

Table des matières

[1. Introduction 2](#_Toc184823404)

[1.1 Contexte 2](#_Toc184823405)

[1.2 Objectifs 2](#_Toc184823406)

[1.3 Gestion de projet 2](#_Toc184823407)

[2. Analyse préliminaire 3](#_Toc184823408)

[2.1 Analyse Initiale 3](#_Toc184823409)

[2.2 Planification initiale 4](#_Toc184823410)

[3. Mise en place 5](#_Toc184823411)

[3.1 Prérequis 5](#_Toc184823412)

[3.2 Description du Workflow 7](#_Toc184823413)

[3.3 Diagramme de flux du workflow 11](#_Toc184823414)

[4. Conclusion 11](#_Toc184823415)

[4.1 Rapport de déploiement 11](#_Toc184823416)

[4.2 Schéma CI/CD Github Action 13](#_Toc184823417)

[4.3 Utilisation de l’ia 14](#_Toc184823418)

# Introduction

## Contexte

Le projet « P\_DevOps » est réalisé dans le cadre du module 324, qui a comme thème les processus DevOps. Le but est déployé une application de To Do List en CI/CD avec Github Action, tout en utilisant les outils et méthodes vue en classe. Le projet utilise GitHub Project pour la gestion des taches et un journal de travail afin de suivre le déroulement du projet. Le chef de projet est Cédric Schaffter notre enseignant.

## Objectifs

Les objectifs du projet « P\_DevOps » sont les suivants :

**Mettre en place un workflow CI/CD avec GitHub Actions** :

1. Déploiement de test déclenché automatiquement sur chaque « push » vers la branche principale (« main »).
2. Déploiement de production déclenché manuellement.

**Actions à inclure pour le frontend et backend :**

1. Audit des packages pour détecter et éviter les vulnérabilités dans les dépendances.
2. Linting pour détecter les erreurs potentielles dans le code.

**Optimisation du workflow :**

1. Exécution parallèle de certaines tâches pour gagner en efficacité.
2. Mise en cache pour réduire les temps de déploiement.

**Environnement de déploiement :**

1. Les déploiements de test et de production se feront sur un serveur dédié.

## Gestion de projet

Ce projet est effectué en groupe de deux. Pour cela, nous utilisons l'outil GitHub Project afin de lister les tâches et attribuer les tâches à réaliser. Nous utilisons également GitHub Desktop ainsi que des commits pour que chacun puisse voir une version du projet à jour.

Un journal personnel est aussi utilisé sous forme de fichier CSV. Ce journal permet de suivre l'avancement du projet et de recenser les problèmes rencontrés. Ainsi, si une des deux personnes est absente, il est toujours possible de suivre l'avancement du projet. Le but est qu'aucun événement ne ralentisse l'avancement et le bon déroulement du projet.

Le nommage des commits est également un point important, car il permet au professeur d'évaluer le projet à 80 % et d'obtenir un retour.

# Analyse préliminaire

## Analyse Initiale

Afin de mener à bien les objectifs du projet nous l’avons découpé en plusieurs partie distinct que voici :

***Déploiement local***

Avant tous nous allons commencer par faire tourner l’application sur notre environnement local tout d’abord sans conteneurisation et en la containerisant par la suite avec un dockerfile pour le backend et un pour le frontend, cette étape nous permettra de nous familiariser avec le code et les différentes dépendances du projet, vérifier le bon fonctionnement de l’application et si besoins gérer certaines erreurs pouvant empêcher le déploiement final de l’application.

***Création des workflows***

Une fois que l’application est fonctionnelle dans l’environnement local, nous mettrons en place un workflow permettant de pousser les images des différents dockerfile sur dockerhub, le workflow construira les images avec la même procédure qu’en local et finira par les pousser sur dockerhub, cette étape permettra d’assurer un déploiement continu dès qu’une nouvelle version de l’application sera push sur la branche main du répertoire GitHub.

Nous créerons aussi un workflow permettant d’exécuter les différents tests unitaires de la partie backend afin de s’assurer du bon fonctionnement du code.

