ESIREM - 4A - ILC/SQR

CI/CD MODULE

PRACTICE EXAM : PROJECT

January 2023

Practice exam for CI/CD module will evaluate skills and development good practice see in class. The final grade will be based on API's functionality, meeting the project requirements and collaboration within developper pair.

1 Project requirements

End of project is set at *February 1, 2023 at 11:59 pm*
No changes on repository or in the code will be considered after this date.

1.1 GitHub

- A GitHub repository dedicated to the project, you shall add me as collaborator.
- Change history have to display your collaboration with your binomial (two source of commit).
- Automate your repository with GitHub actions.
- Repository should be documented by README.

1.2 Docker

- Push at least 3 different images on the container registry (GCR).
- Image generated from Dockerfile should be runnable using docker run.

1.3 Documentation

- A main README with at least : **project topic**, **binomial members**, **used languages**, **status badges** with last workflows outputs (passed/failed) and a explaination to how to start the project on a computer.
- Add a README for each folder to describe the folder content.
- Swagger file, valide when put in Swagger Editor. (cf. *https://editor.swagger.io/*)

Every additionnal informations on endpoints treatment, API architecture or goals are welcomed.

2 Topics

2.1 Guided topic : A clear path

Goal : Build a Flask API to manage a money transaction system (CRUD).
Language : *Python* (need to be precise in README) you can choose another language.

Say we define a **transaction** as a tuple (P1, P2, t, s), where **s** is the amont of money transferred form person **P1** to person **P2** at **t** time.

2.1.1 Release a first API version

Using Flask, build a first API version with the following functionality. En utilisant Flask, réaliser une première version de l'API. Voici une liste des actions (aussi appelés routes ou endpoints) qui doivent être mises à la disposition via un API HTTP par API:

F1 - Add a tansaction.

F2 - List all transactions in chronological order

F3 - List all transactions in chronological order link to someone

F4 - Display total of money hold bu someone.

F5 - Import transactions from a csv file. (should be documented)

2.1.2 Document using READMEs and a Swagger file

- Document and explain your choice of topic in README.md.
- Add details about how to load data using from a csv file
- Describe endpoints using Swagger file. cf. *https://editor.swagger.io/*

2.1.3 Prepare continuous intégration (CI)

Code three Github Actions :

- One triggered at every change to build the API (can be only dockerise if you use python).
- One manually triggered to build, dockerise and push builded image to GCR.
- One trigger on semver tags to build, dockerise and push builded image to GCR with semver tag.

2.1.4 Prepare continuous deployment (CD)

Now you will automatically publish new version to a container registry at Google (GCR). In the manually and tag triggered workflow, use the **job** and **environment variable** from the following url to use docker to push your version to GCR : *https://github.com/JeromeMSD/module_ci-cd/blob/main/.github/workflows/Docker_push_GCR.yaml*

Change **tags** parameter in "Build and push Docker images" step to set :

gcr.io/esirem/4A_[ILC ou SQR]/[NOM1_NOM2]/[nom_du_projet]/:${{ github.ref_name }}

Change **file** and **context** parameters in "Build and push Docker images" step to match your project files.

2.1.5 First adventure

Release now your first public version of your API using Github interface with the right semver tag.

2.1.6 Improve API

For each of the following functionalities, release a new API version with the right semver.

Add a hash in the transaction model to match (P1, P2, t, s, h) where **s** is the amont of money transferred form person **P1** to person **P2** at **t** time and **h** is the hash of P1, P2, s. Document your hash function choice in your README.

> Release a version with the right semver tag.

Add the new funitionality : Integrity check of tuple using hash calculation which compare sended data and hash to stored data and hash.

> Release a version with the right semver tag.

Fix hash calculation to take account of t : the tranfert date.

> Release a version with the right semver tag.

2.2 free Flask topic : A bit of choice

Goal : **By meeting project requirements in page 1**. Build an Flask API to CRUD manage a personal theme (ex: bakery, parking, film, series...)
Langage : Python is recommended, but you can choose another language.

2.2.1 API REST

Define at least one tuple of data, explain your model in README file.
Implement a API REST that allow to manage multi instance of your tuple, you should code every actions of CRUD model (Create, Read, Update, Delete).

2.2.2 Document using READMEs and a Swagger file

- Document and explain your choice of topic in README.md.
- Add details about how to load data using from a csv file
- Describe endpoints using Swagger file. cf. *https://editor.swagger.io/*

2.2.3 Prepare continuous intégration (CI)

Code three Github Actions :

- One triggered at every change to build the API (can be only dockerise if you use python).
- One manually triggered to build, dockerise and push builded image to GCR.
- One trigger on semver tags to build, dockerise and push builded image to GCR with semver tag.

2.2.4 Prepare continuous deployment (CD)

Now you will automatically publish new version to a container registry at Google (GCR). In the manually and tag triggered workflow, use the **job** and **environment variable** from the following url to use docker to push your version to GCR : *https://github.com/JeromeMSD/module_ci-cd/blob/main/.github/workflows/Docker_push_GCR.yaml*

Change **tags** parameter in "Build and push Docker images" step to set :

gcr.io/esirem/4A_[ILC ou SQR]/[NOM1_NOM2]/[nom_du_projet]/:${{ github.ref_name }}

Change **file** and **context** parameters in "Build and push Docker images" step to match your project files.

2.2.5 First adventure

Release now your first public version of your API using Github interface with the right semver tag.

2.2.6 Improve API

For each of the following functionalities, release a new API version with the right semver.

1. Add and document a new endpoint to load data from a csv file.
2. Add and save a hash of an instance of your modèle. Document your hash function choixe in in the README.md file.
3. Add and document **breaking changes**, **changes** and **fixes** and release version with SemVer (x.y.z) tags for each of them.