



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

Département de génie informatique et génie logiciel

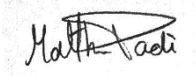


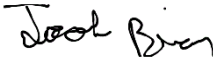
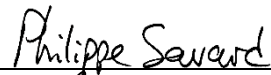

INF3995

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	

15 février 2021

Table des matières

1. Vue d'ensemble du projet	2
1.1 But du projet, porté et objectifs	2
1.2 Hypothèse et contraintes	3
1.3 Biens livrables du projet	4
2. Organisation du projet	5
2.1 Structure d'organisation	5
2.2 Entente contractuelle	6
3. Solution proposée	7
3.1 Architecture logicielle générale	7
3.2 Architecture logicielle embarquée	8
3.3 Architecture logicielle station au sol	9
4. Processus de gestion	11
4.1 Estimations des coûts du projet	11
4.2 Planification des tâches	11
4.3 Calendrier de projet	13
4.4 Ressources humaines du projet	14
5. Suivi de projet et contrôle	15
5.1 Contrôle de la qualité	15
5.2 Gestion de risque	15
5.3 Tests	16
5.4 Gestion de configuration	17

1. Vue d'ensemble du projet

1.1 But du projet, porté et objectifs

Le but du projet est de concevoir un groupe de logiciels qui permettront, à travers une interface opérateur Web, d'envoyer des commandes à un essaim d'un nombre arbitraire de drones afin que ceux-ci explorent et cartographient une pièce de 100 m² maximum de manière autonome. Il est donc question de pouvoir se connecter à cette interface Web sur n'importe quelle plateforme pour ensuite envoyer des commandes simples aux drones et de pouvoir visualiser en direct la cartographie de la pièce ainsi que des informations générales sur les drones et sur leurs déplacements. Il est aussi demandé d'utiliser le logiciel de simulation ARGoS afin de tester le code des drones dans un environnement de simulation avant de l'installer sur les drones réels. Il est donc possible de séparer le projet en trois parties:

La station au sol: un ordinateur qui possède l'interface Web et qui envoie les commandes aux drones à l'aide de la Bitcraze Crazyradio PA qui est connectée sur un port USB de celle-ci.

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

La partie simulation: le logiciel de test qui sera implémenté dans ARGoS, afin de tester et d'observer le comportement des drones avant d'adapter le code pour la partie embarquée.

Pour ce qui en est de la station au sol et de l'interface, les requis globaux sont les suivants:

- Les commandes permises sur l'interface seront 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 par les drones 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.
- Toutes les communications avec les drones seront faites à travers des envois de paquets grâce à la Bitcraze Crazyradio PA.

Pour la partie embarquée :

- Un nombre arbitraire, mais d'au moins deux drones devront parcourir une pièce de 100 m² maximum en évitant les obstacles et en se communiquant avec l'API P2P de Bitcraze.
- 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%.

Pour la partie de la simulation:

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

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 présentes. En effet, nous faisons l'hypothèse que la connexion entre notre application Back End et notre plateforme sera assez rapide afin de pouvoir transférer les informations voulues à une fréquence minimale de 1 Hz. En plus, nous pensons qu'il est possible d'utiliser un Back End en Python afin de contrôler les drones réels ainsi que ceux simulés dans ARGoS, de plus que de communiquer avec la plateforme Web (les détails de notre architecture seront expliqués dans la section 3.1). Nous posons aussi l'hypothèse qu'il sera possible d'utiliser une librairie commune de fonctions de base pour les drones réels et pour ceux simulés dans ARGoS, afin de réutiliser le code créé pour la simulation. Nous pensons aussi qu'il sera possible de réutiliser des fonctions fournies dans l'API Python de Bitcraze afin de communiquer avec le drone à travers la Bitcraze Crazyradio PA.