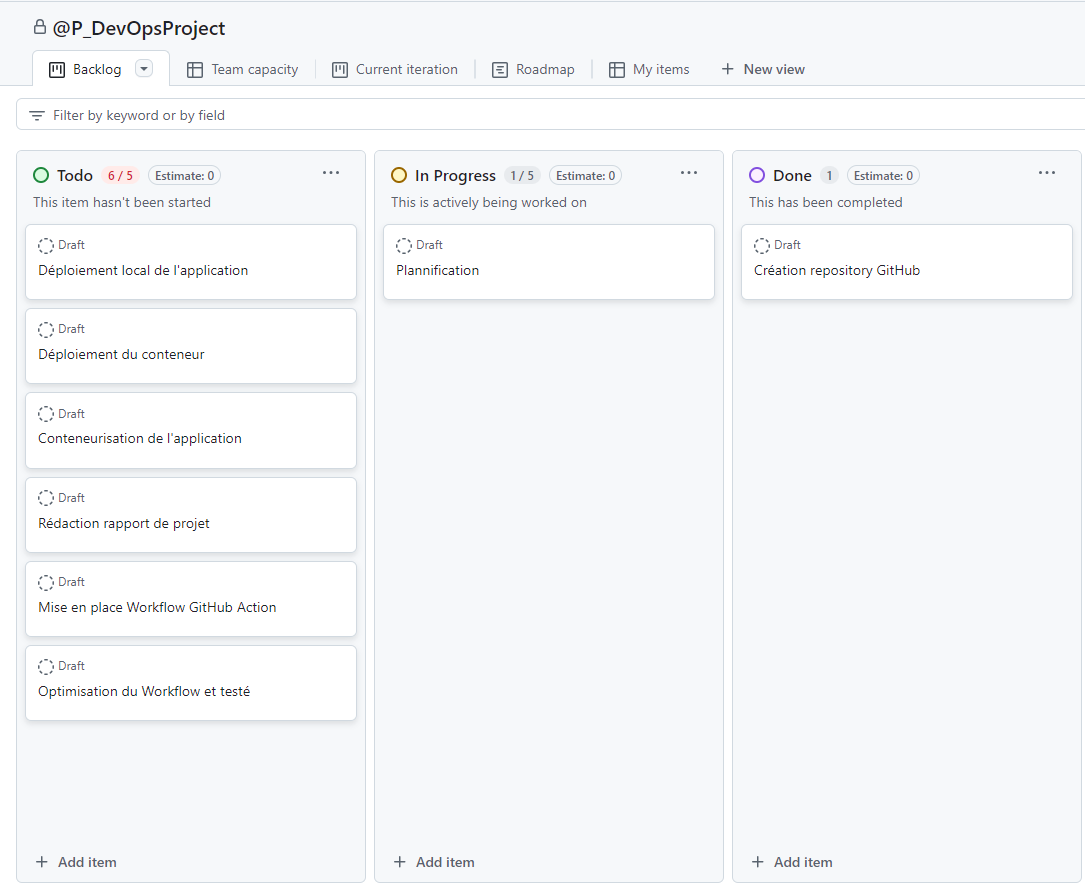
***Ecriture du rapport***

L’écriture du rapport sera une étape récurrente du projet, à chaque fois qu’une étape cruciale du projet est passée nous écrirons comment l’étape s’est déroulée ainsi que son fonctionnement.

## 2.2 Planification initiale

Dans ce projet, nous partons sans base en DevOps car nous n’avons pas encore eu la théorie du module 324. Nous nous sommes donc concentrés sur ce qui nous était spécifiquement demandé et avons cherché à comprendre le CDC afin de pouvoir au mieux planifier les tâches principales.

Nous avons réfléchi à ce qui était le plus important à faire au début, comme la mise en place de l’environnement, ainsi qu’aux différentes étapes nécessaires pour avancer efficacement, petit à petit, dans le projet.

Nous n’allons pas directement commencer par le CI/CD, mais d'abord déployer l'application petit à petit, ce qui donne les tâches ci-dessous :

To do list

# Mise en place

## Prérequis

***Installation***

La première étape afin de pouvoir mettre en place le projet est d’installer le dossier zip « todo-app.zip » [ici](https://apps.pm2etml.ch/jobDefinitions/124).

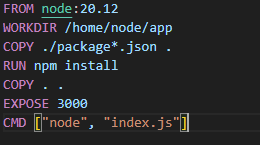
***BackEnd***

Une fois le dossier installé il est possible de créer le « dockerfile » à la racine du dossier « backend ».

