



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

Département de génie informatique et génie logiciel


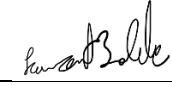


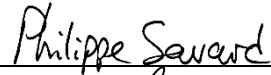

INF3995

Projet de conception d'un système informatique

Proposition répondant à l'appel d'offres
no. H2021-INF3995 du département GIGL.

Conception d'un système aérien minimal pour exploration

Équipe No 6

Matthew Paoli	
Laurent Bolduc	
Simon Tran	
Jacob Brisson	
Philippe Savard	
Jordan Lecourtois	

8 Mars 2021

Table des matières

1. Vue d'ensemble du projet	2
1.1 But du projet, porté et objectifs	2
L'interface Web	2
La station au sol	2
La partie embarquée	3
La partie simulation	3
1.2 Hypothèse et contraintes	3
1.3 Biens livrables du projet	4
2. Organisation du projet	7
2.1 Structure d'organisation	7
3. Solution proposée	9
3.1 Architecture logicielle générale	9
3.3 Architecture logicielle station au sol	13
4. Processus de gestion	16
4.1 Estimations des coûts du projet	16
4.2 Planification des tâches	16
4.3 Calendrier de projet	19
4.4 Ressources humaines du projet	21
5. Suivi de projet et contrôle	22
5.1 Contrôle de la qualité	22
5.2 Gestion de risque	22
5.3 Tests	23
5.4 Gestion de configuration	24
6. Références	25

1. Vue d'ensemble du projet

1.1 But du projet, porté et objectifs

Le mandat est de concevoir un groupe de logiciels qui permettent, à travers une interface Web, d'envoyer des commandes à un essaim d'un nombre arbitraire de drones afin que ceux-ci explorent et cartographient, de manière autonome, une pièce d'au maximum 100 m². Notre plateforme Web permet donc à un opérateur de contrôler l'état des missions d'exploration, en plus de visualiser, en temps réel, la cartographie de la pièce explorée par les drones ainsi que des renseignements quant à leur vitesse, position et niveau de batterie. Le mandat requiert également l'implémentation de notre logique d'exploration dans une simulation ARGoS [8], afin de vérifier le comportement d'une mission d'exploration avec un nombre arbitraire de drones. La simulation permet également d'assurer le bon fonctionnement de la logique d'exploration avec l'intégration sur les drones physiques. Il est donc possible de séparer le projet en quatre parties : l'interface Web, la station au sol, le logiciel embarqué ainsi que la simulation.

L'interface Web

Cette dernière permet à l'opérateur de contrôler le lancement et l'interruption des missions, tout en visualisant la carte générée par les drones ainsi qu'une multitude de renseignements sur l'état des drones. La plateforme web communique seulement avec la station au sol, qui héberge le back-end.

Pour ce qui en est de l'interface, les requis globaux sont les suivants :

- Les commandes permises sur l'interface sont de faire décoller les drones, de les faire atterrir, de mettre à jour leur logiciel et de les ramener à la base, où le poste d'opération se trouve.
- Les informations recueillies et affichées sur la plateforme seront le nombre de drones, l'état des drones, la vitesse courante des drones, le niveau de batterie des drones et la carte générée par ceux-ci.

La station au sol

Celle-ci héberge le back-end qui sert d'interface entre la plateforme web, les drones et la simulation. Le back-end implémente le patron adaptateur afin de permettre à l'opérateur d'interagir avec la simulation et avec les drones réels de façon identique. Il transmet les commandes reçues par l'interface Web aux drones ou à la simulation, dépendamment du mode activé, en plus de transmettre les informations des drones vers l'interface Web. Afin de communiquer avec les drones, la station au sol utilise une Bitcraze Crazyradio PA[3] qui est connectée sur un port USB de celle-ci.

Les requis globaux sont les suivants:

- Les informations recueillies et transférées à l'interface Web sont les suivantes: le nombre de drones, l'état des drones, la vitesse courante des drones, le niveau de batterie des drones et la carte générée par ceux-ci.
- Toutes les communications avec les drones seront faites à travers des envois de paquets grâce à la Bitcraze Crazyradio PA[3].

La partie embarquée

La partie embarquée est le logiciel qui est installé sur le microcontrôleur des drones et qui leur permet de parcourir la pièce en évitant les obstacles et en se communiquant entre eux.

Pour la partie embarquée, les requis principaux à implémenter sont les suivants:

- Un nombre arbitraire, mais d'au moins deux drones devra parcourir une pièce de 100 m² maximum en évitant les obstacles et en se communiquant avec l'API P2P de Bitcraze[7].
- Le retour à la base devra poser les drones à une distance de 1m maximum du poste de contrôle. Ce retour devra être automatique lorsque la batterie des drones passe sous la barre des 30%. Un drone ne peut donc pas décoller avec un niveau de batterie inférieur à 30%.

La partie simulation

Celle-ci est le logiciel de test qui est implémenté dans ARGoS. La simulation assure une bonne logique d'exploration en plus de réaliser une mission avec un nombre arbitraire de drones. Elle est utilisée afin d'observer le comportement des drones avant d'adapter le code pour la partie embarquée.

Pour la simulation, les requis principaux sont les suivants:

- La simulation complète du système doit générer un environnement aléatoirement et doit pouvoir être exécutée avec une seule commande.

En ce qui concerne les biens livrables attendus, le projet est divisé en 3 remises, soient la PDR, la CDR et la RR. Ces trois livrables seront détaillés dans la section "1.3 Biens livrables du projet".

1.2 Hypothèse et contraintes

Pour la réalisation des logiciels demandés, plusieurs hypothèses sont de mises. En effet, nous faisons l'hypothèse que la connexion entre notre station au sol et les drones, ainsi que la connexion entre notre station au sol et l'interface Web sont assez rapides pour permettre le transfert des informations voulues à une fréquence minimale de 1 Hz. De plus, nous posons l'hypothèse qu'il est possible d'utiliser un back-end en Python afin de contrôler les drones réels ainsi que ceux simulés dans ARGoS, en plus

que de communiquer avec la plateforme Web (les détails de notre architecture seront expliqués dans la section 3.1).

Nous posons également l'hypothèse qu'il est possible d'utiliser une librairie commune de fonctions implémentant la logique d'exploration pour les drones réels et pour ceux simulés dans ARGoS, afin de réutiliser le code créé pour la simulation. Cela implique que la simulation ARGoS soit suffisamment représentative de la réalité physique pour permettre une seule implémentation qui convient aux deux situations. Nous supposons également qu'il est possible de réutiliser des fonctions fournies dans l'API Python de Bitcraze[2] afin de communiquer avec le drone à travers la Bitcraze Crazyradio PA[3].

Pour les contraintes, il est évident que le code embarqué sur les drones peut être testé pour un nombre arbitraire de drones dans la simulation. Or, dans la réalité, nous sommes contraints à utiliser exactement 2 drones, à cause des limitations matérielles. De plus, le temps de travail des membres de l'équipe ne doit pas excéder 630 heures-personnes, sans quoi la solution fournie n'est pas acceptée. Enfin, il est nécessaire de pouvoir exécuter la simulation ainsi que le reste des logiciels avec une seule commande.

