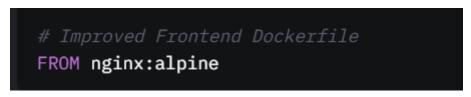
1. Dockerfile Frontend

Ce fichier définit comment construire l'image Docker pour le frontend de l'application. Le frontend utilise **Nginx** (un serveur web léger) pour servir une page HTML statique avec du JavaScript qui communique avec le backend.

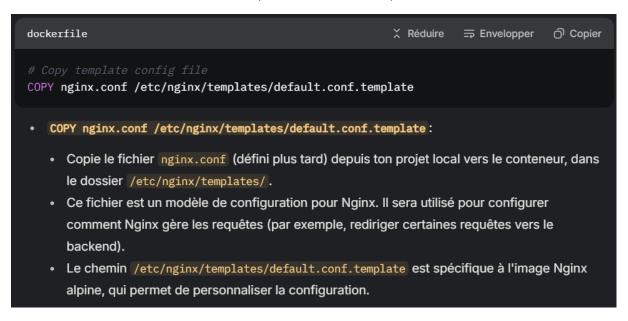


- ② On part de l'image officielle Nginx basée sur Alpine Linux (une version légère de Linux pour réduire la taille de l'image).
- 2 Cela signifie que l'image contient déjà un serveur web Nginx prêt à l'emploi.



WORKDIR:

- Définit le répertoire de travail à l'intérieur du conteneur, ici /usr/share/nginx/html. C'est là où Nginx cherche les fichiers à servir (comme des pages HTML).
- Toutes les commandes suivantes (comme COPY ou RUN) s'exécuteront dans ce dossier.



```
# Create a more detailed index.html
RUN echo '<html><body style="font-family: Arial, sans-serif; max-width: 800px; margin: 0

• RUN echo ... > index.html:

• Crée un fichier index.html dans le répertoire de travail (/usr/share/nginx/html).

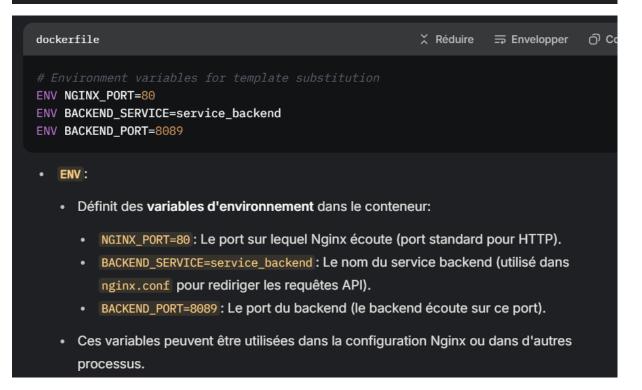
• Le contenu est une page HTML avec:

• Un titre: NexGen Foyer Application.

• Une version dynamique ${BUILD_NUMBER}} (remplacée lors du déploiement).

• Une section pour afficher une liste des foyers (récupérée depuis le backend via une API).
```

- · Un formulaire pour rechercher un foyer par ID.
- · Un script JavaScript qui:
 - Fait une requête HTTP GET à /api/foyer/findAll pour récupérer tous les foyers et les afficher dans une liste.
 - Fait une requête GET à /api/foyer/findById?id=X pour chercher un foyer spécifique par ID et afficher ses détails (ID, nom, capacité).
 - Gère les erreurs en affichant des messages comme "Erreur: ...".
- Le fichier est écrit directement dans le conteneur avec la commande echo et redirigé (>)
 vers index.html.



- Ici, elle lance Nginx avec les arguments:
 - <u>-g "daemon off;"</u>: Force Nginx à s'exécuter au premier plan (plutôt qu'en arrière-plan), ce qui est nécessaire pour que le conteneur reste actif.

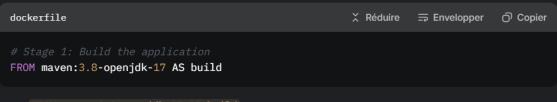
Résumé du Dockerfile frontend:

- · Construit une image basée sur Nginx.
- Copie une configuration personnalisée (nginx.conf).
- Crée une page HTML dynamique avec JavaScript pour interagir avec le backend.
- Configure Nginx pour écouter sur le port 80 et communiquer avec le backend via des variables d'environnement.
- · Lance Nginx quand le conteneur démarre.

2. Dockerfile Backend

Ce fichier définit comment construire l'image Docker pour le backend, une application **Spring Boot** écrite en Java. Il utilise une approche en deux étapes : une pour compiler l'application, une autre pour l'exécuter.

Étape 1 : Compilation (Build Stage)



- FROM maven:3.8-openjdk-17 AS build:
 - Utilise l'image officielle **Maven** avec Java 17 pour compiler l'application.
 - Le mot-clé AS build donne un nom à cette étape (utile pour copier des fichiers dans la deuxième étape).

```
dockerfile

    X Réduire
    X Rédui
                                                                                                                                                                                                                                                                                                                                                              ○ Copier
                                                                                                                                                                                                                                                                                                   ⇒ Envelopper
WORKDIR /app
  WORKDIR /app:
               • Définit le répertoire de travail à /app dans le conteneur.
               • Toutes les commandes suivantes s'exécuteront dans ce dossier.
dockerfile
                                                                                                                                                                                                                                                    ○ Copier
RUN echo "nameserver 8.8.8.8" > /etc/resolv.conf && \
                  echo "nameserver 1.1.1.1" >> /etc/resolv.conf
         RUN echo ... > /etc/resolv.conf:
               • Configure les serveurs DNS dans le fichier /etc/resolv.conf pour garantir une résolution
                          de noms stable.
               • Utilise les serveurs DNS publics de Google (8.8.8.8) et Cloudflare (1.1.1.1).
               • Cela aide Maven à se connecter aux dépôts de dépendances sans problèmes réseau.
```

```
dockerfile
                                                         ○ Copier
COPY pom.xml .
RUN mkdir -p ~/.m2 && \
    echo '<settings><mirror><id>central</id><url>https://repo1.maven.org/maven2
COPY pom.xml .:
   • Copie le fichier pom.xml (fichier de configuration Maven) dans le répertoire de travail
      ( /app ).
   • Ce fichier liste les dépendances et les instructions pour compiler l'application.
  RUN mkdir -p ~/.m2 && echo ... > ~/.m2/settings.xml:
   • Crée le dossier ~/.m2 (où Maven stocke ses configurations et dépendances).
   • Crée un fichier settings.xml pour configurer Maven pour utiliser le dépôt central officiel
      (repo1.maven.org) comme miroir.
   • Cela garantit que Maven télécharge les dépendances depuis une source fiable.
```

```
dockerfile 

# Attempt to resolve dependencies

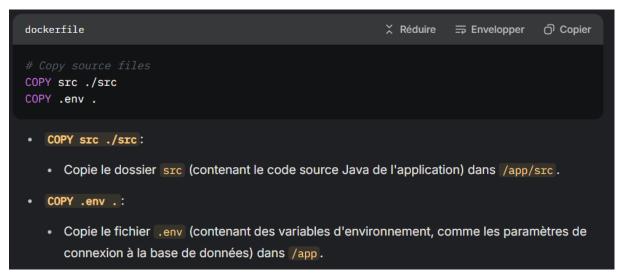
RUN mvn dependency:go-offline -B || echo "Dependency resolution issues, but continuing...

• RUN mvn dependency:go-offline -B:

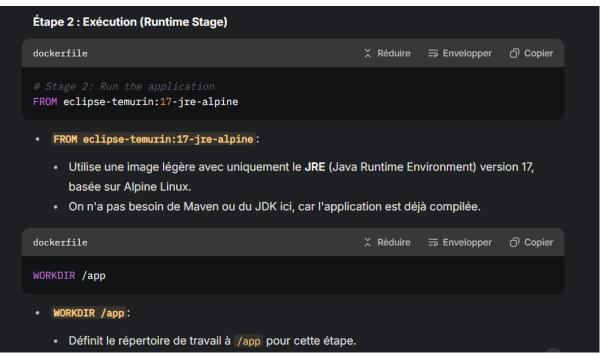
• Exécute une commande Maven pour télécharger toutes les dépendances listées dans pom.xml à l'avance.

• L'option -B (batch mode) rend l'exécution non interactive (utile pour l'automatisation).

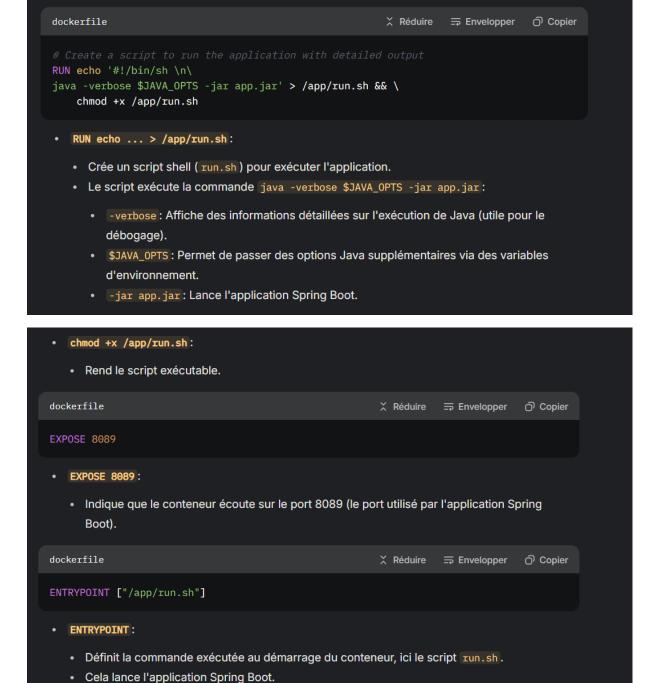
• Si la résolution échoue, un message est affiché (echo ...), mais le processus continue (grâce à || ).
```





