

RAPPORT FINAL - PROJET SOA



Mars Y, to the space and beyond

Equipe D :

**Al Achkar Badr
Ben Aissa Nadim
Gazze Sourour
Yahiaoui Imène**

SOMMAIRE

INTRODUCTION :	3
RAPPEL DES PERSONAS :	3
USER STORIES DEVELOPPEES :	3
HYPOTHESES FAITES PAR L'EQUIPE :	6
ARCHITECTURE COMPLÈTE :	7
Diagramme d'architecture :	7
DÉTAILS DES SERVICES :	7
Mission Service :	7
ControlPad Service :	9
Weather Service :	11
Payload Service :	12
BoosterControl Service :	13
Telemetry Service :	15
Web caster Service :	18
HardwareMock Service :	19
GuidanceHardwareMock Service :	20
PayloadHardwareMock Service :	21
Client Service :	22
Broadcast Service :	23
Pilot Service :	24
SCÉNARIOS :	25
Scénario 1 : Mission réussite :	25
Scénario 2 : Anomalie détectée :	31
Scénario 3 : Envoie d'une deuxième fusée avec un service qui tombe :	32
Test de charge :	35
PRISE DE RECUL :	37
Forces et faiblesse :	37
Forces :	37
Faiblesses :	38

INTRODUCTION :

RAPPEL DES PERSONAS :

- Richard, le **commandant de la mission**, qui **supervise l'ensemble de la mission**.
- Elon, le chef du **département de la fusée**, qui **supervise la fusée** elle-même et ses systèmes.
- Tory, l'officier **météo du lancement**, qui s'assure que les conditions météorologiques sont sûres sur l'ensemble des sites.
- Jeff, l'officier de **télémétrie**, qui **s'occupe de la communication des données entre le matériel de vol et le sol**.
- Gwynne, le chef du **département Payload**, responsable de la **livraison de la Cargo du client** et de l'efficacité de la trajectoire ou de l'insertion en orbite.
- Peter, le directeur général de Mars Y, qui dirige stratégiquement les activités et les objectifs de l'entreprise.
- Marie, le **Webcaster des lancements**, qui **explique aux téléspectateurs les objectifs de la mission** et détaille les **événements** du lancement **en temps réel**.

USER STORIES DEVELOPPEES :

- 1) En tant que Tory (Launch Weather Officer), je dois vérifier l'état des conditions météorologiques, afin de pouvoir m'assurer que les conditions sont valables pour un fonctionnement sûr de la fusée.
- 2) En tant qu'Elon (chef du service des fusées), je dois surveiller l'état de la fusée, afin que je puisse être sûr que la fusée se comporte correctement avant le lancement.
- 3) En tant que Richard (Commandant de la mission), je dois effectuer un sondage Go/No Go auprès de chaque service de contrôle avant de donner le feu vert final au lancement, afin d'être sûr que tout est nominal avant le lancement.
Le sondage Go/No Go doit être effectué dans l'ordre suivant :
 - Département météo (Tory)
 - Département des fusées (Elon)
 - Commandant de la mission (Richard)
- 4) En tant qu'Elon (chef du département des fusées), je dois envoyer l'ordre de lancement à la fusée après le GO de Richard, afin que la fusée puisse décoller dans l'espace et livrer la payload.
- 5) En tant que Jeff (Telemetry Officer), je veux recevoir, stocker et consulter les données télémétriques de la fusée pendant toute la séquence de lancement (avant le lancement, jusqu'à la

fin de la séquence de lancement), afin de pouvoir vérifier que tout fonctionne comme prévu, et que si quelque chose ne va pas, je puisse trouver la cause du problème.

- 6) En tant qu'Elon (chef du département des fusées), je veux mettre la fusée à l'étage à mi-vol (séparer la fusée en 2 parties), afin que la fusée reste aussi efficace que possible, en laissant derrière elle le premier étage, désormais vide de carburant, et en continuant avec le deuxième étage et la fusée.
- 7) En tant que Gwynne (chef du département des charges utiles), je veux livrer la payload (satellite/sonde) dans l'espace sur la bonne orbite ou trajectoire, de sorte que les désirs du client aient été satisfaits par la mission.
- 8) En tant que Richard (commandant de mission), je veux pouvoir donner l'ordre de destruction de la fusée en cas d'anomalie grave, afin que la fusée ne puisse pas suivre une trajectoire incontrôlée et pour éviter des dommages potentiels au sol.
- 9) En tant que Peter (Directeur général), je veux que le booster (premier étage que nous avons précédemment mis au rebut après la séparation (staging) de la fusée en 2 parties) de la fusée atterrisse, afin que je puisse le réutiliser plus tard et avoir ainsi un coût d'exploitation moins élevé pour chaque lancement, afin d'être compétitif sur le marché des lancements spatiaux.
- 10) En tant que Jeff (Telemetry Officer), je veux recevoir, stocker et consulter les données télémétriques du premier étage, afin de pouvoir assurer le fonctionnement du booster depuis la rampe de lancement jusqu'à son atterrissage.
- 11) En tant que Gwynne (Chief Payload Department), je souhaite recevoir, stocker et consulter les données télémétriques de la charge utile, afin que Mars Y puisse certifier que les paramètres de l'orbite souhaités par le client sont assurés.
- 12) En tant qu'Elon (Chief Rocket Department), je veux que la fusée traverse Max Q de façon à ce que la payload et le matériel de vol ne subissent pas de contraintes excessives. niveau de sécurité. Pour ce faire, les moteurs de la fusée doivent ralentir pour réduire la charge. Max Q est la phase de vol atmosphérique où le vol du véhicule atteint la pression dynamique maximale en raison de la densité de l'air et de la vitesse de la fusée.
- 13) En tant que Richard (Commandant de la mission), je veux que la procédure de lancement suive les événements ultérieurs afin d'avoir une vue d'ensemble de la mission.

- (1) Préparation de la fusée (ravitaillement en carburant, vérification de l'état de la fusée, etc.)
- (2) Fusée sur alimentation interne
- (3) Démarrage (T-00:01:00)
- (4) Démarrage du moteur principal (T-00:00:03)
- (5) Décollage/Lancement (T+00:00:00)
- (6) MaxQ
- (7) Arrêt du moteur principal
- (8) Séparation des étages
- (9) Démarrage du deuxième moteur
- (10) Séparation du carénage

(11) Arrêt du deuxième moteur

(12) Séparation/déploiement de la charge utile

La séquence d'atterrissage du premier étage est également inclus, mais les événements ne sont pas directement liés. La chronologie ci-dessus après la séparation de l'étage, mais celle ci-dessous :

- (1) Manœuvre de retournement
 - (2) Combustion d'entrée
 - (3) Guidage
 - (4) Combustion d'atterrissage
 - (5) Déploiement des jambes d'atterrissage
 - (6) Atterrissage
- 14) En tant que Richard (commandant de la mission), je veux que les journaux de mission soient stockés et consultables afin de respecter les réglementations des autorités aérospatiales et pour les enquêtes de l'entreprise.
 - 15) En tant que Marie (Webcaster), je veux être au courant des événements de la procédure de lancement en temps réel afin de pouvoir en parler sur le flux Web en temps réel.
 - 16) En tant que Peter (directeur général), je veux que mon entreprise puisse gérer des lancements multiples en utilisant des fusées différentes afin de soutenir la croissance de l'entreprise et de renforcer sa position dans l'industrie spatiale.
 - 17) En tant que Richard (commandant de la mission), je veux que le système au sol déclenche des anomalies et envoie des notifications en cas de dysfonctionnement de la fusée, afin d'assurer le contrôle de la mission.
 - 18) En tant que Richard (Commandant de la mission), je veux que la mission soit avortée (destruction de la fusée, etc.) dès qu'une anomalie de gravité critique est détectée afin de ne pas mettre en danger la zone environnante avec une fusée incontrôlable sans aucune intervention humaine.

User stories ajoutées par nous :

- 19) En tant que client de la mission, je souhaite recevoir une notification indiquant que le payload a été livré avec succès. Cela me permettra de déclencher le démarrage du satellite correspondant et d'informer un autre service (la télévision) des détails.
- 20) En tant que service tiers (i.e. chaîne de télévision, ...), je veux diffuser publiquement l'événement et partager les informations avec mon audience. Les informations reçues devront être au format compatible avec notre système de communication. Une fois ces informations sont validées, je veux partager les informations avec mon audience.
- 21) En tant que client, je souhaite que mon équipe puisse piloter le satellite et changer sa direction (paramètres orbitaux), afin de résoudre tout problème de diffusion auquel notre chaîne de diffusion est confronté.

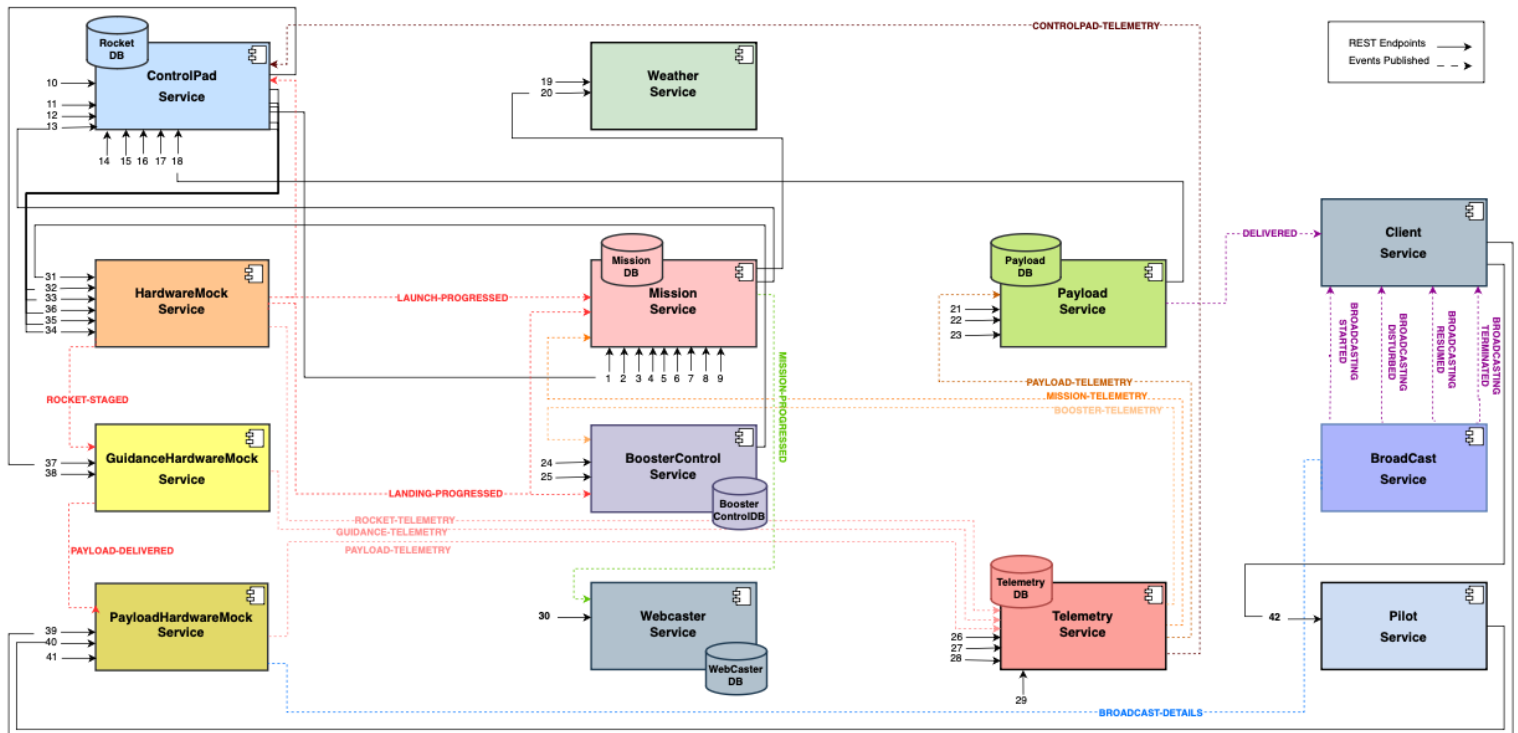
HYPOTHESES FAITES PAR L'EQUIPE :

- Les données de télémétrie sont générées automatiquement toutes les 3 secondes par les différents hardwares : propergol (hardware dans la première phase), booster (hardware dans la deuxième phase), guidance et payload.
- Les télémesures agrégées dans le service de télémétrie sont traduites et transformées par le service de télémétrie lui-même en données adéquates dont les autres départements ont besoin et qui leur sont ensuite envoyées.
- Le service HardwareMock représente le propergol de la fusée qui fait corps avec la fusée avant de sa séparation du reste. C'est lui qui devient le booster dans la deuxième phase. Il envoie la télémétrie de la fusée pendant la propulsion, mais après le staging, il envoie la télémétrie du booster.
- La détection des anomalies est effectuée dans le service de télémétrie. Il signale les dysfonctionnements au service des missions. Le service des missions décide alors d'ordonner au service de contrôle (Controlpad) de détruire la fusée dysfonctionnelle.
- Seul le service du contrôle donne des ordres à la fusée lors de la première phase (propulsion avant le staging).
- C'est le service de booster qui reçoit les télémetries du booster et qui le contrôle et déclenche son atterrissage lorsqu'il s'approche du sol.
- Dans la deuxième phase, le système de guidage poursuit le voyage vers l'orbite et envoie des données télémétriques qui sont ensuite utilisées pour déterminer le moment de la livraison de la payload.
- La progression du système de guidage dans l'espace est suivie par le service des payloads.
- Le service des payloads lance une demande de livraison lorsque les paramètres orbitaux correspondent aux souhaits du client. Un ordre de livraison de la payload est ensuite émis par le service de contrôle et exécuté par le système de guidage.
- Une fois livrée, la payload elle-même envoie des données télémétriques concernant ses paramètres orbitaux et son état.

