
Dernière modification: 19 avril 2022

INF3995: Projet de conception d'un système informatique
Hiver2022
Documentation du projet

# Système aérien d'exploration

Documentation du projet répondant à l'appel d'offres no. H2022-INF3995 du département GIGL

Équipe No. 105

Éloïse Brosseau

Samuel Crosilla

Steven Duchêne

Félix Gauthier

Juliette Morin

Théo St-Denis

# Table des matières

# 1 Vue d'ensemble du projet

## 1.1 But du projet, porté et objectifs (Q4.1)

Le but de ce projet est de concevoir un système d'exploration pour un essaim de drones miniatures. Pour ce faire, il nous faudra développer un client Web qui communiquera non seulement avec les drones, mais aussi avec un environnement de simulation pour ces derniers. De plus, nous développerons nos capacités de gestion de projet. Plusieurs artefacts devront être réalisés afin d'atteindre nos objectifs. Ces derniers seront produits tout au long de la session. Tout d'abord nous répondrons à l'appel d'offres en présentant un prototype préliminaire pour démontrer notre compréhension de la problématique et notre maîtrise du sujet. Toutefois, avant de produire le prototype nous dévoilerons notre plan afin d'organiser le projet. Nous planifions remettre trois livrables incluant celui-ci. Le deuxième livrable détaillera complètement l'organisation du projet. Finalement, le dernier livrable inclura toutes les fonctionnalités complétées. Ce dernier sera présenté par l'équipe à l'Agence spatiale de Poly et sera remis sous forme d'un rapport exhaustif. De plus, lors de chaque livrable, nous produirons des vidéos qui démontreront que les fonctionnalités de notre produit satisfont les différents requis. Ce projet se déroulera tout au long de la session d'hiver 2022 et se terminera le 19 avril 2022.

## 1.2 Hypothèse et contraintes (Q3.1)

Tout d'abord, il est nécessaire de comprendre que nous sommes dans le contexte d'une session universitaire et que notre charge de travail ainsi que notre capacité à travailler sur le projet varient au cours des semaines à cause des remises, laboratoires, examens, etc. Cependant, plusieurs types de dates de remise seront établies. Nous avons bien sûr les remises fixes (PDR, CDR et RR), mais des dates de remises hebdomadaires à l'interne seront implémentées pour chaque requis à réaliser afin d'assurer un respect maximal de l'échéancier établi. Il est possible que celles-ci varient de quelques jours avant ou après le jour en question, mais l'équipe fera tout en son possible pour les respecter. Cela dépendra de la disponibilité des membres de l'équipe au cours de la session. Nous tenterons bien évidemment de respecter au maximum ces dates précises, mais il faut comprendre qu'il pourra y avoir de légers changements. On assume que les membres de l'équipe contribueront à leur capacité maximale. Cependant, il est toutefois possible que certaines personnes réalisent une plus grande partie du travail, car nous n'avons pas tous les mêmes expériences, ni les mêmes forces en informatique. Aussi, comme la taille de notre équipe est relativement limitée, certains membres pourront avoir plus d'un rôle afin de mener à bien le projet. Par exemple, il est possible qu'une même personne occupe parfois le rôle de coordonnateur de projet et que d'autre fois, il occupe le poste d'analyste-développeur. De plus, nous supposerons qu'il n'y aura pas de changements dans le projet et que les exigences qui nous ont étés présentés en classe ne changeront pas au cours de la session. Cependant, certains requis facultatifs pourraient toutefois être retirés du projet si l'équipe juge que leur réalisation ne respecterait pas le temps imparti à la réalisation de tâches obligatoires. Également, nous supposerons qu'il n'y a pas de frais d'assurances à prendre en compte dans le projet et que seul les frais reliés à la main d'oeuvre et le développement seront considérés.

### 1.3 Biens livrables du projet (Q4.1)

Premièrement, nous produirons les trois jalons principaux, soit le PDR, le CDR et le RR. Ces derniers comprendront chacun plusieurs artéfacts. Pour la réponse à l'appel d'offres, nous réaliseront plusieurs diagrammes qui nous permettront de schématiser les différentes composantes du projet. Les cinq composantes nécessitant un diagramme sont l'architecture logicielle, la station au sol, le logiciel embarqué, la simulation et l'interface utilisateur. Il sera parfois nécessaire d'utiliser plus qu'une figure afin de bien représenter nos idées et concepts. Aussi, pour nous aider à schématiser rapidement le client, nous produirons une maquette du site web. Cela donnera une idée du produit final sans demander trop de temps. Elle sera réalisée à l'aide de la plateforme *Figma* [5]. Cette dernière est un outil collaboratif efficace pour la création de l'interface utilisateur. Nous produirons également les produits logiciel et physique nécessaires à la réalisation du projet, c'est-à-dire un client web, une station au sol, un système embarqué pour les drones ainsi qu'un simulateur fonctionnant sur Argos3. Un prototype de ces produits sera remis le 4 février afin de démontrer notre compréhension ainsi que nos compétences sur le sujet. Ensuite, un système avec fonctionnement partiel ainsi que la documentation présentant notre concept complet sera remis le 18 mars. Une présentation qui aura lieu le 21 mars permettra de démontrer l'avancement du projet ainsi que les ajustements qui ont été apportés depuis le PDR. Ces produits seront entièrement terminés d'ici le 19 avril, à temps pour le Readiness Review. Nous produirons également plusieurs artéfacts qui ne seront pas directement reliés à la gestion de projet. En effet, nous produirons un calendrier qui indiquera les dates limites assignées à l'interne. Nous produirons plusieurs sprints ayant une durée de 2 semaines. Ceux-ci seront situés sur Gitlab afin d'aider l'équipe de développement. Chaque sprint sera composé de plusieurs tâches ayant chacune des objectifs à compléter afin de pouvoir être considéré comme étant terminé. Aussi, nous ferons un diagramme indiquant le temps alloué pour chaque tâche afin de pouvoir planifier à long terme et ainsi estimer le coût du projet.

## 2 Organisation du projet

### 2.1 Structure d'organisation (Q6.1)

Comme mentionné dans la section d'hypothèse, le fait que nous ayons une équipe de petite taille nous forcera à occuper plus d'un rôle. Ainsi, nous aurons un coordonnateur de projet qui veillera à ce que les tâches et les livrables soient terminés à temps. Cependant, cette personne devra occuper plus d'un rôle afin de mener à bien le projet. Nous avons séparé le projet en 3 parties distinctes. Tout d'abord, une partie de l'équipe s'occupera de la partie application du projet. C'est-à-dire qu'ils auront pour tâche tout ce qui est relié au client, au serveur ainsi qu'à la base de données. Cette équipe sera donc composée de 2 développeurs full-stack. Une de ces deux personnes sera également responsable de communiquer les avancements et les imprévus au coordonnateur de projet. Pour la partie des drones et du système d'exploration, nous aurons une équipe de quatre personnes séparée en deux sous-équipes. Comme il y a une partie en commun entre les drones et la simulation, ces deux équipes travailleront en collaboration lorsque nécessaire. De plus, nous avons considéré que la partie du client web et la base de données prendrait moins de temps, alors les membres de cette sous-équipe contribueront également à la simulation et à la partie embarquée des drones. Les membres de l'équipe web et base de données sont Félix Gauthier et Juliette Morin, ceux de l'équipe simulation sont Éloïse Brosseau et Steven Duchêne et finalement Samuel Crosilla et Théo St-Denis font partie de l'équipe systèmes embarqués. Il faut cependant garder en tête que tous les membres

de l'équipe vont s'aider même s'ils ne font pas partie de la même sous-équipe.

### 2.2 Entente contractuelle (Q11.1)

Ici, nous soumissionnons pour un contrat de type clé en main. Ainsi, nous garantissons la remise d'un produit final à une date fixe, soit le 19 avril. Quant à lui, le prix du contrat sera fixe et nous vous garantissons qu'il n'y aura pas de frais supplémentaires. En effet, dans ce rapport nous vous fournirons les frais reliés à ce projet. Il vous sera donc possible de comprendre comment nous planifions notre budget tout au long du projet. De plus, si nous avons des imprévus, nous vous aviserons dans les délais les plus brefs afin que vous soyez au courant de la situation. Ce sera avantageux pour vous, car vous n'aurez pas à faire de supervision intensive tout au long du projet. Vous devrez simplement faire une inspection à la fin lors de la remise finale du produit. Comme nous savons exactement ce que vous voulez grâce au document d'exigences techniques, nous serons aptes à détailler notre plan ainsi que le budget du projet.

## 3 Description de la solution

### 3.1 Architecture logicielle générale (Q4.5)

Pour réussir à concevoir notre solution complète, il est important de connaître chaque système et comment ils communiquent ensemble. Le schéma ci-dessous présente l'architecture globale de notre projet :

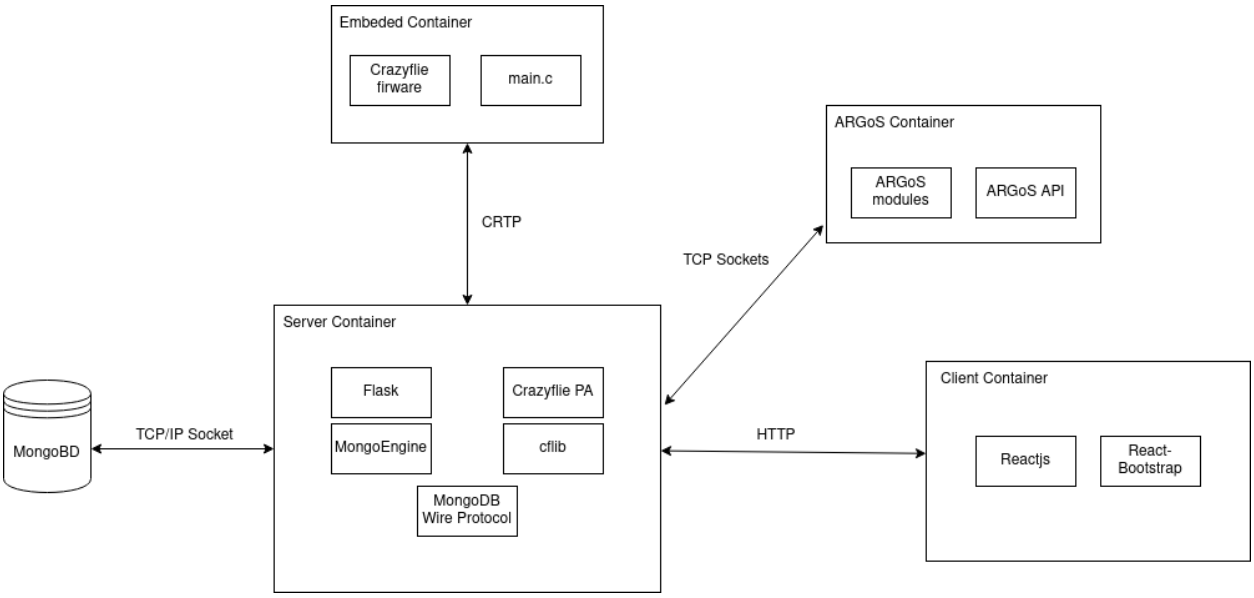



Figure 1 : Architecture générale

À première vue, on voit que la totalité du projet se trouve dans des containers Docker à l'exception de la base de données qui est hébergée sur le service de base de données MongoDB Atlas directement. Nous répondons donc au requis R.L.4, qui indique que toutes les composantes logicielles à l'exception de l'embarquée doivent être placées dans des containers Docker. Dans notre

cas, nous avons décidé de conteneuriser aussi la partie embarquée. Par partie embarquée, on parle ici de l'environnement de développement à partir duquel les drones vont être *flashés* ; le code est ensuite exécuté directement sur les drones. Même si une plus grande période de temps a dû être attribuée à cette tâche, cela permet d'avoir un environnement de travail uniforme sur tous les postes utilisés. Cela évite aussi de devoir télécharger les dépendances du projet sur l'ordinateur-même. Ces éléments bénéfiques sont aussi observés pour tous les autres systèmes. On s'assure ainsi que la version des logiciels et de leurs dépendances soit toujours la même et que la solution soit fonctionnelle peu importe l'ordinateur utilisé.

Chaque système a donc son propre Dockerfile et devcontainer.json. Le Dockerfile contient toutes les instructions permettant la construction de l'image de l'environnement de développement de chacun. Le devcontainer.json est ce qui permet de faire le lien entre tous les containers une fois rendu dans le docker-compose.yml, en indiquant comment accéder ou comment créer un container. C'est aussi dans ce fichier qu'il est possible d'indiquer les différentes extensions ou autres configurations que nous souhaitons avoir dans la fenêtre Visual Studio Code de chaque container. Visual Studio Code est l'éditeur de code que nous avons établi d'utiliser pour sa simplicité et facilité d'utilisation. De plus, tous les membres de l'équipe en ont une très bonne connaissance et ont beaucoup d'expérience avec cet outil.

Le fait que la solution complète soit conteneurisée avec Docker permet aussi de répondre au requis R.C.2. En effet, grâce au docker-compose.yml mentionné plus haut, il suffit d'utiliser la commande docker-compose up pour *build* le projet et le lancer. Ce fichier contient l'ordre dans lequel ouvrir chaque container ainsi que les informations et les configurations nécessaires pour le faire. Ainsi, le client n'a pas à connaître les mécaniques spécifiques du projet pour être capable de l'utiliser. Il n'a qu'à lancer la solution avec la commande « ./start.sh » contenant l'appel au docker-compose et ensuite se connecter au client web pour partir des missions d'exploration avec les drones ou la simulation. Grâce à cela, le client n'a pas besoin de comprendre comment chaque application fonctionne pour lancer notre application.

Chaque container contient l'environnement de développement spécifique à chaque sous-système. Le sous-système central est celui du serveur. Il s'agit du coeur de notre système ; toutes les requêtes passent par lui. C'est le seul système qui communique avec tous les autres systèmes. Il s'occupe aussi de gérer tout le traitement de données afin de rendre le client le plus léger possible. Ainsi, le serveur comporte plusieurs modules lui permettant d'effectuer la communication et la gestion de la logique. Le tout est exécuté grâce au framework Flask. Premièrement, on utilise MongoEngine qui fait appel au MongoDB Wire Protocole pour l'échange de données avec la base de données. Il s'agit du protocole TCP/IP Sockets. Deuxièmement, on rejoint les drones avec la librairie cflib fournit par Bitcraze et la Crazyflie Radio PA qui sont directement sur le serveur. Le protocole CRTP est requis ici pour l'envoie et la réception de données avec les drones physiques. Finalement, le serveur utilise la librairie socket afin d'ouvrir la communication par sockets. Il peut ainsi établir un lien de communication avec la simulation ARGoS par sockets TCP et avec le client par le protocole HTTP. Le client de son côté utilise ReactJS, un cadriciel JavaScript, pour créer l'interface avec laquelle l'utilisateur va interagir. Nous utilisons aussi la collection d'outils React-Bootstrap qui remplace Bootstrap JavaScript pour nous aider dans le design de notre application web. En effet, React-Bootstrap, contrairement à Bootstrap JavaScript, est fait pour fonctionner de pair avec ReactJS. La communication avec le serveur à l'aide du protocole HTTP sera aussi implémentée. Le client ne va nous servir que pour l'affichage ; il ne gère aucune logique de fonctionnalité. Pour ce qui est de la simulation ARGoS, nous utilisons l'API d'ARGoS, ainsi que les différents modules qui sont fournit par ARGoS. C'est d'ailleurs directement dans les différents modules que nous modifions le code afin que la simulation corres-

ponde à ce qui nous est demandé dans le document des exigences techniques. Les modules exacts que nous utilisons sont mentionnés dans la section 3.4, celle portant sur la simulation. En ce qui concerne l'embarqué, nous utilisons le Crazyflie firmware qui nous est fournit par Bitcraze. Ainsi, il nous est possible d'utiliser plusieurs fonctionnalités déjà implémentées par Bitcraze et de les ajuster selon nos besoins pour répondre aux différents requis. Ainsi, notre code faisant appel au firmware se trouve dans le *main* ou d'autres classes que nous allons créer. Finalement, comme indiqué plus haut, la base de données n'est pas hébergée sur la station au sol, mais par un service offert par un partie tierce, soit MongoDB Atlas. Cela nous fait donc un système de moins à gérer. Nous devons seulement nous occuper d'envoyer des données vers le service.

