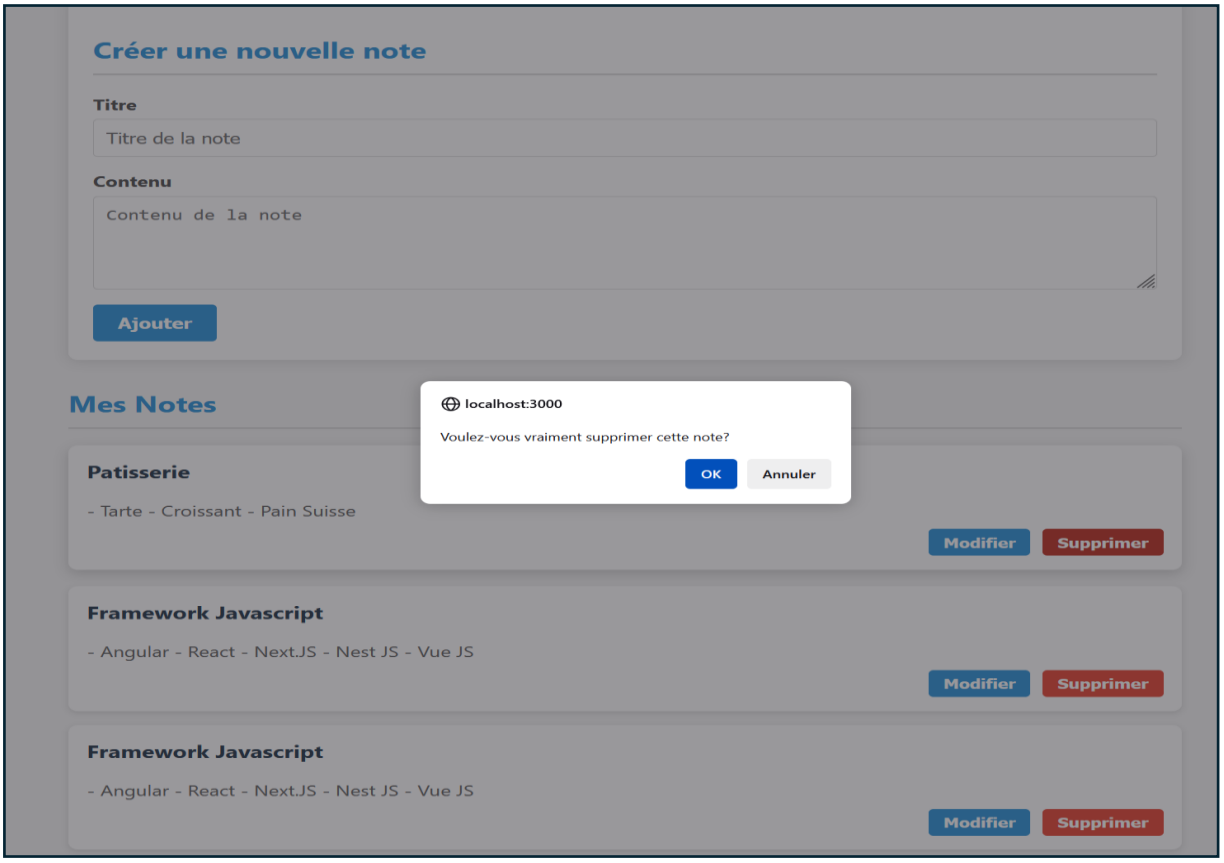
# Rapport de Fonctionnement du Pipeline CI/CD pour Todolist-CICD-App




## 1. Tests automatiques

**Configuration :**

Le pipeline CI/CD exécute automatiquement les **tests unitaires** à chaque **push ou pull request**.

Les tests unitaires sont configurés avec Jest (jest --detectOpenHandles).
Le fichier package.json définit la commande de test: "test": "jest --detectOpenHandles".

Les test sont donc configuré avec Jest et permettent de vérifier :

**Les routes API backend** dans le fichier server.testjs :

**Les fonctionnalités frontend** en Javascript dans le fichier public/script.test.js

**Résultats :**

Grâce à une configuration adaptée, les tests s'exécutent dans des environnements appropriés :