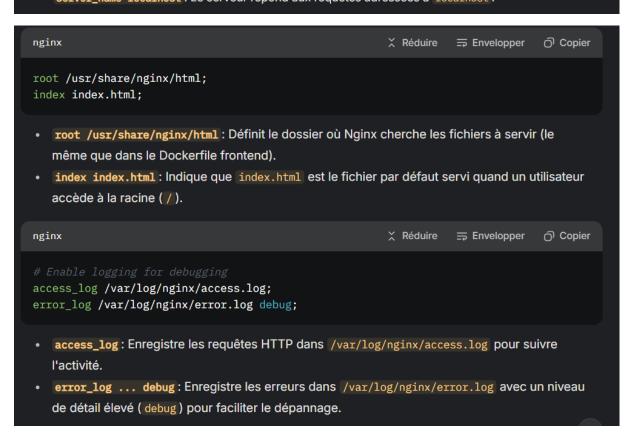


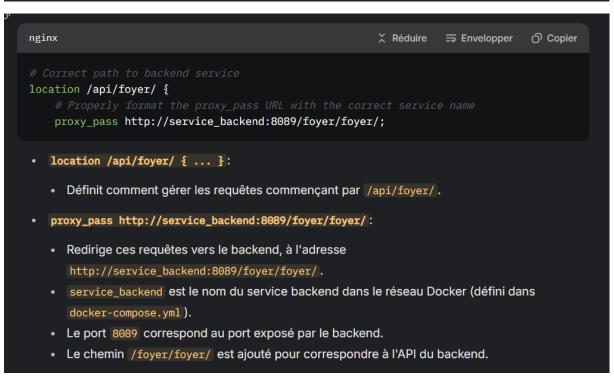




Résumé du Dockerfile backend:

- Étape 1: Compile l'application Spring Boot avec Maven, télécharge les dépendances, et génère un fichier JAR.
- Étape 2: Crée une image légère pour exécuter le JAR avec Java 17, copie le JAR et le fichier .env , et configure un script pour lancer l'application.
- L'application écoute sur le port 8089.



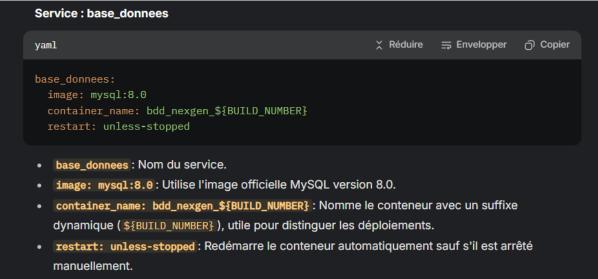


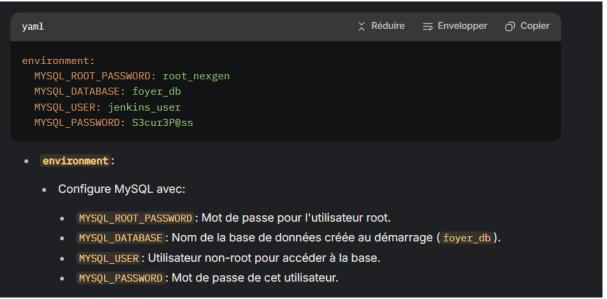
• X-Forwarded-Proto \$scheme: Le protocole utilisé (http ou https).

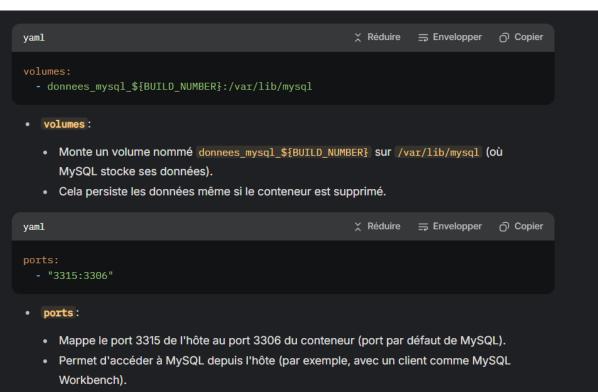
nginx add header 'Access-Control-Allow-Origin' '*' always; add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS' always; add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-S add_header ...: · Ajoute des en-têtes CORS (Cross-Origin Resource Sharing) pour permettre au frontend (servi sur un domaine différent) de communiquer avec le backend. • Access-Control-Allow-Origin '*': Autorise toutes les origines (pas très sécurisé, mais pratique pour le développement). • Access-Control-Allow-Methods: Liste les méthodes HTTP autorisées. Access-Control-Allow-Headers: Liste les en-têtes HTTP autorisés dans les requêtes. Résumé de nginx.conf: • Configure Nginx pour: Servir index.html depuis /usr/share/nginx/html. • Rediriger les requêtes /api/foyer/ vers le backend (service_backend:8089). • Ajouter des en-têtes CORS pour permettre la communication frontend-backend. · Activer les journaux pour le débogage.

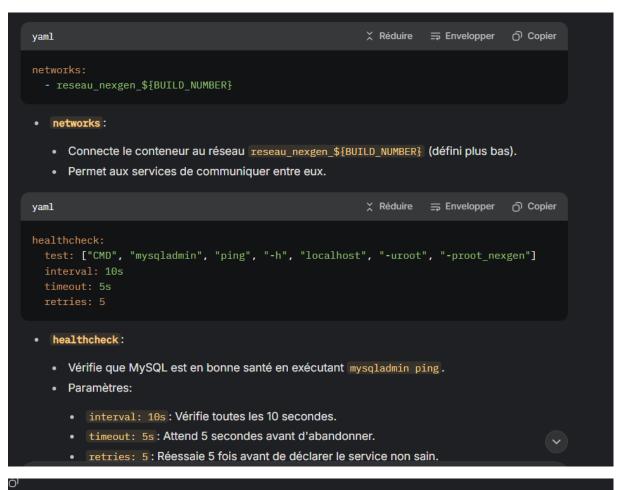
4. docker-compose.yml Ce fichier définit comment les services (base de données, backend, frontend) sont configurés et interconnectés dans un environnement Docker. yaml x Réduire Envelopper Copier version: '3.8': Spécifie la version du format de fichier Docker Compose (3.8 est compatible avec les fonctionnalités utilisées ici). yaml x Réduire Envelopper Copier

services : Liste les services (conteneurs) à lancer.



