

# Architecture des Composants d'Entreprise

Gestion des RH

ABALLAGH Nezar / FRIHA Yassir  
2023/2024

## Table des matières

Introduction .....	2
Aperçu du Projet .....	2
Importance de l'Architecture Microservices.....	2
Architecture Microservices.....	2
Architecture .....	2
Description des Services .....	3
Service Employés : .....	3
Service Recrutement :.....	3
Service Positions : .....	3
Service Paie : .....	3
Service Congés/Jours Fériés : .....	3
Mécanismes de Communication .....	4
Conception des Microservices .....	4
Approche de Conception pour Chaque Service .....	4
Conteneurisation avec Docker .....	4
Implémentation et Avantages.....	4
CI/CD avec Jenkins.....	8
Processus et Configuration .....	8
Déploiement Automatique .....	14
Utilisation de Ngrok ou Azure Cloud .....	14
Intégration de SonarQube .....	14
Configuration et Bénéfices pour la Qualité du Code .....	14
Conclusion.....	15
Résumé des Accomplissements .....	15
Perspectives Futures.....	15

# Introduction

## Aperçu du Projet

Le projet de gestion des ressources humaines s'articule autour d'une architecture microservices, offrant une solution complète pour les besoins complexes des départements RH. Cette approche modulaire vise à faciliter la gestion des employés, des recrutements, des positions, de la paie et des congés au sein d'une entreprise.

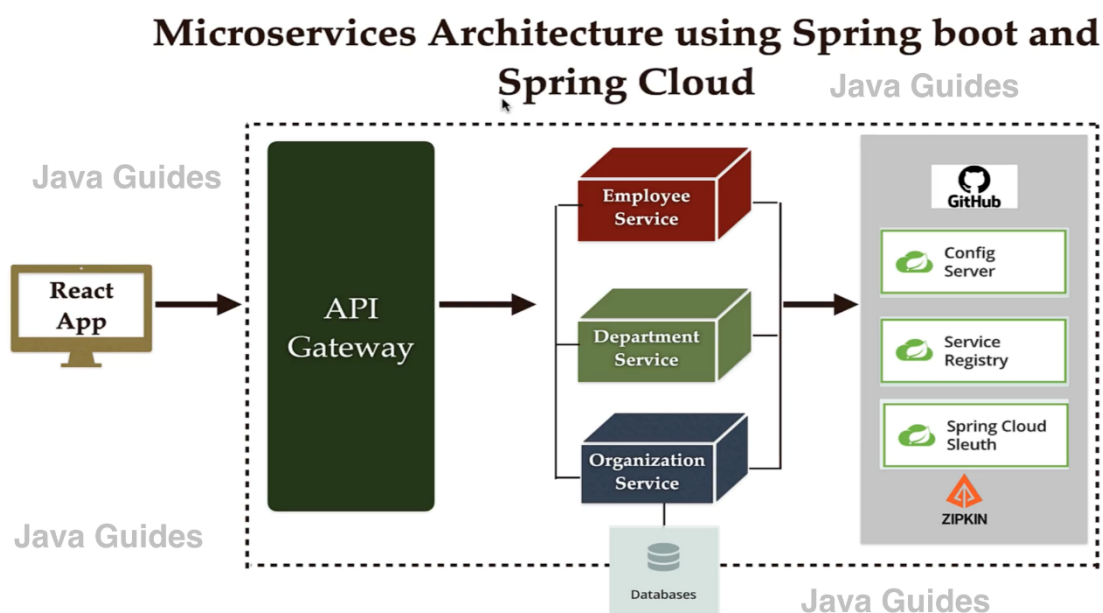
## Importance de l'Architecture Microservices