ARCHITECTURE COMPLÈTE :

Diagramme d'architecture :

[draw.io link](#)



DÉTAILS DES SERVICES :

Dans cette section, nous allons exposer les micro-services que nous avons développés au cours du projet. Chaque description sera structurée en quatre parties distinctes :

- **Rôle** : L'objectif fondamental du micro-service.
- **Utilité** : Les raisons qui ont motivé la création de ce micro-service pour répondre à un/des cas d'utilisation(s) particulier(s).
- **Données manipulées (Interfaces REST + Model de données)** : Les opérations sur les données circulant à travers chaque micro-service sont clairement définies.
- **Communications par bus** : Pour les micro-services impliqués dans l'émission ou la réception de messages via un bus de communication, les bus spécifiques utilisés et leurs objectifs respectifs sont identifiés.

Mission Service :

Rôle :

Le rôle de ce microservice est d'organiser et de gérer les missions qui concernent le lancement des fusées dans l'espace orbital. Il assure également le suivi des événements survenant au cours de la mission. Une mission commence par la procédure de lancement et se termine par la livraison. Elle possède un attribut qui indique son état qui peut être "NOT_STARTED", "ON_GOING", "SUCCESSFUL", "FAILED", "ABORTED".

La progression du booster fait également partie de l'état de la mission et est conservée dans l'attribut booster state qui peut être "ATTACHED", "DETACHED", "IS_LANDING", "LANDED".

Une mission possède également une date et un nom spécifique qui est un identifiant naturel. Ces attributs sont notamment utiles pour Richard car ils lui permettent d'analyser et surveiller les missions.

Utilité :

Le micro-service a été conçu de cette manière car nous voulions avoir un endroit où stocker et gérer les missions. En effet, même si une mission concerne toujours une seule fusée, nous ne pouvions pas stocker tous les détails de ces missions dans le micro-service relatif aux fusées, car l'ensemble des missions est un agrégat en soi aussi.

Nous avons également fait le choix de stocker les événements qui se déroulent, car Richard voudra analyser une mission, et plus particulièrement ses journaux afin de respecter les réglementations des autorités aérospatiales et pour les enquêtes de l'entreprise.

Interfaces REST :

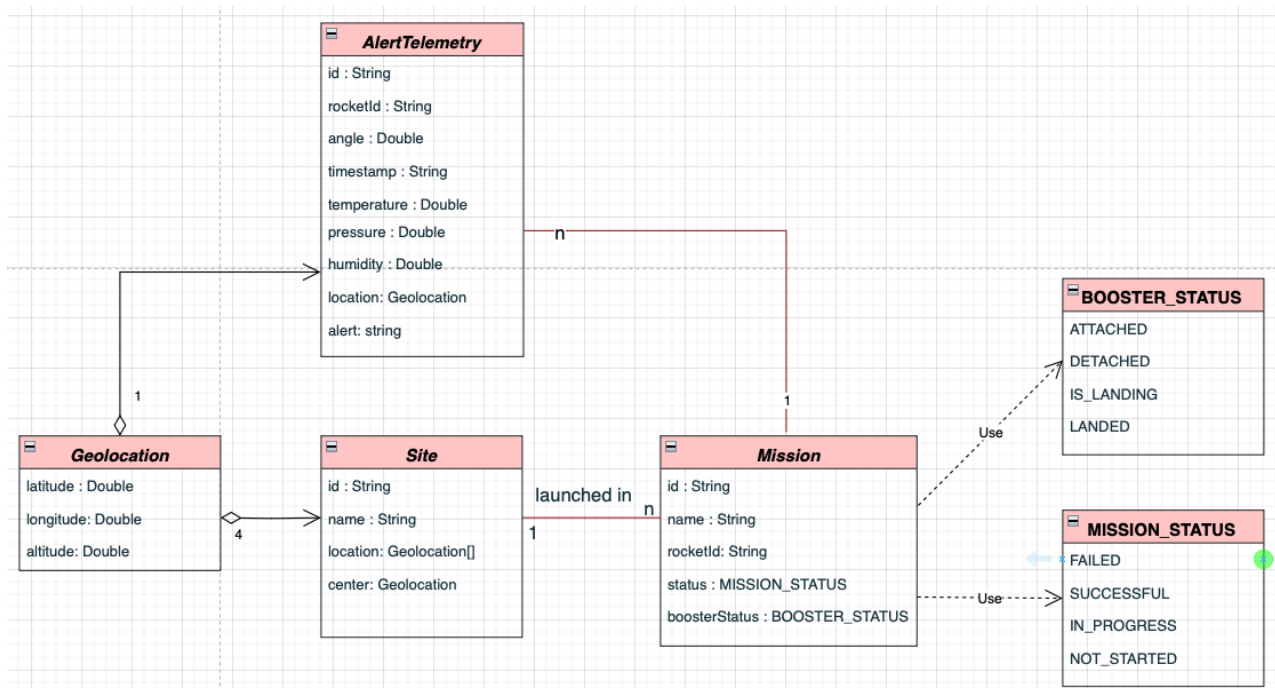
1. **POST /missions/:id/poll** : Lance un sondage pour déterminer si une mission peut être lancée.
2. **GET /missions** : Récupère toutes les missions.
3. **GET /missions/search** : Récupère des missions en fonction des paramètres de requête fournis (`rocketId` et `status`).
4. **GET /missions/:id** : Récupère une mission spécifique par son ID.
5. **POST /missions** : Crée une nouvelle mission
Body : {
 name: string,
 status: MissionStatus,
 site: string,
 rocket: string
}
6. **GET /missions/:id/logs** : Récupère les journaux associés à une mission spécifique.
7. **GET /sites** : Récupère tous les sites.
8. **POST /sites** : Crée un nouveau site

Body : {

```
    name: string;  
    center : Geolocation;  
    location : Geolocation[];  
}
```

9. **DELETE /sites/:id** : Supprime un site spécifique identifié par id.

Modèle de données relatif à ce micro-service :



COMMUNICATIONS PAR BUS

Ce microservice écoute les événements enregistrés dans les topics Kafka "mission-telemetry" et "mission-events". Il publie également des événements dans le topic "events-web-caster".

Les données télémétriques qui déclencheraient des politiques de mission et qui auraient pu être envoyées par n'importe quel hardware sont publiées par le service de télémétrie dans le topic "mission-telemetry". Les différents événements majeurs de la mission, tels que les 12 étapes de lancement, les 6 étapes d'atterrissage et les éventuelles perturbations, sont publiés dans le topic "mission-events" suivant le principe de l'événement sourcing et sont également utilisés pour mettre à jour l'état de la mission. Cela permet de conserver un snapshot de la progression de la mission dans le magasin d'événements.

Au fur et à mesure que les étapes de lancement se succèdent, elles sont publiées sous la forme d'un événement "Launch-Progressed" contenant les informations relatives à l'étape. De la même manière, les étapes d'atterrissage sont publiées au fur et à mesure en tant qu'événement "Landing-Progressed", contenant les informations relatives à l'étape d'atterrissage.

ControlPad Service :

Rôle :

Ce microservice est utilisé pour créer, gérer et contrôler les fusées pendant leur voyage. Il suit l'état et les coordonnées de la fusée grâce à la télémétrie qu'il reçoit et ajuste le statut de la fusée. La fusée peut avoir différents états : "PRELAUNCH_CHECK", "READY_FOR_LAUNCH", "IN_FLIGHT", "STAGED", "PAYLOAD_DELIVERED", "PAYLOAD_DELIVERY_FAILED", "DESTROYED".

Les fusées sont les entités centrales du service et forment un agrégat. Elles représentent ce que la société possède d'équipement aérospatial qu'elle peut utiliser pour livrer les payloads des clients en mission.

Ce microservice gère à la fois la création des fusées et leur contrôle pendant la mission. Il a semblé logique de garder les deux dans un seul Microservice plutôt que de diviser ces responsabilités sur deux Microservices car ils seraient extrêmement bavards, et l'état de la fusée est directement manipulé pendant son contrôle. Les commandes mettent à jour l'agrégat.

Utilité :

Ce microservice tient un registre des possessions aérospatiales de l'entreprise (les fusées). Il répond aux besoins d'Elon, le chef du département des fusées, qui souhaite garder un œil sur l'état de la fusée, même avant le lancement. Il lui permet de **déclencher la préparation** et d'émettre un ordre de **lancement** après le **vote par scrutin**, ainsi que d'autres ordres concernant la progression de la fusée dans les airs, tels que le **throttling** de la fusée pour sécuriser la **traversée de maxQ**, le **staging** et la **livraison de la payload**.

Interfaces REST :

10. **GET /rockets/:rocketId** : Permet de récupérer les détails d'une fusée spécifique identifiée par ":rocketId"

11. **POST /rockets** : permet de créer une fusée et l'enregistrer dans la base de données

Body : {

name: string,

status: string

}

12. **PUT /rockets/:rocketId/status** : permet de mettre à jour le status de la fusée

13. **POST /rockets/:rocketId/poll** : effectue un polling et renvoie true si l'état de la fusée permet un lancement.

14. **POST /rockets/:rocketId/launch** : permet d'effectuer le lancement de la fusée une fois que les conditions ont été vérifiées auprès du service mission.

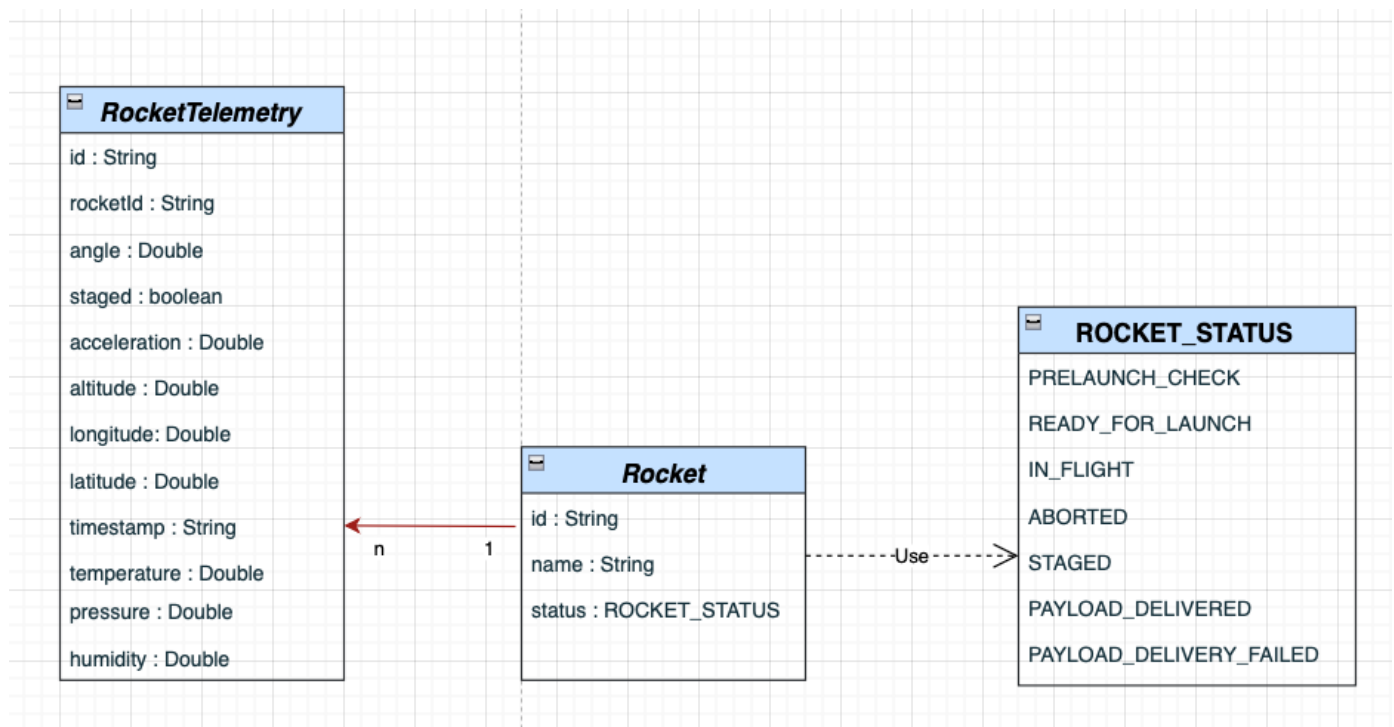
15. **POST /rockets/:rocketId/stage** : permet à Richard de faire le staging de la fusée en appelant le service du HardwareMock.

16. **POST /rockets/:rocketId/prepare** : Lance le processus de préparation pour la fusée spécifiée par ":rocketId".

17. **POST /rockets/:rocketId/powerOn** : Met sous tension la fusée spécifiée par ":rocketId".

18. **POST /rockets/:rocketId/payload-delivery** : appelée par le service de payload pour larguer le payload lorsque l'orbite souhaitée est atteinte.

Modèle de données relatif à ce micro-service :



Communications par bus :

Ce microservice écoute les événements enregistrés dans les topics Kafka "controlpad-telemetry" et "mission-events".

Tout d'abord, il consomme les messages envoyés dans 'controlpad-telemetry' qui sont des mesures de l'état et des coordonnées de la fusée provenant du service de télémétrie pour déclencher les politiques adéquates qui concernent la progression de la fusée pendant la mission, ces politiques aboutissent ensuite aux commandes mentionnées précédemment.

