ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Rapport Projet : Architecture des composants d'entreprise

Réaliser par :

BOTAZIT Ihssane

MAKOUDI Nouhaila

OUAAZIZ Mohamed

1. Introduction

1.1 Aperçu du projet

Le projet du blog personnel repose sur une architecture microservices, offrant une plateforme flexible et évolutive pour la publication de contenu. Les principaux services du projet sont le "Service Utilisateur" et le "Service Article".

- Service Utilisateur :

Ce service gère les aspects liés à l'authentification et à la gestion des comptes. Les propriétaires du blog peuvent s'authentifier de manière sécurisée, leur accordant des droits étendus pour créer, modifier et supprimer du contenu. Blog visiteurs peuvent naviguer sur le blog sans nécessité d'authentification. Ce service assure la sécurité de l'application en gérant les comptes des propriétaires.

- Service Article :

Le service Article est responsable de la gestion du contenu du blog. Il inclut la création et la gestion de catégories pour organiser les articles thématiquement. Les propriétaires peuvent créer, modifier et supprimer des articles, tandis que les visiteurs ont la possibilité de lire et de commenter les publications. Ce service garantit une expérience utilisateur interactive en permettant aux visiteurs d'interagir avec le contenu du blog.

La communication entre ces services est optimisée grâce à l'utilisation d'OpenFeign, une bibliothèque Java simplifiant les appels HTTP entre les microservices de manière déclarative. Cette approche facilite la gestion des interfaces distantes et contribue à une architecture plus cohérente.

L'ensemble du projet vise à offrir une plateforme conviviale et extensible pour la gestion de blogs personnels. La modularité des microservices assure une maintenance aisée, tandis que la flexibilité de l'architecture permet une évolution adaptée aux besoins changeants des utilisateurs.

1.2 Importance de l’architecture microservices

L'importance de l'architecture microservices réside dans ses nombreux avantages qui contribuent à améliorer la conception, le déploiement et la maintenance des applications. Voici une synthèse des points clés :

- Isolation et Résilience

Les microservices offrent une isolation efficace, ce qui signifie que chaque composant fonctionne de manière indépendante. En cas de défaillance d'un composant, les développeurs peuvent aisément basculer vers un autre service, assurant ainsi la résilience de l'application. Cette indépendance facilite la construction et le déploiement de services sans perturber l'ensemble de l'application.

- Scalabilité :

L'architecture microservices, basée sur de petits composants, simplifie l'évolutivité. Les équipes de développement peuvent ajuster la capacité vers le haut ou vers le bas en fonction des besoins spécifiques d'un élément. L'isolation garantit que les applications continuent de fonctionner correctement, même lors de changements massifs, ce qui en fait une approche idéale pour les entreprises travaillant avec différentes plates-formes et appareils.

- Productivité :

La facilité de compréhension des microservices par rapport aux applications monolithiques complètes contribue à une meilleure productivité. En envisageant l'expansion de l'équipe de développement, les microservices se révèlent être un choix judicieux, permettant une collaboration efficace.

- Flexibilité :

L'approche microservices offre une flexibilité exceptionnelle, permettant aux développeurs de choisir les outils adaptés à chaque tâche. Chaque serveur peut être construit avec le langage ou le framework nécessaire, sans impact sur la communication entre les microservices.

- Développement de Projet Plus Rapide :

Grâce à l'indépendance des microservices, il n'est pas nécessaire de modifier la base de code pour effectuer des modifications. Les développeurs peuvent modifier, tester et déployer individuellement chaque composant, accélérant ainsi le processus de développement de l'application.

- Évolutif :

L'architecture microservices est particulièrement adaptée aux applications dont l'évolution et les futurs appareils d'exécution sont difficiles à prédire. Les mises à niveau rapides et contrôlées peuvent être fournies sans ralentir ou arrêter les applications existantes.

Bien que les microservices offrent ces avantages, il est important de noter que des inconvénients tels que l'utilisation de langages de codage, de Framework et de bibliothèques différentes par les équipes peuvent entraîner des défis de coordination. Cependant, pour les applications complexes à grande échelle, l'architecture micro-services demeure un choix optimal.