Pour que le « dockerfile » fonctionne il doit exécuter les étapes suivantes :

* Pull Node.js version 20.12.
* Indiquer le répertoire de travail.
* Copier le fichier « package.json » dans le répertoire de travail.
* Installer les dépendances du projet.
* Recopier les fichiers dans le répertoire de travail.
* Exposer le port 3000 du container.
* Exécuter la commande « node index.js »

***Contenu du dockerfile***

******

***Modifications dans le code***

Avant de pouvoir build l’image du frontend il est primordiale de modifier quelques lignes du frontend.

Pour ce faire :

* Ouvrez le fichier package.json du dossier frontend
* Trouver la ligne « dev » de la partie s’occupant des scripts 
* Modifier la ligne afin que vite s’exécute sur le port 4173 

Puis :

* Ouvrir le fichier « vite.config.ts »
* Modifier la ligne target : 
* En 

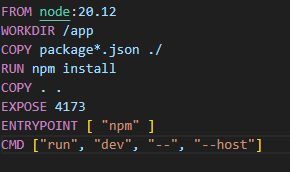
***FrontEnd***

Tout comme le backend il est nécessaire de créer le « dockerfile » à la racine du dossier « frontend »

Pour que le dockerfile fonctionne il doit exécuter les étapes suivantes :

* Pull Node.js version 20.12.
* Indiquer le répertoire de travail.
* Copier le fichier « package.json » dans le répertoire de travail.
* Installer les dépendances du projet.
* Recopier les fichiers dans le répertoire de travail.
* Exposer le port 4173 du container.
* Ajouter « npm » comme exécutable principale du container.
* Lancer le script « dev » du package.json avec comme paramètre de dev « --host ».

***Contenu du dockerfile***

******

***Publication des images sur dockerHub***

Une fois les différents dockerfiles effectué il est nécessaire de les build et de les publié sur docckerhub afin de les utiliser dans le docker compose.

Pour build les images il suffit d’exécuter, pour chacune des images, depuis le dossier ou elles se trouve, cette commande :

* docker build -t nomDeLimage . (Le «. » est nécessaire).

Une fois que les images ont été build, il est possible de les publié, pour cela il suffit de suivre les différentes étapes ci-dessous :

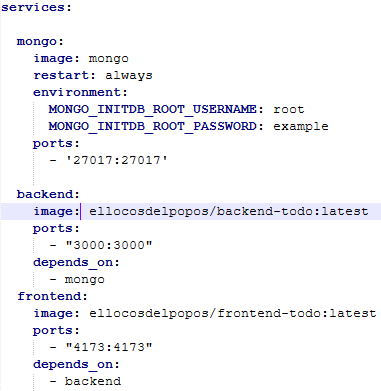
* Créer un compte docker sur dockerhub
* Depuis le cmd se connecter à docker avec la commande :
  + docker login -u nomUtilisateur -p motDePasse
* Tager l’image avec cette commande :
  + docker tag nom-de-limage nom-utilisateur/nom-de-limage :version
  + Mettre « latest » a la place de version si c’est la version la plus récente.
* Publier l’image avec la commande :
  + docker push nom-utilisateur/nom-image :version

***Docker compose***

Une fois toutes ces étapes effectuées il est désormais temps de mettre en place le docker compose, pour ce faire il suffit :

* Créer la section services.
* Mettre dedans le service « backend »
  + Lui attribuer le port 3000
* Mettre en place le service « frontend »
  + Lui attribuer le port 4173.
  + Ne le lancer que si le backend s’est lancé correctement.

***Contenu du docker compose***

******

A noté que tous ce qui est lié au container mongoDb est facultatif et n’est pas encore fonctionnel.

## Description du Workflow

***Audit & Linting***

La première étape du workflow consiste à vérifier le code grâce a au linter (lint) et vérifier les différentes dépendances grâce à l’audit.

*Etapes :*

* Installer les dépendances du backend.

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

* Installer les dépendances du frontend.

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

* Mise en place du linter pour le backend.

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

* Mise en place di linter pour le frontend.