Étant donné que les commandes émises sont principalement "chronophages", telles que le staging, le throttling, le launch, etc., leurs réponses, qui proviennent du hardware lui-même et confirment leur occurrence, sont non bloquantes, c'est-à-dire asynchrones. Le hardware publie un tel événement naturel, dans la topic 'mission-events' pour que le service ControlPad mette à jour l'état de la fusée.

Weather Service :

Rôle :

Le rôle de ce microservice est assez direct. Il prend la géolocalisation qui lui est fournie et renvoie l'état de la météo. Il est également en mesure de déterminer si ces conditions météorologiques sont propices à un lancement.

Utilité :

Ce microservice est particulièrement utile à Tory, le météorologue. Il lui fournit directement les conditions météorologiques pour lui permettre de s'assurer de la capacité de la fusée à fonctionner si elle est lancée dans de telles circonstances.

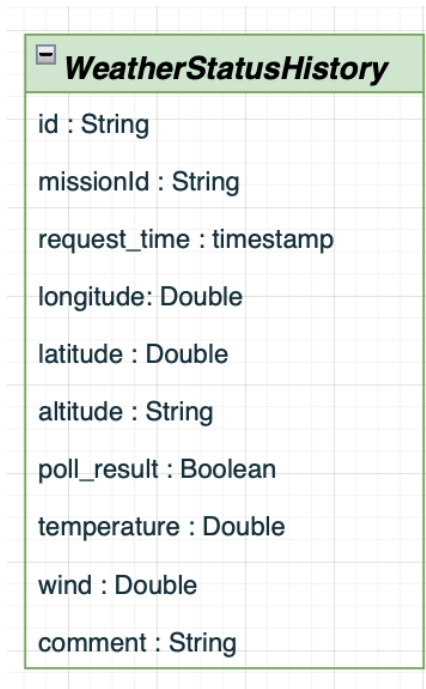
En effet, Tory sera sollicité par Richard, le chargé de mission, pour voter pour ou contre le lancement de la fusée.

Ce qui suit n'est pas mis en œuvre dans le code, mais il pourrait stocker ces conditions météorologiques lors des demandes de sondage avant de les renvoyer, afin de suivre les tendances des conditions de lancement qui pourraient favoriser ou non le voyage de la mission.

Interfaces REST :

19. **GET /weather/status?lat=&long=** permet de connaître le statut météorologique
20. **POST /weather/poll** utilise les paramètres de requête **lat** et **long** pour déterminer si le lancement est autorisé (go) ou non (no-go).

Modèle de données relatif à ce micro-service :



Payload Service :

Rôle :

Ce microservice est chargé de sauvegarder et de superviser les souhaits de nos clients en termes de livraison de payload. Il est utilisé pour aligner les résultats de la mission avec ce qui est attendu. Il enregistre les informations relatives aux payloads des clients qui sont attachées à une fusée spécifique à lancer. Il permet de suivre la progression du vol dans l'air jusqu'à ce que les paramètres orbitaux soient atteints et signalent la nécessité de larguer le satellite ou la sonde.

Utilité :

Ce microservice est directement destiné à être utilisé par Gwynn. En tant que chef du département des charges utiles qui doit suivre, stocker et consulter les données télémétriques des payloads, afin de pouvoir les utiliser ultérieurement pour certifier que les paramètres d'orbite souhaités par le client sont assurés.

Interfaces REST :

21. **PUT /payload/:rocketId/telemetry/delivery** : utilisé à l'origine pour mettre à jour l'état de la livraison

```
Body : {
    rocketId : string,
    delivered: boolean,
}
```

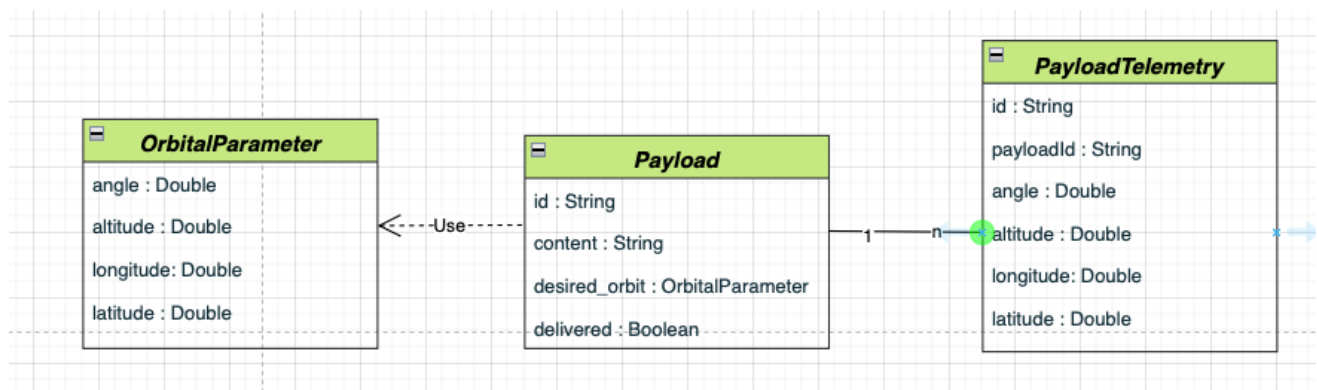
22. **POST /payload/:rocketId/telemetry** : utilisé à l'origine pour traiter et enregistrer de nouvelles données télémétriques sur les payloads.

```
Body : {
    rocketId : string,
    latitude : string,
    altitude : string,
    longitude : string,
    angle : string,
}
```

23. **POST /payload** : utilisé pour enregistrer la charge utile d'une fusée.

```
Body : {
    rocketId: string;
    content : string;
    desiredOrbit : OrbitalParameter;
    delivered : boolean;
}
```

Modèle de données relatif à ce micro-service :



Communications par bus :

Ce microservice écoute les événements publiés dans le topic Kafka "payload-telemetry".

Le service de Payload consomme des télémétries au fur et à mesure que le système de guidage progresse dans l'espace. Ces télémétries sont publiées sous forme de messages plutôt que des appels REST afin de ne pas le surcharger pendant qu'il effectue des vérifications. Les messages sont étiquetés "rocket" avant la livraison et "payload" après la livraison, la première étiquette étant utilisée pour déclencher la politique de livraison et la seconde pour suivre directement les orbites de la payload lorsqu'elle stabilise sa position sur l'orbite choisie.

BoosterControl Service :

Rôle :

Ce microservice est légèrement similaire au service du ControlPad. Nous avons choisi de séparer le contrôle du booster qui supervise le matériel du booster qui descend, du reste qui continue à monter vers l'espace orbital puisque les deux ont une logique et une séquence d'exécution disjointes

L'entité centrale de ce microservice est le booster, qui n'est qu'une sous-partie de la fusée elle-même et qui a donc le même identifiant. C'est notre agrégat racine.

Il gère le contrôle du booster pendant son atterrissage, puisqu'il émet la télémétrie nécessaire à cet effet, afin de lui ordonner de commencer le processus d'atterrissage et de déployer ses pattes lorsqu'il atteint l'altitude appropriée. Il met à jour l'état du booster au fur et à mesure qu'il passe par les 6 phases d'atterrissage.

Utilité :

Ce microservice a été créé non seulement pour démêler les responsabilités dans notre système, mais aussi pour permettre au département des fusées de contrôler cette partie distincte de la fusée qui passe par sa propre phase. Il peut être utilisé soit par Elon directement, soit par quelqu'un d'autre de son département. Quoi qu'il en soit, la nécessité de cette supervision est implicitement exprimée dans le cas d'utilisation du staging et dans l'autre cas d'utilisation qui détaille les différentes phases de la mission qui suivent.

Interfaces REST :

- 24. POST /booster/:rocketId/telemetry** : utilisé à l'origine pour traiter et enregistrer de nouvelles données télémétriques sur les boosters.

Body : {

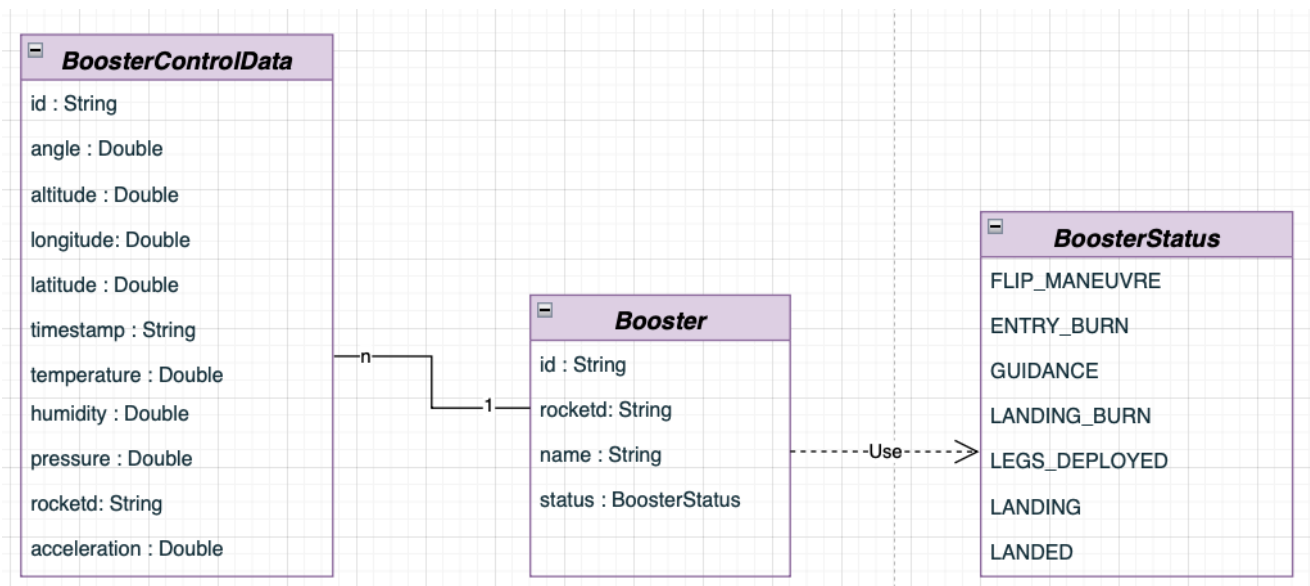
```
    rocketId : string,
    latitude : string,
    altitude : string,
    longitude : string,
    angle : string,
    temperature: number;
    pressure: number;
    acceleration: number;
    humidity: number;
}
```

- 25. POST /booster**: utilisé pour enregistrer le booster.

Body : {

```
    rocketId: string;
    name: string;
    status : BoosterStatus
}
```

Modèle de données relatif à ce micro-service :



Communication par bus :

Ce microservice écoute les événements publiés dans le topic Kafka "booster-telemetry".

Comme les autres microservices qui consomment les messages de télémétrie provenant du service de télémétrie, il est principalement asynchrone afin de ne pas submerger le service d'appels bloquants et de lui permettre de traiter l'évolution du processus d'atterrissage au fur et à mesure qu'il se produit. De même, la consommation de ces événements finira par déclencher la politique d'atterrissage.

Telemetry Service :

Rôle :

Comme son nom l'indique, il consomme la télémétrie provenant des hardwares, la stocke telle quelle, et la traduit en un modèle de lecture compréhensible par les autres services avant de la leur envoyer.

La mission de ce microservice est de surveiller et sauvegarder (activement et en continu) l'état des différents composants matériels. Il agit comme un intermédiaire entre les différents composants de la fusée contenant des capteurs et envoyant de la télémétrie, et les autres départements de l'organisation Mars Y souhaitant surveiller une partie particulière ou avoir une vue globale de l'état de la fusée.

D'une certaine manière, il s'agit de notre source de vérité brute puisqu'il stocke la télémétrie telle quelle et qu'elle s'agit de son agrégat unique.

Utilité :

Ce microservice est principalement utilisé par Jeff. Comme mentionné précédemment, Jeff souhaite recevoir et suivre la télémétrie de la fusée pendant toute sa séquence de lancement. L'objectif est de surveiller l'état de la fusée au fur et à mesure que le lancement progresse, et de signaler les comportements bizarres particuliers au service de mission ainsi que de garder une trace de leur source.

Il n'agit pas en cas de détection, il signale seulement ce qui lui semble incohérent.

En effet, ce microservice reçoit des données télémétriques de trois sources différentes. Nous pensons qu'il aurait été possible d'améliorer ce point en répartissant d'autres nouveaux services, ce qui aurait permis

d'avoir un service de surveillance/sauvegarde de l'état par composant matériel, permettant ainsi une meilleure séparation de la logique de surveillance si cette logique évolue avec le temps dans l'entreprise.

Pour l'instant, le seul contrôle effectué par ce service est celui de la phase de propulsion, le reste des anomalies étant soit détecté automatiquement par le matériel lui-même, soit par le pupitre de contrôle.

