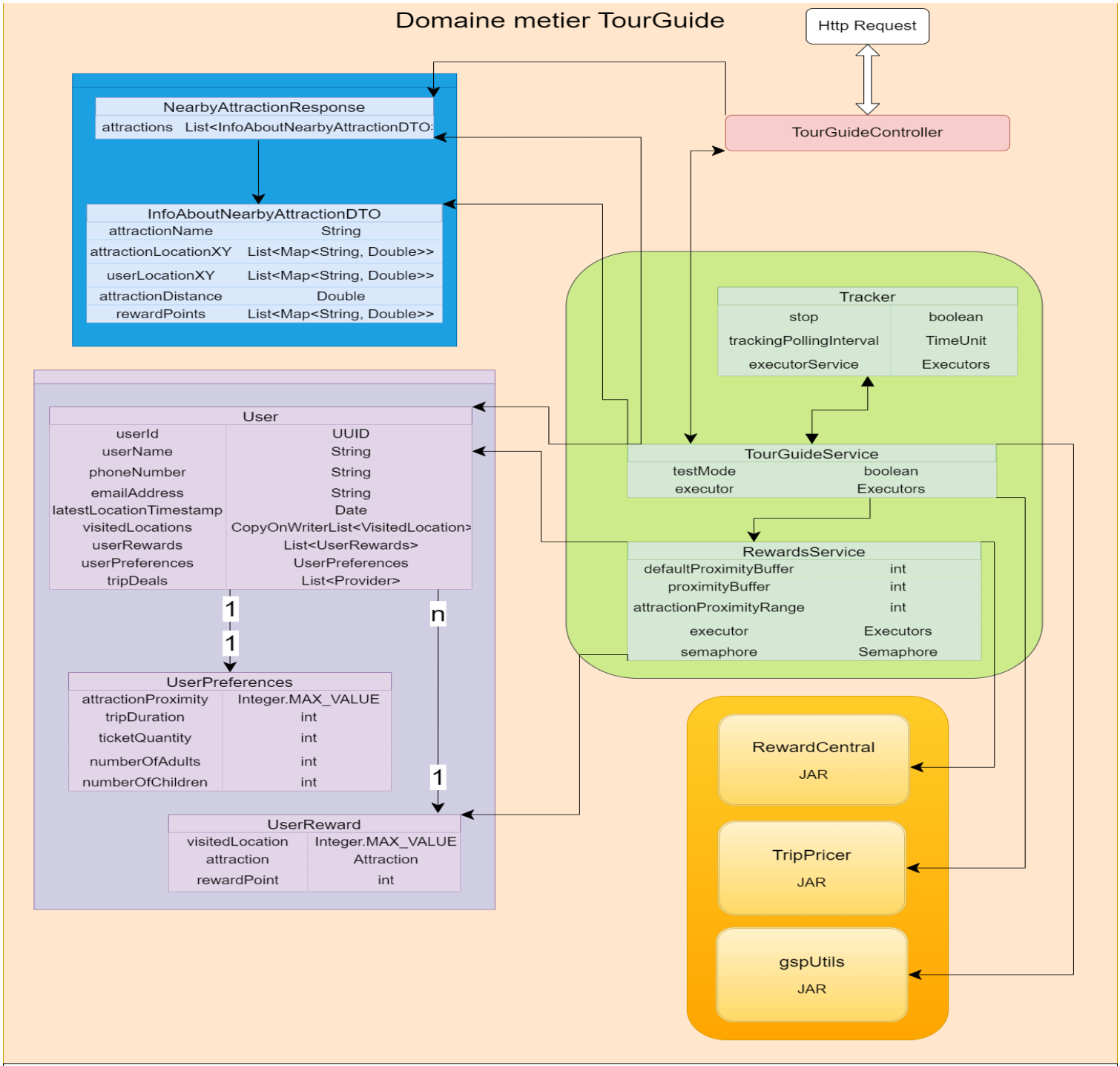
- Amélioration des Performances : Optimisation pour obtenir des récompenses pour 100 000 utilisateurs en moins de 20 minutes.

Pipeline d'Intégration Continue (CI/CD)

- Automatisation du Build : Configure un pipeline CI /CD pour automatiser la compilation du code et la mettre à disposition de l'équipe de développement

3. Spécifications techniques

3.1 Schémas de conception technique



Legende :

data transfer object | Controller | Model USER and its objects

Third-Party Libbrairy | Service

3.2 Glossaire

Document

1. Attractions Touristiques

Définition : Sites ou lieux d'intérêt visités par les touristes, tels que monuments, musées, parcs, et autres points d'intérêt.