- **Node.js** pour le backend
- **JSDOM** pour le frontend

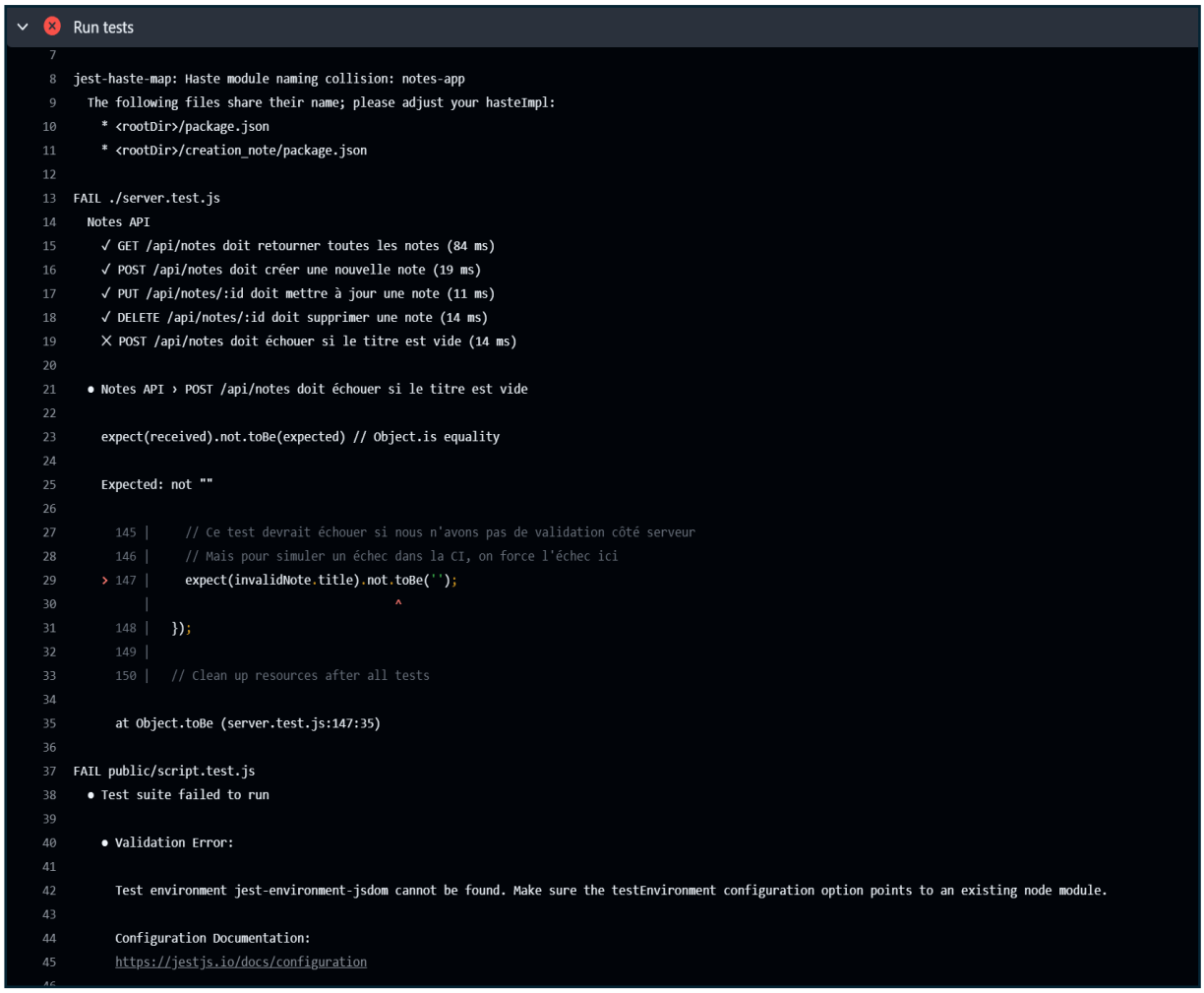
Les principaux scénarios de test Jest incluent :

- La création, récupération, mise à jour et suppression des notes
- **La validation des données** pour éviter les entrées invalides

## 2. CI qui échoue en cas d'erreur

**Configuration :**

Le pipeline GitHub Actions est conçu pour **interrompre l'exécution** si un test échoue.



La configuration est définie dans .github/workflows/main.yml, où le **job test** est exécuté en premier

**Résultats :**

Si un test échoue, l'erreur est détectée immédiatement. Par exemple :
**POST /api/notes doit échouer si le titre est vide.**
**Le pipeline s'arrête automatiquement**, empêchant ainsi le déploiement d'un code défectueux.

Un retour visuel est affiché directement sur **GitHub** avec un indicateur :
✅ **Vert** si les tests réussissent
❌ **Rouge** si un échec est détecté

## 3. Déploiement automatique après merge

**Configuration :**

Le **déploiement est conditionnel** et ne s'exécute que si un **push** est effectué sur main ou master.
Définition dans le workflow YAML :

```yaml
deploy:
  needs: docker-build
  runs-on: ubuntu-latest
  # S'exécute uniquement lors d'un push sur la branche principale (merge) et non lors des pull requests
  if: github.event_name == 'push' && (github.ref == 'refs/heads/main' || github.ref == 'refs/heads/master')
```

