

Table des matières

[1. Analyse préliminaire 2](#_Toc184311171)

[1.1 Introduction 2](#_Toc184311172)

[1.2 Objectifs 2](#_Toc184311173)

[1.3 Gestion de projet 2](#_Toc184311174)

[1.4 Planification initiale 2](#_Toc184311175)

[1.5 Compte rendu 4](#_Toc184311176)

[2. Développement 4](#_Toc184311177)

[2.1 Configuration de l’environnement 4](#_Toc184311178)

[2.2 Journal des commits 5](#_Toc184311179)

[2.3 Développement et test 6](#_Toc184311180)

[7. Conclusion 8](#_Toc184311181)

[7.1 Résultats finaux et conclusion 8](#_Toc184311182)

[7.2 Utilisation de l’IA 8](#_Toc184311183)

# Analyse préliminaire

## Introduction

Ce rapport est la seconde partie du projet Install\_In\_Nutshell qui est effectuée dans le cadre du module ICT 324. Cette partie a pour but d’explique la partie « OPS » du projet, elle passera donc en revue la mise en place des tests unitaires du script ainsi que les processus de déploiement du script préalablement créé dans la partie « Dev ».

## Objectifs

Les objectifs de la partie « OPS » d’Install\_In\_Nutshell sont les suivants :

**Plateforme de téléchargement du script** :

* Mise en place d’une plateforme web permettant le téléchargement du script « Install\_In\_Nutshell ».
* Mise en place d’une ressource « Azure Web app Service » ayant pour but d’hébergé la plateforme de téléchargement.
* Mise en place d’un « serveur de base de données flexible » sur Azure cloud.
* Création d’un workflow GitHub Action permettant d’automatiser le déploiement de la plateforme de téléchargement.

**Framework de test unitaire :**

* Création d’un script de test unitaire PowerShell nommé « unitTest.ps1 » permettant de tester certaines fonctionnalités du code.
* Mise en place d’un workflow exécutant automatiquement les différents tests unitaires afin de s’assurer du fonctionnement du code.

**Déploiement du script :**

* Mise en place d’un workflow permettant d’automatiquement insérer la dernière version du script dans la base de de donnée lors d’un push.

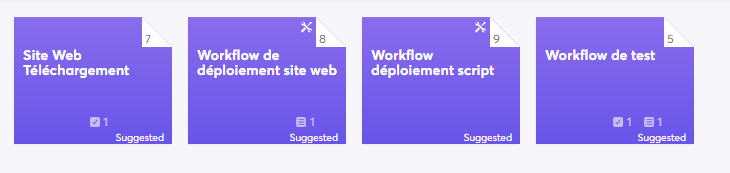
## Gestion de projet

Tout comme la partie dev, cette partie est réalisée en binôme, IceScrum restera l’outil permettant de définir les attentes et les différentes cibles du projet et tout comme la première partie GitHub sera utiliser comme outils de versionning ainsi que de déploiement grâce à GitHub Action qui nous permettra d’automatiser les différentes étapes de déploiement du projet. De plus un nouveau journal de travail avec « OPS » dans le nom sera créé exclusivement pour cette partie.

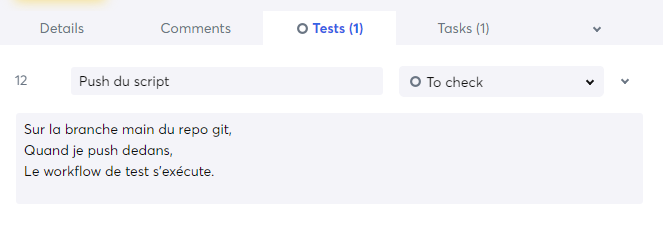
## Planification initiale

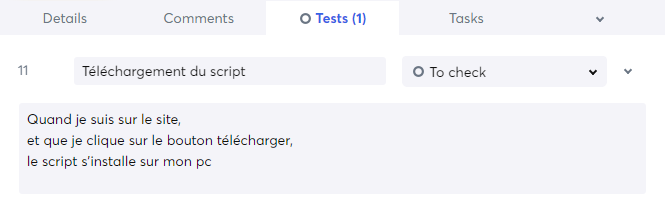
Afin de bien débuter et de ne pas être perdus dans les étapes à venir, nous avons d’abord créé des user stories pour chaque étape de déploiement que nous souhaitions implémenter. Chacune de ces user stories a été accompagnée de tests d’acceptation qui nous permettront de bien comprendre les attentes du projet.

