

Département de génie informatique et génie logiciel

INF3995

**Projet de conception d’un système informatique**

Documentation du projet répondant à l’appel d’offres

H2024-INF3995 du département GIGL.

***Conception d’un système d’exploration spatiale***

Équipe No 102

*Majeed Abdul Baki*

*Étienne Perron*

*Julien Roux*

*Victor Gilbert*

*Abdel Madjid Sant’anna*

*Étienne Hourdebaigt*

*Février 2024*

# Vue d’ensemble du projet

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

*[Décrire brièvement le but et les objectifs du projet ainsi que des biens livrables attendus.]*

Le projet vise à concevoir un système novateur d'exploration planétaire à travers une flotte de robots. ~~L'objectif principal est de fournir une preuve de concept démontrant que l'exploration, la communication et le contrôle à distance des robots sont réalisables.~~ Le projet possède plusieurs objectifs.

Tout d’abord, l’objectif principal est de fournir une preuve de concept que l’exploration, la communication et le contrôle à distance des robots sont réalisables. Pour cela nous avons pour objectifs de :

1. Planifier et exécuter une simulation de mission d'exploration avec des robots Agilex Limo [8] en tant que Rover
2. Pour cela il faudrait concevoir et implémenter un système informatique novateur en utilisant ROS2 pour la programmation des robots, assurant une exécution optimale de la mission d'observation
3. Garantir la fiabilité et la robustesse du système informatique développé pour la mission d'exploration au travers de l’assurance qualité
4. Un objectif secondaire est de démontrer qu’il est possible de réaliser les requis sur une flotte de robots plutôt qu’un seul.

Les livrables incluront le logiciel embarqué sur les robots ainsi que la station au sol, permettant le contrôle des robots et la récupération de leurs données. En adoptant cette approche, le projet ambitionne d'accélérer la découverte des points d'intérêt planétaire par rapport aux solutions antérieures qui reposaient sur l'utilisation de robots plus imposants et moins agiles.

## Hypothèse et contraintes (Q3.1)

*[Énumérer les hypothèses sur lesquelles repose ce plan ainsi que les contraintes dans le cadre de ce projet. Pas seulement des éléments techniques, mais aussi des éléments externes à l’équipe.]*

Notre plan repose sur un certain nombre d’hypothèses qui sont basées sur les connaissances que nous avons au début du projet :

* Nous supposons que la simulation Gazebo est équivalente à l’utilisation des robots LIMO.
* Nous supposons que notre environnement pour rouler les différents serveurs n’engendrera pas d’erreurs lorsqu’il sera déployé sur un réseau plus large lors des vraies missions.
* Nous formons l’hypothèse que nous pouvons négliger l’effet qu’aura le terrain d’une planète sur la mobilité du robot, puisque nos tests se font sur un terrain très plat dans une salle de test.
* Le système testé sur 2 robots, mais on suppose qu’il va être applicable pour un nombre arbitraire de robots

Un projet comme celui-ci contient bien sûr des contraintes qui affecteront le travail de l’équipe :

* L’équipe a comme contrainte d’implémenter une solution pour les robots LIMO AgileX.
* Nous sommes aussi limités à ne pas dépasser 630 heures de travail en tout, et avons 3 mois environ pour présenter notre produit.
* Les ressources sont aussi limitées puisqu’une seule salle de test et seulement deux robots sont disponibles pendant la phase de développement.

## Biens livrables du projet (Q4.1)

*[Énumérer les artéfacts qui devront être créés durant le projet avec leurs dates prévues de publication.]*

Le premier livrable sera le PDR (*Preliminary Design Review*), dont l’échéance est le 16 février 2024. Nous avons plusieurs étapes à respecter :

* Il faut d’abord répondre à l’appel d’offres, ce qui demande plusieurs documents :
  + La création de documents d’architecture. Un pour l’architecture logicielle générale du projet et trois autres diagrammes spécialisés sur les architectures suivantes : systèmes embarqués, simulation ainsi que pour l’interface utilisateur.
  + Nous avons aussi listé les plus grandes étapes d’avancement dans le développement des applications et nous l’avons mis sous forme d’un diagramme de Gantt afin de présenter les différentes phases.
  + Planification de projet et prévision de coûts.
* Ensuite, il faut aussi livrer un prototype minimal qui comprend plusieurs parties reliées ensemble, soit l’interface de base de la station de sol, 2 robots identifiables par l’interface, et le lancement de l’exploration sur la simulation. Les artéfacts sont les suivants :
  + Une interface graphique qui permet à l’utilisateur de communiquer avec les robots réels ou dans une simulation (frontend Angular).
  + Un programme pour les robots utilisant la librairie ROS (réel et simulation).
  + Un serveur qui servira à faire l’interface entre les robots et l’utilisateur (backend Python avec librairies FastAPI et ROS).
  + Chaque programme roule sur des conteneurs Docker différents

Le deuxième livrable est le CDR (Critical Design Review), à rendre pour le 22 mars 2024. La plupart des artéfacts ou documents seront créés, mais il s’agira de les mettre à jour.

* Nous aurons la documentation:
  + Architecture logicielle ajustée s’il y a lieu
  + Mises à jour sur le déroulement du projet
  + Résultats des tests effectués
* Ainsi que la partie technique avec la 2e itération du prototype
  + Cela inclut de régler des problèmes détectés auparavant et aussi ajouter de nouvelles fonctionnalités afin de remplir plus de requis.
  + Remise de la base de données PostgreSQL, liée au serveur Python.

Pour la remise finale, soit le RR, le but est de terminer tous ces chantiers amorcés au fur et à mesure de la session:

* Rédaction de la documentation finale avec résultats et conclusion
* Développement du prototype final
* Élaboration d’un plan de démonstration afin d’atteindre un niveau de maturité 4 de la solution.
* Et afin de présenter nos résultats, nous préparerons une présentation orale.

# Organisation du projet

## Structure d’organisation (Q6.1)

*[Décrire la structure d’organisation de l’équipe de projet et les différents rôles des membres.]*

Afin d’assurer un bon fonctionnement de notre équipe, nous avons établi une structure qui régira l’organisation de l’équipe. Cette structure comprend plusieurs règles. La première est que les décisions de conceptions architecturales seront prises en consensus lors de rencontre d’équipes. Lors des rencontres d’équipes, un coordonnateur est assigné afin d’assurer un déroulement fluide de la réunion.

Nous avons organisé nos semaines de manière à tenir trois réunions Scrum hebdomadaires, dans le but de maintenir le cap vers nos objectifs tout au long de la session. Pendant ces rencontres un secrétaire est nommé pour garder des traces écrites des nouveaux éléments discutés en réunion. Ce fonctionnement Agile va nous aider à nous entraider et qu’il y ait le moins de bloquants possibles.

Encore pour nous aider à rester organisés, nous avons assigné des rôles chaque membre de l’équipe. Julien Roux assurera le rôle de coordonnateur, ce qui signifie que son rôle principal est d’assurer l’assignation des taches et le respect des différents échéanciers. Le reste de l’équipe, soit : Victor Gilbert, Majeed Abdul Baki, Étienne Perron, Abdel Madjid Sant’anna et Étienne Hourdebaigt qui auront le rôle de développeurs.

## Entente contractuelle (Q11.1)

*[Décrire le type d’entente contractuelle proposée pour projet et les raisons de ce choix]*

Notre équipe propose à l’agence spatiale un contrat à prix ferme pour différentes raisons. D’abord, le niveau de spécification pour le projet est élevé. Chaque livrable est très précis et ne devrait pas évoluer pendant le développement. C’est une caractéristique clé de ce type de contrat.

Aussi, les requis étant établis, le promoteur pourra faire des vérifications de manières régulières par rapport à nos avancements. Ce contrat offre aussi plus de tranquillité d’esprit pour l’agence puisque nous nous engageons à respecter les requis donnés en entier.