Pour les contraintes, il est évident que le code embarqué sur les drones pourra être testé pour un nombre arbitraire de drones dans la simulation. Or, dans la réalité, nous sommes contraints à utiliser exactement 2 drones et pas plus. Aussi, étant donné la présente situation de pandémie, il est difficile de se rencontrer afin de travailler en équipe et de tester les drones. Alors, on sera contraint à avoir une personne à la fois qui teste le code embarqué sur les drones et la connexion avec l'interface Web, ce qui va ralentir les progrès du travail. De plus, le temps de travail des membres de l'équipe ne doit pas excéder 630 heures-personnes, sans quoi la solution fournie ne sera pas acceptée. Enfin, il est nécessaire de pouvoir exécuter la simulation ainsi que le reste des logiciels avec

une seule commande « docker-compose », ce qui pose une contrainte technique de devoir travailler avec des conteneurs.

1.3 Biens livrables du projet

1.3.1 - PDR

Le premier livrable est la PDR (*Preliminary Design Review*). Pour cette remise, il sera nécessaire de présenter une simulation de deux drones suivant un parcours quelconque dans le logiciel ARGoS. De plus, il faudra 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.
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.



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 où les murs de l'environnement sont 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.
2. Implémenter la logique de parcours des drones dans la simulation à l'aide des capteurs de distance.

3. Faire le lien de communication entre la station au sol (interface web) et la simulation ARGos.
4. Implémenter le transfert d'informations de manière bidirectionnelle entre le drone et la station au sol.
5. Transformer l'information reçue des drones en une carte d'exploration.



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. Pour ce livrable, on peut trouver ces sous-tâches:

1. Finalisation de la carte de la région explorée fournie à l'aide de l'information récoltée des drones.
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.
3. Implémentation des commandes de Land/End mission ainsi que Software update.



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 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 sera formé de trois membres:

Simulation : Matthew, Jordan et Philippe.

Plateforme Web : Laurent, Simon et Jacob.

Chaque semaine, lors de la période de travail du lundi, chaque sous-groupe présentera ses avancées et expliquera 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 servira aussi pour effectuer un retour sur le sprint précédent et pour définir les tâches de chacun dans le nouveau sprint. L'affectation de tâches et l'organisation en générale seront faites sur Gitlab. En effet, nous utiliserons 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 seront 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 aura quelques sous-tâches à effectuer, ce qui permettra de séparer de manière efficace les différents objectifs.

Pour les différents rôles, nous avons opté pour une organisation décentralisée, ce qui fait qu'il n'y a pas de personne centrale qui coordonne l'équipe. Or, nous partagerons les informations entre tous les membres tout au long du projet.