Voici donc les user stories et les technical stories qui ont été créées :



Comme mentionné précédemment, chaque user story a été créée avec des tests d’acceptance. Voici un exemple concret de tests d’acceptance qui ont pu être mis en place sur les user stories :





Une fois les user stories créées et acceptées, nous avons commencé à réfléchir à la manière dont nous allions mettre en place le déploiement et organiser le déroulement de la seconde partie du projet : quelles technologies utiliser pour le site, quelle ressource Azure Cloud choisir, quelles seraient les étapes du déploiement, et à quel moment nous commencerions à rédiger le rapport de projet.

***Plateforme de téléchargement***

Il a été décidé de créer un site de téléchargement avec HTML/CSS/JS basique (sans Framework ou librairie) pour le front, étant donné le peu de temps mis à disposition et le fait que nous avons besoin d’un site simple permettant simplement d’installer le script, le backend quant à lui sera créé avec la version 20.12.0 de NodeJS car nous avons déjà l’habitude de l’utiliser.

Le site sera déployé sur une ressource Azure Web App Service car c’est la ressource offrant le déploiement le plus simple et rapide.

La base de données quant à elle se trouvera sur un serveur Azure Database for MySql flexible server nous permettons d’y accéder depuis le site ainsi que le workflow de déploiement.

***Déploiement***

Le déploiement sera effectué avec un workflow depuis GitHub Actions cela nous permettra de déployer les éléments de notre repository Git sans problème, de plus, GitHub offre une très bonne compatibilité avec Microsoft Azure Cloud.

Le déploiement est composé de 4 étapes clef :

* Analyse du message du commit.
* Exécution des différents tests unitaire du script grâce au script « unitTest.ps1 » si le commit contient « feat ».
* Insertion du script dans la base de données si les tests sont passé.
* Déploiement du site web si le commit contient (feat web).

## Compte rendu

Etant donne que ce projet a été fait par deux, nous avons dû nous répartir les tâches et prendre des décisions. Voici qui a fait quoi et comment nous avons réparti les tâches et pourquoi ces choix :

|  |  |  |
| --- | --- | --- |
| **Tâche** | **Responsable** | **Motif du choix** |
| Création des user stories | Belkhiria Sofiène | Ce choix à été fait car Mathis s’était chargé de cette tache lors de la partie dev. |
| Création du site web | Botteau Mathis | Pendant que Sofiène mettais en place le workflow de test Mathis créait le site web, ce choix paraissait logique étant donné que Sofiène avait créer les tests unitaires lors de la partie dev et que mathis est plus à l’aise pour les taches graphique. |
| Workflow de test | Belkhiria Sofiène | Ce choix à été pris car c’est sofiène qui s’est occupé de la création des tests unitaire. |
| Workflow Déploiement site web | Botteau Mathis | Ce choix paraissait logique étant donné que la création du site a été effectué par Mathis. |
| Workflow déploiement script | Belkhiria Sofiène | Ce choix à été fait car Sofiène voulait essayer de faire communiquer un workflow avec une base de données. |
| Rédaction du rapport | Botteau Mathis & Belkhiria Sofiène | Etant donné que le projet à été effectué a deux, le choix été logique que chacun parle des choses qu’il a fait. |

# Développement

## Configuration de l’environnement

L’environnement qui a été mis en place pour travailler sur ce projet est le suivant :

* Installation de NodeJS v 20.12.0, l’utilisation de cette version est dû au fait qu’elle est préinstallée sur les ordinateurs ayant été utilisé lors du projet.
* Installation de Github Desktop et clonage du répertoire du projet.
* Mise en place d’une ressource Azure Web App Service utilisant NodeJs v20 et GitHubAction en tant que « Deployment Provider ».
* Mise en place d’une ressource Azure Database for MySql flexible server.