Interfaces REST :

26. **GET /telemetry/missionId** : permet de récupérer les enregistrements de télémétrie pour une mission spécifique

27. **POST /telemetry/payload** : utilisé à l'origine pour créer de nouvelles données télémétriques sur les payload

Body : {

```
        rocketId : string,
        latitude : string,
altitude : string,
longitude : string,
angle : string,
    }
```

28. **POST /telemetry/booster** : utilisé à l'origine pour créer de nouvelles données télémétriques sur les boosters

Body : {

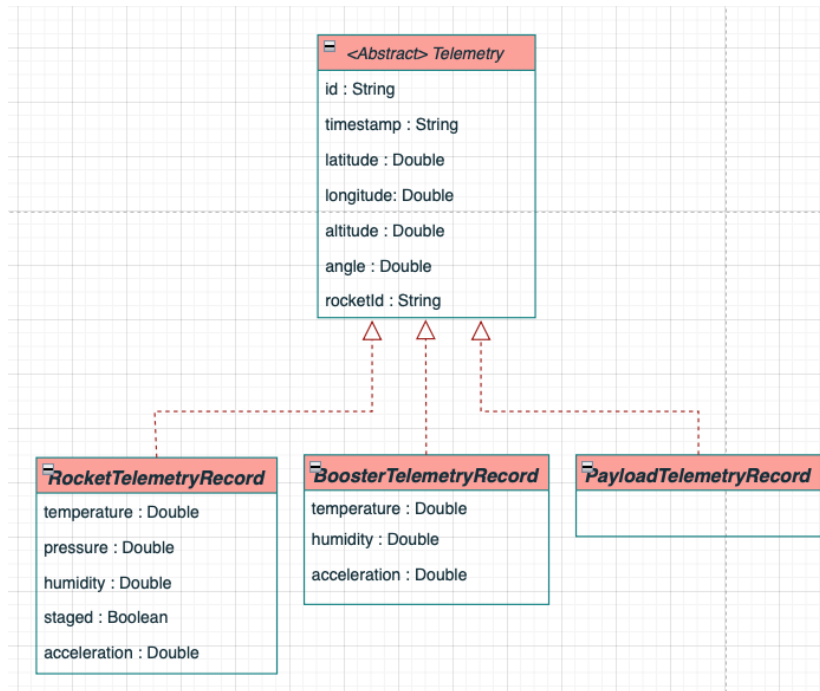
```
        rocketId : string,
        latitude : string,
altitude : string,
longitude : string,
angle : double,
temperature : double,
humidity : double,
acceleration : double
    }
```

29. **POST /telemetry** : utilisé à l'origine pour créer de nouvelles données télémétriques sur les rockets

Body : {

```
        rocketId : string,
        latitude : string,
altitude : string,
longitude : string,
angle : double,
temperature : double,
humidity : double,
acceleration : double,
        stage : boolean,
pressure : double
    }
```

Modèle de données relatif à ce micro-service :



Communications par bus :

Ce microservice écoute et consomme les événements publiés dans le topic Kafka "telemetry". Il publie également des événements dans les topics "payload-telemetry", "controlpad-telemetry" et "mission-telemetry".

Il consomme toutes les télémetries publiées dans le topic "telemetry", étant donné que toutes les télémetries provenant d'un matériel spécifique sont sous forme de messages étiquetés avec le type de ce matériel : 'rocket' pour signaler la télémétrie provenant du propulseur et du système de guidage plus tard, représentant la partie de la fusée qui monte, 'booster' pour celle provenant du booster qui descend, 'payload' pour celle émise par la charge utile elle-même une fois qu'elle est déposée.

Il est clair que ce service consomme beaucoup d'informations qui, si elles avaient été fournies de manière synchrone, l'auraient complètement détruit, ce qui justifie le passage à la communication asynchrone.

D'autre part, il traduit les données brutes en modèles de lecture pour les autres et les publie dans le sujet approprié qu'ils écoutent. En ce qui concerne les informations qui intéressent le service du ControlPad, il les publie dans "controlpad-telemetry". Il s'agit d'informations sur l'état de la fusée (propulsant + système de guidage) elle-même, y compris des paramètres tels que l'accélération et les niveaux de carburant, ainsi que ses coordonnées orbitales au fur et à mesure qu'elle progresse dans l'espace.

Pour ceux qui s'intéressent au service de Payload, il les publie dans "payload-telemetry", qui sont les coordonnées orbitales de la payload. Avant la livraison, il s'agit des mêmes paramètres que le corps de la fusée qui monte, plus tard, lorsque la charge utile est déployée, il s'agit des paramètres émis par les capteurs de la payload eux-mêmes.

Enfin, la télémétrie qui pourrait concerner la mission est publiée dans 'mission-telemetry' et ce sont toutes les données contrôlées par et ce sont toutes les données contrôlées par ce service de télémétrie qui ont été signalées car elles sembleraient suspectes, alertant ainsi le service de mission ET lui fournissant l'information

qui a déclenché cette alerte, lui permettant ainsi de décider de déclencher ou non une politique de destruction.

Web caster Service :

Rôle :

Ce microservice est chargé de réarticuler les messages consommés à partir du topic "events-web-caster", avec pour objectif d'afficher les événements de manière plus compréhensible.

Utilité :

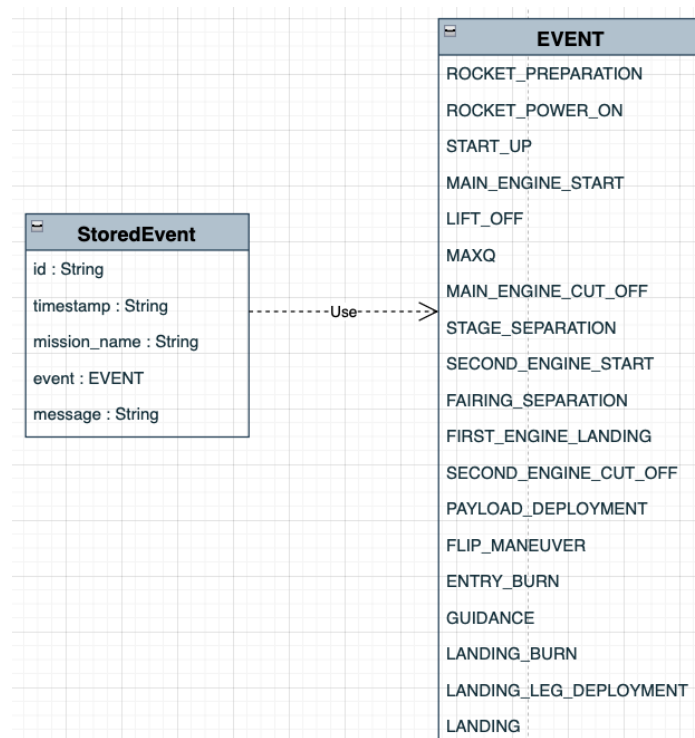
Ce micro-service répond au cas d'utilisation de Marie, car il lui permettra de parler des événements de la mission en temps réel dans le flux web. Il rend ces événements intéressants dans un langage lisible par l'homme. Pour l'instant, ils sont affichés dans la console du service sous forme des logs.

Nous n'avons pas implémenté ce qui suit, mais nous pensons qu'il serait préférable de publier ces événements avec leurs textes lisibles par l'homme dans un topic dédié, afin qu'ils puissent être consommés par d'autres médias à l'avenir !

Interfaces REST :

30. **GET /events** : permet de renvoyer la liste d'événements diffuser par webcaster

Modèle de données relatif à ce micro-service :



Communications par bus :

Ce microservice écoute les événements publiés dans le topic Kafka "events-web-caster ". Il consomme les évènements de la mission tel quels et il les affiche sous une forme textuelle plus explicite.

HardwareMock Service :

Rôle :

Ce micro-service a pour rôle de simuler la partie inférieure de la fusée représentant le propergol dans la première phase et devenant le booster dans la seconde phase après s'être détaché du reste de la fusée. Lorsqu'il simule le propulseur, il traite toutes les commandes reçues du service du pavé de commande et renvoie le résultat de manière asynchrone. Plus tard, lorsqu'il simule le propulseur, il traite les commandes reçues du service de contrôle du propulseur et renvoie le résultat de manière asynchrone.

Il détecte les anomalies graves à partir des données de ses propres capteurs et s'autodétruit s'il les juge graves afin d'éviter de causer des problèmes et de tomber sur des civils.

Il génère des données télémétriques toutes les 3 secondes, c'est-à-dire les données recueillies des capteurs.

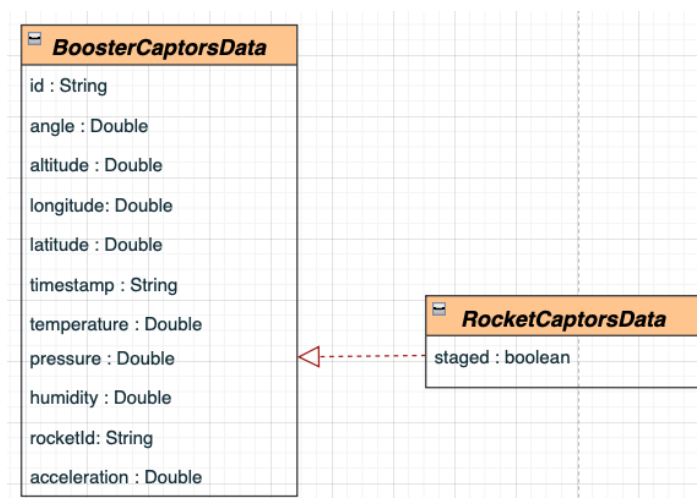
Utilité :

L'utilité de ce microservice s'étend à toutes les histoires d'utilisateurs sollicités, car il joue le rôle d'une partie de la fusée elle-même et répond à toutes les commandes qu'il reçoit. Il informe périodiquement le service de télémétrie de son état en lui fournissant les données qu'il a obtenues sur la fusée. Cela permet aux superviseurs concernés de suivre l'évolution de la situation et de réagir en conséquence.

Interfaces REST :

31. **POST /mock/:idrocket/land** : permet de déclencher le processus d'atterrissage du booster d'une fusée
32. **POST /mock/:idrocket/prepare**: Initialise la préparation de la fusée avec l'identifiant spécifié.
33. **POST /mock/:idrocket/power-on** : Active la mise sous tension interne de la fusée avec l'identifiant spécifié.
34. **POST /mock/:idrocket/throttle-down**: Initialise la régulation des gaz de la fusée avec l'identifiant spécifié
35. **POST /mock/:idrocket/launch**: permet de faire le lancement de la fusée et de la génération automatique des télémétries.
36. **POST /mock/:idrocket/stage** : permet de faire le staging de la fusée

Modèle de données relatif à ce micro-service :



Communications par bus :

Ce microservice publie des événements dans les topics Kafka "telemetry" et "mission-events".

Dans le topic "télémétrie", il publie les télémétries qu'il a recueillies auprès des capteurs matériels. Au lieu d'appeler le service de télémétrie, il les publie dans le broker de messages pour ne pas le surcharger. Ces télémétries sont étiquetées 'rocket'

Lorsqu'il exécute une commande, il génère un événement et le place dans le topic "mission-events" pour que les autres services puissent l'utiliser.

GuidanceHardwareMock Service :

Rôle :

Comme le précédent, le rôle de ce microservice est de simuler la partie centrale de la fusée, représentant le système de guidage dans la deuxième phase, après que le propulseur s'est détaché du reste de la fusée.

Lors de la simulation du système de guidage, il traite toutes les commandes reçues du service ControlPad et renvoie le résultat de manière asynchrone, notamment l'ordre de larguer la charge utile.

Il génère des données télémétriques toutes les 3 secondes, c'est-à-dire des données collectées par les capteurs. Il ne commence à le faire qu'après avoir été activé, ce qui signifie qu'il ne le ferait qu'après le staging.

Utilité :

Même utilité que le service précédent.

Interfaces REST :

- 37. **POST /:idrocket/deliver** : permet de déclencher le processus de livraison de la charge
- 38. **POST /:idrocket/launch** : utilisé à l'origine pour déclencher l'envoi de la télémétrie du système de guidage

Modèle de données relatif à ce micro-service :

GuidanceCaptorsData
id : String
angle : Double
altitude : Double
longitude: Double
latitude : Double
timestamp : String
temperature : Double
pressure : Double
humidity : Double
rocketId: String
acceleration : Double

Communications par bus :

Ce microservice publie également des événements dans les topics Kafka "telemetry" et "mission-events". Il suit la même approche que le précédent, la seule différence étant que ses messages sont étiquetés "guidance".

PayloadHardwareMock Service :

Rôle :

Légèrement similaire aux précédents, le rôle de ce microservice est de simuler la partie supérieure de la fusée, représentant la payload après qu'elle ait été délivrée par le système de guidage. En simulant la payload, il traite toutes les commandes reçues du service de pilotage (que nous avons ajouté dans nos nouvelles user stories et qui seront expliquées prochainement) et ajuste ses paramètres orbitaux.

De même, il génère des données télémétriques toutes les 3 secondes, c'est-à-dire des données collectées par les capteurs. Il ne commence à le faire qu'après avoir été activé, ce qui signifie qu'il ne le fera qu'après la livraison. Il cesse d'envoyer cette télémétrie lorsqu'il se stabilise sur son orbite.

Plus tard, il génère les informations de diffusion qui seront consommées par le service de diffusion (également ajoutées dans les nouvelles histoires d'utilisateurs).

Utilite :

Même utilité que le service précédent.

Interfaces REST :

39. **POST /broadcast/:rocketId** : permet d'envoyer les détails de lancement du satellite à notre service de diffusion

40. **POST /orient/:rocketId** :

Body : {

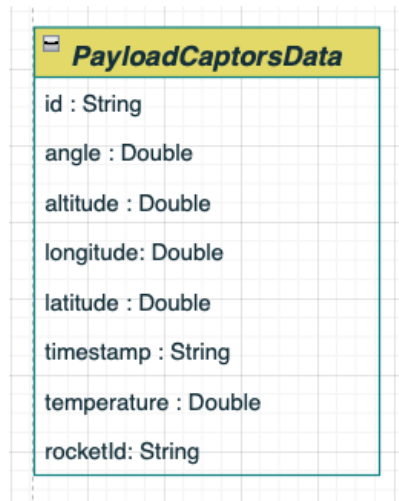
```
rocketId: number;  
latitude: number;  
longitude: number;  
speed: number;  
direction: string;
```

}

Permet d'orienter une fusée spécifiée selon les coordonnées, la vitesse et la direction spécifiées.

