République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Rapport README de Projet
Module : **DevOps**

Industrialisation DevOps d'une application Web existante

Projet de Fin de Module – DevOps

Réalisé par :
— Islem Mbarki
— Rahma Ben Omrane
— Hazem Ben Saria
— Aziza Fguir
— Nour El houda Hmani
— Yasmine belkhiria
Encadrant : Dr Marouen kachroudi

Année universitaire : **2025 – 2026**

Table des matières

1 Présentation du projet

Ce projet consiste à concevoir et développer une application web moderne permettant la gestion des réservations et des prêts de ressources au sein d'un réseau de bibliothèques. La plateforme vise à améliorer l'expérience des usagers tout en offrant aux bibliothécaires et administrateurs des outils efficaces de gestion, de suivi et de contrôle.

2 Objectifs du projet

— Digitaliser et automatiser la gestion des prêts et réservations

— Simplifier l'accès aux ressources pour les usagers

— Mettre en place un système de notifications automatiques par email

— Assurer la sécurité des accès à l'aide d'une gestion par rôles

— Garantir la qualité du système par des tests et une documentation claire

3 Architecture technique

Couche	Technologies utilisées
Backend	Spring Boot, Spring MVC
Sécurité	Spring Security, JWT, BCrypt
Persistance	Spring Data JPA
Base de données	MySQL (production) / H2 (tests)
Frontend	Thymeleaf
Email	Spring Mail
Tests	JUnit, Spring Boot Test
Build	Maven
Versionnement	Git / GitHub

4 Répartition des tâches (travail en groupe)

4.1 Hazem Ben Saria — Authentification et gestion des utilisateurs

— Conception et implémentation des entités `User` et `Role`

— Configuration complète de **Spring Security**

— Mise en place des fonctionnalités de connexion (Login) et d'inscription (Register)

— Chiffrement des mots de passe à l'aide de **BCrypt**

— Gestion des restrictions d'accès selon les rôles

— Développement des pages Thymeleaf : login, register et profil

— Gestion du profil utilisateur (consultation des prêts et réservations)

4.2 Rahma Ben Omrane — Catalogue des ressources

— Conception de l'entité `Ressource` (livres et documents)

— Implémentation d'un CRUD complet accessible aux administrateurs et bibliothécaires

— Gestion de l'upload des images de couverture via `MultipartFile`

— Mise en place des fonctionnalités de recherche, filtrage, pagination et tri

— Développement des interfaces Thymeleaf : liste, détail, ajout et modification

— Gestion des bibliothèques et de leurs ressources associées

4.3 Aziza Fguir — Workflow des prêts et réservations

— Conception de l'entité `Loan`

— Implémentation du cycle de vie des prêts :

Réservé → Emprunté → Retourné

— Implémentation des règles métier :

  — Vérification de la disponibilité des ressources

  — Limitation du nombre de prêts par utilisateur

— Développement des services métier pour la gestion du workflow

— Interfaces Thymeleaf : réservation d'un exemplaire, liste des prêts et validation par le bibliothécaire

4.4 Islem Mbarki — Notifications, dashboard et tests

— Configuration du service d'envoi d'emails avec **Spring Mail**

— Envoi automatique des notifications :

  — Confirmation de réservation

  — Rappel de retour en cas de retard

— Développement d'un mini tableau de bord statistique

— Mise en place des tests unitaires et d'intégration

— Rédaction de la documentation technique et du fichier README

5 Tests et Qualité

Plusieurs types de tests ont été réalisés afin d'assurer la robustesse et la fiabilité de l'application. Les tests sont realiser avec JUunit.

5.1 Types de tests

— **Tests unitaires** : validation de la logique métier de base

— **Tests de persistance** : vérification des repositories avec H2

— **Tests d'intégration** : chargement du contexte Spring et des contrôleurs REST

— **Tests de sécurité** : validation de l'intégration Spring Security et JWT

5.2 Exécution des tests

```
./mvnw test
```

Tous les tests ont été exécutés avec succès, garantissant la cohérence globale du système.

6 Lancement de l'application

6.1 Prérequis

— Java 17+

— Maven

— MySQL ou H2

6.2 Commandes

```
mvn clean install
mvn spring-boot:run
```

Accès local : `http://localhost:8082`

7 Instructions pour les DevOps :

7.1 Conteneurisation de l'application

Dans ce projet, l'ensemble du backend, incluant l'API REST et le frontend web intégré, a été conteneurisé avec Docker. La base de données MySQL est également exécutée dans un conteneur séparé. Cette approche garantit l'isolation des services, la portabilité et la reproductibilité de l'environnement applicatif.

7.1.1 Fichier Dockerfile pour le backend Spring Boot

Le backend Spring Boot est packagé dans un fichier JAR et exécuté dans un conteneur Java 17. Le `Dockerfile` définit les étapes suivantes :

— sélection de l'image de base Java 17 (`eclipse-temurin:17-jre`),

— définition du répertoire de travail (`/app`),

— copie du fichier JAR généré par Maven dans le conteneur,

— exposition du port 8080 utilisé par Spring Boot,

— et commande de démarrage de l'application avec `java -jar`.

Le backend contient donc à la fois la logique serveur et le frontend web intégré (Spring MVC / Thymeleaf), ce qui permet d'accéder aux pages et aux API via le même conteneur.