Ainsi, cette architecture va nous permettre de répondre à tous les requis demandés. Les détails de chacun et quels systèmes ils concernent sont explicités dans les sections suivantes qui explorent plus en détail chaque système.

### 3.2 Station au sol (Q4.5)

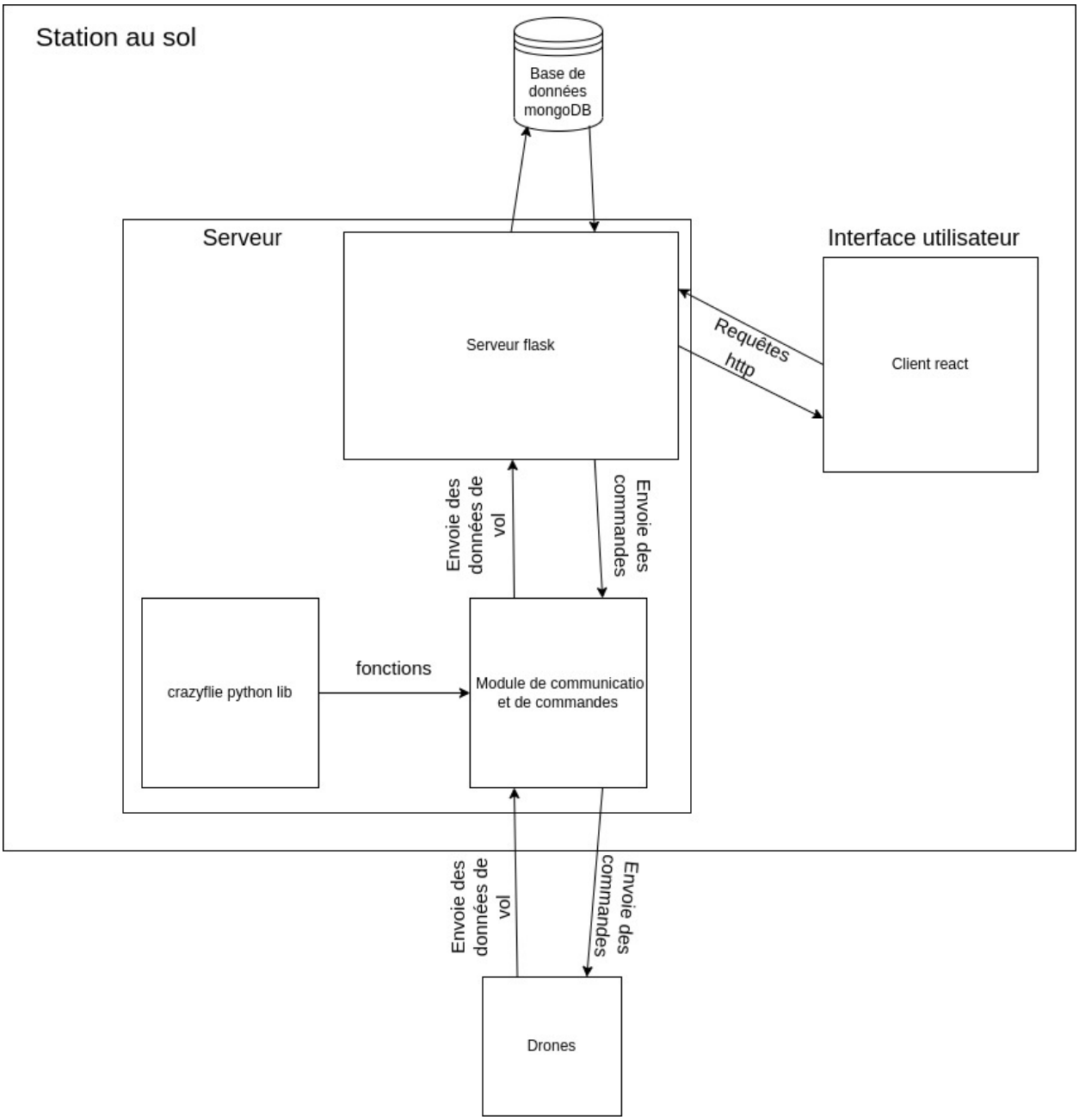



Figure 2 : Schéma de l'architecture de la station au sol

La station au sol englobe deux systèmes, soit le serveur, incluant l'antenne Crazyradio PA, et l'interface utilisateur. Cette dernière est présentée en détail à la section 3.5 et ne sera donc pas abordée dans la section ci-présente. Considérant le fait que Bitcraze fournit déjà une librairie en python pour la communication avec les drones Crazyflie 2.0, la Crazyflie python library (cflib), nous avons convenu de développer notre serveur en python. La composante principale de ce serveur est un framework Flask. Il est conteneurisé dans un Docker container (R.L.4). Celui-ci reçoit et traite des demandes de l'utilisateur qui les envoie à partir de l'interface utilisateur. Il fait ensuite appel à un

module de communication et de commandes afin d'envoyer les commandes directement aux drones que ce soit à travers l'antenne Crazyradio PA pour les drones physiques (voir section 3.3) ou pour les drones simulés (voir section 3.4). Le module utilise le protocole de Bitcraze, Crazy RealTime Protocol (CRTP), pour l'envoie de packet des composantes du serveur en python au système embarqué des drones en C/C++. La station comporte aussi une base de donnée MongoDB qui gardera dans un document les informations sur chaque vol et chaque mission. Celles-ci pourront être consultées par l'utilisateur à partir de l'interface (requis R.F.17). Les données seront récoltées par les drones-mêmes et envoyées à l'aide du CRTP. Les fonctions de communication CRTP sont encapsulées dans le module de communication du serveur flask.

Comme mentionné un peu plus haut, le *framework* pour le serveur se devait d'être en python. Plusieurs options s'offraient alors à nous. Celles que nous avons ressorties sont les suivantes : Django, Flask et FastAPI. Nous avons éliminé Django, puisque son apprentissage nécessite beaucoup de temps et qu'il nécessite une plus grande quantité de code pour accomplir une tâche simple. De plus, Django est un *framework* utilisé et conçu pour développer des projets de grande envergure ce qui n'est pas notre besoin [15]. Nous avons ensuite éliminé FastAPI. Il s'agit d'un *framework* qui aurait amplement pu répondre à nos besoins. Toutefois, il s'agit d'une technologie plus récente et par conséquent, il y a un peu moins de documentation disponible en ligne. Nous avons donc opté pour une option plus sécuritaire avec Flask, qui a déjà largement fait ses preuves [14]. Flask est un *framework* permettant de créer un serveur en utilisant des threads. Il dépend du modèle du moteur Jinja ainsi que des outils Werkzeug WSGI toolkit. On retrouve une documentation complète de Flask sur leur site web officiel [12]. Il est utilisé pour faire le développement d'application web et permet de créer un prototype rapide [14]. De plus, certains membres de l'équipe avaient déjà de l'expérience avec Flask.

Plus précisément, notre serveur est composé de deux fichiers, soit app.py et drone.py. Dans app.py, on retrouve toutes les routes permettant la communication avec le client et la base de données. C'est dans ce fichier que le serveur reçoit les commandes envoyées du client, les traitent, puis les envoie vers les drones ou la simulation. Il s'occupe aussi de la communication dans l'autre sens, c'est-à-dire du serveur vers le client où le serveur envoie les données devant être affichées du côté client, c'est-à-dire, les positions pour les cartes et les données de débogage. De son côté, drone.py est spécifiquement pour le traitement des données des drones. Une classe Drone contient les fonctions de haut niveau qui sont communes aux deux systèmes (drones physiques et simulation), puis chaque système possède sa propre classe. Il est donc possible d'établir la communication entre le serveur et les drones ou la simulation. C'est aussi dans ce fichier qu'on effectue le traitement et le formatage des données qui devront ensuite être envoyées vers le client. En effet, il n'y a pas de communication client-BD. Seulement le serveur peut envoyer et recevoir des données de la base de données qu'il peut transmettre au client au besoin.

Pour ce qui est de la base de données en-soi, nous en avons choisi une de type NoSQL en raison de la flexibilité qu'elle nous offre pour l'enregistrement des données. En effet, cela nous permettra de placer toutes les informations demandées sur les missions précédentes ainsi que les cartes et les logs de vol à même la base de données (R.F.17, R.F.18 et R.C.1). Comme la grande majorité des membres de l'équipe avait déjà travaillé avec la base de données MongoDB, notamment lors du projet intégrateur 2, nous avons décidé de réutiliser ce service pour ce projet. Nous en avions retiré une bonne expérience et avions apprécié sa facilité d'utilisation. De plus, il est possible d'utiliser une base de données hébergée sans frais, nous n'avons donc pas à nous soucier de cet aspect pour le projet. Pour l'hébergement de celle-ci, nous avons choisi la plateforme MongoDB Atlas et AWS.

Pour que la base de données MongoDB puisse communiquer avec le serveur, cela nous prend des

*drivers*. Ceux-ci nous sont fournis par MongoDB [9]. La communication s'effectue grâce au MongoDB Wire Protocol implémenté dans les drivers de MongoDB. Ce protocole utilise une communication TCP/IP socket pour parler au client, qui est, dans notre cas, notre serveur [10]. En faisant un peu de recherche, nous avons constater que le *driver* de choix pour travailler avec MongoDB et Flask était soit PyMongo ou MongoEngine. Nous avons décidé d'y aller avec PyMongo qui a un API facile à utiliser et qui répond a nos besoins. Avec PyMongo, les modèles créés sont des documents envoyés sous forme de dictionnaire python. Ainsi, tout ce qui nous faut pour insérer un document dans la base de données en ligne est un dictionnaire avec des clés et des valeurs. Pour la réception des données, il faut simplement prendre l'objet représentant la base de données et utiliser la fonction find() [13].

## 3.3 Logiciel embarqué (Q4.5)

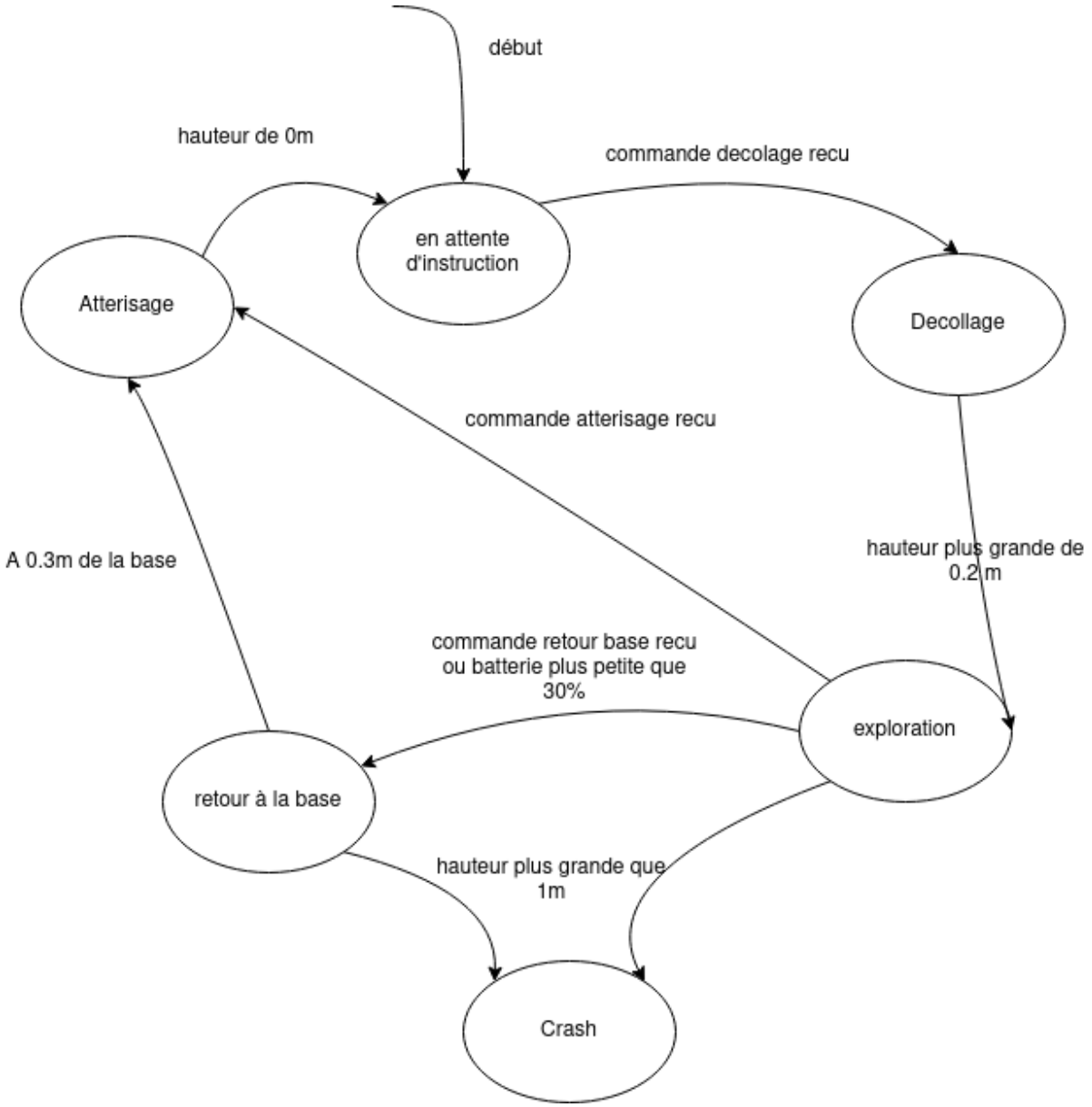


Figure 3 : Machine à état d'un drône

Les drones sont programmés à l'aide d'un Crazyradio PA 2.4 GHz USB dongle et utilisent la librairie crazyflie-firmware. Une fois programmés, les drones vont répéter la même boucle d'exécution. Avec la fonction appchannelReceiveDataPacket, ils attendent de voir si le serveur envoie une instruction. Les instructions envoyées vont permettre d'implémenter les requis R.F.1, R.F.2, R.F.6 et R.F.7. À chaque 1 Hz, le drone envoie à la station au sol son état, sa position, son orientation et l'information récoltés par les capteurs. Toutes ces données sont utilisées dans les autres systèmes pour répondre aux requis. Le drone reste aussi en connexion avec les autres drones à l'aide de l'API Peer to Peer pour pouvoir s'éviter. Si le drone est en état d'exploration ou de retour à la base (R.F.6), il va passer par l'état de déplacement. Nous allons utiliser le random walk pour l'exploration (R.F.4) tout en évitant les autres drones et obstacles (R.F.5). Cet algorithme d'exploration permet aux drones de vérifier de manière aléatoire s'il peut se diriger dans une direction en vérifiant la valeur du capteur associé à la direction souhaitée. S'il n'y a pas d'obstacle, le drone avance, puis fait un nouvel appel aléatoire sur l'un de ses capteurs pour vérifier que la nouvelle direction désirée est aussi possible. Dans le cas où ce ne serait pas le cas, il fait un nouvel appel aléatoire sur un autre capteur jusqu'à en trouver un lui permettant d'avancer. Tant que le drone n'a pas détecté une direction valide, il reste à sa position actuelle. De plus, le niveau de la batterie sera vérifié à chaque boucle pour que s'il est en bas de 30%, l'état retour à la base soit activé (R.F.7). Pour le requis de crash de drone (R.F.13), l'information sera transmis dans l'état d'envoi d'information. La mise à jour du code sur le drone n'est pas dans la machine à état puisque celui-ci change complètement le code implémenté sur le drone (R.F.14).