41. **POST /payload/launch** : utilisé à des fins initiales pour que la payload commence à envoyer des données télémétriques après sa livraison.

Modèle de données relatif à ce micro-service :



Communications par bus :

Ce microservice publie des événements dans les topics Kafka "telemetry" et "broadcast-service". Il suit la même approche que le précédent, la seule différence étant que ses messages sont étiquetés "payload".

Pour la diffusion de l'émission du client, il publie dans le topic "broadcast-service", fournissant ainsi des informations détaillées sur le déplacement du satellite.

Client Service :

Rôle :

Ce microservice, en cas de succès du processus de livraison, communique avec le service payload hardware pour déclencher son démarrage étant donné que c'est un satellite. Il est notifié par le service de télévision pour une début diffusion. En cas de coupure de la diffusion, ce microservice est également informé afin de prendre des mesures correctives et de contacter le service de pilotage (pilot service) pour mettre en œuvre les mesures nécessaires.

Utilité :

Une fois la livraison du payload assurée avec succès, le client se concentre immédiatement sur son objectif principal : le démarrage du satellite. En effet, le client, ayant assuré la livraison réussie de la charge utile, prend alors activement part au processus suivant, collaborant avec le service payload hardware pour garantir le démarrage efficace satellite correspondant. Ce changement d'orientation reflète une transition entre les phases de livraison et de lancement, mettant en lumière la fluidité de l'intégration des différentes étapes de la mission.

Interfaces REST :

Le microservice ne propose actuellement aucune API. L'absence d'API est justifiée par sa nature de coordination interne avec le service broadcast et payload service, sans nécessité directe d'interaction avec d'autre service.

Communications par bus :

Ce microservice fonctionne en écoutant et en consommant activement les événements publiés dans le topic Kafka "client-service-events". Il réagit à quatre événements distincts :

- **Événement DELIVERED :**

Cet événement est déclenché lorsqu'un payload est livré avec succès, signalant ainsi l'accomplissement de la phase de livraison.

- **Événement BROADCASTING DISTURBED**

Cet événement indique toute perturbation ou interruption dans le processus de diffusion, alertant le microservice pour prendre des mesures correctives.

- **Événement BROADCASTING RESUMED:**

Cet événement signale que la diffusion a été reprise après l'interruption.

- **Événements BROADCASTING TERMINATED et BROADCASTING STARTED:**

Ces événements signalent respectivement le début et la fin d'une diffusion, permettant au microservice de synchroniser ses actions en conséquence.

Broadcast Service :

Rôle :

Le microservice de diffusion facilite la réception et la diffusion des détails du lancement du satellite, y compris son déplacement. En cas de problème avec les détails reçus, il informe également le service client pour une prise en charge rapide.


Utilité :

Le client de la mission profite de ce microservice pour annoncer le lancement du satellite au public et partager les événements clés avec un auditoire plus vaste, renforçant ainsi l'aspect public et médiatique de la mission spatiale.

Interfaces REST :

Ce microservice actuel ne propose aucune API REST, car il se concentre principalement sur sa fonction de service de diffusion. Une évolution envisageable serait d'exposer des API, offrant ainsi la possibilité à ceux qui le souhaitent de s'inscrire pour recevoir des détails spécifiques.

Modèle de données relatif à ce micro-service :

 SatelliteDetails
id : String
angle : Double
altitude : Double
longitude: Double
latitude : Double
timestamp : String
temperature : Double
rocketId: String

Communications par bus :

Ce microservice publie des événements dans le topic Kafka "client-service-events" afin d'informer sur l'état de la diffusion (**BROADCASTING STARTED, BROADCASTING TERMINATED ,BROADCASTING DISTURBED, BROADCASTING RESUMED**). Il écoute également les événements publiées dans le topic 'broadcast-service', permettant ainsi l'acquisition d'informations détaillées sur le déplacement du satellite.

Pilot Service :

Rôle :

Le microservice "pilot" a pour objectif de permettre au client et à son équipe de prendre le contrôle opérationnel du satellite. Concrètement, il offre la possibilité de piloter le satellite en modifiant ses paramètres orbitaux, tels que sa direction.

Utilité :

En cas de problème ou de nécessité d'intervention, ce microservice offre au client et à son équipe la possibilité de prendre le contrôle du satellite et d'ajuster ses paramètres orbitaux.

Interfaces REST :

42. **POST /takeControl/:rocketId** : permet de déclencher la réorientation du satellite pour une fusée spécifique

Communications par bus :

L'absence de communication événementielle est justifiée par la nécessité d'une réponse instantanée, essentielle pour garantir un contrôle immédiat et sûr du pilotage.

SCÉNARIOS

Lors de l'exécution du script `./run.sh`, trois scénarios successifs seront mis en œuvre. Le premier scénario décrit une exécution sans accroc, où tout se déroule conformément aux prévisions. Dans le deuxième scénario, la fusée détecte une anomalie et s'auto-détruit. Enfin, le troisième scénario présente le lancement d'une deuxième fusée avec un microservice qui rencontre des problèmes.

Scénario 1 : Mission réussite

Pendant ce scénario, nous illustrons le déroulement d'une mission lorsqu'elle se déroule conformément aux attentes. Voici une explication détaillée des événements qui se produisent dans les logs :

- 1- **Setup** : Le service "Mission" reçoit une demande d'ajout d'un site et d'une mission dans la base de données.
- 2- **Initiation de la séquence de lancement** : Le service "ControlPad" reçoit une demande d'initiation de la séquence de lancement pour la fusée.
- 3- **Vérification des conditions météorologiques et de l'état de la fusée** : Le service "MissionService" communique avec le service météorologique ("Weather service") et "Controlpad" pour effectuer les vérifications nécessaires avant le lancement.
- 4- **Annonces de WebCasterService** : Le service "WebCasterService" annonce différentes étapes du processus de lancement (préparation, mise sous tension interne, début du lancement, démarrage du moteur principal, etc.).
- 5- **Démarrage du lancement et de l'envoi de la télémétrie** : Le service "HardwareService" commence à envoyer la télémétrie de la fusée.
- 6- **Évaluation de la télémétrie** : Le service "TelemetryService" évalue la télémétrie pour s'assurer que la fusée fonctionne dans des paramètres sécuritaires.
- 7- **Vérification de l'altitude et du carburant** : Le service "ControlPad" vérifie l'altitude et le niveau de carburant de la fusée à intervalles réguliers.
- 8- **Staging** : La séquence de lancement atteint le point de "staging", où certains composants de la fusée sont séparés. À cette nouvelle étape, le service "GuidanceHardware" sera responsable de l'envoi des données de télémétrie du deuxième compartiment, tandis que "HardwareService" sera responsable de l'envoi des télémétries du booster, supervisées par le service "boosterControl".
- 9- **Payload Delivery** : Une fois en orbite, le service "PayloadService" annonce que la fusée a atteint une altitude spécifique et envoie des détails sur la position. Le "ControlPad" envoie la commande de la livraison. Une fois le payload livré, on commence à recevoir sa télémétrie.
- 10- **Notifications** : Des notifications seront envoyées vers le "ControlPad" et les clients pour les informer de la livraison du payload.
- 11- **Lancement du satellite** : Le service client communique avec le service "payload hardware" pour déclencher le lancement du satellite.
- 12- **Broadcasting** : Le service "Payload Hardware" envoie les détails du lancement et du déplacement du satellite au service de diffusion ("Broadcast Service"). Cette dernière analyse et affiche ces détails.
- 13- **Contrôle du satellite par l'équipe de pilotage** : Le "Broadcast Service" reçoit des informations dégradées, entraînant ainsi un contact avec le service client, qui délègue ensuite le contrôle au service de pilotage.
- 14- **Fin de la mission** : On arrête d'envoyer la télémétrie du payload.
- 15- **Historique de la mission** : À la fin de la mission, les différents événements survenus sont affichés, accompagnés de leur date et heure exactes.

Logs associés :

```
LOG [SiteService] Received request to add site name : testSite 1
LOG [MissionService] Received request to add mission name : testMission
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14) 2
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [WebCasterService] News from 278 Just in : the rocket is being prepared (us 15) 4
LOG [WebCasterService] News from 278 Just in : the rocket internal power is on (us 15)
LOG [ControlPadService] Initiating launch sequence for rocket 278.
LOG [MissionService] Received request to perform a go/no go for mission A1B
LOG [MissionService] checking rocket 278 status
LOG [MissionService] weather status OK for rocket 278
LOG [MissionService] checking rocket 278 status 3
LOG [ControlPadService] Monitor rocket status (us 2)
LOG [ControlPadService] All checks passed for rocket 278. starting launch. (us 4)
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [HardwareService] started sending telemetry for the rocket 278 (us 5) 5
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [WebCasterService] News from 278 Just in : the rocket start up T-00:01:00 (us 15) 4
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [WebCasterService] News from 278 Just in : the rocket main engine start T-00:00:03 (us 15) 4
LOG [WebCasterService] News from 278 Just in : the rocket is launching T-00:00:00 (us 15)

LOG [TelemetryService] Evaluating telemetry for rocket: 278
LOG [TelemetryService] Telemetry for rocket 278 is within safe parameters. No need for destruction. 6
LOG [ControlPadService] Checking if approaching MaxQ for rocket 278 - Altitude: 4066 meters.
DEBUG [ControlPadService] Approaching MaxQ for rocket 278
DEBUG [ControlPadService] Throttling down engines for rocket 278
DEBUG [HardwareService] Rebooted : Resending telemetry
LOG [MarsyMockHardwareProxyService] Request to start throttling down engines for rocket : 278 7
DEBUG [ControlPadService] Reached MaxQ for rocket 278
DEBUG [ControlPadService] Throttling up engines for rocket 278
LOG [ControlPadService] checking fuel level for rocket 278 - Fuel: 90 liters.
LOG [HardwareService] Throttling down the rocket 278

LOG [ControlPadService] checking fuel level for rocket 278 - Fuel: 0 liters. 8
LOG [ControlPadService] issuing fuel depletion mid-flight for rocket 278
LOG [ControlPadService] staging mid-flight for rocket 278 (us 6)
LOG [MarsyMockHardwareProxyService] Request to start performing staging for rocket: 278
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [GuidanceHardwareService] Started guidance sending telemetry for the rocket 278
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [WebCasterService] News from 278 Just in : the rocket stage separation (us 15)
LOG [TelemetryService] Retrieving telemetry from the booster of the staged rocket 278 (us 10)
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [WebCasterService] News from 278 Just in : the rocket main engine cutoff (us 15)
LOG [WebCasterService] News from 278 Just in : the rocket second engine start (us 15)
LOG [GuidanceHardwareService] Sending telemetry from the hardware of 278
LOG [TelemetryService] Retrieving telemetry from the booster of the staged rocket 278 (us 10)

LOG [PayloadService] Orbit reached for 278 - altitude: 9900 - latitude: 282 - longitude: 78 - angle: 86 9
LOG [TelemetryService] Retrieving telemetry from the booster of the staged rocket 278 (us 10)
LOG [PayloadService] Event sent to inform the client service about the payload delivery of rocket ID 278
LOG [MarsyLaunchpadProxyService] Notifying command pad of rocket 278 10
LOG [ClientServiceProxy] Requesting launch updates 11
LOG [ControlPadService] Sending payload delivery command for rocket 278 (us 7)
LOG [MarsyGuidanceHardwareProxyService] Requesting payload delivery for rocket 278 (us 7)
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [PayloadHardwareService] Started sending satellite details of rocket 278 to broadcast service
LOG [WebCasterService] News from 278 Rocket 278 is staged. Initiating payload delivery. (us 15)
LOG [GuidanceHardwareService] Delivering the payload on the rocket 278 (us 7)
LOG [MissionService] checking rocket 278 status
LOG [MissionService] Saving event for mission 278 (us 14)
LOG [MarsyLaunchpadProxyService] Command pad notified of reaching orbital reach
LOG [WebCasterService] News from 278 Payload delivered successfully for rocket 278 (us 15)
LOG [TelemetryService] Retrieving telemetry from the booster of the staged rocket 278 (us 10) 12
LOG [BroadcastService] start broadcasting
LOG [BroadcastService] New message received with satellite details of rocket 278 (us 19)
LOG [BroadcastService] - Latitude: 57.793971151450364
LOG [BroadcastService] - Longitude: -69.61981790717628
LOG [BroadcastService] - Speed: 4519.229645778894
LOG [BroadcastService] - Direction: south
```

```

LOG [BroadcastService] New message received with satellite details of rocket 278 (us 19)
LOG [BroadcastService] - Latitude: undefined
LOG [BroadcastService] - Longitude: undefined
LOG [BroadcastService] - Speed: undefined
LOG [BroadcastService] - Direction: undefined
LOG [BroadcastService] broadcasting disturbed
LOG [PilotServiceProxy] requesting pilot details for rocket 278 (us 20)
LOG [PayloadHardwareService] Adjusting satellite positioning for rocket with ID 278 and transmitting details
LOG [PayloadHardwareService] adjustment of satellite of rocket with id 278 sent to broadcast service
LOG [BroadcastService] broadcasting resumed of rocket 278:
LOG [BroadcastService] New message received with satellite details of rocket 278 (us 19)
LOG [BroadcastService] - Latitude: 52.3747462768373
LOG [BroadcastService] - Longitude: -26.106740115816365
LOG [BroadcastService] - Speed: 1324.0026006123849
LOG [BroadcastService] - Direction: south
LOG [PayloadHardwareService] Satellite details of rocket with id 278 sent to broadcast service
LOG [BroadcastService] broadcasting terminated
LOG [PayloadHardwareService] Retrieved telemetry for mission A1B
LOG [PayloadHardwareService] STOPPING SENDING TELEMETRY PAYLOAD - MISSION SUCCESSFUL

