

Sommaire

1	GENERALITES	3
1.1	Exigences de bases.....	3
1.2	Analyse des exigences et propositions.....	3
1.2.1	Gestionnaire de versions de code	3
1.2.2	Workflows runners: git runners versus hosted runners	3
1.2.3	Visibilité du repos	3
2	CONFIGURATIONS GENERALES.....	3
2.1	Structure du dépôt et organisation du code	3
2.1.1	.github/workflows – Définition des pipelines CI/CD.....	4
2.1.2	backEnd – API .NET + Tests unitaires	5
2.1.3	frontEnd – Application Angular	5
2.1.4	infra – Configuration d'environnement	5
2.1.5	docs – Documentation technique	5
2.1.6	.gitignore – Fichiers ignorés par Git.....	5
2.1.7	docker-compose.yml – Orchestration multi-conteneurs.....	5
2.1.8	README.md – Introduction au projet.....	6
3	BACKEND : ASP.NET CORE 8.....	6
3.1	Synthèse du projet.....	6
3.1.1	Framework utilisé : ASP.NET Core 8 (Minimal API).....	6
3.1.2	Langage : C#	6
3.1.3	Base de données :	6
3.1.4	Construction :	6
3.1.5	Image Docker :	6
3.1.6	Test local réussi :	6
3.2	Rendu du projet	6
4	FRONTEND : ANGULAR 21 + DOCKER.....	7
4.1	Synthèse du projet.....	7
4.1.1	Framework utilisé : Angular version 21.....	7
4.1.2	Version Node : 20 (compatibilité Angular 21)	7
4.1.3	Construction :	7
4.1.4	Image Docker :	7

4.1.5	Test local réussi :	7
4.2	Rendu du projet	7
5	WORKFLOWS CI.....	8
5.1	Workflow Backend CI	8
5.1.1	Objectif	8
5.1.2	Déclencheurs	8
5.1.3	Jobs	8
5.1.4	Étapes :.....	8
5.2	Workflow Frontend CI.....	8
5.2.1	Objectif	8
5.2.2	Déclencheurs	8
5.2.3	Jobs	9
5.2.4	Étapes	9
6	EXECUTIONS DU PROJET	9
6.1	Builder et Démarrer le projet (FrontEnd et BackEnd).....	9
6.2	Arrêter le projet.....	9
7	WORKFLOWS	10

1 GENERALITES

1.1 Exigences de bases

- des environnements de développement, test & production
- avec pipelines CI/CD pour continuellement merger, tester, et déployer le code
- je suis ouvert à tout, avec un petit budget, donc privilégier les outils open-source ou les tiers d'entrée
- propose moi des solutions, avec un calendrier de livraison, j'en ai besoin depuis un certain temps
- C# .net + ReactJs + Angular
- PostgreSQL + RabbitMQ + docker
- Git

1.2 Analyse des exigences et propositions

1.2.1 Gestionnaire de versions de code

Le projet sera hébergé sur GitHub. Un seul dépôt regroupera à la fois le code FrontEnd et BackEnd, afin de simplifier la gestion des workflows et de faciliter l'accès au code.

1.2.2 Workflows runners: git runners versus hosted runners

Nous privilégierons l'utilisation de GitHub runners afin de réduire les coûts, compte tenu de notre budget limité. L'option consistant à déployer nos propres machines comme runners aurait également été possible, mais elle s'avère plus onéreuse.

1.2.3 Visibilité du repos

Le choix de la visibilité du dépôt a un impact direct sur le coût du projet. Pour des raisons budgétaires, nous opterons pour un dépôt public. En effet, les dépôts privés disposent d'un quota limité concernant le temps d'exécution des workflows lorsque les runners utilisés sont les GitHub runners.

