MVP SOA

Mars Y, to the space and beyond

Equipe D

Yahiaoui Imène
Gazzeh Sourour
Ben Aissa Nadim
Al Achkar Badr

Plan du rapport

Introduction

Le présent rapport a pour but de présenter le Produit Minimal et Viable (MVP) élaboré dans le cadre du projet.

Voici les technologies et outils qui ont été employés :

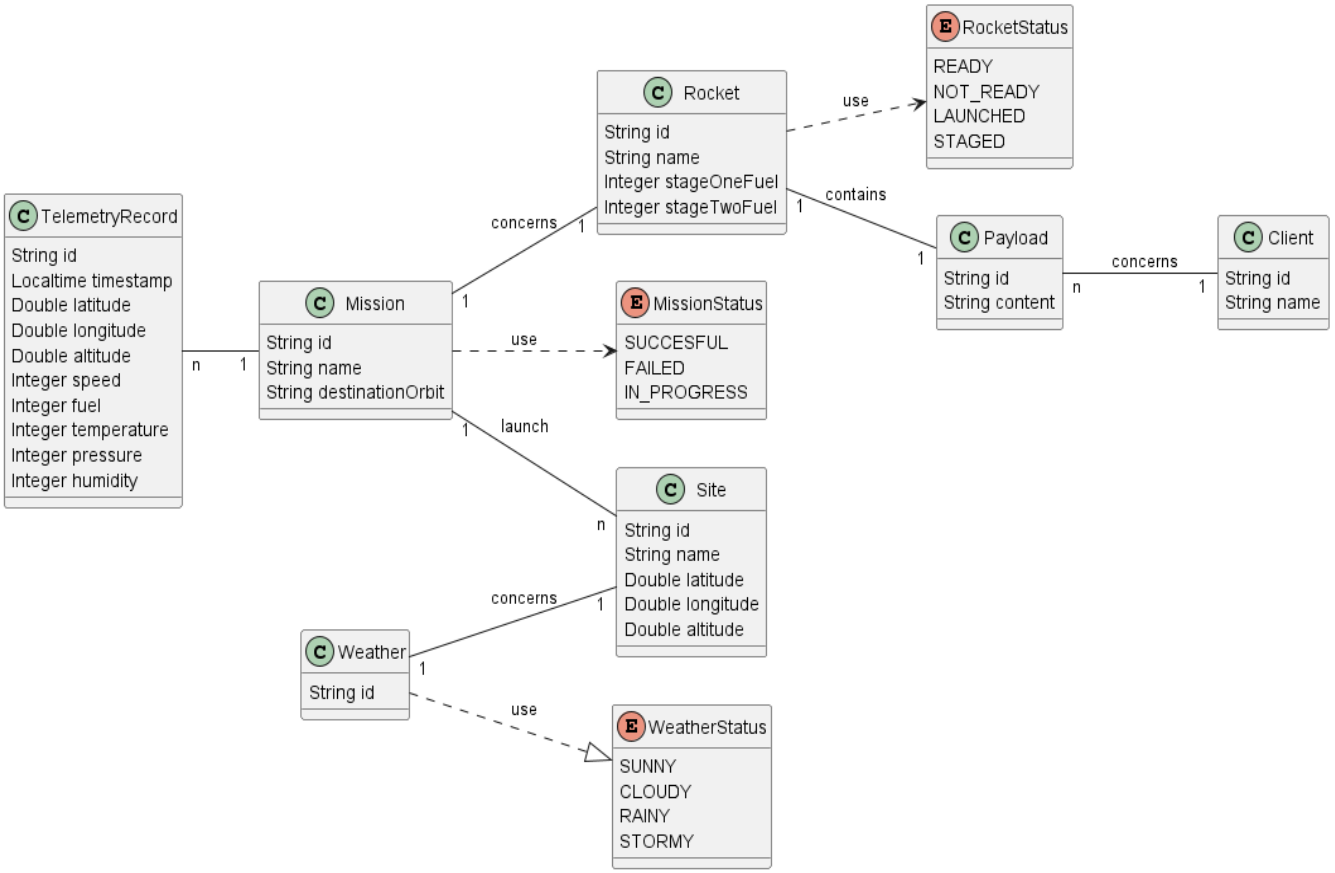
- Pour l'ensemble des services, le framework de programmation utilisé est NestJs.
- La base de données choisie est MongoDB.
- Pour la gestion de projet, Nous avons intégré des tests automatisés dans notre processus de développement grâce à GitHub Actions

Hypothèses

Suite à la lecture du sujet, nous avons émis les hypothèses suivantes :