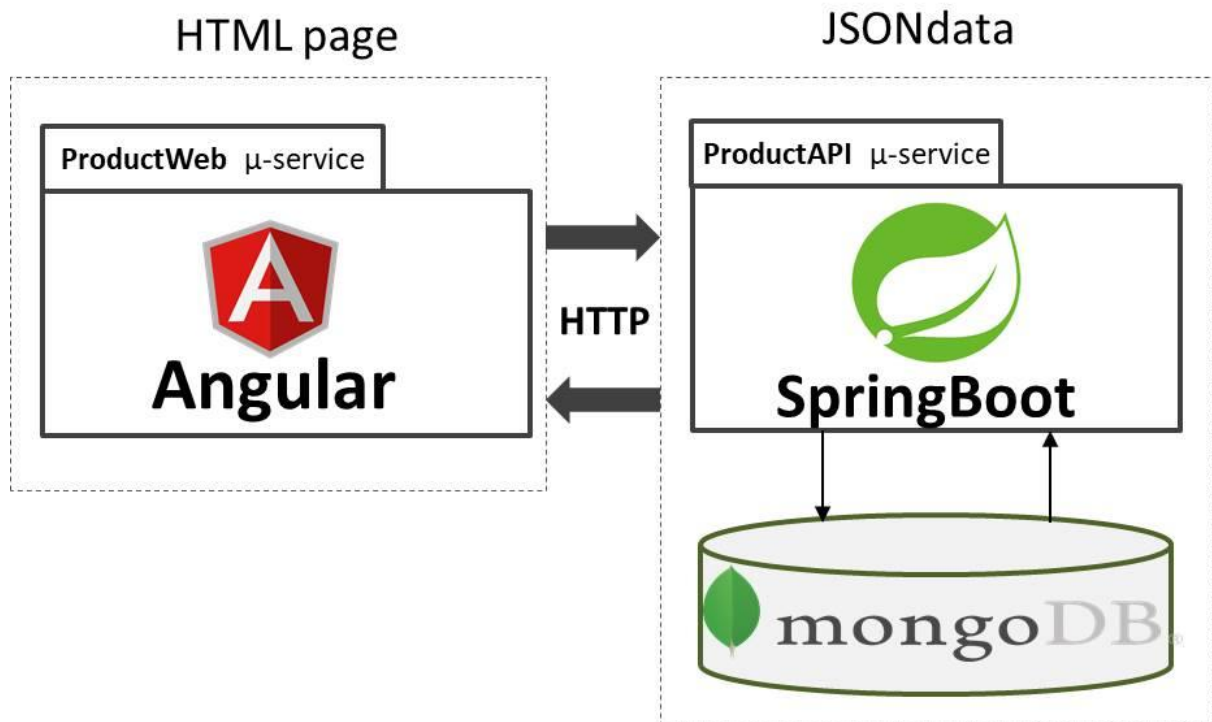
L'architecture microservices a été privilégiée pour sa capacité à offrir une modularité accrue, une évolutivité simplifiée et une maintenance indépendante des services. Cette structure permet de répondre efficacement aux demandes changeantes de la gestion des ressources humaines dans un environnement d'entreprise dynamique.

## Architecture Microservices

### Architecture

L'architecture microservices repose sur cinq modules distincts, chacun dédié à une fonctionnalité spécifique de la gestion des ressources humaines. Ces microservices, développés en Java avec Spring Boot et en JavaScript avec NodeJS, couvrent un large spectre des opérations RH.





## Description des Services

### Service Employés :

Gère les informations personnelles des employés et des candidats.

Offre des fonctionnalités telles que la date d'embauche, l'e-mail, le rôle, la date de naissance, etc.

### Service Recrutement :

Gère les informations liées au recrutement, notamment les offres d'emploi et les candidatures.

### Service Positions :

Gère les informations relatives aux postes, incluant les désignations, les noms de départements et les employés associés.

