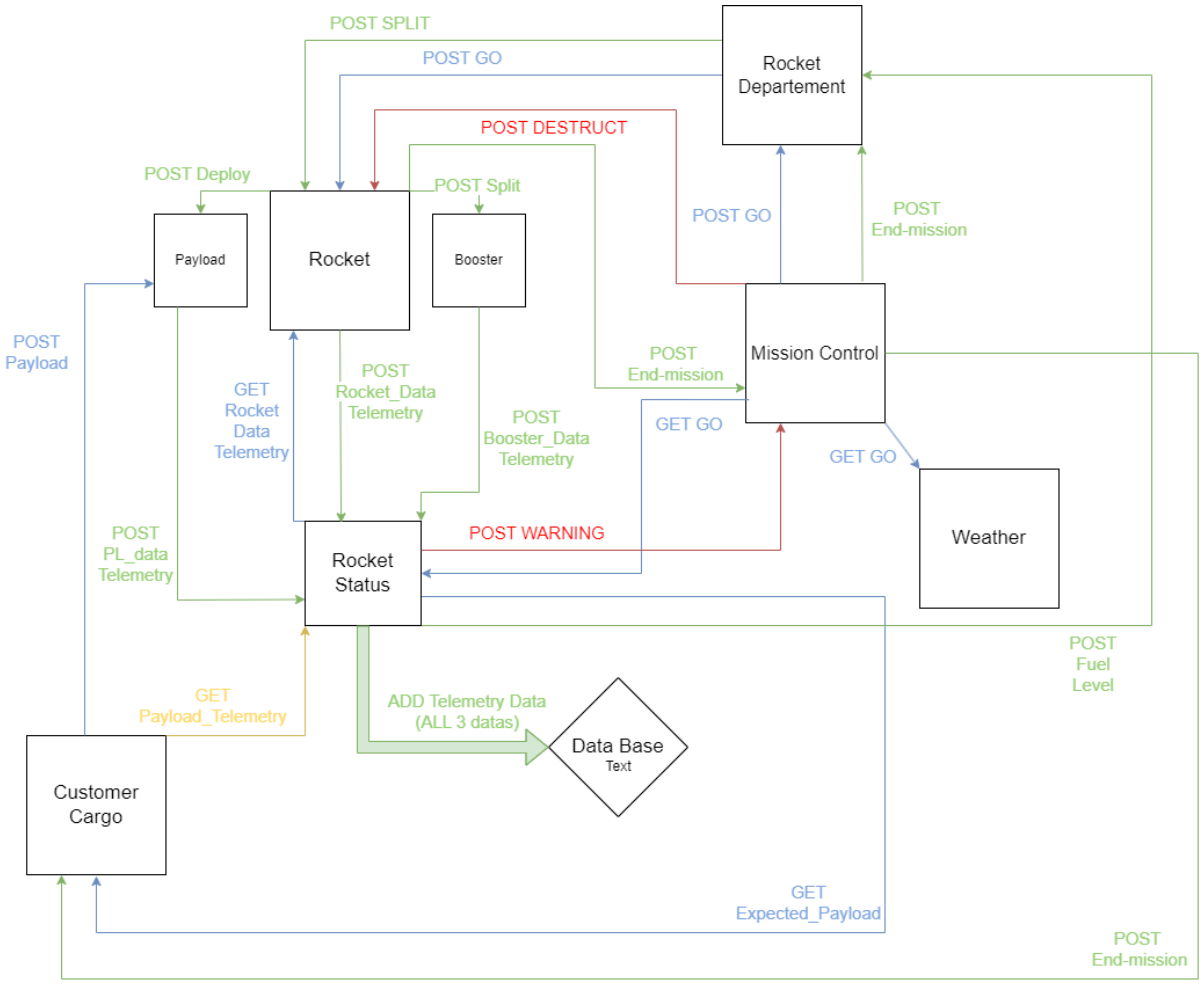
Projet Intégration de services et micro-services