## 3.4 Simulation (Q4.5)

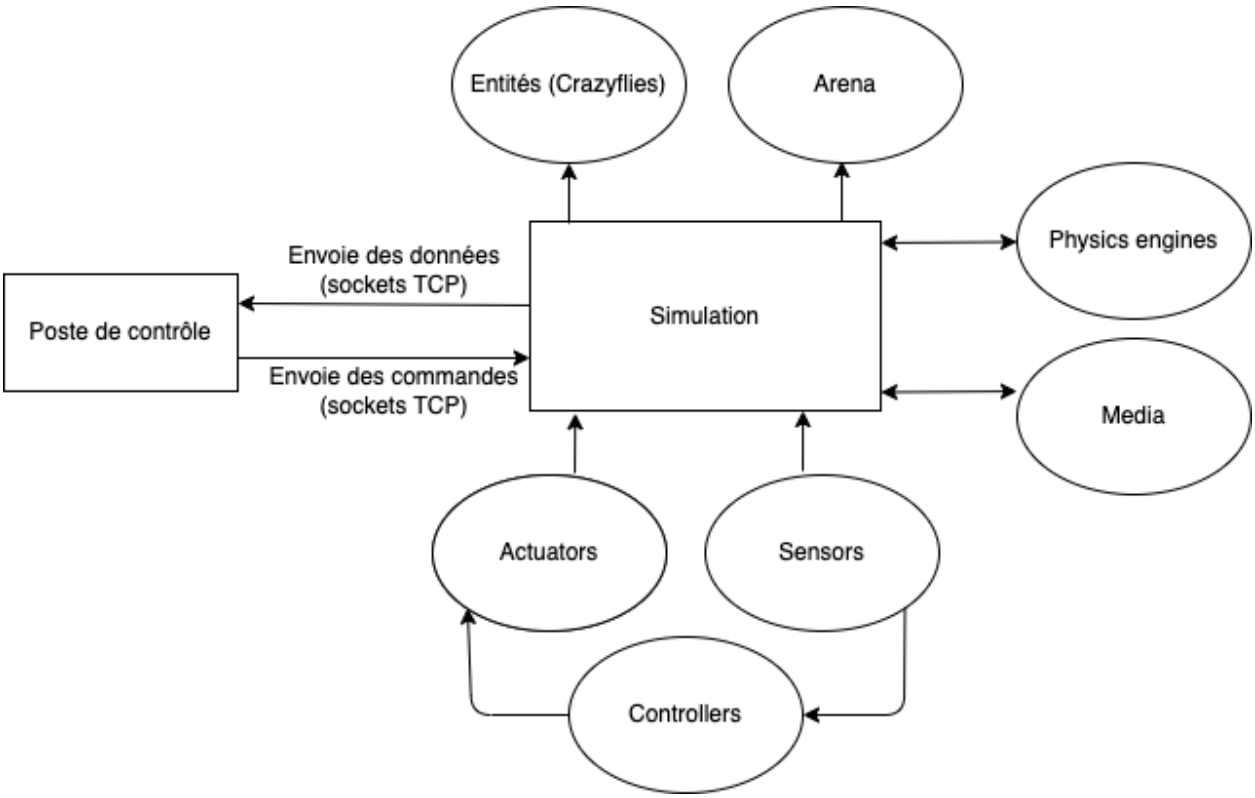


Figure 4 : Architecture de la simulation

Pour obtenir l'information sur les commandes à exécuter et pour envoyer l'information qui permet de dessiner la carte, la simulation communique avec le poste de contrôle avec des sockets TCP/IP. Nous avions tout d'abord commencé la conception de la communication en utilisant des sockets Unix. Un socket Unix permet un échange de données bidirectionnel entre deux processus sur le même ordinateur fonctionnant à l'aide du système d'exploitation Linux. Nous avions donc considérer que ce type de communication serait idéal pour nous puisque la simulation et le poste de contrôle sont sur le même ordinateur. Cependant, plusieurs problèmes d'intégration nous ont forcé à modifier notre choix et y aller avec des sockets TCP/IP. Même si ceux-ci sont notre deuxième choix, ils n'en sont pas moins efficaces. En effet, la connexion s'effectue avec le numéro de port du client. Il nous sera donc facile de connecter plusieurs drones et de communiquer individuellement avec chacun. L'implémentation de ce type de communication nous permet donc de répondre à différents requis en lien avec la simulation. En effet, l'utilisation de sockets TCP/IP sera utile pour les requis R.F.2 et R.F.6 afin de recevoir les demandes de lancement et de fin de mission ainsi que de retour à la base émis par l'interface utilisateur par l'entremise du serveur. Cette communication permettra également de remplir les requis R.F.3, R.F.7, R.F.8 et R.F.9, puisque la simulation pourra envoyer à la station au sol l'état de drones, le niveau de leur batterie, les données de leurs capteurs de distances ainsi que leur position dans l'environnement.

Afin de s'assurer que les drones ont le bon comportement, nous utilisons ARGoS, un environnement de simulation qui nous permet de simuler les essaims de robots en déplacement. La simulation nous assura de vérifier si notre code permet aux crazyflies de se déplacer de manière indépendante sans entrer en collision avec les murs ou entre eux avant de tester le tout avec les drones physiques. Notre partie simulation peut être séparée en plusieurs modules. Tout d'abord, il y a l'espace de simulation en elle-même contenant les *entities*, soit, dans ce cas-ci, des crazyflies. Ces derniers font appel aux modules *sensors* et *actuators* qui interagissent avec le module *controllers* relatif aux crazyflies afin d'obtenir l'état de la simulation. Par exemple, le module des *sensors* s'occupe, entre autres, de la simulation du capteur de distances des crazyflies pour que ceux-ci puissent détecter des obstacles dans l'espace de simulation, tel que le demande le requis R.F.5. Il permet également de simuler le niveau de batterie des robots : fonctionnalité nécessaire à la réalisation du requis R.F.7. Pour sa part, le module des *actuators* est utile pour le contrôle de la position des quadricoptères, tel qu'imposé pour la réalisation des requis R.F.9, R.F.11 et R.F.12. Ensuite le module *physics engines* sert à établir le type de moteurs physiques attribués aux robots. Dans le cas des crazyflies, une simple représentation 2d dynamique est suffisante pour illustrer leurs déplacements dans l'environnement. Par ailleurs, le module *media* permet aux quadricoptères de communiquer entre eux durant la simulation. La communication utilisée sera celle du *range-and-bearing* qui fonctionne de façon locale lorsque les drones sont à porter de vue l'un des autres. Cette fonctionnalité peut être pertinente pour permettre aux robots d'échanger des données lorsqu'ils sont en exploration. Finalement, le module *arena* contient les configuration pour les différents murs à créer. Cette configuration pourra être modifiée afin de générer aléatoirement les obstacles de l'environnement d'exploration, tel que demandé par le requis R.C.3 [7].

### 3.5 Interface Utilisateur (Q4.6)

Avant même de commencer à programmer, nous avons décidé de concevoir une maquette afin de guider notre développement. Nous avons utilisé le logiciel Figma, qui est gratuit et qui nous permet même de travailler à plusieurs. Nous avons décidé de faire un client à plusieurs pages. La figure suivante représente la page d'accueil :


Figure 5 : Maquette page d'accueil de l'interface utilisateur

Ici, on constate que nous faisons la séparation de la simulation et de la mission physique dès le début et qu'après l'utilisateur ne verra plus la différence entre la simulation et une vraie mission. Pour cela, nous créerons le reste des composantes de la page Web en leur précisant ce choix. Le but est de cacher le plus possible cette différence à l'utilisateur afin que le client soit le plus facile possible à utiliser. Grâce à cela, nous répondrons au requis R.C.4 en respectant notamment le 4e principe du groupe Nielsen Norman indiquant que l'application doit être la plus consistante possible. Ainsi l'utilisateur n'aura pas à se demander par exemple si le bouton X lors d'une simulation à le même impact que le bouton Y puisqu'ils seront les mêmes. Nous éviterons ainsi le plus de confusion possible. On remarque également que nous permettons de consulter les anciennes missions. Avant de rentrer dans les détails techniques, nous allons d'abord vous montrer ce que l'interface client affichera lorsqu'on clique sur ce bouton.

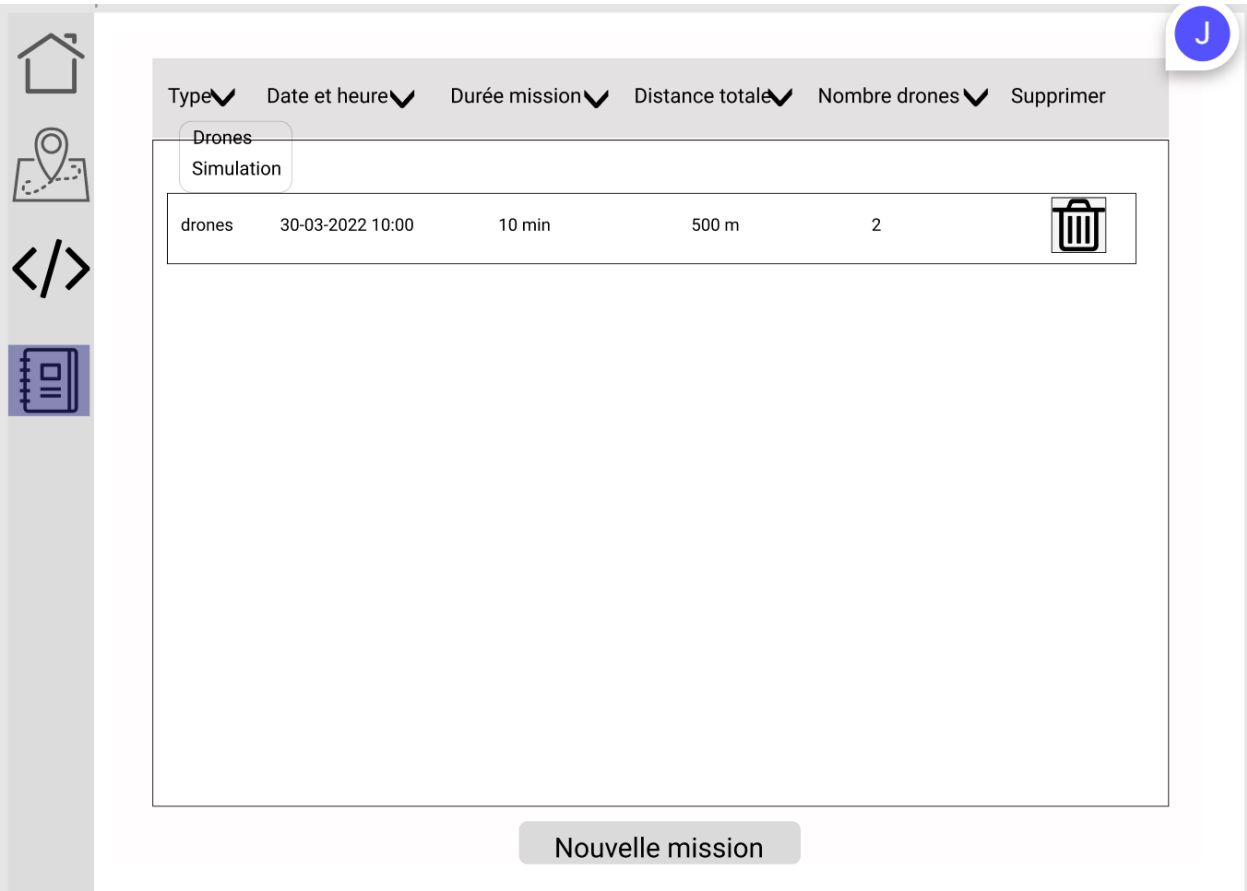



Figure 6 : Maquette page des missions de l'interface utilisateur

Ici, nous remplirons donc le requis R.F.17, en permettant à l'utilisateur de consulter les missions qu'il a effectuées précédemment. Il est possible que nous rajoutions des données plus tard dans le projet, mais pour l'instant nous ne planifions de faire que ces requis, car c'est ce qui est demandé. Comme nous voulons nous laisser la possibilité de modifier facilement notre base de données, nous allons utiliser du NoSQL afin d'être plus libre dans notre projet. Plus particulièrement nous utiliserons MongoDB. Avec MongoDB, nous avons la chance d'héberger nos données jusqu'à 5GB gratuitement donc c'est parfait pour le développement du projet. De plus, MongoDB est réputé pour avoir une architecture qui évolue facilement. La partie du contrôle en temps réel est expliquée dans la figure suivante.

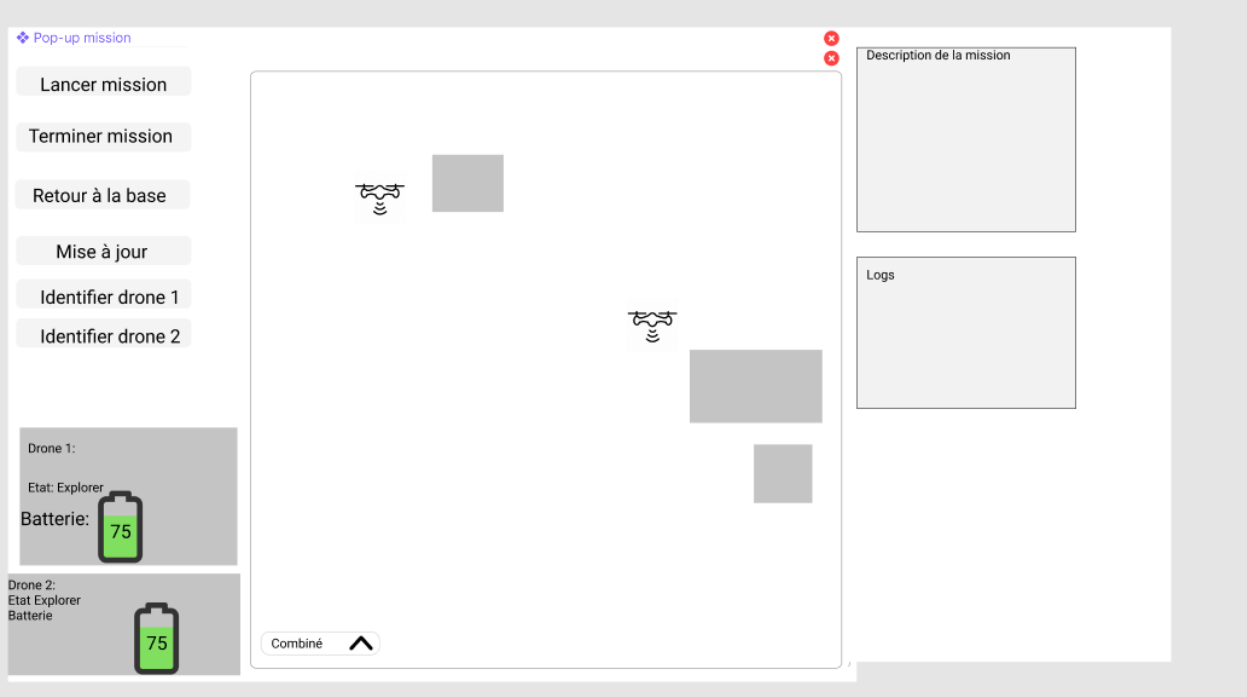



Figure 7 : Maquette carte de la mission et des boutons de contrôle de l'interface utilisateur

Avec cette fenêtre nous remplirons plusieurs requis. Tout d'abord, vous pouvez observer la présence des boutons *Identifier drone 1* et *Identifier drone 2*. Ces boutons nous permettront de remplir du côté client le requis R.F.1. On remarque également les boutons *Lancer Mission* et *Terminer Mission* qui permettront à l'utilisateur de réaliser ces fonctionnalités. Dans le coin gauche inférieur, nous mettrons à jour l'état des drones. L'état des drones sera l'état courant de la machine à état des drones. Nous pourrons également voir la batterie du drone. Ensuite, nous pouvons voir qu'au centre de la figure nous retrouvons la carte. Nous aurons une carte pour chaque drone ainsi qu'une carte qui combinera les cartes de tous les drones en une seule. Finalement, nous pouvons voir les logs afin de voir les messages des drones ou de la simulation afin de vérifier que tout se déroule comme prévu. Si jamais nous avons encore du temps de disponible à la fin du projet, nous allons essayer de concevoir un éditeur de texte ce qui nous permettrait de rencontrer les critères du R.F.14 et R.F.16. La figure suivante illustre ce que nous avons en tête pour l'éditeur de texte.