1.3 Biens livrables du projet

1.3.1 - PDR

Le premier livrable est la PDR (Preliminary Design Review). Pour cette remise, il est nécessaire de présenter une simulation de deux drones suivant un parcours quelconque dans le logiciel ARGoS. De plus, il faut que le serveur Web soit créé et disponible sur la station au sol pour pouvoir allumer et éteindre les DEL des drones avec un bouton et pour afficher le niveau de batterie de ceux-ci. Cette remise peut donc être séparée en deux parties:

1. Faire fonctionner la simulation ARGoS avec deux drones et effectuer des tests et des modifications afin de comprendre le fonctionnement de ce logiciel et de pouvoir les faire voler à notre convenance.

Détail du livrable : Le code remis pour la simulation doit être en mesure d'afficher une arène valide contenant deux drones. Ces drones doivent être en mesure d'exécuter des commandes de décollage et d'exploration quelconque dans l'arène.

2. Créer la page Web et la mettre disponible sur le réseau, pour ensuite comprendre la méthode de communication entre le poste au sol et les drones réels afin de pouvoir effectuer la connexion entre l'interface et les drones.

Détail du livrable : Ce livrable comprend le code du Micrologiciel du back-end et du front-end permettant de contrôler la DEL. Ce même code permet également d'afficher le niveau de batterie sur le serveur Web.

Nous devons également remettre une première version de la réponse à l'appel d'offres.

Ce livrable est dû le 15 février 2021.

1.3.2 - CDR

Le second livrable est la CDR (Critical Design Review). Pour cette remise, il question de présenter une simulation dans ARGoS avec 4 drones volant en utilisant les capteurs de distance pour éviter les obstacles et les murs générés aléatoirement. Pour la plateforme Web, il doit être possible d'envoyer les commandes de décollage et de retour à la base aux drones simulés. De plus, les informations des drones simulés et réels doivent pouvoir être aperçues sur la plateforme Web en direct. La visualisation de la carte en direct doit être au niveau du prototype. Pour les drones réels, le code embarqué doit être capable d'envoyer les données requises à l'interface. Il est alors possible de diviser la tâche en sous-étapes :

1. Implémenter la détection et l'évitement de collision entre les drones ainsi que la génération d'obstacle de manière aléatoire dans la simulation.

Détails du livrable : Ce livrable comprend le code fonctionnel de la simulation permettant à 4 drones d'éviter les obstacles tels que les murs et les autres drones. Cette fonctionnalité doit également assurer que ces obstacles soient générés dans l'arène de façon aléatoire par le code de mission dans le simulateur ARGoS.

2. Implémenter la logique de parcours des drones dans la simulation à l'aide des capteurs de distance.

Détail du livrable : Ce livrable contient une première implémentation du code de la simulation permettant aux drones d'explorer l'arène de manière indépendante.

3. Faire le lien de communication entre la station au sol (interface Web) et la simulation ARGoS.

Détail du livrable : Ce livrable contient le code du back-end et de la simulation permettant d'établir une communication entre le client (back-end) et le serveur (simulation ARGos). Le code permet à la simulation d'être interactive avec le back-end et donc d'attendre des commandes avant de lancer des états de mission (décollage, exploration, retour à la base, etc.).

4. Implémenter le transfert d'informations de manière bidirectionnelle entre le drone et la station au sol.

Détail du livrable : Au même titre que le point précédent, ce livrable contient du code permettant autant la communication du serveur vers le client que du client vers le serveur.

5. Transformer l'information reçue des drones en une carte d'exploration.

Détail du livrable : Ce livrable contient le code front-end permettant en temps réel de convertir les données entreposées dans le back-end en une carte d'exploration visible par l'utilisateur.

Nous devons également mettre à jour la réponse à l'appel d'offres et remettre une seconde version plus étoffée du premier rapport remis à la PDR.

Ce livrable est dû le 8 mars 2021.

1.3.3 - RR

Le dernier livrable est la RR (Readiness Review). Pour celle-ci, il est question de fournir tous les logiciels requis au bon fonctionnement du système global. Alors, il faut que la plateforme puisse effectuer toutes les commandes nécessaires, et ce, sur les drones simulés et réels. Le code embarqué et simulé des drones doit aussi respecter tous les requis spécifiés. Une vidéo du bon fonctionnement du système doit être remise en même temps que les logiciels. Ce livrable est composé des sous-tâches ci-dessous:

1. Finalisation de la carte de la région explorée fournie à l'aide de l'information récoltée des drones.

Détail du livrable : Ce livrable doit contenir une version finalisée du code permettant la visualisation des informations recueillies par les drones autant au niveau de la simulation que par les drones réels.

2. Implémentation de l'algorithme de parcours des drones afin de maximiser la surface parcourue entre les deux drones. Cet algorithme doit être testé auparavant dans la simulation ARGos.

Détail du livrable : Ce livrable devra inclure une version améliorée et optimisée de l'algorithme d'exploration présenté à la CDR. Le code remis sera en mesure d'effectuer l'exploration complète d'une carte générée aléatoirement à l'aide d'un essaim de drones. Cet algorithme sera à la fois fonctionnel dans la simulation ARGos et sur les drones réels.

3. Implémentation des commandes *Land/End mission* ainsi que *Software update*.

Détail du livrable : Pour ce livrable nous devons faire l'implémentation du code permettant à l'utilisateur du front-end d'envoyer les commandes d'atterrissage et de fin de mission aux drones réels et simulés. Ce livrable devra également inclure une implémentation de la mise à jour du logiciel pour les drones réels.

Ce livrable est dû le 12 avril 2021.

2. Organisation du projet

2.1 Structure d'organisation

Notre équipe est formée de 6 membres. Nous avons désigné notre coordinateur comme étant Jacob Brisson, puisqu'il est en contact avec les différents sous-groupes. Nous avons également décidé de séparer les rôles de l'équipe en deux sous-équipes: l'équipe de simulation et du code qui sera implémenté sur les drones et l'équipe de la plateforme Web qui s'occuperont de la communication entre le poste au sol et les drones réels ou simulés, ainsi que de l'affichage. Chaque sous-groupe est formé de trois membres:

Simulation : Matthew Paoli, Jordan Lecourtois et Philippe Savard.

Les membres de la sous-équipe responsables de la simulation ont des rôles spécifiques. Pour ce qui est de Matthew, sa charge est principalement l'implémentation d'un *socket* du côté de la simulation, ce qui implique qu'il a également contribué à la communication. Ensuite, Jordan s'occupe principalement de la génération d'arènes aléatoires et de l'algorithme d'exploration dans la simulation. Finalement, Philippe est chargé de l'algorithme de retour à la base.

Plateforme Web et drones réels : Laurent Bolduc, Simon Tran et Jacob Brisson.