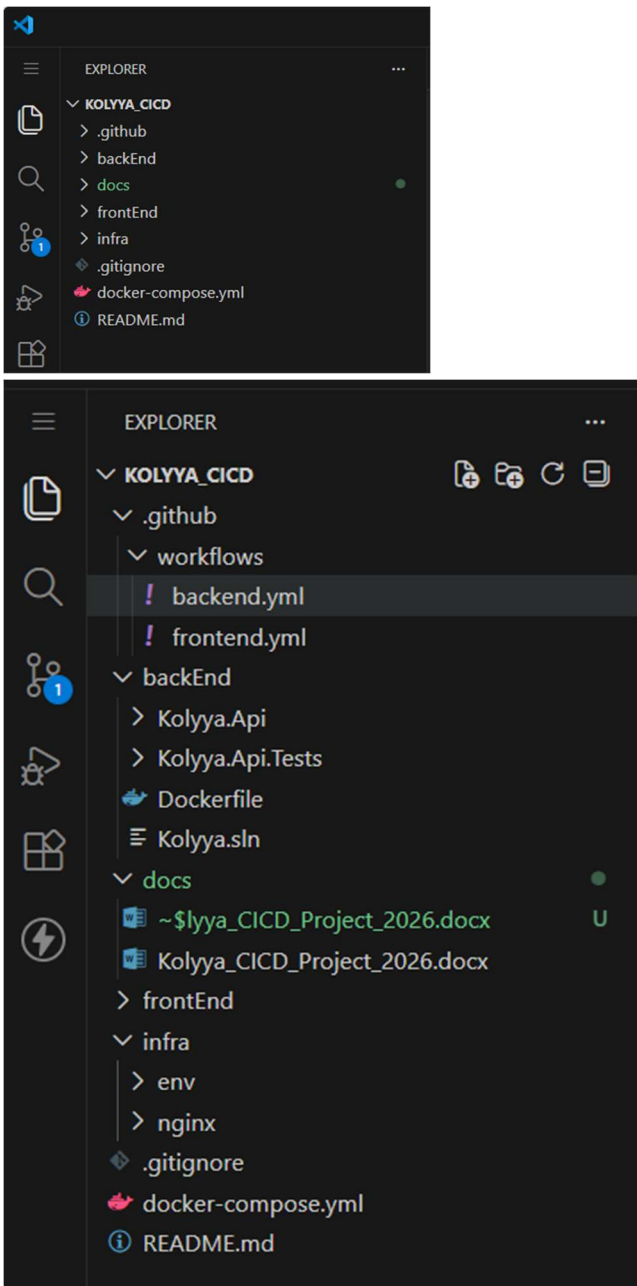
2 CONFIGURATIONS GENERALES

2.1 Structure du dépôt et organisation du code

Le dépôt GitHub est structuré de manière à bien séparer les différentes couches de l'application :

/.github/workflows	→ Contient les pipelines CI/CD
/backEnd	→ Contient l'API en .NET
/frontEnd	→ Contient les interfaces utilisateurs (ReactJS ou Angular)
/infra	→ Contient les scripts d'infrastructure (les variables d'Env, nginx si besoin, etc.)
/docs	→ Contient la documentation technique

Kolyya CICD Project



Le dépôt Git principal regroupe l'ensemble des composants du projet afin d'assurer une meilleure cohérence, visibilité et efficacité pour la mise en place de l'intégration et du déploiement continus (CI/CD).

2.1.1 .github/workflows – Définition des pipelines CI/CD

Ce dossier contient les fichiers YAML définissant les workflows GitHub Actions. Ces workflows sont déclenchés à chaque modification du code dans les branches principales (main, dev).

Le fichier backend.yml configure notamment une pipeline CI qui exécute les étapes suivantes pour le backend :

- Restauration des dépendances .NET
- Vérification du style de code (dotnet format)
- Exécution des tests unitaires via xUnit
- Vérification des vulnérabilités dans les dépendances
- Build de l'image Docker du backend

À terme, un second workflow pourra être ajouté pour le front-end avec les mêmes principes.