Exemple : La Tour Eiffel, le Louvre, Central Park.

2. Récompenses

Définition : Avantages ou incitations offerts aux utilisateurs, souvent sous forme de réductions ou de points, en échange de leur utilisation de services ou d'achat de billets pour des attractions.

Exemple : Réduction de 20% sur un billet pour un spectacle ou accumulation de points échangeables contre des séjours gratuits.

3. Service GPS Util

Définition : Service responsable de la collecte et du traitement des données de localisation des utilisateurs en temps réel.

Objectif : Améliorer les recommandations en fonction de l'emplacement actuel de l'utilisateur.

4. Pipeline CI/CD (Intégration Continue / Livraison Continue)

Définition : Ensemble de pratiques de développement logiciel visant à automatiser les processus de build, test, et déploiement du code.

Objectif : Faciliter les mises à jour fréquentes et fiables du code tout en maintenant une qualité élevée.

5. Filtrage par Proximité

Définition : Technique pour afficher les attractions les plus proches de l'utilisateur en fonction de leur position géographique actuelle.

Exemple : Affichage des 5 attractions touristiques les plus proches d'un utilisateur.

6. Temps de Réponse

Définition : Durée nécessaire pour qu'une application ou un service réponde à une demande d'utilisateur.

Objectif : Mesurer l'efficacité et la rapidité des services comme gpsUtil et RewardsCentral.

7. Performance

Définition : Capacité d'un système ou d'une application à fonctionner efficacement, en traitant les demandes dans un délai raisonnable, même sous une forte charge d'utilisation.

Objectif : Assurer une utilisation fluide même avec un grand nombre d'utilisateurs simultanés.

8. Tests Unitaires

Définition : Méthode de test logiciel où des unités individuelles de code sont testées indépendamment pour s'assurer qu'elles fonctionnent correctement.

Objectif : Identifier les bogues au niveau des composants individuels du code.

9. Simulations de Charge

Définition : Tests visant à évaluer la capacité d'un système à fonctionner sous des conditions de forte demande ou de charge.

Objectif : Vérifier la robustesse et la stabilité du système lorsqu'il est soumis à un grand nombre d'utilisateurs ou de requêtes simultanées.

10. Recommandations Personnalisées

Définition : Suggestions d'attractions ou de services basées sur les préférences et l'historique de l'utilisateur, ainsi que sur leur emplacement actuel.

Objectif : Offrir une expérience utilisateur sur mesure pour améliorer l'engagement et la satisfaction.

11. Réduction

Définition : Diminution du prix d'un produit ou service, souvent offerte comme incitation ou récompense.

Exemple : 10% de réduction sur une réservation d'hôtel en utilisant l'application.

12. Données de Localisation

Définition : Informations géographiques sur la position actuelle d'un utilisateur, généralement fournies par des dispositifs GPS ou des services de géolocalisation.

Objectif : Utiliser ces données pour personnaliser les recommandations et services.

13. Amélioration des Performances

Définition : Optimisation des systèmes et services pour qu'ils fonctionnent plus efficacement, en réduisant les temps de réponse et en augmentant la capacité de traitement.

Objectif : Assurer une expérience utilisateur fluide et rapide même avec une grande quantité de données ou d'utilisateurs.

14. Interface Utilisateur (UI)

Définition : Partie de l'application avec laquelle les utilisateurs interagissent, incluant les éléments visuels et les contrôles.

Objectif : Offrir une expérience utilisateur intuitive et agréable.

15. Bug

Définition : Erreur ou défaut dans le code d'une application qui cause un comportement incorrect ou inattendu.

Objectif : Identifier et corriger les bugs pour améliorer la stabilité et la fonctionnalité de l'application.

Classe

User

Définition : représente un utilisateur du système Tour Guide. Elle stocke des informations personnelles de l'utilisateur, ses préférences, les lieux qu'il a visités, ses récompenses, et les offres de voyage disponibles pour lui.