Équipe E

Mohamed MAHJOUB
Thomas FARINEAU
Léo KITABDJIAN
Ludovic BAILET

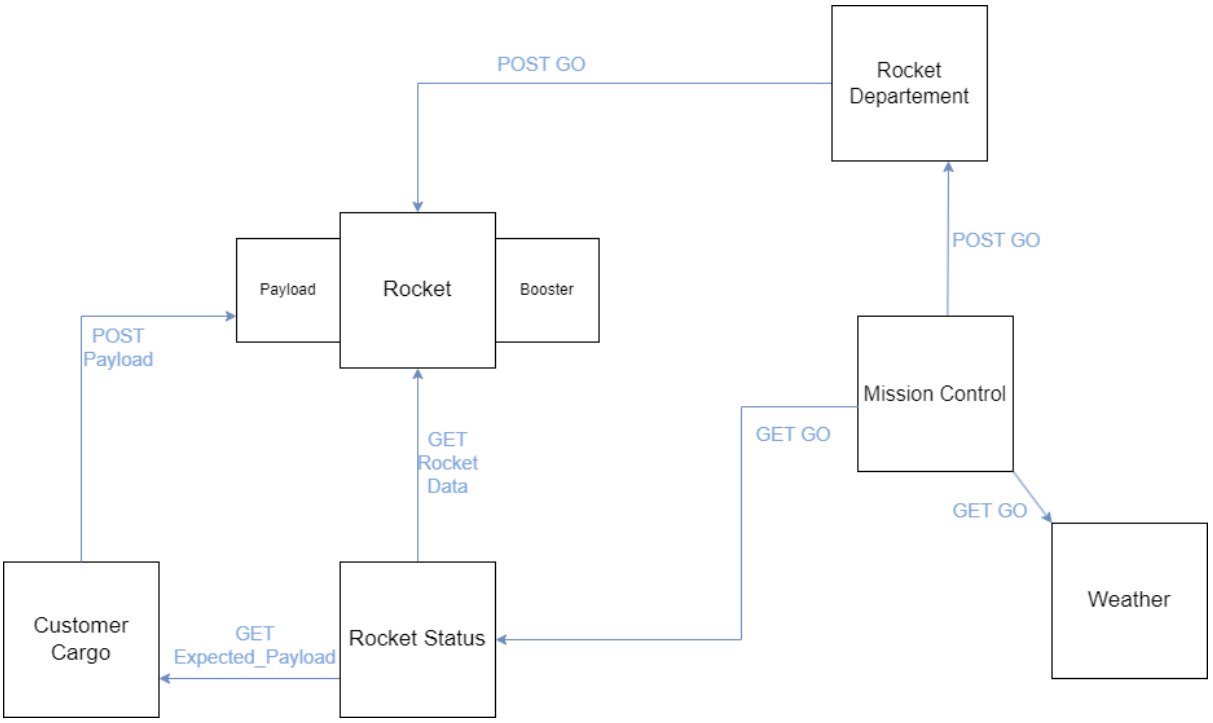
Polytech Nice Sophia SI5-M2 2023-2024

Schéma Architecture Logicielle



Voici le diagramme d'architecture de notre projet. Pour faciliter son explication, nous avons séparé les opérations dans 3 groupes diffèrents , les Opérations au sol (en bleu) les opérations en plein vol concernant l'opération initiale , (en vert) et les opérations secondaires (en rouge et jaune)

Les Opérations en Bleu



Ces opérations concernent toutes les communications qui se déroulent avant le lancement de la fusée. Afin de mieux les expliquer, nous allons suivre un scénario classique du lancement de la fusée.

- Avant le début de la mission, le **Customer Cargo** charge le payload à envoyer dans la fusée (avec la requête *POST* Payload).

