**Rapport Projet**

**1. Introduction**

Au cœur de ce projet se trouve le développement d'une application de gestion des liens favoris, une plateforme conçue pour simplifier et améliorer l'expérience de l'utilisateur dans l'organisation et la gestion de ses ressources en ligne préférées. Dans cette introduction, nous mettrons en lumière l'essence de cette application et soulignerons l'importance cruciale de l'architecture microservices dans sa conception.

* Aperçu du Projet

L'application de gestion des liens favoris vise à offrir aux utilisateurs un moyen efficace de sauvegarder, organiser et partager leurs liens web préférés. Cette plateforme répond à un besoin croissant d'optimiser la gestion de la multitude de ressources en ligne auxquelles les utilisateurs accèdent quotidiennement. Que ce soit pour des articles intéressants, des tutoriels, des vidéos ou d'autres contenus web, l'application permettra aux utilisateurs de centraliser et de personnaliser leurs favoris de manière intuitive.

* Importance de l'Architecture Microservices

L'adoption de l'architecture microservices revêt une importance majeure dans le cadre de ce projet. En décomposant l'application en services indépendants et spécialisés, nous visons à garantir une flexibilité et une évolutivité optimales. Chaque microservice sera dédié à des fonctionnalités spécifiques, tels que la gestion des favoris, l'authentification des utilisateurs, et la gestion des liens.

Cette approche modulaire permet non seulement une maintenance simplifiée, mais elle offre également la possibilité d'évoluer chaque composant indépendamment en fonction des besoins changeants de l'application. Les avantages de l'architecture microservices se manifesteront dans la facilité de déploiement, la résilience du système, et la possibilité de mettre en œuvre des mises à jour sans perturber l'ensemble de l'application.

Dans les sections à venir, nous explorerons en détail l'architecture de l'application, la conception spécifique des microservices, la conteneurisation avec Docker, le déploiement continu avec Jenkins, ainsi que l'intégration de SonarQube pour garantir la qualité du code. Ces éléments, combinés, contribueront à faire de cette application de gestion des liens favoris une solution robuste, efficace et adaptable aux besoins divers des utilisateurs.

**2. Architecture Microservices**

* Architecture

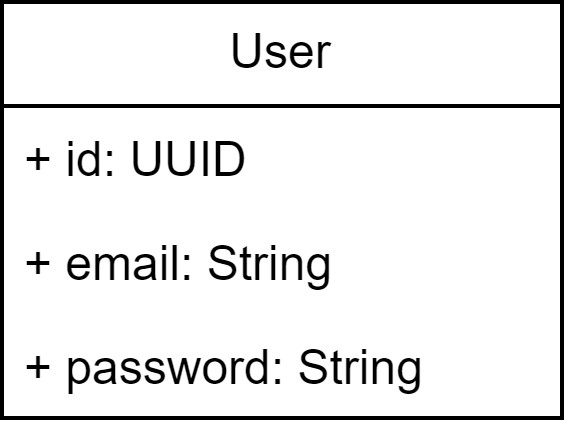
A diagram of a ship

Description automatically generated

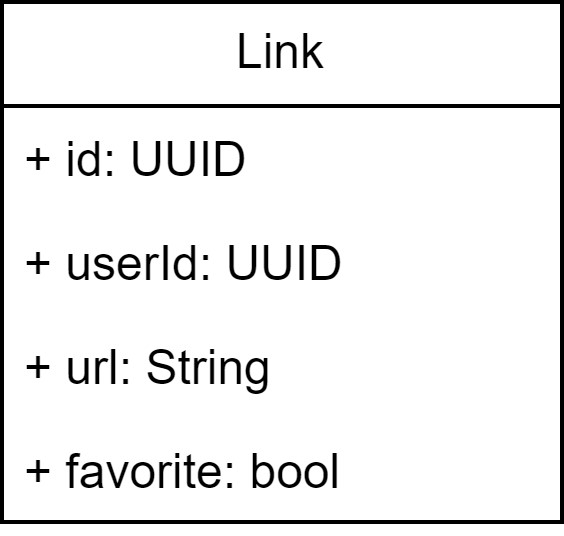
* Description des services
  + Auth service pour l’authentification et inscription d’un utilisateur.
  + Links service qui permettent à chaque utilisateur de gérer ses liens favoris.
  + Gateway service est l’enter à l’API c’est lui le responsable de vérifier le JWT de chaque utilisateur selon le besoin et aussi il joue le rôle d’un proxy qui dispatch selon l'endpoint vers le microservice associé.
  + Eureka service pour découverte du service il permet aux services de se trouver et de communiquer entre eux sans coder le nom d’hôte et le port.