Attributs

- **`userId`** (UUID) : Identifiant unique de l'utilisateur.
- **`userName`** (String) : Nom de l'utilisateur.
- **`phoneNumber`** (String) : Numéro de téléphone de l'utilisateur.
- **`emailAddress`** (String) : Adresse e-mail de l'utilisateur.
- **`latestLocationTimestamp`** (Date) : Date et heure de la dernière mise à jour de la localisation de l'utilisateur.
- **`visitedLocations`** (CopyOnWriteArrayList<VisitedLocation>) : Liste des lieux visités par l'utilisateur, stockée de manière thread-safe.
- **`userRewards`** (List<UserReward>) : Liste des récompenses que l'utilisateur a accumulées.
- **`userPreferences`** (UserPreferences) : Préférences de l'utilisateur pour les recommandations.
- **`tripDeals`** (List<Provider>) : Liste des offres de voyage disponibles pour l'utilisateur.

UserPreferences

Définition : La *classe* `UserPreferences` représente les préférences d'un utilisateur en matière de recommandations pour les attractions touristiques et les voyages. Elle permet de personnaliser les suggestions en fonction des besoins et des attentes de l'utilisateur.

Attributs :

- **`attractionProximity`** (int)

  Distance maximale que l'utilisateur est prêt à parcourir pour visiter une attraction. *Valeur par défaut :* `Integer.MAX_VALUE` (distance illimitée).
- **`tripDuration`** (int)

  Durée souhaitée du voyage en jours. *Valeur par défaut :* `1` jour.
- **`ticketQuantity`** (int)

  Nombre de billets nécessaires pour les attractions ou les Événements. *Valeur par défaut :* `1` billet.
- **`numberOfAdults`** (int)

  Nombre d'adultes pour lesquels les recommandations doivent être adaptées. *Valeur par défaut :* `1` adulte.
- **`numberOfChildren`** (int)

  Nombre d'enfants pour lesquels les recommandations doivent être adaptées. *Valeur par défaut:* `0` enfant.

UserReward

Définition : La classe `UserReward` représente une récompense qu'un utilisateur peut obtenir en fonction de ses visites d'attractions touristiques. Elle associe un lieu visité à une attraction spécifique et attribue des points de récompense en conséquence.

Attributs :

- **`visitedLocation (VisitedLocation)`**
  Lieu visité par l'utilisateur pour lequel la récompense est attribuée.
- **`attraction (Attraction)`**
  Attraction associée à la récompense.
- **`rewardPoints (int)`**
  Nombre de points de récompense attribués pour la visite de l'attraction.
  Valeur par défaut : Non spécifiée ; doit être définie lors de l'instanciation.

NearbyAttractionResponse

Définition : La classe `NearbyAttractionResponse` est utilisée pour encapsuler les informations sur les attractions touristiques à proximité, qui sont renvoyées en réponse à une requête pour obtenir des recommandations d'attractions.

Attributs :

- **`attractions`** (List<InfoAboutNearbyAttractionDTO>)
  Liste des objets contenant des informations sur les attractions touristiques situées à proximité de l'utilisateur, chaque objet `InfoAboutNearbyAttractionDTO` dans la liste représente une attraction spécifique et contient des détails pertinents pour l'utilisateur.

InfoAboutNearbyAttractionDTO

Définition : La classe `InfoAboutNearbyAttractionDTO` est un objet de transfert de données (DTO) utilisé pour transmettre des informations détaillées sur une attraction touristique à proximité, incluant sa localisation, la distance par rapport à l'utilisateur, et les points de récompense associés.

Attributs :

- **`attractionName`** (String)

  Nom de l'attraction touristique.

- **`attractionLocationXY`** (List<Map<String, Double>>)

  Liste de coordonnées géographiques (en X et Y) représentant la localisation de l'attraction. Chaque élément de la liste est un `Map` contenant les clés `"x"` et `"y"` pour les coordonnées.

- **`userLocationXY`** (List<Map<String, Double>>)

  Liste de coordonnées géographiques (en X et Y) représentant la localisation de l'utilisateur. Chaque élément de la liste est un `Map` contenant les clés `"x"` et `"y"` pour les coordonnées.

- **`attractionDistance`** (double)

  Distance entre l'utilisateur et l'attraction touristique, mesurée en unités de distance (par exemple, mètres ou kilomètres).

- **`rewardPoints`** (int)

  Nombre de points de récompense associés à l'attraction touristique, pouvant être gagnés ou échangés par l'utilisateur.

RewardsService

Définition : La classe `RewardsService` est responsable du calcul des récompenses pour les utilisateurs en fonction de leurs visites et des attractions disponibles. Elle utilise des services externes pour obtenir les informations sur les attractions et les récompenses, et elle gère la concurrence pour optimiser les performances.