2. Architecture Microservices

2.1 Architecture

Eureka Server :

Eureka Server agit en tant que registre des services dans l'écosystème. Chaque service s'inscrit auprès d'Eureka Server, permettant ainsi aux autres services de le découvrir et de communiquer avec lui.
Gateway :

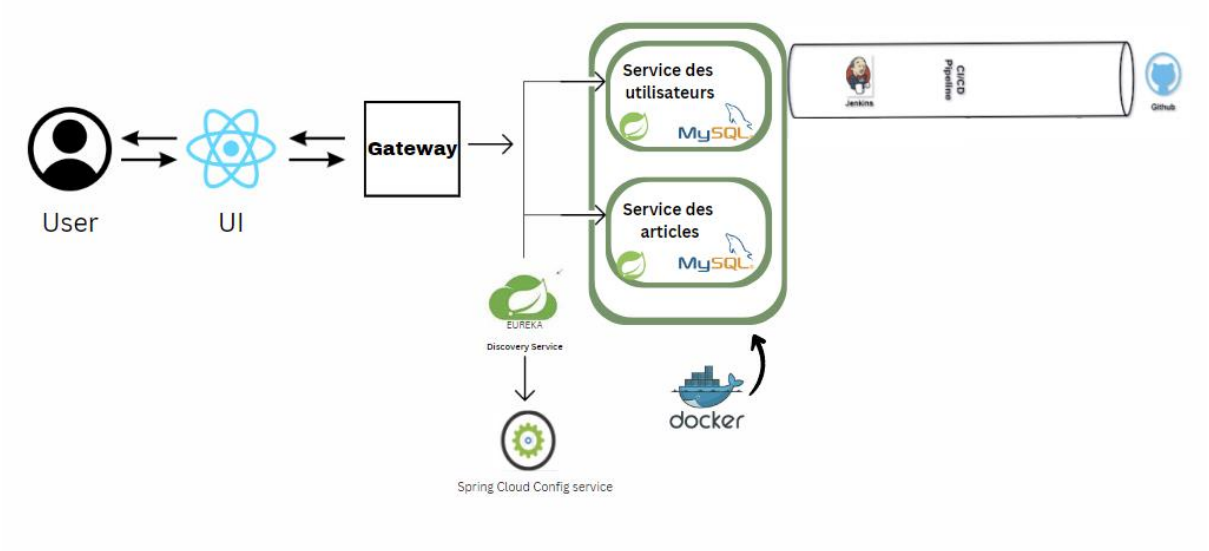
Le Gateway est la passerelle d'entrée principale pour l'application. Il gère le routage des requêtes et peut effectuer des tâches telles que l'authentification, l'autorisation et la limitation du trafic. Le Gateway est souvent utilisé pour offrir une API unifiée aux clients.
Services (2 Services) :

Les services représentent les différentes fonctionnalités de votre application. Dans votre cas, vous avez deux services distincts. Chaque service a ses propres responsabilités et communique avec d'autres services au besoin.
Docker :

Docker est utilisé pour encapsuler chaque service et ses dépendances dans des conteneurs. Les conteneurs offrent une isolation légère, garantissant que chaque service fonctionne de manière cohérente indépendamment de l'environnement.
Jenkins :

Jenkins est un outil d'intégration continue qui automatise le processus de construction, de test et de déploiement des services. Il peut être configuré pour déclencher automatiquement le déploiement des services lorsqu'il détecte des changements dans le référentiel.
GitHub :

GitHub est utilisé comme référentiel de code source pour l'ensemble du projet. Il héberge le code des services, du Gateway, et éventuellement des scripts de configuration pour Jenkins. Les développeurs peuvent collaborer, suivre les versions et gérer les modifications de code via GitHub.



2.2 Description des services

Pour une application web dédiée à un blog personnel, voici une description modifiée des principaux services :

- **Service Utilisateur :**
  Ce service englobe toutes les fonctionnalités liées à la gestion des utilisateurs. Il comprend l'authentification, permettant aux propriétaires du blog de se connecter de manière sécurisée. Les visiteurs peuvent parcourir le blog sans nécessité d'authentification. Les propriétaires ont des droits étendus pour créer, modifier et supprimer du contenu.