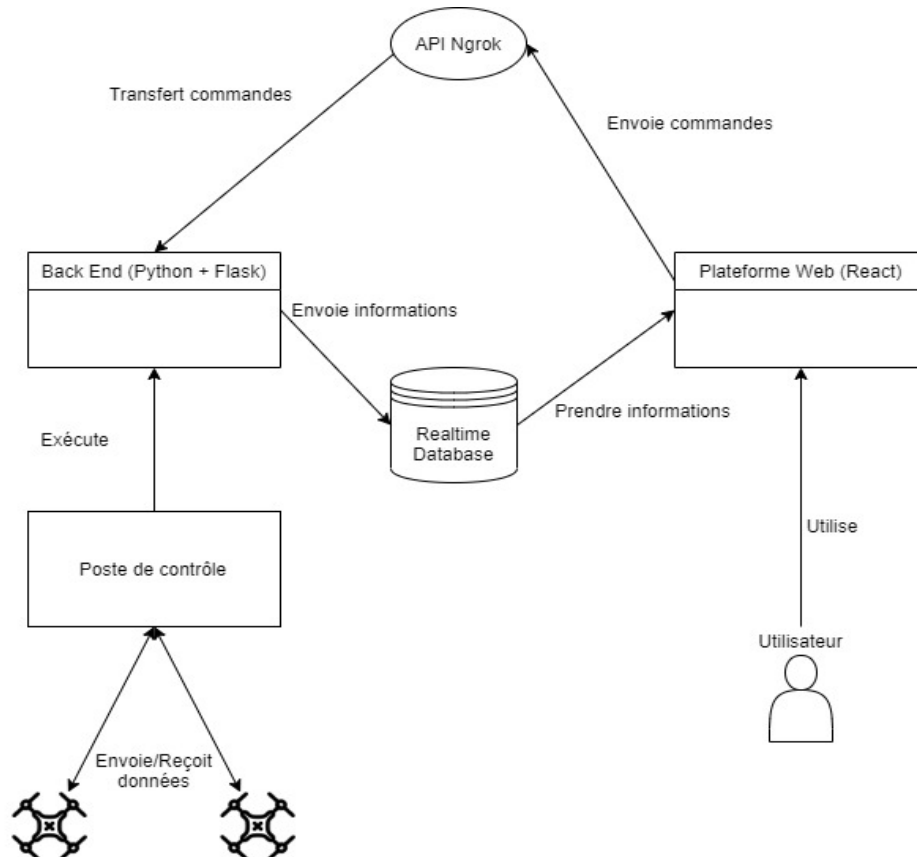
2.2 Entente contractuelle

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. La raison est 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. Alors, il n'est pas acceptable de prendre ce type de contrat. Pour ce qui en est du contrat de partage des économies, ici, celui-ci ne s'applique pas non plus. Effectivement, les artefacts 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 fixée. 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. C'est cette interface Web qui sera disponible pour les utilisateurs, et ce, en l'hébergeant sur internet avec Firebase.

Ensuite, pour ce qui en est du poste de contrôle, celui-ci va exécuter le Back End qui 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. Or, lorsque la plateforme est hébergée sur le Web, il est alors impossible de communiquer avec cet API local, ce qui nous a amenés à utiliser Ngrok avec l'application Python utilisant Flask. Ce service permet de créer un tunnel entre

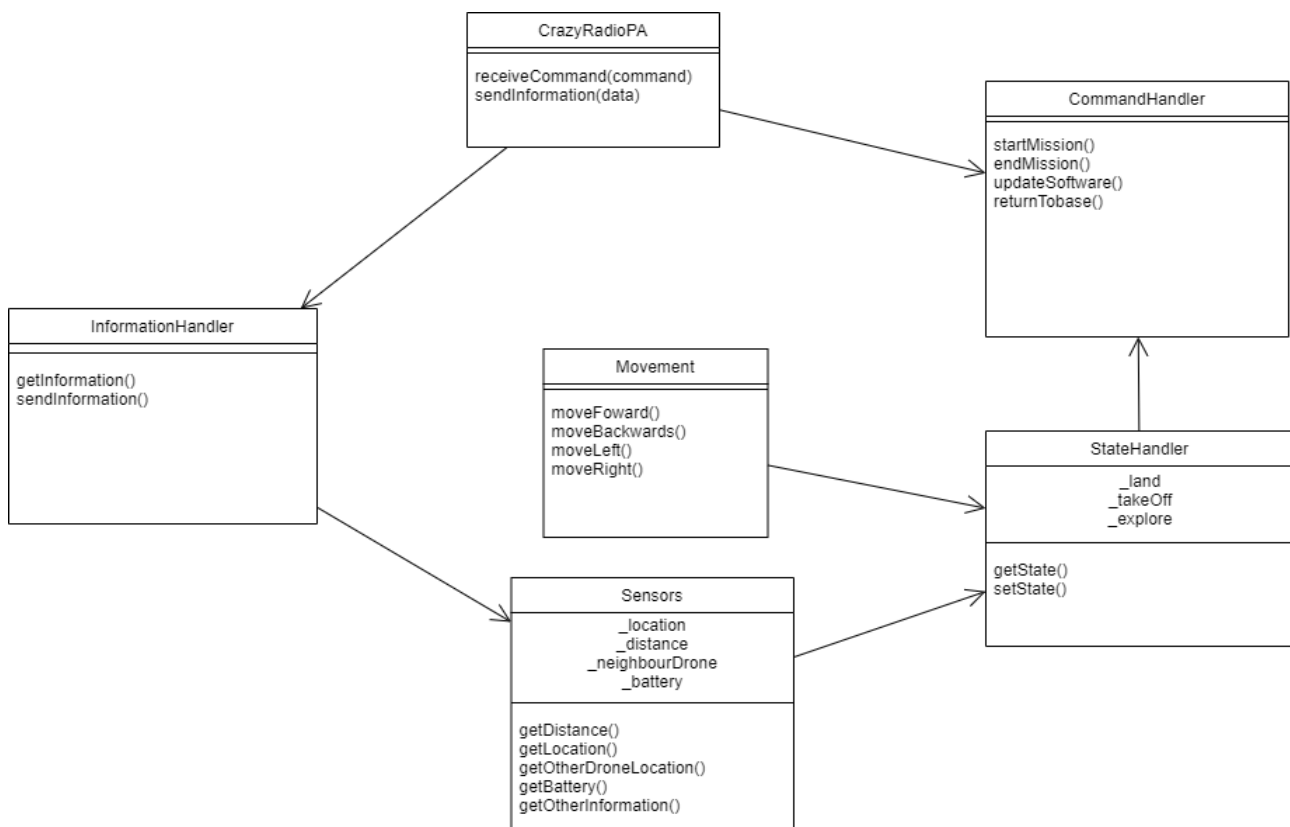
l'adresse publique fournie par Ngrok et de rediriger les commandes vers l'API local Python et donc vers le Back End. De cette manière, toutes les commandes envoyées aux drones de l'interface Web passeront par ce tunnel créé dynamiquement lors du lancement de l'application Python.