- Les données de télémétrie seront générées automatiquement toutes les 3 secondes par le système hardware. Cette automatisation élimine le besoin d'une intervention manuelle pour obtenir ces données, par les différents départements ce qui est essentiel pour une surveillance en temps réel de la mission et la détection rapide de problèmes éventuels.
- L'objectif de l'orbite est atteint lorsque la fusée atteint une certaine latitude, longitude, et une altitude spécifiée par notre système.
- La payload émet de la télémétrie à partir de l'orbite sur laquelle elle a été livrée, une fois la mission terminée.
- Étant donné que le missile se divise après l'étape du ‘staging’, nous avons supposé que nous aurions besoin de simuler trois services matériels : un pour la fusée avant la mise à disposition, un pour le booster après la mise à disposition et un pour la charge utile après sa livraison.
- Le throttle Max Q sera activé lorsque le rocket atteint une certaine altitude spécifiée par notre système, représentant ainsi la phase atmosphérique du vol où le rocket réduira la puissance des moteurs et la vitesse.
- Nos hardwares cessent d'envoyer des données lorsque l'orbite est atteinte et que le booster a atterri. À ce moment, la mission est considérée comme terminée.

Diagramme de classes



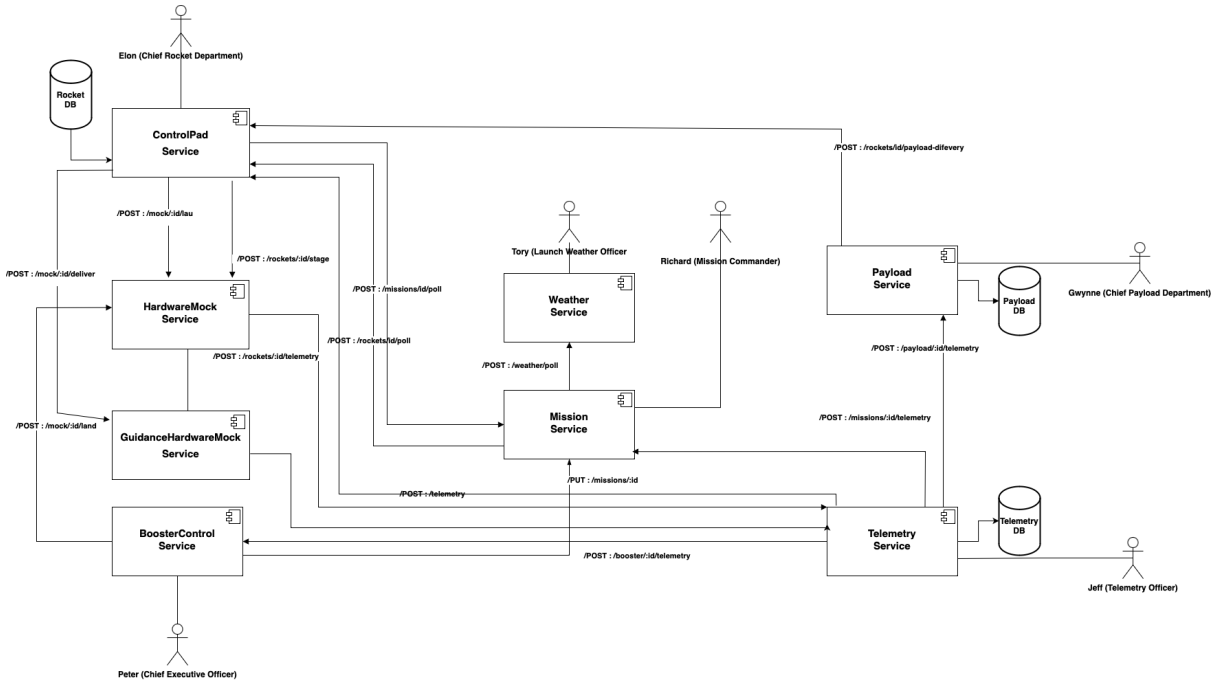
Voici le diagramme de classe global pour notre système. Nous avons les missions qui sont enregistrées avec un statut spécifique ainsi que le statut du booster pour assurer un suivi tout au long de leur déroulement et une gestion efficace des différentes missions du système.

Chaque mission est associée à un site spécifique ayant une altitude, une longitude et une latitude. Cela facilite l'obtention des données météorologiques du site et permet de déterminer si le booster a effectué son atterrissage dans la zone spécifiée. Elle est également associée à un fusil fourni par Rocket, lequel présente différents statuts tout au long de la mission. Ce fusil est également lié à un payload.

Nous avons stocké également les télémétries, qui constituent la base du suivi de la mission après son lancement. De plus, nous avons créé une télémesure spécifique au booster, qui sera générée lors de la deuxième étape après le staging.

Diagramme d’architecture :

[draw.io link]



Description détaillée des micro service :

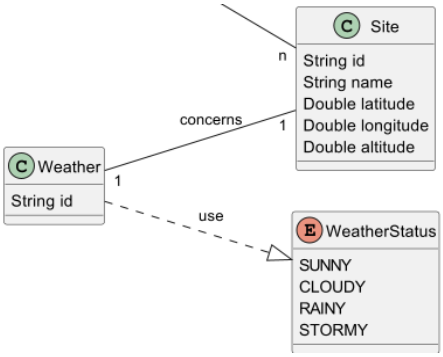
Service Weather

Rôle :

Ce service est géré par Tory, l'officier de la météo de lancement, qui transmet des données météorologiques à l'aide de la longitude et de la latitude fournies d’un site d’une mission.