- **Service Article :**
  Le Service Article, en plus de gérer la publication de contenu, offre des fonctionnalités avancées pour une expérience utilisateur plus riche. Il propose une organisation thématique grâce à la création

et à la gestion de catégories. Les propriétaires du blog ont la possibilité de créer, modifier et supprimer des articles, tandis que les visiteurs peuvent lire et interagir avec le contenu en laissant des commentaires.

De manière spécifique, le Service Article met en œuvre une filtration par catégorie, permettant aux utilisateurs de découvrir facilement des articles spécifiques dans des domaines qui les intéressent. De plus, une fonction de recherche par mot offre la possibilité de trouver rapidement des articles en fonction des termes clés. En outre, une filtration par date permet aux utilisateurs de parcourir les articles en fonction de leur date de publication, offrant ainsi une exploration chronologique du contenu.

Chaque service fonctionne de manière indépendante, contribuant à la modularité de l'application. Le service Utilisateur assure la sécurité et la gestion des comptes des propriétaires, tandis que le service Article gère le contenu dynamique du blog. Cette architecture permet une évolutivité et une maintenance plus aisées de l'application, offrant une expérience fluide tant pour les visiteurs que pour les propriétaires du blog.

2.3 Mécanismes de communication

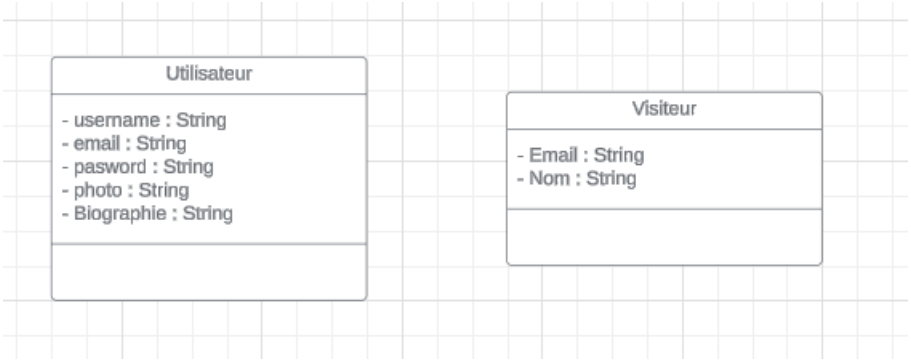
Les mécanismes de communication dans notre application reposent sur l'utilisation d'OpenFeign. OpenFeign est une bibliothèque Java qui simplifie la communication entre les microservices en permettant aux clients de faire des requêtes HTTP à des services distants de manière déclarative. Avec OpenFeign, nous avons mis en place des interfaces décrivant les services distants, et la bibliothèque génère automatiquement le code nécessaire pour effectuer les appels HTTP.

L'utilisation d'OpenFeign simplifie la communication entre nos microservices en évitant la nécessité d'écrire manuellement le code pour les appels HTTP, tout en fournissant une approche déclarative et intuitive. Cette approche facilite la maintenance et l'évolutivité de notre architecture microservices, en assurant une communication efficace et cohérente entre les différents composants de notre application.