# Description de la solution

## Architecture logicielle générale (Q4.5)

*[Un diagramme qui résume l’architecture. Un texte qui décrit et justifie les choix. Mention des requis concernés.]*

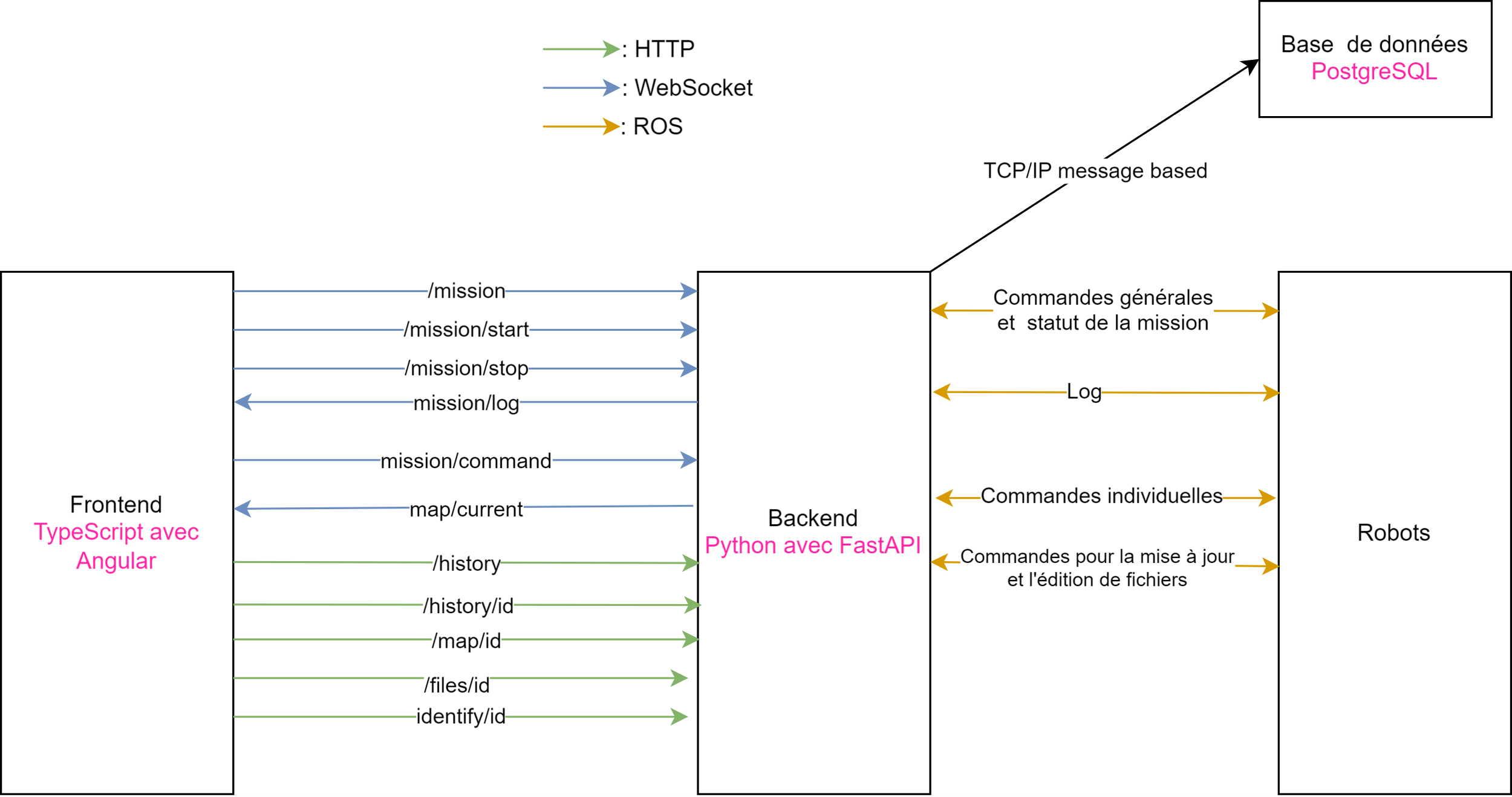


Figure 1. Diagramme d’architecture logicielle général

~~Ce diagramme~~ La figure 1 décrit les différentes routes et protocoles qui seront utilisés afin de communiquer entre les différentes parties du projet. Notre frontend désigne l’interface utilisateur, c’est la partie de notre projet qui permet de lancer et paramétrer les différents programmes de missions. L’application frontend utilise le cadriciel Angular [1] et le langage de programmation TypeScript. Pour accomplir ces tâches, notre frontend communique avec notre backend à travers certaines routes HTTP pour les demandes qui ne sont pas lié à une mission ou que ne nécessite pas d'être initié pas le serveur. Pendant les missions, les communications se font grâce à des connexions *WebSocket* afin de simplifier les demandes et envois des nouvelles informations disponibles. La communication *WebSocket* est implémentée dans la librairie SocketIO [2]. Notre backend emploie le cadriciel FastAPI qui offre un serveur http et s’intègre avec SocketIO [3], et le langage de programmation Python. Il fait le pont entre plusieurs entités qui sont, notre frontend, comme expliquée précédemment, les différents robots, et nos bases de données PostgreSQL. La communication entre le backend et les robots est intégrée à travers ROS qui couvre d’autres protocoles. Les différentes flèches sur le schéma montrent les différents *Topics* et *Services* qui seront implémentés sur ROS. ~~L’~~ Le backend gère aussi les entrées avec les différentes bases de données qui vont garder toutes les données qui doivent persister. Cela inclut entre autres les historiques des missions et les différentes cartes. Cette communication est faite grâce au protocole mis en place par la base de données PostgreSQL, une base de données utilisant le langage SQL [4].

## Station au sol (Q4.5)

*[Quelques blocs des principaux modules ou classes seulement. Des diagrammes sont nécessaires. Un texte qui décrit et justifie les choix.*

*Mention des requis concernés.]*

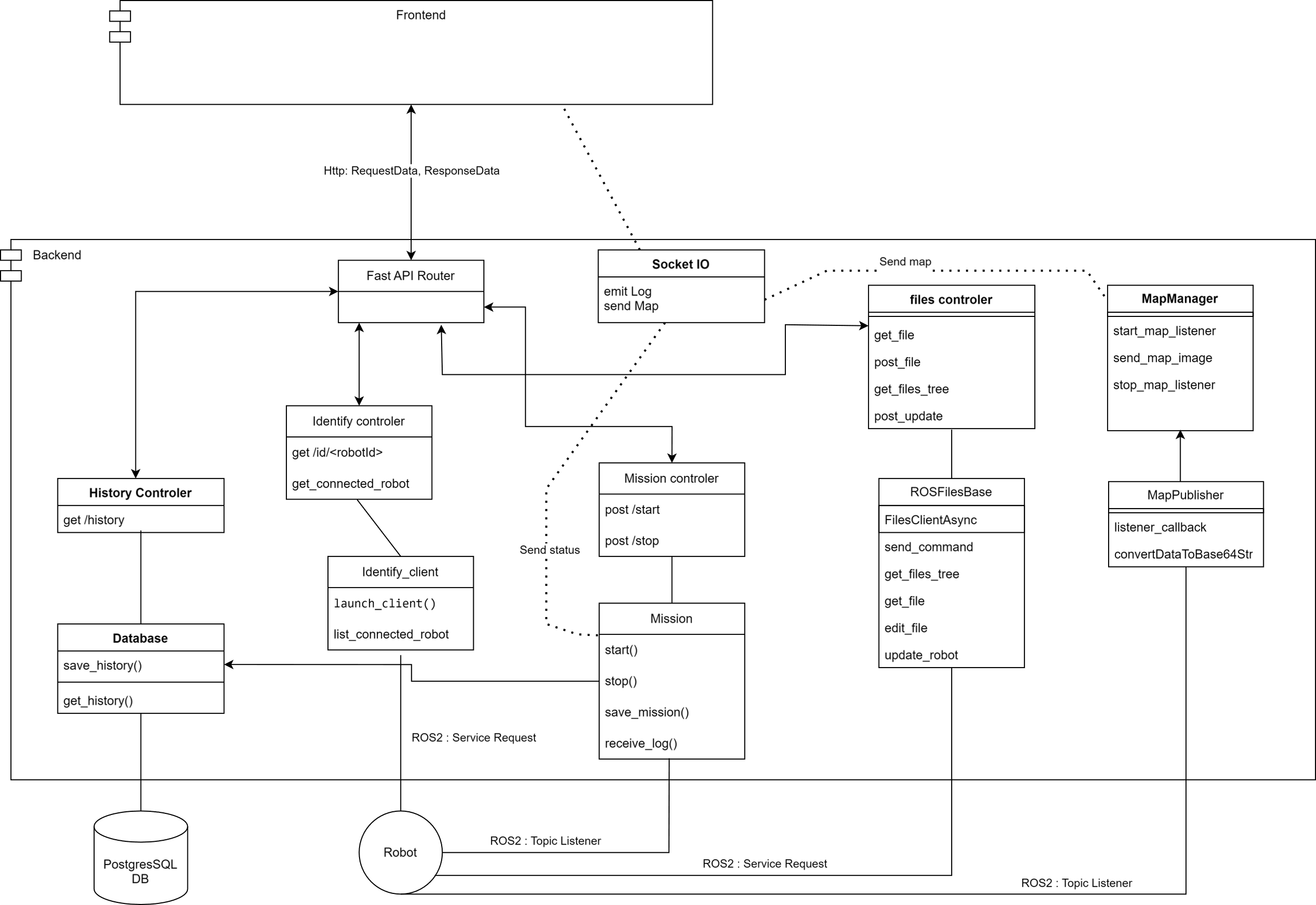


Figure 2. Diagramme d’architecture serveur

* Routeur FastAPI:
  + Gère les requêtes HTTP et les dirige vers les contrôleurs appropriés
* Contrôleur *Identify*:
  + Gère l’identification des robots
* Contrôleur d'historique:
  + Gère l'historique des missions
* Contrôleur de mission:
  + Gère le lancement, l'arrêt et la surveillance des missions
* Base de données:
  + Stocke les données des missions et des utilisateurs