L'objectif d'utiliser le langage Python est de pouvoir utiliser l'API de Bitcraze afin de communiquer avec les drones. Le poste de contrôle permet donc d'exécuter cette application Python et d'envoyer et de recevoir des données des drones réels avec la Bitcraze Crazyradio PA connectée par porte-USB au poste. De plus, ce même processus sera utilisé afin d'envoyer ou de recevoir des données des drones simulés dans ARGoS.

Enfin, l'utilisation d'une base de données est nécessaire afin de pouvoir stocker les informations reçues par les drones, surtout en ce qui concerne les données de cartographie. Alors, nous avons décidé d'utiliser le système de base de données « Realtime Database » de Firebase. Cela nous permettra d'envoyer les informations des drones et de pouvoir les récupérer en temps réel dans la plateforme Web, pour les afficher en direct.

3.2 Architecture logicielle embarquée

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



Tout d'abord, nous avons décidé de séparer la structure du logiciel embarqué en plusieurs *Handlers*. Ces *Handlers* ont la responsabilité de gérer les commandes de haut niveau. Premièrement, la classe *CrazyRadioPA* a pour but de communiquer entre la station au sol et les drones. Elle a donc la responsabilité de prendre la commande envoyée par la station et la propager au *CommandHandler*. De plus, toute l'information que les drones acquièrent pendant l'exploration doit aussi passer par *CrazyRadioPA* afin d'être acheminée vers la station au sol et éventuellement à la base de données. C'est pour cette raison que ces deux fonctions sont : la réception de commande et l'envoi d'informations.

Dans le cas de la réception de commande, on va faire appel à *CommandHandler* et lui passer la commande à effectuer. Par la suite, cette classe est capable de changer l'état du robot dépendamment de la commande qu'il a reçue à l'aide de *StateHandler*. Comme on peut le voir sur le schéma, la classe *StateHandler* utilise la classe *Movement* et *Sensors*. Dépendamment de l'état du drone, on peut faire appel à des fonctions de mouvements et aussi à l'information fournie par les capteurs afin d'accomplir la tâche.