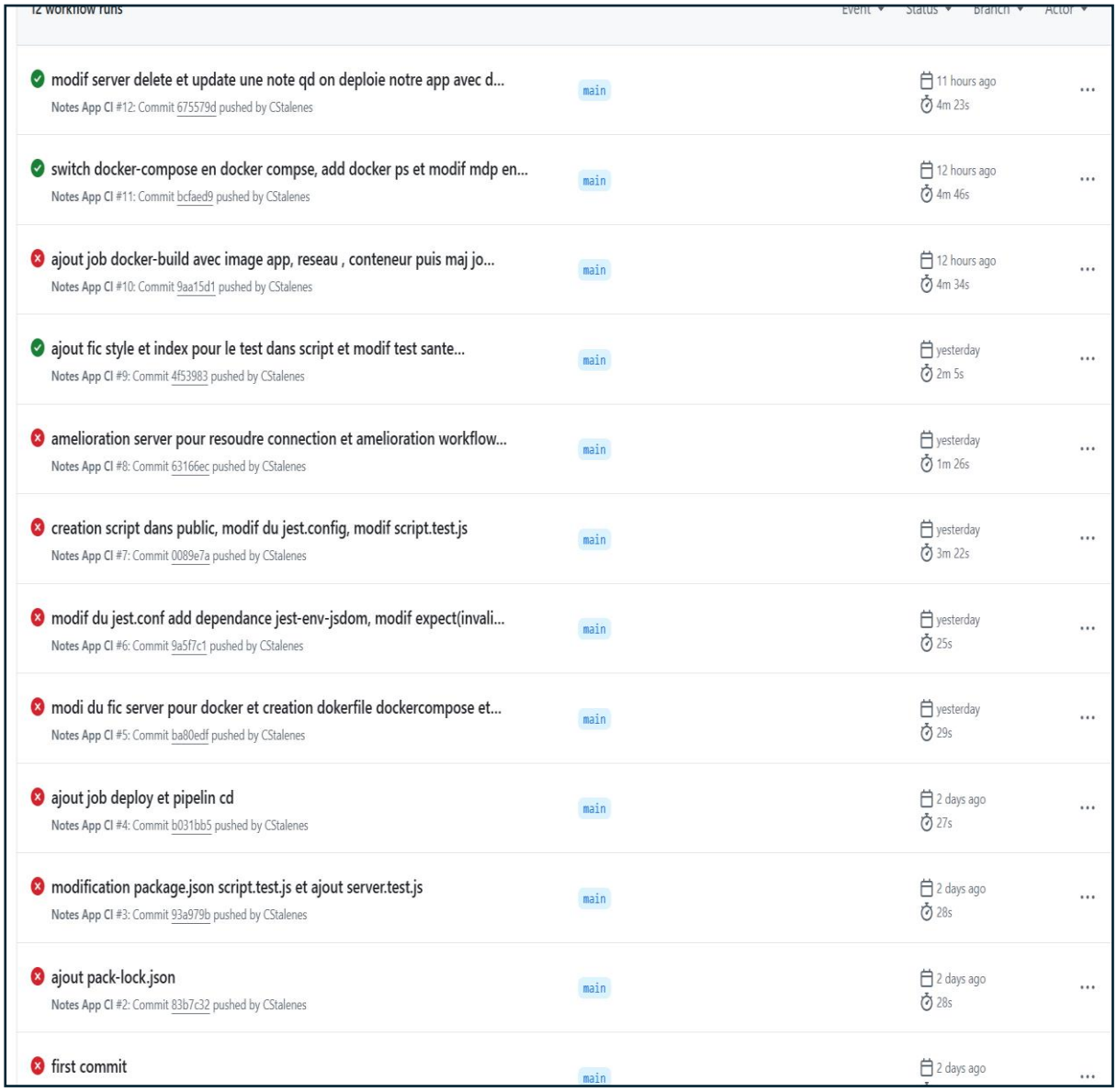
Le processus simulé pour démonstration est appliqué sur le server.js à la racine du projet.

**Résultats :**

Ce processus **ne se déclenche pas sur les pull requests**, garantissant ainsi une stabilité avant mise en production.

La séquence complète du pipeline suit donc ces étapes :
**Tests → Build → Docker Build → Déploiement**

Une notification est générée avec **les informations du dernier commit** et un journal détaillé des étapes exécutées.




# 4. Déploiement via Docker

**Configuration :**

Maintenant que nous avions fini la configuration du pipeline en CI/CD on va utiliser  docker pour déployer notre application avec une image.

On va donc procéder en finalisant la construction de notre fichier Dockerfile pour la construction de l'image de l'application à la racine du projet puis un fichier docker-compose.yml pour orchestrer le lancement des service Node.js et MySQL.

L'application est **conteneurisée avec Docker**, permettant un **déploiement stable et reproductible**.

On a donc :

- Job docker-build qui construit et teste l'image
- Job deploy qui utilise Docker Compose pour le déploiement

**Test d'integration docker :**

✔ Création d'un **réseau Docker dédié**
✔ Lancement des **conteneurs MySQL et Node.js**
✔ Vérification du bon fonctionnement des **routes API**

L'image Docker est correctement construite et donc on va procéder au Déploiement complet via docker compose sur server.js avec commande :

**docker compose up -d**

Les **logs** des conteneurs permettent de vérifier le bon fonctionnement de l'application.

## Conclusion :

Le pipeline CI/CD implémente toutes les étapes demandées et fonctionne correctement :

1. ✅ Tests automatiques avec Jest qui exécute tous les tests unitaires
2. ✅ CI avec détection d'erreur qui implique un échec du pipeline en cas de test non passant
3. ✅ Déploiement automatique après merge déclenché uniquement sur main/master après validation des modification.
4. ✅ Déploiement en production avec Docker pour la construction et déploiement de conteneurs

Cette structure assure ainsi la qualité du code, l'intégrité de l'application et un déploiement sans friction après validation des modifications.