## Requis concernés

* Le système doit être capable de gérer plusieurs utilisateurs (R.F.10) ~~et missions.~~
* Le système doit identifier chaque robot séparément (R.F.1 et R.C.5)
* Le système doit être capable de stocker et de récupérer des données des missions précédentes. (R.F.17)
* Le système doit être capable de surveiller l'exécution des missions. (R.C.1 et R.F.9)

## Logiciel embarqué (Q4.5)

*[Un diagramme qui résume l’architecture. Un texte qui décrit et justifie les choix.*

*Mention des requis concernés.]*

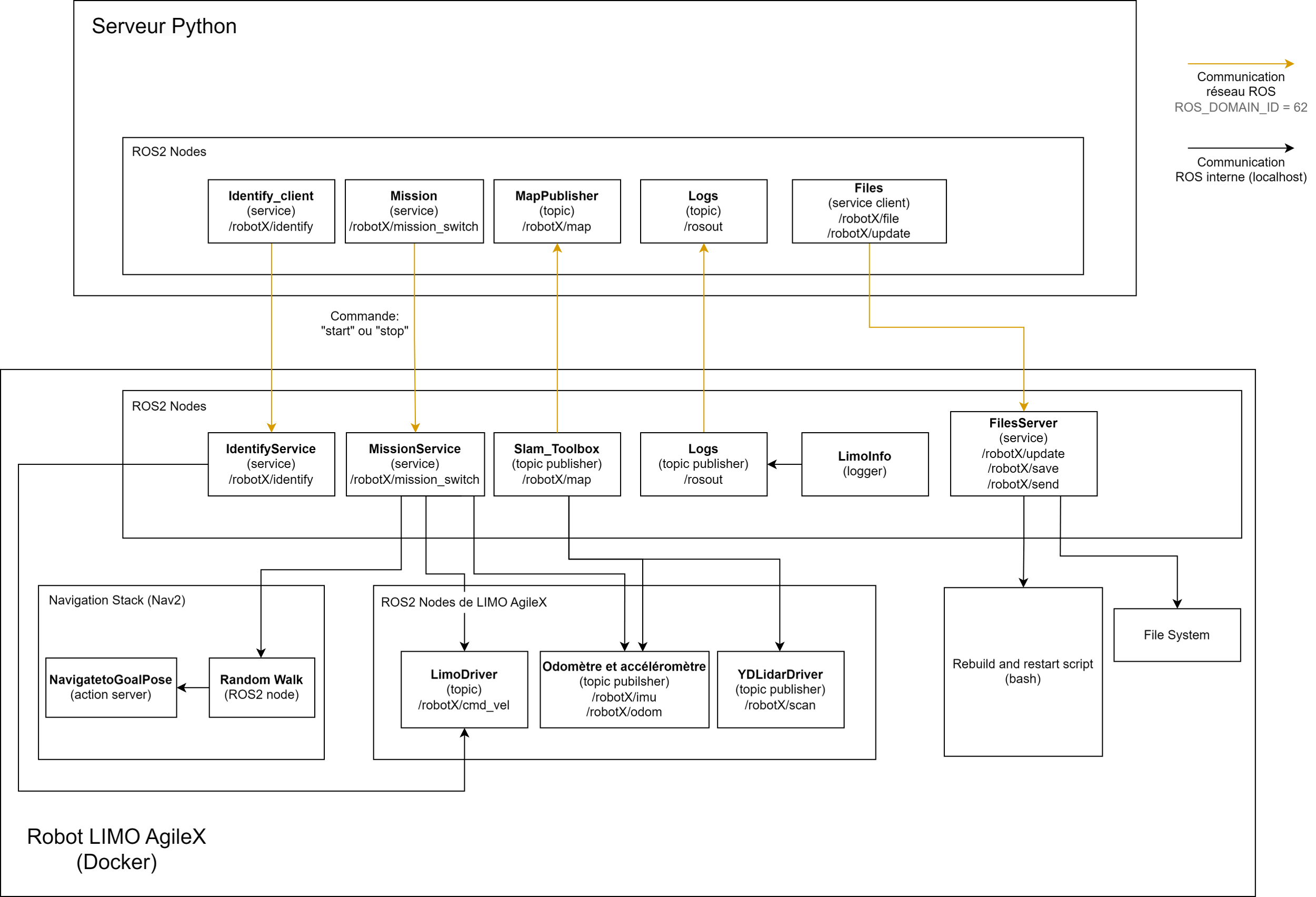


Figure 3. Diagramme du logiciel embarqué

Avant d’entamer la description des différents modules présents sur les robots, il est important de noter que les services et topics offerts par ceux-ci sont tous précédés d’un espace de nom (namespace) qui permet de différencier les différents robots lors d’une communication. Ceci nous offre donc la possibilité de communiquer avec les robots séparément. Les namespaces ont tous la forme /robotX/<…> où le X correspond au numéro de chaque robot (1, 2, etc). Ce numéro identifiant le robot sera conservé sur chaque robot sous la forme d’une variable d’environnement. De plus, nos robots seront dans leur propre identifiant de domaine *(DOMAIN\_ID)* [5] pour éviter tout conflit avec les autres équipes étant donné nos espaces de test partagés. Le robot utilise des capteurs pour réaliser les différentes tâches. Le capteur le plus mobilisé est le lidar qui permet de numériser les alentours. Cette numérisation permet de créer la carte utilisée par le robot et de montrer à l’utilisateur. Il y a aussi l’accéléromètre, gyroscope (qui forme un IMU) qui permet de corriger la position du robot dans le temps. Le robot se sert aussi des données des moteurs qui complètent le calcul de la position ainsi que la vitesse et distance parcourue.

Identify:

* Au moyen d’un service ROS (/robotX/identify), le serveur backend demande au robot de s’identifier. Le robot répond alors en tournant sur lui-même. Cette réponse permet à l’opérateur de savoir quel robot physique correspond au robot numéroté “1”, par exemple. Le nœud Identify permet de répondre au requis R.F.1.

Mission:

* Le serveur backend demande au robot de lancer la mission au moyen d’un service ROS (/robotX/mission\_switch). Le robot démarre alors la mission et retourne une rétroaction pour informer le serveur que tout fonctionne toujours bien, grâce à un objet de type MissionSwitch, une interface que nous avons créée à cet effet [6]. L’avantage d’utiliser un service est que le serveur aura une réponse confirmant la réception, permettant d’annuler la mission à tout moment ce qui est un requis. Le nœud Mission permet de répondre au requis R.F.2.

Map Stream:

* Au moyen d’un topic (/robotX/map), le robot communique les informations qu’il découvre au moyen de ses caméras et LIDAR et qui permettent de modéliser la salle dans laquelle il se trouve. Le serveur backend s’inscrit à ce topic et reçoit les informations en continu, ce qui lui permet de retransmettre ces informations au client pour qu’il affiche le modèle de la salle à l’opérateur. Le nœud Map permet de répondre aux requis R.F.8, R.F.9, R.F.11 et R.F.18.

Robot Logs:

* Les journaux d’événements (logs) sont générés par le système embarqué sur le robot afin de garder un suivi des informations récupérées par les capteurs. Le robot transmet les informations récupérées en les publiant sur un topic (/robotX/logs) auquel le serveur backend se souscrit afin de récupérer les informations pour les traiter. Le nœud Logs permet de répondre au requis R.C.1.