Il permet de vérifier si le statut météorologique est favorable ou non pour s'assurer que la fusée se comporte correctement avant le lancement lors du l’appel mission service.

Modèle métier :



Weather est l'agrégat de ce service.

Interface REST :

Ce service est accessible à l'adresse http://localhost:3002 avec une documentation swagger API à /doc/weather

- ***POST /weather/status?lat=&long=*** permet à Tory de connaître le statut météorologique : nuageux, ensoleillé...
- ***POST /weather/poll*** prend un JSON de latitude et longitude, permettant au service mission de déterminer si le lancement est autorisé (go) ou non (no-go).

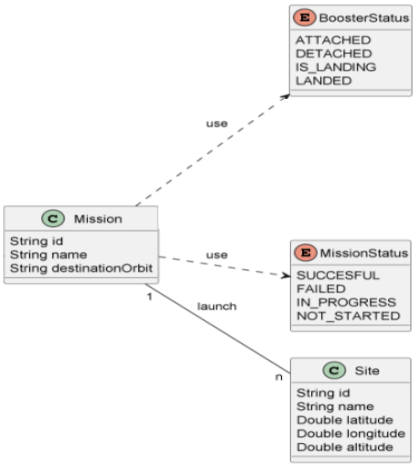
Service Mission

Rôle :

Ce service est géré par Richard, le Commandant de la mission. Il permet de créer des missions et des sites, et chaque mission est associée à une fusée. Il permet, au début de la mission, de vérifier si les conditions sont favorables au lancement en utilisant les services météorologiques et de contrôle.

Lors du lancement de la mission, il recevra les données de télémétrie et s'assurera de leur sécurité en utilisant des informations telles que la vitesse, l'altitude, etc. Si ces données ne sont pas sécurisées, il contactera le service ControlPad pour déclencher la destruction de la fusée. De plus, la mission pourra recevoir des demandes du service BoosterControl pour modifier le statut de son booster en cas de changements.

Modèle métier :



Mission est l'agrégat du modèle de ce service.

Interface REST :

Ce service est accessible à l'adresse http://localhost:3000 avec une documentation swagger API à /doc/mission

- ***/POST /mission/:id/poll*** : permet d'avoir la décision finale du lancement de fusée.
- ***/POST /rockets/:id/telemetry*** : permet de recevoir les télémétries générées et d'analyser s'il y a une anomalie.
- ***PUT /missions/:id*** : permet de changer le statut de la mission et du booster à l'appel du service boosterControl.

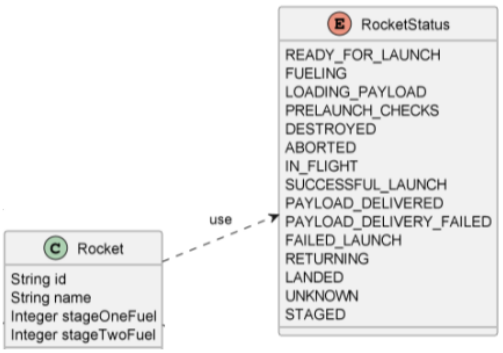
Service ControlPad

Rôle :

Ce service est géré par Elon, le Chef du Département des Fusées, qui supervise les fusées et leur statut tout au long de la mission, depuis avant le lancement jusqu'à la fin.

Au début, il vérifie le statut de la fusée pour autoriser le lancement et démarrer la mission. Après le lancement, ce service surveille les données de télémétrie reçues, et lorsque le carburant est épuisé, il effectue le "staging" en appelant le simulateur matériel (hardware mock). De plus, avant d'atteindre le point MaxQ, il fait appel au service guidance responsable de la deuxième étape du "staging". Ce service gère également le changement de statut de la fusée à chaque étape de la mission

Modèle métier :



Rocket est un modèle partagé avec le service Mission**.**

Interface REST :

Ce service est accessible à l'adresse http://localhost:3001 avec une documentation swagger API à /doc/launchpad

- ***GET /rockets/:idRocket/status*** : permet à Richard de savoir le statut de la fusee avant le lancement

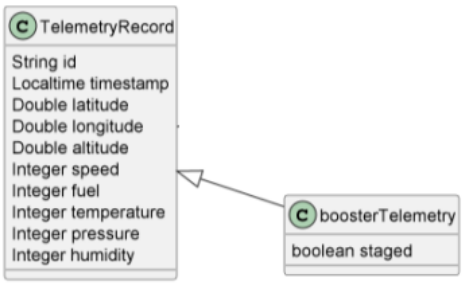
- ***POST /rockets/:idRocket/poll*** : permet au service mission de savoir à travers le le statut si le lancement est autorisé.
- ***POST /rockets/:id/launch*** : permet d'effectuer le lancement de la fusée une fois que les conditions ont été vérifiées auprès du service mission.
- ***POST /rockets/:id/stage*** : permet à Richard de faire le staging de la fusée en appelant le service du HardwareMock.
- ***POST /rockets/:id/payload-delivery*** : appelée par le service de télémétrie pour être notifié lorsque l'orbite souhaitée est atteinte.

