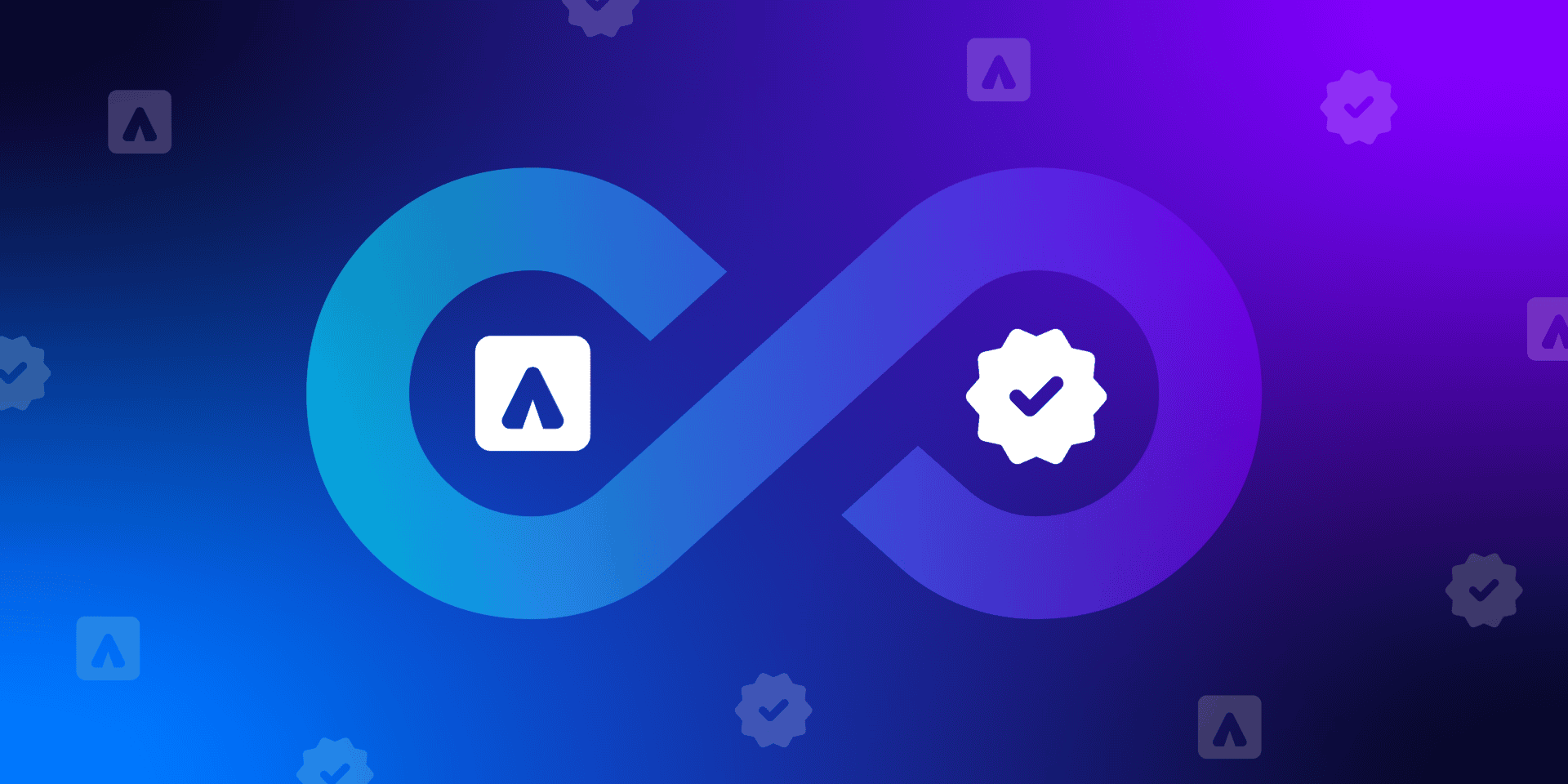
**PORTFOLIO**



William Trelles – Classe CID 3A

ETML-Lausanne-Vennes

Professeur : Mathieu Meylan

Table de matière

[1 Introduction 3](#_Toc187401906)

[2 Planification initiale 3](#_Toc187401907)

[3 Actions nécessaires 3](#_Toc187401908)

[4 Etapes du workflow 3](#_Toc187401909)

[4.1 Étapes de validation 4](#_Toc187401910)

[4.1.1 1.1 Backend-check 4](#_Toc187401911)

[4.2 Frontend-check 4](#_Toc187401912)

[4.3 Étapes de test 4](#_Toc187401913)