Update codebase:

* Le serveur backend envoie les fichiers modifiés au robot au moyen d’un service. Le backend est le client et le robot est le serveur du point de vue du service (/robotX/update). Lorsque le robot reçoit les nouveaux fichiers, il lance un script *bash* qui fait une sauvegarde de l’état présente du code embarqué, modifie les fichiers, reconstruit l’environnement (*colcon build*) et le relance. S’il y a des erreurs, le script ramène les fichiers à la version précédente et redémarre le système embarqué avec cette version qui fonctionne. Les erreurs sont alors retournées au serveur backend qui les retransmet au client. Le nœud *Update* permet de répondre au requis R.F.14.

## Simulation (Q4.5)

*[Un diagramme qui résume l’architecture. Un texte qui décrit et justifie les choix.*

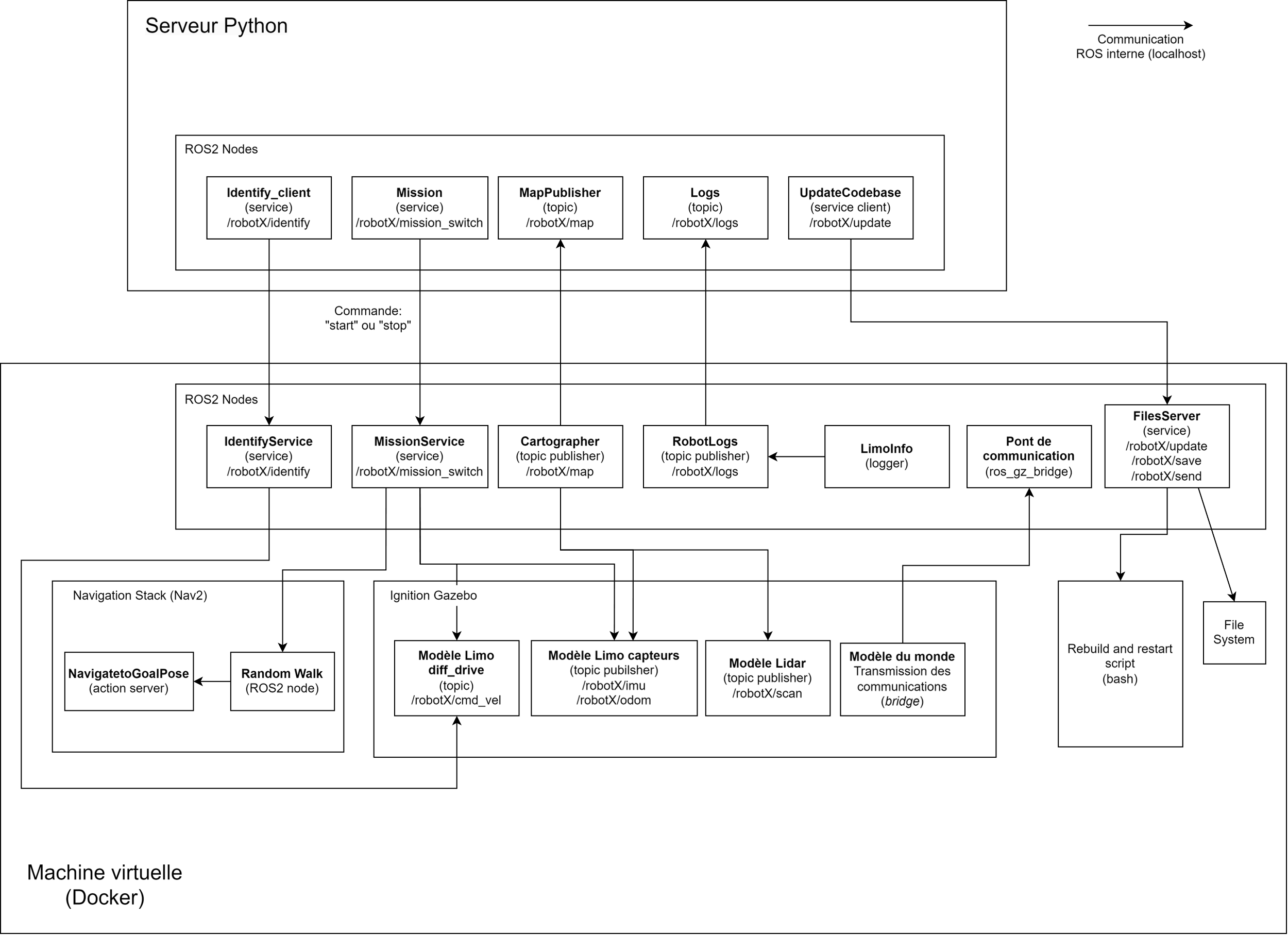
*Mention des requis concernés.]*

Figure 4. Diagramme de la simulation

La simulation adoptera une architecture très similaire au système embarqué. En effet, tous les éléments du système embarqué seront intégrés à la simulation afin de bien imiter la version qui sera sur le robot.

Ainsi, nous ajouterons un nœud ros\_gz\_bridge [9] qui permettra de faire le pont entre le logiciel de simulation Ignition Gazebo [7] et les nœuds ROS2 de notre système embarqué. Nous choisissons le simulateur Ignition Gazebo puisqu’il est adapté pour les applications utilisant ROS2, et utilise les mêmes conventions de ROS2 telle que développée dans la documentation officielle [7]. Un exemple de ceci est l’utilisation du topic *cmd\_vel* pour contrôler le déplacement du robot.

Finalement, la simulation et les nœuds ROS2 seront placés dans une machine virtuelle Docker [10] qui permettra de rendre la solution portable sur les différentes machines où la simulation sera exécutée.

## Interface utilisateur (Q4.6)

*[Un diagramme qui résume l’architecture. Un texte qui décrit et justifie les choix.*

