Documentation technique

Configuration CircleCI pour projet Next.js avec TypeScript et Jest

1. **Gestion des branches**
   a. *Structure des branches*

      La configuration respecte le pattern GitFlow :

      - Branches principales :
        - main → Prod
        - integration → Pré-prod/intégration
        - develop → Développement
      - Branches secondaires :
        - feature/* → Développement de nouvelle fonctionnalité
        - hotfix/* → Corrections urgentes de bugs en prod

   b. *Initiatlisation des branches principales*

```
git checkout -b develop
git push -u origin develop

git checkout -b integration
git push -u origin integration
```

   c. *Workflow de travail avec GitFlow*

      a. *Développement d'une fonctionnalité*

```
git checkout -b feature/ma-fonctionnalité
git add .
git commit -m "Decription fonctionnalité"
git push origin feature/ma-fonctionnalité

git checkout develop
git pull
```

b. *Correction d'un bug urgent*

```
git checkout -b hotfix/mon-hotfix
git add .
git commit -m "Description hotfix"
git push origin hotfix/mon-hotfix
```

2. **Description Technique du workflow et des jobs**

a. *Stage "Build"*

Ce stage à pour objectif de préparer le code pour les tests et le déploiement.

Job	Description	Branches	Justification
install_dependencies	Installation des dépendances npm	Toutes	Nécessaire pour tout type de travail sur le code
code_analysis	Analyse statique du code (ESLint, TypeScript)	Toutes	Détection rapide des problèmes de qualité
cleaning_packaging	Nettoyage et préparation des builds	Toutes sauf `feature/*`	Évite des builds inutiles pour les branches temporaires

b. *Stage “Tests”*

Ce stage vérifie la qualité et la fiabilité du code sous différents aspects.

Job	Description	Branches	Justification
unit_tests	Tests unitaires avec Jest	Toutes	Validation basique du code
integration_tests	Tests d'intégration entre composants	Toutes sauf `feature/*`	Vérification des interactions entre composants
regression_tests	Tests de non-régression	`develop`, `integration`, `main`, `hotfix/*`	Vérification que les bugs corrigés ne reviennent pas
performance_tests	Tests de performance (Lighthouse)	`integration`, `main`	Validation des performances avant déploiement en production
security_tests	Tests de sécurité	Toutes sauf `feature/*`	Protection contre les vulnérabilités
compatibility_tests	Tests de compatibilité navigateurs	`integration`, `main`	Validation sur les différents navigateurs
accessibility_tests	Tests d'accessibilité (WCAG)	`integration`, `main`	Conformité aux normes d'accessibilité

c. *Stage “Deploy”*

Ce stage gère la mise en production du code sur les différents environnements.

Job	Description	Branches	Justification

prepare_environment	Préparation de l'environnement cible	`develop`, `integration`, `main`	Configuration spécifique à chaque environnement
deploy_application	Déploiement effectif de l'application	`develop`, `integration`, `main`	Déploiement adapté à l'environnement cible
verification_tests	Tests de vérification post-déploiement	`develop`, `integration`, `main`	Vérification du bon fonctionnement après déploiement
functional_validation	Validation fonctionnelle	`develop`, `integration`, `main`	Tests fonctionnels sur l'environnement réel
load_tests	Tests de charge	`integration`, `main`	Vérification de la tenue en charge avant production
monitoring	Mise en place du monitoring	`develop`, `integration`, `main`	Surveillance des environnements déployés

3. ## Optimisation du workflow

   a. ### *Filtrage par branches*

   Les filtres sont mis en place pour exécuter certains jobs uniquement sur les branches concernées pour éviter d'exécuter des jobs coûteux (comme les tests de performance) sur des branches temporaires ou de développement :

```
filters:
  branches:
    only:
      - main
      - integration
```

b. *Dépendances entre jobs*

Chaque job dépend explicitement d'autres jobs pour garantir que les jobs s'exécutent dans le bon ordre et évite les exécutions inutiles :

```
requires:
  - install_dependencies
```

c. *Mise en cache*

Nous utilisons le cache pour les dépendances Node.js et le cache de Next.js afin d'accélérer considérablement les builds en évitant de réinstaller les dépendances à chaque fois :