Attributs :

- **`defaultProximityBuffer`** (int)

  Tampon de proximité par défaut en miles utilisé pour déterminer si une attraction est proche d'un lieu visité. *Valeur par défaut:* `10` miles.

- **`proximityBuffer`** (int)

  Tampon de proximité actuel en miles utilisé pour évaluer la proximité des attractions par rapport aux lieux visités.

  Valeur par défaut: `defaultProximityBuffer`.

- **`attractionProximityRange`** (int)

  Plage de proximité en miles pour les attractions que l'utilisateur est susceptible de visiter.

  Valeur par défaut: `200` miles.

- **`executor`** (Executor)

  Exécuteur de tâches asynchrones qui utilise un pool de threads pour gérer les tâches I/O-bound.

  Valeur par défaut: `Executors.newCachedThreadPool()`.

- **`semaphore`** (Semaphore)

Sémaphore utilisé pour limiter le nombre de tâches concurrentes pouvant s'exécuter simultanément. *Valeur par défaut:* `new Semaphore(100)`.

TourGuideService

Définition : La classe `TourGuideService` gère les opérations principales du système Tour Guide, notamment la récupération des récompenses des utilisateurs, le suivi de leur localisation, et la fourniture de recommandations pour les attractions touristiques et les offres de voyage. Elle intègre des services externes pour obtenir des données géographiques, des récompenses et des offres de voyage.

Attributs :

- **`testMode`** (boolean)

  Indicateur pour activer ou désactiver le mode test, qui initialise des utilisateurs internes pour les tests.

- **`executor`** (Executor)

  Exécuteur de tâches pour gérer l'exécution des tâches asynchrones, configurés avec un pool de threads fixe.

Classe Tracker

Définition : La classe `Tracker` est responsable de la surveillance régulière des emplacements des utilisateurs du système Tour Guide. Elle s'exécute dans un thread dédié pour interroger et mettre à jour les localisations des utilisateurs à intervalles réguliers.

Attributs :

- **`trackingPollingInterval`** (long)

  Intervalle de temps en secondes entre chaque mise à jour de suivi des u tilisateurs. *Valeur par défaut :* 300 secondes (5 minutes).

- **`executorService`** (ExecutorService)
  Service d'exécution responsable de la gestion du thread dédié au suivi des utilisateurs. *Valeur par défaut :* Instance de Executors.newSingleThreadExecutor().

- **`tourGuideService`** (TourGuideService)

  Service utilisé pour interagir avec les données des utilisateurs et mettre à jour leurs localisations.

- **`stop`** (boolean)

  Indicateur pour arrêter l'exécution du thread de suivi lorsque nécessaire. *Valeur par défaut :* `false`.

TourGuideController

Définition : La classe `TourGuideController` est un contrôleur Spring Boot responsable de la gestion des requêtes HTTP pour l'application Tour Guide. Elle interagit avec le service TourGuideService pour fournir des informations sur la localisation des utilisateurs, les attractions à proximité, les récompenses, et les offres de voyage.

3.3 Solutions techniques

Spring Boot 3.3.2

Justification : Utilisation de Spring Boot pour simplifier le développement d'applications Java en fournissant des configurations prêtes à l'emploi et un support intégré pour de nombreuses fonctionnalités, telles que les contrôleurs REST, la gestion des dépendances, et l'intégration avec diverses bases de données et services.

Java 21

Justification: Choix de la version Java la plus récente pour bénéficier des dernières fonctionnalités du langage, des améliorations de performance et des corrections de sécurité. Java 21 assure également la compatibilité avec Spring Boot 3.3.2.

Asynchrone et Parallélisme

Mise en place de l'asynchrone et du parallélisme

Justification : L'utilisation de traitements asynchrones et parallèles permet de gérer les tâches gourmandes en ressources de manière plus efficace, en améliorant les temps de réponse et en optimisant les performances de l'application, surtout lors de la gestion de grandes quantités de données ou de requêtes simultanées.

Refactoring avec Streams

Utilisation des Streams en Java

Justification : Le refactoring avec Streams permet de simplifier le traitement des collections de données en remplaçant les boucles `for` traditionnelles et les opérations de filtrage manuelles par des opérations déclaratives plus lisibles. Les Streams en Java facilitent le traitement parallèle des données, ce qui améliore les performances pour des opérations sur de grandes collections tout en rendant le code plus lisible et maintenable.