*Mention des requis concernés.]*

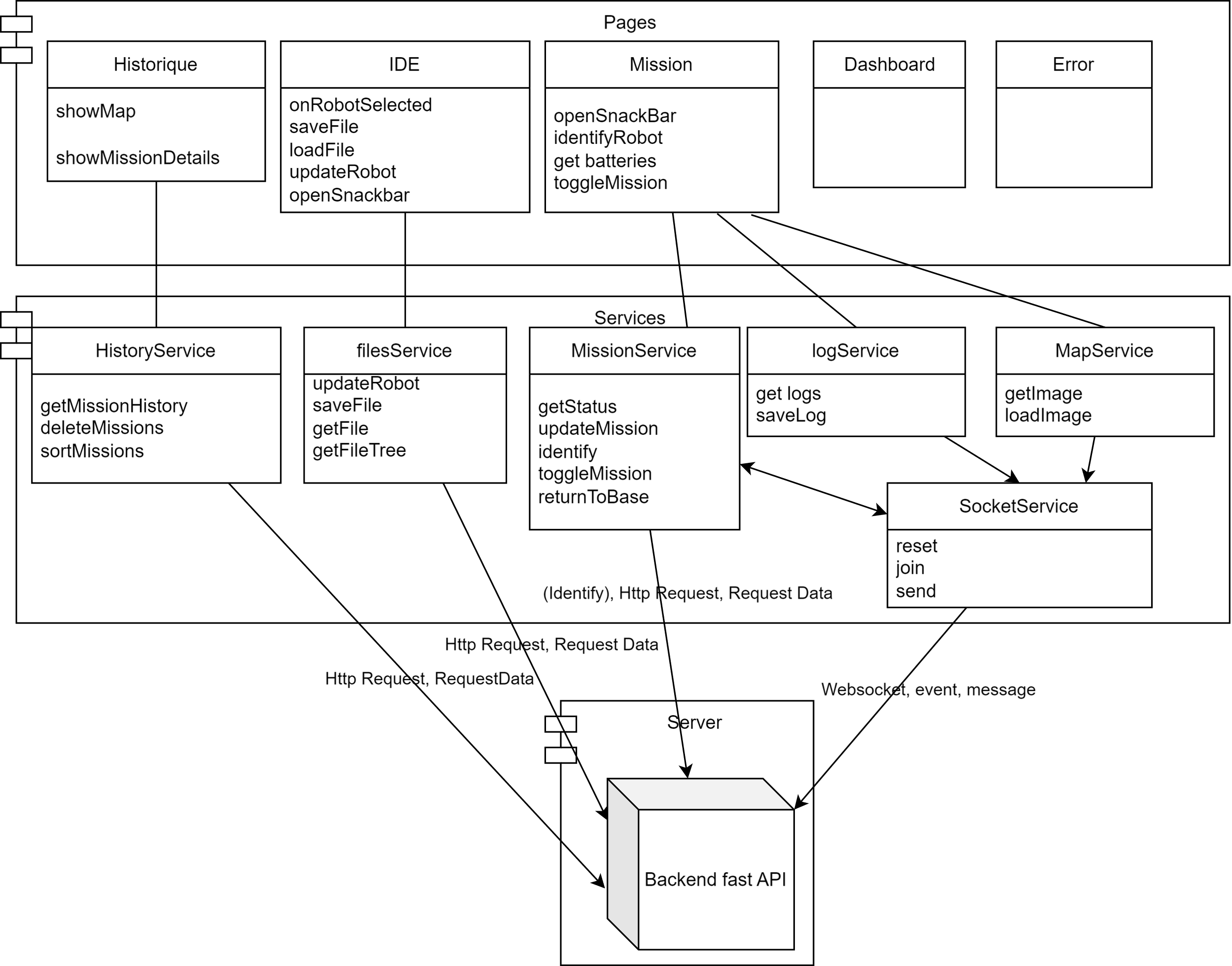


Figure 5. Diagramme de l’interface utilisateur

Le diagramme ci-dessus décrit l'architecture de notre interface utilisateur, mettant en évidence les différentes couches ainsi que les différentes interactions. On peut voir dans ce diagramme que nous avons fait le choix d'un design en 4 pages avec:

* La page "Mission" qui est responsable d'une mission en cours. En effet, elle permet de lancer et d'arrêter une mission, mais aussi de contrôler la mission en cours et de lire les informations liées à cette dernière, telles que la carte et les logs. Ceci est rendu possible par son association avec le service de mission, le service logs et le service Map qui possèdent toute la logique des fonctions à appeler pour réaliser les actions. Le service Mission communique en HTTP avec le backend pour répondre à Identify et utilise le SocketService pour une communication en Websocket avec le BackEnd afin de performer les opérations de la mission et récupérer en temps réel son statut ainsi que l’état des robots. Le logService et le MapService quant à eux utilisent exclusivement le SocketService pour leurs communications avec le backend afin de récupérer en temps réel les informations dont ils ont besoin via websocket.
* La page "Historique" permet d'afficher et de gérer les informations des dernières missions, et de les trier. Elle communique avec le HistoryService afin de récupérer les données en provenance du backend et d'effectuer des opérations sur ces dernières.
* La page "IDE" est en fait un éditeur de code. C'est une interface permettant de modifier le code source des robots et de le mettre à jour. Elle interagit avec le service "fileService” afin de récupérer le code source du robot et de mettre à jour le code des robots à la suite des modifications. Il communique avec le backend via HTTP
* La page "Dashboard" est la page de présentation de l'interface utilisateur. Elle présente notre projet et interagit avec le IdentifyService afin de permettre aux usagers d'identifier les robots. Le IdentifyService communique avec le backend avec HTTP, qui lui communique avec les robots afin de compléter les actions d'identification.

## Fonctionnement général (Q5.4)

*[Comment arriver à faire fonctionner votre système à partir du code développé. Interface, usager, téléchargement, initialisation et démarrage, etc…*

*Aussi, ajouter les détails pour que le correcteur puisse faire fonctionner votre système]*

**Requis :**

*Pour lancer notre projet, il est nécessaire d’avoir le programme Docker installé sur le système. Le système d’exploitation préférable est Linux (Ubuntu), mais nous avons des scripts adaptés pour Windows ou Mac si vous téléchargez l’équivalent des programmes (exemple Docker Desktop pour Windows).*

**Procédure :**

La première étape pour faire fonctionner notre système est d’abord de télécharger le code. Ceci peut être réalisé avec la commande suivante:

git clone --recurse-submodules git@gitlab.com:polytechnique-montr-al/inf3995/20241/equipe-102/INF3995-102.git

Pour démarrer le serveur frontal, il suffit de se déplacer dans le répertoire INF3995-Frontend, puis de lancer la commande suivante:

docker compose up

Il en va de même pour le serveur dorsal (répertoire INF3995-Backend). Les robots sont quant à eux différents. Dans leur cas, il convient de se déplacer dans le répertoire “INF3995-Robot/robot” et de lancer le script “launch-robot.sh”. Ce script s’occupe de préparer automatiquement l’environnement sur les robots puis de démarrer l’exécution du code. Il est possible de lui fournir le paramètre “-h” afin de connaître les paramètres qu’il prend en entrée. Des valeurs par défaut sont préétablies, ce qui rend les paramètres optionnels.

Lorsque ces étapes d’installation sont terminées avec succès, un opérateur peut interagir avec le robot au moyen de l’interface utilisateur, disponible à l’adresse [localhost:4200](http://localhost:4200/).

**Précision pour la simulation :**

Les étapes et programmes pour déployer la simulation se trouvent aussi dans le répertoire INF3995-Robot, mais il faut cette fois-ci se diriger vers le dossier “simulation” plutôt que “robot”. Il est important d’avoir d’abord installé l’outil rocker pour accéder à une interface graphique. Pour démarrer la simulation, il suffit alors de rouler les deux commandes suivantes:

docker build -t "rosignbase"

rocker --x11 --device=/dev/dri --volume $(pwd):/root/INF3995-Robot --port 22900:22900 --port 22901:22901 --port 22902:22902 --port 22910-22921:22910-22921 --image-name=rosign --name=simulation-ign rosignbase

Le code pour la simulation est dans le répertoire INF3995-Robot car il est similaire à celui qui roule sur les vrais robots. Pour la lancer, il faut

* Dans le cas de Windows, il faut ouvrir une console à la racine du répertoire, et lancer le docker adapté à Windows.

cd INF3995-Robot/simulation/windows

docker build -t simulation-docker .

cd ../..

docker run --rm -p 5901:5901 -v .:/root/INF3995-Robot -it simulation-docker bash -c "TVNC\_WM=xfce /opt/TurboVNC/bin/vncserver -securitytypes None; /bin/bash"

Une fois le docker lancé, on peut se connecter grâce à un client VNC tel RealVNC, à l’adresse localhost:5901.

# Processus de gestion

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

*[Fournir les estimations de coûts du projet.]*

# Dépenses liées aux employés :

En prenant en compte les rôles de chaque membre de l’équipe et les taux horaires reliés à chaque rôle. Nous partons du principe que nous utiliserons les 630 heures-personnes maximales. On déduit alors ces résultats :

* heures par employé
* Donc en tout pour les employés le salaire s’élève à 83 475$

# Dépenses liées au matériel :

Pour les deux robots que nous devons nous procurer, le prix unitaire de chaque robot est de 2 499,95$US; en ajoutant les frais estimés de livraison qui sont de 80$US, on obtient pour finir 3 460,47$CA, soit 6 920,94$CA au total pour les deux en tenant compte du taux de change en date du 16 février 2024.

Ensuite pour notre environnement de tests, nous avons besoin d’un parcours avec obstacles amovibles ainsi qu’un routeur avec des câbles Ethernet. Le coût de ce matériel est estimé à 527,36$CA.