2.1.2 backEnd – API .NET + Tests unitaires

Le dossier backEnd contient deux projets .NET :

- Kolyya.Api : l'API principale développée en ASP.NET Core
- Kolyya.Api.Tests : le projet de tests unitaires utilisant xUnit, Moq et FluentAssertions
Ce projet référence directement Kolyya.Api.csproj pour effectuer des tests sur les classes de l'API (comme SampleItem).

Un fichier solution Kolyya.sln regroupe les deux projets, facilitant la compilation et les tests depuis la CI et les IDEs.

2.1.3 frontEnd – Application Angular

Ce dossier (prévu) contiendra l'application front-end du projet. Sa séparation claire du backend permet :

- d'organiser les builds spécifiques
- de configurer un pipeline CI/CD distinct
- de supporter facilement plusieurs frameworks JS (Angular)

2.1.4 infra – Configuration d'environnement

Le fichier dev.env présent dans ce dossier stocke les variables d'environnement nécessaires au backend, comme la connexion PostgreSQL :

POSTGRES_CONN=Host=kolyya.db;Port=5432;Username=postgres;Password=devpassword;Database=myappdb

Ce fichier est injecté dans Docker grâce à l'option --env-file de docker compose.

Des fichiers similaires pourront être créés pour les environnements test et prod.

2.1.5 docs – Documentation technique

Ce dossier contiendra toute la documentation du projet :

- guides d'installation
- architecture du système
- notes de développement
- consignes pour les développeurs

2.1.6 .gitignore – Fichiers ignorés par Git

Le fichier .gitignore est configuré pour exclure les fichiers inutiles ou sensibles du suivi Git :

- artefacts de compilation .NET (bin, obj)
- fichiers temporaires d'IDE (.vscode, .idea)
- secrets et variables d'environnement (*.env)
- dépendances front-end (node_modules)

2.1.7 docker-compose.yml – Orchestration multi-conteneurs

Le fichier docker-compose.yml permet de démarrer l'environnement de développement local avec :

- l'API backend (port 5000 → 8080 exposé)
- la base de données PostgreSQL (port 5432)

Ce fichier est essentiel pour reproduire un environnement cohérent sur chaque machine, mais aussi pour les phases de test dans les workflows CI/CD.

2.1.8 README.md – Introduction au projet

Le fichier README.md (à enrichir) permet de :

- présenter rapidement le projet
- expliquer comment lancer les services en local avec Docker
- documenter la structure des dossiers
- inclure les badges de CI/CD et des liens vers les outils externes

3 BACKEND : ASP.NET CORE 8.

3.1 Synthèse du projet

3.1.1 Framework utilisé : ASP.NET Core 8 (Minimal API)

3.1.2 Langage : C#

3.1.3 Base de données :

- PostgreSQL version 16
- Connexion configurée via la variable d'environnement `ConnectionStrings__DefaultConnection`
- Dépendance explicitement définie dans `docker-compose.yml`

3.1.4 Construction :

- Le projet est compilé avec `dotnet build` puis publié avec `dotnet publish`
- Utilisation d'un Dockerfile adapté aux projets ASP.NET Core :
 - `mcr.microsoft.com/dotnet/sdk:8.0` pour la compilation
 - `mcr.microsoft.com/dotnet/aspnet:8.0` pour l'image finale

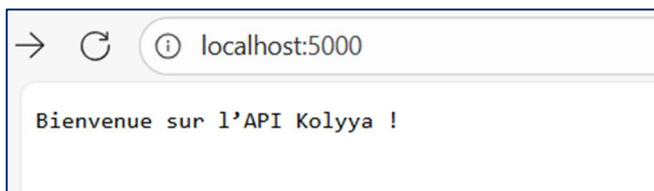
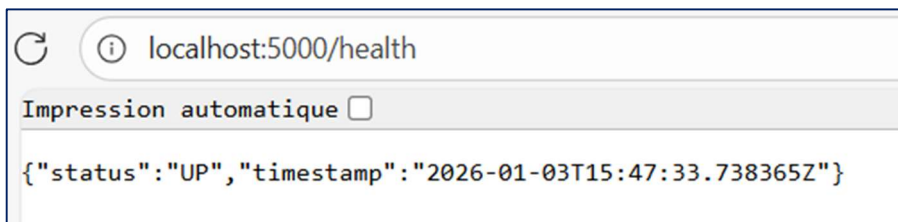
3.1.5 Image Docker :

