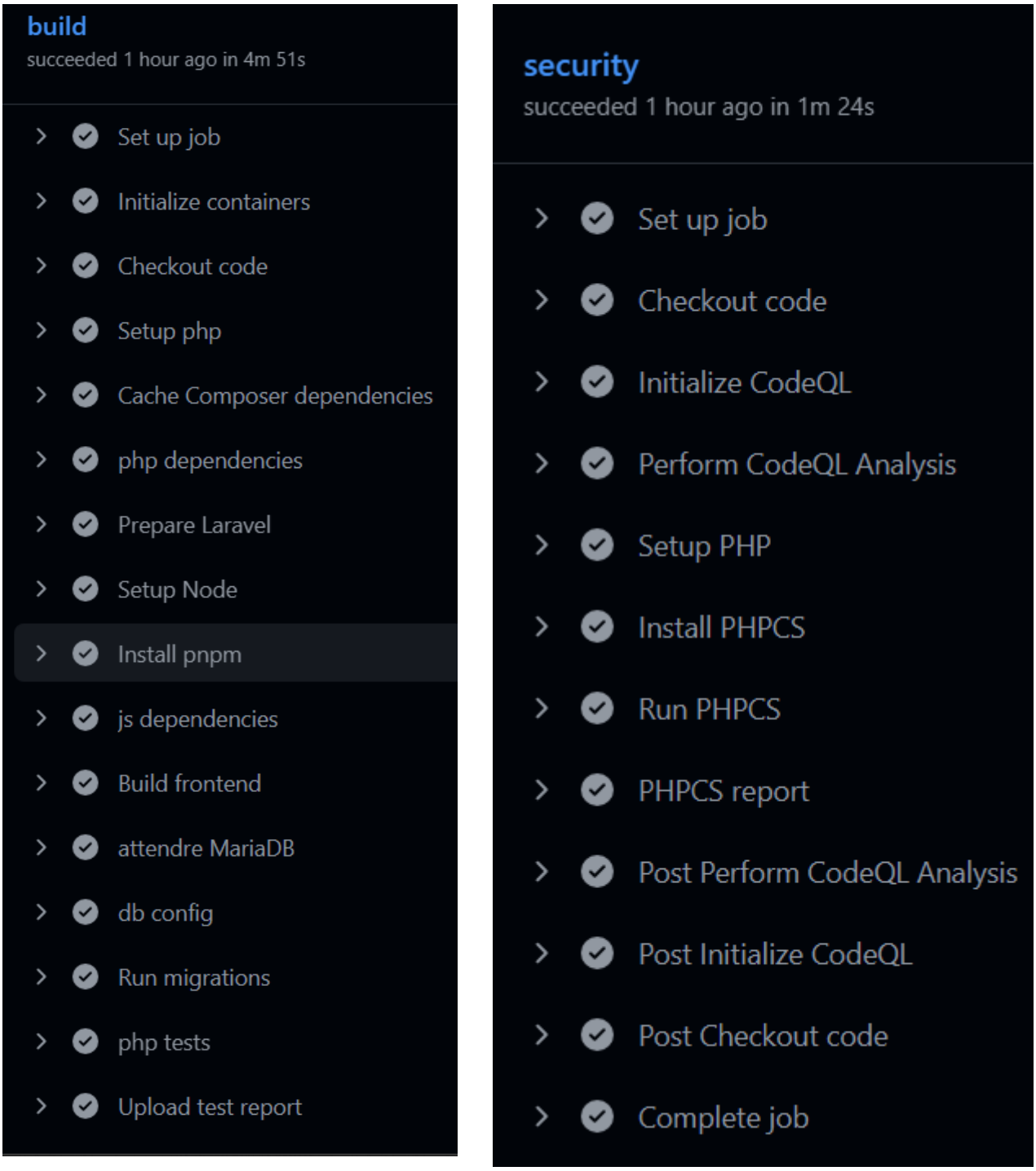
Projet
P_App-CICD

Veronika Skupovska
FID2

Un screenshot de l'exécution du pipeline principal



Description

l'objectif était de mettre en place un pipeline CI/CD intégrant des outils de vérification statique de sécurité afin d'analyser automatiquement le code source lors de chaque modification.

Outils disponibles sur le marché

Outil	Type	Avantages	Inconvénients
Snyk	Analyse de dépendances	Très précis, intégré aux CI	Fonctionnalités clés payantes
SonarQube	Analyse statique	Très complet, règles nombreuses	Version gratuite très limitée
GitHub CodeQL	Analyse statique	Gratuit, intégré à GitHub	Support PHP limité et contraignant
PHP_CodeSniffer (PHPCS)	Qualité / sécurité	Gratuit, simple, local	Analyse syntaxique, pas sémantique

Choix effectués et justification

Les outils sélectionnés doivent etre gratuits, simples à intégrer dans GitHub Actions, et compatibles avec les technologies utilisées (PHP, JavaScript)

Au vu de ces contraintes, deux outils ont été choisis :

1. Code QL - analyse de sécurité du code JavaScript (dans le cadre du projet). permet de détecter automatiquement des vulnérabilités connues et des failles potentielles dans le code. Les résultats de cette analyse sont accessibles dans l'onglet Security / Code Scanning. Mais le support du PHP est incomplet et le dépôt doit être public ou disposer des permissions adaptées pour que l'analyse fonctionne correctement.