### Service Paie :

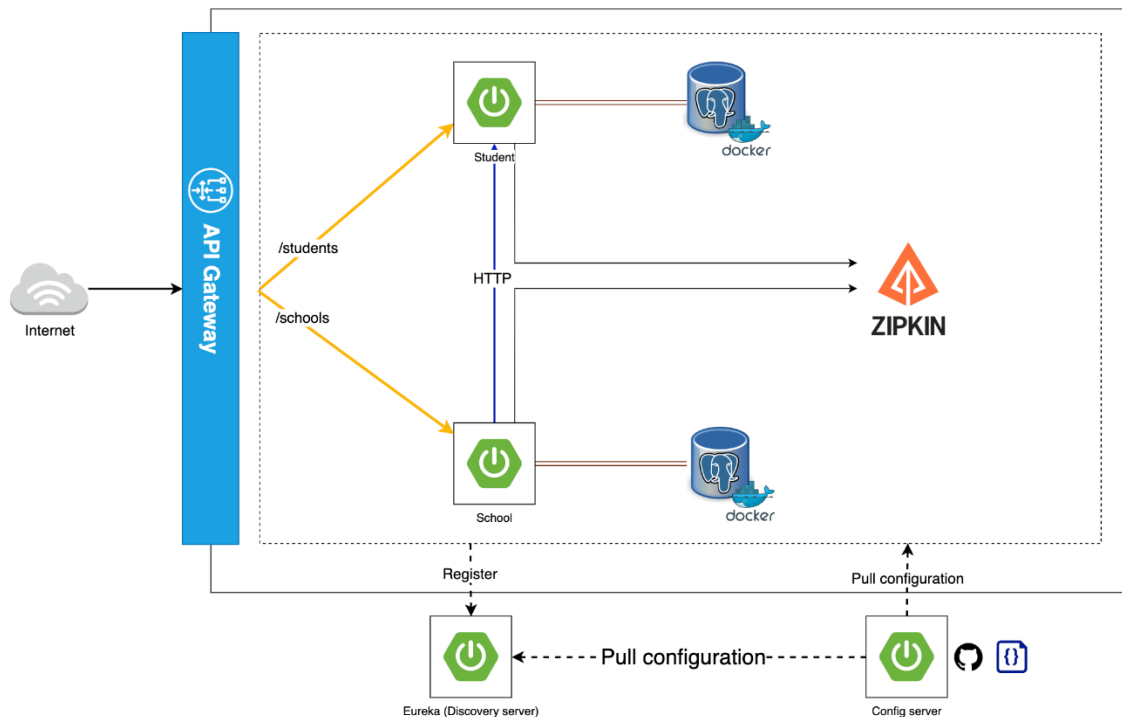
Gère les informations de paie, y compris les salaires des employés.

### Service Congés/Jours Fériés :

Gère les informations sur les congés et les jours fériés, facilitant la gestion des absences et la définition des jours fériés.

## Mécanismes de Communication

Les microservices communiquent entre eux via des API RESTful. Un service de découverte (Eureka) est utilisé pour l'enregistrement et la localisation dynamique des services, assurant une connectivité transparente.



## Conception des Microservices

### Approche de Conception pour Chaque Service

Chaque microservice est conçu de manière indépendante, avec une base de données dédiée. L'approche modulaire permet une évolution aisée de chaque service. Les microservices sont développés en utilisant Spring Boot pour les services Java et NodeJS pour les services JavaScript, garantissant cohérence et performance.

## Conteneurisation avec Docker

### Implémentation et Avantages

La conteneurisation avec Docker a été mise en œuvre pour assurer un environnement d'exécution cohérent et isolé pour chaque microservice. Cette approche simplifie le déploiement, la mise à l'échelle et la gestion des dépendances, assurant une portabilité maximale.

```
Dockerfile X
account-service > Dockerfile > ...
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/account-service.jar account-service.jar
  Comment Code
3 | EXPOSE 8541
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/account-service.jar"]
```

```
Dockerfile X
auth-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM node:16
  Comment Code
2 | WORKDIR /usr/src/app
  Comment Code
3 | COPY package*.json ./
  Comment Code
4 | RUN npm install
  Comment Code
5 | COPY . .
  Comment Code
6 | CMD ["npm", "start"]
```

```
Dockerfile X
conge-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/conge-service.jar conge-service.jar
  Comment Code
3 | EXPOSE 5555
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/conge-service.jar"]
5 |
```

```
Dockerfile X
contrat-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | + ADD target/contrat-service.jar contrat-service.jar
  Comment Code
3 | EXPOSE 2222
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/contrat-service.jar"]
5 |
```

```
Dockerfile X
customer-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/customer-service.jar customer-service.jar
  Comment Code
3 | EXPOSE 8542
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/customer-service.jar"]
5 |
```

```
Dockerfile X
discovery-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/discovery-service.jar discovery-service.jar
  Comment Code
3 | + EXPOSE 8761
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/discovery-service.jar"]
5 |
```

```
employee-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/employee-service.jar employee-service.jar
  Comment Code
3 + EXPOSE 1111
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/employee-service.jar"]
5
```

```
gateway-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 + FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/gateway-service.jar gateway-service.jar
  Comment Code
3 | EXPOSE 8765
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/gateway-service.jar"]
5
```

```
position-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 + FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/position-service.jar position-service.jar
  Comment Code
3 | EXPOSE 3333
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/position-service.jar"]
5
```



```
help ← → control_microservice

Dockerfile X

recruitment-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/recruitment-service.jar recruitment-service.jar
  Comment Code
3 | EXPOSE 4444
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/recruitment-service.jar"]
5 |
```

```
Dockerfile X

zipkin-service > Dockerfile > FROM
  Click here to ask Blackbox to help you code faster | Comment Code |
1 | FROM openjdk:8-jdk-alpine
  Comment Code
2 | ADD target/zipkin-service.jar zipkin-service.jar
  Comment Code
3 + EXPOSE 9411
  Comment Code
4 | ENTRYPOINT ["java", "-jar", "/zipkin-service.jar"]
5 |
```