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

* Mise en place de l’audit pour le backend en fixant les problèmes dans le cas où des vulnérabilités sont détecté.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* Mise en place de l’audit pour le backend en fixant les problèmes dans le cas où des vulnérabilités sont détecté.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

***Déploiement de l’image du backend sur DockerHub***

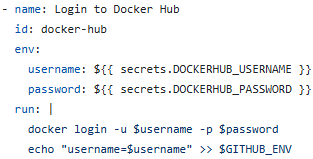
Cette étape permet au workflow de déployer l’image du backend sur DockerHub une fois que l’audit est passée.

*Etapes :*

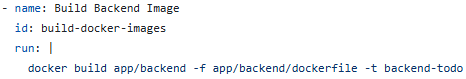
* Ne démarrer le déploiement que quand l’audit est passé.



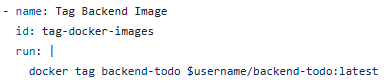
* Login a docker afin de déployer l’image en utilisant les différends secrets du répertoire githiub.



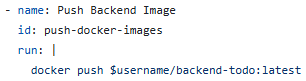
* Build de l’image du backend.



* Tag de l’image du backend avec le tag latest.



* Push de l’image du backend sur DockerHub.



***Déploiement de l’image du frontend sur DockerHub***

Cette étape permet au workflow de déployer l’image du backend sur DockerHub une fois que l’audit est passée.

*Etapes :*

* Ne démarrer le déploiement que quand l’audit est passé.



* Login a docker afin de déployer l’image en utilisant les différends secrets du répertoire githiub.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* Build de l’image du frontend.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* Tag de l’image du frontend avec le tag latest.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* Push de l’image du frontend sur DockerHub.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

***Installation de docker-compose et test des containers***

Installation de docker-compose afin de lancer et vérifier le bon fonctionnement des containers backend et frontend en lançant une requête curl aux différents services.

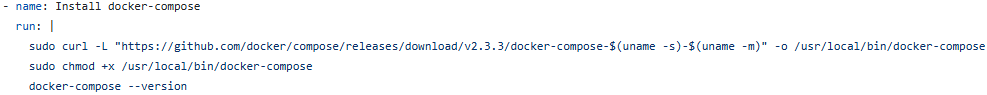
*Etapes :*

* Attendre que le build des images soient finis.

Une image contenant texte, Police, outil, conception

Description générée automatiquement

* Installer docker compose.



* Lancer le docker compose.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* Mise en place d’un time out afin de s’assurer que les containers soit bien actif et disponible.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

* Envoie d’une requête curl aux différents containers afin de s’assurer qu’ils fonctionnent.

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

* Fermeture des containers.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

## Diagramme de flux du workflow

Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

# Conclusion

## Rapport de déploiement

Notre déploiement de notre application est divisé en deux workflows :

1. Un premier qui va nous permettre de déployer l’application avec son environnement.
2. Un autre qui va tester l’application backend et frontend.

Pour l’instant, seul le workflow de déploiement de l’application est mis en place et est fonctionnel. Nous ne nous sommes pas encore penchés sur le workflow des tests. Voici comment celui-ci fonctionne, comment il est structuré, quels sont ses statuts et ce qu’il fait.

**Workflow Déploiement Application**

Notre workflow est divisé en plusieurs jobs. Un job contient plusieurs étapes qui sont exécutées les unes après les autres. Cependant, plusieurs jobs sont parfois exécutés en parallèle afin d’optimiser le temps et l’utilisation des ressources pendant l’exécution du workflow.

1. ***Job d’installation, d’audit et de linting :***

* Ce job nous permet de nous assurer que l’application et son code ne comportent pas de failles ou de problèmes dans les packages ou son code.
* C’est le premier job. Une fois qu’il est validé, cela déclenche l’exécution de deux autres jobs.

1. ***Jobs de build et de push des images Docker :***

* Ces deux jobs sont exécutés en parallèle et effectuent presque la même chose à un détail près :
  + L’un build l’image backend.
  + L’autre build l’image frontend.
* Les images sont ensuite poussées sur Docker Hub afin qu’on puisse, par la suite les exécuter sur notre docker-compose et mettre à jour l’application en temps réel.