## Journal des commits

Etant donne que nous avons utilisé github pour versionner notre projet, nous avons décidé d’utiliser une structure de commits comme celle-ci : « type(thème) : Description »

Voici notre journal de commits :

|  |  |  |
| --- | --- | --- |
| Auteur | Date | Message |
| termimi | Thu Dec 19 13:41:16 2024 +0100 | [doc] (journal de travail ops) Update journal de travail |
| termimi | Thu Dec 19 13:34:57 2024 +0100 | [Deploy] (deployToDB.yml) Update deploy to db workflow |
| termimi | Thu Dec 19 13:34:02 2024 +0100 | [infra] (localDb) Suppression du dossier local db car il n'est plus utile |
| termimi | Thu Dec 19 13:26:23 2024 +0100 | Update deployToDB.yml colonne table |
| termimi | Thu Dec 19 13:24:28 2024 +0100 | Update deployToDB.yml cat script |
| Mathis Botteau | Thu Dec 19 13:23:04 2024 +0100 | Merge branch 'main' of https://github.com/termimi/Install\_in\_NutShell |
| Mathis Botteau | Thu Dec 19 13:23:01 2024 +0100 | doc(journal de travail): Mise a jour des journaux de travaux / séparation |
| termimi | Wed Dec 18 16:32:32 2024 +0100 | Merge branch 'main' of https://github.com/termimi/Install\_in\_NutShell |
| termimi | Wed Dec 18 16:32:28 2024 +0100 | [doc] (journal travail) update journal de travil ops |
| termimi | Wed Dec 18 16:28:46 2024 +0100 | Update deployToDB.yml verify loadfile dir |
| termimi | Wed Dec 18 16:24:25 2024 +0100 | Update deployToDB.yml absolute link |
| termimi | Wed Dec 18 16:18:51 2024 +0100 | Update deployToDB.yml ls |
| termimi | Wed Dec 18 16:17:38 2024 +0100 | Update deployToDB.yml |
| termimi | Wed Dec 18 16:16:27 2024 +0100 | Update deployToDB.yml |
| Mathis Botteau | Wed Dec 18 16:15:35 2024 +0100 | Merge branch 'main' of https://github.com/termimi/Install\_in\_NutShell |
| Mathis Botteau | Wed Dec 18 16:15:19 2024 +0100 | doc(journal de travail): Ajout des heures de travail du 18.12.2024 |
| termimi | Wed Dec 18 16:11:53 2024 +0100 | Update deployToDB.yml delete table creation |
| termimi | Wed Dec 18 16:08:08 2024 +0100 | Update deployToDB.yml guillemet |
| termimi | Wed Dec 18 16:02:23 2024 +0100 | Update deployToDB.yml directly form bash |
| Mathis Botteau | Wed Dec 18 15:59:56 2024 +0100 | Merge branch 'main' of https://github.com/termimi/Install\_in\_NutShell |
|  |  |  |

## 

## Développement des tests unitaire

***Objectifs :***

Cette a partie avait pour but de créer un Framework permettant d’exécuter des tests unitaires sur les différentes fonctions du script, étant donné le peu de temps alloué au projet, nous avons décidé de nous concentrer un maximum sur l’ajout d’un utilisateur au group « docker-users ».

Pour se faire nous avons créer un second script nommé « unitTest.ps1 » nous permettons de lancer le script principale « install\_in\_nutshell.ps1 » avec des condition précise afin de vérifier le comportement de l’ajout d’un utilisateur au groupe « docker-users ».

***Script :***

*# Fonction pour vérifier si deux valeurs sont égales*

function Assert-Equal {

    param (

        [scriptblock]$action,  *# Le bloc de code à exécuter*

        $expected             *# La valeur attendue*

    )

*# Le symbole & est l'opérateur d'appel en PowerShell. Il est utilisé pour exécuter un script ou un bloc de script.*

*# La ligne permet de recupérer le retour de l'action*

    $result = (& $action | Out-String).Trim()

    if([string]$expected -eq [string]$result){

        Write-Host "test réussi"

    }

    else {

        Write-Host "test echoué"

        Write-Host("expected :" + $expected)

        Write-Host("result :" + $result)

    }

}

*# Fonction pour vérifier si une valeur contient une autre valeur*

function Assert-Like {

    param (

        [scriptblock]$action,  *# Le bloc de code à exécuter*

        $expected             *# La valeur attendue qui doit contenir le résultat*

    )

*# Le symbole & est l'opérateur d'appel en PowerShell. Il est utilisé pour exécuter un script ou un bloc de script.*

*# La ligne permet de recupérer le retour de l'action*

    $result = (& $action | Out-String).Trim()

    if("$expected" -like "\*$result\*"){

        Write-Host "test réussi"

    }

    else {

        Write-Host "test echoué"

        Write-Host("expected :" + $expected)

        Write-Host("result :" + $result)

    }

}

*# Test 1: Vérifie que le script gère correctement un utilisateur inexistant*

Write-Host "Test1"

$excpectedResult = "L'utilisateur ou groupe global suivant n'existe pas : wefwefd."