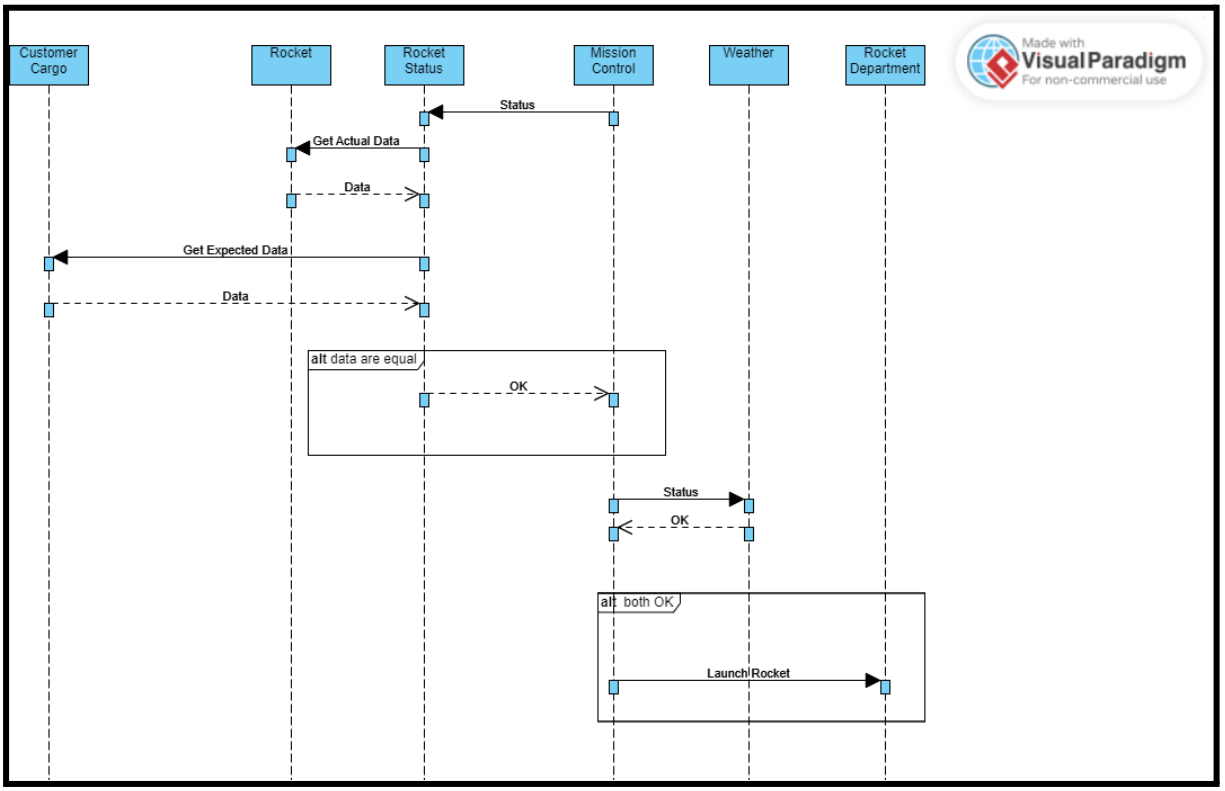
- Le **Mission Control** veut lancer la fusée, il va donc créer un sondage (Poll) et consulte les départements **Weather** et **Rocket Status** pour vérifier si les conditions sont valides pour lancer la fusée. (Requêtes *GET* GO)[User Story 3]

- Le **Weather** se contente de renvoyer GO ou NO GO. (il répond à la requête de **Mission Control)** [User Story 1]