Les membres de la sous-équipe s'occupant de la plateforme Web ont également des rôles spécifiques. Du côté de Laurent, il s'occupe principalement du *back-end* de la plateforme, et donc touche à la communication avec la simulation et le *front-end*. Simon est principalement chargé du *front-end*, de l'apparence et de la communication. Jacob s'occupe de la communication du *back-end* à la simulation et aux drones ainsi que du code embarqué sur les drones réels.

Chaque semaine, lors de la période de travail du lundi, chaque sous-groupe présente ses avancées et explique le fonctionnement de celles-ci, gardant ainsi tous les membres de l'équipe informés de la partie de chacun et augmentant la compréhension globale du projet pour les membres. Cette rencontre sert aussi à effectuer un retour sur le sprint précédent et à définir les tâches de chacun dans le nouveau sprint. L'affectation de tâches et l'organisation en générale sont faites sur Gitlab. En effet, nous utilisons les ressources des «milestones» afin d'identifier les principaux objectifs du projet, présentés lors de la section 1.3, ainsi que pour nommer les quelques objectifs principaux de chaque sprint. Ensuite, pour chaque «milestone», des sous-tâches associées aux membres de l'équipe sont présentées sous la forme de «issues», et celles-ci doivent être mises dans la section «closed» avant la fin du sprint et donc de la semaine. Alors, pour chaque semaine, chaque membre de l'équipe a une ou plusieurs sous-tâches à effectuer, ce qui permet de séparer de manière efficace les différents objectifs.

Pour les différents rôles, nous avons opté pour une organisation décentralisée, ce qui encourage l'autogestion et la coopération entre chacun des membres de l'équipe. Le coordonnateur a donc pour objectif de s'assurer de la distribution et de la complétion de chacune des tâches et sert de soutien aux autres membres, plutôt qu'à la gestion de l'équipe à proprement dit.

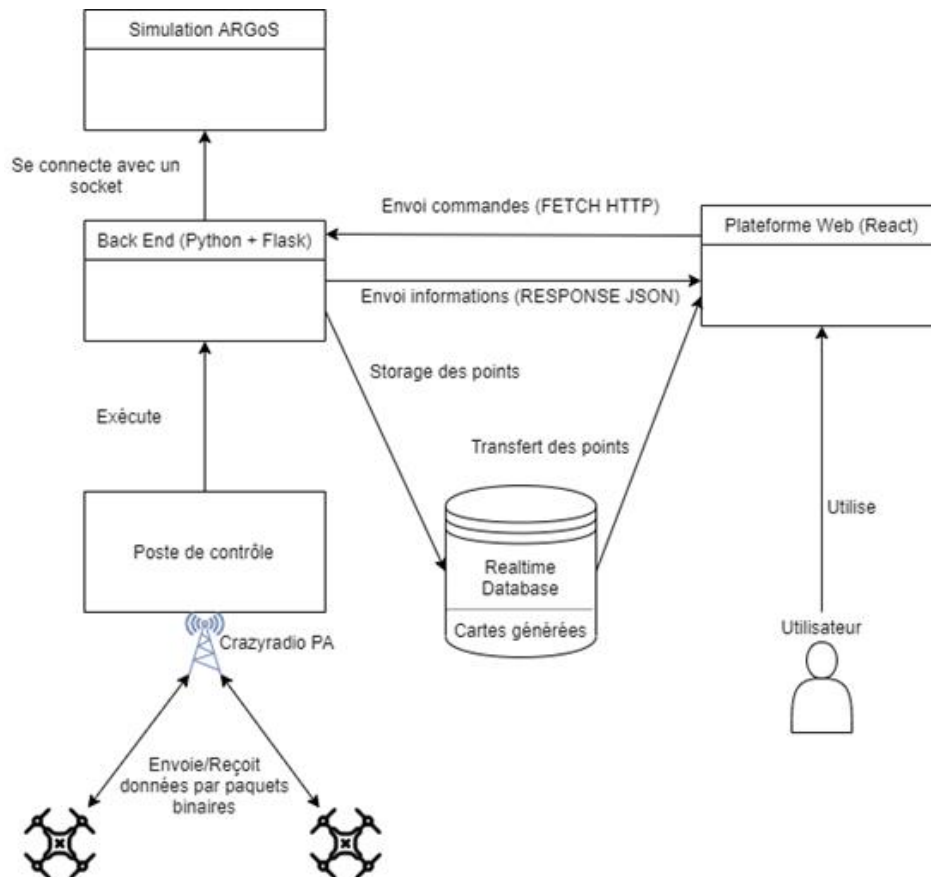
2.2 Entente contractuelle

Le type d'entente contractuelle proposée pour ce projet est le contrat livraison clé en main - prix ferme. En effet, nous avons rejeté l'option du contrat à terme - temps plus frais étant donné que l'utilité de ce type de contrat est de demander l'expertise d'un contracteur, afin d'obtenir ses conseils, sur une certaine période de temps et non sur la totalité du projet. Or, nous avons été choisis afin de fournir un produit final à une date précise. Ainsi, ce type de contrat n'est pas envisageable. En ce qui a trait au contrat de type partage des économies, ces derniers ne s'appliquent pas non plus. Effectivement, les artéfacts produits lors de ce présent projet seront utilisés afin d'étudier les bénéfices d'un système exploratoire utilisant des drones. Il n'est pas question ici de créer un produit qui sera par la suite vendu et profitable sur une longue période de temps, ce qui rend ce type de contrat inutile. De ce fait, nous avons choisi le contrat à prix ferme, ce qui correspond directement aux demandes de ce projet. Le produit final à livrer est clair et défini, les différents livrables sont définis et ont une date de remise fixe. De plus, il est possible d'estimer les coûts finaux pour la globalité du projet, tel que présenté dans la section 4.1, ce qui prouve que ce contrat est le plus adapté à la situation.

3. Solution proposée

3.1 Architecture logicielle générale

Figure 1 - Schéma de l'architecture générale des logiciels à produire



D'abord, nous avons choisi d'implémenter la plateforme Web avec l'architecture «React» et en Typescript. Nous avons fait ce choix puisqu'il est simple d'utiliser les «hooks» de «React» afin de faire une application dynamique en utilisant des composants fonctionnels, comme il sera détaillé dans l'architecture de la station au sol. Nous avons décidé d'utiliser Typescript, puisque ce langage a déjà été utilisé par les membres de l'équipe lors d'un précédent projet.

Le back-end sera en Python. Nous avons décidé d'utiliser la librairie «Flask» avec notre back-end, afin de pouvoir fournir un API à notre plateforme Web et de pouvoir envoyer des commandes de manière réactive au back-end[1]. L'objectif d'utiliser le langage Python est de pouvoir utiliser des librairies pour la communication avec les drones.