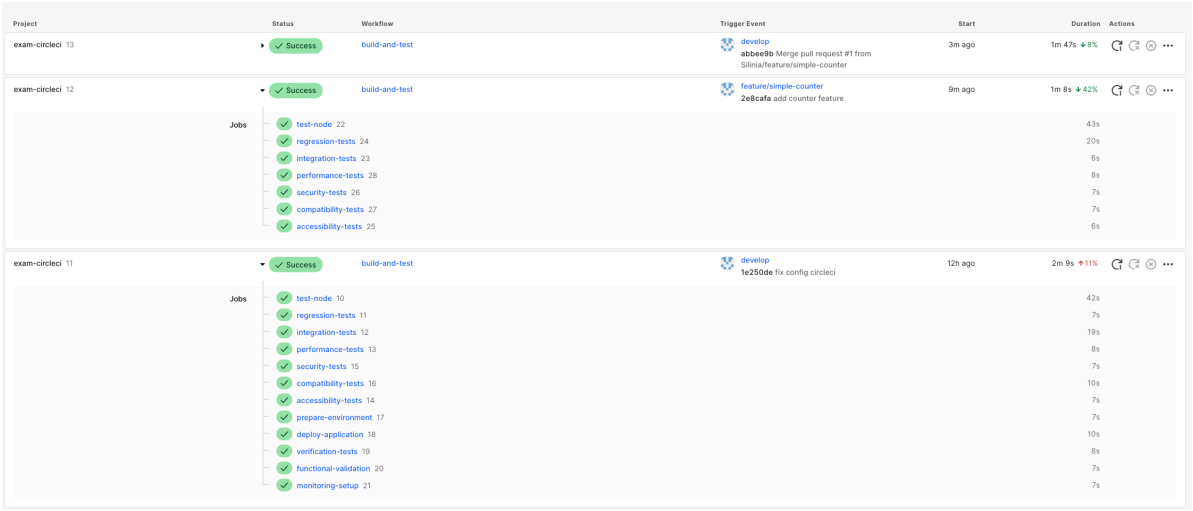
```
save_cache:
  key: node-deps-v1-{{ checksum "package-lock.json" }}
  paths:
    - node_modules
```