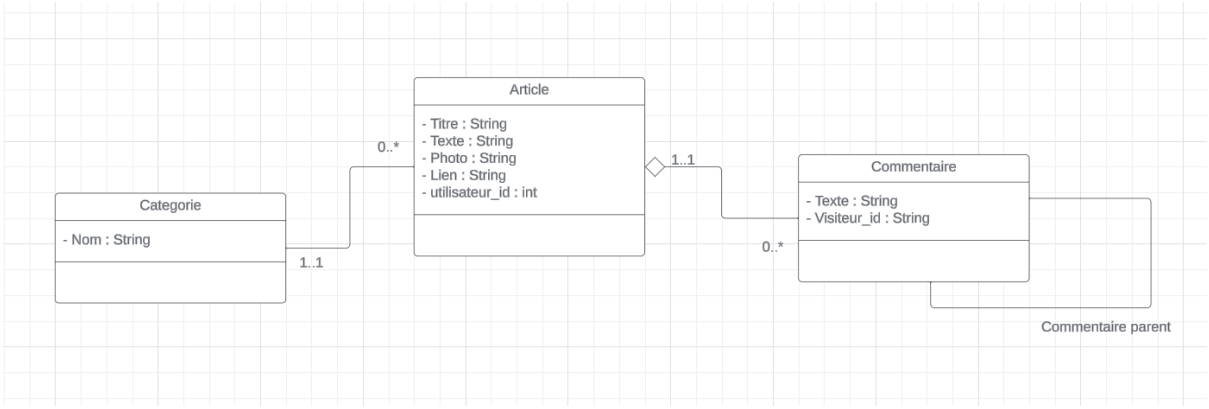
3. Conception des Microservices

Avant de plonger dans les détails visuels,l'essence de notre conception des microservices pour notre projet de blog personnel. Chaque service, qu'il s'agisse du Service Utilisateur ou du Service Article, est soigneusement élaboré selon une approche de conception minutieuse. voila notrestructure et la logique qui sous-tendent ces services à travers nos diagrammes de classe à venir.

- **Service Utilisateur :**



- Service Article :



4. Conteneurisation avec Docker

4.1 Avantage de docker

Docker présente plusieurs avantages, et dans le contexte du projet de blog personnel, il apporte des bénéfices significatifs :

Isolation et Portabilité : Docker encapsule une application et ses dépendances dans des conteneurs légers. Cela offre une isolation efficace, garantissant que chaque composant du projet, tel que le Service Utilisateur et le Service Article, fonctionne de manière cohérente et indépendante de l'environnement.

Facilité de Déploiement : Les conteneurs Docker peuvent être déployés de manière rapide et reproductible. Cela simplifie le processus de déploiement des microservices, garantissant que l'application fonctionne de manière cohérente sur différents environnements, qu'il s'agisse d'un environnement de développement, de test ou de production.

Gestion des Dépendances : Docker permet de spécifier les dépendances de manière déclarative, ce qui facilite la gestion des bibliothèques et des composants nécessaires à chaque service du projet. Cela évite les conflits de dépendances et assure une cohérence dans les environnements de développement et de production.

Évolutivité : Les conteneurs Docker peuvent être mis à l'échelle de manière flexible, permettant une gestion efficace des ressources en fonction des besoins. Cela facilite l'adaptation de l'infrastructure aux variations de charge de l'application, assurant une performance optimale.

4.2 Implémentation

4.2.1 Configuration docker pour :

- Microservices article :

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /article-app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} article-app.jar
ENTRYPOINT ["java","-jar","/article-app.jar"]
```

- **eureka server :**

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /eureka-server
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} eureka-server.jar
ENTRYPOINT ["java","-jar","/eureka-server.jar"]
```

- **gateway :**

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /eureka-server
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} eureka-server.jar
ENTRYPOINT ["java","-jar","/eureka-server.jar"]
```

- **Utilisateur :**

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /utilisateur-app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} utilisateur-app.jar
ENTRYPOINT ["java","-jar","/utilisateur-app.jar"]
```

- **Front**

```
FROM node:14.15.0-alpine as builder
WORKDIR /usr/src/app
COPY . .
RUN npm install
RUN npm run build
FROM nginx:alpine
COPY --from=builder /usr/src/app/build /usr/share/nginx/html
```