Le poste de contrôle permet donc d'exécuter le back-end et la plateforme Web et la communication entre ces deux applications se fait à travers des appels «FETCH» http de la part de la plateforme Web lorsque l'utilisateur appuie sur un bouton ou à chaque

seconde afin de demander les informations des drones. Ces appels activent des fonctions du back-end et celles-ci retournent des réponses JSON, afin de pouvoir transmettre plusieurs données à la fois de manière structurée. De plus, un code d'erreur est retourné en cas de défaillance de la fonction appelée.

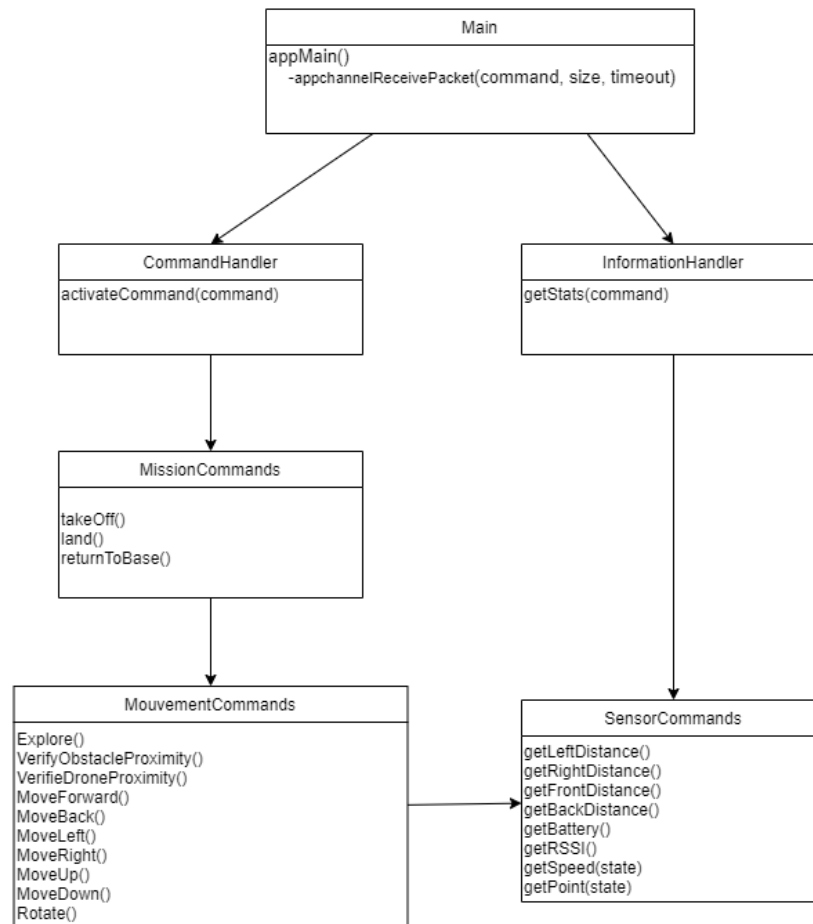
Pour la communication avec les drones réels, le back-end utilise la classe «AppChannel», qui utilise la librairie «cflib» de python pour définir les méthodes de communication avec les drones[2]. Le protocole de transfert des données est le suivant: un caractère, sur un octet, est envoyé au drone et selon le caractère envoyé, le drone va réagir de la manière voulue. Le drone renvoie alors une structure comportant un caractère, qui indique le type d'information retournée, et la valeur de l'information, que ce soit la vitesse, le niveau de batterie, etc. Ce protocole est possible grâce à la «Bitcraze Crazyradio PA», qui est connectée par porte-USB au poste de contrôle[3].

Pour le côté simulation, un socket est utilisé pour connecter le back-end et la simulation. Pour ce faire, on utilise les librairies fournies de «socket» en Python[4] et en C++[5]. L'utilisateur pourra choisir au lancement de l'application Web s'il désire utiliser les drones réels ou ceux simulés. Cela changera un paramètre booléen dans le back-end qui modifiera le comportement des fonctions, mais les fonctions appelées par l'application Web restent les mêmes.

Enfin, l'utilisation d'une base de données est nécessaire afin de pouvoir enregistrer les points reçus par les drones. Par conséquent, nous avons décidé d'utiliser le système de base de données «Realtime Database» de Firebase. Cela nous permettra d'envoyer les points des drones et de pouvoir les récupérer en temps réel dans la plateforme Web, pour les afficher en direct dans la carte. De plus, nous pourrions sauvegarder les cartes générées par les drones.

3.2 Architecture logicielle embarquée

Figure 2 - Schéma de l'architecture logicielle embarqué à produire



D'abord, la routine principale des drones est définie le fichier «Main.c», qui s'occupe d'écouter, avec la fonction «appchannelReceivePacket()», les paquets provenant du back-end, et ce, dans une boucle infinie. Ensuite, selon le caractère contenu dans le paquet reçu par le drone, le drone appellera la fonction «getStats()», s'il doit retourner une information, ou bien «activateCommand()», s'il doit effectuer une commande indiquée. Si le caractère est 'b' (batterie), 'v' (vitesse), 's' (état), ou bien 'p' (point), le «InformationHandler.c» s'occupera de retrouver la bonne information en appelant une fonction de «SensorCommands.c». Ce fichier fournit les fonctions de bases des capteurs du crazyflie-firmware[6] encapsulées de manière à les rendre faciles d'utilisation.

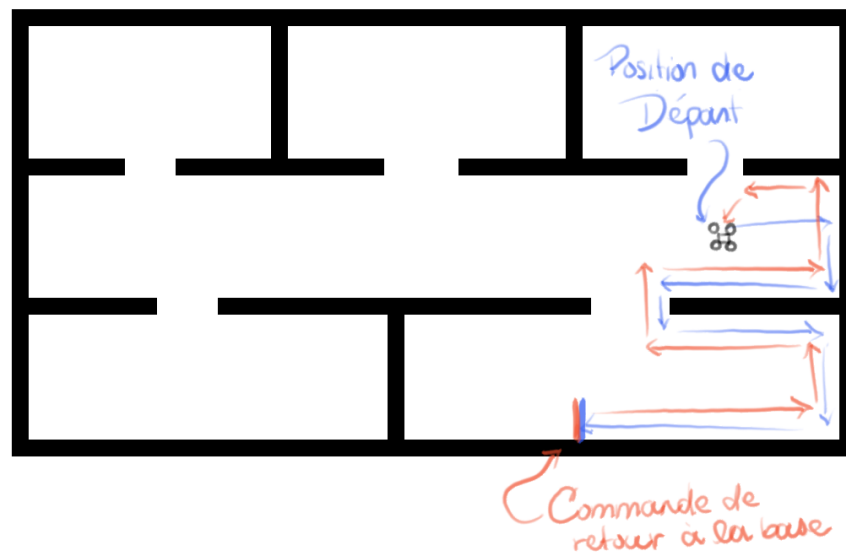
En revanche, si le caractère est 't' (Take Off), 'l' (Land) ou 'r' (Return to base), ce sera le «MissionCommands.c» qui s'occupera d'appeler la bonne fonction. Ce fichier contient les

différentes étapes de la mission, soit: takeOff, returnToBase et Land. Ce fichier va utiliser les fonctions de «MovementCommands.c» afin d'implémenter les différents stages de la mission. Le fichier de mouvement va lui-même utiliser «SensorsCommands.c» et va permettre d'encapsuler les fonctions de mouvements du crazyflie-firmware[6] afin de pouvoir les utiliser de manière similaire à dans la simulation. Alors, le code testé dans la simulation pourra être implémenté dans le fichier «MovementCommands.c».