[4.3.1 Tests End-to-End (E2E) 4](#_Toc187401914)

[4.4 Étapes de construction 5](#_Toc187401915)

[4.4.1 Frontend-build 5](#_Toc187401916)

[4.4.2 Artefacts 5](#_Toc187401917)

[4.5 Étapes de déploiement 5](#_Toc187401918)

[4.5.1 Configuration et déploiement 5](#_Toc187401919)

[5 Diagramme de flux 5](#_Toc187401920)

[6 Préparation du serveur 6](#_Toc187401921)

[6.1 Installation et configuration de PM2 6](#_Toc187401922)

[6.2 Utilisation du port 80 avec authbind 6](#_Toc187401923)

[6.3 Configuration d’un alias pour authbind avec PM2 6](#_Toc187401924)

[7 Configuration SSH pour les VM 6](#_Toc187401925)

[8 Installation de NGINX pour servir le frontend 7](#_Toc187401926)

[8.1 Installation de nginx 7](#_Toc187401927)

[8.2 Créer le fichier où sera le config du serveur de l’application. 7](#_Toc187401928)

[8.3 Activer le fichier de configuration. 8](#_Toc187401929)

[8.4 Vérifier les erreurs de syntaxe : 8](#_Toc187401930)

[8.5 Recharger la configuration : 8](#_Toc187401931)

[9 Rapport de déploiement 8](#_Toc187401932)

[9.1 Environnements de déploiement 8](#_Toc187401933)

[URL de production 8](#_Toc187401934)

[URL de test 8](#_Toc187401935)

[9.2 Etat actuel de l'application 9](#_Toc187401936)

[9.3 Conclusion 9](#_Toc187401937)

[10 Schéma CI/CD 9](#_Toc187401938)

[11 Usage de l’IA 10](#_Toc187401939)

# Introduction

Le but de ce projet est de déployer une application « todo – app » sur un serveur dédié à l’aide d’un workflow de GitHub actions. Le déploiement de production sera déclenché manuellement et le déploiement de test lors de chaque « push » sur la branche main. Pour optimiser le workflow certaines techniques de mise en cache et exécution des taches en parallèle seront mises en place.

# Planification initiale

J’ai réalisé la planification à l’aide de l’outil Trello et elle disponible à partir de ce [lien](https://trello.com/invite/b/6736fc8dbc46fe8f42ee4e5d/ATTI8ad95915af8108fe0b97e55a9c1f695b8C6BA079/pdevops-324-william).

# Actions nécessaires

**Contrôle initial et arrêt en cas d’échec** : Le déclenchement du workflow se produit dès qu’un nouveau code est poussé sur la branche main. Si certaines conditions préliminaires ne sont pas remplies, le processus s’interrompt immédiatement pour éviter tout déploiement d’un code potentiellement instable.

**Vérifications du backend** : Le premier ensemble d’actions consiste à récupérer le dépôt, installer la version spécifiée de Node.js, puis exécuter une installation des dépendances backend. S’ensuivent un audit de sécurité et des contrôles de qualité du code (lint) afin de détecter des vulnérabilités potentielles et de garantir la conformité stylistique. Vérifications du frontend : De façon analogue, le frontend subit une série de tests similaires : récupération du dépôt, configuration de l’environnement Node.js, installation des dépendances, audit de packages et exécution du lint. Toute anomalie détectée dans le code frontend provoque l’arrêt du processus, évitant ainsi la propagation de problèmes en aval.

**Contrôle combiné et tests end-to-end (E2E)** : Une fois les vérifications du backend et du frontend validées, la chaîne d’intégration passe à l’étape des tests E2E. Ces derniers, exécutés à l’aide d’outils spécifiques (par exemple Cypress), valident le comportement global de l’application, simulent les interactions des utilisateurs et garantissent que tous les composants fonctionnent de manière cohérente et intégrée.

**Construction et déploiement** : Si les contrôles et tests antérieurs sont satisfaisants, la construction du backend et du frontend s’effectue, produisant des artefacts prêts à être déployés. Dans le cas où cette phase se déroule sans incident, le déploiement peut avoir lieu, assurant que seules les versions conformes du code parviennent à l’environnement de production.

# Etapes du workflow

**Description du workflow CI/CD**

Le workflow CI/CD suivant est conçu pour automatiser les étapes de validation, de tests, de construction et de déploiement pour une application TodoApp comprenant un backend et un frontend. Il est déclenché lorsqu'un commit est poussé sur la branche principale (main).