- Multi-stage build pour optimiser la taille
- L'API est exposée sur le port 8080 dans le container

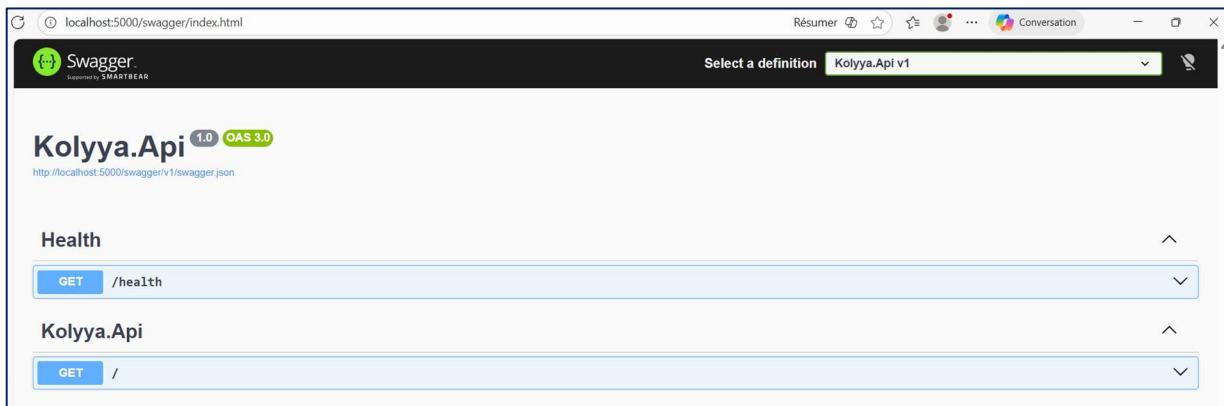
3.1.6 Test local réussi :

- Lancement via `docker-compose up --build`
- L'API est accessible sur `http://localhost:5000` (port mappé via `5000:8080`)
- Connexion établie avec la base de données PostgreSQL (port 5432)

3.2 Rendu du projet



Kolyya CICD Project



4 FRONTEND : ANGULAR 21 + DOCKER

4.1 Synthèse du projet

4.1.1 Framework utilisé : Angular version 21

4.1.2 Version Node : 20 (compatibilité Angular 21)

4.1.3 Construction :

- L'application est compilée avec ng build
- La sortie est générée dans dist/kolyya-frontend/