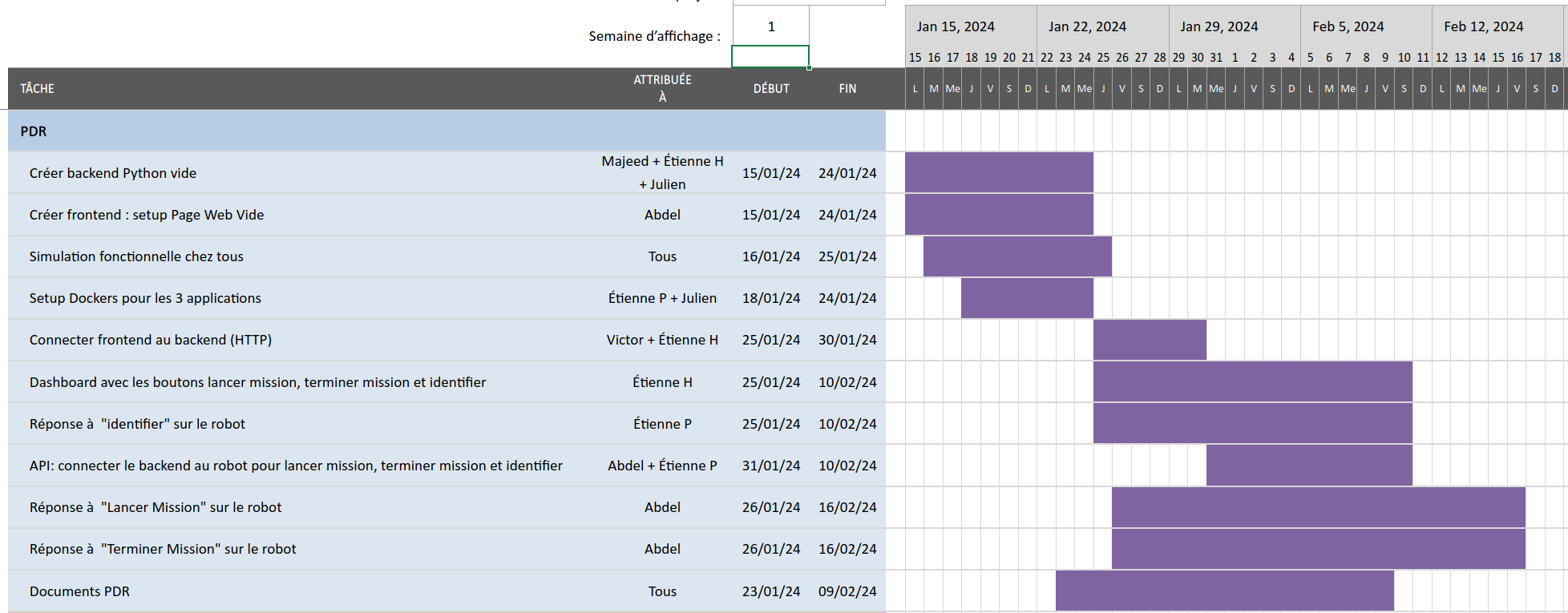
# Dépenses totales :

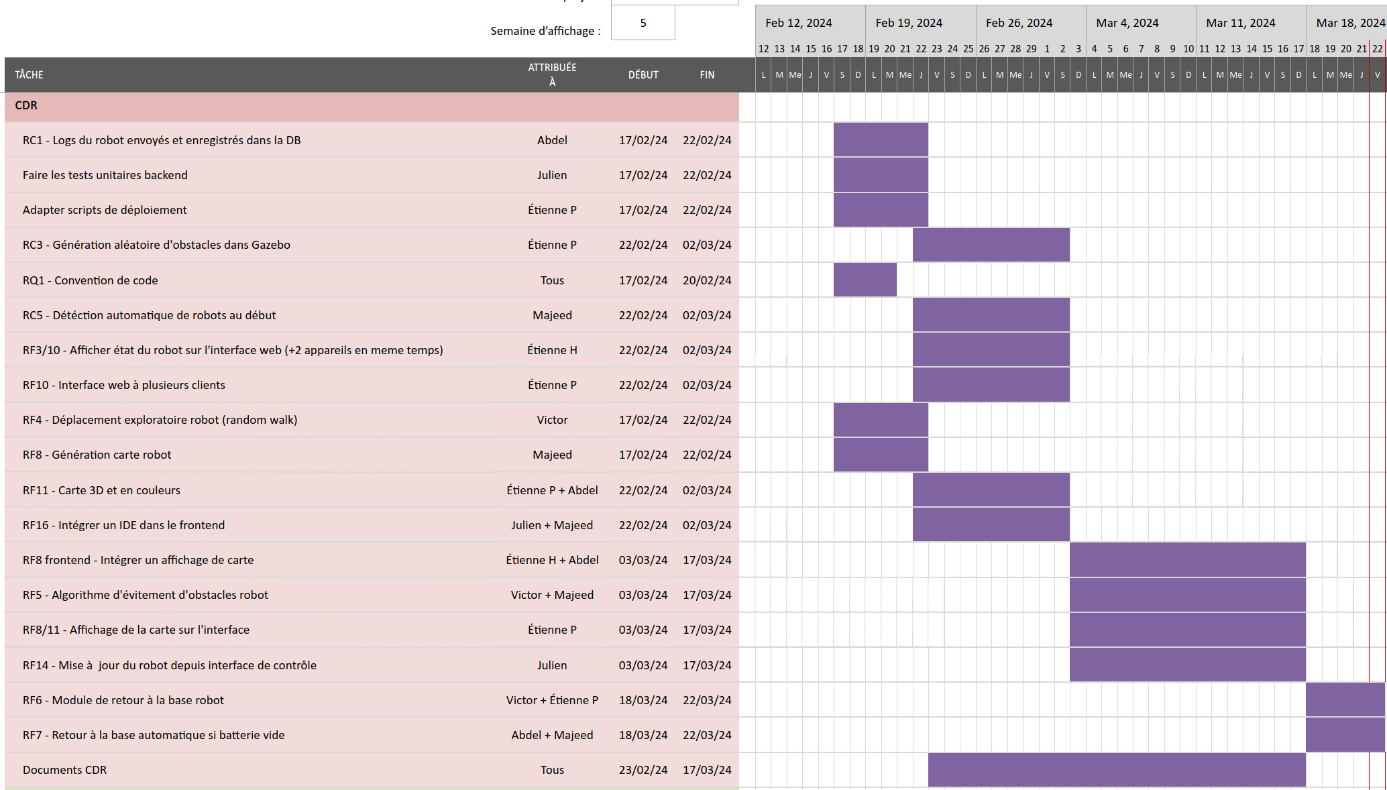
Si nous faisons l’addition de tous ces coûts nous avons : 83 475$CA + 6 920.94$CA + 527.36$CA = 90 923.30$CA

## Planification des tâches (Q11.2)

*[Inclure: Un diagramme indiquant l’allocation du temps pour chaque tâche. Fournir une vue d’ensemble de l’horaire des 2, 3, 4 (maximum) principaux jalons (milestones). On doit aussi voir la répartition des tâches entre les membres de l’équipe.]*

Voici les différentes répartitions du temps pour nos différents jalons.







Ce diagramme de Gantt démontre la division des tâches à effectuer en fonction du temps. À partir des requis fonctionnels sélectionnés, nous avons établi les tâches à effectuer et les avons réparties comme montré dans le calendrier, en fonction des forces de chaque membre. L’ordre des tâches a été décidé en fonction des livrables attendus des revues de conception, et selon l’ordre de progression des fonctionnalités (par exemple le lancement de mission est prévu d’être terminé avant le début de l’historique des missions).

Les jalons principaux sont le PDR, le CDR et le RR.

Dans ce diagramme, les tâches ne sont pas toutes indépendantes. En effet, les requis RF3 et RF10 dépendent de RC5, les requis RF5, RF6 et RF7 dépendent de RF4. De plus, RF9 et RF11 dépendent de RF8.

## Calendrier de projet (Q11.2)

*[Insérer un tableau qui indique les dates cibles de terminaison des phases importantes, des dates de version et autres jalons. Un résumé seulement.]*

|  |  |  |
| --- | --- | --- |
| Date cible | Nom du Jalon | Description |
| 2024-01-25 | Préparations préliminaires  V0.5 | Fin de créations et paramétrages des différents environnements de développement. |
| 2024-02-10 | Communication de base avec le robot | Communication de base fonctionnelle avec les robots pour accomplir des tâches simples. |
| 2024-02-15 | V1.0 | Version 1 fonctionnelle pour les robots, ce qui comprend les programmes pour: Frontend, Backend et robots |
| 2024-02-16 | PDR | Remise du document PDR et réponse à l’appel d’offres. |
| 2024-03-02 | V1.5 | Intégration de nouvelles fonctionnalités comme : carte 2D/3D, chercher statut du robot sur l’interface, déplacement exploratoire robot et intégration IDE dans l’interface. |
| 2024-03-17 | V2.0 | Intégration de nouvelles fonctionnalités comme : algorithme d’évitement d’obstacles, une carte fonctionnelle sur l’interface, mise à jour du robot depuis l’interface. |
| 2024-03-17 | CDR | Remise du CDR |
| 2024-03-30 | V2.5 | Intégration de nouvelles fonctionnalités sur le contrôle du positionnement du robot. |
| 2024-04-12 | V3.0 | Tous les requis sont fonctionnels sur interface, backend et robots et la base de données. |
| 2024-04-16 | RR | Remise du RR |
| 2024-04-16 | Présentation orale | Notre équipe est prête à faire la présentation orale du projet. |

## Ressources humaines du projet (Q11.2)

*[Indiquer le nombre et le type de ressources humaines nécessaires, incluant les qualifications spéciales ou l’expérience des membres de l’équipe.]*

Le projet repose sur les 6 membres de l’équipe qui ont tous une bonne expérience en programmation générale, mais aussi en développement embarqué et *full stack*. Notre équipe bénéficie aussi d’expérience en conteneurisation (Docker) et Linux. Tout ce savoir acquis est un grand avantage puisque toute l’équipe devra participer au niveau technique.