Service Télémétrie

Rôle :

Ce service est géré par Jeff, le responsable de la télémétrie, et il joue un rôle crucial dans notre système. Il est chargé de gérer les données de télémétrie depuis le lancement de la fusée jusqu'à la fin de la mission avec succès. Il reçoit, stocke et traite les données de télémétrie de la fusée et du booster provenant des appels effectués par les services HardwareMock et Guidance, puis les transmet aux différents départements du système selon les besoins.

Modèle métier :



Interface REST :

Ce service est accessible à l'adresse http://localhost:3004 avec une documentation swagger API à /doc/marsy-telemetry

- ***GET /telemetry*** : permet d'afficher les différents télémétries
- ***POST /telemetry*** : permet de stocker les telemetry d'un fusée
- ***POST /telemetry/:idRocket/booster*** : permet de stocker les télémétries d'un booster d'une fusée

Service boosterControl

Rôle :

Ce service est sous la supervision de Peter, le Directeur Général, et a pour mission de surveiller les données de télémétrie du booster, de son décollage jusqu'à son atterrissage. Il coordonne également avec le service HardwareMock pour gérer l'atterrissage du booster, et il communique avec la mission pour mettre à jour le statut du booster, que ce soit à l'atterrissage ou à la fin de la mission.

Interface REST :

Ce service est accessible à l'adresse http://localhost:3030 avec une documentation swagger API à /doc/marsy-boostercontrol

- ***POST /booster/:rocketId/telemetry*** : permet de recevoir les télémétries du booster liée à une fusée.

Service HardwareMock

Rôle :

Ce service a pour rôle de générer automatiquement les données de télémétrie de la fusée avant le "staging", puis du booster jusqu'au "staging", à intervalles de 3 secondes. De plus, il est capable de prendre des décisions cruciales pendant le vol, telles que le "staging" et l'atterrissage.

Interface REST :

Ce service est accessible à l'adresse http://localhost:3005

- ***POST /mock/:idRocket/launch*** : permet de faire le lancement de la fusée et de la génération automatique des télémétries.
- ***POST /mock/:idRocket/land*** : permet de faire le land du booster de fusée.
- ***POST /mock/:idRocket/destroy*** : permet de détruire la fusée et de stopper ainsi la génération des télémétries.
- ***POST /mock/:idRocket/stage*** : permet de faire le stage de la fusée

Service GuidanceHardware :

Rôle :

Ce service intervient pendant la deuxième étape du "staging" pour générer les données de télémétrie de la fusée, ce qui évite de submerger le système hardware avec des

requêtes simultanées du booster et de la fusée. De plus, il gère la phase du "Max Q" où la fusée doit traverser en toute sécurité.

Interface REST :

- ***POST /mock-guidance/:idRocket/launch*** : permet la génération automatique des données de télémétrie de la fusée lors de la deuxième étape du "staging".
- ***POST /mock-guidance/:idRocket/delive***r : permet de réaliser la livraison du payload sur la fusée.
- ***POST /mock-guidance/:idRocket/throttle-down*** : permet de réduire la poussée des moteurs lors de l'atteinte de la phase MAX Q.

Service Payload

Rôle :

Ce service, dirigé par Gwynne (Chef du Département des Charges Utiles), a pour mission d'assurer la livraison réussie de la charge utile dans l'espace en respectant l'orbite ou la trajectoire requise. Il est également responsable de la réception, du stockage et de la consultation des données de télémétrie de la charge utile afin de garantir l'atteinte de l'orbite souhaitée.

Interface REST :