2. PHP_CodeSniffer (PHPCS) - analyse statique du code PHP. Détecte les mauvaises pratiques de développement, les violations de standards de codage (PSR-12) et certains comportements à risque. Il est simple à utiliser, gratuit et

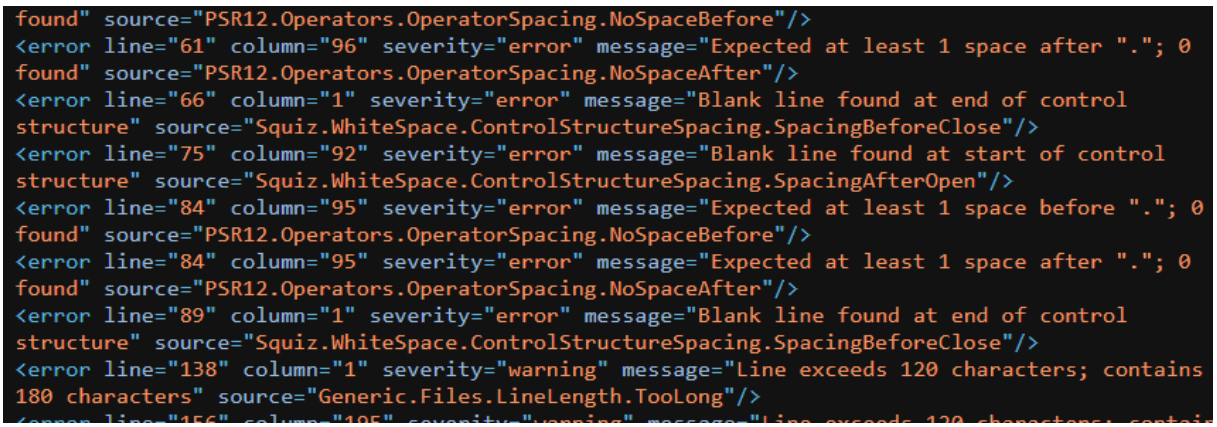
facilement intégrable dans un workflow GitHub Actions. Les résultats sont générés sous forme de rapport et joints au pipeline comme artefact

Les solutions Snyk et SonarQube ont été écartées. Bien que puissantes et reconnues, leurs fonctionnalités utiles pour une analyse de sécurité complète sont majoritairement payantes

Analyse des résultats et gestion des faux positifs

Résultats PHP_CodeSniffer :

- la gestion des espaces et des retours à la ligne (espaces manquants ou superflus)
- l'indentation incorrecte dans les structures multilignes
- le placement des parenthèses et des accolades
- des lignes dépassant la longueur maximale recommandée (120 caractères)



donc les alertes sur non-uniformité du style de code

Résultats CodeQL :

- utilisation de générateurs de nombres aléatoires non sécurisés
- problèmes d'échappement ou d'encodage incomplet de chaînes de caractères

Une part significative de ces alertes concerne des fichiers situés dans les répertoires vendor et public/assets

Structure du pipeline

Le pipeline CI/CD mis en place est composé de deux jobs :

- build est chargé de vérifier que l'application peut être installée, configurée et exécutée correctement.
  Il effectue le checkout du code source, configure les environnements PHP et Node.js, installe les dépendances PHP et JavaScript, génère le build du frontend. Initialise MariaDB , les migrations sont exécutées et les tests PHP sont lancés à l'aide de l'outil Pest. Les résultats des tests sont sauvegardés sous forme d'artefact afin de pouvoir être consultés après l'exécution du pipeline.

- security est exécuté uniquement si le job *build* s'est terminé avec succès
  Il se concentre sur l'analyse statique du code. CodeQL est utilisé pour analyser le code JavaScript, tandis que PHP_CodeSniffer analyse le code PHP. Les rapports générés sont également stockés sous forme d'artefacts.

avantages / inconvénients

Scan code and beyond for any exposed secrets

Conclusion

Ce projet a permis de comprendre le fonctionnement d'un pipeline CI/CD, et de se familiariser avec les outils de sécurité automatisés et de confronter les limites des solutions gratuites disponibles sur le marché. Malgré de nombreuses difficultés liées aux permissions GitHub, au support des langages, versions, à la configuration des tests et à la gestion des artefacts, le workflow est fonctionnel.