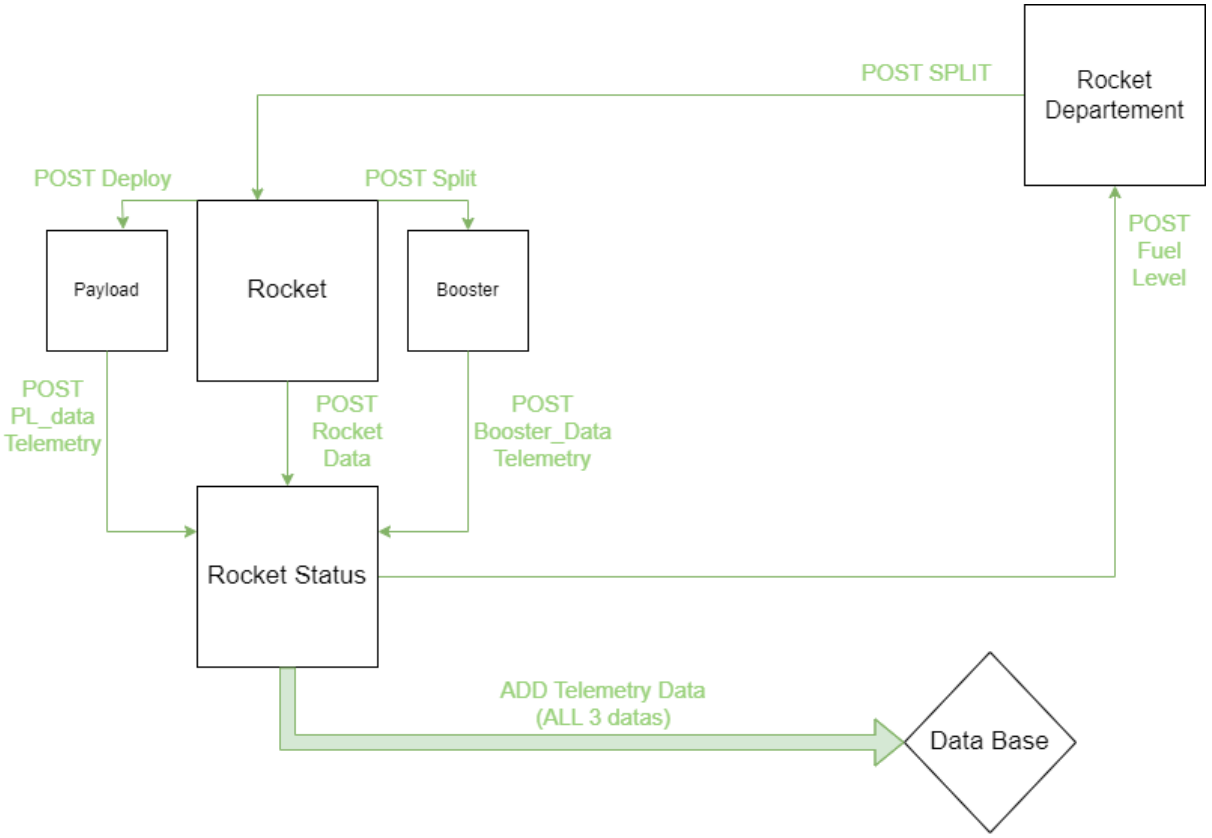
- Le **Rocket Statuts** afin de répondre à la requête de **Mission Control** va vérifier si la **Rocket** est prête (Il envoie une requête *GET* à la **Rocket)**, de plus il va aussi vérifier le contenue du Payload en le comparant avec ce que le **Customer Cargo** souhaite envoyer (à l'aide d'une requête *GET*) . Si tout est valide il renvoie GO au **Mission Control.** [User Story 2]

- Une fois le Poll complété (et validé) le **Mission Control** envoie une confirmation au **Rocket Department** (POST GO) une fois cette

confirmation reçue, le **Rocket Department** va envoyer un *POST* GO à la **Rocket** de démarrer. [User Story 4]



Les Opérations en Vert



Ces opérations concernent toutes les communications qui se poursuivent pendant le lancement de la fusée. Afin de mieux les expliquer nous allons suivre le scénario du vol avec succès.

- Une fois en plein vol, la **Rocket** envoie continuellement (à l'aide de requêtes *POST*) au **Rocket Status** ses données, celles-ci seront envoyées au service de **Telemetry** (avec requête *POST*.) qui se chargera de les stocker dans une **Data Base**. [User Story 5]

- Durant le vol, **Rocket Status** qui reçoit et traite les données de la **Rocket et du Booster** va envoyer les informations concernant l'essence contenue dans le **Booster** à **Rocket Departement**, lorsque le niveau d'essence dans le **Booster** atteint un niveau critique celui-ci envoie la tâche à la **Rocket** de se séparer du **Booster** (avec une requête *POST*). [User Story 6]

  *Le **Booster** se met à envoyer ses données au Rocket Status pour les sauvegarder à ce stade.*

- Une fois que la **Rocket** a atteint son objectif, il va déployer le **Payload** dans l'espace. [User Story 7]

  *Le **Payload** se met à envoyer ses données au Rocket Status pour les sauvegarder à ce stade.*