Nos développeurs ont des qualifications en développement Web avec Angular, et des connaissances *full stack* pour créer une passerelle de communication avec le serveur backend python. Ils ont aussi de l’expérience avec Python.

Chaque membre a aussi de l’expérience en travail d’équipe, ce qui est indispensable pour un projet avec des dates serrées et des livrables fréquents. Le membre possédant de l’expérience en gestion, comble le rôle de coordonnateur dans notre équipe. Ce rôle est grandement nécessaire dans des défis d’envergure. Il va nous permettre de rester organisés et assurer que les communications dans l’équipe sont bonnes.

# Suivi de projet et contrôle

## Contrôle de la qualité (Q4)

*[Tous les biens livrables doivent être soumis à un processus de révision. Une révision est requise afin de s’assurer, au moyen de lignes directrices et de listes de vérification, de la qualité de chaque bien livrable.]*

Le contrôle de la qualité pour notre projet est pris à cœur par les membres. Nous avons donc mis en place plusieurs outils et conventions qui nous permettent de garder un code fonctionnel et lisible au fur et à mesure du projet. Comme premiers outils, nous implémentons des tests unitaires automatisés sur : notre frontend, notre backend et le code des robots. L’équipe s’engage à une couverture de 80% ou plus et de garder les tests à jour afin qu’ils restent pertinents et utiles au développement. Ces tests seront roulés automatiquement (grâce aux *pipelines* Gitlab) avant que les changements soient intégrés au code général.

La gestion de version est opérée grâce à Git et GitLab. Notre utilisation de Gitlab est normée pour rester organisés et uniforme. Nous utilisons les outils intégrés afin d’implémenter un tableau Kanban et assigner les différentes tâches. De plus, la procédure pour intégrer des changements au projet est de passer par des *merge requests* qui permettent de revoir les modifications effectuées et vérifier la qualité de ceux-ci. Grâce à ces différentes mesures, nous assurons un projet de bonne qualité.

## Gestion de risque (Q11.3)

*[Par exemple : Lister les principaux risques de ce projet et estimer leur importance. Donner quelques solutions de remplacement possibles et la façon dont l’équipe entend gérer les changements en cours de projet.]*

Dans ce type de projet, certains évènements peuvent se produire et nuire au projet dans son ensemble. Nous avons alors anticipé certains risques et les avons classés par ordre d’importance.

**Pour tout problème :**

Réunion d’urgence si besoin. Analyse du problème et prise d’une décision adaptée afin de garder le projet sur les bons rails.

**Risque #1 : Nombre d’heures-personnes :** **[Risque de gestion]**

Un risque de gestion important serait de dépasser le nombre d’heures prévues, mais ne pas avoir accompli toutes les missions du mandat. Si ceci est atteint, il faudra alors négocier pour augmenter le budget alloué et le temps total. Nous sommes cependant bien au fait avec ce risque. L’équipe s’organisera alors pour faire bon usage du temps et remettre un produit de bonne facture.

**Risque** **#2 : Ne pas rencontrer les requis aux dates clés : [Risque de gestion]**

Si des imprévus frappent notre équipe, il est possible de prendre du retard. Si ce retard devient trop important alors nous pourrions passer à côté de certaines dates importantes. Si cela arrive bel et bien, des négociations seront faites avec l’agence afin de discuter de différentes solutions. Nous proposons d’ores et déjà que si le problème se réalise nous livrions un produit fonctionnel, mais non complet et pouvons terminer le projet avec plus de temps.

**Risque #3 : Risques de bris matériels : [Risque technique]**

Nous travaillons avec beaucoup d’outils onéreux et certains sont fragiles. En cas de problèmes techniques, si le problème fait que l’objet est inutilisable alors il sera remplacé si son remplacement coûte moins de 300$, qui est la valeur de la caution prélevée. En cas de casse au-dessus de 300$ alors une demande de prêt d’équipement sera faite à l’Agence. La demande sera pour soutenir l’équipe jusqu’à la fin du développement du produit. Ces risques techniques doivent être réfléchis et gardés en tête puisqu’ils sont bien réels, surtout lorsqu’on travaille avec des robots physiques comme c’est notre cas.

**Risque #4 : Perte d’un ou plusieurs membres de l’équipe : [Risque de ressources humaines]**

Si un membre de l’équipe vient à la quitter, cela va empiéter sur la réussite du projet. ~~Celui-ci fût pensé~~ La réponse à ce risque de gestion a été élaborée pour une équipe d’entre 5 et 6 personnes. Nous considérons alors ce tableau :

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre de membres quittant le projet | Entre la date *x* et *y* | Le projet peut être mené à bien dans les délais énoncés | Chercher de membre(s) de remplacement |
| 1 | 8 Janvier 2024 et 9 Février 2024 | OUI | OUI |
| 1 | 9 Février 2024 et 17 Mars 2024 | OUI | NON |
| 1 | 18 Mars 2024 et 16 Avril | NON | NON |
| 2 | 8 Janvier 2024 et 9 Février 2024 | OUI | OUI |
| 2 | 9 Février 2024 et 16 Avril | NON | NON |
| Plus de 2 | 8 Janvier 2024 et 16 Avril | NON | NON |

## Tests (Q4.4)

*[Identifier et préciser quelques tests pour chaque sous-système, tant pour le matériel que le logiciel. Il devrait y avoir un lien entre ces tests et les tâches décrites plus haut.]*

Backend:

* Vérifier le type et la mise en forme des données envoyées et reçues par les différentes routes HTTP (mission, logs...)
* Tester la logique associée à chaque route, une entrée A doit donner un résultat attendu (ex: le traitement de la route /mission/start doit exécuter les fonctions nécessaires au démarrage du robot).
* Vérifier comment s’effectue l’enregistrement des données avec une BD de test. Nous aurons des requêtes SQL pour confirmer que les bonnes données sont présentes.
* Vérifier comment sont lues les données depuis la base de données de tests.

Frontend:

* Vérifier que les fonctions associées aux interactions vont effectuer le bon traitement. (ex: la fonction associée au bouton start mission doit déclencher une requête vers le serveur sur la route /mission/start)
* Vérifier que pour un type de données attendues envoyé par le serveur (serveur *mock*), la fonction effectuera le traitement et donnera les résultats attendus.

Robots et simulation:

* Il est important de mettre en place une routine de test en conditions réelles pour les robots et la simulation.
* Les procédures de tests pour chaque requis sont décrites dans le fichier pdf Tests.pdf

## Gestion de configuration (Q4)

*[Par exemple : Donner quelques renseignements sur le système de contrôle de version, l’organisation du code source, des tests et les fichiers de données ainsi que la documentation relative au code source et à la documentation de conception. La séparation et l’intégration entre les fichiers de description du logiciel]*

Nous utilisons Gitlab pour la gestion de versions, nous disposons d’un dépôt global en ligne que nous avons subdivisé en dépôts individuels pour chaque composante du système: Robot, Backend, Frontend Web. En plus de stocker le code source du système, nous avons ajouté l’organisation de notre travail sur ce projet Gitlab, c’est-à-dire l’assignation de tache, l’automatisation de tests, et les revues de codes.

Une documentation de notre projet est faite avec le système Wiki de Gitlab, et nous l’utilisons pour les informations liées au fonctionnement de nos robots, et des procédures de déploiement et de tests. Ceci est à la fois utile pour les robots et pour la simulation.

## Déroulement du projet (Q2.5)

***[CDR et RR seulement]***

*[Dans votre équipe, qu’est-ce qui a été bien et moins bien réussi durant le déroulement du projet par rapport à ce qui était prévu dans l’appel d’offre initialement.]*

Le respect de l’échéancier a été plus difficile que prévu initialement. En effet, dans l’appel d’offre nous avions prévu terminer des tâches à un moment précis, mais nous ne sommes pas toujours parvenus à compléter nos objectifs chaque semaine. À titre d’exemple, nous espérions avoir terminé la génération aléatoire d’obstacles pour Gazebo avant le 2 mars, mais cette tâche s’est terminée le 5 mars.

