Documentation du Projet : Application de Réseau Social en Micro services

Introduction

Les réseaux sociaux ont transformé la manière dont les individus interagissent en ligne. Ce projet vise à développer une application de réseau social moderne en utilisant une architecture en micro services. L'approche en micro services permet une meilleure flexibilité, évolutivité, maintenabilité, sécurité et performances de l'application.

Objectifs du Projet

Le projet consiste à créer une application de réseau social moderne, en mettant en œuvre une architecture en micro services. Les objectifs principaux incluent la flexibilité pour s'adapter facilement aux nouveaux besoins, l'évolutivité pour gérer la croissance des utilisateurs et de la charge, la maintenabilité pour faciliter les mises à jour, la sécurité pour protéger les données des utilisateurs, et des performances optimales pour une expérience utilisateur fluide. Cette architecture en micro services implique la décomposition de l'application en plusieurs services indépendants et communicants, chacun étant responsable d'une fonctionnalité spécifique, comme l'authentification ou la gestion des contenus.

Fonctionnalités

Langage de Programmation

- **Java** : Java sera utilisé comme langage principal pour le développement de l'application en raison de sa robustesse et de sa compatibilité multiplateforme.

Frameworks de Développement

- **Spring Boot** : Spring Boot sera le framework principal utilisé pour développer les micro services en raison de sa facilité de configuration et de son support pour le développement rapide d'applications.

Bases de Données

- **PostgreSQL** : PostgreSQL sera utilisé comme base de données relationnelle principale pour stocker les données structurées de l'application.
- **Redis** : Redis sera utilisé comme base de données clé-valeur pour le stockage de données en cache et la gestion des sessions utilisateur.

Testing

- **JUnit** : JUnit sera utilisé pour les tests unitaires afin d'assurer la qualité du code.
- **Mockito** : Mockito sera utilisé pour créer des objets simulés lors des tests unitaires afin d'isoler les composants à tester.
- **Postman** : Postman sera utilisé pour tester les API et les services de manière automatisée et manuelle.

Versionning

- **Git** : Git sera utilisé comme système de contrôle de version pour gérer le code source de l'application.

- **GitHub** : GitHub sera utilisé comme plateforme de gestion de code source pour héberger le code et faciliter la collaboration entre les développeurs.

Outils de Communication

- **http** : HTTP sera le protocole principal utilisé pour la communication entre les micro services et les clients.
- **gRPC** : gRPC sera utilisé pour la communication RPC (Remote Procedure Call) entre les micro services, offrant une communication plus efficace et structurée.

Technologies Spring Cloud - Spring Cloud Netflix

- **Eureka (Service Discovery)** : Eureka sera utilisé pour le service de découverte afin de localiser dynamiquement les instances de services.
- **Spring Cloud Load Balancer** : Spring Cloud Load Balancer sera utilisé pour équilibrer la charge entre les instances de micro services.
- **Resilience4j (Circuit Breaker)** : Resilience4j sera utilisé pour gérer les erreurs et les défaillances des services en mettant en œuvre le modèle de circuit breaker.
- **Spring Cloud (API Gateway)** : Spring Cloud API Gateway sera utilisé comme point d'entrée principal de l'application, offrant un moyen centralisé pour gérer les requêtes et les routes.
- **Spring Cloud Config** : Spring Cloud Config sera utilisé pour la gestion centralisée de la configuration des micro services.
- **Spring Cloud Sleuth** : Spring Cloud Sleuth sera utilisé pour le traçage distribué des requêtes afin de faciliter le débogage et la surveillance des services.

L'implémentation d'un micro service

Lors de l'implémentation d'un micro service en utilisant Spring Boot, un framework Java populaire pour le développement d'applications Java, voici quelques détails supplémentaires sur cette étape :

Initialisation du projet : Utilisez Spring Initializr pour générer un projet Spring Boot avec les dépendances nécessaires. Vous pouvez choisir les dépendances telles que Spring Web (pour créer des API REST), Spring Data JPA (pour l'accès aux données.

Structuration du projet : Organisez votre projet de manière à ce qu'il soit modulaire et facile à naviguer. Divisez-le en packages logiques selon les fonctionnalités (par exemple, controllers, services, repositories) pour suivre les conventions de conception de Spring.

Développement des contrôleurs REST : Écrivez des contrôleurs REST pour définir les points d'accès de votre API. Utilisez les annotations de Spring MVC telles que `@RestController` et `@RequestMapping` pour créer des endpoints RESTful.

Accès aux données avec Spring Data JPA : Utilisez Spring Data JPA pour simplifier l'accès aux données dans votre microservice. Créez des interfaces de repository en étendant `JpaRepository` et utilisez les annotations de mapping JPA pour mapper les entités aux tables de la base de données.

Tests unitaires avec JUnit et Mockito : Écrivez des tests unitaires pour vos composants Spring Boot en utilisant JUnit et Mockito. Testez les contrôleurs, les services et les repositories pour vous assurer que chaque composant fonctionne comme prévu.

La communication entre les micro services :

Configuration de FeignClient : Déclarez une interface Java annotée avec @FeignClient pour définir le client Feign qui communique avec le service distant. Spécifiez le nom du service distant et les endpoints à appeler dans cette interface.

Injection de dépendance : Déclarez une interface Java annotée avec @FeignClient pour définir le client Feign qui communique avec le service distant. Spécifiez le nom du service distant et les endpoints à appeler dans cette interface.