Bénéfices :

- **Lisibilité et Maintenabilité :** Le code devient plus clair et plus concis, rendant les opérations sur les collections plus faciles à comprendre et à maintenir.
- **Performance :** Les Streams offrent des fonctionnalités de traitement parallèle qui peuvent accélérer le traitement des données volumineuses.
- **Modularité :** La programmation fonctionnelle rend le code plus modulaire et réutilisable.

Gestion de Version et CI/CD

GitHub pour le Versionnage du Code Source

Justification: Utilisation de GitHub pour le versionnage du code source permet de suivre l'historique des modifications, de gérer les branches et les merges, et de collaborer efficacement avec l'équipe de développement. GitHub offre des outils puissants pour la gestion des pull requests et la révision du code, facilitant ainsi la collaboration et la qualité du développement.

GitHub Actions pour CI/CD

Justification: GitHub Actions a été choisi comme solution CI/CD en raison de son intégration native avec GitHub, simplifiant ainsi le flux de travail de développement et de déploiement. Voici les raisons principales du choix de GitHub Actions :

- **Intégration Transparente avec GitHub :** GitHub Actions est directement intégré avec GitHub, ce qui élimine le besoin de configurations complexes pour la communication entre le dépôt de code et le système CI/CD. Cette intégration simplifie la mise en place des pipelines CI/CD et réduit les risques d'erreurs liées à la configuration.

- **Gestion des Conteneurs Docker avec GitHub Container Registry (GHCR) :** GitHub Actions prend en charge la gestion des images Docker grâce à GitHub Container Registry (GHCR). Cela permet de stocker, versionner et distribuer les images Docker directement dans la plateforme GitHub, centralisant ainsi la gestion des conteneurs et simplifiant les workflows CI/CD. Les images Docker peuvent être construites, poussées vers GHCR et extraites lors des déploiements automatiquement, réduisant ainsi la complexité et garantissant la cohérence entre les environnements.

- **Facilité d'Utilisation :** GitHub Actions propose une interface conviviale et des workflows définis en YAML, ce qui rend la configuration des pipelines CI/CD intuitive et accessible. Cette simplicité permet de mettre en place et de gérer les processus d'intégration et de déploiement de manière efficace sans nécessiter une expertise approfondie en CI/CD.

- **Flexibilité et Extensibilité :** GitHub Actions offre une large gamme d'actions prédéfinies et la possibilité de créer des actions personnalisées, ce qui permet d'adapter les workflows CI/CD aux besoins spécifiques du projet. Cela inclut des intégrations avec des outils de tests, de couverture de code, et de déploiement, ce qui assure une couverture complète du processus de développement.

- **Écosystème GitHub :** L'utilisation de GitHub Actions s'inscrit dans l'écosystème GitHub, facilitant la gestion des versions, des branches, et des demandes de tirage (pull requests) tout en automatisant les tâches de build, test, et déploiement. Cette intégration renforce la cohérence des processus de développement et simplifie la gestion du code source.

Docker pour la Conteneurisation

Justification: Docker est utilisé pour conteneuriser l'application, ce qui facilite le déploiement, l'exécution, et la gestion des environnements de développement et de production. Les images Docker garantissent que l'application fonctionne de manière cohérente quel que soit l'environnement.

Structure du Pipeline CI/CD :

- **Étape 1 : Build**

  Description: Compilation du code source et construction des artefacts (par exemple, des images Docker). Cette étape assure que le code est correctement construit avant d'être testé ou déployé.

- **Étape 2 : Tests**

  Description: Exécution des tests unitaires et d'intégration pour vérifier que le code fonctionne comme prévu. Les outils comme Surefire et JaCoCo sont utilisés pour les tests et la couverture de code. Cette étape aide à détecter les problèmes avant le déploiement.

- **Étape 3 : Analyse de Qualité du Code**

  Description: Intégration d'outils comme SonarCloud pour effectuer une analyse statique du code, détecter les vulnérabilités et les défauts de qualité. Cette étape est cruciale pour maintenir un code propre et sécurisé.

- **Étape 4 : Déploiement**

  Description: Déploiement automatique des artefacts construits dans des environnements de test ou de production à l'aide de Docker. Cette étape assure que l'application est mise à jour de manière cohérente et rapide.