De plus, nous avons eu de la difficulté à prévoir les responsables de chaque tâche. En effet, bien que nous l’eussions préparé, il s’est avéré que certaines tâches sont plus longues à exécuter que prévu. En plus de cela, les tâches suivantes dépendent parfois de l’achèvement des tâches plus longues que prévu, ce qui nous a forcés à nous ajuster et à réassigner les tâches en cours de route.

Ensuite, nous étions préparés à faire face à des problèmes d’ordre technique dans l’appel d’ordre. Cependant, ceux-ci ont été légèrement plus intenses que prévu. En effet, notre compréhension de ROS et des outils adjacents était très faible, voire nulle, au début du projet, ce qui nous a donné une vision incomplète de la complexité du projet. Lorsque nous avons réellement commencé à développer des outils et programmes pour le robot, nous avons été frappés par la réalité qui est que ROS est beaucoup plus complexe que prévu. Par ailleurs, en plus de la complexité de ROS, nous avons frappé des difficultés à faire interagir ensemble les différents composants, comme rviz, Gazebo ainsi que les *packages* ROS fournis par Agilex.

Au niveau de ce qui a été bien réussi, notre équipe possède un très bon esprit d’équipe. En effet, les membres travaillent tous très bien et sont prêts à mettre une pause à leur tâche pour aider un membre en difficulté. De plus, nous sommes tous très compréhensifs lorsqu’un autre membre ne réussit pas à atteindre un objectif en raison d’un événement externe.

D’autre part, notre équipe possède une forte cohésion. En effet, nous en sommes à la dernière étape des stades de cohésion, puisque nous avons traversé l’euphorie collective, mais nous avons aussi terminé l’étape de tensions secondaires puisque nous sommes tous très motivés en plus d’accepter la diversité et de posséder une ouverture sur les différents membres. L’état de notre groupe nous permet donc d’être productifs tout en ayant du plaisir à travailler en équipe.

Finalement, la ponctualité de nos membres a été un autre point qui s’est bien déroulé au courant du projet. Comme prévu, les membres étaient tous assez ponctuels pour permettre à l’équipe de bien fonctionner et de mener ses activités. Bien qu’il y ait eu quelques petits échappements, la ponctualité de l’équipe était, en somme, excellente.

1. **Résultats des tests de fonctionnement du système complet (Q2.4)**

***[CDR et RR seulement]***

*[Qu’est-ce qui fonctionne et qu’est-ce qui ne fonctionne pas*.*]*

Pour nous assurer de la qualité du produit que nous développons ainsi que de sa résilience, nous avons en continu testé notre solution dans des situations différentes. Les requis du CDR ont été tous respectés, cela dit, certains ont eu plus de mal à fonctionner sur le robot, par rapport à la simulation. C’est pour nous le cas de la navigation. En effet, l’effet aléatoire de l’exploration entraine parfois des blocages du robot, ce qui fait qu’il lui arrive de passer trop de temps avant de contourner un obstacle ou de se repositionner pour explorer une nouvelle zone.

Cela a été mis en lumière par des tests en faisant varier le nombre d’obstacles et d’éléments présents dans son circuit, permettant donc d’évaluer sa réactivité et sa capacité à reconnaitre les situations. En dehors de ça, les autres fonctionnalités fonctionnent bien sur le robot et la simulation. Nous avons essayé d’identifier des cas limites en lançant des missions à 1 ou à 2 robots, en faisant varier l’environnement autour des robots, en faisant varier la quantité de taches que le système devait traiter en même temps (ajouter des publisher, des subscriber, des services) à la date de cette remise, notre système a été résiliant. Les échecs rencontrés en phase de développement ont permis d’améliorer et de faire évoluer la solution.

Nos requis sont séparés en 5 groupes généraux : l’identification, le changement d’état de la mission, les journaux d’événements, la carte ainsi que la mise à jour du robot.

Tout d’abord, pour tester l’identification nous lançons la commande identifier pour chaque robot séparément et vérifions que ceux-ci tournent bien sur eux-mêmes. L’angle total de rotation n’est pas toujours équivalent puisque des paquets sont parfois perdus en raison de réseau trop congestionné. La simple réaction à la commande confirme son fonctionnement.

Ensuite, le changement d’état de la mission est vérifiable en lançant le démarrage de la mission. Cette action devrait mettre en marche les robots qui se mettront alors à suivre un trajet aléatoire. Si le robot ne réagit pas ou s’il cesse complètement de bouger pour plus de 30 secondes, on peut confirmer qu’une erreur s’est produite et que la mission n’a pas pu être exécutée complètement.

Les journaux d’événements sont testables assez facilement. En effet, il suffit de démarrer le système et de consulter l’interface utilisateur. Celui-ci devrait afficher des messages. Si ce n’est pas le cas, on peut confirmer qu’il y a un problème dans l’envoi ou dans la génération des journaux.

La carte est aussi une partie importante de notre système. Il est possible de vérifier son fonctionnement en lançant une mission et en vérifiant que la carte s’affiche bien sur l’interface utilisateur. Si la carte n’est pas présente ou si le logo de Google s’affiche, on peut conclure qu’un problème s’est produit lors de l’établissement de la carte.

Finalement, pour vérifier que la mise à jour du robot s’est bien exécutée, il suffit de lancer la mise à jour (en modifiant un fichier), puis en demandant à un robot de s’identifier. En effet, ceci permet de vérifier que le programme sur le robot n’a pas été brisé, ce qui confirme que la mise à jour s’est exécutée avec succès.

1. **Travaux futurs et recommandations (Q3.5)**

***[RR seulement]***

*[Qu’est-ce qui reste à compléter sur votre système? Recommandations et possibles extensions du système.]*

1. **Apprentissage continu (Q12)**

***[RR seulement]***

*[Un paragraphe par membre (identifié en début de paragraphe) de l’équipe qui doit aborder chacun de ces aspects de façon personnelle:*

1. *Lacunes identifiées dans ses savoirs et savoir-faire durant le projet.*
2. *Méthodes prises pour y remédier.*
3. *Identifier comment cet aspect aurait pu être amélioré.]*
4. **Conclusion (Q3.6)**

***[RR seulement]***

*[Par rapport aux hypothèses et à la vision que vous aviez du système lors du dépôt de la réponse à l’appel d’offre, que concluez-vous de votre démarche de conception maintenant que le système est complété?]*

# Références (Q3.2)

*[Inclure toute documentation supplémentaire utilisable par le lecteur. Ajouter ou référencer toute norme technique de projet ou plans applicables au projet.]*

[1] Angular (2023), *Introduction to Angular concepts*, Documentation officielle d’Angular.

[En ligne] Disponible: <https://angular.io/guide/architecture>

[2] SocketIO (2024), *Socket.IO Introduction*, Documentation SocketIO Version 4.

[En ligne] Disponible: <https://socket.io/docs/v4/>

[3] FastAPI (2024), *FastAPI Documentation* [En ligne] Disponible: <https://fastapi.tiangolo.com>

[4] PostgreSQL (2023), *PostgreSQL About Page* [En ligne] Disponible: <https://www.postgresql.org/about/>

[5] Open Robotics. (2024) The ROS\_DOMAIN\_ID. [En ligne]. Disponible :

<https://docs.ros.org/en/humble/Concepts/Intermediate/About-Domain-ID.html>

[6] Open Robotics. (2024) Interfaces. [En ligne]. Disponible :

<https://docs.ros.org/en/humble/Concepts/Basic/About-Interfaces.html>

[7] Open Source Robotics Foundation. (2014) Gazebo Tutorials. [En ligne]. Disponible :

<https://classic.gazebosim.org/tutorials>

[8] Trossen Robotics. (2022) AgileX LIMO. [En ligne]. Disponible :

[/docs.trossenrobotics.com/agilex\_limo\_docs/](https://docs.trossenrobotics.com/agilex_limo_docs/)

[9] ROS 2 Documentation (2021), *Welcome to the documentation for ros\_gz\_bridge*

[En ligne] Disponible: <https://docs.ros.org/en/humble/p/ros_gz_bridge/>

[10] Docker Docs (2024), *Docker overview,* [En ligne] Disponible: <https://docs.docker.com/get-started/overview/>