Quant à l'algorithme d'exploration, la logique implémentée est que les drones suivent les murs à leur gauche ou à leur droite, dépendamment de leur initialisation. Pour faire une analogie, l'algorithme implémenté est donc l'équivalent de faire le tour de l'arène tout en conservant en tout temps notre main gauche (ou droite) en contact avec le mur. Cette stratégie nous assure que, peu importe la quantité de drones, ceux-ci finiront nécessairement par explorer tous les murs d'un ensemble de pièces de façon à cartographier l'arène en entier. Également, afin d'augmenter la rapidité d'exploration, certains drones sont programmés pour suivre les murs à leur gauche et d'autres les murs à leur droite, permettant ainsi à plusieurs drones de parcourir la pièce dans des directions opposées.

Pour ce qui est du retour à la base, lorsque les drones reçoivent la commande de retour à la base ou que leur niveau de batterie descend sous la barre des 30%, les drones effectuent un virage de 180 degrés et inversent le sens de leur exploration. Autrement dit, si un drone suivait les murs de gauche, il suit désormais les murs de droite, et vice-versa, jusqu'à ce qu'il arrive suffisamment près de sa base pour y aller directement. La figure suivante représente le parcours que pourrait emprunter un drone suivant les murs de gauche (bleu) et son retour à la base (rouge).

Figure 3 - Schéma de l'arène de simulation



3.3 Architecture logicielle station au sol

Tout d'abord, nous avons décidé d'utiliser une architecture de trois niveaux. En effet, notre système est séparé en trois modules distincts : le front-end, le back-end et la base de données *Firebase*.

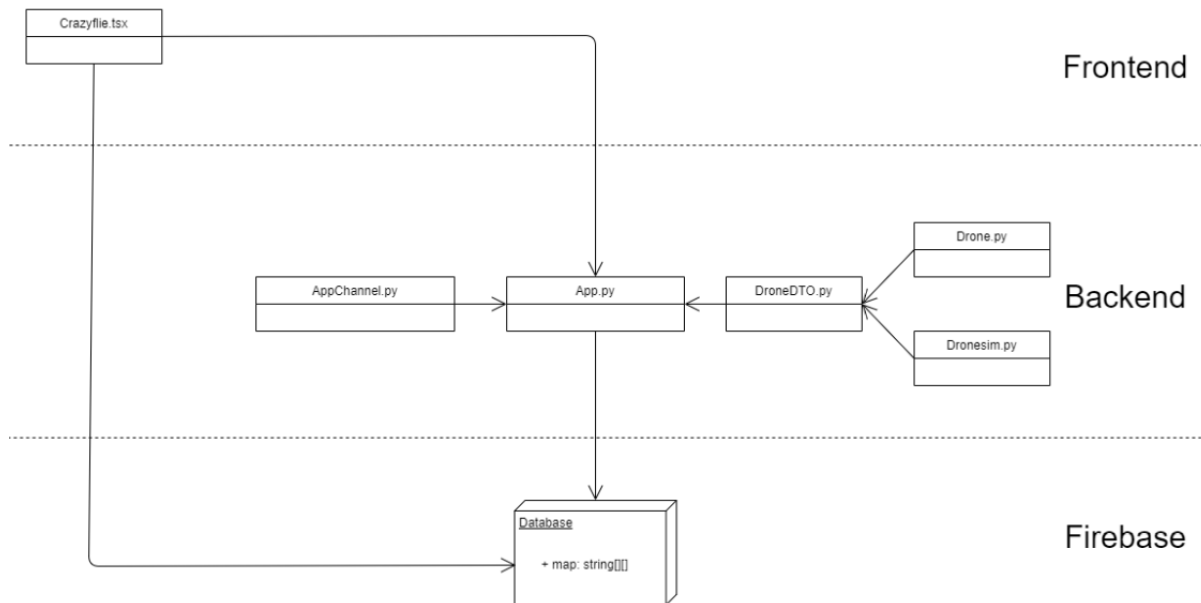
Pour ce qui est du front-end, ce module est la partie visuelle de l'interface utilisateur. Il est codé en Typescript à l'aide du cadre logiciel React. Il utilise fortement les contextes de React afin de partager l'information entre les composants. Par exemple, le *CFCContext* contient certains état de l'application, comme la liste des drones, ainsi que les fonctions qui permettent de communiquer avec le back-end.

Dans l'interface utilisateur, on affiche un champ textuel qui permet à l'utilisateur d'entrer l'adresse IP du back-end si celui-ci n'est pas local. Ainsi, il est possible d'utiliser notre interface sur un appareil mobile et de communiquer avec le serveur.

La carte est affichée à l'aide de figures SVG. Des cercles bleus représentent les drones, des carrés rouges les murs. De plus, l'arrière-plan de la carte est blanc là où les capteurs des drones n'ont rien détecté, et gris lorsque l'information est inconnue. Ainsi, il est possible de découvrir le "brouillard de guerre" peu à peu tout au long de l'exploration. Les informations de la carte sont sauvegardées dans une base de données en temps réel *Firebase* afin de pouvoir persister l'information.

Les requêtes vers le back-end sont encapsulées dans des méthodes statiques de la classe *back-endREST*. Ces méthodes effectuent des requêtes REST aux back-end. En premier lieu, une méthode *liveCheck* est appelée dès le premier rendu de l'application, puis chaque fois que l'url du back-end est modifié par l'utilisateur. Cette méthode a pour simple but de retourner un code OK lorsque la connexion au back-end est bien établie. Les autres méthodes ne peuvent pas être appelées si la connexion est invalide. Il existe une méthode *scan*, permettant d'effectuer un balayage pour se connecter aux drones, une commande *updateStats* pour obtenir les informations des drones à jour telles que la vitesse, la position ou le niveau de batterie. De plus, il y a les méthodes *takeoff* et *land* pour démarrer et arrêter l'exploration. La méthode *reset* permet de réinitialiser les données du back-end. Le back-end se déconnecte alors de tous les drones et supprime l'information qu'il avait en mémoire. Cette méthode possède aussi un argument optionnel permettant de demander au back-end de se connecter à la simulation ou aux drones réels.