Toutefois, dans le cas où on veut envoyer de l'information, *CrazyRadioPA* va utiliser la classe *InformationHandler*. Le but de cette classe est de gérer l'information que l'on reçoit et que l'on envoie. La classe utilise aussi les fonctions de la classe *Sensors* afin d'obtenir les informations nécessaires à l'envoi afin de fournir une cartographie de la région à explorer.

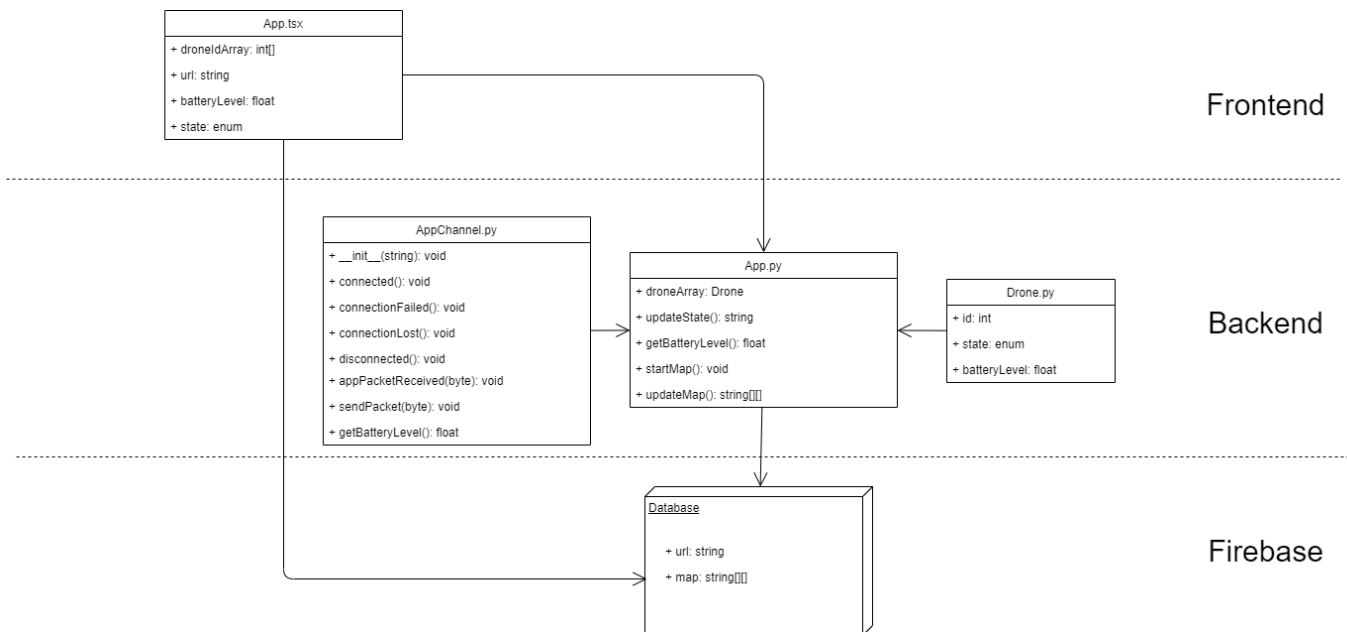
La raison pour laquelle on utilise des *Handlers* est parce qu'il est plus facile de gérer le cheminement de l'information. Plutôt que d'appeler plusieurs fonctions de plusieurs classes lors de la réception d'une commande on fait juste acheminer cette commande au *handler* et par la suite il va faire les appels appropriés. Vu que les commandes qui peuvent être appelées par la station au sol sont *Start mission*, *End mission*, *Software update* et *return to base*, les fonctions qui peuvent être appelées dans le *CommandHandler* son identique. De plus, la décision d'ajouter *InformationHandler* est pour faciliter l'accès à l'information. Sans cette classe, pour obtenir de l'information, il aurait fallu passer par le *CommandHandler*, ensuite le *StateHandler* et finalement au *Sensors* ce qui est illogique et d'une trop grande complexité pour rien.