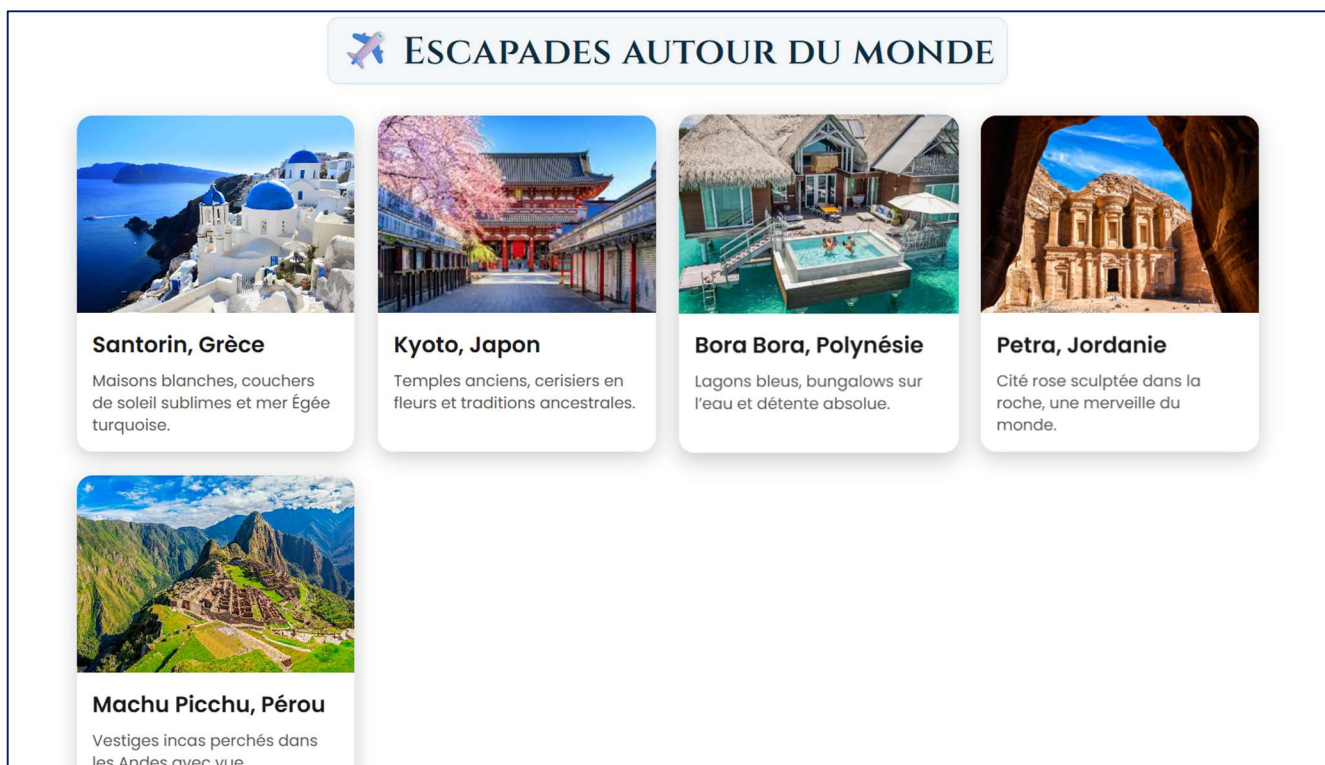
4.1.4 Image Docker :

- Construite avec un Dockerfile multi-stage (Node + Nginx)
- Le build Angular est copié dans /usr/share/nginx/html

4.1.5 Test local réussi :

- Lancement via docker run -p 3000:80 kolyya-frontend
- Résultat visible sur http://localhost:3000 (accès confirmé au frontend)

4.2 Rendu du projet



5 WORKFLOWS CI

5.1 Workflow Backend CI

5.1.1 Objectif

Ce workflow est conçu pour automatiser les tâches de build, test, linting et analyse de code du backend .NET dès qu'un changement est détecté.

5.1.2 Déclencheurs

Push ou pull request sur les branches `main` ou `dev`

Si des fichiers changent dans :

`backEnd/**` (code backend)

`infra/**` (infrastructure)

`docker-compose.yml`, `.env`, ou ce fichier workflow

5.1.3 Jobs

`build-and-test`

S'exécute sur `ubuntu-latest`

Définit des variables d'environnement :

`DOTNET_VERSION: "8.0.x"`

`IMAGE_NAME: kolyya-backend`

5.1.4 Étapes :

- Checkout du dépôt (`actions/checkout`)
- Setup .NET SDK (`actions/setup-dotnet`)
- Restauration des dépendances avec `dotnet restore`
- Linting (vérification du style de code) avec `dotnet format`
- Tests unitaires avec `dotnet test`

5.2 Workflow Frontend CI

5.2.1 Objectif

Ce workflow GitHub Actions vise à automatiser le processus de **CI (intégration continue)** pour l'application **frontend Angular**. Il permet de :

- Vérifier la qualité du code avec `ng lint`.
- Exécuter les **tests unitaires avec Vitest**.
- Construire l'application Angular.
- Construire l'image Docker correspondante.
- (Optionnellement) préparer l'image à être poussée sur Docker Hub, avec un **tag dynamique** basé sur le commit.
-

5.2.2 Déclencheurs

Le workflow est déclenché automatiquement :

Lors de **push** ou **pull request** sur les branches `main` ou `dev`.