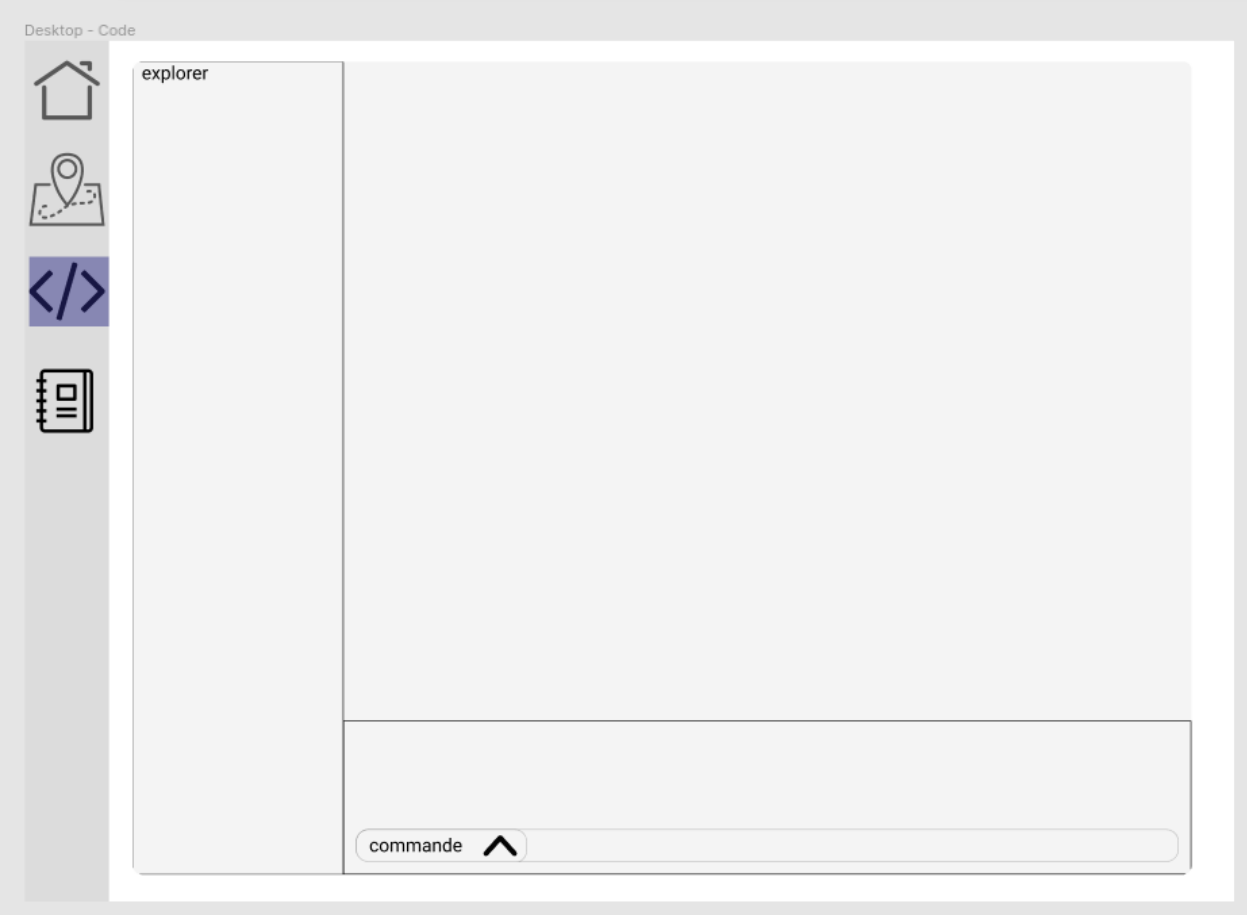



Figure 8 : Maquette éditeur de texte de l'interface utilisateur

Maintenant que nous avons décrit l'interface du client, nous allons décrire ce que nous allons faire pour réaliser le client.

Tout d'abord, le client ne communique qu'avec le serveur et aucun des autres systèmes du projet. Ainsi, nous limitons la complexité de notre client. En effet, celui-ci ne sert qu'à l'affichage des données. Tout le traitement des données sera géré par le serveur qui, lui, communique avec la simulation et les drones. Pour la communication avec le serveur, nous planifions utiliser des requêtes HTTP via une API REST. En effet, lors de d'une exploration les drones devront envoyer les données collectées concernant leur environnement ainsi que leur position actuelle à raison de 1Hz. Le client devra donc être en mesure de changer l'affichage des cartes à chaque seconde. À la suite de quelques tests d'envoi de données, nous avons convenu que le protocole HTTP nous permettait de répondre à ce requis sans problème.

Pour programmer le client, nous utiliserons la librairie Reactjs. Nous avons décidé d'utiliser cette librairie, car elle est réputée pour sa simplicité. Aussi, elle permet de faire facilement des applications avec plusieurs pages. De plus, cette librairie est considérée comme étant très facile à apprendre ce qui est bon pour nous, car cela nous prendra moins de temps afin de réaliser ce que nous désirons. Au commencement du projet, nous avions quelques autres technologies en tête. Premièrement, ayant vu le requis R.F.10 qui indique que l'application web doit être disponible sur différents types de d'appareils, nous avons pensé à Flutter et React Native. Toutefois, nous avons constaté que le requis demandait simplement que l'affichage soit adapté pour des appareils mobiles. Il n'est donc

pas nécessaire de faire une application dédiée au mobile. Nous avons donc délaissé ces deux technologies qui ne permettent que de faire des applications mobiles. Il y avait aussi Angular que nous avons considéré utiliser. Toutefois, comme il s'agit du *framework* que nous avons utilisé tout au long de notre projet de deuxième année, nous avions envie d'apprendre un nouvel outil. Surtout que sur le marché du travail, React est aussi souvent demandé en compétence. Ainsi, cela nous donnait l'occasion de travailler avec ReactJS. De plus, React est idéal pour la création d'application avec des données qui varies fréquemment, ce qui va être notre cas pour l'affichage des cartes. En effet, React utilise le DOM virtuel contrairement à Angular qui utilise le DOM réel ce qui facilite la manipulation du DOM [2, 4].

Pour nous aider à rendre le client plus attrayant à l'utilisateur nous utiliserons également la librairie bootstrap qui a déjà plusieurs composants déjà pré-fabriqués ce qui nous permettra encore une fois de sauver du temps. Ainsi, nous n'aurons pas à faire une tonne de code CSS sur nos composants, car cette librairie vient déjà avec plusieurs options de personnalisation. Cela permettra également de rendre l'application web *responsive*, puisque nous devons nous assurer que celle-ci puisse s'afficher correctement sur plusieurs types d'appareils, tel que demandé par le requis R.F.10.

Notre client fonctionnera sur plusieurs pages. Chaque page affichera une composante. Ainsi, nous aurons la page qui sera située dans la direction de base qui sera la page d'accueil où vous aurez le choix de démarrer soit une nouvelle mission, soit une nouvelle simulation. Également, nous avons l'option de consulter les anciennes missions. Chacun de ces boutons sera en fait un lien qui nous amènera vers une nouvelle page qui affichera un nouveau composant. Cependant, si nous choisissons de démarrer une nouvelle mission nous créerons une nouvelle fenêtre "pop-up" pour que le client puisse contrôler la mission comme il le souhaite. Ensuite, on remarque qu'il y a une barre latérale. Cette barre sera également un composant et nous permettra de naviguer entre les différentes fenêtres de l'application.

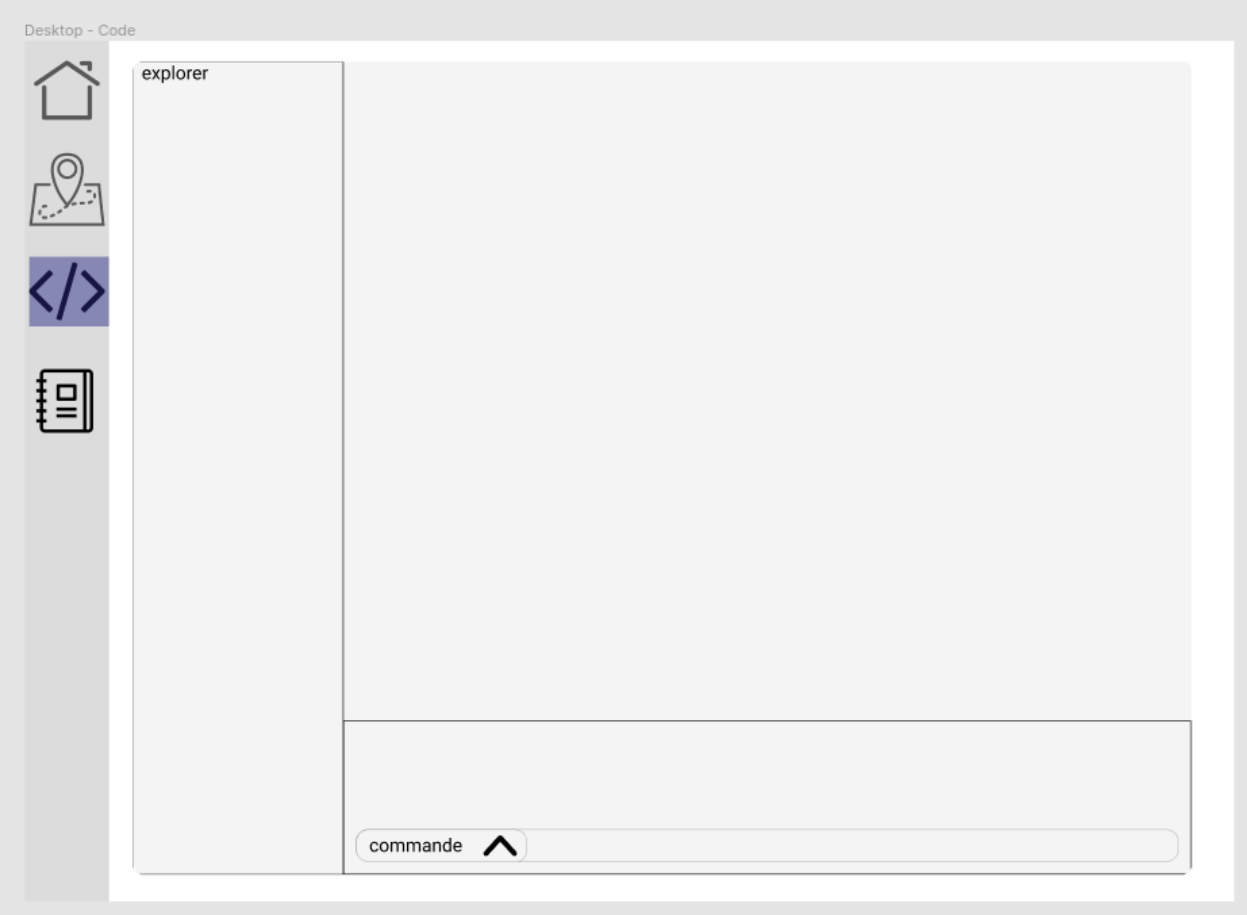



Figure 8 : Maquette éditeur de texte de l'interface utilisateur

Voici l'apparence finale du client :

Figure 9 : Page d'accueil

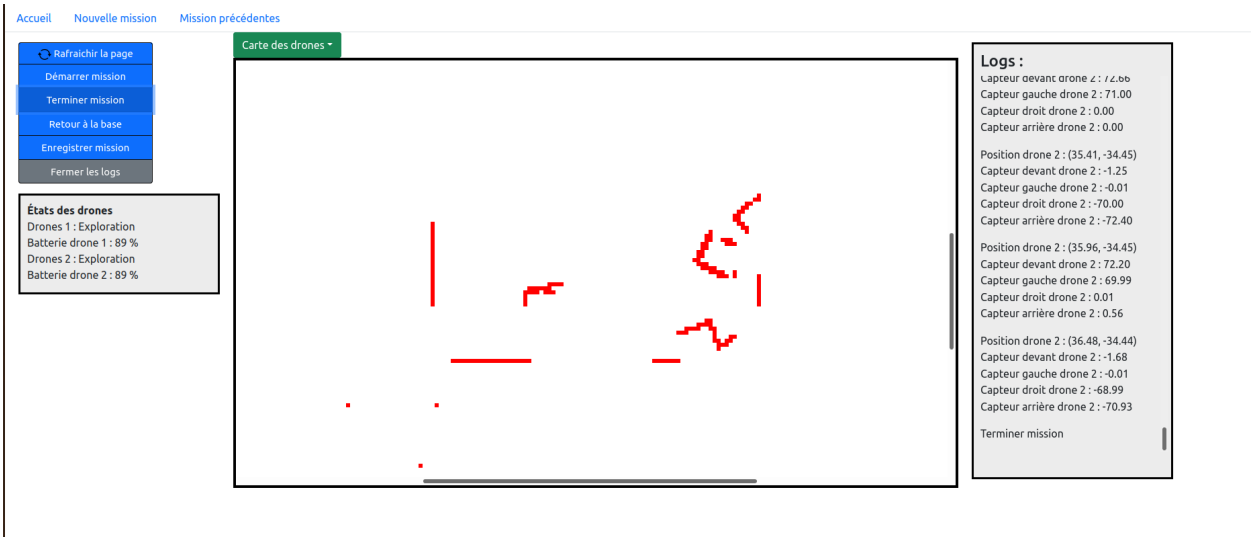



Figure 10 : Nouvelle mission

Notre interface à maintenant une page d'accueil d'où il est possible d'accéder aux pages de démarrage de nouvelles missions pour les drones et pour la simulation. Il est aussi possible de se rendre à la page de consultation des missions précédentes. La page de mission comporte beaucoup plus de logique et est plus complexe que la page d'accueil. Pour ce faire, chaque élément distinct de la page a un composant propre. Cela permet d'avoir une meilleure organisation pour l'architecture du client. Ainsi, chaque composant a sa propre tâche et on évite d'avoir des fichiers énormes qui prennent en charge la logique complète de l'application. On a premièrement les boutons qui permettent d'envoyer les commandes aux drones et l'affichage de l'état des drones et le niveau de leur batterie. À la fin d'une mission, il est possible d'enregistrer la mission sur la base de données en appuyant sur le bouton correspondant. Les états des drones possibles sont en attente, en décollage, en exploration, retour à la base, atterrissage et crash. Cette dernière option est seulement disponible sur les drones physiques. Deuxièmement, nous avons l'affichage de la carte. Il est possible d'afficher la carte de chacun des deux drones individuellement et la carte globale de la mission grâce au menu déroulant en haut à gauche. Finalement, à droite, on retrouve l'affichage des données de débogage. Il présente les commandes envoyées, les positions des drones et les lectures des capteurs. Il est possible d'afficher ou de retirer les données de débogage avec le bouton gris à gauche.

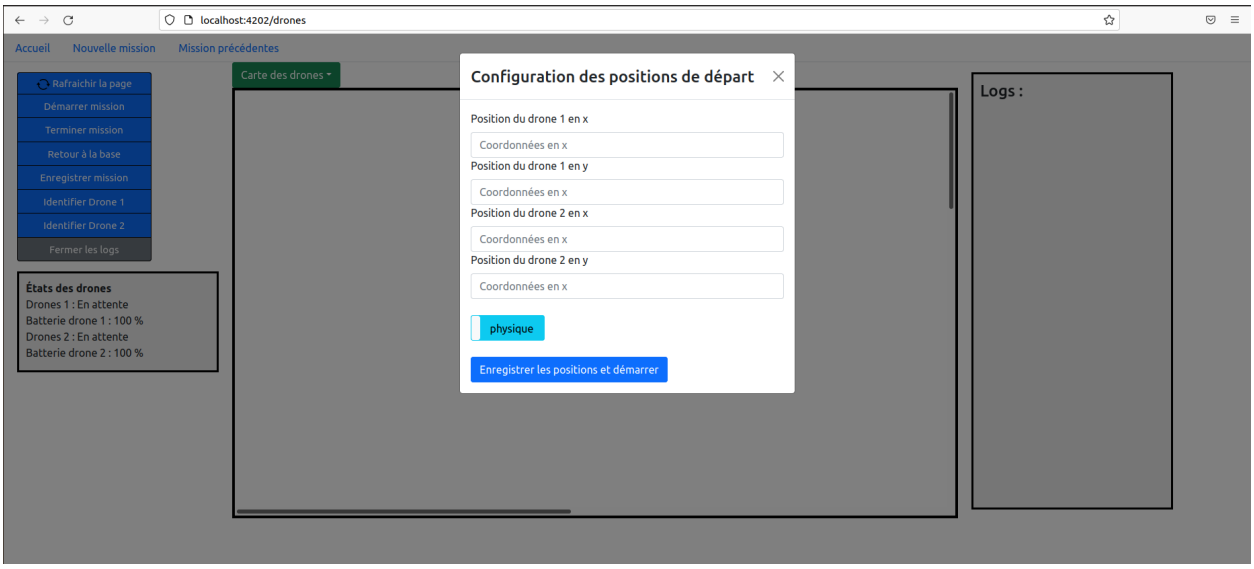



Figure 11 : Menu démarrage

Sur la figure 11, on voit la fenêtre qui apparaît lorsqu'on appuie sur le bouton "Démarrer mission". On voit qu'il faut entrer les positions initiales des drones afin de pouvoir aligner la carte globale des drones. De plus, il y un bouton au bas qui permet de sélectionner une mission avec les drones physiques ou avec la simulation. Pour la simulation, les positions initiales sont prises directement sur le serveur. Ce bouton change aussi les options disponibles à gauche. En effet, pour les drones physiques, l'option "Identifier les drones" apparaît.