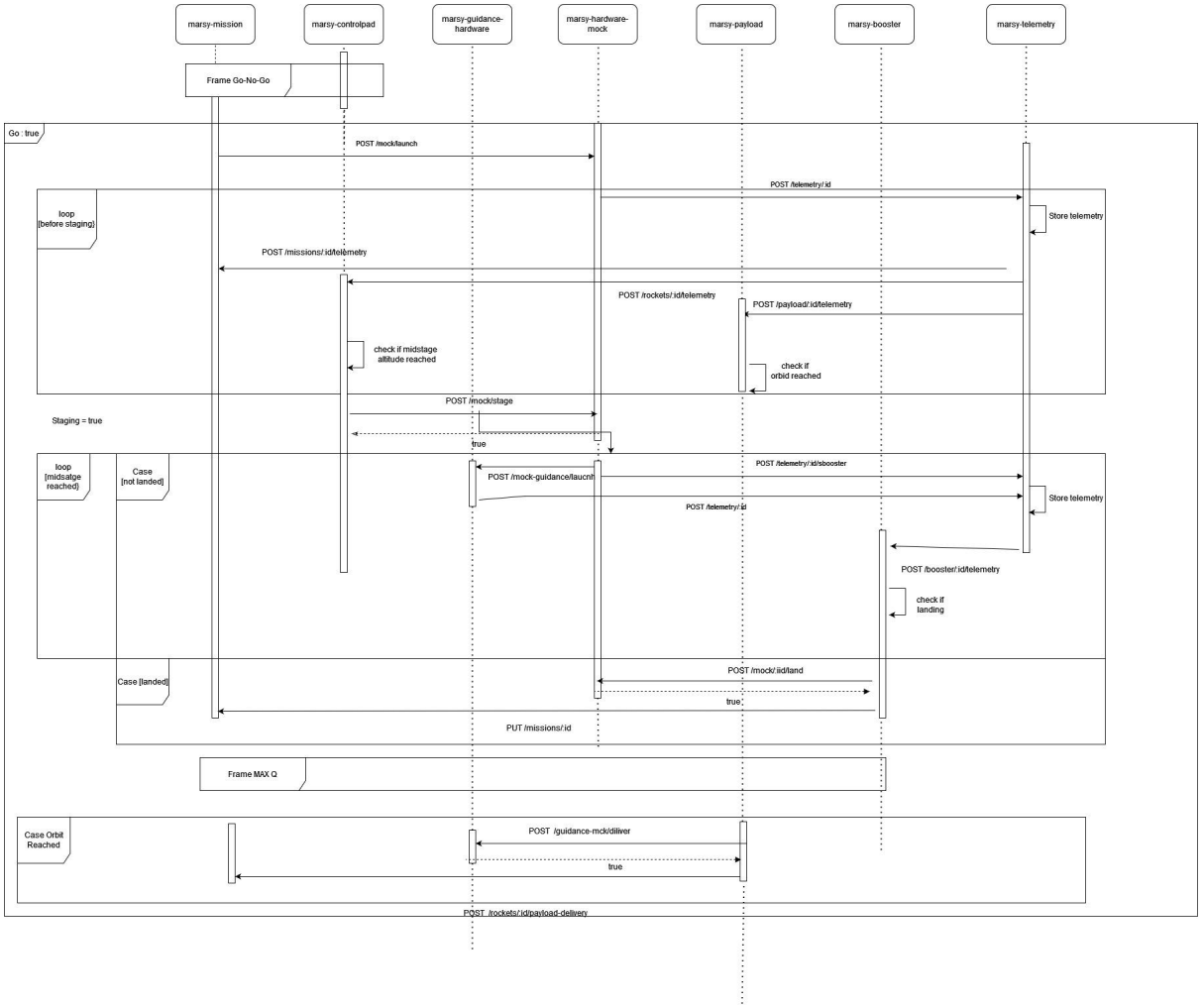
Ce service est accessible à l'adresse http://localhost:3006/rockets avec une documentation swagger API à /doc/marsy-payload

- ***POST /payload/:rocketId/telemetry*** : permet de recevoir les télémétries liées au payload.