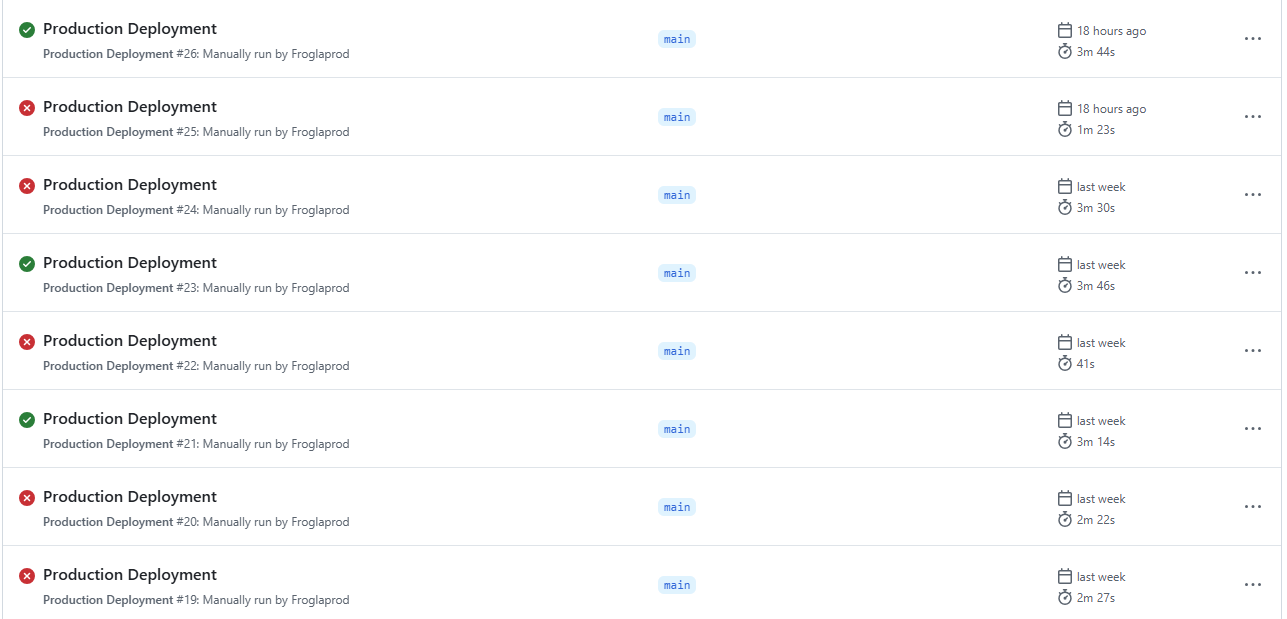
1. ***Jobs de build et de push des images Docker :***

* Ce job va tester et valider la création de notre application sur Docker. Il crée les containers et attend une réponse de leur part.
* Si cela fonctionne, il stoppe les containers afin de ne pas utiliser plus de ressources. Le déploiement est alors effectué.

Cette structure permet d’exécuter étape par étape le déploiement de l’application sans avoir de conflits entre les jobs. Chaque étape est isolée dans son environnement et dans son job associé.

**Historique et Optimisation**

Durant notre déploiement de notre application, nous avons dû déployer et tester plusieurs fois notre workflow. C’est pour cela que dans notre historique de déploiement, nous pouvons voir que le workflow était parfois en statut réussi et parfois en échec. Cela se produisait lorsque nous ajoutions de nouvelles étapes ou des jobs afin d’optimiser celui-ci ou de mieux le tester. Par exemple, lors de l’ajout du job de validation des containers.



**Etat actuel**

Actuellement, le workflow CI/CD de déploiement comprend les étapes suivantes :

1. Setup : Installation, audit, et linting.
2. Build et push : Construction et envoi des images Docker.
3. Validation des containers : Vérification de la fonctionnalité des images Docker.

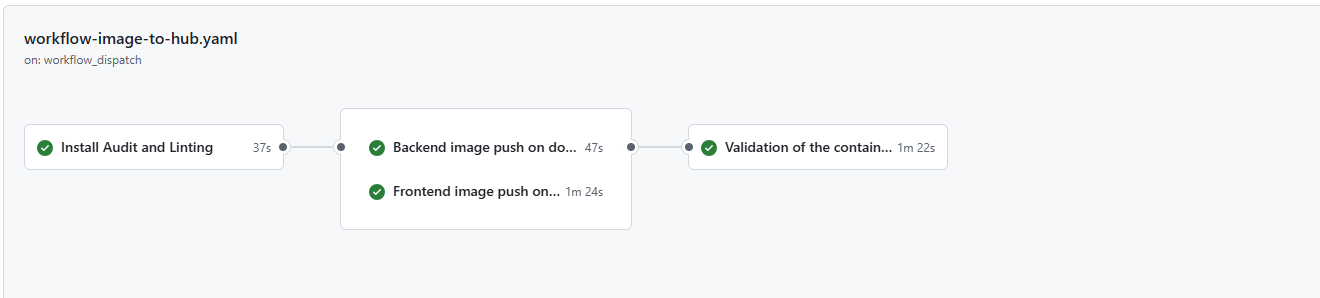
Les étapes manquantes sont les suivantes :