Assert-Like -action { .\install\_in\_nutshell.ps1 -CompleteInstall $false -UserToAdd wefwefd > $null 2>&1} -expected $excpectedResult

*# Supprime l'utilisateur du groupe docker-users pour préparer le test suivant*

net localgroup docker-users po66qga /DELETE > $null 2>&1

*# Test 2: Vérifie que l'ajout d'un nouvel utilisateur fonctionne*

Write-Host "Test2"

$excpectedResult = "La commande s'est terminée correctement."

Assert-Like -action { .\install\_in\_nutshell.ps1 -CompleteInstall $false -UserToAdd po66qga > $null 2>&1} -expected $excpectedResult

*# Test 3: Vérifie que le script gère correctement la tentative d'ajout d'un utilisateur déjà présent*

Write-Host "Test3"

$excpectedResult = "Le nom de compte spécifié est déjà membre du groupe."

Assert-Like -action { .\install\_in\_nutshell.ps1 -CompleteInstall $false -UserToAdd po66qga > $null 2>&1} -expected $excpectedResult

***Test Utilisateur Inexistant***

Ceci est le premier test unitaire, son but est de vérifier que la script renvoie bien un message dans le cas où le script essaie de rajouter un utilisateur dans le groupe « docker-users » alors que l’utilisateur n’existe pas sur l’ordinateur.

***Test Ajout Nouvelle Utilisateur***

Ce test vérifie que le script ajoute bel et bien un utilisateur dans le groupe « docker-users » si celui-ci n’est pas déjà présent.

***Test Ajout Utilisateur Présent***

Ce test Vérifie que le script renvoie bien un message dans le cas ou l’utilisateur devant être ajouté est déjà présent dans le groupe « docker-users »

***Problème rencontré***

Le seul problème rencontré été dû au fait que la commande utilisée afin d’ajouter un utilisateur ne renvoie pas d’erreur ni d’output (commande « write-output ») ce qui rendait compliqué de capturer toute sortie de la commande.

***Solution***

Nous avons donc décider de renvoyer la commande entière dans une variable plutôt que seulement son retour, cette solution nous as permis de pouvoir vérifier les messages retourner par la commande afin de pouvoir exécuter nos tests.



A noté que cette modification a dû être faites dans le script principale et non dans le script de test.

***Conclusion***

Grâce à ces tests nous nous assurons que le script n’agit pas ou ne plante pas d’une manière inattendue en plus de nous assurer que le script renvoie bel et bien des messages clairs à l’utilisateur.

## Développement du site web

***Objectifs :***

Dans le cadre de notre script PowerShell d'installation, nous avons réfléchi à comment faire pour pouvoir le déployer en production et permettre à des utilisateurs de l'utiliser facilement. Nous sommes arrivés à la conclusion qu'un site web qui permettrait de télécharger le script serait utile.

Grâce à cela, un utilisateur pourrait simplement accéder au site web et, en un clic, télécharger le script afin de l'utiliser. C'est pourquoi nous avons décidé de tout d'abord créer un site web basique, afin de pouvoir le déployer par la suite.

***Technologies :***

Nous avons décidé d'utiliser HTML, CSS, JS et Node.js, car ce sont des langages que nous avions déjà utilisés et qui nous étaient familiers. Nous n'avons donc pas été en terrain inconnu.

***Structure :***

Le site web est un site en HTML, CSS et JS, rien de complexe : un affichage basique et simple avec un seul bouton permettant de télécharger le script. Pas de fonctionnalités complexes, juste un squelette HTML/CSS, et une méthode JS qui permet de récupérer via un backend Node.js notre script stocké dans une base de données MySQL et de nous le retourner pour que le téléchargement puisse être effectué.

La structure du site est la suivante :

**Dossier frontend :**

* + *Dossier script* : contient un fichier main.js.
  + *Dossier src* : contient un dossier img avec les images et logos.
  + *Dossier styles* : contient un fichier style.css pour le code du style.
  + *À la racine* : le fichier index.html, qui est la page HTML.