- **Étape 5 : Monitoring et Rollback**

  Description: Surveillance des déploiements pour détecter tout problème en production et mise en place de mécanismes de rollback pour revenir à une version antérieure en cas de problème. Cette étape garantit la stabilité et la résilience de l'application.

Qualité du Code et Sécurité

SonarCloud avec SAST (Static Application Security Testing)

Justification: SonarCloud est intégré pour analyser la qualité du code et la sécurité. Il fournit des rapports sur les vulnérabilités de sécurité, les défauts de qualité du code, et les mauvaises pratiques. L'analyse statique aide à identifier et corriger les problèmes avant qu'ils n'affectent l'application en production.

JaCoCo

Justification: JaCoCo est utilisé pour la couverture de code des tests unitaires. Il permet de mesurer le pourcentage de code testé par les tests automatisés, garantissant ainsi que le code est correctement testé et réduisant le risque de bugs non détectés.

Surefire

Justification: Surefire est un plugin Maven utilisé pour exécuter les tests unitaires et d'intégration. Il assure que les tests sont correctement exécutés et fournit des rapports détaillés sur les résultats des tests, ce qui aide à maintenir une qualité élevée du code.

3.4 Autres solutions non retenues

1. Outils d'Intégration Continue (CI/CD)

Jenkins

- **Description :** Jenkins est un serveur d'intégration continue open-source qui offre une grande flexibilité et des possibilités étendues pour automatiser les tâches de build, test, et déploiement.
- **Raisons de Non-Retour :** Bien que Jenkins soit un choix populaire pour l'intégration continue, le choix de GitHub Actions a été privilégié en raison de sa simplicité d'intégration avec GitHub, notre plateforme principale pour le code source. Étant donné que nous n'avons pas eu de besoin spécifique pour les fonctionnalités avancées offertes par Jenkins, et que GitHub Actions répondait parfaitement à nos besoins en matière d'intégration continue avec moins de complexité, Jenkins n'a pas été considéré plus en détail.

GitLab CI/CD

- **Description :** GitLab CI/CD propose des fonctionnalités d'intégration continue et de livraison continue directement intégrées à la plateforme GitLab.
- **Raisons de Non-Retour :** GitLab CI/CD aurait pu être une option viable, mais notre choix s'est porté sur GitHub Actions principalement en raison de l'intégration native avec GitHub, ce qui facilite la gestion des versions et des workflows CI/CD au sein de la même plateforme. La nécessité de gérer une double plateforme pour les versions et CI/CD a été un facteur déterminant pour privilégier GitHub Actions.

2.Registres de Conteneurs

Docker Hub

- **Description :** Docker Hub est le registre de conteneurs public le plus utilisé pour stocker et partager des images Docker.
- **Raisons de Non-Retour :** Nous avons choisi GitHub Container Registry (GHCR) plutôt que Docker Hub principalement pour centraliser la gestion des images Docker avec notre dépôt de code source sur GitHub. Bien que Docker Hub soit largement utilisé, GHCR a simplifié notre flux de travail en permettant de gérer les conteneurs directement à partir de la même plateforme.

Amazon Elastic Container Registry (ECR)

- **Description :** ECR est un service de registre de conteneurs proposé par AWS pour stocker et gérer des images Docker.
- **Raisons de Non-Retour :** ECR est une solution robuste, mais l'intégration directe avec GitHub et la gestion centralisée offerte par GHCR ont été des facteurs déterminants. La gestion des images Docker avec GHCR est alignée avec notre usage principal de GitHub, simplifiant ainsi l'ensemble du processus CI/CD.

3. Outils de Gestion de Qualité du Code

SonarQube

- **Description :** SonarQube est une plateforme de gestion de la qualité du code qui fournit des analyses détaillées des défauts et des vulnérabilités de sécurité.
- **Raisons de Non-Retour :** Bien que SonarQube soit puissant, SonarCloud a été choisi pour sa meilleure intégration avec GitHub. SonarCloud offre des fonctionnalités similaires tout en s'intégrant directement dans notre environnement GitHub, simplifiant ainsi l'analyse continue du code.

Coveralls

- **Description :** Coveralls est un service dédié à la couverture de code pour les projets open-source.
- **Raisons de Non-Retour :** Coveralls a été envisagé, mais JaCoCo a été préféré en raison de sa compatibilité directe avec Maven et sa capacité à générer des rapports détaillés intégrés au pipeline CI/CD. Cette intégration directe avec les outils de build existants a facilité la gestion de la couverture de code.