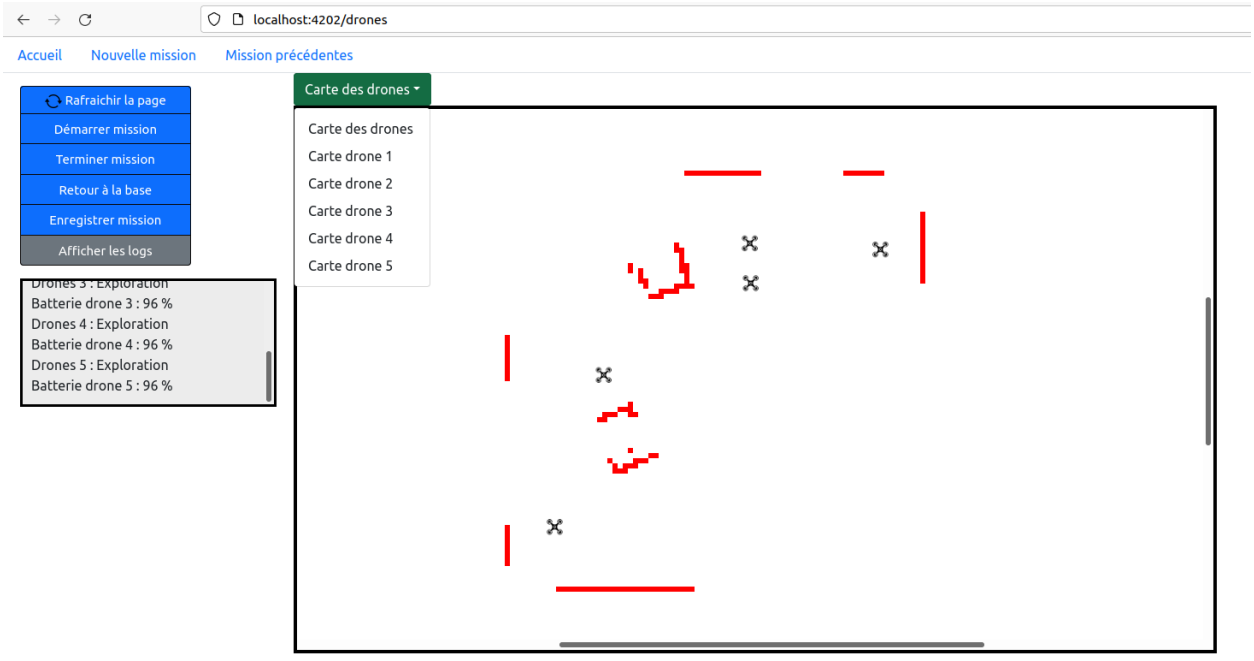



Figure 12 : Nombre arbitraires de drones

La figure 12 illustre qu'il est possible d'avoir un nombre arbitraire de drones dans la simulation. En effet, on voit qu'il y a maintenant cinq drones sur la carte, et non seulement deux. L'affichage est

donc adapté dans la boîte de l'état des drones. On voit qu'il y a maintenant cinq états et cinq niveaux de batterie. On peut aussi naviguer entre les cinq cartes individuelles des drones.

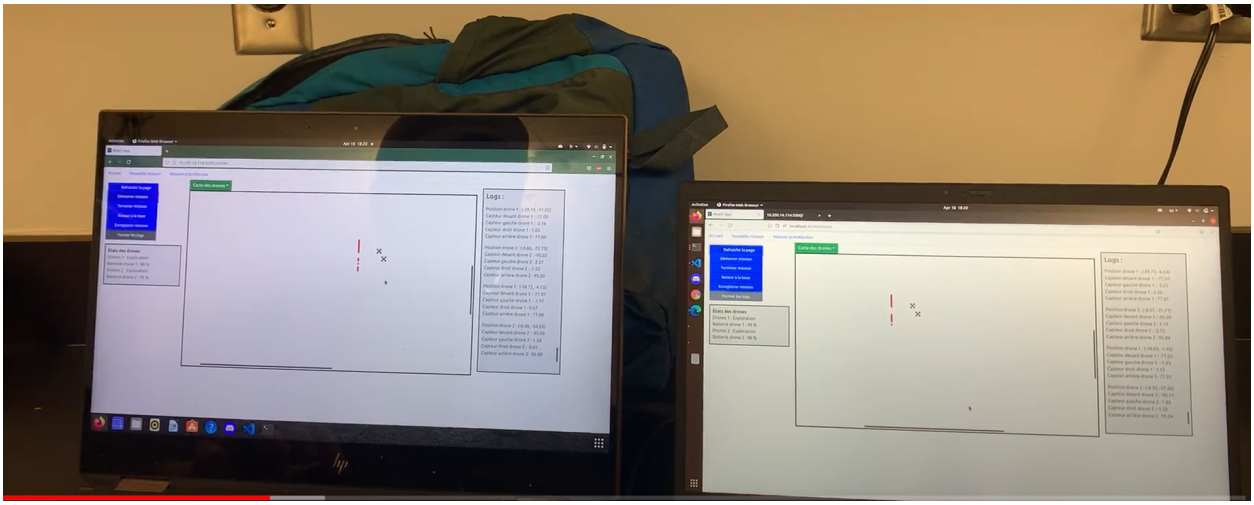


Figure 13 : Client offert sur plus qu'un appareil

La figure 13 montre qu'il est possible d'accéder au client à partir de plus qu'un seul appareil. Les fonctionnalités sont toutes disponibles sur les deux appareils. Il suffit d'appuyer sur le bouton "Rafraîchir la page" afin de voir apparaître la mission en cours. Si une mission est déjà en cours et qu'un autre utilisateur tente d'appuyer sur "Démarrer mission", une alerte est lancée pour indiquer qu'il y a déjà une mission en cours.

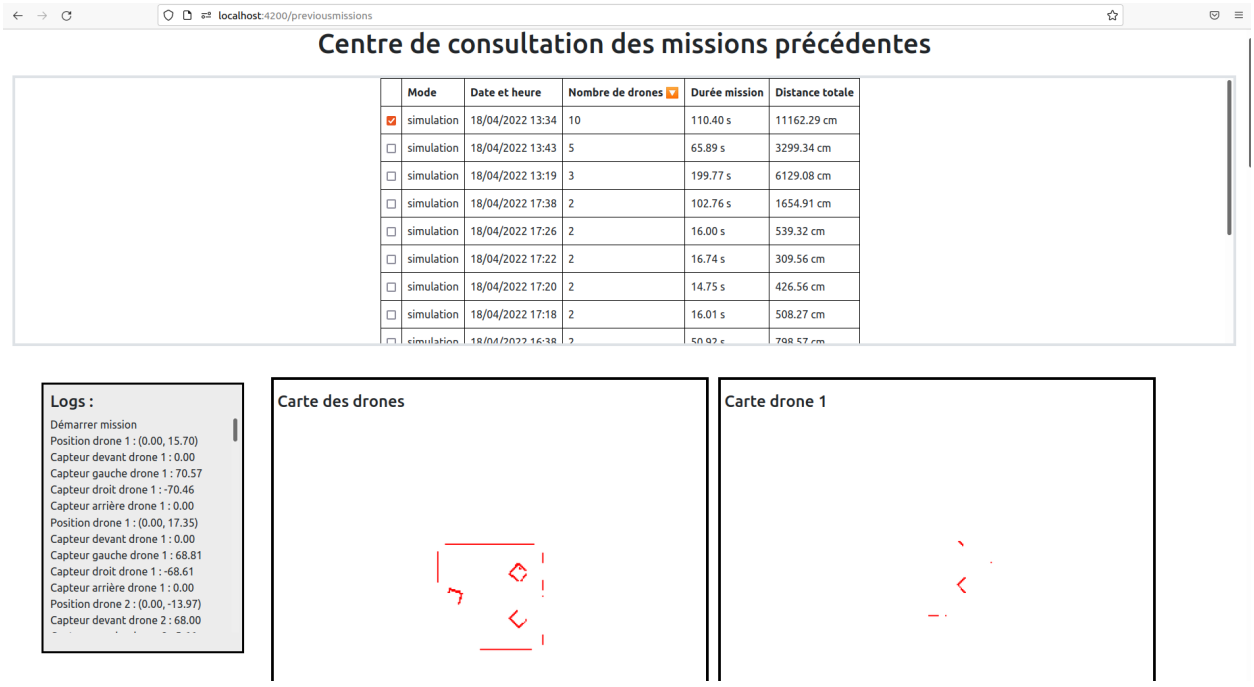



Figure 14 : Base de donnée et consultation des missions précédentes

Finalement, la figure 14 présente la page de consultation des missions précédentes. Lorsqu'on entre sur la page, une requête est envoyée au serveur afin de récupérer les données qui sont hébergées

sur le serveur. On obtient alors le tableau qui présente la liste de toutes les missions qui ont été enregistrées. Chaque mission est décrite par son mode, sa date et son heure, son nombre de drones, sa durée et la distance totale parcourue par les drones. Il est possible de les trier selon leur attribut en appuyant sur le champ voulu. On peut consulter les cartes et les données de débogage de chaque mission en cochant la case correspondante.

### 3.6 Fonctionnement général (Q5.4)

Afin de faire fonctionner notre système, on doit d'abord lancer la commande *docker-compose up -d –build* afin de *build* de tous les containers de notre projet. Il faut ensuite ouvrir les containers en les attachant à une fenêtre Visual Studio Code. On commence ensuite en lançant le serveur. Pour se faire, dans le terminal du container correspondant, on va dans le dossier app-server (*cd server/app-server*), pour ensuite entrer la commande *flask run*. Le serveur est maintenant en marche sur le port 5000. Ensuite, pour le client, dans le terminal de son container, on se rend dans le dossier client (*cd /workspace/client*) et on lance le tout avec *sudo yarn start*. Le client devrait s'ouvrir automatiquement dans le navigateur sur le port 4200, sinon il est possible d'y accéder avec l'adresse suivante : localhost :4200. Puis, pour ouvrir l'environnement de la simulation, on commence par lancer x11docker avec : *sudo x11docker –hostdisplay –hostnet –user=RETAIN – –privileged -v – inf3995-105_projet3-simulation*. Il faut ensuite ouvrir un second terminal à l'intérieur de dossier .devcontainer (cd /INF3995-105/argos/.devcontainer) et y inscrire : *docker exec -it <id du container> /bin/bash*, puis *argos3 -c experiments/crazyflie_sensing.argos* à l'intérieur du dossier *examples* (*cd /root/examples/*). Une fois tout cela complété, il faut appuyer sur le bouton *play* dans ARGoS. Finalement, pour faire rouler du code sur les drones, il faut aller dans le répertoire embedded/src (*cd embedded/src*). Pour *flasher* les drones, on fait *make*, suivi de *make cload*. Pour la commande make cload, il faut s'assurer d'avoir l'antenne crazyflie de brancher a l'ordinateur et que les drones soient en mode flash (tenir le bouton alimentation jusqu'à ce que les lumières bleues clignotes).

Afin de visualiser l'exploration à l'aide du client, il faut démarrer le serveur, la simulation et les drones physiques de la même façon que présentée ci-haut. Toutefois, dans le cas de la simulation, il faut attendre de cliquer sur *Démarrer mission* et d'enregistrer les positions initiales des drones avant de cliquer sur le bouton *play* de la fenêtre Argos pour que les données puissent être affichées à l'écran.

Afin de répondre au R.C.2 qui demande de lancer le projet en une seule étape, il est possible de lancer le système avec la commande suivante : ./start.sh . Dans le cas de la simulation, il faut tout de même appuyer sur le bouton *play* de la fenêtre Argos après avoir fait *Démarrer mission* dans le client pour que les drones commencent leur exploration.

Pour lancer les tests du client, il est possible de suivre la procédure suivante dans un terminal lorsque le système est lancé : *docker ps*, *docker exec -it <id du container> /bin/bash*, *cd client* et *yarn test*.

## 4 Processus de gestion

### 4.1 Liste des requis du projet

**Requis fonctionnels**

— Chaque drone physique doit individuellement répondre à la commande "Identifier" disponible dans l'interface utilisateur. Lors de cette commande, au moins une des DELs du drone doit

clignoter ou changer de couleur.

- L'essaim de drones doit répondre aux commandes : "Lancer la mission" et "Terminer la mission".
- Pour chaque drone, l'interface utilisateur doit montrer l'état des drones mis à jour avec une fréquence minimale de 1 Hz.
- Après le décollage, les drones doivent explorer l'environnement de façon autonome. L'algorithme est à la discrétion du contractant. Cela peut être une séquence de mouvements aléatoires(random walk).
- Les drones doivent éviter les obstacles détectés par leurs capteurs (objets statiques ou autres drones sans distinction).
- Le retour à la base doit rapprocher les drones de leur position de départ pour qu'ils soient à moins de 1 m de celle-ci. Une commande de "Retour à la base" doit être disponible sur l'interface utilisateur.
- Le retour à la base et l'atterrissage doit être activé automatiquement dès que le niveau de batterie devient moins de 30%. Les drones ne doivent pas décoller avec un niveau de batterie inférieur à 30%. Le niveau de batterie des drones doit être affiché sur l'interface utilisateur.
- La station au sol doit collecter les données des capteurs de distance des drones et générer des cartes de l'environnement. Il doit y avoir une carte individuelle pour chaque drone. Ces cartes doivent être affichées en continu lors de la mission dans l'interface utilisateur. Les cartes générées doivent ressembler à l'environnement exploré.
- Lors d'une mission, la position d'un drone dans la carte doit être affichée en continu.
- L'interface utilisateur pour l'opérateur doit être disponible comme service Web et visualisable sur plusieurs types d'appareils via réseau. Au moins deux appareils doivent pouvoir être connectés en même temps pour la visualisation des données lors d'une mission.
- La station au sol doit intégrer les données de tous les drones et créer une seule carte globale de l'environnement exploré.
- La position et l'orientation initiale respective des drones dans l'environnement doit pouvoir être spécifiés par l'opérateur dans l'interface utilisateur avant le début de la mission.
- Le système doit pouvoir détecter un crash de drone. Dans ce cas, l'interface utilisateur doit montrer l'état "Crashed" pour ce drone.
- Une base de données doit être présente sur la station au sol et enregistrer les missions précédentes.
- Les cartes générées lors d'une mission doivent être enregistrée sur la station au sol.
- Les drones doivent pouvoir s'entendre sur la couleur d'une ou plusieurs de leur DELs. Chaque drone communique en continu aux autres drones sa distance de son point de départ ou de la station au sol.

**Requis de conception**

- Des logs de débogages doivent être disponibles en continu lors de chaque mission.
- Le logiciel complet de la station au sol doit pouvoir être lancé avec une seule commande sur un terminal Linux.
- L'environnement virtuel pour les tests dans ARGoS doit pouvoir être généré aléatoirement.
- L'interface utilisateur doit être facile d'utilisation et lisible.

— Le système doit être conçu pour fonctionner avec un nombre arbitraire de drones, entre 2 et 10 drones.

**Requis de qualité**

— Le format du code doit être standardisé à travers le projet et suivre des conventions de codage reconnues et non des conventions maisons.

— Chaque composant du logiciel doit avoir un test unitaire correspondant.

## 4.2 Estimations des coûts du projet (Q11.1)

Pour estimer les coûts du projet, nous n'avons qu'à considérer les coûts reliés aux développeurs et au gestionnaire de projet. En effet, toutes les technologies utilisées pour s'assurer de la communication, de l'interface Web ainsi que du comportement des drones sont gratuites. De plus, les drones, les capteurs de flux optiques et de distance ainsi que la Crazyradio PA nous sont fournis, donc aucun coût ne sera relié au matériel physique.

