Organisation du travail

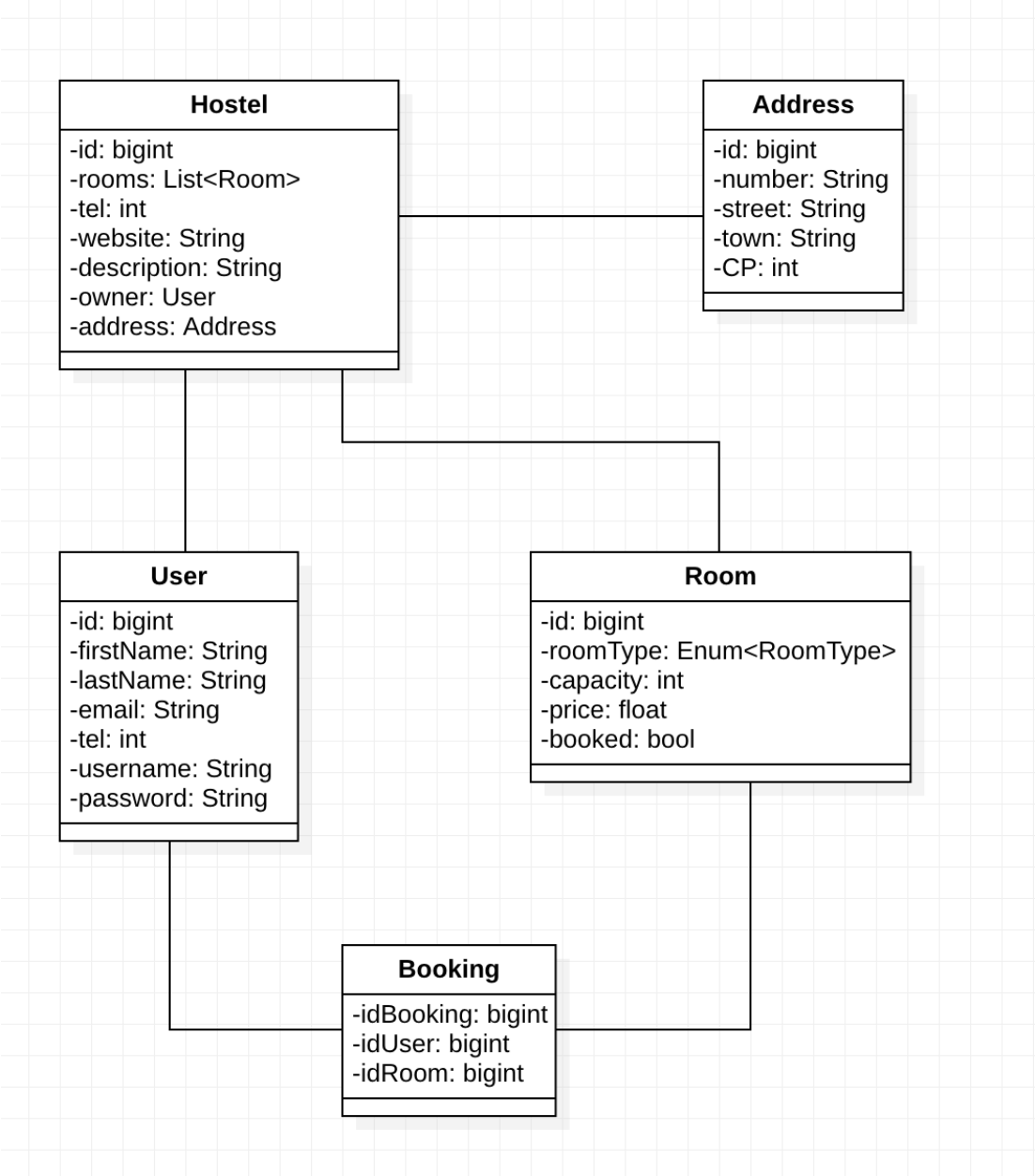
Objectifs :

- réalisation d'une application web en microservices sur une infrastructure en cluster
- Contrainte majeure : déploiement continu de votre travail

Cahier des charges :