## Étapes de validation

### 1.1 Backend-check

**Objectif** : Valider le code backend en effectuant un audit et en appliquant des vérifications de style (lint).

**Étapes** :

1. Récupération du dépôt : Utilisation de l'action actions/checkout pour récupérer le code source.
2. Installation de Node.js : Installation de Node.js version 21.1.0 à l'aide de actions/setup-node.
3. Installation des dépendances : Utilisation de npm ci --prefer-offline pour installer rapidement les dépendances.
4. Audit des packages et vérification de style : Commandes npm run lint et npm audit exécutées grâce à une stratégie matricielle pour tester plusieurs commandes.

## Frontend-check

**Objectif** : Valider le code frontend avec les mêmes vérifications que le backend.

**Étapes** : identiques au backend-check, mais adaptées au dossier todo-app/frontend.

## Étapes de test

### Tests End-to-End (E2E)

**Objectif** : Tester le frontend dans un environnement simulé avec Cypress pour garantir une expérience utilisateur fonctionnelle.

Étapes :

1. Installation des dépendances backend : npm ci pour le backend et frontend.
2. Démarrage du backend : commande node index.js pour lancer le backend en arrière-plan.
3. Construction du frontend : construction de la version finale avec npm run build.
4. Tests E2E :

* action cypress-io/github-action pour tester avec plusieurs navigateurs (Chrome, Edge, Firefox).
* Vérification que l'application fonctionne correctement avec Cypress.

## Étapes de construction

### Frontend-build

**Objectif** : Construire le frontend après avoir validé les tests E2E.

**Étapes :**

1. Installation des dépendances : npm ci pour installer les dépendances du frontend.
2. Build avec npm run build
3. Sauvegarde de la build : création d'un artefact nommé frontend-build contenant les fichiers nécessaires.

### Artefacts

Artefacts sauvegardés :

* frontend-build : Contient les fichiers du frontend.

## Étapes de déploiement

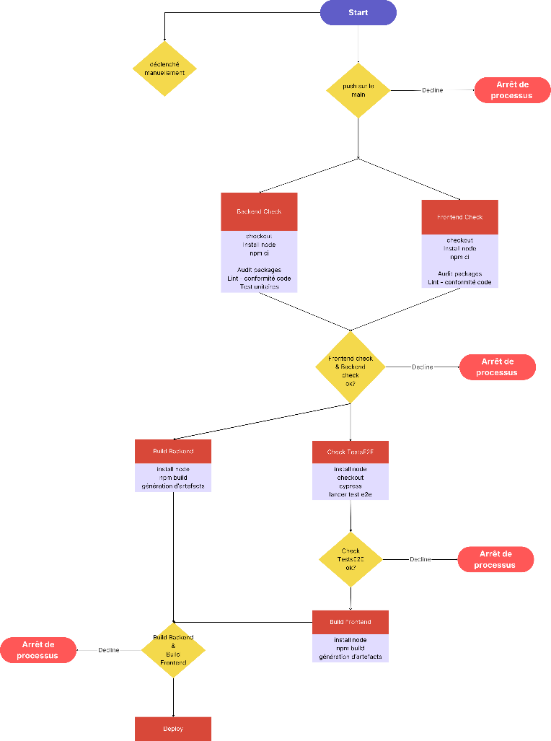
### Configuration et déploiement

**Objectif** : Déployer le backend et le frontend sur le serveur de test.

**Étapes** :

1. Connexion au VPN via des secrets GitHub.
2. Copie des fichiers backend et frontend sur le serveur (via scp).
3. Déploiement du backend avec PM2.
4. Déploiement du frontend, suivi d’un redémarrage de Nginx.

# Diagramme de flux



# Préparation du serveur

Pour pouvoir déployer l’application sur les différents serveurs prod et test, il a d'abord été nécessaire de les configurer.

## Installation et configuration de PM2

**sudo apt install npm**

**sudo npm install pm2 -g**

## Utilisation du port 80 avec authbind

L’application étant configurée pour être lancée sur le port 80 (réservé normalement aux processus root), il a été nécessaire de configurer les VMs afin de permettre son exécution sur ce port sans utiliser l’utilisateur root. Pour cela, j’ai utilisé authbind, qui permet à des processus non-root d'écouter sur des ports inférieurs à 1024. Voici les étapes :

Installer authbind :

**sudo apt-get install authbind**

Créer un fichier pour autoriser l’accès au port 80 :

**sudo touch /etc/authbind/byport/80**

Donner les permissions nécessaires à l’utilisateur :

**sudo chown ubuntu /etc/authbind/byport/80**

**sudo chmod 755 /etc/authbind/byport/80**

## Configuration d’un alias pour authbind avec PM2

Ajouter un alias pour utiliser authbind, pour cela rajouter cette ligne à la fin du fichier .bashrc :