Pour la réalisation de toutes les tâches, le nombre d'heures de travail est estimé à 630 heures-personnes au total. Le tout est réparti selon deux parties : le développement, pour un total de 605 heures de travail et la gestion du travail, pour 2 heures par semaine, donc 25 heures total. Le coût total du projet est donc de 82 275 $ et est calculé à l'aide de la formule suivante :

$$Cout = salaire_{dveloppeur} \times heure_{dev} + salaire_{coordonnateur} \times heure_{gestion}$$

$$Cout = 130 \times 605 + 145 \times 25 = 82275\$$$

## 4.3 Planification des tâches (Q11.2)

Ce tableau, mis à jour à la suite de la complétion des tâches du PDR, présente les tâches importantes et une estimation du nombre d'heures qui sera nécessaire pour les compléter. Chaque tâche a été assigné à une équipe de développement. L'équipe Simulation s'occupe de la simulation, l'équipe Embarqué, des drones, et l'équipe WebDev, de ce qui est relié au client et au serveur. Certaines tâches sont aussi reliées à tous les membres de l'équipe puisqu'elles requerront une participation de tous. Chaque tâche est aussi regroupée par jalon.

| Planification des tâches | | | |
|---|---|---|---|
| **Remise** | **Tâche** | **Nombre d'heures** | **Équipe de dévelopement** |
| **PDR** | Déterminer les tâches | 1 | Ensemble |
| | Mise en place de l'environnement pour la simulation | 2 | Simulation |
| | Mise en place de l'environnement du serveur et du client web | 2 | DevFullStack |
| | Établir la communication entre le serveur et la simulation | 20 | Simulation-DevFullStack |
| | Concevoir une interface de base | 6 | DevFullStack |
| | Lancer/arrêter la simulation à partir du client | 8 | Simulation |
| | Établir la communication entre le serveur et les drones | 12 | Embarqué |
| | Écriture du PDR | 24 | Ensemble |
| | Implémenter la commande "Identifier" sur les drones (DELs) | 8 | Embarqué |
| **CDR** | Concevoir le back-end | 16 | DevFullStack |
| | Finaliser l'interface client | 16 | DevFullStack |
| | Générer des obstacles aléatoires dans la simulation | 10 | Simulation |
| | Afficher les cartes en continu durant une mission | 18 | DevFullStack |
| | Générer des cartes de l'environnement à partir des capteurs | 20 | DevFullStack |
| | Afficher et enregistrer les logs pour chaque mission | 12 | DevFullStack |
| | Afficher le niveau de la batterie | 15 | Ensemble |
| | Implémenter l'évitement des obstacles pour les drones | 20 | Embarqué |
| | Implémenter l'évitement des obstacles dans la simulation | 20 | Simulation |
| | Implémenter l'algorithme d'exploration | 40 | Simulation-Embarqué |
| | Afficher l'état des drones sur l'interface | 16 | Ensemble |
| | Lancer/arrêter une mission à partir du client (drones) | 12 | Embarqué-DevFullStack |
| | Gestion de projet | 15 | Ensemble |
| | Révision CDR | 15 | Ensemble |
| **RR** | Adapter la simulation selon le nombre de drones | 10 | Simulation |
| | Lancer le logiciel à partir d'une seule commande | 8 | Ensemble |
| | Implémentation de la communication Peer to peer | 28 | Embarqué |
| | Implémentation du changement de la DEL selon la distance | 15 | Embarqué |
| | Implémenter la détection de crash pour les drones | 15 | Embarqué |
| | Mettre en place la base de données | 10 | DevFullStack |
| | Spécifier la position/l'orientation initiale des drones | 12 | DevFullStack |
| | Créer une seule carte à partir des données de tous les drones | 15 | DevFullStack |
| | Gérer la connection d'au moins 2 appareils à l'interface web | 24 | DevFullStack |
| | Gérer l'enregistrement des cartes (BD) | 8 | DevFullStack |
| | Implémenter la consultation des missions précédentes | 8 | DevFullStack |
| | Implémenter le retour à la base si la batterie est faible | 10 | Embarqué |
| | Implémentation du retour à la base pour la simulation | 37 | Simulation |
| | Implémentation du retour à la base pour les drones | 37 | Embarqué |
| | Gestion de projet | 10 | Ensemble |
| | Révision du système complet | 15 | Simulation |

## 4.4 Calendrier de projet (Q11.2)

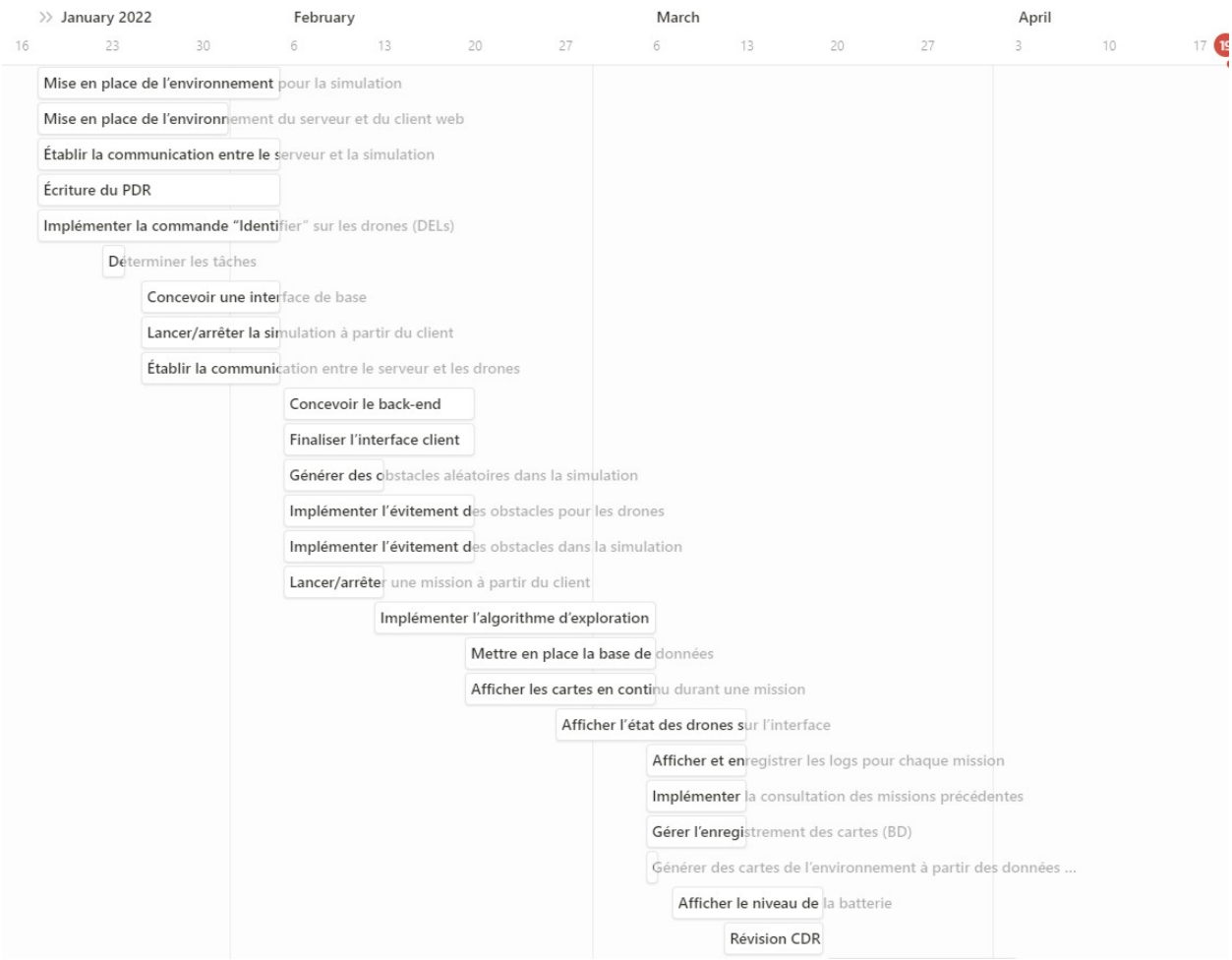
Ce tableau, mis à jour à la suite de la complétion des tâches du PDR, présente les tâches importantes et une estimation de la date à laquelle elles devraient être complétées. Toutes les tâches ayant une date antérieure à celle d'une remise doivent être totalement complété avant celle-ci.

| Calendrier de complétion | | |
|---|---|---|
| **Date** | **Tâche** | **Remise** |
| **23 Janvier** | Déterminer les tâches | |
| **28 Janvier** | Mise en place de l'environnement pour la simulation les tâches | |
| | Mise en place de l'environnement du serveur et du client web | |
| **31 Janvier** | Concevoir une interface de base | |
| | Établir la communication entre le serveur et la simulation | |
| | Établir la communication entre le serveur et les drones | |
| **2 Février** | Établir la communication entre le serveur et les drones | |
| | Implémenter la commande "Identifier" sur les drones (DELs) | |
| **4 Février** | Écriture du PDR | Remise PDR |
| **12 Février** | Générer des obstacles aléatoires dans la simulation | |
| | Lancer/arrêter une mission à partir du client | |
| **19 Février** | Concevoir le back-end | |
| | Finaliser l'interface client | |
| **5 Mars** | Afficher les cartes en continu durant une mission | |
| | Générer des cartes de l'environnement à partir des capteurs | |
| **12 Mars** | Afficher et enregistrer les logs pour chaque mission | |
| | Implémenter l'algorithme d'exploration | |
| | Afficher l'état des drones sur l'interface | |
| **18 Mars** | Afficher le niveau de la batterie | Remise CDR |
| | Implémenter l'évitement des obstacles pour les drones | |
| | Implémenter l'évitement des obstacles dans la simulation | |
| | Révision CDR | |
| **26 Mars** | Implémentation du retour à la base pour la simulation[1] | |
| | Implémentation du retour à la base pour les drones[1] | |
| **2 Avril** | Implémentation de la communication Peer to peer | |
| | Mettre en place la base de données[2] | |
| | Implémenter le retour à la base si le niveau de la batterie est faible[1] | |
| **13 Avril** | Adapter la simulation selon le nombre de drones | |
| | Afficher les cartes en continu durant une mission | |
| | Implémenter la consultation des missions précédentes[2] | |
| | Gérer l'enregistrement des cartes (BD)[2] | |
| **19 Avril** | Lancer le logiciel à partir d'une seule commande | Remise RR |
| | Implémenter la détection de crash pour les drones | |
| | Spécifier la position/l'orientation initale des drones | |
| | Révision du système complet | |

1. Nous allons tout d'abord développé le retour à la base pour nos drones. Il nous sera alors plus facile d'implémenter le retour selon le niveau de batterie puisque l'algorithme sera déjà créé, il ne nous restera qu'à suivre la diminution du niveau de la batterie.

2. Il faut tout d'abord mettre en place la base de données. Par la suite, nous implémenterons la page qui nous permettra de consulter les missions précédentes et finalement, nous enregistrerons les cartes.

Voici l'organisation de nos tâches présentée avec un diagramme de Gantt :


January 2022
February
March
April
16
23
30
6
13
20
27
6
13
20
27
3
10
17
Mise en place de l'environnement pour la simulation
Mise en place de l'environnement du serveur et du client web
Établir la communication entre le serveur et la simulation
Écriture du PDR
Implémenter la commande "Identifier" sur les drones (DELs)
Déterminer les tâches
Concevoir une interface de base
Lancer/arrêter la simulation à partir du client
Établir la communication entre le serveur et les drones
Concevoir le back-end
Finaliser l'interface client
Générer des obstacles aléatoires dans la simulation
Implémenter l'évitement des obstacles pour les drones
Implémenter l'évitement des obstacles dans la simulation
Lancer/arrêter une mission à partir du client
Implémenter l'algorithme d'exploration
Mettre en place la base de données
Afficher les cartes en continu durant une mission
Afficher l'état des drones sur l'interface
Afficher et enregistrer les logs pour chaque mission
Implémenter la consultation des missions précédentes
Gérer l'enregistrement des cartes (BD)
Générer des cartes de l'environnement à partir des données ...
Afficher le niveau de la batterie
Révision CDR

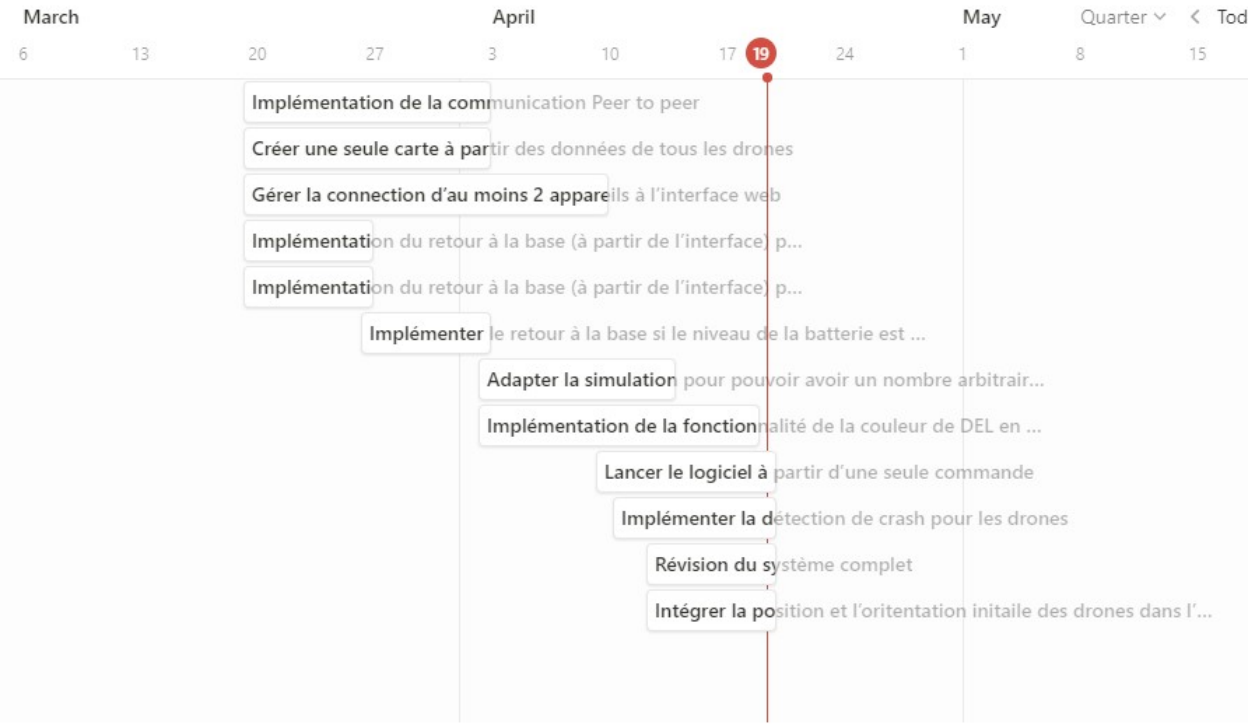



Figure 15 : Diagramme de Gantt

## 4.5 Ressources humaines du projet (Q11.2)

Pour effectué ce projet, un total de 6 personnes sont requises.

Afin de s'assurer que tous les membres de l'équipe soient au courant des différents changements apportés par chaque membre et que le projet respecte l'échéancier et les requis, Steven Duchêne a été assigné au rôle de gestionnaire de projet. Son expérience de gestion d'équipe qu'il a acquis durant son stage chez Hydro-Québec ainsi que sa connaissance personnelle de chaque membre de l'équipe faisait de lui un bon candidat. Son expérience en développement d'algorithme de recherche indépendant ainsi que le fait qu'il avait déjà travaillé avec des simulateurs sont les deux raisons pour lesquelles il a été assigné à l'équipe relié au développement de la simulation des drones.

Pour organiser nos réunions, nous avons décidé de travailler en suivant la méthodologie agile et nous avions besoin d'un SCRUM master afin d'organiser nos réunions au cours des semaines. Pour remplir ce rôle, nous avons choisi Félix Gauthier. Durant ses stages, il a rempli souvent ce rôle dans différents projets et tout indique qu'il sera adéquat pour cette tâche également.

Dans l'équipe Web et base de données, nous retrouvons Félix Gauthier, son expérience en développement web acquise lors de son projet final de deuxième année lui sera grandement utile. Également, il a de l'expérience en base de données NoSQL et MongoDB ce qui permettra au client d'enregistrer ses missions et de consulter celles qu'il a déjà réalisées. Juliette Morin fait aussi partie de cette équipe. Ayant fait un stage en tant que développeuse full-stack, elle pourra user de son expérience afin de développer l'interface web et sa connection au serveur. Tout comme Félix, elle a aussi, lors de son projet intégrateur 2 pu acquérir de l'expérience en développement web.