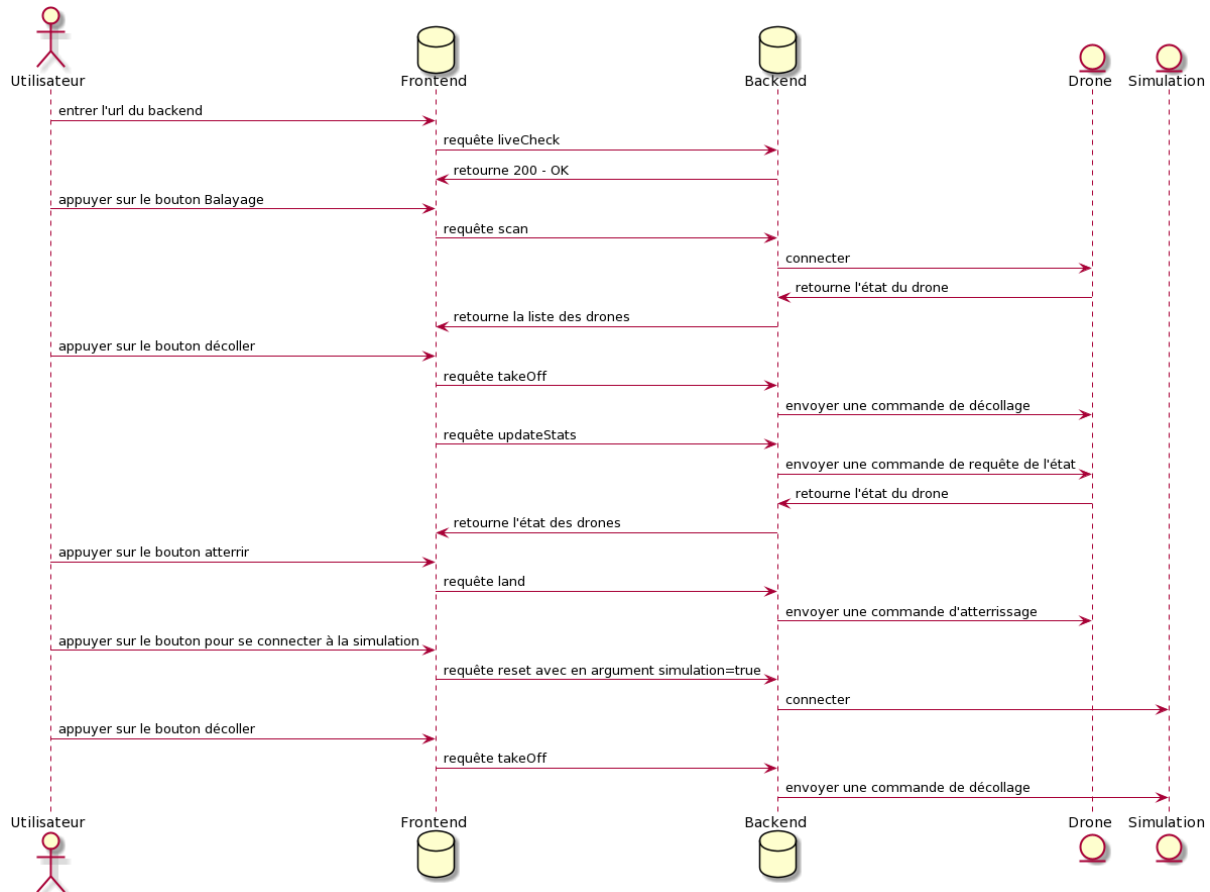
Figure 4 - Schéma de l'architecture logicielle de la station au sol



Pour ce qui est du back-end, ce module est la partie qui communique avec le drone et est exécuté en *Python* avec la librairie Flask. Selon nos prévisions, ce module devrait contenir trois classes. Premièrement, la classe *AppChannel* qui contient les fonctions de communication avec les drones. Deuxièmement, la classe *App* qui contient l'état des drones et de la communication et qui permet la communication avec le front-end et la base de données. Finalement, la classe *DroneDTO*, qui contient la définition des attributs des drones et les transforme en objet, tout en uniformisant l'envoi au front-end peu importe si l'information provient de la simulation ou du drone. Le back-end est en fait un serveur qui ne fait que répondre aux diverses requêtes REST du front-end. Différents points de terminaison permettent d'effectuer toutes les opérations nécessaires. Par exemple, une requête GET vers '/takeOff' permet de démarrer l'exploration.

La figure suivante présente un exemple de communication typique entre le front-end et le back-end. La situation présentée tient pour acquis que l'utilisateur a activé la fonctionnalité de rafraîchissement automatique. Ainsi, le front-end émet par lui-même périodiquement des requêtes `updateStats` afin de mettre à jour l'état des drones.

Figure 5 - Diagramme de séquence de la communication front-end-back-end



On voit que l'utilisateur commence en utilisant l'application afin de communiquer avec les drones réels. Ensuite, il désire travailler avec la simulation. On voit que le front-end envoie la requête '/reset' avec un argument permettant de spécifier que l'on désire basculer vers la simulation. Le back-end se connecte alors à cette dernière et toutes les commandes de drone suivantes seront effectuées sur la simulation.

4. Processus de gestion

4.1 Estimations des coûts du projet

Pour implémenter notre solution, l'Agence fournit l'ensemble du matériel (drone, capteur et autres) nécessaire au bon fonctionnement de notre application. Ainsi, aucun coût associé au matériel n'est endossé par notre compagnie.

Également, l'ensemble des services web utilisés pour l'interface web et la communication entre le front-end et le back-end sont des services gratuits.

L'ensemble de coûts reliés à la réalisation du mandat est relié au travail des développeurs et du coordonnateur de projet. Nous estimons la charge de travail à 535 heures-personnes pour le développement technique, ainsi que 4 heures-personnes par semaine, pour une durée de 10 semaines, pour la coordination du projet. Ainsi, le montant associé aux ressources humaines est de 75,350.00\$, qui peut être calculé comme suit :

$$\text{Coût} = \text{salaire}_{\text{développeur}} \times \text{heure}_{\text{développeur}} + \text{salaire}_{\text{coordonnateur}} \times \text{heure}_{\text{coordonnateur}}$$

$$\text{Coût} = 130\$/h \times 535h + 145\$/h \times 40h$$

Encore une fois, comme mentionné plus haut, Jacob est le coordonnateur.

4.2 Planification des tâches

Le tableau ci-dessous présente les tâches importantes pour chacun des jalons, en plus d'estimer le temps nécessaire pour leur réalisation pour chacune des deux équipes présentées à la section 2.1.