**3. Conception des Microservices**

* Microservice auth



* Microservice links



**4. Conteneurisation avec Docker**

* Implémentation et avantages
* Backend
  + docker-compose.yml

version: '3'  
services:  
 auth-service:  
 container\_name: auth-service  
 build:  
 context: ./auth-service  
 dockerfile: Dockerfile  
 environment:  
 - SPRING\_DATASOURCE\_URL=jdbc:mysql://db:3306/auth-service  
 - SPRING\_DATASOURCE\_USERNAME=root  
 - SPRING\_DATASOURCE\_PASSWORD=  
 - EUREKA\_DEFAULT\_ZONE=http://eureka-service:8761/eureka  
 depends\_on:  
 - db  
 - eureka-service  
 - gateway-service  
  
 links-service:  
 container\_name: links-service  
 build:  
 context: ./links-service  
 dockerfile: Dockerfile  
 environment:  
 - SPRING\_DATASOURCE\_URL=jdbc:mysql://db:3306/links-service  
 - SPRING\_DATASOURCE\_USERNAME=root  
 - SPRING\_DATASOURCE\_PASSWORD=  
 - EUREKA\_DEFAULT\_ZONE=http://eureka-service:8761/eureka  
 depends\_on:  
 - db  
 - eureka-service  
 - gateway-service  
   
 db:  
 container\_name: db  
 image: mysql:8  
 volumes:  
 - ./docker/db:/var/lib/mysql  
 - ./docker/initdb:/docker-entrypoint-initdb.d  
 environment:  
 MYSQL\_ROOT\_PASSWORD: ""  
 MYSQL\_ALLOW\_EMPTY\_PASSWORD: yes  
 phpmyadmin:  
 container\_name: phpmyadmin  
 image: phpmyadmin:latest  
 environment:  
 PMA\_HOST: db  
 PMA\_PORT: 3306  
 ports:  
 - '8090:80'  
 depends\_on:  
 - db  
  
 gateway-service:  
 container\_name: gateway-service  
 build:  
 context: ./gateway-service  
 dockerfile: Dockerfile  
 ports:  
 - '9090:8080'  
 environment:  
 - EUREKA\_DEFAULT\_ZONE=http://eureka-service:8761/eureka  
 depends\_on:  
 - eureka-service  
   
 eureka-service:  
 container\_name: eureka-service  
 build:  
 context: ./eureka-service  
 dockerfile: Dockerfile  
 ports:  
 - '8761:8761'

* + Dockerfile

Pour chaque microservice ils ont le même Dockerfile mais avec une différence dans le ENTRYPOINT ou on spécifie le nom du jar générer spécifique au microservice

FROM maven:3.6.3-openjdk-17  
  
WORKDIR /app  
  
COPY . /app  
  
RUN mvn clean package  
  
ENTRYPOINT ["java", "-jar", "target/links-service.jar"]

* Frontend
  + Dockerfile

FROM node:20.9 as *build*WORKDIR /app  
  
COPY . /app  
  
RUN npm install -g @angular/cli  
  
RUN npm install  
  
RUN ng build  
  
FROM nginx:stable  
  
COPY --from=*build* /app/dist/my-links-web/browser /usr/share/nginx/html  
  
COPY --from=*build* /app/nginx.conf /etc/nginx/conf.d/default.conf  
  
ENTRYPOINT ["nginx", "-g", "daemon off;"]  
  
EXPOSE 80

* + Configuration Nginx

server {  
 listen 80;  
  
 root /usr/share/nginx/html;  
 index index.html;  
  
 location / {  
 try\_files $uri $uri/ /index.html;  
 }  
}

**5. CI/CD avec Jenkins**

* Cree un nouveau item
* Cocher et remplissez si possible les champs

A screenshot of a computer

Description automatically generated



* Script pipeline
  + Backend

pipeline {

agent any

tools {

maven 'maven'

}

environment {

SONAR\_TOKEN = credentials('sonarcloud-token')

}

stages {

stage('Git') {

steps {

git branch: 'main', url: 'https://github.com/voidnowhere/my-links-api.git'