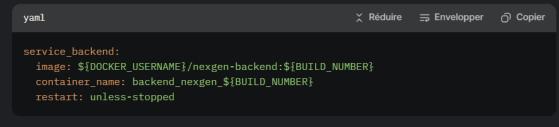




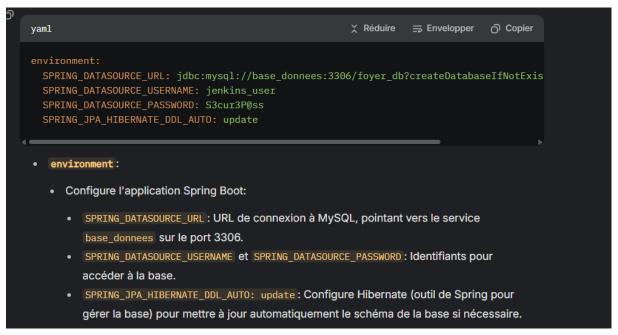


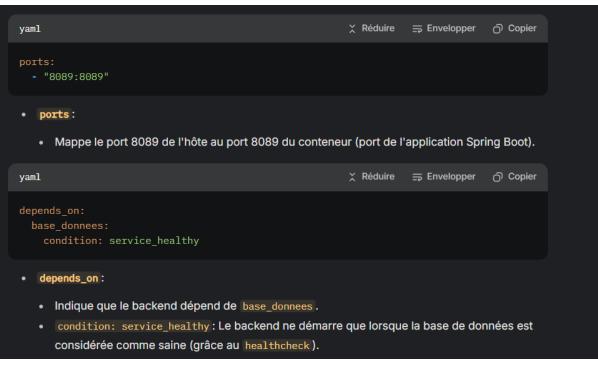
• Utilisé par le backend pour attendre que la base soit prête.

Service: service_backend



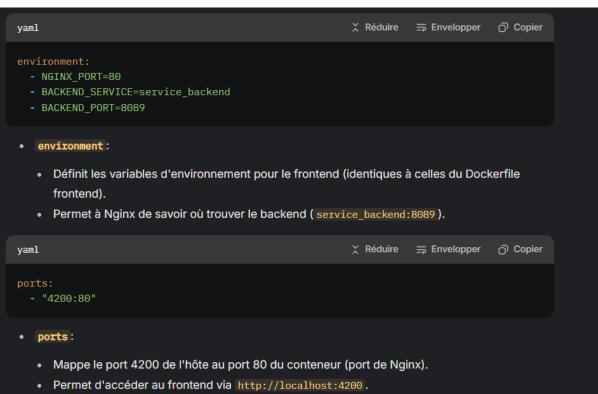
- image: \${DOCKER_USERNAME}/nexgen-backend:\${BUILD_NUMBER}:
 - Utilise une image personnalisée (construite à partir du Dockerfile backend) stockée dans un registre Docker.
 - \${DOCKER_USERNAME} et \${BUILD_NUMBER} sont des variables pour identifier l'image.
- container_name: Nomme le conteneur.
- restart: unless-stopped: Redémarre automatiquement sauf si arrêté manuellement.















Résumé de docker-compose.yml:

- Configure trois services:
 - base_donnees : Une instance MySQL avec une base foyer_db et des données persistantes.
 - service_backend : Une application Spring Boot connectée à MySQL, exposée sur le port 8089.
 - service_frontend: Un serveur Nginx servant une page HTML, connecté au backend, exposé sur le port 4200.
- Les services sont interconnectés via un réseau Docker et dépendent les uns des autres (base
 → backend → frontend).

Comment tout cela fonctionne ensemble

1. Base de données:

 MySQL démarre en premier, crée la base foyer_db, et devient "saine" après le healthcheck.

2. Backend:

- L'application Spring Boot se connecte à MySQL via jdbc:mysql://base_donnees:3306/foyer_db.
- Elle expose une API REST sur le port 8089 (par exemple, /foyer/foyer/findAll).

3. Frontend:

- Nginx sert index.html sur le port 80 (mappé à 4200 sur l'hôte).
- La page HTML utilise JavaScript pour faire des requêtes à /api/foyer/ (par exemple, /api/foyer/findAll).
- Nginx redirige ces requêtes vers http://service_backend:8089/foyer/foyer/.

4. Réseau:

• Tous les services communiquent via le réseau reseau_nexgen_\${BUILD_NUMBER}, utilisant les noms de service (base_donnees, service_backend, service_frontend).

Pipeline:

Vue d'ensemble de la pipeline

La pipeline est écrite en **Groovy** (le langage utilisé par Jenkins pour les pipelines). Elle définit un flux de travail automatisé pour :

- 1. Configurer l'environnement.
- 2. Cloner le code source depuis GitHub.
- 3. Compiler et tester l'application avec Maven.
- 4. Analyser la qualité du code avec SonarQube.
- 5. Déployer les artefacts vers Nexus.
- 6. Construire et publier des images Docker pour le frontend et le backend.
- 7. Déployer l'application avec Docker Compose.
- 8. Vérifier le déploiement.
- 9. Nettoyer l'environnement après exécution.

Elle inclut également des notifications par e-mail et un nettoyage final pour éviter l'accumulation de ressources inutilisées.

```
groovy

    X Réduire
    X Rédui
                                                                                                                                                                                                                                                      ○ Copier

    □ Envelopper

environment {
            GITHUB REPO = 'https://github.com/Houssine2001/NexGen Coders 4TWIN6 2425.git'
            MAVEN_REPO = "${env.WORKSPACE}/.m2/repository"
            DB_HOST = '172.27.222.141'
            MAVEN OPTS = "-Dmaven.repo.local=${MAVEN REPO}"
            PROJECT = 'NexGen_Coders_4TWIN6_2425'
            TEAM = '4TWIN6'
            GROUP = 'NexGen_Coders'
            DELIVERABLE = "${TEAM}-${GROUP}-${PROJECT}"
            DOCKER HUB REPO = "bahadridi2001/nexgen-foyer"
            DOCKER IMAGE TAG = "${env.BUILD NUMBER}"
            DOCKER_USERNAME = "bahadridi2001"
}
 • GITHUB_REPO: URL du dépôt GitHub contenant le code source.
 • MAVEN_REPO : Chemin du répertoire local pour stocker les dépendances Maven
         (${env.WORKSPACE} est l'espace de travail Jenkins).

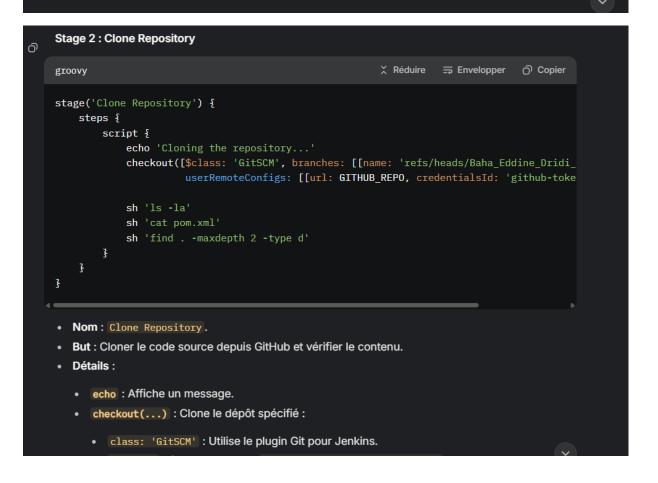
    DB_HOST: Adresse IP de l'hôte de la base de données MySQL (172.27.222.141).

 • MAVEN_OPTS: Options Maven pour utiliser le répertoire local défini (-Dmaven.repo.local).
 • PROJECT, TEAM, GROUP: Métadonnées du projet pour identifier l'application.
 • DELIVERABLE: Nom combiné pour l'artefact (4TWIN6-NexGen_Coders-
         NexGen_Coders_4TWIN6_2425).
```

- DOCKER_HUB_REPO: Nom du dépôt Docker Hub (bahadridi2001/nexgen-foyer).
- DOCKER_IMAGE_TAG: Tag des images Docker, basé sur le numéro de build Jenkins (\${env.BUILD_NUMBER}}).
- DOCKER_USERNAME: Nom d'utilisateur Docker Hub.