```

13

14

```

LOG [MissionService] ALL EVENTS STORED FOR MISSION 6C3
DEBUG [MissionService] 1- 18 nov. 2023, 21:30:17 UTC+1 => Just in : the rocket is being prepared
DEBUG [MissionService] 2- 18 nov. 2023, 21:30:17 UTC+1 => Just in : the rocket internal power is on
DEBUG [MissionService] 3- 18 nov. 2023, 21:30:17 UTC+1 => Just in : the rocket start up T-00:01:00
DEBUG [MissionService] 4- 18 nov. 2023, 21:30:17 UTC+1 => Just in : the rocket main engine start T-00:00:03
DEBUG [MissionService] 5- 18 nov. 2023, 21:30:17 UTC+1 => Just in : the rocket is launching T-00:00:00
DEBUG [MissionService] 6- 18 nov. 2023, 21:30:26 UTC+1 => Just in : the rocket is at maximum dynamic pressure
DEBUG [MissionService] 7- 18 nov. 2023, 21:30:30 UTC+1 => Just in : the rocket stage separation
DEBUG [MissionService] 8- 18 nov. 2023, 21:30:30 UTC+1 => Just in : the rocket main engine cutoff
DEBUG [MissionService] 9- 18 nov. 2023, 21:30:30 UTC+1 => Just in : the rocket second engine start
DEBUG [MissionService] 10- 18 nov. 2023, 21:30:39 UTC+1 => Rocket 6C3 is staged. Initiating payload delivery.
DEBUG [MissionService] 11- 18 nov. 2023, 21:30:39 UTC+1 => Payload delivered successfully for rocket 6C3
DEBUG [MissionService] 12- 18 nov. 2023, 21:30:42 UTC+1 => Just in : the rocket is performing a flip maneuver
DEBUG [MissionService] 13- 18 nov. 2023, 21:30:42 UTC+1 => Just in : the rocket deployed its payload
DEBUG [MissionService] 14- 18 nov. 2023, 21:30:43 UTC+1 => Just in : the rocket is performing an entry burn
DEBUG [MissionService] 15- 18 nov. 2023, 21:30:44 UTC+1 => Just in : the rocket is guiding itself to the landing pad
DEBUG [MissionService] 16- 18 nov. 2023, 21:30:45 UTC+1 => Just in : the rocket is performing a landing burn
DEBUG [MissionService] 17- 18 nov. 2023, 21:30:46 UTC+1 => Just in : the rocket is deploying its landing legs
DEBUG [MissionService] 18- 18 nov. 2023, 21:30:47 UTC+1 => Just in : the rocket landed

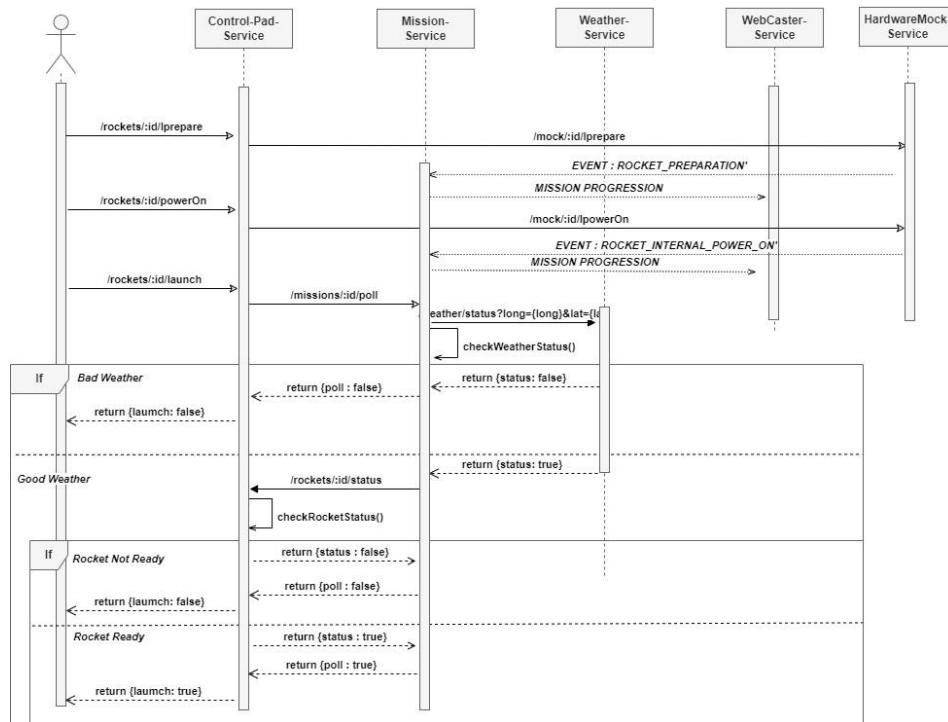
```

Diagrammes de séquence associés :

[Vous pouvez accéder à tous les diagrammes de séquences via le lien](#)

Verification des conditions avant lancement

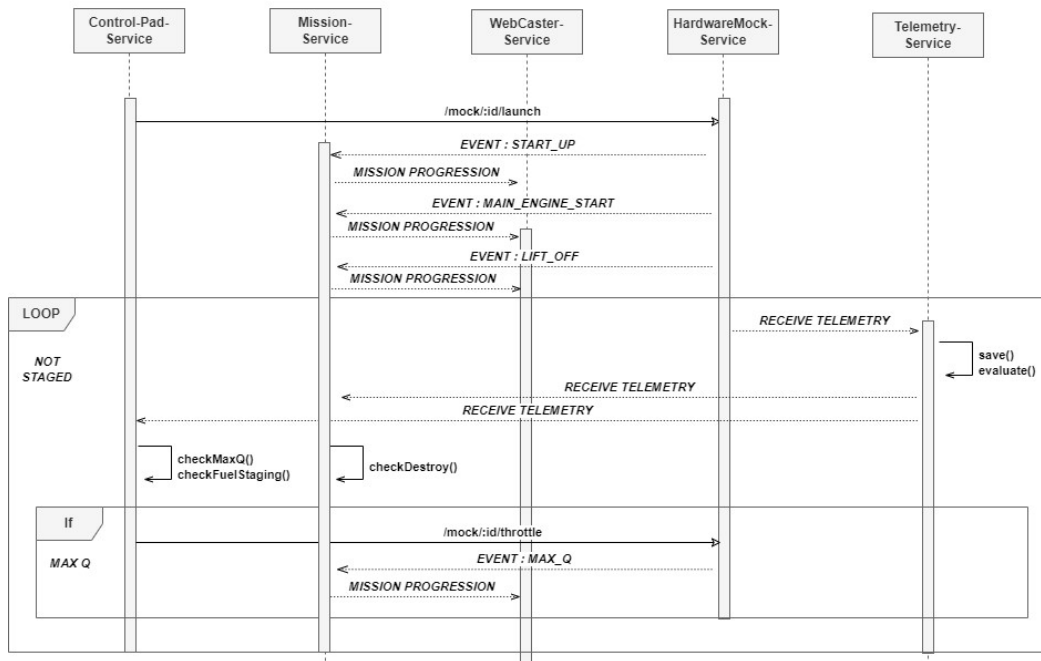
ETAPE 2+3



- Au début de la mission, deux appels REST (**POST /rockets/:id/prepare** et **POST /rockets/:id/powerOn**) sont effectués vers le service ControlPad pour la préparation de la fusée. ControlPad lance ensuite deux appels REST **POST/mock/:id/prepare** et **POST/mock/:id/powerOn** afin de préparer le hardware et de le mettre en marche. Deux événements, *ROCKET_PREPARATION* et *ROCKET_INTERNAL_POWER_ON*, sont générés à la fin de cette étape. Ils sont consommés par la mission à travers *topic-mission-events* et envoyés vers le WebCaster à travers le *topic events-web-caster*.
- Pour le lancement, un appel REST **POST /rockets/:id/launch** est effectué, suivi par l'appel de la mission avec **POST /missions/:id/poll**. La mission vérifie de manière asynchrone les conditions météorologiques avec **POST /weather/status?long={long}&lat={lat}**. Si les conditions ne sont pas favorables, la mission échoue. Sinon, l'état de la fusée est vérifié avec **GET /rockets/:id/status**. Si elle est prête, la mission peut commencer.

Lancement de la fusée

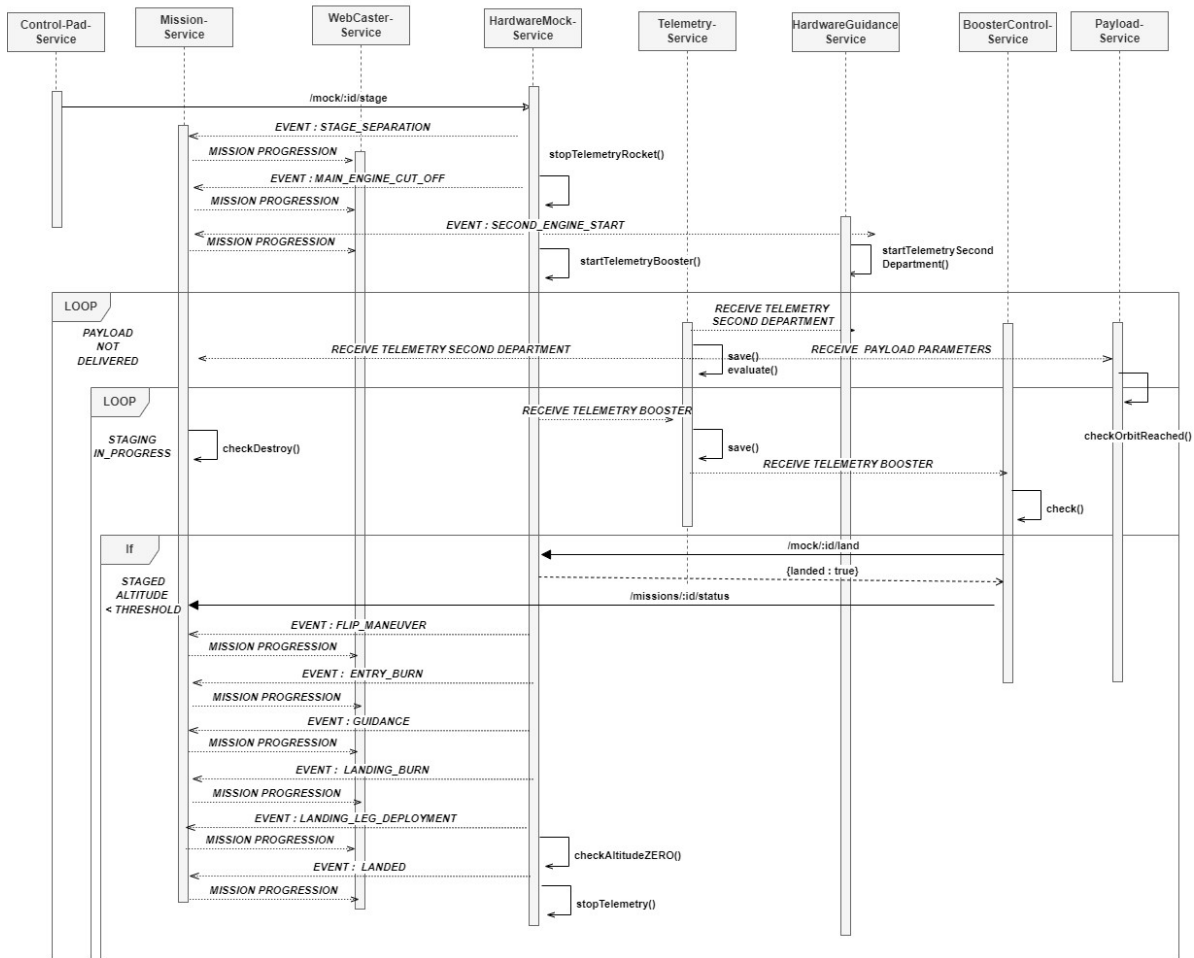
ETAPE 5+6+7



- À la fin de l'étape précédente, le hardware effectue le démarrage du moteur et procède au lancement, notifiant la mission des différents états : *START_UP*, *MAIN_ENGINE_START*, et *EVENT_LIFT_OFF* à travers *topic-mission-events*
- Ensuite, le hardware commence à générer les télémétries et les envoie vers le service de télémétrie via le topic *telemetry*. Ce service évalue les télémétries, les enregistre, puis les transmet à la mission à travers le topic *mission-telemetry*, qui effectue des vérifications pour s'assurer la validation de toutes les conditions, et vers ControlPad via le topic *controlpad-telemetry*, qui vérifie le niveau de carburant pour le staging et le maxQ.
- Au moment du maxQ, le controlPad envoie une requête **POST /mock/:id/throttle-down** vers le hardware pour diminuer la vitesse. En cas de réussite de cette étape, un événement *MAX_Q* est envoyé vers la mission.