Injection de dépendance : Injectez l'interface dans vos composants Spring qui ont besoin de communiquer avec le service distant.

Utilisation de Feign pour la communication : Injectez l'interface dans vos composants Spring qui ont besoin de communiquer avec le service distant.

Routage des Requêtes :

Dépendance de Gateway : Ajoutez la dépendance spring-cloud-starter-gateway dans le fichier pom.xml de votre projet Maven pour inclure Spring Cloud Gateway.

Configuration dans le fichier :

Configurez Spring Cloud Gateway pour utiliser la découverte de services (discovery) en activant la recherche des services enregistrés dans le registre de service.

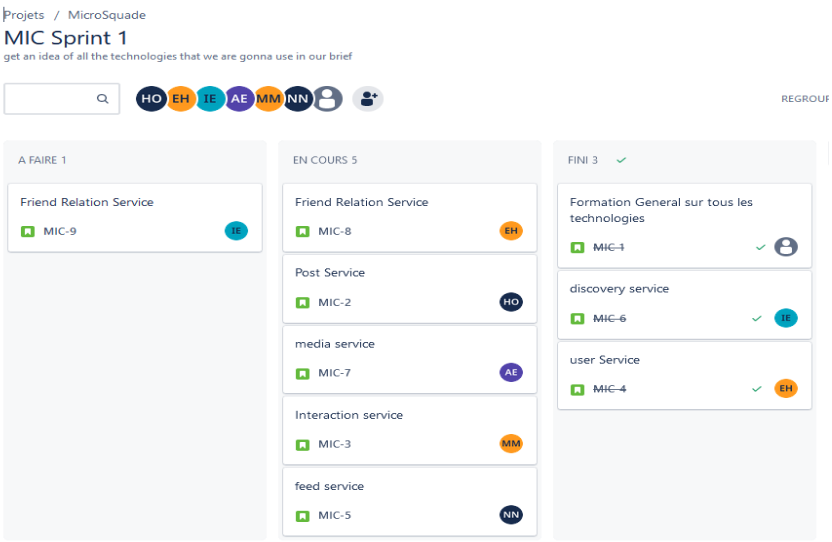
Définissez les routes dans la configuration pour spécifier comment le trafic doit être dirigé vers les micro services, en utilisant des prédicats basés sur les chemins d'URL.

Définition des routes : Pour chaque route, spécifiez un identifiant unique, l'URI du service de destination, et les prédicats qui déterminent si une requête doit être dirigée vers cette route.

Gestion de Projet avec Jira et GitHub

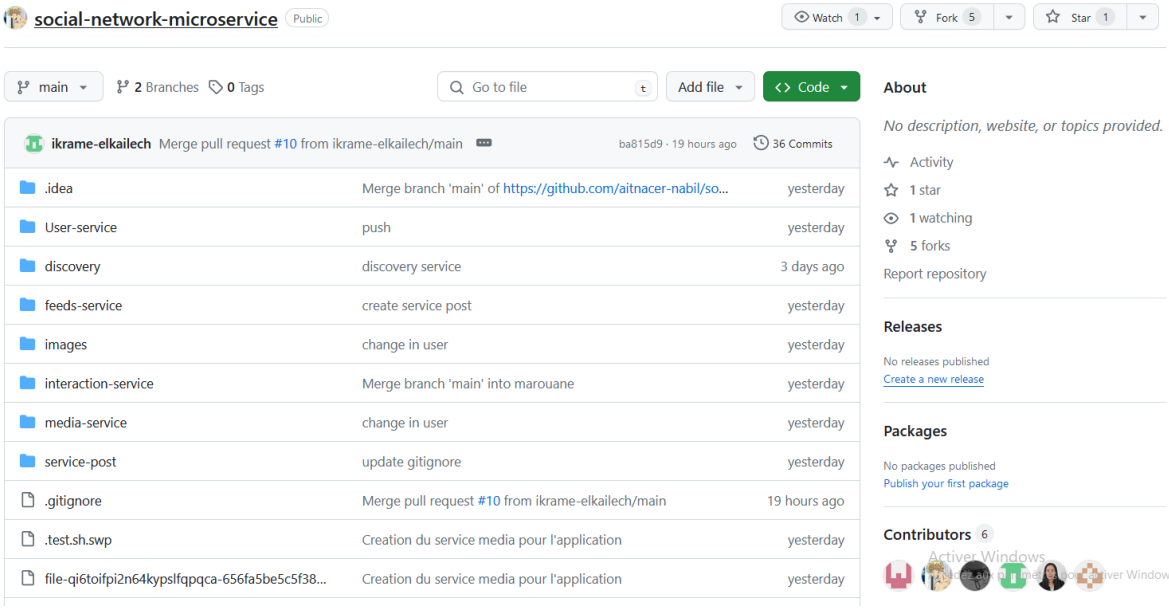
Utilisation de Jira

Nous avons utilisé Jira comme outil principal de gestion de projet pour planifier, suivre et organiser notre travail. Voici comment nous avons utilisé Jira dans notre processus de développement :



Intégration avec GitHub

GitHub a été notre plateforme de développement centralisée où nous avons géré les branches, examiné les pull requests et suivi les problèmes.



Les membres d'équipe :

Adil ERRAI

Halima EL AMRI

Hasna OUBALI

Ikrame ELKAILEC

Marouane MOUSLIH

Nabil AIT NACER