Pourquoi ? Ces variables centralisent les configurations, évitant de répéter les mêmes valeurs dans plusieurs étapes.

Stage 1: Environment Setup X Réduire ☐ Envelopper ☐ Copier groovy stage('Environment Setup') { steps { script { echo 'Checking Maven and Java versions...' sh 'mvn -version' sh 'java -version' sh 'docker --version' 3 3 } Nom: Environment Setup. But : Vérifier que les outils nécessaires (Maven, Java, Docker) sont installés sur l'agent Jenkins. Détails : • echo : Affiche un message dans la console Jenkins. • sh 'mvn -version' : Exécute la commande shell pour afficher la version de Maven. sh 'java -version' : Vérifie la version de Java. sh 'docker --version' : Vérifie la version de Docker. • Pourquoi ? Confirme que l'environnement est prêt avant de commencer.



- branches : Clone la branche Baha_Eddine_Dridi_Gestion_Foyer.
- userRemoteConfigs: Utilise l'URL du dépôt (GITHUB_REPO) et les identifiants (githubtoken, stockés dans Jenkins).
- sh 'ls -la': Liste les fichiers dans l'espace de travail pour vérification.
- sh 'cat pom.xml' : Affiche le contenu du fichier pom.xml (fichier Maven).
- sh 'find . -maxdepth 2 -type d' : Liste les répertoires jusqu'à une profondeur de 2 pour vérifier la structure.
- Pourquoi ? Assure que le code source est correctement récupéré et accessible.

- Nom: Maven Build.
- But: Compiler et installer l'application avec Maven.
- Détails :
 - echo : Affiche un message.
 - sh """..."" : Exécute une commande Maven :
 - mvn clean install: Supprime les fichiers temporaires (clean) et construit/installer
 l'application dans le dépôt local (install).
 - -Dspring.datasource.url : Configure l'URL de la base de données MySQL (jdbc:mysql://\${DB_HOST}:3306/foyer).
 - -Dspring.datasource.username : Utilisateur de la base (jenkins_user).
 - -Dspring.datasource.password : Mot de passe (S3cur3P@ss).
- Pourquoi ? Compile le backend Spring Boot et le prépare pour les étapes suivantes.

Stage 4: Run Tests X Réduire ⇒ Envelopper → Copier groovy stage('Run Tests') { steps { script { echo 'Running tests using Maven...' sh """ -Dspring.datasource.username=jenkins_user \\ -Dspring.datasource.password=S3cur3P@ss 3 • Nom: Run Tests. • But : Exécuter les tests unitaires de l'application. Détails : • echo : Affiche un message. • sh """..."" : Exécute mvn test avec les mêmes paramètres de connexion à la base que l'étape précédente.

Pourquoi ? Vérifie que l'application fonctionne correctement avant de continuer.

```
Stage 5 : SonarQube Analysis
                                                       groovy
stage('SonarQube Analysis') {
   steps {
        withCredentials([string(credentialsId: 'sonarqube-token', variable: 'SONAR_TOKEN
           sh """
           mvn sonar:sonar \\
            -Dsonar.host.url=http://172.27.222.141:9000 \\
            -Dsonar.login=${SONAR_TOKEN} \\
            -Dsonar.projectKey=${PR0JECT} \\
            -Dsonar.projectName="${GROUP}-${PROJECT}"
       }
   }
3
• Nom: SonarQube Analysis.
· But : Analyser la qualité du code avec SonarQube.

    Détails :

   • withCredentials : Récupère le token SonarQube (sonarqube-token) et le stocke dans
      SONAR_TOKEN.
   • sh """..."" : Exécute mvn sonar:sonar :
     • -Dsonar.host.url: URL du serveur SonarQube (http://172.27.222.141:9000).
      • -Dsonar.login : Token d'authentification.
         -Dsonar.projectKey: Identifiant unique du projet (${PROJECT}).
        -Dsonar.projectName: Nom affiché dans SonarQube (${GROUP}-${PROJECT}).
• Pourquoi ? Vérifie les bugs, la couverture des tests, et la qualité générale du code.
```

Stage 6 : Package Project X Réduire □ Envelopper □ Copier groovy stage('Package Project') { steps { script { echo 'Packaging the project...' mvn clean package -Ddeliverable.name=\${DELIVERABLE} 3 3 } • Nom: Package Project. • But : Générer un fichier JAR pour l'application. Détails : • echo : Affiche un message. • sh """..."" : Exécute mvn clean package : • clean: Supprime les fichiers temporaires. • package : Crée un fichier JAR dans le dossier target . • -Ddeliverable.name : Définit un nom pour l'artefact (\${DELIVERABLE}}). • Pourquoi ? Prépare l'application pour le déploiement.

```
groovy
                                                        X Réduire
                                                                  Copier
stage('Prepare Maven Settings for Nexus') {
   steps {
        withCredentials([usernamePassword(credentialsId: 'nexus-credentials',
                                          usernameVariable: 'NEXUS_USERNAME',
                                          passwordVariable: 'NEXUS_PASSWORD')]) {
            writeFile file: '.m2/settings.xml', text: """
            <settings>
                <servers>
                    <server>
                        <id>snapshotRepo</id>
                        <username>${NEXUS_USERNAME}</username>
                        <password>${NEXUS_PASSWORD}</password>
                </servers>
            </settings>
        }
}
  Nom: Prepare Maven Settings for Nexus.
  But : Configurer Maven pour se connecter au dépôt Nexus.
  Détails :
   • withCredentials : Récupère les identifiants Nexus (nexus-credentials) dans
      NEXUS_USERNAME et NEXUS_PASSWORD.
   • writeFile : Crée un fichier settings.xml dans .m2 avec :
      • Un serveur nommé snapshotRepo.
      · Les identifiants pour s'authentifier auprès de Nexus.
  Pourquoi ? Permet à Maven de déployer des artefacts vers Nexus.
```

Stage 8: Deploy to Nexus

- Nom: Deploy to Nexus.
- . But: Publier l'artefact JAR sur Nexus.
- Détails :
 - timeout : Limite l'exécution à 10 minutes.
 - sh """..."" : Exécute mvn deploy :
 - -DskipTests : Ignore les tests.
 - Configure la connexion à la base de données (comme dans les étapes précédentes).
 - --settings .m2/settings.xml : Utilise la configuration Nexus créée.
- Pourquoi ? Stocke l'artefact dans un dépôt centralisé pour une réutilisation future.