Diagrammes de Séquence

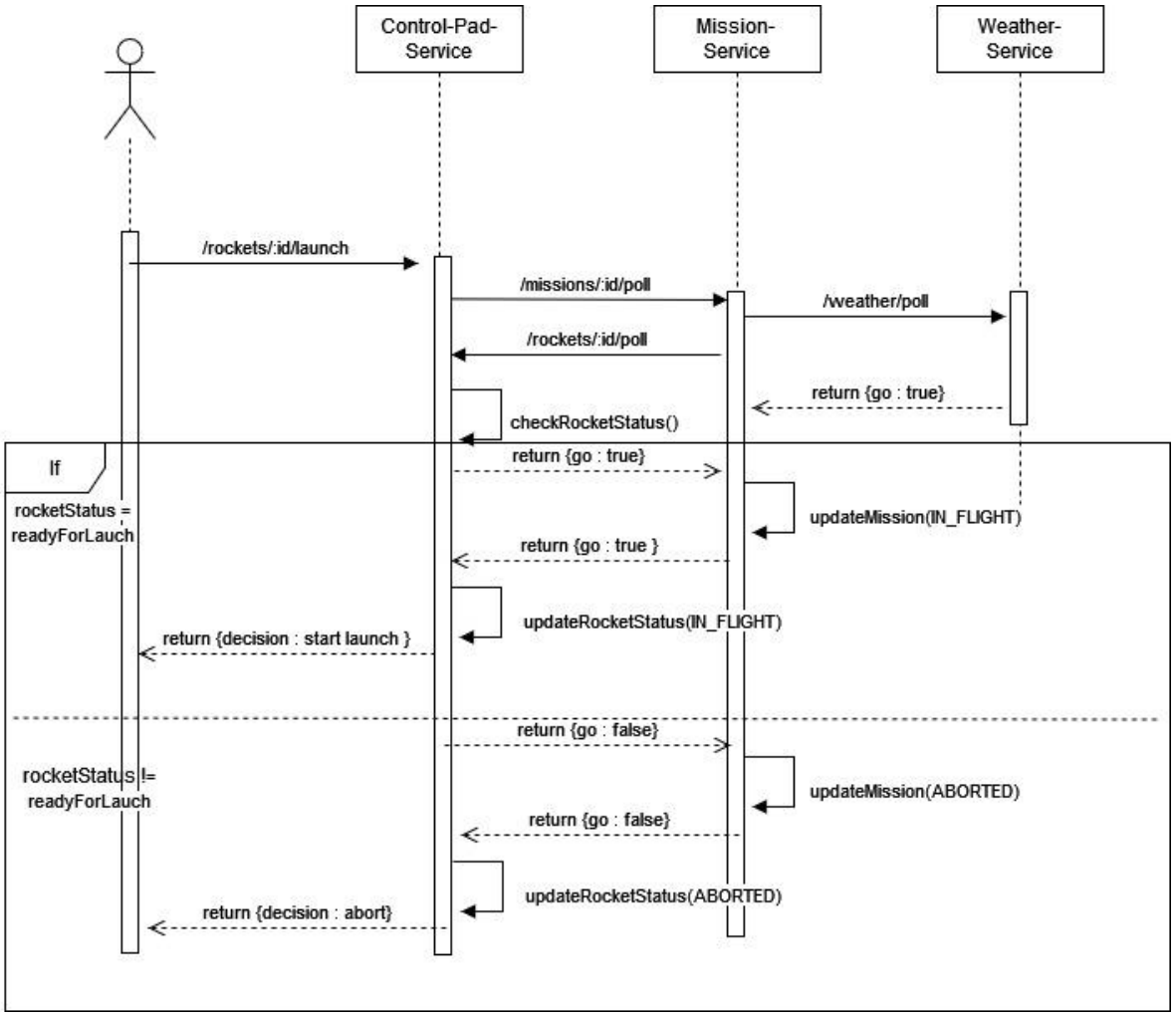
Scénario principale

[draw.io link]