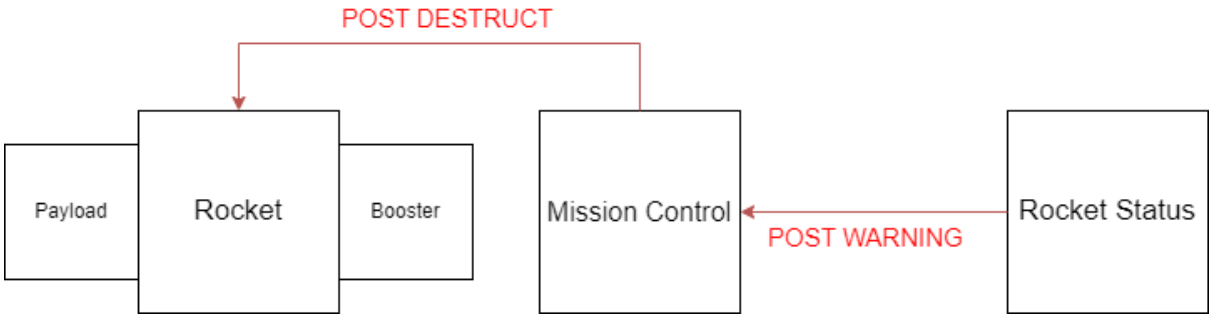
Les différents éléments de la **Rocket** devenus indépendants ont donc désormais leurs propres tâches à effectuer, si l'on part du moment où le **Booster** s'est séparé de la **Rocket** (avec un *POST*)

- Le **Booster** va chercher à atterrir sur le sol de manière contrôlée afin de pouvoir être réutilisé dans le futur. [User Story 9]

- De plus, le **Booster** va envoyer sa télémétrie (avec des requêtes *POST*) de façon continue à partir du moment où il s'est détaché au service de **RocketStatus**. [User Story 10]

Le **Payload** lui aussi devient indépendant après son déploiement, il se met à envoyer ses propres données de télémétrie au service de **Rocket Status.** [User Story 11]

L’Opération d’urgence Rouge



Jusqu'à présent, nous avons supposé que toutes les opérations se déroulaient parfaitement. Cependant, nous devons envisager l'hypothèse de problèmes de trajectoire en plein vol.

Afin de gérer cette éventualité, nous avons mis en place une fonctionnalité répondant à la [User Story 8], qui nous permet de détruire la Rocket en cours d'opération si nécessaire.

Pour satisfaire cette condition, nous utilisons le service **Rocket Status**, qui reçoit en permanence les données de la **Rocket** durant la mission. Grâce à ces données, il est en mesure de déterminer si la trajectoire de la Rocket dévie. Si c'est le cas, il partage cette information avec le **Mission Control** (à l'aide d'une requête *POST*).
Ensuite, la décision de faire exploser la **Rocket** en plein vol, y compris ses composants non déployés, revient à **Mission Control**. Si cette décision est prise, **Mission Control** envoie une demande de destruction à la **Rocket**(à travers une requête *POST*) et celle-ci sera alors détruite, mettant ainsi fin à la mission.