Stage 9: Construction Images Docker X Réduire □ Envelopper □ Copier stage('Construction Images Docker') { steps { withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]) { script { sh 'cat pom.xml' writeFile file: 'Dockerfile.backend', text: ''' FROM maven:3.8-openjdk-17 AS build WORKDIR /app COPY pom.xml . RUN mvn dependency:go-offline COPY src ./src COPY .env . RUN mvn clean package -DskipTests FROM openjdk:17 WORKDIR /app COPY --from=build /app/target/*.jar app.jar COPY .env . EXPOSE 8089 ENTRYPOINT ["java", "-jar", "app.jar"] writeFile file: 'Dockerfile.frontend', text: ''' FROM nginx:alpine WORKDIR /usr/share/nginx/html COPY nginx.conf /etc/nginx/templates/default.conf.template

RUN echo '<html>...</html>' > index.html

ENV NGINX PORT=80

```
writeFile file: 'nginx.conf', text: '''
        server {
            listen ${NGINX_PORT};
            server_name localhost;
            root /usr/share/nginx/html;
            index index.html;
            access_log /var/log/nginx/access.log;
            error_log /var/log/nginx/error.log debug;
                try_files $uri $uri/ /index.html;
            location /api/foyer {
                proxy_pass http://${BACKEND_SERVICE}:${BACKEND_PORT}/foyer/foyer
        sh """
            echo "=== Building backend image ==="
            docker build --network host -t ${DOCKER_USERNAME}/nexgen-backend:${B
            echo "=== Building frontend image ==="
            docker build --network host -t ${DOCKER_USERNAME}/nexgen-frontend:${
3
```

- Nom: Construction Images Docker.
- But: Construire les images Docker pour le backend et le frontend.
- Détails :
 - withCredentials : Récupère les identifiants Docker Hub.
 - sh 'cat pom.xml' : Affiche pom.xml pour vérification.
 - writeFile 'Dockerfile.backend' : Crée un Dockerfile pour le backend :
 - Étape 1: Compile avec Maven (mvn clean package).
 - Étape 2 : Exécute le JAR avec Java 17 sur le port 8089.
 - writeFile 'Dockerfile.frontend' : Crée un Dockerfile pour le frontend :
 - Utilise Nginx pour servir une page HTML.
 - Configure des variables d'environnement pour communiquer avec le backend.
 - writeFile 'nginx.conf' : Configure Nginx pour rediriger les requêtes /api/foyer vers le backend.
 - sh """docker build ..."" : Construit les images :
 - Backend: \${DOCKER_USERNAME}/nexgen-backend:\${BUILD_NUMBER}.
 - Frontend: \${DOCKER_USERNAME}/nexgen-frontend:\${BUILD_NUMBER}.
 - --network host : Utilise le réseau de l'hôte pour éviter des problèmes de résolution DNS.
- Pourquoi ? Crée des images prêtes à être déployées.

```
X Réduire
groovy

    □ Envelopper    □ Copier
stage('Publication Images Docker') {
    steps {
        withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials',
                        usernameVariable: 'DOCKER_USERNAME',
                        passwordVariable: 'DOCKER_PASSWORD')]) {
            sh """
            echo "${DOCKER_PASSWORD}" | docker login -u "${DOCKER_USERNAME}" --password-
            docker push ${DOCKER_USERNAME}/nexgen-backend:${BUILD_NUMBER}
            docker push ${DOCKER_USERNAME}/nexgen-frontend:${BUILD_NUMBER}
            docker tag ${DOCKER_USERNAME}/nexgen-backend:${BUILD_NUMBER} ${DOCKER_USERNA
            docker tag ${DOCKER_USERNAME}/nexgen-frontend:${BUILD_NUMBER} ${DOCKER_USERN
            docker push ${DOCKER_USERNAME}/nexgen-backend:latest
            docker push ${DOCKER_USERNAME}/nexgen-frontend:latest
        }
    3
3
• Nom: Publication Images Docker.
· But: Publier les images sur Docker Hub.
· Détails :
   • withCredentials : Récupère les identifiants Docker Hub.
   • sh """...":
      • docker login : Se connecte à Docker Hub.

    docker push: Publie les images avec le tag ${BUILD_NUMBER}.

      · docker tag : Crée des tags latest pour les images.
         docker push: Publie les images avec le tag latest.
  Pourquoi ? Rend les images accessibles pour le déploiement.
```

Stage 11: Déploiement Application X Réduire □ Envelopper □ Copier groovy stage('Déploiement Application') { steps { withCredentials([usernamePassword(credentialsId: 'dockerhub-credentials', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]) { script { writeFile file: 'docker-compose.yml', text: """ version: '3.8' base_donnees: ... service_backend: ... service_frontend: ... networks: ... volumes: ... sh """ lsof -ti:80 | xargs -r kill -9 || true lsof -ti:86 | xargs -r kill -9 || true lsof -ti:89 | xargs -r kill -9 || true OLD_CONTAINERS=\$(docker ps -q -f publish=80 -f publish=86 -f publish=89) if [! -z "\$OLD_CONTAINERS"]; then docker stop \$OLD_CONTAINERS || true docker rm \$0LD_CONTAINERS || true docker rm -f frontend backend mysql || true docker rm -f bdd_nexgen_\${BUILD_NUMBER} backend_nexgen_\${BUILD_NUMBER} f docker network rm reseau_nexgen_\${BUILD_NUMBER} || true docker-compose -p nexgen_\${BUILD_NUMBER} up -d docker port frontend_nexgen_\${BUILD_NUMBER}

```
docker rm $OLD_CONTAINERS || true
                docker rm -f frontend backend mysql || true
                docker rm -f bdd_nexgen_${BUILD_NUMBER} backend_nexgen_${BUILD_NUMBER} f
                docker network rm reseau_nexgen_${BUILD_NUMBER} || true
                docker-compose -p nexgen_${BUILD_NUMBER} up -d
                docker port frontend_nexgen_${BUILD_NUMBER}
                docker port backend_nexgen_${BUILD_NUMBER}
                sleep 5
                docker ps

    Nom: Déploiement Application.

• But : Déployer l'application avec Docker Compose.

    Détails :

   • writeFile 'docker-compose.yml' : Crée un fichier docker-compose.yml :

    base_donnees: MySQL avec la base foyer_db.

    service_backend : Backend Spring Boot connecté à MySQL.

    service_frontend : Frontend Nginx connecté au backend.

      • Configure un réseau et un volume pour persister les données.
```

```
sh """..."" :
lsof -ti:80 | xargs -r kill -9 : Tue les processus utilisant les ports 80, 86, ou 89.
docker stop/rm : Supprime les anciens conteneurs.
docker network rm : Supprime le réseau existant.
docker-compose -p nexgen_${BUILD_NUMBER} up -d : Lance les services en mode détaché.
docker port : Affiche les mappings de ports.
sleep 5 : Attend 5 secondes.
docker ps : Liste les conteneurs actifs.

Pourquoi ? Déploie l'application complète localement.
```

```
Stage 12 : Vérification Déploiement
                                                          X Réduire 

□ Envelopper 
□ Copier
groovy
stage('Vérification Déploiement') {
    steps {
        script {
            sh """
            sleep 60
            docker ps | grep nexgen_${BUILD_NUMBER}
            docker logs frontend_nexgen_${BUILD_NUMBER} 2>&1 | tail -20
            docker logs backend_nexgen_${BUILD_NUMBER} 2>&1 | tail -20
            docker port frontend_nexgen_${BUILD_NUMBER}
            docker port backend_nexgen_${BUILD_NUMBER}
            curl -I http://localhost:8089/foyer/foyer/findAll || echo "Backend non acces
            curl -I http://localhost:4200/api/foyer/findAll || echo "API via Frontend no
3

    Nom: Vérification Déploiement.

    But : Vérifier que l'application est correctement déployée.

  Détails :
   • sleep 60 : Attend 60 secondes pour que les services démarrent.
   • docker ps | grep ... : Liste les conteneurs liés au build.
   • docker logs ... | tail -20 : Affiche les 20 dernières lignes des logs du frontend et du
      backend.

    docker port : Vérifie les ports.

  • curl -I : Teste les endpoints :
```

- curl -I : Teste les endpoints :
 - http://localhost:4200 : Page d'accueil du frontend.
 - http://localhost:8089/foyer/foyer/findAll:API du backend.
 - http://localhost:4200/api/foyer/findAll: API via le frontend.

Pourquoi ? Confirme que l'application fonctionne.

```
always {
   script {
        try {
            def namespace = "myapp-${env.BUILD_NUMBER}"
            sh """
            cd ${env.WORKSPACE}
            if [ -f docker-compose.yml ]; then
                docker-compose -p ${namespace} down -v || true
            fi
            docker rm -f mysql-db-${env.BUILD_NUMBER} backend-service-${env.BUILD_NU
            docker network rm ${namespace}_app-network || true
            docker volume rm ${namespace}_mysql-data || true
            docker system prune -f --volumes || true
        } catch (Exception e) {
            echo "Warning: Docker cleanup failed: ${e.message}"
        cleanWs()
    }
3
```

Always:

- · Exécute un nettoyage :
 - docker-compose down: Arrête et supprime les services.
 - docker rm : Supprime les conteneurs.
 - docker network/volume rm : Supprime le réseau et le volume.
 - docker system prune : Nettoie les ressources inutilisées.
 - cleanWs(): Supprime l'espace de travail Jenkins.
- try/catch : Gère les erreurs de nettoyage sans faire échouer la pipeline.
- Pourquoi ? Notifie l'équipe et maintient un environnement propre.