Staging

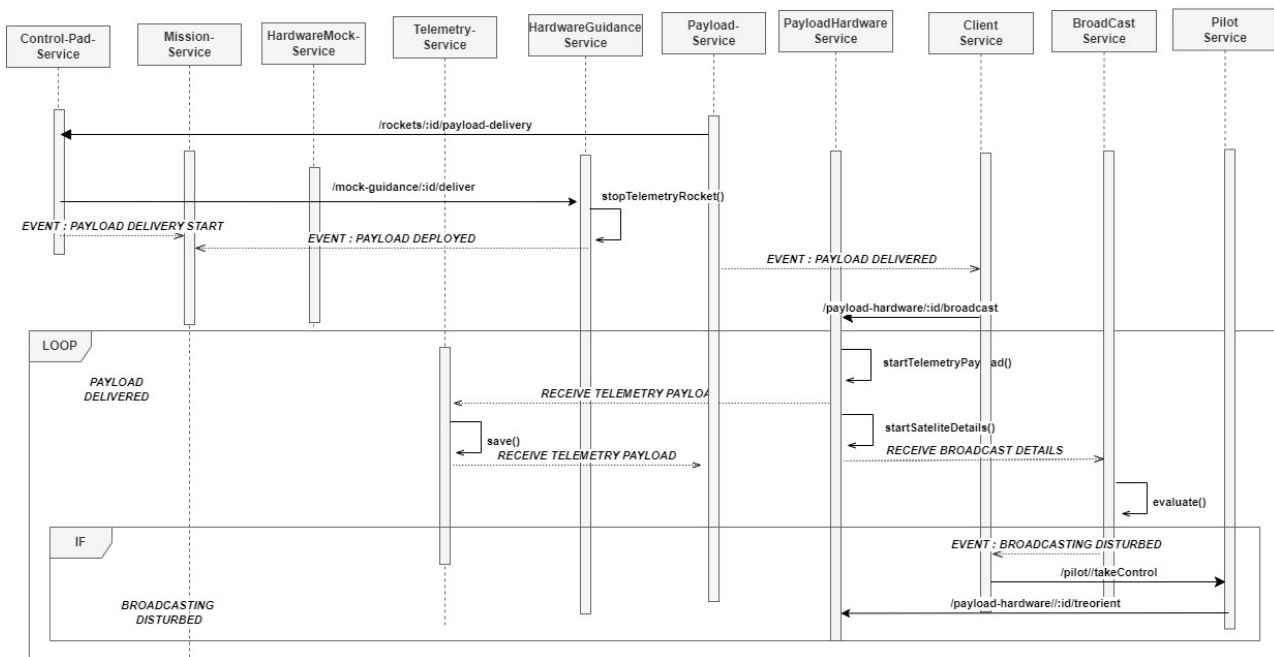
ETAPE 8



- Recevant les télémetries, le controlPad, lorsque le niveau de carburant est à zéro, appelle le hardware avec **POST /mock/:id/stage** pour effectuer le staging. Pendant cette phase, plusieurs événements sont déclenchés et envoyés vers la mission via *le topic-mission-events* : *STAGE_SEPARATION* et *MAIN_ENGINE_CUT_OFF*. Ensuite, le hardware cesse la transmission des télémetries de la fusée pour se charger des télémessures du booster, puis envoie un événement *SECOND_ENGINE_START* au hardware guidance pour débiter la transmission des télémetries du deuxième compartiment.
- Les télémetries du booster et du deuxième compartiment sont également envoyées dans le même topic *telemetry*, et le service de télémétrie joue le rôle de distribuer ces différentes télémessures via des topics différents : *booster-telemetry* pour suivre le booster depuis *boosterControl*, de même pour *controlPad-telemetry*, *mission-telemetry*, et *payload-telemetry*, chacun vérifie les conditions de ces télémetries reçus.
- À une altitude spécifique, le boosterControl envoie une demande **POST /mock/:id/land** pour effectuer l'atterrissage. Si tout fonctionne comme prévu, le statut de la mission est modifié, et le hardware commence les différentes étapes de l'atterrissage en envoyant les événements suivants : *FLIP_MANEUVER*, *ENTRY_BURN*, *LANDING_BURN*, *LANDING_LEG_DEPLOYMENT*, et à l'altitude zéro, l'événement *LANDED*. Ensuite, les télémetries du booster sont arrêtées.

Livraison du payload + BroadCasting

ETAPE 9+10+11



- Le service payload vérifie l'emplacement correct et envoie une requête **POST rockets/:id/payload-delivery** au service control pad pour annoncer la livraison de la charge. Ce dernier envoie à son tour une requête au service hardware guidance pour suspendre la génération de télémetries et un évènement "PAYLOAD DELIVERY START" au service mission
- En cas d'une livraison réussie, le service de payload envoie un évènement "DELIVERED" au service client et le service payload hardware.
- À son tour le service client envoie une requête **POST /payload-hardware/:id/broadCast** au service matériel de chargement (payload hardware) pour lancer le satellite et transmettre les détails au service de diffusion.
- Le payload hardware envoie les détails de déplacement du satellite au service de diffusion via le broker. Le broadcast service envoie "BROADCASTING STARTED" au client service la première fois pour annoncer le début de la diffusion.
- En cas de réception d'informations dégradées, le service de diffusion (broadcast) envoie "BROADCASTING DISTURBED" au service client pour annoncer la perte de la diffusion. Le service client, à son tour, envoie une requête **POST /pilot/takeControl** au service de pilotage pour ajuster le déplacement du satellite, qui envoie une requête **POST /payload-hardware/:id/reorient** pour réorienter le satellite. Le service payload hardware transmet les ajustements au service de diffusion, qui notifie ensuite le service client que la diffusion a été reprise "Broadcasting RESUMED".
- À la fin, le service de diffusion (broadcast service) envoie un évènement "BROADCASTING TERMINATED" au service client pour annoncer la fin de la diffusion.

Scénario 2 : Anomalie détectée

Dans ce scénario, la destruction de la fusée est déclenchée en raison d'un angle dépassant la limite autorisée. Voici une analyse des parties pertinentes des logs :

1- **Angle dépassé pour la fusée :**

Les données de télémétrie indiquent que l'angle de la fusée la limite autorisée. Dans ce cas, l'angle était de 45 degrés.

2- **Émission d'un ordre de destruction pour la fusée :**

Le hardware a répondu à l'angle excessif en émettant un ordre d'auto-destruction

3- **Notification du service mission :**

Une fois qu'il a reçu l'évènement, il l'enregistre et l'envoie au service webcaster pour qu'il puisse le publier.

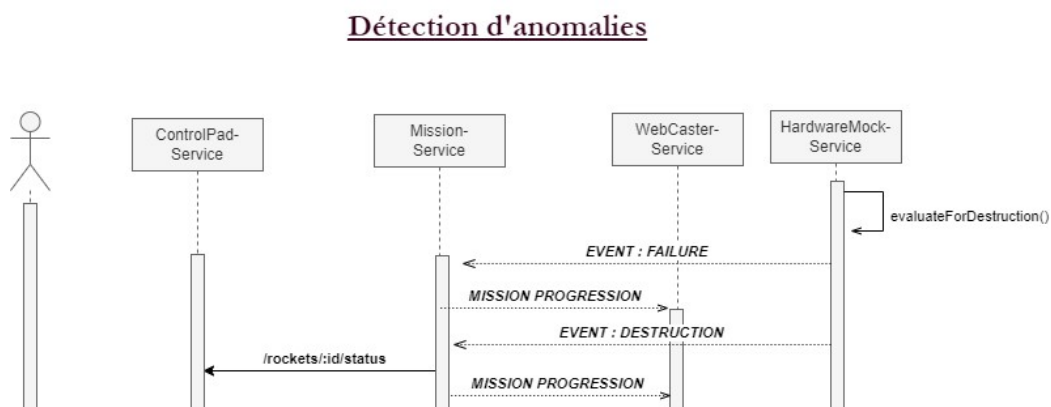
4- **Webcasting :**

Le service rapport les nouvelles liées à la fusée. Initialement, il mentionne une panne de démarrage, puis il signale que la fusée était détruite.

Logs associés :

```
LOG [SiteService] Received request to add site name : testSite9
LOG [MissionService] Received request to add mission name : testMission9
LOG [HardwareService] Angle exceeded for rocket 1A1. Angle: 45 1
LOG [HardwareService] Issuing order to destroy rocket 1A1. Reason: Angle exceeded (us 18) 2
LOG [MissionService] checking rocket 1A1 status
LOG [MissionService] Saving event for mission 1A1 (us 14) 3
LOG [MissionService] checking rocket 1A1 status
LOG [MissionService] Saving event for mission 1A1 (us 14)
LOG [WebCasterService] News from 1A1 Just in : start up failure (us 15) 4
LOG [WebCasterService] News from 1A1 Just in : the rocket is destroyed (us 15)
```

Diagramme de séquence associé :



- Le service hardware responsable de la génération des télémétries les évalue. En cas d'anomalie, un événement *FAILURE* est envoyé à la mission pour changer le statut de la mission en "failed", puis *DESTRUCTION*. À ce moment-là, la mission envoie **PUT /rockets/:id/status** pour déclencher la destruction.

Scénario 3 : Envoie d'une deuxième fusée avec un service qui tombe

Dans ce scénario, la destruction de la fusée est déclenchée en raison de problèmes techniques. Dans ce cas, le hardware ne répond pas.

Voici une analyse des parties pertinentes des logs :

- 1- Le scénario débute de la même manière que le premier, avec les étapes suivantes : **setup, initiation de la séquence de lancement, vérification des conditions météorologiques et de l'état de la fusée, puis démarrage du lancement et envoi de la télémétrie.**

- 2- Suite à la vérification de l'approche MAX Q, le Control Pad, recevant les télémesures, demande de réduire la poussée (**throttle down**).
- 3- Le système hardware est hors service et répond avec un **statut 400**.
- 4- Le Control Pad effectue des retrys, avec un délai d'attente augmentant de manière exponentielle à chaque nouvel essai.
- 5- Lorsque le nombre d'essais atteint la limite, le Control Pad appelle le service de la mission pour changer le statut en "échec" (**failed**).
- 6- Lorsque le hardware revient **opérationnel**, il redémarre et envoie les télémesures. La mission, recevant les télémesures, envoie une demande de destruction au hardware qui exécute alors l'ordre de **destruction de la fusée**.

Logs associés :

```

DEBUG [ControlPadService] Reached MaxQ for rocket 8AF 2
DEBUG [ControlPadService] Throttling up engines for rocket 8AF
LOG [ControlPadService] Checking fuel level for rocket 8AF - Fuel: 90 liters.
ERROR [MarsyMockHardwareProxyService] Error while throttling down engines for rocket 8AF: Request failed with status code 400 3

LOG [MarsyMockHardwareProxyService] Retrying in 1s...
LOG [MarsyMockHardwareProxyService] Request to start throttling down engines for rocket :

ERROR [MarsyMockHardwareProxyService] Error while throttling down engines for rocket 8AF: Request failed with status code 400

LOG [MarsyMockHardwareProxyService] Retrying in 2.718281828459045s... 4
LOG [MarsyMockHardwareProxyService] Request to start throttling down engines for rocket :

ERROR [MarsyMockHardwareProxyService] Error while throttling down engines for rocket 8AF: Request failed with status code 400

LOG [MarsyMockHardwareProxyService] Retrying in 7.38905609893065s...
LOG [MarsyMockHardwareProxyService] Request to start throttling down engines for rocket :

ERROR [MarsyMockHardwareProxyService] Error while throttling down engines for rocket 8AF: Request failed with status code 400

ERROR [MarsyMockHardwareProxyService] Max attempts reached. Giving up. 5
DEBUG [MarsyMissionProxyService] Declaring the failure of the mission of the rocket 8AF
LOG [MissionService] Mission of the rocket 8AF failed
DEBUG [HardwareService] Rebooted : Resending telemetry
LOG [TelemetryService] Evaluating telemetry for rocket: 8AF
LOG [TelemetryService] Telemetry for rocket 8AF is within safe parameters. No need for des

LOG [ControlPadService] Checking if approaching MaxQ for rocket 8AF - Altitude: 6052 meter 6

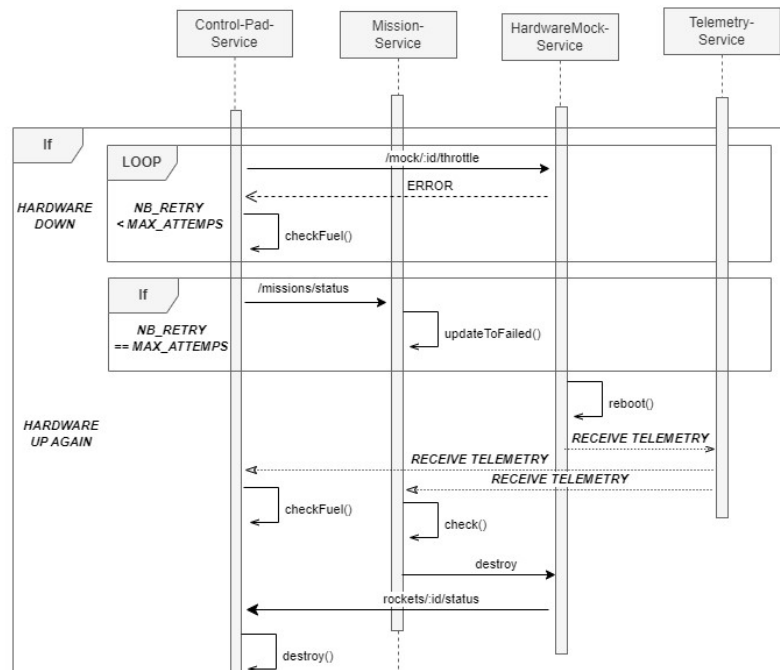
LOG [ControlPadService] Checking fuel level for rocket 8AF - Fuel: 85 liters.
LOG [MissionService] Received telemetry from malfunctioning rocket 8AF for failed mission

LOG [MissionService] Issuing order to destroy rocket 8AF. Reason: Mission failed (us 8)
LOG [HardwareProxyService] Sending explosion order to rocket : 8AF
LOG [MarsyRocketProxyService] Rocket 8AF has been successfully destroyed.
LOG [HardwareProxyService] Rocket exploded

```

Diagramme de séquence associé :

Mécanisme de Réessai avec Délai d'Attente



- Au moment du maxQ, un appel REST **/mock/:id/throttle** du controlPad vers le hardware est effectué. Si ce service répond avec une erreur 400, le controlPad réessaie en envoyant à chaque fois le même appel REST.
- Atteignant le nombre maximal de tentatives, le controlPad appelle la mission avec PUT **/missions/:id/status** pour changer son statut en "failed".
- Lorsque le hardware effectue un redémarrage et la mission reçoit ces télémessures, elle demande alors la destruction. Le hardware appelle ensuite **PUT /rockets/:id/status** pour déclencher la destruction.