Justifications et explications de l’architecture

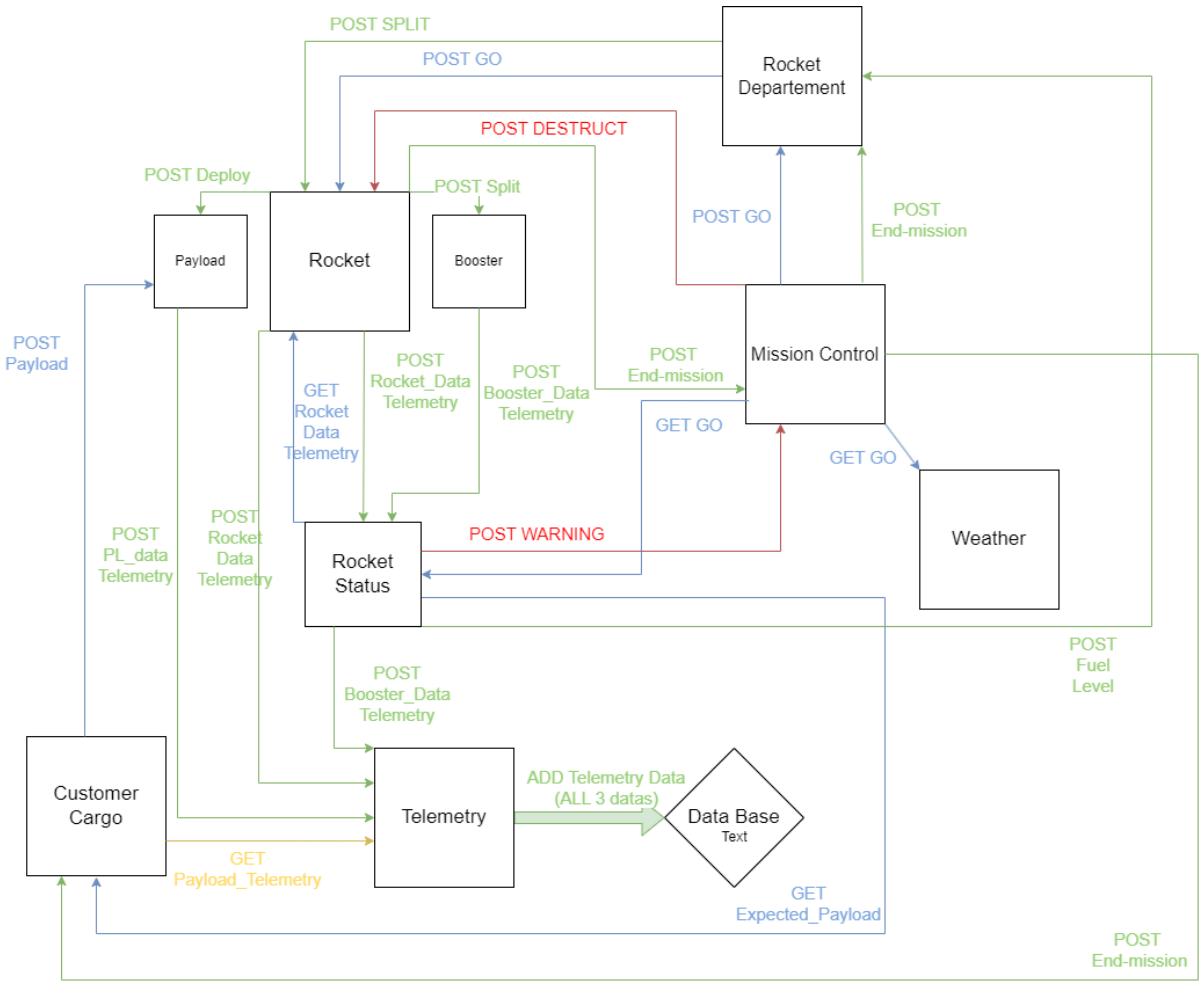
Nous avons réalisé une architecture basée sur les différents métiers et interfaces des user stories. Les services sont donc ici basés sur les besoins des différentes personnes décrites dans les US. Le Customer Cargo par exemple va regrouper tout ce qui concerne la demande du client (envoyer les données attendues, les définir, savoir quand la mission est terminée, etc...). Notre but a été de définir des services s’occupant de sections bien distinctes de la mission et de son déroulement. Le Mission Control valide les données avant la mission et est au courant de son déroulement, le Rocket Department sera celui qui donne les ordres à la fusée (démarrer, split le booster, délivrer le payload).
Les 3 services Rocket, Payload et Booster sont distincts car ils correspondent à des équipements qui seront embarqués sur ces équipements, et étant donné qu’ils se