Comment tout cela fonctionne ensemble

- 1. Environnement: Vérifie les outils (Maven, Java, Docker).
- 2. Code: Clone le dépôt GitHub.
- 3. Build/Tests: Compile et teste l'application Spring Boot avec Maven.
- 4. Qualité: Analyse le code avec SonarQube.
- 5. Artefact : Package et déploie le JAR sur Nexus.
- 6. Images Docker: Construit et publie les images frontend/backend sur Docker Hub.
- 7. **Déploiement**: Lance l'application (MySQL, backend, frontend) avec Docker Compose.
- 8. Vérification: Teste les endpoints pour confirmer le fonctionnement.
- 9. Post-actions: Envoie des e-mails et nettoie l'environnement.

Nexus: Un serveur de gestion de dépôts qui stocke des artefacts comme des JARs ou des packages, permettant leur partage et leur gestion centralisée dans les projets.

SonarQube : Un outil d'analyse continue qui inspecte le code source pour détecter les bugs, les vulnérabilités et mesurer la couverture des tests.

Docker Hub : Une plateforme cloud qui héberge des images Docker publiques ou privées, facilitant leur distribution et leur réutilisation.

Docker Compose: Un utilitaire qui utilise un fichier YAML pour configurer et lancer plusieurs conteneurs Docker comme une application cohérente.

Tests unitaires JUnit : Framework Java pour écrire et exécuter des tests automatisés vérifiant le bon fonctionnement d'unités de code individuelles.

Mock : Technique utilisant des objets simulés (mocks) pour imiter le comportement de dépendances externes, isolant ainsi le code testé.

Mock : Créer un objet fictif qui imite le comportement d'une dépendance réelle (comme une base de données ou une API) pour tester une partie du code isolément, sans utiliser la vraie dépendance.

Exemple simple: Si ton code appelle une API, un mock simule les réponses de l'API (par exemple, un JSON prédéfini) pour tester ton code sans faire de vraies requêtes réseau. Cela rend les tests plus rapides et fiables.

FoyerRepositoryTest (Test unitaire Junit)

1. FoyerRepositoryTest

Cette classe teste les méthodes du repository FoyerRepository (une interface Spring Data JPA pour interagir avec la base de données). Elle utilise **JUnit** et **Spring Boot** avec l'annotation @DataJpaTest pour tester les interactions avec la base de données en mémoire (H2 par défaut).

```
@ExtendWith(SpringExtension.class)
@DataJpaTest
class FoyerRepositoryTest {
    @Autowired
    private FoyerRepository foyerRepository;
    @Autowired
    private UniversiteRepository universiteRepository;
    @Autowired
    private BlocRepository blocRepository;
    @Autowired
    private ChambreRepository chambreRepository;
    private Foyer foyer1, foyer2;
    private Foyer foyer;
    private Bloc bloc;
    private Chambre chambre;
}
```

Annotations:

- @ExtendWith(SpringExtension.class) : Intègre JUnit 5 avec Spring pour permettre l'injection de dépendances (@Autowired).
- @DataJpaTest : Configure un environnement de test pour JPA (persistance des données), utilisant une base de données en mémoire (H2) et charge uniquement les composants JPA (repositories).

Attributs :

- @Autowired : Injecte les repositories (FoyerRepository, UniversiteRepository, etc.) pour interagir avec la base.
- foyer1, foyer2, foyer, bloc, chambre: Objets utilisés pour configurer les données de test.

```
Méthode setUp
                                                     java
@BeforeEach
void setUp() {
    foyer1 = new Foyer();
    foyer1.setNomFoyer("Foyer Alpha");
    foyer1.setCapaciteFoyer(200);
    foyer2 = new Foyer();
    foyer2.setNomFoyer("Foyer Beta");
    foyer2.setCapaciteFoyer(100);
    foyerRepository.save(foyer1);
    foyerRepository.save(foyer2);
    foyer = new Foyer();
    foyer.setNomFoyer("Foyer Test");
    foyer.setCapaciteFoyer(300);
    foyer = foyerRepository.save(foyer);
    bloc = new Bloc();
    bloc.setNomBloc("Bloc A");
    bloc.setFoyer(foyer);
    bloc = blocRepository.save(bloc);
    chambre = new Chambre();
    chambre.setTypeC(TypeChambre.SIMPLE);
    chambre.setBloc(bloc);
    chambre = chambreRepository.save(chambre);
}
```

- But : Initialiser les données avant chaque test.
- · Explication:
 - Crée deux foyers (foyer1 : "Foyer Alpha", capacité 200 ; foyer2 : "Foyer Beta", capacité
 100) et les sauvegarde dans la base avec foyerRepository.save().
 - Crée un autre foyer (foyer : "Foyer Test", capacité 300) et le sauvegarde.
 - Crée un bloc ("Bloc A") lié à foyer et le sauvegarde.
 - Crée une chambre de type SIMPLE liée à bloc et la sauvegarde.
- Pourquoi ?:
 - @BeforeEach garantit que chaque test commence avec une base de données propre et des données prédéfinies.
 - Les relations (foyer → bloc → chambre) sont configurées pour tester des requêtes complexes.

Test 1: testFindByNomFoyer iava @Test void testFindByNomFoyer() { Foyer found = foyerRepository.findByNomFoyer("Foyer Alpha"); assertNotNull(found); assertEquals("Foyer Alpha", found.getNomFoyer()); } • But: Tester la méthode findByNomFoyer du repository. Explication : Appelle foyerRepository.findByNomFoyer("Foyer Alpha") pour chercher un foyer par son assertNotNull(found) : Vérifie que le foyer existe. assertEquals("Foyer Alpha", found.getNomFoyer()): Confirme que le nom du foyer trouvé est correct. Pourquoi?: Valide que Spring Data JPA génère correctement une requête pour chercher un foyer par foyer1 ("Foyer Alpha") a été sauvegardé dans setUp, donc on s'attend à le trouver.