Test de charge :

Nous avons réalisé un test de charge sur le topic "telemetry" étant donné que le service de télémétrie ("telemetry-service") reçoit l'ensemble des données de télémétrie. Pour ce faire, nous avons utilisé l'outil de test de charge k6. Nous avons inondé le topic Kafka de messages de télémétrie et surveillé les résultats.

Nous avons simulé un scénario impliquant 10 fusées virtuels, reproduisant des télémétries réalistes sur une période de 30 secondes. Cela s'est traduit par l'envoi de 28440 événements en total.

```
checks.....: 88.85% ✓ 36032      x 4519
data_received.....: 0 B      0 B/s
data_sent.....: 0 B      0 B/s
iteration_duration.....: avg=66.86ms min=9.6ms med=16.54ms max=1.61s p(90)=31.73ms p(95)=602.91ms
iterations.....: 4519      146.928005/s
kafka_reader_dial_count.....: 10      0.325134/s
kafka_reader_dial_seconds.....: avg=64.32µs min=0s med=0s max=40.88ms p(90)=0s p(95)=0s
✓ kafka_reader_error_count.....: 0      0/s
kafka_reader_fetch_bytes.....: 2.2 MB 72 kB/s
kafka_reader_fetch_bytes_max.....: 1000000 min=1000000 max=1000000
kafka_reader_fetch_bytes_min.....: 1 min=1 max=1
kafka_reader_fetch_size.....: 10947 355.92407/s
kafka_reader_fetch_wait_max.....: 10s min=10s max=10s
kafka_reader_fetches_count.....: 66 2.145884/s
kafka_reader_lag.....: 100339 min=0 max=242848
kafka_reader_message_bytes.....: 4.6 MB 149 kB/s
kafka_reader_message_count.....: 46100 1498.867235/s
kafka_reader_offset.....: 4870 min=20 max=5080
kafka_reader_queue_capacity.....: 100 min=100 max=100
kafka_reader_queue_length.....: 91 min=10 max=100
kafka_reader_read_seconds.....: avg=12.95ms min=0s med=0s max=9.14s p(90)=0s p(95)=0s
kafka_reader_rebalance_count.....: 0 0/s
kafka_reader_timeouts_count.....: 1 0.032513/s
kafka_reader_wait_seconds.....: avg=52.36µs min=0s med=0s max=16.76ms p(90)=0s p(95)=0s
kafka_writer_acks_required.....: 0 min=0 max=0
kafka_writer_async.....: 0.00% ✓ 0 x 451900
kafka_writer_attempts_max.....: 0 min=0 max=0
kafka_writer_batch_bytes.....: 61 MB 2.0 MB/s
kafka_writer_batch_max.....: 1 min=1 max=1
kafka_writer_batch_queue_seconds.....: avg=8.26µs min=0s med=4.61µs max=7.04ms p(90)=13.98µs p(95)=19.56µs
kafka_writer_batch_seconds.....: avg=232.74µs min=4.05µs med=14.2µs max=798.26ms p(90)=34.66µs p(95)=49.02µs
kafka_writer_batch_size.....: 451900 14692.800513/s
kafka_writer_batch_timeout.....: 0s min=0s max=0s
✓ kafka_writer_error_count.....: 0 0/s
kafka_writer_message_bytes.....: 123 MB 4.0 MB/s
kafka_writer_message_count.....: 903800 29385.601027/s
kafka_writer_read_timeout.....: 0s min=0s max=0s
kafka_writer_retries_count.....: 0 0/s
kafka_writer_wait_seconds.....: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
kafka_writer_write_count.....: 903800 29385.601027/s
kafka_writer_write_seconds.....: avg=263.18µs min=4.35µs med=10.38µs max=798.28ms p(90)=29.57µs p(95)=43.57µs
kafka_writer_write_timeout.....: 0s min=0s max=0s
vus.....: 10 min=10 max=10
vus_max.....: 10 min=10 max=10
```

Comme nous pouvons le voir, le taux de réussite indique que 88 % des requêtes effectuées pendant le test de charge ont passé les contrôles et validations spécifiés du côté du consommateur kafka, tandis que les 12 % restants ont échoué.

En outre, la métrique **kafka_reader_lag** indique un retard actuel de **100339µs**, ce qui signifie que le consommateur Kafka, Telemetry Service, est toujours en retard par rapport au dernier message produit. Le décalage minimum observé est de **0µs**, ce qui suggère des cas où le consommateur a suivi le rythme du producteur, tandis que le décalage maximum a atteint **242848µs**, ce qui indique le retard le plus important enregistré. Bien qu'un certain degré de retard soit normal, la valeur actuelle est excessive dans le contexte de notre application qui traite des valeurs en temps réel, ce qui signale le problème de surcharge.

La cause de ces échecs peut être attribuée à la charge significative supportée par le consommateur "telemetry". Une solution possible est celle que nous avons mentionnée précédemment et qui consiste à alléger la charge du service de télémétrie en le fragmentant en plusieurs microservices distincts. Chaque microservice serait dédié à la réception de types spécifiques de télémétrie. Cette approche modulaire permettrait une gestion plus efficace des différents flux de données, améliorant ainsi la résilience et la réactivité globales du système.

En complément, s'assurer que les topic kafka soient configurés avec un nombre approprié de partitions peut permettre une meilleure répartition des charges et une gestion plus efficace du traitement des messages. Pour le moment nous n'utilisons qu'une seule partition dans ce topic.

PRISE DE RECUL :

Depuis la première livraison, notre architecture a évolué afin de répondre à de nouvelles fonctionnalités et d'assurer une meilleure scalabilité. Nous avons introduit des nouveaux services spécifiquement conçus pour les nouveaux user stories, visant à fournir un service dédié à chaque ensemble de fonctionnalités partageant des activités similaires.

Une transition notable a été l'élimination de certains appels REST au profit de l'utilisation du broker de messagerie.

En rétrospective, l'examen approfondi de notre architecture actuelle et des développements réalisés nous a permis de prendre du recul et d'identifier clairement les points forts et les faiblesses de notre structure. Nous sommes fiers de constater que nous avons pu répondre de manière exhaustive à tous les besoins des utilisateurs, démontrant ainsi l'efficacité des ajustements apportés à notre architecture au fil du temps.

Forces et faiblesse :

Forces :

- a. Notre système repose sur une architecture orientée service, où chaque service est Stateless, dédié à un domaine particulier avec des responsabilités claires. Pour être plus précis, nous sommes particulièrement fiers d'avoir essayé de suivre une approche DDD (Domain Driven Design), du moins en termes de contexte délimité, puisque chaque service gère une partie du modèle de domaine de manière indépendante. Par exemple, la gestion de la mission est confiée à Richard le commandant de la mission, le control pad est sous la responsabilité du chef du département des fusées, le Weather Service à Tory, le météorologue, le Payload Service à Gwynne, la chef du département Payload, BoosterControl à Peter, le directeur général, le WebCaster à Marie... Également, pour optimiser la gestion des données et des ressources, nous avons divisé la responsabilité de l'envoi de la télémétrie : le HardwareMock est chargé de l'envoi des données de la fusée pendant la première période, le Guidance Hardware prend le relais pendant la deuxième période pour les télémétries du deuxième compartiment, et enfin, le PayloadHardware prend en charge les télémétries associées au déploiement du payload.

De plus, chaque microservice a des modèles de données qui évoluent ensemble, et globalement, nous avons essayé de maintenir un seul agrégat par microservice, permettant aux agrégats d'être notre unité de transactions (mises à jour dans la base de données).

- b. Par ailleurs, nous avons suivi une approche d'"event sourcing" au niveau de la conception, permettant à notre event store Kafka d'être une source de vérité pour l'avancement de la mission. Les commandes claires de nos scénarios ont produit des événements et déclenché des politiques. Cela permet à notre architecture d'être un système réactionnel en temps réel, ce qui non seulement s'aligne directement avec le métier de Mars y, mais favorise également la flexibilité et l'évolution, car les nouveaux microservices potentiels peuvent simplement brancher leurs listeners sur les topics appropriés pour consommer les événements et réagir à leur tour. Par exemple, en écoutant le topic des événements de la mission.
- c. Nous sommes aussi fiers d'avoir atténué le SPOF que nous avons reconnu dans notre première livraison, à savoir la "télémétrie", en nous appuyant sur le broker de messagerie pour tamponner la transmission de la télémétrie à partir des différents hardware. Il n'est donc pas bombardé d'appels REST. Il s'agit de continuer à honorer le concept d'"event sourcing" qui va pair avec le CQRS. Dans notre cas, le service de commande "C" est le service télémétrie qui reçoit et stocke la télémétrie et gère les télémétries erronées en envoyant des alertes au service de mission. Les services de requête "Q" sont tous les autres services utilisés pour suivre un point de vue spécifique de la mission qui consomme la télémétrie adaptée à leurs besoins et publiée par le service de télémétrie dans les topics appropriées.

- d. En outre, les interfaces que nous avons fournies à chaque niveau des microservices sont basées sur des cas d'utilisation, ce qui limite la prolifération des messages lors de l'interaction avec notre système. Ceci est particulièrement important pour réduire la consommation de ressources lors de l'interaction avec notre architecture de microservices.
- e. Finalement, nous avons partiellement mis en œuvre un mécanisme de réessai basé sur le délai d'attente et avec un temps de recul exponentiel pour les commandes émises par le ControlPad sur le service hardware. Après trois essais, la mission est considérée comme un échec. Nous pensons que la mise en œuvre de tels mécanismes globalement en interne entre les services et a conception en vue d'une éventuelle défaillance contribuent à rendre notre système plus résilient. Par exemple, lorsque la limite de tentatives de commandes est atteinte et que le ControlPad le déclare, la mission est considérée comme un échec, toutes les parties en sont informées et l'état est mis à jour dans les microservices pour le refléter.

Faiblesses :

- a. Une faiblesse constatée est l'acheminement systématique de toutes les télémétries vers le topic "telemetry". Cela expose le système à des risques de surcharge du cluster Kafka, entraînant des problèmes de performance. Bien que la messagerie asynchrone ait été utilisée, notre point unique de défaillance persiste, générant un volume de messages élevé. Il serait plus avisé de répartir le stockage et la surveillance de la télémétrie de chaque service matériel sur différents sous-systèmes pour optimiser la gestion des messages et améliorer la fiabilité globale.
- b. Le nommage des microservices aurait pu être amélioré. Le Telemetry Service aurait pu être renommé "Monitoring Service", le HardwareMock Service en "Propellant Service". Pour mieux s'aligner avec le DDD.
- c. Le service Controlpad tel qu'il est actuellement et comme mentionné précédemment gère à la fois l'agrégat de fusées et le contrôle des fusées en émettant des ordres pendant qu'elles sont en l'air. Au préalable, nous avions une vision tunnel dans laquelle nous ne voulions pas de services bavards et nous voulions modéliser des choses qui changent ensemble dans les mêmes Microservices. Cependant, cette vision est étroite. Voilà que nous faisons le point sur nos choix, Il semble logique de profiter de l'"event sourcing" pour les séparer en deux services qui consomment les mêmes événements à partir du même sujet Kafka : un service de contrôle en charge du contrôle et un service de fusées qui est en charge de la mise à jour de l'état de l'agregat. Une telle séparation permettrait une évolution autonome de la logique de contrôle et de la logique des fusées séparément, en profitant de l'architecture des microservices pour déployer ces changements de manière indépendante.
- d. Il convient de préciser que nous n'avons pas mis en œuvre de " API Gateway ", car nous nous sommes concentrés sur le développement des fonctionnalités qui répondent aux besoins des utilisateurs. L'absence de passerelle vers notre système est l'une de ses faiblesses, car tout est exposé de manière brute au public, sur différents ports, ce qui rend son utilisation très compliquée. Une passerelle d'api permettrait d'y remédier, car elle connaît l'emplacement de tous les services, sinon les clients auraient besoin d'un répertoire de découverte des services.

Répartition des tâches :

Chaque semaine, nous examinons les nouvelles user stories, définissant les critères d'acceptation spécifiés. Ensuite, nous créons des tâches sur GitHub pour suivre attentivement l'avancement de chaque user story et la contribution de chaque membre. En cas de blocage, nous encourageons la collaboration et le partage de connaissances pour surmonter les obstacles. En récompense, un score de **(100)** points est attribué à chaque membre de l'équipe.