}

}

stage('SonarQube') {

steps {

dir('links-service') {

sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=voidnowhere\_my-links-api-links-service -Dsonar.organization=voidnowhere -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

dir('auth-service') {

sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=voidnowhere\_my-links-api-auth-service -Dsonar.organization=voidnowhere -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

dir('gateway-service') {

sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=voidnowhere\_my-links-api-gateway-service -Dsonar.organization=voidnowhere -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

}

}

stage('Docker build') {

steps {

sh 'docker compose build --no-cache'

}

}

stage('Docker run') {

steps {

sh 'docker compose up -d'

}

}

stage('Cleanup') {

steps {

sh 'docker image prune -f'

}

}

}

}

* + Frontend

pipeline {

agent any

environment {

SONAR\_TOKEN = credentials('sonarcloud-token')

}

stages {

stage('Git') {

steps {

git branch: 'main', url: 'https://github.com/voidnowhere/my-links-web.git'

}

}

stage('SonarQube') {

steps {

script {

def scannerHome = tool 'sonar-scanner'

sh "${scannerHome}/bin/sonar-scanner -Dsonar.organization=voidnowhere -Dsonar.projectKey=voidnowhere\_my-links-web -Dsonar.sources=. -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

}

}

stage('Docker build') {

steps {

sh 'docker build -t my-links-web .'

}

}

stage('Docker run') {

steps {

sh 'docker rm -f my-links-web'

sh 'docker run -d -p 4200:80 --name my-links-web --rm my-links-web'

sh 'docker image prune'

}

}

stage('Cleanup') {

steps {

sh 'docker image prune -f'

}

}

}

}

**6. Déploiement Automatique**

Un VPS sous DigitalOcean contient Jenkins et on a configurer les webhooks dans chaque GitHub repository

A screenshot of a computer error

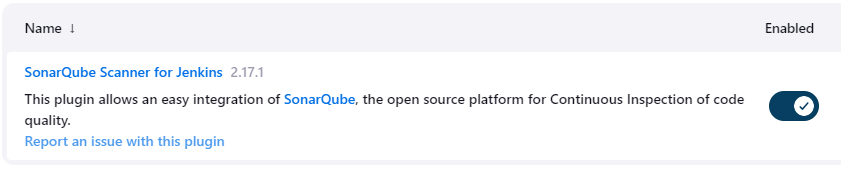
Description automatically generated

A screenshot of a webbook

Description automatically generated

**7. Intégration de SonarQube**

* Installation de l’outil sous Jenkins



* Ajouter sonar scanner à Jenkins

A white paper with blue lines

Description automatically generated

* Génération du token sous sonarcloud

A screenshot of a computer

Description automatically generated

* Ajouter le token générer dans Jenkins

A screenshot of a computer

Description automatically generated

* Intégration dans le pipeline

Il faut déclarer le token comme variable d’environnement dans le pipeline

environment {

SONAR\_TOKEN = credentials('sonarcloud-token')

}

* + Backend

stage('SonarQube') {

steps {

dir('links-service') {

sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=voidnowhere\_my-links-api-links-service -Dsonar.organization=voidnowhere -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

dir('auth-service') {

sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=voidnowhere\_my-links-api-auth-service -Dsonar.organization=voidnowhere -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

dir('gateway-service') {

sh "mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=voidnowhere\_my-links-api-gateway-service -Dsonar.organization=voidnowhere -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

}

}

* + Frontend

stage('SonarQube') {

steps {

script {

def scannerHome = tool 'sonar-scanner'

sh "${scannerHome}/bin/sonar-scanner -Dsonar.organization=voidnowhere -Dsonar.projectKey=voidnowhere\_my-links-web -Dsonar.sources=. -Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${SONAR\_TOKEN}"

}

}

}

**8. Conclusion**

Dans cette conclusion, le projet de développement d'une application de gestion des liens favoris repose sur une architecture microservices, avec une mise en avant de l'importance de cette approche pour la flexibilité et l'évolutivité. L'utilisation de Docker pour la conteneurisation et Jenkins pour la CI/CD est soulignée, tout comme l'intégration de SonarQube pour garantir la qualité du code. Les succès comprennent la mise en place réussie de l'architecture, le déploiement automatisé sur un VPS DigitalOcean, et l'amélioration continue grâce à la détection précoce des problèmes de code. Pour l'avenir, l'accent est mis sur l'expansion des fonctionnalités et l'adaptation aux retours des utilisateurs. En résumé, le projet offre une base solide pour une application moderne et évolutive.