Du côté de l'équipe Simulation, Éloïse Brosseau a une bonne base en C++, ce qui est un atout utile, puisque le code de cette partie du projet devra être réalisé avec ce langage. Par ailleurs, ses connaissances avec le système d'expoitation Linux ainsi que ses systems calls seront pertinentes pour assurer une bonne communication inter-processus entre la simulation et le serveur. Son expérience en conception d'un serveur lors du projet intégrateur de deuxième année sera avantageux pour permettre la réception et assurer un bonne structure des données recueillies à partir de la simulation pour que celles-ci soient envoyées dans un bon format pour l'affichage du côté client.

Pour ce qui est de la partie embarquée, Samuel Crosilla est dans le société technique de la machine depuis 3 ans. Ceci fait en sorte qu'il a une large connaissance en systèmes embarqués, Docker et et l'architecture d'un serveur puisqu'il a déjà conçu, avec une équipe, plusieurs robots. Il travaillera avec Théo St-Denis qui a aussi de l'expérience dans le domaine en ayant suivi les quatre cours obligatoires en *hardware* (INF1500, INF1600, INF3500 et INF3610) en plus du cours processeurs embarqués configurables. De plus, ses expériences de stage en agrégation de données et en conteneurisation de serveur Flask lui permettera de faire lien entre l'équipe embarquée et les autres équipes du projet.

# 5 Suivi de projet et contrôle

## 5.1 Contrôle de la qualité (Q4)

Afin de s'assurer que le travail effectué est de qualité, un contrôle du code sera nécessaire au moyen de *merge requests* sur le répertoire Git de l'équipe. Chaque fonctionnalité sera implémentée sur une branche précise qui sera intégrée à la branche principale lorsque le code qu'elle contient est jugé satisfaisant. Chaque membre de l'équipe est encouragé à lire le code rédigé et à le commenter afin de demander des précisions ou à apporter des suggestions pour assurer que des erreurs de logiques ou le non respect de convention de programmation ne soient incorporer par mégarde. Par ailleurs, au moins une personne n'ayant pas travaillé sur la branche à ajouter à la branche principale devra donner son approbation à la *merge request* lorsque la personne ayant créé la *merge request* aura répondu à chaque commentaire et apporté les corrections nécessaires. Ce processus propose donc une approche itérative afin d'assurer que les livrables soumis aient été révisés de façon continue par l'ensemble des membres de l'équipe. Cela permettra également de cibler les erreurs d'implémentation et de conception le plus tôt possible, allouant ainsi suffisamment de temps pour corriger ces problèmes de manières adéquates avant la remise d'un livrable. De plus, l'utilisation de linters selon les langages utilisées pour les différents systèmes du projet permettront une uniformité du code ajouté à la branche *main*. En effet, tel que mentionné dans le README à la racine du projet, les linters ESLint, Flake8 et Cpplint permettront respectivement de formatter le code en suivant respectivement les style StandardJS pour ce qui est de JavaScript, PEP8 en ce qui concerne Python et Google C++ pour ce qu'il s'agit de C et C++.

## 5.2 Gestion de risque (Q11.3)

En regardant les erreurs possibles dans la conception du projet, il nous est possible de remarquer trois types de risques :

- Le premier risque est du domaine matériel et représente le bris du drone. Ce risque peut à la fois être peu important et avoir un grand impact. Dans le meilleur cas, le bris est minime et facilement

réparable. On peut parler ici d'une hélice qui se brise lorsqu'un drone entre en contact avec le mur par exemple. Dans le pire cas, le moteur ou la radio intégrée sont endommagés et le drone ne peut plus remplir ses fonctions. Nous n'aurions alors pas d'autre choix que de le remplacer complètement.

- Le second risque est un risque technique. Il est possible, avec les drones, qu'il y ait une imprécision des résultats obtenus à l'aide des capteurs de localisation. Il est aussi possible qu'il y ait de l'interférence, ce qui viendrait falsifier nos résultats pour la carte par exemple. Le drone pourrait détecter un mur alors qu'il n'y en avait pas réellement. Il est aussi possible que le débit de communication entre les drones et la station au sol présente des problemes de ralentissement au niveau de l'envoie des données. Finalement, la batterie des drones pourrait etre défectueuse et donc réduire le temps de vol et d'utilisation des drones.

- Le dernier risque est relié à la gestion du temps. On parle ici d'une modification de code qui mènerait à un dysfonctionnement des drones et qui serait non réversible. Pour avoir un impact majeur, le développeur pourrait se rendre compte de son erreur plusieurs jours après l'implémentation de l'erreur et ne plus se souvenir de son ajout dans le code. Cette erreur pourrait alors grandement ralentir le développement du projet puisqu'elle nous forcerait soit à passer plusieurs heures à chercher la solution, soit à revenir en arrière. Nous avons cependant déjà discuté de ce potentiel risque et nous avons conclu qu'il nous faut faire des commits régulièrement et s'assurer du bon fonctionnement avant de fusionner des branches. De plus, avec les nombreux requis à réaliser, il est facile d'oublier une tâche ou encore de sous-estimer la durée qu'elle prendra à réaliser. Pour s'assurer de rester à jour et de toujours remettre des comptes-rendus complets, nous nous rencontrons régulièrement pour faire le point sur les avancements. Au moindre problème d'un développeur, nous nous attendons que celui-ci aille chercher de l'assistance auprès de ses collèges.

## 5.3 Tests (Q4.4)

Dans le cadre de ce projet, nous devons réaliser une preuve de concept dont le résultat sera un prototype fonctionnel avec un niveau de maturité de la solution 4. Ainsi, à la fin du niveau 4, le but est une validation de principe qui repose sur l'intégration d'application et de concepts afin de démontrer que notre solution est viable [1]. Considérant cela, ainsi que notre date de remise finale qui n'est pas si éloignée dans le temps, notre solution finale ne comportera pas une grande quantité de tests unitaires. En effet, nous nous assurerons plutôt que chaque fonctionnalité se comporte comme désiré. Nous allons surtout nous concentrer sur le développement du prototype et de ces nombreuses fonctionnalités, car notre temps est limité et que le but ici est de montrer qu'il est possible de concevoir un tel système. Lors de phases de conception plus avancées et si le projet est accepté par l'Agence spatiale, il deviendra alors très important de tester le système de façon exhaustive.

### 5.3.1 Tests client

Commençons par les vérifications qui seront faites au niveau du client. Il faudra s'assurer que les boutons permettant la navigation sur les différentes pages mènent bien à la bonne page et qu'il est possible d'accéder à toutes les pages à partir de n'importe quel point de départ. Il faudra aussi vérifier que le client soit en mesure d'envoyer des requêtes vers le serveur, que ce soit pour envoyer une commande aux drones/simulation ou encore pour l'obtention de données à afficher. De plus, il faut vérifier que le client soit capable de recevoir la réponse du serveur. Le client devra aussi

tester la communication en temps réel, c'est-à-dire s'il est en mesure de recevoir et d'afficher les informations qu'il reçoit sans délai et à raison de 1Hz pour l'affichage des cartes par exemple. Nous effectuerons quelques tests unitaires avec le *framework* Jest. Il s'agit d'un *framework* de test très populaire, surtout pour ce qui est des applications développées sur React. Il est reconnu pour ça facilité d'apprentissage et d'utilisation, en plus d'être très simple à installer [16].

### 5.3.2 Tests serveur

Poursuivons avec le serveur qui sera testé par l'envoi de requêtes et de données vers le serveur et en sa partance. Nous nous servirons aussi de log dans la console afin de voir si le comportement est bien celui qui était attendu. Il faudra tester tout cela pour chaque autre système afin de s'assurer que la communication soit bien établie avec tous les systèmes. Une faille dans le serveur peut mener à une défaillance de la solution complète considérant que le serveur en est le coeur. Tout transige par lui. Pour le tester, nous allons observer son comportement et s'assurer que l'information transige correctement entre les différents systèmes avec, notamment, des *prints* dans la console. Ainsi, si l'affichage dans la console est valide pour chaque système avec lequel le serveur communique, il nous sera possible de considérer que le serveur gère bien l'ensemble des données qu'il reçoit et qu'il envoie.

### 5.3.3 Tests embarqué

Puis, le code embarqué pourra être testé en observant le comportement des drones dans la volière pour les fonctionnalités nécessitant le vol des drones. Sinon les autres fonctionnalités peuvent être testées avec les drones stationnaires comme la communication avec le serveur, l'identification des drones, leur communication P2P, l'échange de distance pour la couleur des DELS, la fonctionnalité de crash, etc. Nous allons donc choisir le comportement que nous voulons tester, n'appeler que les fonctionnalités qui sont nécessaires pour que le comportement voulu soit observé, et ce, à plusieurs reprise. Ensuite, pour le même comportement, nous allons tester différents cas, comme les cas limites afin de voir si tout fonctionne peu importe les conditions. Pour s'assurer de minimiser les bris nous avons établi un protocole de test simple. La première étape consiste à tester les fonctionnalités souhaitées à l'aide de la simulation. Lorsque celles-ci fonctionnent de façon adéquate, chaque fonctionnalité sera testée de manière individuelle sur un drone. Enfin, lorsque l'implémentation sur un drone sera fonctionnelle, nous testerons sur deux drones afin d'observer le comportement complet attendu.

### 5.3.4 Tests simulation

Ensuite, concernant la simulation, grâce à l'interface d'ARGoS, la même stratégie de tests que les drones physiques sera adoptée. En effet, les requis pour les drones physiques sont en grande majorité les mêmes que pour les drones en simulation. Il est toutefois beaucoup plus facile d'observer le comportement des drones dans la simulation, puisque ARGoS nous fournit une interface où il est possible de voir les drones en action. La simulation nous permet donc de tester une fonctionnalité dans le monde virtuel avant de l'essayer dans le monde réel évitant ainsi de causer des bris de drones en raison d'une collision par exemple.

### 5.3.5 Tests base de données

Finalement, pour la base de données, il est très simple, grâce à l'interface utilisateur offert par MongoDB Atlas, d'aller consulter le contenu de nos tables. Nous pourrons ainsi facilement envoyer des données vers la base de données et aller vérifier si les données sont bien enregistrées et si la table se remplit comme nous le désirions. Dans le sens inverse, nous pourrons aussi facilement vérifier le contenu de la base de données pour ensuite l'afficher dans l'interface client et voir si le transfert s'est effectué comme il se doit.

Toutes les procédures nous permettant de tester nos fonctionnalités sont décrites en détail dans un document nommé « test.pdf » à la racine du projet afin de répondre au requis R.Q.2.

## 5.4 Gestion de configuration (Q4)

Pour le contrôle des versions de notre projet, nous fonctionnons avec Gitlab. Le développement se fait à partir de la branche nommée *dev*. Ainsi, on s'assure que tout notre code est fonctionnel avant de le mettre sur la branche principale du projet, soit celle du *main*. Lors du développement, une branche est créée pour le développement de chaque fonctionnalité. Une fois que l'implémentation est complétée et fonctionnelle, un *merge request* est fait sur la branche *dev*, se fait accepter ou refuser après un *code review*, puis la branche est supprimée. La documentation expliquant comment faire fonctionner le projet sera écrite sous la forme d'un README à la racine du projet sur Gitlab. Toutes autres documentations, comme les rapports et les remises de livrables écrits, seront placer dans des répertoires dédiés.

Pour ce qui est de l'organisation du code, chaque système a un répertoire particulier où tous les fichiers pour le développement sont placés. Pour le client, le code est dans le dossier *src*. Chaque composant à aussi un dossier qui lui est réservé. Les répertoires de tous les composants sont ensuite rassemblés sous un seul dossier. Pour le serveur, les fichiers de développement sont regroupés sous *app-server*. Ensuite, les fichiers pour la simulation se trouvent dans le répertoire *src* où les différents modules importés d'ARGoS se retrouvent. Finalement, pour le code embarqué, le code est écrit dans le dossier *src*.

## 5.5 Déroulement du projet (Q2.5)

Durant le déroulement du projet, le respect de l'échéancier a été difficile à réaliser. En effet, certaines tâches nous ont retardées dans le développement ce qui nous a forcé à faire des sacrifices sur certains requis. Initialement, nous avions prévu gérer l'enregistrement des cartes et des informations de la mission dans la base de données pour la remise du CDR (R.F.17 et R.F.18). Cependant, nous n'avions pas pris en compte les retards potentiels des autres tâches plus prioritaires. C'est pour cette raison que nous avons préféré repousser le développement de la BD étant donné que celle-ci n'avait pas d'impact sur la réalisation des autres requis prévus dans l'échéancier du projet. L'implémentation de la communication entre chaque drone de la simulation et le serveur ainsi que l'envoi des données dans le but de générer les cartes ont été les deux tâches principales qui ont provoqué un retard par rapport à l'échéancier. Ce n'était pas les tâches en tant que tel qui ont retardé l'avancement, mais la vision tunnel des développeurs, qui cherchaient à terminer les tâches rapidement pour aider les autres. Cela a provoqué des erreurs et donc ralenti le processus de développement. Les deux requis mentionnés plus haut ont donc été implémentés pour cette remise finale.

Néanmoins, l'équipe a su surmonter ces imprévus avant la remise du CDR en redistribuant les tâches à compléter au sein des membres et en s'assurant que chacun prenait le temps de bien faire les choses pour garantir que les erreurs incorporer plus tôt étaient réglées et qu'il n'y en avait pas davantage qui s'ajoutaient. D'ailleurs, cette même stratégie a été employée pour la réalisation du RR. L'équipe s'est assurée que les tâches étaient bien réalisées dès le début afin d'éviter que des fonctionnalités aient à être implémenter une seconde fois pour corriger des erreurs engendrer par des développeurs voulant aller trop vite. Pour ce qui est des autres requis anticipés au début du projet pour cette dernière remise, aucune difficulté marquante n'a été soulevée, outre la difficulté à bien évaluer le temps nécessaire à la réalisation d'une tâche. Ainsi, certains requis facultatifs ont dû être retirés de la planification afin de s'assurer que suffisamment de temps était imparti à une réalisation adéquate des requis obligatoires.

# 6 Résultats des tests de fonctionnement du système complet (Q2.4)

Pour ce qui concerne les tests de fonctionnement, nous avons effectué différents tests sur chaque partie du système pour vérifié que les fonctionnalités étaient implementées de la façon souhaitée. Tout d'abord, nous avons affiché les données reçu par le client pour s'assurer que celle-ci étaient adéquates et dans le format attendu. Par la suite, nous avons vérifié que les requêtes envoyées et reçues par le client et le serveur étaient conformes à la communication souhaitée entre ces deux parties du systèmes en affichant celle-ci lors de chaque envoi et réception des données. Par ailleurs, nous avons testé de façon visuelle l'affichage des positions des drones à raison de 1Hz sur la carte. Ensuite, nous avons vérifié le comportement à reproduire par les drones physiques grâce à l'implémentation de ces derniers dans la simulation. Par exemple, nous avons testé l'implémentation de l'algorithme d'exploration à l'aide de la simulation pour vérifier que celui-ci était satisfaisant. Par ailleurs, nous avons confirmé que les drones physiques étaient capable de se déplacer dans l'environnement grâce à l'algorithme d'exploration et que ceux-ci étaient capable d'éviter les obstacles qu'ils rencontraient, tout en s'évitant entre eux, en les faisant se déplacer dans la volière et en plaçant des objets dans la direction où ils se dirigeaient pour observer leurs changements de direction. De plus, nous avons vérifié le bon fonctionnement du retour à la base, et ce, autant pour les drones physiques que simulés. Grâce à des tests comportementaux mentionnés dans le fichier test.pdf à la racine du projet, nous nous sommes assurés que que les drones sont capables d'effectuer leur chemin inverse afin de retourner à leur point de départ, tout en s'évitant entre eux et en évitant les obstacles rencontrés.

Afin de s'assurer que le fonctionnement du système complet répond à tous les requis. Nous avons exécuté le projet à l'aide du fichier start.sh. Une fois toutes les parties du système démarrées, nous avons lancé une mission en simulation de façon à voir les cartes individuelles ainsi que la carte commune affichées l'environnement explorée par les drones. De plus, le bouton d'affichage des logs nous a permis de confirmer que l'affichage était représentatif des données envoyées. Si nous attendons suffisamment longtemps, nous avons observé que les drones effectuent un retour à la base si leur niveau de batterie est en-bas de 30%. Par ailleurs, les drones font un atterissage immédiat si le bouton « terminer mission » est appuyé. De plus, nous avons observé que les drones se déplacent dans l'environnement selon notre algorithme de *random walk* pour explorer, tout en évitant les différents obstacles rencontrés. De la même façon, nous avons observé que les drones physiques reproduisaient le même comportement dans l'arène.