Test 2: testFindByCapaciteFoyerGreaterThan java void testFindByCapaciteFoyerGreaterThan() { List<Foyer> foyers = foyerRepository.findByCapaciteFoyerGreaterThan(150); assertEquals(2, foyers.size()); assertEquals("Foyer Alpha", foyers.get(0).getNomFoyer()); } • But: Tester la méthode findByCapaciteFoyerGreaterThan. · Explication: • Cherche les foyers avec une capacité > 150. • assertEquals(2, foyers.size()): Vérifie que deux foyers sont trouvés (foyer1:200, foyer: 300). assertEquals("Foyer Alpha", foyers.get(0).getNomFoyer()): Vérifie que le premier foyer est "Foyer Alpha". Pourquoi ? : · Confirme que la requête JPA filtre correctement les foyers par capacité. • L'ordre des résultats peut dépendre de l'implémentation, mais ici, on assume que "Foyer Alpha" est le premier.

```
Test 3: testFindByCapaciteFoyerLessThan
                                                       java
@Test
void testFindByCapaciteFoyerLessThan() {
   List<Foyer> foyers = foyerRepository.findByCapaciteFoyerLessThan(150);
   assertEquals(1, foyers.size());
   assertEquals("Foyer Beta", foyers.get(0).getNomFoyer());\\
}
• But: Tester la méthode findByCapaciteFoyerLessThan.
· Explication:
   • Cherche les foyers avec une capacité < 150.
   • assertEquals(1, foyers.size()) : Un seul foyer doit être trouvé (foyer2 : 100).

    assertEquals("Foyer Beta", foyers.get(θ).getNomFoyer()): Vérifie que c'est "Foyer

     Beta".
Pourquoi?:
   · Valide la précision des requêtes JPA pour les comparaisons inférieures.
```

```
○ Copier
                                                          X Réduire

    □ Envelopper
@Test
void testFindByUniversiteNomUniversite() {
    Universite universite = new Universite();
    universite.setNomUniversite("Université de Test");
    universite = universiteRepository.save(universite);
    assertTrue(universite.getIdUniversite() > 0, "L'université n'a pas été sauvegardée c
    Foyer foyer11 = new Foyer();
    foyer11.setNomFoyer("Foyer Alpha");
    universite.setFoyer(foyer11);
    foyer11.setUniversite(universite);
    foyerRepository.save(foyer11);
    Foyer found = foyerRepository.findByUniversiteNomUniversite("Université de Test");
    assertNotNull(found, "Aucun foyer trouvé pour l'université donnée");
    assertEquals("Foyer Alpha", found.getNomFoyer());
3
• But: Tester la méthode findByUniversiteNomUniversite.
  Explication:
   • Crée et sauvegarde une université ("Université de Test").
   • assertTrue(universite.getIdUniversite() > 0) : Vérifie que l'université a un ID valide.
   • Crée un foyer ("Foyer Alpha") et l'associe à l'université (relation bidirectionnelle).
   · Sauvegarde le foyer.
   · Cherche le foyer lié à "Université de Test".

    assertNotNull(found): Vérifie que le foyer existe.

    assertEquals("Foyer Alpha", found.getNomFoyer()): Confirme le nom du foyer.

  Pourquoi?:
```

Teste une requête JPA complexe impliquant une relation entre Foyer et Universite.

Test 6: testGetByBlocsChambresTypeC java **@Test** void testGetByBlocsChambresTypeC() { List<Foyer> foyers = foyerRepository.getByBlocsChambresTypeC(TypeChambre.SIMPLE); assertFalse(foyers.isEmpty(), "Aucun foyer trouvé pour ce type de chambre."); assertEquals("Foyer Test", foyers.get(0).getNomFoyer()); • But: Tester la méthode getByBlocsChambresTypeC. · Explication: • Cherche les foyers ayant des chambres de type SIMPLE via leurs blocs. • assertFalse(foyers.isEmpty()) : Vérifie qu'au moins un foyer est trouvé. assertEquals("Foyer Test", foyers.get(θ).getNomFoyer()): Confirme que c'est "Foyer Test". • Pourquoi?: Valide une requête JPA traversant plusieurs relations (Foyer → Bloc → Chambre). • setUp a créé une chambre SIMPLE dans "Foyer Test", donc on s'attend à ce résultat.

```
Test 7: testFindByBlocNom
                                                          X Réduire 

□ Envelopper 
□ Copier
java
@Test
void testFindByBlocNom() {
    List<Foyer> foyers = foyerRepository.find("Bloc A");
    assertFalse(foyers.isEmpty(), "Aucun foyer trouvé pour ce bloc.");
    assertEquals("Foyer Test", foyers.get(0).getNomFoyer());
• But: Tester la méthode find (probablement findByBlocsNomBloc).
Explication :
   · Cherche les foyers liés à un bloc nommé "Bloc A".
   • assertFalse(foyers.isEmpty()) : Vérifie qu'un foyer est trouvé.
   • assertEquals("Foyer Test", foyers.get(0).getNomFoyer()) : Confirme que c'est "Foyer
      Test".
  Pourquoi?:
   · Teste une requête JPA basée sur le nom du bloc.

    setUp a créé "Bloc A" dans "Foyer Test", donc le résultat est attendu.

   • Note: La méthode find("Bloc A") semble inhabituelle pour Spring Data JPA; il est
      possible qu'elle soit une méthode personnalisée ou que son nom réel soit
      findByBlocsNomBloc.
```

```
@ExtendWith(MockitoExtension.class)
class FoyerRepoTest {
    @Mock
    private FoyerRepository foyerRepository;
    @Mock
    private UniversiteRepository universiteRepository;
    @Mock
    private BlocRepository blocRepository;
    @InjectMocks
    private FoyerService foyerService;
}
```

Annotations :

• @ExtendWith(MockitoExtension.class) : Intègre JUnit 5 avec Mockito pour gérer les mocks.

• Attributs :

- @Mock : Crée des mocks pour les repositories, simulant leur comportement sans accéder à une vraie base de données.
- @InjectMocks : Injecte les mocks dans FoyerService pour que ses dépendances soient simulées.

• Pourquoi?:

 Les mocks permettent de tester FoyerService isolément, sans dépendre d'une base de données réelle.

```
@Test
                                                          X Réduire
                                                                     Copie
void testAddOrUpdate() {
    Foyer foyer = new Foyer();
    foyer.setIdFoyer(1L);
    foyer.setNomFoyer("Foyer Test");
    when(foyerRepository.save(foyer)).thenReturn(foyer);
    Foyer result = foyerService.addOrUpdate(foyer);
    assertNotNull(result);
    assertEquals(foyer.getIdFoyer(), result.getIdFoyer());
    assertEquals(foyer.getNomFoyer(), result.getNomFoyer());
    verify(foyerRepository, times(1)).save(foyer);
}
   But : Tester la méthode add0rUpdate du service.

    Explication :

    Crée un foyer avec ID 1 et nom "Foyer Test".

    when(foyerRepository.save(foyer)).thenReturn(foyer) : Configure le mock pour

      retourner le même foyer quand save est appelé.

    Appelle foyerService.addOrUpdate(foyer).

    assertNotNull(result) : Vérifie que le résultat n'est pas null.

    assertEquals(...): Confirme que l'ID et le nom du foyer retourné sont corrects.

    verify(foyerRepository, times(1)).save(foyer): Vérifie que save a été appelé

      exactement une fois.
```

- Pourquoi ? :
 - Valide que add0xUpdate sauvegarde correctement un foyer via le repository.
 - Mockito simule la sauvegarde, isolant le test de la base de données.

Test 2: testFindAll X Réduire ≡ Envelopper ⊙ Copier java @Test void testFindAll() { List<Foyer> foyers = new ArrayList<>(); foyers.add(new Foyer()); foyers.add(new Foyer()); when(foyerRepository.findAll()).thenReturn(foyers); List<Foyer> result = foyerService.findAll(); assertEquals(2, result.size()); verify(foyerRepository, times(1)).findAll(); • But : Tester la méthode findAll. Explication : · Crée une liste de deux foyers fictifs. when(foyerRepository.findAll()).thenReturn(foyers): Configure le mock pour retourner cette liste. • Appelle foyerService.findAll(). • assertEquals(2, result.size()) : Vérifie que deux foyers sont retournés. • verify(foyerRepository, times(1)).findAll():Confirme que findAll a été appelé une • Pourquoi?:

• Teste que le service récupère tous les foyers correctement.

```
Test 3: testFindById_Success
                                                           java
@Test
void testFindById_Success() {
    Foyer foyer = new Foyer();
    foyer.setIdFoyer(1L);
    \label{lem:when for the continuous} when (foyer Repository.find By Id(1L)).then Return (Optional.of(foyer));
    Foyer result = foyerService.findById(1L);
    assertNotNull(result);
    assertEquals(1L, result.getIdFoyer());
3
  But: Tester findById quand le foyer existe.

    Explication :

   • Crée un foyer avec ID 1.

    when(foyerRepository.findById(1L)).thenReturn(Optional.of(foyer)):Le mock

      retourne un Optional contenant le foyer.
   • Appelle foyerService.findById(1L).
   • assertNotNull(result) : Vérifie que le résultat existe.

    assertEquals(1L, result.getIdFoyer()): Confirme I'ID.

Pourquoi?:
   · Valide que le service retourne un foyer existant.
```