7.1.2 Docker Compose pour orchestrer les services

L'orchestration des conteneurs est assurée par Docker Compose, défini dans le fichier `docker-compose.yml`. Deux services principaux sont déclarés :

— **mysql** : conteneur exécutant la base de données MySQL. Les variables d'environnement permettent de créer automatiquement la base de données, un utilisateur dédié et un mot de passe.

— **app** : conteneur exécutant le backend Spring Boot (API + frontend). Il dépend du service MySQL et est configuré via des variables d'environnement pour se connecter à la base.

Un réseau Docker de type `bridge` est utilisé pour permettre la communication interne sécurisée entre les conteneurs. Les données MySQL sont persistées via un volume Docker, assurant la conservation des informations même après l'arrêt ou la suppression des conteneurs.

7.2 Déploiement Continu (Continuous Deployment – CD)

1. Objectif du déploiement continu Le déploiement continu vise à automatiser la mise en production de l'application après validation complète des étapes d'intégration continue. Dans ce projet, le CD permet de garantir que chaque version validée sur la branche `main` est automatiquement déployée sur l'environnement Azure.

2. Choix de la solution de déploiement Pour le déploiement, nous avons utilisé les services cloud Microsoft Azure, notamment :

— Azure Container Registry (ACR) pour le stockage des images Docker ;

— Azure Container Apps pour l'exécution et l'hébergement de l'application.

Ce choix permet :

— une gestion simplifiée des conteneurs ;

— une intégration native avec GitHub Actions ;

— un déploiement scalable et sécurisé.

3. Azure Container Registry (ACR) Un registre de conteneurs Azure a été créé afin de stocker les images Docker générées par le pipeline CI/CD.

Caractéristiques :

— registre privé ;

— authentification sécurisée via les secrets GitHub ;

— gestion de plusieurs versions d'images (tags).

Chaque image est poussée avec :

— le tag `latest` ;

— un tag correspondant au hash du commit Git.

4. Azure Container Apps Azure Container Apps est utilisé pour déployer et exécuter l'image Docker stockée dans ACR.

Fonctionnalités principales :

— exécution de l'application dans un conteneur managé ;

— attribution automatique d'une URL publique ;

— supervision de l'état de l'application (*Running*).

L'application est hébergée dans la région *France Central* et reste accessible tant que le service est actif.

5. Pipeline de Déploiement Continu Le déploiement est géré par GitHub Actions et déclenché uniquement lorsque :

— un *push* ou un *merge* est effectué sur la branche `main` ;

— les étapes CI et le *smoke test* sont validées avec succès.

6. Description du Job de Déploiement Le job de déploiement s'exécute sur une machine Ubuntu, dépend du succès du *smoke test* et est limité à la branche `main`.

```
deploy:
  name: Deploy to Azure
  runs-on: ubuntu-latest
  needs: smoke-test
  if: github.ref == 'refs/heads/main'
```

Le job :

— s'exécute sur une machine Ubuntu ;

— dépend du succès du smoke test ;

— est limité à la branche `main`.

7. Étapes du déploiement Récupération du code

```
- uses: actions/checkout@v4
```

Permet de récupérer le code source du dépôt.

Téléchargement de l'artefact JAR

```
- uses: actions/download-artifact@v4
```

Télécharge le fichier JAR généré lors de la phase CI.

Authentification Azure

```
- uses: azure/login@v2
```

Connexion sécurisée à Azure via un service principal stocké dans les secrets GitHub.

Connexion au registre ACR

```
- uses: docker/login-action@v3
```

Authentification au Azure Container Registry pour permettre le push des images Docker.

Build et Push de l'image Docker

```
docker build -t image:latest -t image:${{ github.sha }} .
docker push image:latest
docker push image:${{ github.sha }}
```

Cette étape :

— construit l'image Docker ;

— applique deux tags (versionnement) ;

— pousse l'image vers Azure Container Registry.

8. Résultat du déploiement À l'issue du pipeline :

— l'image Docker est disponible dans ACR ;

— une nouvelle révision est déployée sur Azure Container Apps ;

— l'application est accessible via une URL Azure.

Le déploiement est automatique, reproductible et sécurisé.

9. Conclusion de la partie CD La mise en place du déploiement continu permet :

— une mise en production rapide ;

— une réduction des erreurs humaines ;

— une meilleure traçabilité des versions ;

— une approche conforme aux pratiques DevOps professionnelles.

7.3 Commandes pour lancer l'application

Le processus de déploiement se fait avec les commandes suivantes :

1. Compilation du backend Spring Boot :

```
mvn clean package
```

2. Construction des images et lancement des conteneurs :

```
docker-compose up --build
```

Le backend Spring Boot est accessible depuis le navigateur ou d'autres clients HTTP via :

`http://localhost:8080`

Cette configuration permet de lancer l'ensemble de l'application (backend + frontend intégré + base de données) avec une seule commande, facilitant le développement, les tests et la démonstration.

8 Conclusion

Ce projet a permis de développer une application web complète, sécurisée et modulaire basée sur Spring Boot. La répartition équilibrée des tâches, l'intégration de mécanismes de sécurité, de notifications et de tests ainsi que la documentation fournie ont contribué à la qualité et à la maintenabilité de la solution. Cette application constitue une base solide et évolutive pour des améliorations futures.