Jalon	Tâche	Développeur Simulation (h)	Développeur Web (h)	Total (h)
Preliminary Design Review (PDR)	Déployer la simulation Argos sur un container Docker	20	0	20
	Implémenter la logique d'évitement d'obstacle dans la simulation	30	0	30

	Simulation d'un parcours avec deux drones	10	0	10
	Concevoir l'interface Web	0	10	10
	Concevoir le back-end Flask	0	10	10
	Établir la communication entre le front-end et le back-end	0	10	10
	Établir la communication entre le back-end et les drones, puis récupérer le niveau de batterie et allumer les DEL.	0	15	15
	Total	60	45	105
Critical Design Review (CDR)	Généré une arène de simulation de façon aléatoire	10	0	10
	Implémenter la logique d'exploration d'une pièce dans la simulation	40	0	40
	Simulation d'une exploration avec 4 drones	10	0	10
	Interfacer la simulation avec le serveur Web	30	0	30
	Concevoir un API qui permet de communiquer de façon identique entre les drones et la simulation	30	30	60

	Recevoir et traiter les données des capteurs des drones	0	20	20
	Total	120	50	170
Readiness Review (RR)	Concevoir le système qui permet de lancer la simulation avec une seule commande Linux	15	0	15
	Implémentation de la communication entre les drones (P2P de BitCraze[7])	25	5	30
	Concevoir une simulation fonctionnant avec un nombre arbitraire de drones	25	0	25
	Finaliser la cartographie de la pièce par les drones explorateurs	10	20	30
	Implémenter la fonctionnalité de mise à jour du logiciel par envoi de paquet binaire (R.F.4.1)	30	0	30
	Implémenter l'exploration d'une pièce quelconque avec les drones	0	20	20
	Implémenter la fonctionnalité de retour à la base pour les drones réels	15	15	30
	Implémenter la sauvegarde et la visualisation des pièces explorées par plusieurs drones	10	70	80

	Total	130	130	260
Grand Total				535

4.3 Calendrier de projet

Semaine	Remise(s)	Tâches à terminer
Semaine du 8 février		<ul style="list-style-type: none"> ➤ Simulation fonctionnelle avec deux drones ➤ Interface Web fonctionnelle avec les drones ➤ Réponse d'appel d'offre terminée
Semaine du 15 février	Remise PDR et Réponse d'appel d'offre	<ul style="list-style-type: none"> ➤ Simulation fonctionnelle avec 4 drones et murs générés aléatoirement ➤ Serveur Web connecté à la simulation ARGoS
Semaine du 22 février		<ul style="list-style-type: none"> ➤ Implémenter la fonction <i>Take off</i> ➤ Commandes et interface avec simulation ARGoS

Semaine du 1 mars		<ul style="list-style-type: none"> ➤ Implémenter la fonction <i>Return to Base</i> ➤ Interface avec les informations des drones réels ➤ Prototype de visualisation de la carte générée terminée
Semaine du 8 mars	Remise CDR	
Semaine du 15 mars		<ul style="list-style-type: none"> ➤ Implémenter la fonction <i>Land/End mission</i> et <i>Take Off</i> sur les drones réels ➤ Communication entre les deux drones réels implémentée ➤ Envoyer les points des drones de la simulation pour construire la carte
Semaine du 22 mars		<ul style="list-style-type: none"> ➤ Implémenter la fonction <i>Software Update</i> ➤ Finaliser l'algorithme pour le parcours des drones ➤ Implémenter la fonction <i>Return to Base</i> sur les drones réels ➤ Toutes les commandes fonctionnelles sur les drones réels
Semaine du 29 mars		<ul style="list-style-type: none"> ➤ Finaliser la visualisation de la carte générée fonctionnelle ➤ Implémenter la sauvegarde/visualisation des cartes d'explorations

Semaine du 5 avril		<ul style="list-style-type: none"> ➤ Lancement du système de simulation complet avec docker-compose ➤ Vidéo du fonctionnement du système terminée
Semaine du 12 avril	Remise RR	

4.4 Ressources humaines du projet

Notre équipe est composée de six candidats au poste d'ingénieur ayant des expertises complémentaires, permettant la réalisation du mandat. Les six ressources humaines de l'équipe sont les suivantes, dans aucun ordre particulier :

1. **Simon Tran** possède quelques années d'usage professionnel de React. Son expérience sera donc utile pour la partie front-end. Il est aussi habile avec Git et Docker.
2. **Jordan Lecourtois** possède une certaine expérience avec les applications Flask, en python, qui est le framework utilisé pour le back-end, ainsi qu'avec le service de tunnel permettant la communication entre un serveur web et un appareil local.
3. **Philippe Savard** possède une certaine expérience avec les systèmes embarqués et démontre un intérêt pour le développement de simulateur. De plus, il possède de l'expérience dans l'utilisation de méthodes agiles dans le processus de développement.
4. **Jacob Brisson** possède une bonne compréhension des algorithmes et un intérêt pour l'organisation, ce qui sera utile pour la structure de l'équipe et des sprints, ainsi que pour la division des tâches.
5. **Matthew Paoli** possède de l'expérience avec les systèmes embarqués, notamment tout ce qui concerne le matériel, capteur et autre. De plus, il a de l'expérience avec des simulateurs ce qui sera pratique pour l'utilisation d'ARGoS.
6. **Laurent Bolduc** possède de l'expérience avec React et des protocoles de communication modernes dans un contexte professionnel. De plus, il a un certain intérêt pour l'algorithmique et de l'expérience avec le travail dans un contexte agile.

5. Suivi de projet et contrôle

5.1 Contrôle de la qualité

Lors de la réalisation du prototype, notre équipe s'engage à assurer le bon fonctionnement des biens livrables. Tous les membres de l'équipe effectueront une étape de déverminage entre chaque étape du développement. Lors de la réalisation des différents éléments du projet, une étape de révision par les pairs devra avoir été faite afin de faire une modification du code assurant la fonctionnalité du produit. La révision de code devra être approuvée par au minimum 2 des 6 membres de l'équipe avant d'être intégrée au produit existant. Le code modifié doit impérativement être fonctionnel et ne doit pas détériorer le bon fonctionnement des autres composants du projet. Lors d'une révision, le réviseur s'engage à :

- Comprendre la modification effectuée;
- Faire la lecture complète ou partielle des modifications afin d'assurer que l'appliquant respecte les bonnes pratiques de programmation;
- Donner un contre rendu et des commentaires permettant à l'appliquant d'améliorer les modifications proposées.

5.1.1 - Vérification comportementale et sécurité

Afin d'assurer le bon fonctionnement des drones, notre équipe utilisera le simulateur physique ARGoS 3.0. Celui-ci utilisera un code en tout point équivalent à celui utilisé pour le prototype réel. Tous les comportements du drone réel devront avoir été prouvés fonctionnels sur le simulateur avant d'être testés sur le drone. Aucun essai ne sera effectué sans que la condition préalable ne soit remplie. En tout temps le ou les membres de l'équipe assignés à la réalisation des essais, à la manipulation d'un drone en fonction ou toutes autres circonstances qui impliquent l'utilisation d'un drone en état de voler devront porter de l'équipement de protection oculaire. Advenant que l'un des prototypes soit défectueux, endommagé, brisé ou dépourvu de pièces nécessaires à son fonctionnement, celui-ci ne sera pas utilisé jusqu'à l'obtention des pièces de rechange. Dans le cas échéant, un rapport devra être rédigé afin de faire état de la situation et décrire les circonstances entourant le bris ou la perte de matériel.