**Dossier backend :**

* + *Fichier app.js* : c'est l'application backend qui permet de se connecter à une base de données MySQL et d'y récupérer le script***.***

***Mise en place de l’environnement :***

Pour tester les fonctionnalités, nous avons lancé un conteneur Docker avec une interface phpMyAdmin et un conteneur MySQL. Cela nous a permis de créer une base de données de test***.***

***Développement :***

Nous avons commencé par créer la page web en elle-même, en travaillant sur l'apparence du site. Nous avons opté pour un design simple et efficace, avec seulement quelques textes, un logo et un bouton de téléchargement.

Voici le site web : Une image contenant texte, capture d’écran, Police, Système d’exploitation

Description générée automatiquement

Ensuite, nous avons créé une méthode qui permet de récupérer, par l'intermédiaire de notre fichier app.js, le fichier script. Cette méthode se connecte à la base de données, récupère le nom et le contenu du script, le convertit en un fichier, et retourne ce fichier complet à notre site web.

Ensuite, nous avons créé une méthode qui permet de récupérer, par l'intermédiaire de notre fichier app.js, le fichier script. Cette méthode se connecte à la base de données, récupère le nom et le contenu du script, le convertit en un fichier, et retourne ce fichier complet à notre site web.

La mise en place de la base de données a posé quelques problèmes au départ, notamment avec la connexion via app.js, mais nous avons rapidement résolu ces soucis. Comme nous savions exactement ce que nous voulions faire et comment, nous n'avons pas rencontré d'autres problèmes.

## Développement du workflow

***Déploiement du site web sur Azure :***

Après avoir réalisé le site web, il fallait le mettre en déploiement. Pour cela, nous avons décidé d'utiliser Azure Cloud. Nous avons choisi une Web App car elle permet de déployer facilement un site web tout en gérant la connexion avec une base de données. Contrairement à une VM, la Web App est déjà configurée pour les applications web, ce qui simplifie beaucoup le processus.

Nous avons donc créé **deux Web Apps** :

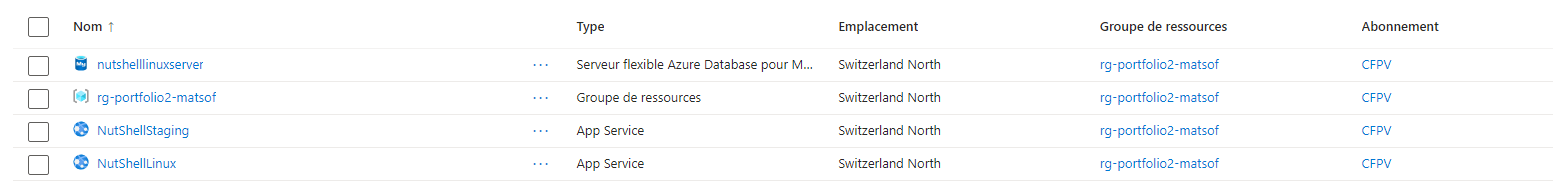
1. Une pour la **production**.
2. Une pour la **préproduction**.

Cela nous permet d’effectuer des tests unitaires sur notre script tout en validant le code dans un environnement de préproduction avant de le pousser en production. Si quelque chose ne fonctionne pas correctement (tests échoués, erreur dans le code, etc.), le déploiement s’arrête ou passe par la préproduction pour une vérification.

***Configuration Azure :***

Avant de lancer un workflow automatisé, nous avons dû configurer quelques éléments :

* Les chemins des fichiers pour le site web.
* La connexion avec la base de données.
* La configuration de la Web App sur Azure.
* Ajouter les commandes de lancement pour s'assurer que le site fonctionne correctement.

Une fois ces étapes réalisées et le site web opérationnel sur Azure, nous avons commencé à mettre en place des workflows automatisés.

***Structure :***

Au début, nous avions créé plusieurs workflows séparés :

* Vérification des commits.
* Lancement des tests unitaires.
* Déploiement de la base de données.
* Déploiement en production.
* Déploiement en préproduction.

Cependant, nous avons rapidement réalisé qu'un fichier unique regroupant toutes les étapes serait beaucoup plus pratique. Cela permet de lancer tout le processus en un seul lancement, simplifiant la gestion des taches.