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 Frontend, le Backend et la base de données *Firebase*.

Pour ce qui est du Frontend, ce module est la partie visuelle de l'interface utilisateur, est compilé en *React*, et est hébergé sur un site *Firebase*. Dans les plans présents, il devrait contenir une classe principale nommée *App*. Cette classe contient tous les éléments visuels html ainsi que des variables qu'elle devra obtenir de plusieurs sources. De plus, cette classe communique avec la base de données afin d'obtenir l'url du tunnel https du site web Backend. Par la suite, le Frontend peut obtenir des informations tel que le niveau de batterie des drones en passant par le Backend.

Figure 3 - Schéma de l'architecture logicielle de la station au sol



Pour ce qui est du Backend, ce module est la partie qui communique avec le drone et est exécuté en *Python*. 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 Frontend et la base de données. Finalement, la classe *Drone*, qui contient la définition des attributs des drones et qui sera utilisée comme objet par la classe *App*. De plus, bien que le Backend soit local afin de pouvoir communiquer avec les drones, nous utilisons la technologie *Ngrok* afin de faire un tunnel https vers un site web. Cela permet au Backend de communiquer avec le Frontend.

Pour ce qui est de la base de données, elle est supportée par la technologie *Realtime Database* de *Firebase*. Elle permet entre autres la communication initiale entre le Frontend et le Backend en stockant le lien du site web *Ngrok*. Elle va également contenir les données qui nécessitent une certaine permanence, telle que la cartographie qui résulte de l'exploration des drones.

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 frontend et le backend 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 455 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 64,950.00\$, qui peut être calculé comme suit :

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

$$\text{Coût} = 130\$/h \times 455h + 145\$/h \times 40h$$

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 et déployer l'interface web sur firebase	0	10	10
	Concevoir le backend Flask	0	10	10

	Établir la communication entre le frontend et le backend	0	10	10
	Établir la communication entre Le backend 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
	Générer et sauvegarder les cartes à partir des informations observées par les drones	0	50	50
	Total	120	100	220
Readiness Review (RR)	Concevoir le système qui permet de lancer la simulation avec une seule commande linux	15	0	15
	Concevoir une simulation fonctionnant avec un nombre arbitraire de drones	25	10	35
	Implémenter l'exploration d'une pièce quelconque avec les drones	0	30	30
	Implémenter la sauvegarde et la visualisation des pièces explorées par plusieurs drones	20	30	50
	Total	60	70	130

Grand Total	455
-------------	-----

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> ➤ Communication entre les deux drones réels implémentée ➤ Toutes les commandes fonctionnelles sur la simulation
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 ➤ Toutes les commandes fonctionnelles sur les drones réels
Semaine du 29 mars		<ul style="list-style-type: none"> ➤ Visualisation de la carte générée fonctionnelle
Semaine du		<ul style="list-style-type: none"> ➤ Lancement du système de simulation complet avec

5 avril		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 frontend. 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 backend, 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 et sont 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. Également, la communication entre notre frontend et backend se fait grâce au service ngrok. Ainsi, un attaquant qui intercepterait un paquet envoyé depuis notre interface web pourrait à son tour envoyer des messages au backend 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* possèdera 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 Frontend

Nous utiliserons Jest afin d'effectuer des tests unitaires sur le code du frontend. 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 Backend

Afin de tester l'application Flask, qui est utilisée pour notre backend, 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 frontend, 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-FrontEnd

L'entrepôt FrontEnd 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-Backend

L'entrepôt Backend est utilisé pour stocker le code du serveur qui roule localement sur l'ordinateur où est connecté le Crazyflie Radio.

3. INF3995-Crazyflie-Firmware

Contiens 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.