ESIREM - 4A - ILC/SQR

Module de CI/CD

Examen pratique : TP Projet

Novembre 2023

Le projet noté du module de CI/CD évaluera compétences et bonnes pratiques de développement vues en cours. La notation tiendra compte des fonctionnalités déployées dans l'API, de la bonne mise en place des points d'exigences projets ainsi que de la collaboration entre les membres du groupe.

1 Exigences générales projet

Le projet est à rendre pour le *20 Décembre 2023 à 23h59*
À partir de cette date, **aucune modification** du dépôt ou de code ne sera prise en compte.

1.1 GitHub

- Un dépôt GitHub dédié au projet, auquel vous m'aurez ajouté en tant que collaborateur.
- L'historique des changements sur le dépôt devra montrer la collaboration entre les membres du groupe (changement de sources différentes).
- Automatiser le build d'image dans des GitHub Actions.
- Le dépôt devra être documenté via README et Swagger.

1.2 Docker

- Pousser au moins trois images différentes sur le Registre d'image. (GCR)
- L'image générée par le Dockerfile peut-être exécutée via docker run.

1.3 Documentation

- Un README principal contenant au minimum : **sujet du projet**, **membres du groupe**, les **technologies utilisées**, des **badges** des résultats des builds des dernières CI exécutées et la **procédure** pour exécuter l'API.
- Un README supplémentaire pour chaque dossier décrivant succinctement son contenu.
- Un fichier Swagger valide au sens de l'éditeur Swagger. (cf. *https://editor.swagger.io/*)

Toutes informations supplémentaires sur le fonctionnement des endpoints et de l'API seront les bienvenues et valorisées.

2 Sujet

Objectif : Créer une API Flask pour de la gestion CRUD d'un calendrier.
Langage : *Python*.

Définissons un évènement comme étant un tuplet (T1, t, p, n), où **t** est égal au temps de l'évènement en seconde à partir du timestamp **T1** (date et heure du début de l'évènement) et **p** la liste des personnes participant à cet évènement. **n** sera le nom de l'évènement.

2.1 Réaliser une première version de l'API REST

En utilisant Flask, réaliser une première version de l'API. Voici une liste des actions (aussi appelées routes) qui doivent être mises à la disposition via un appel HTTP sur API:

E1 - Créer un évènement.

E2 - Afficher une liste de tous les événements dans l'ordre chronologique.

E3 - Afficher une liste de tous les évènements dans l'ordre chronologique liées à une personne.

E3 - Ajouter un participant à un évènement.

E5 - Afficher le détails du prochain cours.

E6 - Importer des données depuis un fichier csv. (à documenter)

2.2 Documenter l'API via des READMEs et un fichier Swagger

- Documenter et justifier vos choix de développement dans le fichier README.md.
- Ajouter le détail pour une procédure de chargement de données dans l'API à partir d'un fichier .cvs dans le dépôt et du endpoint E6.
- Détailler les routes dans un fichier **swagger.yaml** valide pour *https://editor.swagger.io/*

2.3 Préparer l'intégration continue (CI)

Créer trois github actions :

- Une déclenchée à chaque changement pour builder l'application.
- Une déclenchée manuellement pour utiliser le fichier Dockerfile pour créer une image.
- Une déclenchée pour chaque tag **semver** pour utiliser le fichier Dockerfile pour créer et pousser l'image de l'API avec en tag la version **semver** spécifiée.

2.4 Anticiper le déploiement continue (CD)

Vous allez maintenant automatiquement publier les nouvelles versions dans un registre de conteneur Google (GCR).

Pour pousser l'image créé sur le registre GCP, ajoutez à la suite du workflow déclenché pour chaque nouveau tag, les **steps** disponibles dans le fichier à l'adresse suivante :

JeromeMSD/module_ci-cd/.github/workflows/Docker_push_GCR.yaml

Modifiez le paramètre **tags** de l'étape "Build and push Docker images" pour y mettre la valeur :

`gcr.io/esirem/4A_[ ilc ou sqr ]/[nom1_nom2]/[nom_du_projet]`:${{ github.ref_name }}

Modifiez les paramètres **file** et **context** de l'étape "Build and push Docker images" et remplacez les éléments entre crochets dans l'URL pour y mettre les bonnes valeurs suivant votre projet.

2.5 Top départ

Déployez **une première release publique** de l'API via GitHub avec un tag correspondant à la bonne version sémantique.

2.6 Améliorer l'API

Pour chacune des fonctionnalités suivantes, faites une release de votre API avec le numéro de version adapté.

Ajoutez maintenant une **description** aux évènements dans le modèle: (T1, t, p, n, d), où **d** est une courte description de l'évènement. Les endpoints de lectures doivent prendre en compte ce changement.

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

Ajoutez une route pour obtenir le nombre total de temps passé dans des évènements pour une personne sur la journée, sur sept jours et sur un mois.

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

Corriger la fonction de calcul du temps total pour ne prendre en compte que les évènements où il y a **plus d'une personne.**

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.

Ajouter une route qui permet de calculer le temps restant avant une date ou un évènement.
(exemple: temps restant avant 25 décembre 2023)

> Déployez une release publique de l'API à ce stade via GitHub avec un tag équivalent à la bonne version sémantique.