Configuration CI/CD pour MPJFUMS

Ce document explique comment le pipeline d'Intégration Continue et de Déploiement Continu (CI/CD) a été mis en place pour l'application MPJFUMS.

Architecture CI/CD

Nous utilisons GitHub Actions pour l'intégration continue et Railway pour le déploiement continu. Cette architecture permet de :

1. Tester automatiquement le code à chaque push
2. Vérifier la qualité du code
3. Déployer automatiquement les changements validés

Configuration GitHub Actions

Le workflow GitHub Actions est défini dans le fichier `.github/workflows/django-ci.yml` :

```
name: Django CI

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main, develop ]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_USER: postgres
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: test_db
        ports:
          - 5432:5432
        options: --health-cmd pg_isready --health-interval 10s --health-timeout
5s --health-retries 5

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
```

```
          python-version: '3.10'

      - name: Install Dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run Tests
        env:
          DATABASE_URL: postgres://postgres:postgres@localhost:5432/test_db
        run: |
          cd shop
          python manage.py test

  lint:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'

    - name: Install Dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 black
        pip install -r requirements.txt

    - name: Lint with flake8
      run: |
        flake8 shop --count --select=E9,F63,F7,F82 --show-source --statistics

    - name: Check formatting with black
      run: |
        black --check shop
```

Déploiement Continu avec Railway

Railway est configuré pour déployer automatiquement l'application à chaque push sur la branche principale. La configuration est définie dans le fichier `railway.json`.

Configuration des environnements

Nous avons configuré deux environnements sur Railway :

1. **Production** : déploiement automatique depuis la branche `main`
2. **Staging** : déploiement automatique depuis la branche `develop`

Processus de déploiement

Le processus de déploiement se déroule comme suit :

1. Un développeur pousse du code sur GitHub
2. GitHub Actions exécute les tests et les vérifications de qualité du code
3. Si tous les tests passent et que le code est poussé sur une branche de déploiement :
    - Railway détecte automatiquement les changements
    - Railway lance le processus de build
    - L'application est déployée dans l'environnement correspondant

Bonnes pratiques CI/CD implémentées

Tests automatisés

Tous les tests Django sont exécutés à chaque push, garantissant que les nouvelles fonctionnalités ne cassent pas les existantes.

Analyse de code

Le code est automatiquement vérifié par Flake8 pour détecter les erreurs potentielles et par Black pour assurer une mise en forme cohérente.

Déploiement progressif

Les nouvelles fonctionnalités sont d'abord déployées sur l'environnement de staging pour validation avant d'être promues en production.

Surveillance post-déploiement

Après chaque déploiement, Sentry surveille l'application pour détecter les erreurs et les problèmes de performance.

Flux de travail pour les développeurs

1. Créer une branche de fonctionnalité à partir de `develop`
2. Développer et tester localement avec Docker
3. Pousser les changements et créer une pull request vers `develop`
4. GitHub Actions exécute les tests et analyses de code
5. Après révision et validation, fusionner la pull request
6. La branche `develop` est automatiquement déployée sur l'environnement de staging
7. Après validation de staging, créer une pull request de `develop` vers `main`
8. Après révision et validation, fusionner vers `main`
9. L'application est automatiquement déployée en production

Résolution des problèmes CI/CD

Échec des tests

Si les tests échouent dans GitHub Actions :

1. Consulter les logs dans l'onglet "Actions" de GitHub
2. Corriger les problèmes dans l'environnement local
3. Pousser les corrections et vérifier que les tests passent

Échec du déploiement Railway

Si le déploiement échoue sur Railway :

1. Consulter les logs de build dans Railway
2. Vérifier que toutes les dépendances sont correctement spécifiées
3. S'assurer que les variables d'environnement sont correctement configurées