5.2 Gestion de risque

Les risques associés à ce projet sont principalement d'ordre technique ou reliés à l'échéancier. Les risques d'ordre technique comprennent tout ce qui a trait au bris de drones, dû aux collisions ou accidents. La probabilité d'un tel risque est non négligeable et un protocole de remplacement des pièces défectueuses est déjà entendu avec l'agence. Les autres risques techniques peuvent être attribués à des menaces accidentelles ou malveillantes.

Parmi les menaces accidentelles, la mauvaise utilisation des logiciels, menant à un dysfonctionnement de la simulation ou de l'application Web, est une menace éventuelle. Cette dernière est mitigée grâce au lancement des simulations avec une seule commande. Une autre menace accidentelle est liée à la différence entre la physique de la simulation et la physique réelle des drones. Il faudra faire de nombreux tests pour éviter les collisions des drones.

En ce qui a trait aux menaces malveillantes, plusieurs attaques sont possibles sur notre système. Premièrement, étant donné que nous utilisons un nom de domaine public et aucune gestion du trafic, notre application est susceptible d'être victime d'une attaque par inondation. Aussi, un attaquant qui intercepterait un paquet envoyé depuis notre interface web pourrait à son tour envoyer des messages au back-end et contrôler les drones. Afin de mitiger ces menaces, un cryptage des paquets et une authentification pourraient être implémentés.

Les risques associés à l'échéancier sont principalement dus à une mauvaise estimation d'une tâche ou à un problème technique long à résoudre. Puisque les dates des livrables sont fixes, un retard peut causer un stress sur l'équipe. Afin de mitiger ces risques, nous réalisons les tâches le plus rapidement possible pour laisser du temps pour les imprévus.

5.3 Tests

Tel que mentionné à la section 5.1., notre équipe s'engage à tester et valider les biens livrables avant la présentation d'un livrable. Ces tests seront séparés en fonction de leur secteur d'utilisation. Ainsi, chaque entrepôt *git* possède son propre banc de test. Avant le fusionnement des branches vers les branches de développement, un pipeline *gitlab* d'intégration continue exécutera les tests. Aucune branche ne sera fusionnée si les tests ne passent pas.

5.3.1 - Tests front-end

Nous utiliserons Jest afin d'effectuer des tests unitaires sur le code du front-end. Afin d'assurer qu'il n'y ait pas de régression visuelle, nous utiliserons les Snapshots de Jest. Ainsi, aucun code ne sera fusionné si le visuel change par erreur.

5.3.2 - Tests back-end

Afin de tester l'application Flask, qui est utilisée pour notre back-end, le cadre de test utilisé sera pytest. Grâce à ce dernier, nous pourrions tester l'écoute et le traitement de chacune des commandes reçus par le front-end, tels que takeoff, land, return to base et software update. Également, nous testerons la bonne communication avec la simulation et les drones.

5.3.3 - Tests Simulation

L'avantage de l'utilisation d'une simulation est la grande modularité de l'environnement. Ainsi, nous pourrions tester un grand nombre de cas et de conditions pour s'assurer que la logique derrière notre exploration est sans faille avant de l'implémenter sur les vrais drones. Ces tests comprennent, entre autres, l'évitement des collisions, avec les objets et avec les autres drones, ainsi qu'une exploration avec un nombre arbitraire de drones. Une simulation permet également de tester les fonctions de cartographie avec précision puisque nous avons accès à la configuration exacte de l'arène.

5.3.4 - Tests Firmware

Avant d'implémentation du firmware sur les drones, les nombreux tests effectués sur la simulation auront déjà confirmé l'exactitude de la logique d'exploration. Ensuite, les différentes fonctionnalités du drone telles que takeoff, land et return to base pourront être testées, dans cet ordre, pour s'assurer que chacune des fonctions produit le comportement désiré. Ensuite, des tests d'intégration et d'exploration de pièces seront effectués avec un ou les deux drones coordonnés.

5.4 Gestion de configuration

Le gestionnaire de version utilisé est git. Celui-ci sera hébergé sur gitlab. Le git principal du projet sera subdivisé en sous-module. Chacun de ces sous-modules sera en soi un entrepôt indépendant. Cette méthode permettra d'héberger plusieurs entrepôts du projet simultanément.

5.4.1 - Entrepôts

1. INF3995-front-end

L'entrepôt front-end est utilisé pour stocker tout le code utilisé pour l'application web React. Effectivement, elle permet de faire des modifications de l'interface graphique de la station au sol. Les tests unitaires seront placés dans des dossiers «__tests__» et ces fichiers seront nommés sous le format «nom.test.ts».

2. INF3995-back-end

L'entrepôt back-end est utilisé pour stocker le code du serveur qui roule localement sur l'ordinateur où est connectée la Crazyflie Radio[3].

3. INF3995-Crazyflie-Firmware

Contient le code du microprogramme qui contrôle le robot. Il contient un Dockerfile permettant d'obtenir toutes les dépendances afin de pouvoir mettre à jour le code du robot.

4. INF3995-Simulation

L'entrepôt Simulation stocke tout le code nécessaire afin de faire rouler la simulation argos3 et tester nos solutions logiques avant l'implémentation sur le drone. Toutes nos simulations sont exécutées dans un Docker container, alors cet entrepôt contient également un Dockerfile indiquant les téléchargements nécessaires à l'exécution de la simulation.

5. INF3995-Main

C'est l'entrepôt principal qui contient tous les autres en sous-modules git. Il possède un fichier docker-compose permettant de rouler tous les conteneurs simultanément. Ainsi, c'est à partir d'ici qu'il est possible de démarrer l'entièreté de l'application à partir d'une seule commande.

5.4.2 - Documentation

Lors de la rédaction du code, notre équipe s'engage à faire la rédaction de commentaires aidant la lecture et la compréhension des composants. Ces commentaires incluront des descriptions de classes, la description de variables, description du fonctionnement d'algorithmes, des fichiers de description de fonctionnement (p. ex. README), etc. La documentation supplémentaire est entreposée sur le *git* du projet et est accessible par tous les membres de l'équipe.

6. Références

[1] Flask. (2021). *Quickstart*. <https://flask.palletsprojects.com/en/1.1.x/quickstart/>

[2] Bitcraze. (2021). *Crazyflie-lib-python*. <https://github.com/bitcraze/crazyflie-lib-python/tree/master/cflib>

[3] Bitcraze. (2021). *Crazyradio PA*. <https://www.bitcraze.io/products/crazyradio-pa/>

[4] Python. (2021). *socket — Low-level networking interface*. <https://docs.python.org/3/library/socket.html>

[5] GeeksforGeeks. (2019). *Socket Programming in C/C++*. <https://www.geeksforgeeks.org/socket-programming-cc/>

[6] Bitcraze. (2021). *Crazyflie-firmware*. <https://github.com/bitcraze/crazyflie-firmware/tree/1a4cc9acf3edefb3833b72e05095be9c8b0fc887>

[7] Bitcraze. (2020). *Peer to Peer API*. https://www.bitcraze.io/documentation/repository/crazyflie-firmware/2020.02/p2p_api/

[8] ARGos. (2020). *Simulation argos*. <https://www.argos-sim.info/>