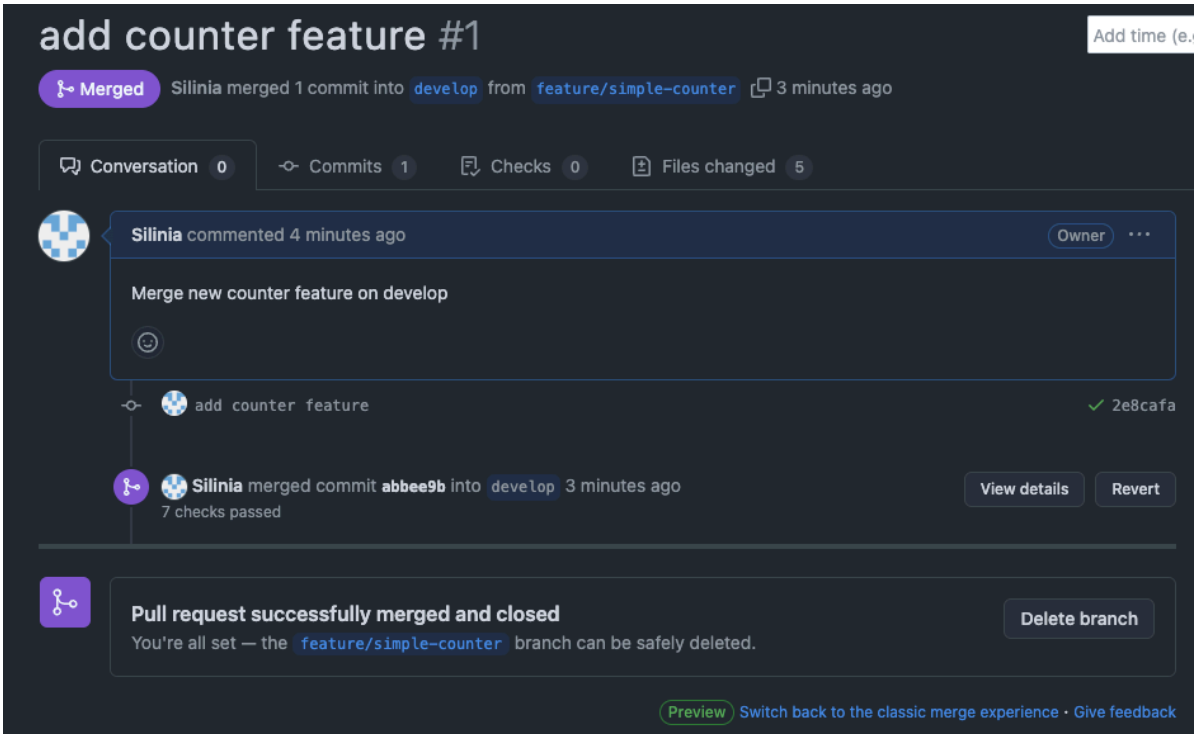
d. *Exécution conditionnelle*

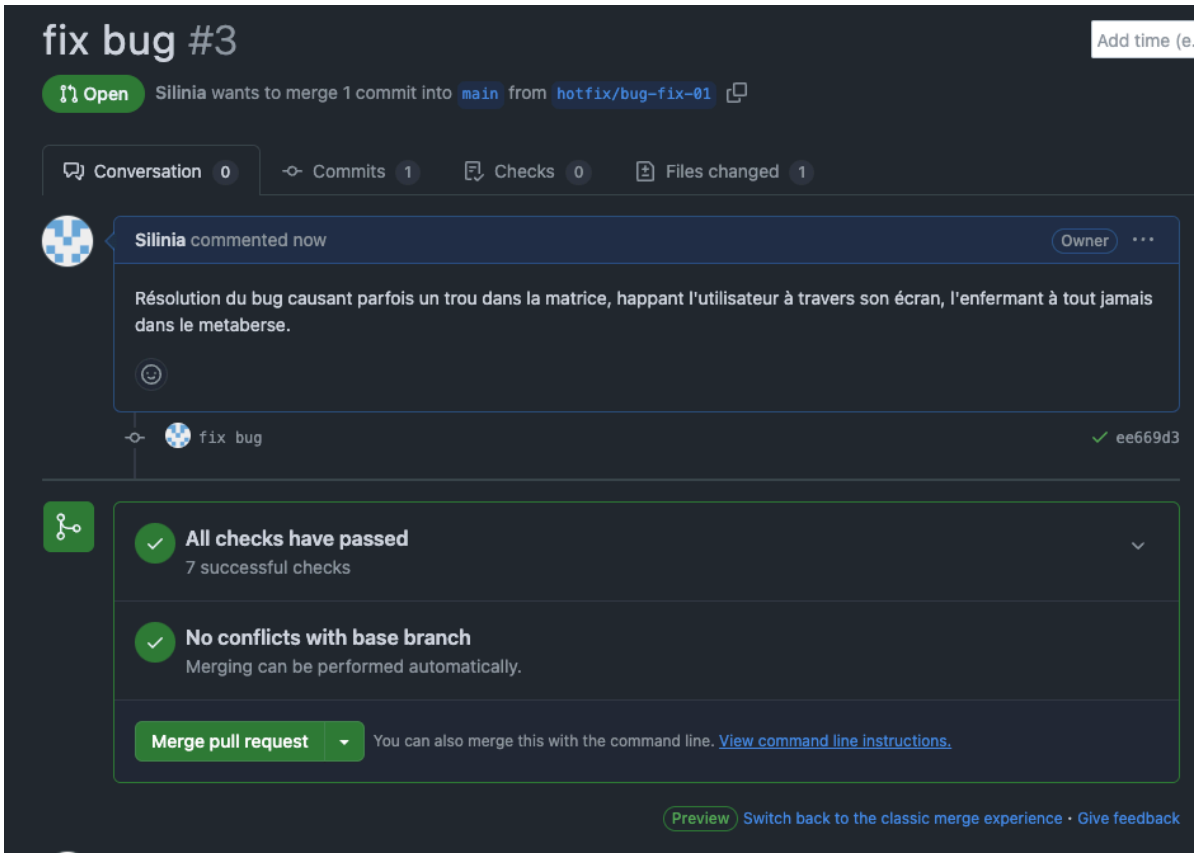
Certains tests plus lourds (compatibilité, performance) ne s'exécutent que sur les branches principales afin de réduire le temps d'exécution global et l'utilisation des ressources :