## CI/CD avec Jenkins

### Processus et Configuration

Jenkins est utilisé pour automatiser l'intégration continue et le déploiement continu. Le pipeline Jenkins englobe les étapes de récupération du code, de construction, de tests, d'analyse de la qualité du code avec SonarQube, et de déploiement des microservices avec Docker Compose.

```
pipeline {
    agent any

    tools {
        maven 'maven'
    }
}
```

```
stages {
    stage('Git Clone') {
        steps {
            script {
                checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url:
'https://github.com/aballagh/control_microservice.git']]])
            }
        }
    }

    stage('Build') {
        steps {
            script{
                dir('POV-JAVA'){
                    bat 'mvn clean install'
                }
            }
        }
    }

    stage('Create Docker Image') {
        steps {
            script{
                dir('POV-JAVA'){
                    bat 'docker build -t lachgar/pos .'
                }
            }
        }
    }
}
```

```
stage('Run') {
    steps {
        script {

            bat "docker run --name test-pos -d -p 8585:8282 lachgar/pos"

        }
    }

}

}

pipeline {
    agent any

    stages {
        stage('Git Clone') {
            steps {
                script {
                    checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url:
'https://github.com/aballagh/control_microservice.git']]])
                }
            }
        }
    }
}
```

```
stages {  
    stage('Checkout') {  
        steps {  
            checkout scm  
        }  
    }  
  
    stage('Build and Package') {  
        steps {  
            script {  
                sh './mvnw clean package -DskipTests=true'  
            }  
        }  
    }  
  
    stage('Unit Tests') {  
        steps {  
            script {  
                sh './mvnw test'  
            }  
        }  
    }  
  
    stage('SonarQube Analysis') {  
        steps {  
            script {  
                // Définition des variables pour les informations du serveur SonarQube  
                def sonarqubeScannerHome = tool 'SonarQubeScanner'  
                def sonarQubeServerUrl = 'http://localhost:9000'  
  
                // Exécution de l'analyse SonarQube avec le Maven Wrapper
```

```

withSonarQubeEnv('SonarQube') {
    sh "${sonarqubeScannerHome}/bin/sonar-scanner \
        -Dsonar.host.url=${sonarQubeServerUrl} \
        -Dsonar.login=${env.SONARQUBE_TOKEN} \
        -Dsonar.projectKey=your-project-key \
        -Dsonar.projectName='microserviceprojet' \
        -Dsonar.sources=."
}
}
}

stage('Dockerize') {
    steps {
        script {
            sh "docker build -t ${DOCKER_IMAGE_PREFIX}/employee-service:latest -f employee-
service/Dockerfile ."
        }
    }
}

stage('Push to Docker Registry') {
    steps {
        script {
            sh "docker push ${DOCKER_IMAGE_PREFIX}/employee-service:latest"
        }
    }
}

stage('Deploy to Docker Compose') {
    steps {