Kolyya CICD Project

Si ces changements affectent l'un des fichiers ou dossiers suivants :

```
frontEnd/**
infra/**
docker-compose.yml
.env
.github/workflows/frontend.yml
```

5.2.3 Jobs

Le workflow contient un **unique job** nommé `build-test-dockerize`, qui s'exécute sur un environnement Linux (`ubuntu-latest`).

5.2.4 Étapes

- **Checkout du code source :**

Utilise l'action officielle `actions/checkout@v3` pour récupérer le dépôt.

- **Configuration de Node.js 20 :**

Utilise `actions/setup-node@v3` pour installer Node.js version 20, compatible avec Angular 21 et Vitest.

- **Installation des dépendances :**

Exécute `npm ci` pour une installation propre basée sur le `package-lock.json`.

- **Linting (analyse statique) :**

Exécute `npm run lint` pour vérifier la qualité du code selon les règles Angular définies dans le projet.

- **Build Angular :**

Compile l'application en mode production avec `npm run build`.

- **Build Docker image :**

Construit une image Docker locale avec un tag dynamique basé sur le `GITHUB_SHA` (commit) :

```
docker build -t kolyya-frontend:${GITHUB_SHA::7} .
```

6 EXECUTIONS DU PROJET

6.1 Builder et Démarrer le projet (FrontEnd et Backend)

`docker-compose --env-file infra/env/dev.env up --build`

```
PS C:\Users\Ousseynou\Documents\Kolyya_CICD> docker-compose --env-file infra/env/dev.env up --build
time="2026-01-03T10:37:33-05:00" level=warning msg="C:\\Users\\Ousseynou\\Documents\\Kolyya_CICD\\docker-compose.yml: the attribute `version` is obsolete, it will be
ignored, please remove it to avoid potential confusion"
[+] Running 11/15
 - db [██████████] 36.7MB / 160.1MB Pulling 69.7s
  ✓ 592e7afa2092 Download complete 2.2s
  ✓ be56d59d6531 Download complete 0.8s
  ✓ ff1a91754d0b Download complete 2.3s
  ✓ 08375b6eb817 Download complete 11.7s
  ✓ 19a4bd73b5b6 Download complete 2.8s
  ✓ 2809b03b8187 Download complete 1.6s
  - 74ee42a72ba8 Downloading [=====>] 13.63MB/113.1MB 67.9s
  ✓ 3efabc308a47 Download complete 2.5s
  - 02d7611c4eae Downloading [=====>] 9.437MB/29.78MB 67.9s
  ✓ 5773fb4a9d72 Download complete 2.9s
  - c3ce0464f73e Downloading [=====>] 5.243MB/8.204MB 67.9s
  ✓ 31172abd8f4a Download complete 2.5s
  ✓ 6b79f2b29884 Download complete 35.8s
  ✓ 7f53036964f1 Download complete 16.7s
```

6.2 Arrêter le projet

`docker-compose down -v`

7 WORKFLOWS

Actions

New workflow

All workflows

Backend CI

Frontend CI

Management

Caches

Attestations

Runners

Usage metrics

Performance metrics

All workflows

Showing runs from all workflows

Filter workflow runs

40 workflow runs

Event Status Branch Actor

Comment for workflow tests

Backend CI #6: Commit 6dc8972 pushed by nabalma

main

2 minutes ago

1m 2s

...

Comment for workflow tests

Frontend CI #26: Commit 6dc8972 pushed by nabalma

main

2 minutes ago

1m 0s

...

vitest is now out , asking for a config

Frontend CI #25: Commit cb47ce2 pushed by nabalma

main

16 minutes ago

1m 11s

...

Lint Errors correction

Frontend CI #24: Commit f1b56ce pushed by nabalma

main

26 minutes ago

20s

...

ESLint installed

Frontend CI #23: Commit 14681c0 pushed by nabalma

main

32 minutes ago

20s

...