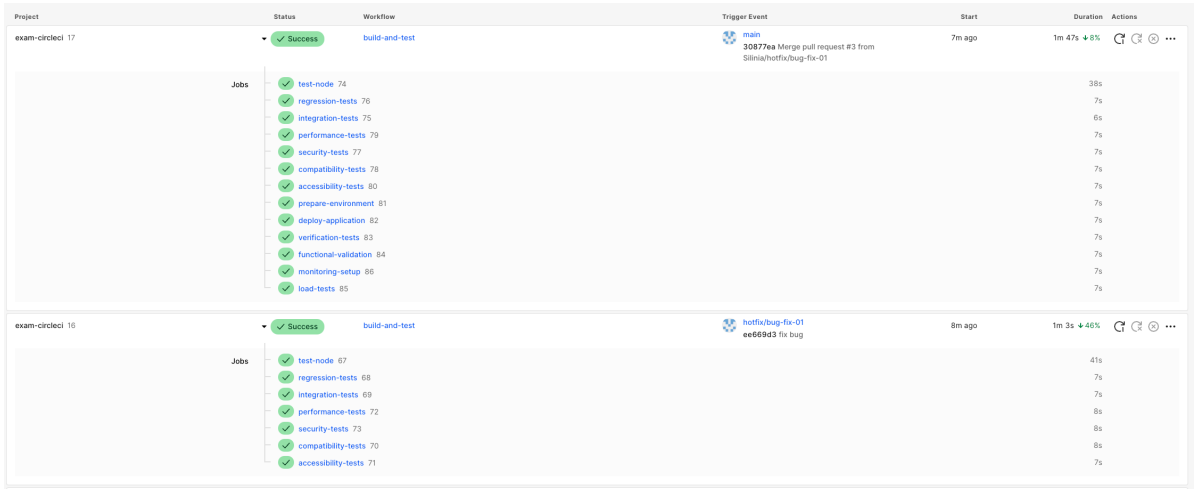
```
performance_tests:
  filters:
    branches:
      only:
        - integration
        - main
```

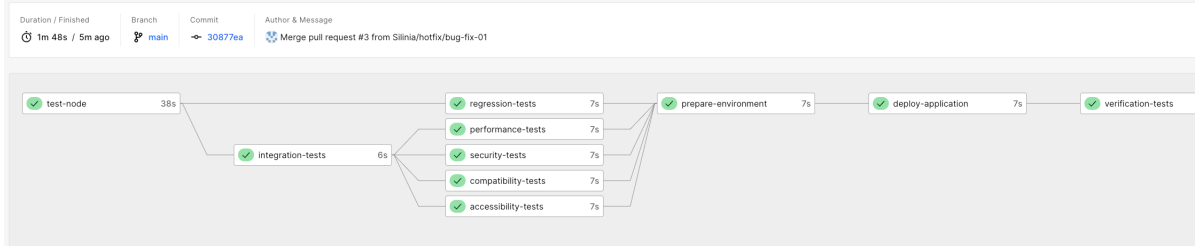
4. Captures d’écran des pipelines






fix bug #3
Add time (e.
Open
Silinia wants to merge 1 commit into main from hotfix/bug-fix-01
Conversation 0
Commits 1
Checks 0
Files changed 1
Silinia commented now
Owner
Résolution du bug causant parfois un trou dans la matrice, happant l'utilisateur à travers son écran, l'enfermant à tout jamais dans le metaberse.
fix bug
ee669d3
All checks have passed
7 successful checks
No conflicts with base branch
Merging can be performed automatically.
Merge pull request
You can also merge this with the command line. View command line instructions.
Preview
Switch back to the classic merge experience
Give feedback