```

```
    script {
        sh 'docker-compose up -d'
    }
}

stage('Integration Tests') {
    steps {
        // Ajoutez des instructions pour les tests d'intégration
    }
}

stage('Quality Checks') {
    steps {
        // Ajoutez des instructions pour d'autres contrôles de qualité
    }
}

stage('Deploy to Production') {
    when {
        expression { params.DEPLOY_TO_PROD == 'true' }
    }
    steps {
        // Ajoutez des instructions pour le déploiement en production
    }
}

post {
    always {
        script {
```

```

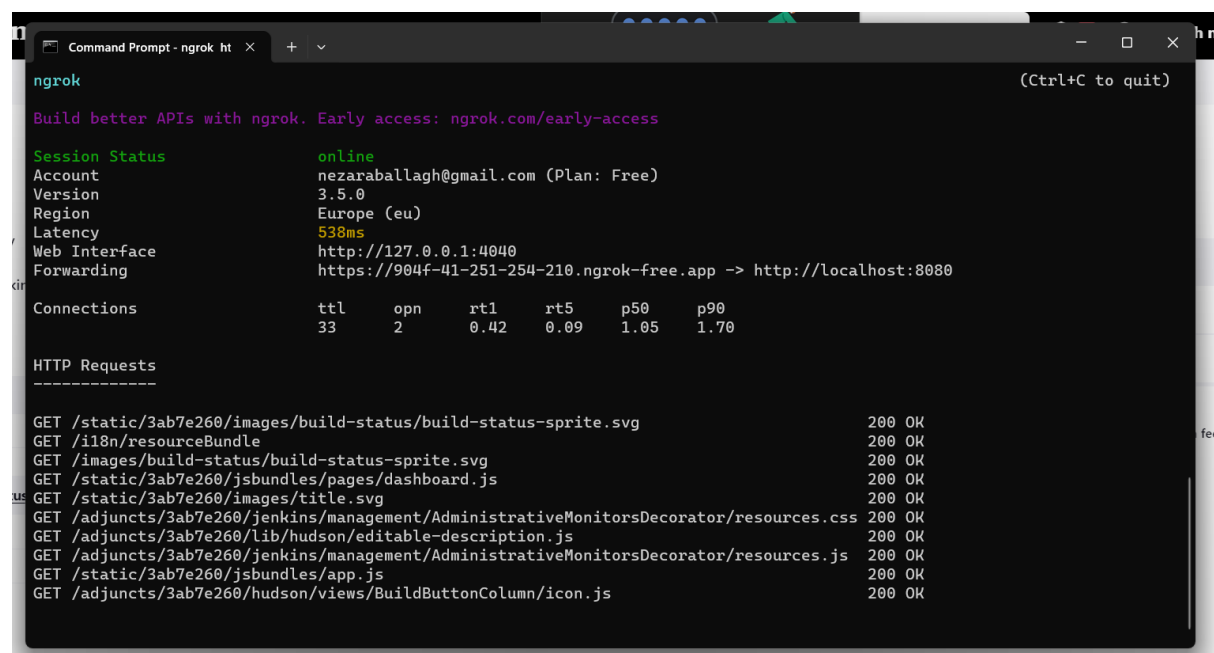
    sh 'docker-compose down'
  }
}
}
}
}