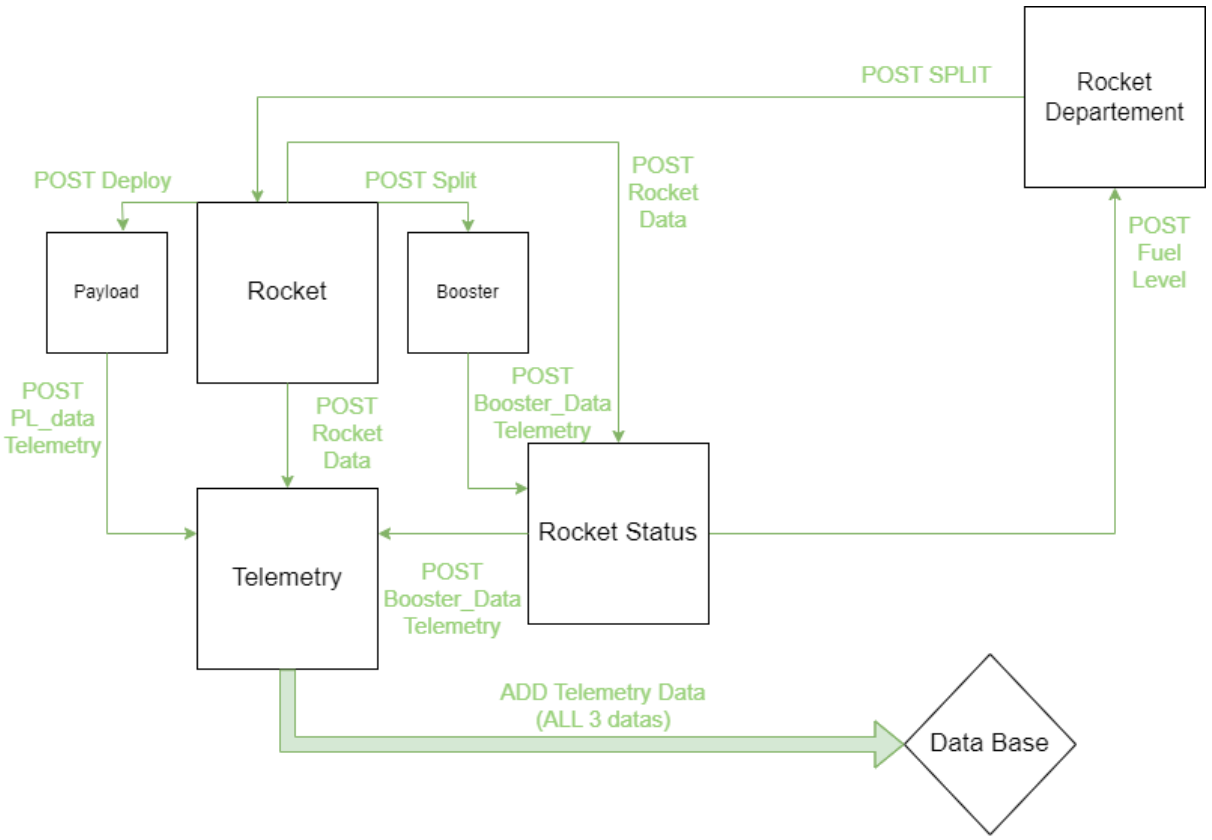
séparent dans les nouvelles US, nous sommes partis du principe qu'ils devaient être indépendants pour assurer leur fonctionnement après les stades de détachement. Enfin le Rocket Status va se charger de gérer les données des éléments de la Rocket et de ses équipements afin de gérer la logique métier et d'éviter au maximum la logique à l'intérieur des services embarqués (les 3 précédents).

Les services et users story manquantes.

Nous n'avons pas encore abordé la User Story 12 dans notre projet de fusée.
De plus, nous n'avons pas encore mis en place le service Telemetry par manque de temps. Pour l'instant, Rocket Status est chargé de la gestion de lecture et écriture de toutes les données. C'est une des limites de notre architecture car cela donne trop de responsabilités au service de Rocket Status (notamment certaines qui ne devraient pas lui incomber comme les données du booster ou du payload) et se rapproche d'un SPOF. On va donc transitionner cela vers une architecture délégant la lecture et l'écriture dans la base de données à un autre service Telemetry pour éviter également les éventuels problèmes de concurrence. Un autre axe d'amélioration serait de clarifier les responsabilités Rocket Status/Telemetry et Mission Control/Rocket Department.

Voici les Diagrammes que l'on souhaiterait avoir :


POST SPLIT
POST GO
Rocket Departement
POST DESTRUCT
POST Deploy
POST Split
POST GO
POST End-mission
Payload
Rocket
Booster
POST Payload
GET Rocket Data Telemetry
POST Rocket_Data Telemetry
POST Booster_Data Telemetry
POST End-mission
Mission Control
GET GO
GET GO
Weather
POST PL_data Telemetry
POST Rocket Data Telemetry
POST WARNING
Rocket Status
POST Fuel Level
POST Booster_Data Telemetry
Customer Cargo
Telemetry
ADD Telemetry Data (ALL 3 datas)
Data Base
Text
GET Payload_Telemetry
GET Expected_Payload
POST End-mission


Rocket Departement
POST SPLIT
POST Deploy
POST Split
POST Rocket Data
POST Fuel Level
Payload
Rocket
Booster
POST PL_data Telemetry
POST Rocket Data
POST Booster_Data Telemetry
Rocket Status
Telemetry
POST Booster_Data Telemetry
ADD Telemetry Data (ALL 3 datas)
Data Base