1. Déploiement serveur : Déployer la production sur un serveur dédié.

La seule étape qu’il manque est le déploiement en production sur un serveur dédié. Cela sera ajouté une fois que le workflow de tests sera mis en place.

## Schéma CI/CD Github Action

Voici le schéma de notre déploiement github action :

**Workflow déploiement production :**

**Workflow de test :**

(Non implémenté)

## Utilisation de l’ia

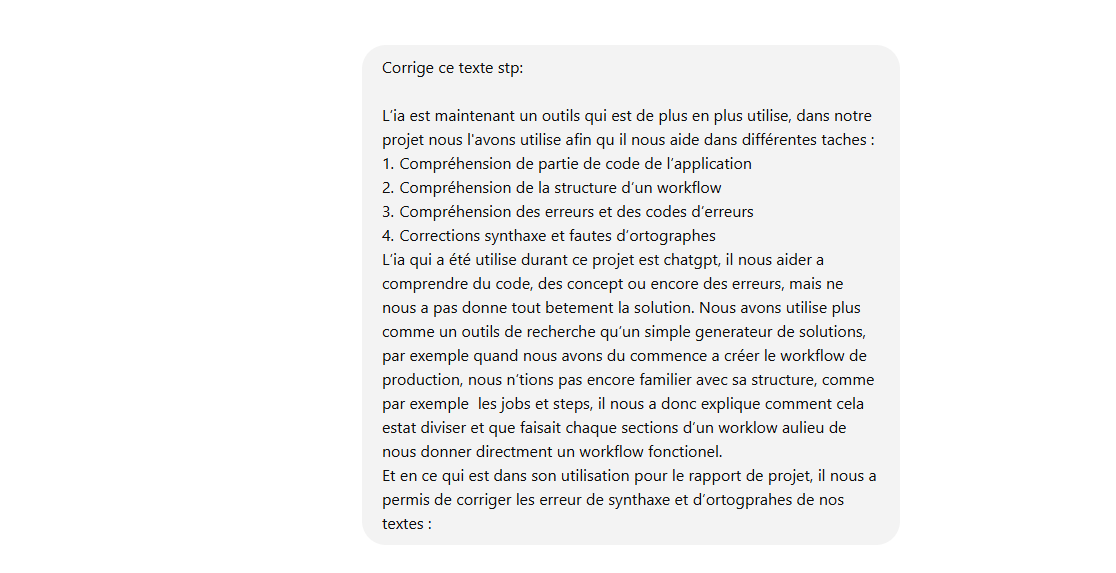
L’IA est maintenant un outil qui est de plus en plus utilisé. Dans notre projet, nous l'avons utilisée afin qu’elle nous aide dans différentes tâches :

1. Compréhension de parties de code de l’application.
2. Compréhension de la structure d’un workflow.
3. Compréhension des erreurs et des codes d’erreur.
4. Correction de la syntaxe et des fautes d’orthographe.

L’IA qui a été utilisée durant ce projet est ChatGPT. Elle nous a aidés à comprendre du code, des concepts ou encore des erreurs, mais ne nous a pas donné tout bêtement la solution. Nous l’avons utilisée davantage comme un outil de recherche qu’un simple générateur de solutions.

Par exemple, quand nous avons dû commencer à créer le workflow de production, nous n’étions pas encore familiers avec sa structure, comme les jobs et les steps. Elle nous a donc expliqué comment cela était divisé et ce que faisait chaque section d’un workflow, au lieu de nous donner directement un workflow fonctionnel.

En ce qui concerne son utilisation pour le rapport de projet, elle nous a permis de corriger les erreurs de syntaxe et d’orthographe de nos textes.

**Exemple concret d’utilisation :**