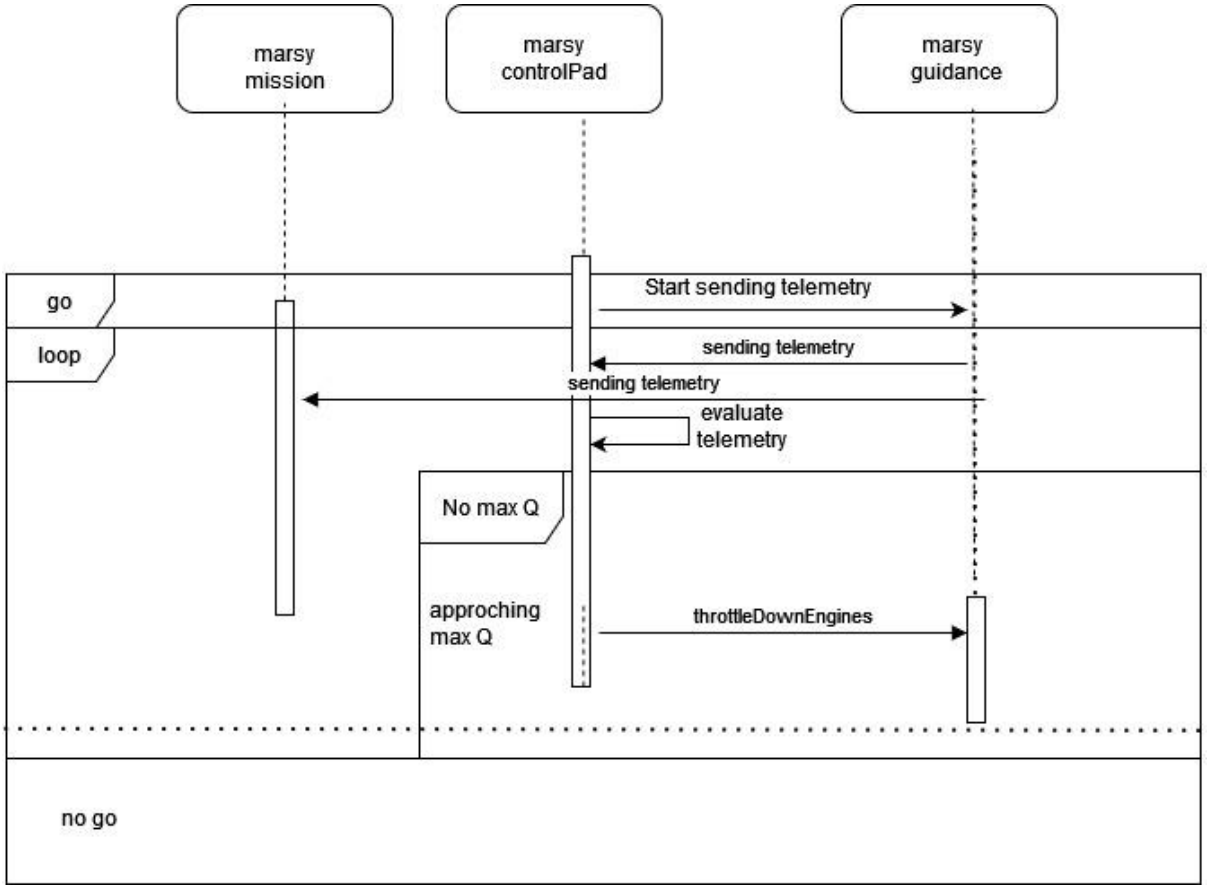
Frame Go-no-Go

[draw.io link]



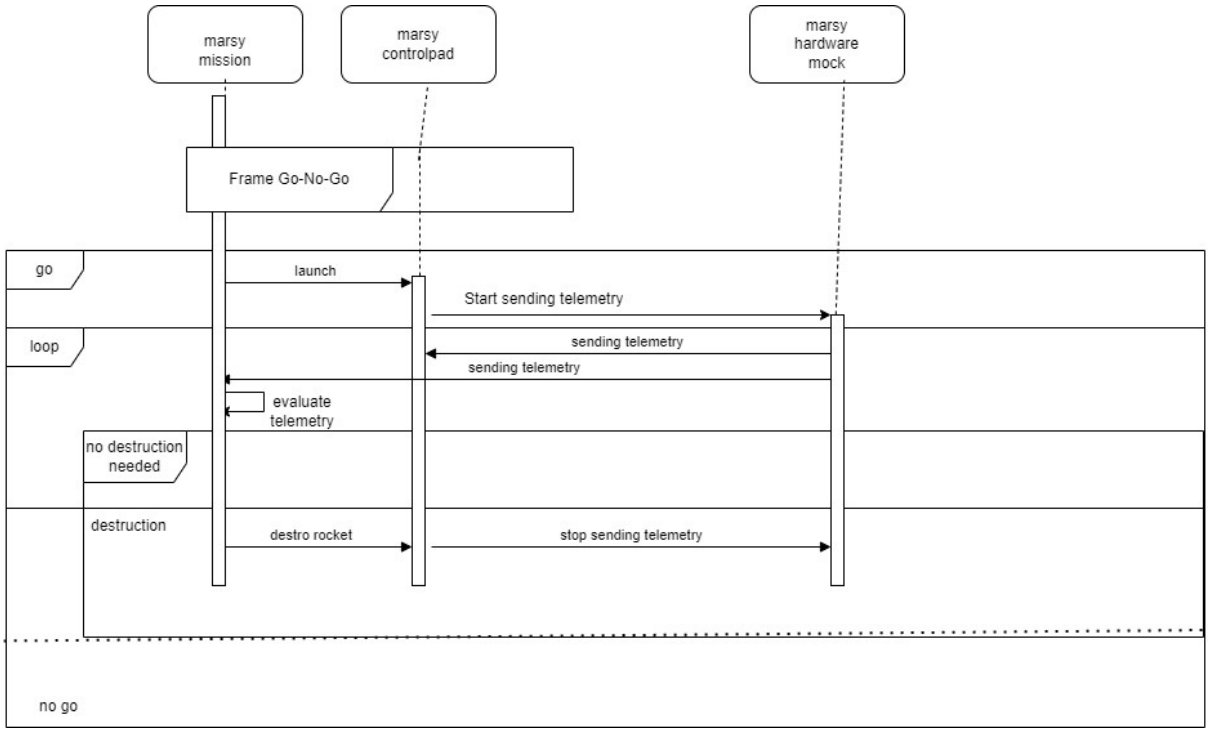
Frame max Q

[draw.io link]



Frame destroy

[draw.io link]



Explications :

- Le lancement de la fusée se fait avec la commande POST /rockets/:id/launch par le service control-pad. A ce moment, le service mission va vérifier le go avec POST /missions/:id/poll en appelant les deux service weather avec POST /weather/poll qui va vérifier le statut de la météo, et le service de la rocket POST /rockets/:id/poll qui va vérifier si le statut de la fusée est prêt pour le lancement.

- Après le lancement de la mission, le service de la mission enverra une requête POST /mock/launch au service hardware-mock. Le service hardware-mock générera à ce moment les données de télémétrie de la fusée et les enverra via POST /telemetry/:id au service de télémétrie pour les stocker. Ensuite, il enverra ces données de télémétrie aux différents départements : payload (POST /payload/:id/telemetry), mission (POST missions/:id/telemetry) et control-pad (POST /rockets/:id/telemetry).