Project
Status
Workflow
Trigger Event
Start
Duration
Actions
exam-circleci 17
Success
build-and-test
main
30877ea Merge pull request #3 from Silinia/hotfix/bug-fix-01
7m ago
1m 47s
Jobs
test-node 74 38s
regression-tests 76 7s
integration-tests 75 6s
performance-tests 79 7s
security-tests 77 7s
compatibility-tests 78 7s
accessibility-tests 80 7s
prepare-environment 81 7s
deploy-application 82 7s
verification-tests 83 7s
functional-validation 84 7s
monitoring-setup 86 7s
load-tests 85 7s
exam-circleci 16
Success
build-and-test
hotfix/bug-fix-01
ee669d3 fix bug
8m ago
1m 3s
Jobs
test-node 67 41s
regression-tests 68 7s
integration-tests 69 7s
performance-tests 72 8s
security-tests 73 8s
compatibility-tests 70 8s
accessibility-tests 71 7s


Duration / Finished
1m 48s / 5m ago
Branch
main
Commit
30877ea
Author & Message
Merge pull request #3 from Silinia/hotfix/bug-fix-01
test-node 38s
integration-tests 6s
regression-tests 7s
performance-tests 7s
security-tests 7s
compatibility-tests 7s
accessibility-tests 7s
prepare-environment 7s
deploy-application 7s
verification-tests

5. **Améliorations**

   a. *Intégration avec Vercel*

```
deploy_vercel:
  steps:
    - run:
        name: "Déploiement sur Vercel"
        command: |
          npx vercel --token ${VERCEL_TOKEN} --prod
```

   b. *Tests automatisés pour les composants Next.js*

```
unit_tests:
  steps:
    - run:
        name: "Exécution des tests unitaires"
        command: npm test
```

6. **Bénéfices**

   a. *Automatisation et qualité*

   - **Tests automatisés** : Tous les tests sont exécutés automatiquement à chaque commit, assurant la qualité du code
   - **Détection précoce des problèmes** : Les erreurs sont identifiées dès le stade de la branche feature
   - **Standardisation** : Tous les développeurs suivent le même workflow, assurant la cohérence du processus

   b. *Optimisation des ressources*

   - **Exécution conditionnelle** : Les jobs lourds ne s'exécutent que sur les branches qui le nécessitent

- **Parallélisation** : Les tests s'exécutent en parallèle quand c'est possible
- **Caching** : Utilisation du cache pour accélérer les builds et réduire les ressources consommées

c. *Traçabilité et visibilité*

- **Historique des exécutions** : Chaque étape du pipeline est documentée et archivée
- **Visibilité des déploiements** : L'équipe sait exactement ce qui est déployé sur chaque environnement
- **Métriques de performance** : Possibilité de suivre l'évolution des temps d'exécution des tests et du pipeline

7. Conclusion

Cette configuration CI/CD est conçue pour soutenir efficacement le développement d'une application Next.js avec TypeScript en suivant le pattern GitFlow. L'implémentation du composant Counter nous a permis de démontrer concrètement le fonctionnement du workflow, depuis la création d'une branche feature jusqu'au déploiement sur l'environnement de développement. L'ensemble des jobs est soigneusement organisé pour fournir un feedback rapide aux développeurs et garantir que seul un code de qualité atteigne les environnements de production. Cette approche contribue à maintenir un niveau élevé de qualité tout en optimisant l'utilisation des ressources.