Nous avons donc créé un workflow complet. Ce fichier inclut toutes les étapes nécessaires pour vérifier, tester et déployer notre site.

***Workflows :***

1. Vérification des commits

* Vérifie si le commit contient des mots-clés spécifiques comme feat ou web.
* Si ces mots sont présents, cela déclenche l’exécution de certains jobs. Par exemple, si feat est détecté, le déploiement en production sera autorisé.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Lancement des tests unitaires

* Exécute les tests unitaires du script PowerShell.
* Si un test échoue, le workflow le détecte et empêche le déploiement en production. Au lieu de cela, l’application sera déployée en préproduction pour effectuer des corrections.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Déploiemen de la db

* Crée une base de données MySQL sur Azure.
* Une image contenant capture d’écran, texte, Police

  Description générée automatiquementInjecte le script dans la base de données, permettant son utilisation directe par l’application.

1. Build et déploiement en préproduction

* Prépare les fichiers nécessaires au déploiement.
* Déploie l’application sur la Web App de préproduction.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Build et déploiement en production

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

  Description générée automatiquementSi toutes les étapes précédentes réussissent, l’application est déployée sur la Web App de production.

Les captures d'écran présentées ici montrent uniquement les jobs et steps principaux du workflow, ceux qui sont pertinents pour expliquer son fonctionnement. Le workflow complet contient d'autres détails techniques et configurations qui n'ont pas été affichés dans ce document.

Pour consulter l'intégralité du code du workflow CI/CD, vous pouvez accéder au dépôt GitHub du projet. Cela vous permettra d'explorer chaque étape en profondeur.

***Scénario de déploiement :***

Voici les différentes possibilités de déploiement dans notre workflow. Chaque scénario est illustré par un schéma décrivant ce qui se passe à chaque étape.

1. **Déploiement en préproduction** :

Si un test unitaire échoue, l’application est déployée dans un environnement simulé de préproduction. Cela nous permet de corriger les problèmes avant tout déploiement en production.

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

Description générée automatiquement

1. **Déploiement en Production** :

Une image contenant texte, capture d’écran, ligne

Description générée automatiquementLorsque tous les tests passent et que le commit contient les bons objets (feat et web), le site web est déployé sur Azure en production.

1. **Déploiement avec un commit sans *feat* et *web***:

Si le commit ne contient ni *feat* ni web, aucun déploiement sur Azure n'est effectué. Le workflow s’arrête après la vérification des commits.

Une image contenant capture d’écran, texte

Description générée automatiquement

1. **Déploiement Déploiement avec un commit feat (modification du script) :**

Une image contenant texte, capture d’écran

Description générée automatiquementSi le commit contient *feat* mais pas *web*, seules les étapes de tests unitaires et de mise à jour de la base de données sont exécutées, sans déploiement en production.

***Résultat :***

Ce workflow structuré et automatisé garantit un déploiement sûr et efficace. En combinant les vérifications des commits, les tests unitaires et les déploiements conditionnels, nous assurons que l’application est toujours en bon état avant d’être mise à disposition des utilisateurs.

# Conclusion

## Résultats finaux et conclusion

Les résultats finaux de cette partie du projet consistent en un script nommé « unitTest.ps1 » permettant de tester le script « install\_in\_nutshell.ps1 », un site web permettant de télécharger le script « install\_in\_nutshell.ps1 » et un workflow GitHubAction nous permettons de :

* Vérifier les messages d’un commit afin d’activer ou non le déploiement du repository.
* Tester automatiquement le script « install\_in\_nutshell.ps1 ».
* Déployer le script dans la base de données si celui-ci passe les tests.
* Déployer le frontend & le backend du site web sur une Azure Web App Service afin de permettre au gens de télécharger le script « install\_in\_nutshell.ps1 ».

## Utilisation de l’IA

Bien que ce ne soit pas notre premier déploiement avec Azure Cloud, nous avons utilisé l’IA afin de nous aider à effectuer des taches jamais faites auparavant tel qu’insérer un fichier binaire dans un base de donnée depuis un workflow, l’intelligence artificiel nous a aussi aider à comprendre pourquoi certaines erreurs apparaissait dans le déploiement ou à mieux comprendre certains paramètres des ressource Azure utilisé, ainsi qu’a corriger les fautes d’orthographes et de syntaxe dans la rédaction de ce document.