- À chaque réception de données de télémétrie, le service du control-pad vérifie si la fusée a atteint une certaine altitude pour effectuer le staging. À ce moment, il appellera le service hardware-mock (POST /mock/stage). Ce service sera alors chargé de collecter les données de télémétrie du booster et les envoyer au service de télémétrie via POST /telemetry/:id/booster pour les stocker. Ensuite, il va envoyer ces données au service du booster-control via POST /booster/:id/telemetry.
- Pendant cette nouvelle étape, le nouveau service hardware guidance-hardware sera responsable de la gestion des données de télémétrie de la fusée.

- Le service control-pad recevant les télémétries après le staging depuis le hardware-guidance, va vérifier si on s'approche à la MAX-Q, si c'est le cas il envoie POST /mock-guidance/:id/throttle-down.

- Le service du booster-control supervisera les données de télémétrie du booster et enverra une requête au service hardware-mock via POST /mock/land au moment de l'atterrissage. Il mettra également à jour le statut de la mission en envoyant une requête PUT à /missions/:id.
- Le service hardware-mock ne transmettra plus de données de télémétrie du booster après un atterrissage réussi.

- Parallèlement, le service payload supervisera tout au long de la mission si l'orbite a été atteinte en utilisant les données de télémétrie reçues. Si tel est le cas, il enverra une requête POST à /guidance-mock/deliver, puis informera la mission via POST /rockets/:id/payload-delivery.

- La mission va également superviser les télémétries et en cas d'anomalie, elle envoie une requête au hardware-mock pour détruire la fusée POST /hardware-mock/:id/destroy

- La mission est fini si le payload est délivré ainsi que le booster atterrit ou bien la fusée est détruite

Tests

Notre approche de validation et d'assurance qualité a évolué pour garantir la fiabilité de nos microservices. Nous avons initié des tests d'intégration en utilisant des scripts avec des opérations curl pour émuler les scénarios, puis nous avons migré vers un script Python encapsulé dans un conteneur Docker. Ce conteneur s'exécute sur le même réseau que nos microservices, facilitant ainsi le suivi des tests en temps réel.

En parallèle, nous avons adopté une méthodologie de gestion du code source utilisant Git Flow strategy. Chaque nouvelle fonctionnalité est développée dans une branche dédiée, avec des tests automatisés. Les PRs déclenchent automatiquement notre pipeline d'intégration continue. Cette approche globale assure la qualité de notre code et la stabilité de notre système, tout en optimisant notre processus de développement.

Evaluation de l’architecture

Notre système repose sur une séparation claire des rôles au sein de nos services, chacun étant adapté à des tâches spécifiques. Cette approche favorise une compréhension aisée de notre système et le rend facilement extensible. Par exemple, la distinction entre les services **HardwareGuidance** et **HardwareMock** offre un avantage significatif en réduisant la charge sur **HardwareMock**, qui n'a pas à gérer simultanément les télémesures du booster, de la fusée et des cas particuliers comme MaxQ. Cette modularité renforce notre capacité à faire évoluer notre système pour répondre aux besoins des changements simplifiant ainsi la gestion des données et des ressources.

De plus, notre système a été délibérément conçu pour favoriser l'évolutivité et l'intégration transparente aux approches basées sur des événements. Notre système s'active automatiquement dès la requête de lancement jusqu'à la conclusion de la mission, garantissant ainsi une gestion fluide et réactive des opérations, en temps réel.

Prise de recul

Nous avons réussi à développer un MVP fonctionnel qui englobe toutes les fonctionnalités requises par les scénarios utilisateurs qui nous ont été assignés. Il nous a permis de valider notre compréhension des microservices en action.

L'utilisation de l'architecture de microservices nous a permis de découper notre application en composants indépendants, ce qui a facilité le développement, le test et la maintenance. Chaque microservice pouvait être développé par un membre de l’équipe, ce qui a accéléré le processus de développement.

Lors de la réalisation du MVP, nous avons été confrontés à des défis liés à la coordination de la communication entre les microservices. Pour surmonter ces obstacles, nous avons pris la décision de définir les points d'entrée de chaque service en amont du développement des nouvelles fonctionnalités.

Nous sommes également conscients qu'un aspect critique de notre architecture actuelle est la dépendance sur le service de télémétrie en tant que point unique de défaillance. Nous reconnaissons l'importance de résoudre cette vulnérabilité pour garantir la fiabilité et la disponibilité continues de notre application. Nous avons l'intention de prioriser la résolution de ce problème dans nos prochaines étapes de développement afin de renforcer la robustesse de notre système global.

En résumé, notre MVP fonctionnel confirme le potentiel des microservices pour la rapidité de développement, tout en soulignant la nécessité d'une coordination efficace.