- Un système de réservation pour tous les hôtels et canaux de ventes
    - Téléphone
    - Accueil
    - Site web
    - App mobile
        -
- Besoins du client :
    - 2 hôtels:
        - 5 / 3 chambres
    - Chaque hôtel à son site web
    - Système de réservation en commun aux deux hôtels (API de Réservation)
    - Possibilité d’ajouter
        - de nouveaux hôtel
        - de nouvelles chambres dans hôtel existant
        - de modifier la politique de prix

Architecture de DB



- Ajout d'une table prix
- Ajout d'une table la politique de prix à faire sur le diagramme (2 tables ?)
- Ou alors ajout d'une table politique de prix avec un champ isDefault qui définirait les politiques de prix par défaut dans le cas où on n'applique aucune politique de prix spécifique

- Type de chambre (enum)
    - SR : Suite présidentielle
    - S : Suite
    - JS : Junior suite
    - CD : Chambre de luxe
    - CS : Chambre standard

Politique de prix

- Des règles de prix sont à appliquer.

- Le client est apte à modifier ces règles de prix
- Modification des règles de prix sur
    - Chambres
        - toutes les chambres
        - un ou plusieurs hôtels en particulier
        - Un ou plusieurs type de chambre
        - Une ou plusieurs chambres
    - Période
        - Semaine / Mois
        - Saison
        - Jour ou série de jours spécifique
- /!\ Si la politique de prix est changer, lorsque cette ci est échus (fin de saison par exemple) la politique de prix initiale doit reprendre la main /!\
- Prévoir de pouvoir modifier la politique de prix initial

Problématiques autres

- Une réservation -> Confirmation par email
- Gestion de réservations en simultanées (asynchrone ? RabbitMQ ?)
    - Incluera probablement une API tierce
- POC sur la CI/CD
    - à terme, commit poussé dans une branche précise, déclencher le pipeline d'intégration et déploiement continu
- Cluster Docker-Swarm
    - multi-tenant (???)
    - multi-environnements
        - pré-prod
        - prod
- API en microservices (probablement plusieurs API)
    - Technos libres avec justification des choix
- Schéma technique d'architecture
- Documentation
- Répartition des fonctionnalités en services
- Tests unitaires
- Gestion des Logs

Technologies obligatoires

- Docker
- Gitlab / Gitlab CI
- Ansible

Ce qu'il va falloir mettre en place

- Installation docker via ansible **(yoann / mathieu)**
- Installation de Gitlab & pipelines via ansible **(yoann / mathieu)**
- Mise en place de la CI / CD **(yoann / boris)**
- Une architecture docker-swarm comprenant les micro services (docker) **(alex / mathieu)**
    - Docker Registry

    - TICK Stack
- Une base de données (MySQL) **(alex / boris)**
    - Diagramme de classes
- Une API **(alex / mathieu / yoann / mathieu)**
    - User API : login / register / reset password / account
    - Booking API : get room / bookroom / cancel room / update room
    - Management : CRUD Hotel / Room / politique pricing
- Doc **(mathieu / yoann)**
- Mise en place tests unitaires **(alex / mathieu / yoann / mathieu)**
- Mailer **(boris / alex)**
- Déploiement de Traefik **(mathieu / boris)**
- Un playbook Ansible comprenant de multiples rôles :
    - Docker Swarm **(boris / alex)**
    - Docker registry **(alex / yoann)**
    - Gitlab **(yoann / mathieu)**
    - ELK (Elastick Search, Kibana, Log Stash) **(boris / mathieu)**
    - TICK stack (**boris/alex**)
- Relance auto playbook KO **(yoann / alex)**

Justification des technologies

Python : Nous avons choisis le langage Python, car c'est un langage que nous connaissons tous au sein du groupe. L'objectif n'étant pas de se concentrer sur la partie microservices et déploiement, il était nécessaire que nous utilisions un langage commun à notre groupe.

FastAPI: nous avons choisi le framwork FastAPI pour développer les différentes API en microservices car ce framework mets à disposition un ensemble d'outil intéressant pour ce projet (swagger, sécurité des routes, authentification normal et OAuth2, intégration de pydantic permettant le typage et la levée d'erreur).

MySQL : Nous n'avons aucune gestion de document json à faire, de plus l'ensemble des tables a un lien relationnel. Une base de données relationnelle nous permettra de construire un modèle plus robuste et moins permissif.

Infrastructure & Architecture

Voici le premier schéma d’architecture comprenant les 3 serveurs ainsi que les services statiques sur leur serveur dédié.