* le code source (qui respecte votre cahier des charges)

Lien github, <https://github.com/jramassamy/cloud_project>

* Documentation de votre api avec OpenApi3

Accessible via l’url <http://localhost:3000/swagger/>

* Les tests (postman)
  + Disponible dans le dossier Postman
  + Nécessite de récupérer le JWT Token avant de faire les tests sur certaines requêtes
  + Informations mises dans le Readme
* Un jeu de données qui initialise votre BDD avec quelques exemples
  + BDD hébergée dans le cloud, données accessibles directement.
* Un Dockerfile pour créer votre image docker
  + Dockerfile dans le front
  + Dockerfile dans le back
  + Docker-compose à la racine
* Un docker compose pour lancer votre application en local
  + A la racine du dossier
* Un Readme pour expliquer comment faire fonctionner votre appli (avec et sans docker)
  + Sans Docker :
    - Front, aller dans le dossier front, npm install && npm start
    - Back, aller dans le dossier front, npm install && npm run build && npm start
  + Avec Docker, à la racine :
    - docker-compose build
    - docker-compose up

Front disponible sur le port d’écoute, Back disponible sur le port 3000.

* (Optionnel) les fichiers kubernetes pour déployer votre application sur un cluster (deployment, service, ingress)

In Progress

* (Optionnel) déployer votre application sur le cluster kubernetes

In Progress

* Un petit fichier qui explique vos difficultés et vos choix d’api quand ils sont complexes
* Découplage du projet via l’utilisation d’une API & d’un Front
* API Rest en Typescript via Node.js
* Front-End utilisant Angular (Typescript)

**FRONT**

Choix du Framework Front pour son côté Orienté Objet, l’utilisation de Typescript nous facilite l’implémentation de ce concept.

Code facilement maintenable, modulaire à l’aide de « composants ».

Cela implique : Réutilisabilité, Lisibilité, Maintenabilité

**BACK**

Choix de l’utilisation de Typescript pour aller de pair avec le développement Orienté Objet que nous apporte la surcouche du language, qui par la suite est Transpilé en Javascript.

**BASE DE DONNEES**

**MongoDB**, pour sa flexibilité, tout comme Node.js, facile d’implémentation & d’utilisation (hébergé dans le Cloud)

* Utilisation de l’ODM Mongoose pour ajouter plus de la validité à l’insertion des données en base.
* Cela nous permet d’avoir JSON Schema intégré directement.

Difficulté :

* **Implémentation du CRUD**
  + Moins de lisibilité
  + Ajoute + de complexité à l’utilisation qu’autre chose pour du NoSQL
  + Pattern beaucoup plus complexe à mettre en place en NoSQL
* **Choix technique pour pallier à ce problème**
  + Endpoints explicite, « register », « login » & traitement des données insérés via le Body ou le Header, beaucoup plus simple pour de l’orienté document, architecture qu’utilise les Base de données NoSQL, dont MongoDB.
* **Refactoring d’une entité complexe**
  + Implique d’énormes changement dont les effets de bord ne génère pas mal de problème
* **Choix technique pour pallier à ce problème**
  + Il aurait fallu faire un refactoring du code avec une implémentation N-Tiers ou Domain Driven Development, 2 concepts qui demandent énormément d’apprentissage.