**alias pm2='authbind --deep pm2'**

* Faire update avec : **source ~/.bashrc et pm2 update**
* Vérifier la création de l’alias avec: **alias pm2**

# Configuration SSH pour les VM

Pour configurer l'accès SSH aux machines virtuelles et permettre une connexion sécurisée, voici les étapes réalisées :

1. Création d’une public key

**ssh-keygen -t rsa -b 4096 -C "email\_git \_hub"**

* Il faudra confirmer un passphrase : **etml123**

1. Se connecter au serveur principal

**ssh** [**admin347@172.26.1.19**](mailto:admin347@172.26.1.19)

1. Connexion à la VM via le serveur principal

« **multipass shell nom\_de\_la\_vm** ».

1. Configuration du répertoire SSH dans la VM

Créer le répertoire ~/.ssh :  
**mkdir -p ~/.ssh**

**chmod 700 ~/.ssh**

Copier la clé publique crée dans le premier pas, pour l’observer il suffit lancer cette ligne dans le cmd :  
**cat ~/.ssh/id\_rsa.pub**

Coller cette clé dans la VM de cette manière :

**echo "CLE\_PUBLIQUE " >> ~/.ssh/authorized\_keys**

Configurer les droits nécessaires :

**chmod 600 ~/.ssh/authorized\_keys**

1. Connexion à la VM avec un saut de serveur

**ssh -J admin347@172.26.1.19 ubuntu@ip\_de\_vm**

# Installation de NGINX pour servir le frontend

## Installation de nginx

**sudo apt-get update -y**

**sudo apt-get install -y nginx**

## Créer le fichier où sera le config du serveur de l’application.

sudo nano /etc/nginx/sites-available/todoapp

server {

listen 80;

server\_name nom\_domaine\_ou\_IP;

root /home/ubuntu/www/frontend/dist;

index index.html;

location / {

try\_files $uri $uri/ /index.html;

}

error\_page 404 /index.html;

}

## Activer le fichier de configuration.

**sudo ln -s /etc/nginx/sites-available/todoapp /etc/nginx/sites-enabled/**

## Vérifier les erreurs de syntaxe :

**sudo nginx -t**

## Recharger la configuration :

**sudo systemctl reload nginx**

# Rapport de déploiement

Chaque environnement dispose de son propre processus de déploiement, avec des étapes manuelles ou automatiques selon l'environnement cible.

## Environnements de déploiement

URL de production : <http://172.26.0.24/>

Pour déployer sur l'environnement de production, il existe un trigger manuel dans le workflow de GitHub Actions.

URL de test : <http://172.26.0.29/>

Le déploiement vers l'environnement de test est automatique. Chaque fois qu'une modification est poussée vers la branche main

#### Déploiement avec le triggeur manuel

## Etat actuel de l'application

Frontend : L'interface de l'application est correctement déployée sur les deux environnements production et test. Les utilisateurs peuvent accéder aux pages statiques.

Backend : À ce stade, le backend n'est pas fonctionnel sur les deux environnements. Cela peut être dû à des problèmes dans la configuration du backend.

## Conclusion

Le déploiement de l’application vers les environnements de production et de test a été effectué avec succès, et l’interface de l’application est désormais accessible sur les deux machines virtuelles. Toutefois, le backend nécessite encore des ajustements pour être fonctionnel et interagir correctement avec l'interface. Une fois ces problèmes résolus, l'application sera pleinement opérationnelle dans les deux environnements.

# Schéma CI/CD



# Usage de l’IA

L’intégration de l’IA dans le projet s’est avérée un outil clé pour améliorer la productivité et la qualité du développement. Voici les différents usages de l’IA dans le contexte de ce projet :

**Assistance** au développement du workflow CI/CD

Une compréhension approfondie des étapes nécessaires pour automatiser la validation, les tests, la construction et le déploiement.

Des suggestions d’amélioration pour :

L’utilisation des artefacts pour partager les builds entre les jobs.

Pour la configuration Nginx.

Le déploiement vers les environnements de test et de production.

**Débogage des erreurs**

Analyser les logs d’erreurs provenant de GitHub Actions.

Identifier les causes probables (par exemple, Host key verification failed, problèmes de permissions SSH, ou absence de connectivité VPN).

Proposer des solutions concrètes, comme l’ordre correct des étapes dans le workflow ou la configuration des clés SSH.

Optimisation de la configuration de Nginx pour éviter les boucles de redirection ou les erreurs de chemin.