Test 4: testFindById_NotFound ○ Copier java @Test void testFindById_NotFound() { when(foyerRepository.findById(1L)).thenReturn(Optional.empty()); assertThrows(FoyerNotFoundException.class, () -> foyerService.findById(1L)); 3 But: Tester findById quand le foyer n'existe pas. Explication : when(foyerRepository.findById(1L)).thenReturn(Optional.empty()): Le mock retourne un Optional vide. assertThrows(FoyerNotFoundException.class, ...): Vérifie que foyerService.findById(1L) lance une FoyerNotFoundException. Pourquoi?: · Teste la gestion des erreurs quand un foyer est introuvable.

```
Test 5: testDeleteById
                                                          X Réduire 

☐ Envelopper ☐ Copier
java
@Test
void testDeleteById() {
    doNothing().when(foyerRepository).deleteById(1L);
    foyerService.deleteById(1L);
    verify(foyerRepository, times(1)).deleteById(1L);
3
• But: Tester deleteById.

    Explication :

   • doNothing().when(foyerRepository).deleteById(1L): Configure le mock pour ne rien
      faire lors de la suppression.
   • Appelle foyerService.deleteById(1L).
   • verify(foyerRepository, times(1)).deleteById(1L): Vérifie que la méthode a été
      appelée une fois.
Pourquoi ? :
   • Valide que le service appelle correctement la suppression par ID.
```

Test 6: testDelete X Réduire ☐ Envelopper ☐ Copier java void testDelete() { Foyer foyer = new Foyer(); doNothing().when(foyerRepository).delete(foyer); foyerService.delete(foyer); verify(foyerRepository, times(1)).delete(foyer); } • But: Tester delete (par objet). • Explication: · Crée un foyer fictif. • doNothing().when(foyerRepository).delete(foyer) : Le mock simule la suppression. • Appelle foyerService.delete(foyer). • verify(...) : Confirme que delete a été appelé. • Pourquoi?: • Similaire à deleteById , mais teste la suppression par instance.

```
@Test
                                                          X Réduire

    ○ Copier

void testAffecterFoyerAUniversite() {
    Foyer foyer = new Foyer();
    foyer.setIdFoyer(1L);
    Universite universite = new Universite();
    universite.setNomUniversite("Test Universite");
    when(foyerRepository.findById(1L)).thenReturn(Optional.of(foyer));
    when(universiteRepository.findByNomUniversite("Test Universite")).thenReturn(universite")
    when(universiteRepository.save(universite)).thenReturn(universite);
    Universite result = foyerService.affecterFoyerAUniversite(1L, "Test Universite");
    assertNotNull(result);
    assertEquals(foyer, result.getFoyer());
3
• But : Tester l'affectation d'un foyer à une université par nom.
 Explication :
   • Crée un foyer (ID 1) et une université ("Test Universite").
   · Configure les mocks pour simuler :

    Recherche du foyer par ID.

      • Recherche de l'université par nom.

    Sauvegarde de l'université.

    Appelle foyerService.affecterFoyerAUniversite(1L, "Test Universite").

   · Vérifie que l'université retournée est liée au foyer.
  Pourquoi?:
```

Teste une opération métier complexe impliquant deux repositories.

```
java
                                                                                 ○ Copier
@Test
void testAjouterFoyerEtAffecterAUniversite() {
    Foyer foyer = new Foyer();
    foyer.setBlocs(new ArrayList<>());
    Universite universite = new Universite();
    universite.setIdUniversite(1L);
    when(foyerRepository.save(foyer)).thenReturn(foyer);
    when(universiteRepository.findById(1L)).thenReturn(Optional.of(universite));
    when(universiteRepository.save(universite)).thenReturn(universite);
    Foyer result = foyerService.ajouterFoyerEtAffecterAUniversite(foyer, 1L);
    assertNotNull(result);
    assertEquals(foyer, universite.getFoyer());
3
• But : Tester l'ajout d'un foyer et son affectation à une université.
Explication :
   • Crée un foyer et une université (ID 1).

    Simule la sauvegarde du foyer, la recherche de l'université, et la sauvegarde de l'université.

   • Appelle foyerService.ajouterFoyerEtAffecterAUniversite.
   • Vérifie que le foyer est lié à l'université.
```

• Pourquoi?:

• Teste une opération combinée (sauvegarde + association).

```
X Réduire

    □ Envelopper

                                                                                  ○ Copier
@Test
void testAjoutFoyerEtBlocs() {
    Foyer foyer = new Foyer();
    List<Bloc> blocs = new ArrayList<>();
    Bloc bloc1 = new Bloc();
    Bloc bloc2 = new Bloc();
    blocs.add(bloc1);
    blocs.add(bloc2);
    foyer.setBlocs(blocs);
    when(foyerRepository.save(foyer)).thenReturn(foyer);
    Foyer result = foyerService.ajoutFoyerEtBlocs(foyer);
    assertNotNull(result);
    verify(blocRepository, times(2)).save(any(Bloc.class));
3
• But: Tester l'ajout d'un foyer avec ses blocs.
• Explication:
   • Crée un foyer avec deux blocs.
   · Simule la sauvegarde du foyer.
   • Appelle foyerService.ajoutFoyerEtBlocs.
   • Vérifie que le foyer est retourné et que blocRepository.save a été appelé deux fois.
· Pourquoi?:
   • Valide que les blocs sont sauvegardés avec le foyer.
```

```
Test 10: testAffecterFoyerAUniversiteById
                                                         X Réduire ≡ Envelopper 🗇 Copier
java
void testAffecterFoyerAUniversiteById() {
    Foyer foyer = new Foyer();
    Universite universite = new Universite();
    when(foyerRepository.findById(1L)).thenReturn(Optional.of(foyer));
    when(universiteRepository.findById(2L)).thenReturn(Optional.of(universite));
    when(universiteRepository.save(universite)).thenReturn(universite);
    Universite result = foyerService.affecterFoyerAUniversite(1L, 2L);
    assertNotNull(result);
    assertEquals(foyer, result.getFoyer());
}
• But: Tester l'affectation par IDs.
Explication :
   • Simule la recherche du foyer (ID 1) et de l'université (ID 2), puis la sauvegarde.
   • Appelle foyerService.affecterFoyerAUniversite(1L, 2L).
   • Vérifie que l'université est liée au foyer.
Pourquoi?:
   • Similaire à testAffecterFoyerAUniversite, mais utilise des IDs.
```

Test 11: testDesaffecterFoyerAUniversite

- But : Tester la désaffectation d'un foyer d'une université.
- Explication :
 - · Crée une université avec un foyer.
 - · Simule la recherche et la sauvegarde de l'université.
 - Appelle foyerService.desaffecterFoyerAUniversite(1L).
 - Vérifie que l'université n'a plus de foyer.
- Pourquoi?:
 - Valide que la relation est correctement supprimée.

Différence entre les deux classes

- FoyerRepositoryTest:
 - Teste directement les méthodes du repository (FoyerRepository) en interagissant avec une base de données en mémoire.
 - Utilise @DataJpaTest pour simuler un environnement JPA.
 - Vérifie les requêtes JPA générées automatiquement (comme findByNomFoyer) ou personnalisées.
 - Pas de mocks, car les tests dépendent des vraies implémentations des repositories.
- FoyerRepoTest:
 - Teste la logique métier dans FoyerService, qui utilise plusieurs repositories.
 - Utilise Mockito pour simuler les repositories, isolant le service de la base de données.
 - Vérifie les interactions entre le service et les repositories (via verify) et les résultats des méthodes.
 - Idéal pour tester des opérations complexes sans dépendre d'une base réelle.

Résumé des concepts

- JUnit:
 - Fournit @Test pour définir des tests.
 - @BeforeEach pour initialiser les données.
 - Assertions (assertEquals, assertNotNull, assertThrows) pour valider les résultats.
- Mockito (dans FoyerRepoTest):
 - @Mock crée des objets simulés (repositories).
 - @InjectMocks injecte les mocks dans le service.
 - when(...).thenReturn(...) définit le comportement des mocks.
 - verify(...) vérifie les appels aux mocks.
 - Permet de tester le service sans base de données.