```

## Déploiement Automatique

### Utilisation de Ngrok ou Azure Cloud

Le déploiement automatique des microservices est réalisé avec Docker Compose. Ngrok est utilisé pour les déploiements en développement local, offrant un accès externe aux services. Azure Cloud est préconisé pour les déploiements en production, assurant une infrastructure robuste et évolutive.



```

ngrok
Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status      online
Account             nezaraballagh@gmail.com (Plan: Free)
Version             3.5.0
Region              Europe (eu)
Latency              538ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://904f-41-251-254-210.ngrok-free.app -> http://localhost:8080

Connections          ttl    opn    rt1    rt5    p50    p90
                   33     2     0.42   0.09   1.05   1.70

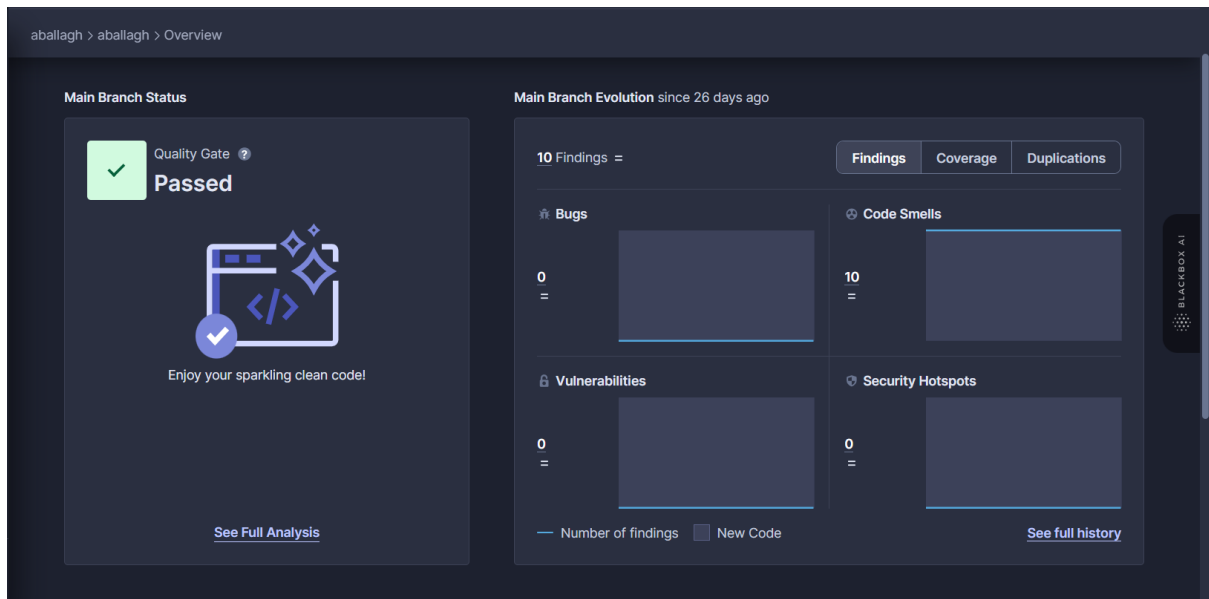
HTTP Requests
-----
GET /static/3ab7e260/images/build-status/build-status-sprite.svg 200 OK
GET /i18n/resourceBundle 200 OK
GET /images/build-status/build-status-sprite.svg 200 OK
GET /static/3ab7e260/jsbundles/pages/dashboard.js 200 OK
GET /static/3ab7e260/images/title.svg 200 OK
GET /adjuncts/3ab7e260/jenkins/management/AdministrativeMonitorsDecorator/resources.css 200 OK
GET /adjuncts/3ab7e260/lib/hudson/editable-description.js 200 OK
GET /adjuncts/3ab7e260/jenkins/management/AdministrativeMonitorsDecorator/resources.js 200 OK
GET /static/3ab7e260/jsbundles/app.js 200 OK
GET /adjuncts/3ab7e260/hudson/views/BuildButtonColumn/icon.js 200 OK

```

## Intégration de SonarQube

### Configuration et Bénéfices pour la Qualité du Code

SonarQube est intégré dans le pipeline Jenkins pour évaluer la qualité du code source. Cette intégration permet d'identifier et de corriger les problèmes potentiels, garantissant la stabilité et la maintenabilité du code.



## Conclusion

### Résumé des Accomplissements

Le projet a réussi à mettre en place une architecture microservices complète, offrant une solution robuste pour la gestion des ressources humaines. L'utilisation de technologies telles que Spring Boot, NodeJS, Docker, Jenkins et SonarQube a contribué au succès global du projet.

### Perspectives Futures

Les développements futurs pourraient inclure l'optimisation de la performance, l'ajout de fonctionnalités avancées et l'exploration de nouvelles technologies pour rester à jour avec les évolutions du secteur. Le projet représente un exemple réussi d'utilisation efficace de technologies modernes pour résoudre des problèmes complexes de gestion des ressources humaines.