4.2.2 Docker compose

Docker Compose, l'outil qui harmonise les services de notre projet de blog personnel. Chaque élément s'emboîte parfaitement dans cette partition de conteneurs, orchestrée pour une expérience de déploiement transparente. Plongeons dans le code de Docker Compose qui donne vie à notre architecture microservices en unifié tout-en-un.

```yaml
version: '3'

services:
  # Eureka Service
  eureka-server:
    build:
      context: ./Back_Blog/Eureka-Server
    ports:
      - "8761:8761"
    networks:
      - n3

  # Gateway Service
  gateway-service:
    build:
      context: ./Back_Blog/Gateway
    ports:
      - "8888:8888"
    depends_on:
      - eureka-server
    environment:
      EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server:8761/eureka/
    networks:
      - n3

  # MySQL Utilisateur
  mysql-utilisateur:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: service-utilisateur
      MYSQL_PASSWORD: root
    ports:
      - "3306:3306"
    networks:
      - n3

  # MySQL Article
  mysql-article:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: service-article
      MYSQL_PASSWORD: root
    ports:
      - "3307:3306"
    networks:
      - n3

  # Utilisateur Service
  utilisateur-service:
    build:
      context: ./Back_Blog/Utilisateur
    ports:
      - "8088:8088"
    depends_on:
      - eureka-server
      - mysql-utilisateur
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql-utilisateur:3306/service-utilisateur
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
      eureka.client.serviceUrl.defaultZone: http://eureka-server:8761/eureka/
    healthcheck:
      test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW DATABASES;\""
      interval: 5s
      timeout: 2s
      retries: 100
    networks:
      - n3

  # Article Service
  article-service:
    build:
      context: ./Back_Blog/Article
    ports:
      - "8089:8089"
    depends_on:
      - eureka-server
      - mysql-article
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql-article:3306/service-article
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
      eureka.client.serviceUrl.defaultZone: http://eureka-server:8761/eureka/
    healthcheck:
      test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW DATABASES;\""
      interval: 5s
      timeout: 2s
      retries: 100
    networks:
      - n3

  frontend:
    build:
      context: ./Front_Blog
    ports:
      - "3000:80"
    depends_on:
      - article-service
      - utilisateur-service

  phpmyadmin-utilisateur:
    image: phpmyadmin/phpmyadmin
    environment:
      PMA_HOST: mysql-utilisateur
      PMA_PORT: 3306
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "8082:80"
    networks:
      - n3

  phpmyadmin-article:
    image: phpmyadmin/phpmyadmin
    environment:
      PMA_HOST: mysql-article
      PMA_PORT: 3306
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "8083:80"
    networks:
      - n3

networks:
  n3:
    external: true
```

4.2.3 Docker

5. CI/CD avec Jenkins

5.1 Processus

Jenkins (https://www.jenkins.io/) est un outil open source d'automatisation des tâches, largement utilisé pour simplifier les processus de construction, de tests et de déploiement de logiciels. En mettant l'accent sur l'intégration continue (CI) et la livraison continue (CD), Jenkins offre une solution polyvalente pour automatiser les tâches récurrentes du cycle de vie du développement logiciel.

- **Processus de CI/CD avec Jenkins :**

1. Intégration Continue (CI) :

- Les développeurs envoient leurs modifications au référentiel Git.

- Jenkins automatise la compilation, les tests, et génère des rapports.

- Les développeurs sont notifiés en cas d'échec pour des corrections rapides.

2. Livraison Continue (CD) :

- Jenkins déploie automatiquement le code sur un environnement de test après une CI réussie.

- Des tests supplémentaires sont effectués pour garantir la stabilité.

- Après approbation, le déploiement automatique en production est déclenché.

Ce processus garantit une automatisation efficace du développement, des tests et du déploiement, assurant une livraison continue de logiciels de qualité. La rétroaction rapide permet d'identifier et de résoudre rapidement les problèmes, contribuant ainsi à des améliorations constantes dans le processus de développement.

5.2 configuration

MYSQL doit être runner dans le port 3306 pendant le build et stopper pendant le run des images docker

Script jenkins :

```
pipeline {
    agent any
    tools {
        maven 'maven'
    }
    stages {
        stage('Git Clone') {
            steps {
                script {
                    checkout([$class: 'GitSCM', branches: [[name : 'main']],
                        userRemoteConfigs: [[url:
'https://github.com/botazitihssane/ACE_Microservices.git']]])
                }
            }
        }

        stage('Build Backend'){
            steps {
                script {
                    dir('Back_Blog/Eureka-Server') {
                        bat 'mvn clean install'
```

```
            }
            dir('Back_Blog/Gateway') {
                bat 'mvn clean install'
            }
            dir('Back_Blog/Utilisateur') {
                bat 'mvn clean install'
            }
            dir('Back_Blog/Article') {
                bat 'mvn clean install'
            }
        }
    }
}

stage('Create Docker Image (Backend) '){
    steps {
        dir('Back_Blog/Eureka-Server') {
            bat 'docker build -t blog/eurekaserver .'
        }
        dir('Back_Blog/Gateway') {
            bat 'docker build -t blog/gateway .'
        }
        dir('Back_Blog/Utilisateur') {
            bat 'docker build -t blog/utilisateur .'
        }
```

```
        dir('Back_Blog/Article') {
            bat 'docker build -t blog/article .'
        }

    }
}

stage('Build Frontend'){
    steps {
        script {
            dir('Front_Blog'){
                bat 'npm install'
            }
        }
    }
}

stage('Create docker image (front)'){
    steps {
        dir('Front_Blog'){
            bat 'docker build -t blog/front .'
        }
    }
}
```