# 7 Travaux futurs et recommandations (Q3.5)

Lors de la rédaction de l'appel d'offres, nous avions planifié le développement de notre projet en considérant tous les requis, autant ceux qui étaient obligatoires que ceux qui étaient facultatifs. Toutefois, le manque de temps causé par la difficulté d'implémentation de certains requis obligatoires nous a forcé à revoir notre planification et à abandonner 4 requis. En effet, les requis permettant la mise à jour du logiciel de contrôle sur les drones (R.F.14), la modification du code des contrôleurs de drones (R.F.16) ainsi que la spécification d'une zone de sécurité (R.F.20) ont dû être mis de côté. La prochaine étape dans le développement de notre système aurait donc été de compléter ces exigences.

Puisque le temps de conception était limité, nous n'avons pas pu développer des comportements qui n'étaient pas déjà présent dans la liste de requis. Nos réflexions nous avaient néanmoins amené à penser à plusieurs extensions. Nous aurions beaucoup aimé implémenter dans l'interface un champ qui permettrait à l'utilisateur de définir le nombres de drones qu'il compte utiliser pour sa mission. Cetet option aurait été implémenté pour la simulation ainsi que pour les drones physiques. Nous aurions aussi aimé pouvoir développer des cartes en trois dimensions. Ce type de carte permettrait aux utilisateurs d'avoir une meilleur compréhension de l'environnement dans lequel les drones se déplacent.

Afin d'améliorer le comportement de notre système, nous recommandons d'améliorer la communication avec les drones. Notre suggestion est de développer davantage la communication unique avec chaque drone. Ceci permettrait de faire atterrir un seul drone ou bien de faire revenir un drone à sa base alors que le second pourrait continuer d'explorer. Nous recommandons aussi l'optimisation de notre algorithme d'exploration. Comme le requis décrivant l'algorithme d'exploration ne mentionnait pas le niveau d'efficacité que le drone devait atteindre, celui-ci explore son environnement selon un algorithme complètement aléatoire. En effet, lorsqu'un drone rencontre un mur, une direction autre que celle où le mur se trouve est générée. Toutefois, le déplacement que le drone était en train d'effectuer n'est jamais sauvegardé. Il est donc possible que le drone effectue des allers-retours. Il serait donc astucieux de considérer son déplacement précédent afin d'augmenter l'efficacité de son exploration.

# 8 Apprentissage continu (Q12)

## 8.1 Steven Duchêne

1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

J'ai remarqué des lacunes relatives à mes connaissances pour les technologies telles que Docker et ARGoS ainsi que sur mon processus de développement. Comme mes connaissances par rapport à certaines technologies étaient moins développées, j'avais parfois l'impression d'être moins efficace que mes collègues. Ce sentiment m'a donc poussé à vouloir rédiger mon code plus rapidement, ce qui a parfois créer des erreurs qui auraient pu être facilement évités.

2. Méthodes prises pour y remédier.

Afin de remédier à mes lacunes, j'ai fait beaucoup de recherche sur les technologies qui me posaient problème et j'ai essayé de communiqué avec mes partenaires le plus possible pour m'assurer que j'appliquais correctement ce que mes recherches m'avaient appris. De plus, je me suis forcé à plusieurs reprises à prendre mon temps et y aller une étape à la fois afin de m'assurer de ne pas sauter d'étapes.

3. Identifier comment cet aspect aurait pu être amélioré.

Dans le but d'être davantage efficace dès le début, j'aurais pu essayer de me familiariser avec ces technologies avec un projet personnel. J'aurais alors démarré la conception du projet avec des connaissances et une expérience plus développées, ce qui m'aurait permis d'être plus efficace, mais aussi de ne pas avoir le sentiment d'être moins efficace.

## 8.2 Félix Gauthier

1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

Tout d'abord, dans les savoirs j'ai remarqué que j'avais quelques faiblesses en développement web et plus particulièrement React. En effet, je ne connaissais pas bien les mécanismes de ce cadriciel alors au début j'ai perdu un peu de temps en programmant d'une manière qui n'était pas compatible en React alors mon code ne fonctionnait pas et je ne comprenais pas bien pourquoi. Aussi, j'ai eu beaucoup de difficultés avec les tests du client web, car j'étais habitué à tester des applications avec Angular mais jamais avec React. Je connaissais plus ou moins bien les bonnes librairies à utiliser alors j'ai également perdu du temps sur cet aspect du projet. Du côté savoir-faire j'ai remarqué que je manquais d'organisation pour le travail d'équipe. Comme j'étais la personne qui, en général, organisait les réunions il arrivait que je ne le fasse pas par oubli et nous ne nous sommes pas très régulièrement en début de projet. Également, j'avais quelques outils avec l'outil git ce qui m'a ralenti quelque fois, car j'avais de la difficulté à résoudre les conflits dans le code.

2. Méthodes prises pour y remédier.

Pour m'améliorer en React, j'ai consulté sites d'apprentissages tels que GeekForGeek [6] ainsi que OpenClassRoom [11]. Également, j'ai beaucoup consulté des questions sur StackOverFlow lorsque j'avais des questions spontanées sur des petits problèmes que j'avais. J'ai également consulté les documentations officiels des différentes librairies de tests que j'ai utilisé soit Enzyme [3] et Jest [8]. Comme je n'avais jamais eu de cours sur aucune des technologies mentionnées j'ai dû me débrouiller en faisant principalement de la recherche. Pour ce qui est de l'organisation, j'ai pris l'habitude de me faire un calendrier personnel avec des objectifs ainsi que des dates importantes afin d'assurer le bon déroulement de mes activités et d'organiser suffisament de réunions.

3. Identifier comment cet aspect aurait pu être amélioré.

Afin d'éviter des problèmes avec React qui étaient facilement évitable si j'avais eu les connaissances suffisantes, j'aurais dû lire plus de documentation avant de commencer directement à développer des nouvelles fonctionnalités. Cela m'aurait fait sauvé beaucoup de temps, car je n'aurais pas écrit de code problématique qui a dû être changé par la suite. Cela s'applique autant au développement du client qu'à l'écriture des tests pour ce dernier. Pour l'organisation des réunions, j'aurais dû commencé

à travailler avec un agenda dès le début. De plus, j'aurais dû organiser une réunion de cohésion ce qui m'aurait rendu plus à l'aise d'organiser des réunions, car j'aurais connu mieux les membres de l'équipe.

## 8.3 Éloise Brosseau

1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

Bien que j'avais déjà travaillé sur un projet faisant l'utilisation de Docker, je n'avais jamais eu à créer les containers et leurs fichiers de configuration respectifs. Cette lacune m'a nuit, particulièrement en début de projet, puisque j'ai eu de nombreux problèmes à comprendre comment faire fonctionner la communication entre le serveur et la simulation lorsqu'ils se trouvaient à l'intérieur des containers. En effet, cela est un problème qui a eu un impact sur la vitesse à laquelle cette fonctionnalité a été implémentée.

2. Méthodes prises pour y remédier.

Afin de régler ces lacunes, j'ai pris le temps de faire des recherches pour bien comprendre le fonctionnement de Docker ainsi que de la communication par socket entre containers. Par ailleurs, je cherchais les erreurs affichées dans ma console sur internet pour tenter de voir si quelqu'un n'aurait pas eu le même problème que moi. J'ai tenté de verbaliser le plus possible mes problèmes aux autres membres de mon équipe lors de nos réunions en leur expliquant les solutions que j'avais essayées et ce que je comprenais du problème.

3. Identifier comment cet aspect aurait pu être amélioré.

Cet aspect aurait pu être amélioré si j'avais pris plus de temps pour me familiariser avec Docker avant le début du projet. Cela m'aurait permis de ne pas perdre de temps à comprendre cette technologie, puisque j'aurais déjà eu les connaissances nécessaires pour réaliser ce qui était demandé. De plus, j'aurais dû prendre plus de temps pour m'assurer que je comprenais bien ce que je faisais afin d'éviter de m'entêter à régler un problème sans savoir exactement la source de celui-ci.

## 8.4 Samuel Crosilla

1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

Pour ce qui est du côté programmation, je suis perfectionniste par rapport à la façon dont le code est écrit. Ceci m'amène a souvent refactoriser le code. Du côté travaille avec l'équipe, j'ai remarqué que je travaille mieux seul. Ceci a fait en sorte que je communiquais moins avec mon équipe.

2. Méthodes prises pour y remédier.

Pour remédier au côté perfectionniste, avant de changer le code complètement, je demandais a mon équipe si les changements étaient nécessaires ou seulement superficiels. Pour ce qui est du travaille d'équipe, je me suis trouvé des périodes de travail seul où j'étais le plus efficace et d'autres périodes de travail où j'étais avec l'ensemble de l'équipe.

3. Identifier comment cet aspect aurait pu être amélioré.

Mon côté perfectionniste aurait pu être plus avantageux pour l'équipe si, pendant la période du début conception, j'avais pris les devant et aider a mieux concevoir la structure du code. Pour le travail d'équipe, organiser de la cohésion et me mettre de l'avant aurait aider à mes performances lors du projet.

## 8.5 Juliette Morin

1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

En programmation, j'avais des lacunes au niveau de React et de python. En effet, je n'avais encore jamais travaillé avec ces technologies. Bien que j'avais déjà travailler avec Angular, un autre quadriciel pour le développement web, j'ai dû apprendre les particularités de React. Cela a considérablement affecté ma productivité lors du développement, puisque j'avais constamment des erreurs ou des comportements non voulus. Pour ce qui est de python, lorsque le temps est venu de lire le code que mes coéquipiers avaient écrit, j'ai eu de la difficulté à comprendre rapidement ce que faisait certaines lignes. Je ne connaissais pas la syntaxe particulière de python.

2. Méthodes prises pour y remédier.

Afin de remédier à cela, j'ai commencé par faire des recherches par rapport à React et ces particularités. Ensuite, lorsque je devais implémenter une fonctionnalité que je n'avais pas rencontrée dans mes recherches, je regardais d'autres exemples qui était directement en lien avec mon besoin. Je prenais le soin de mettre des "console.log()" dans mon code afin de bien comprendre la logique et la suite des événements. Dans le cas où je recevais des erreurs, je retournais faire des recherches pour comprendre ce qui pouvait les causer. Je prenais aussi le temps de demander à mes coéquipiers s'ils avaient déjà rencontré ses erreurs afin d'éviter de passer des heures à chercher la cause, si quelqu'un avait déjà trouvé une solution à ce problème. J'ai aussi poser beaucoup de questions à mes collègues concernant le serveur flask et python, puisque ceux-ci avaient déjà travaillé avec ces technologies.

3. Identifier comment cet aspect aurait pu être amélioré.

Cet aspect aurait pu être améliorer si un effort avait été fait dès le départ du projet pour se familiariser avec les différentes technologies que j'allais devoir utiliser tout au long de la session. En effet, j'aurais pu prendre le temps de créer un petit projet simple avec React pour comprendre les particularité de cette librairie. Ainsi, j'aurais éviter de devoir chercher chaque élément durant toute la session et le développement du client se serait fait beaucoup plus rapidement. De plus, comme je n'étais pas familière avec le développement d'un serveur et du langage python, j'aurais dû m'impliquer dans son développement dès le départ pour mieux le comprendre, au lieu d'arriver au milieu du développement et d'essayer d'ajouter ma partie.

### 8.6 Théo St-Denis

1. Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.

Ma plus grosse lacune que je peux identifier est dans mon savoir-faire. Durant le projet, j'avais de la difficulté à appliquer et à encadrer mes idées. J'avais l'impression de ne pas avoir assez de pratique et ceci à ralenti la réalisation des requis qui m'étaient assignés. De plus, même si j'avais déjà utilisé Docker durant mes stages et dans des cours, mes connaissances en conteneurisation reste limitées et rudimentaires. Le démarrage des tâches en début de projet était aussi difficile et lent. Finalement, mon aversion pour toute technologie web a nuit à l'implémention de la base de donnée et à tout autre tâche qui m'était assignée en lien avec le client.

2. Méthodes prises pour y remédier.

En ce qui concerne le savoir-faire, il n'y avait pas d'autre solution que pratiquer et travailler d'arrache-pied afin d'en accumuler. Sinon, Docker est, en général, une technologie assez simple à utilisé. Après avoir passé la petite courbe d'apprentissage au début du projet, je suis devenu très familier. Finalement, pour ce qui est du web, j'ai dû consacré une fin de semaine seulement pour me familiariser avec notre client et son framework React.

3. Identifier comment cet aspect aurait pu être amélioré.

La solution à ces multiples problèmes est assez claire ; je dois réaliser plus des projets personnels à l'extérieur de mes heures de cours et de stage afin de me pratiquer et d'atténuer la perte de savoir et, surtout, de savoir-faire.

## 9 Conclusion

En prenant le temps de faire une rétrospection sur notre vision initiale du système lors du dépôt de la réponse à l'appel d'offre, nous nous rendons compte que nous avions grandement sous-estimé la difficulté de conception du projet. En effet, notre première approche avait été de bâtir notre horaire du temps en considérant tous les requis mentionnés dans le document de requis. Cependant, durant le développement, un des obstacles qui nous a posé problème a été la gestion du temps. Nous avons été ralenti à plusieurs reprises dans le développement de nos requis par des fonctionnalités qui nous ont pris beaucoup plus de temps que prévu. Toutefois, comme la cohésion dans notre équipe était hors-pair, nous avons réussi à nous adapter et modifier notre stratégie afin de compléter les requis que nous considérions comme essentiels. Nous avons aussi remarqué que comme le projet était très vaste et utilisait plusieurs technologies, il nous a été compliqué de choisir dès le départ la bonne façon de faire. Comme plusieurs options s'offraient à nous et que nous n'en maîtrisions aucune, nos choix initiaux se sont fait en se basant sur les quelques connaissances que nous avions acquises lors nos projets intégrateurs précédents ainsi que sur des recherches brèves que nous avions effectuées au début du projet. Or, ceux-ci n'étaient pas toujours appropriés pour le projet actuel. Ceci nous a donc parfois forcé à changer les technologies que nous utilisions en cours de route. En bref, malgré

quelques embûches rencontrées tout au long du projet, nous avons réussi à rester efficace et nous avons accompli presque tous les requis que nous nous étions planifié.

## Références


[1] Gouvernement du Canada. *Niveau de maturité de la solution*. url : https://www.canada.ca/fr/ministere-defense-nationale/programmes/idees-defense/niveau-maturite-solution.html.

[2] Hiren Dhaduk. *Angular vs React 2022 : Which JS Framework your Project Requires ?* url : https://www.simform.com/blog/angular-vs-react/#quick.

[3] Enzyme. *Enzyme*. url : https://enzymejs.github.io/enzyme/.

[4] Facebook. *DOM virtuel et autres détails*. url : https://fr.reactjs.org/docs/faq-internals.html.

[5] Figma. *Figma : the collaborative interface design tool*. url : https://www.figma.com.

[6] GeeksForGeeks. *ReactJS Tutorials*. url : https://www.geeksforgeeks.org/reactjs-tutorials/.

[7] IEEE. *ARGoS : A modular, multi-engine simulator for heterogeneous swarm robotics*. url : https://ieeexplore.ieee.org/document/6094829/authors#authors.

[8] Jest. *Jest*. url : https://jestjs.io/.

[9] MongoDB. *MongoDB Python Drivers*. url : https://docs.mongodb.com/drivers/python/.

[10] MongoDB. *MongoDB Wire Protocol*. url : https://docs.mongodb.com/manual/reference/mongodb-wire-protocol/.

[11] OpenClassRoom. *Débutez avec React*. url : https://openclassrooms.com/fr/courses/7008001-debutez-avec-react.

[12] Pallets. *Flask web development, one drop at a time*. url : https://flask.palletsprojects.com/en/2.0.x/.

[13] Leodanis Pozo Ramos. *Python and MongoDB : Connecting to NoSQL Databases*. url : https://realpython.com/introduction-to-mongodb-and-python/.

[14] Tonya Sims. *The Ultimate Face-off : Flask vs. FastAPI*. url : https://learn.vonage.com/blog/2021/08/10/the-ultimate-face-off-flask-vs-fastapi/.

[15] Vijay Singh. *Flask vs Django in 2022 : Which Framework to Choose ?* url : https://hackr.io/blog/flask-vs-django.

[16] Testim. *What Is the Best Unit Testing Framework for JavaScript ?* url : https://www.testim.io/blog/best-unit-testing-framework-for-javascript/.