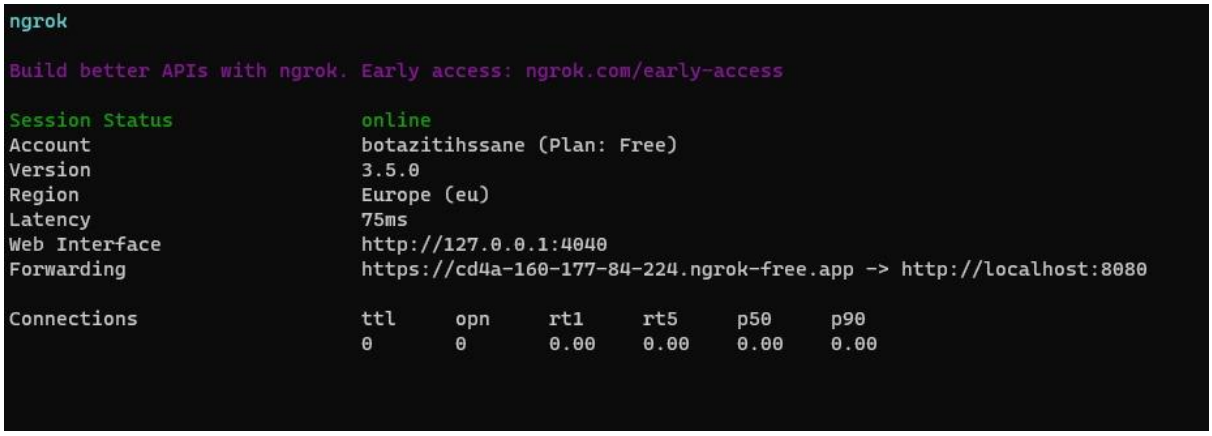
```
        stage('Run'){
            steps {
                script {
                    bat 'docker-compose down'
                    bat 'docker-compose up -d'
                }
            }
        }


    }
}
```

6. Déploiement Automatique

Ngrok est un outil permettant de créer un tunnel sécurisé depuis un point d'accès public vers un serveur local. En utilisant Ngrok, il devient possible d'exposer temporairement l'instance Jenkins locale de manière sécurisée sur Internet. Voici comment utiliser Ngrok

En crée un tunnel pour une instance de Jenkins et on exécute la commande suivante pour pouvoir y accéder : ngrok http 8080



7. Conclusion

Ce projet nous a été d'un très grand bénéfice aussi bien au niveau technique qu'au niveau professionnel. En effet, non seulement nous avons pu raffiner nos capacités d'abstraction et de conception et mettre en œuvre les connaissances acquises durant notre formation, mais nous avons aussi développé notre créativité et notre esprit de travail, ainsi que notre sens relationnel.

Sur le plan personnel, ce projet fut un véritable test de nos capacités à résoudre, d'une façon autonome, les problèmes posés quotidiennement et de notre aptitude à assumer les responsabilités qui nous ont été confiées.

Les difficultés majeures rencontrées durant la réalisation de projet étaient par rapport à la diversité des outils utilisés, majoritairement complexes mais vigoureux et consistants. En plus du travail réalisé, nous avons découvert beaucoup d'outils même si nous ne les avons pas utilisés.

Les perspectives d'évolution restent extrêmement ouvertes que ce soit sur le plan de développement des nouveaux modules ou l'optimisation des performances et sécurité. Pour conclure, nous espérons ainsi avoir répondu aux attentes de nos encadrants et que l'application soit utile.