Projet SOA Selene

RAPPORT FINAL TEAM E

William D'ANDREA – Laurie FERNANDEZ – Emma GLESSER – Arthur SOENS

13-11-2022

Table des matières

Compréhension du sujet

RAPPEL DES PERSONAS

Les différentes persona du projet Selene sont les suivants :

- **Deke**, le chef des opérations, responsable de la division de soutien à la vie sur Terre, s'assurant de la **sécurité des équipages**.
- **Buzz**, le commandant du village lunaire, responsable de la **sécurité de l'équipage du village et des missions lunaires**.
- **Dorothy**, la responsable de la logistique, responsable des missions de **transit Terre-Lune,** s'assurant de la **durabilité à long terme de l'initiative de la base lunaire**.
- **Jim**, un astronaute de la base lunaire, responsable des provisions et de la logistique, s'assurant du **bon déroulement des opérations** sur la base.
- **Gene**, responsable des engins spatiaux et des lancements, qui s'assure **que le matériel de vol se comporte comme prévu (constantes du vaisseau)**
- **Marie**, un officier de relations publiques, responsable de la communication au public de l'Agence, doit **relayer les évènements se déroulant durant le programme Selene au public**.

USER STORIES DEVELOPPEES

Parmi les user stories données en consignes, toutes les user stories obligatoires ont été développées.

1. En tant que **Deke**, je veux m'**assurer à distance que les systèmes de support de vie fournissent des conditions de vie dans les modules lunaires habités**, afin que les occupants puissent survivre.
2. En tant que **Dorothy**, je veux **organiser et suivre les missions de réapprovisionnement**, afin de fournir tous les matériaux et ressources nécessaires à la réussite d'une mission.
3. En tant que **Buzz**, je veux **observer et contrôler l'état des modules de la base** afin d'**effectuer les changements nécessaires** pour soutenir l'équipage **et les activités quotidiennes de l'équipage**.
4. En tant que **Deke**, je veux être capable d'**isoler physiquement tout module de la base** afin de préserver l'intégrité de la base en cas de défaillance d'un des systèmes de support de vie.
5. En tant que **Jim**, je veux **garder une trace de l'inventaire des ressources disponibles sur la base** afin de ne jamais tomber à cours de ressources vitales.
6. En tant que **Buzz**, je veux **organiser et garder une trace des missions EVA (ExtraVehicular Activity)** autour de la base, dans le but de collecter des données scientifiques de valeur.
7. En tant que **Jim**, je veux **surveiller mes signes vitaux et l'équipement de ma combinaison spatiale (niveau d'oxygène, température, pression, puissance)**, afin d'exécuter ma mission en toute sécurité.
8. En tant que **Gene**, je veux **attribuer des vaisseaux spatiaux et programmer des lancements**, afin de soutenir les missions Terre-Lune.
9. En tant que **Jim**, je veux que **les épurateurs de CO_2 s'activent et se désactivent en fonction des niveaux de CO_2** dans les modules de la base, afin de garantir un air respirable sans risque pour l'équipage.
10. En tant que **Buzz**, je veux **être notifié en cas de dysfonctionnement de tout équipement d'une combinaison en utilisation en dehors de la base** de manière à **envoyer une équipe de secours** pour sauver la vie de l'astronaute.

11. En tant que **Deke**, je veux **observer les métriques[séries de valeurs] des combinaisons durant les missions EVA précédentes où elles ont été utilisées** de manière à améliorer les combinaisons suivant les données récoltées.
12. En tant que **Gene**, je veux **pouvoir programmer une nouvelle mission de ravitaillement lorsque la fusée utilisée pour une mission de ravitaillement échoue de manière catastrophique**, afin de pouvoir encore fournir le matériel de vol nécessaire à l'accomplissement de la mission originale.
13. En tant que **Deke**, je veux **suivre les météorites autour de la Lune,** afin de détecter tout danger potentiel pour la base lunaire.
14. En tant que **Deke**, je veux **faire sonner l'alarme qui prévient tout le personnel de se diriger vers le module de sécurité blindé lorsqu'un dangereux nuage de météorites est détecté**, et ensuite **isoler chaque module de la base lunaire**, afin d'éviter des scénarios dramatiques pour l'équipage lorsqu'un ou plusieurs modules sont percés.
15. En tant que **Marie**, je veux **être informée et notifiée des événements qui se déroulent dans la base lunaire, sur les installations terrestres, en vol**, **etc**, afin de pouvoir **relayer ces informations en temps réel au public.**
16. En tant que **Gene**, je veux **faire voler des astronautes de la Terre à la Lune et vice-versa**, afin de **gérer les rotations du personnel sur la base lunaire**.
17. En tant que Deke, je veux **suivre la localisation de chaque astronaute sur la Lune**, afin d'assurer la sécurité de chacun d'entre eux.
18. En tant que **Gene**, je veux **entretenir la base lunaire avec un nombre limité de vaisseaux spatiaux réutilisables**, afin de rester dans les limites du budget alloué.
19. En tant que **Buzz**, je veux **garder un enregistrement de chaque tâche/mission à laquelle chaque personnel de la base a été affecté**, afin de répartir correctement la charge de travail sur le personnel disponible.
20. En tant que **Jim**, je veux **être informé chaque fois qu'une mission de réapprovisionnement est retardée ou échouée**, afin de planifier le retard qui sera pris et le rationnement potentiel des ressources restantes sur la Lune.

Parmi celles où un choix était à faire, les user stories suivantes ont été choisies et développées :

- Partie A : 22. En tant que **Marie**, je veux **avoir accès au calendrier des missions EVA**, afin de planifier des diffusions éducatives en direct de certaines de ces missions pour sensibiliser le grand public au travail effectué sur la Lune.
- Partie B : 25. En tant que **Jim**, je veux **être informé lorsqu'un vaisseau spatial de ravitaillement est sur le point d'atterrir**, afin de décharger les marchandises rapidement et efficacement.

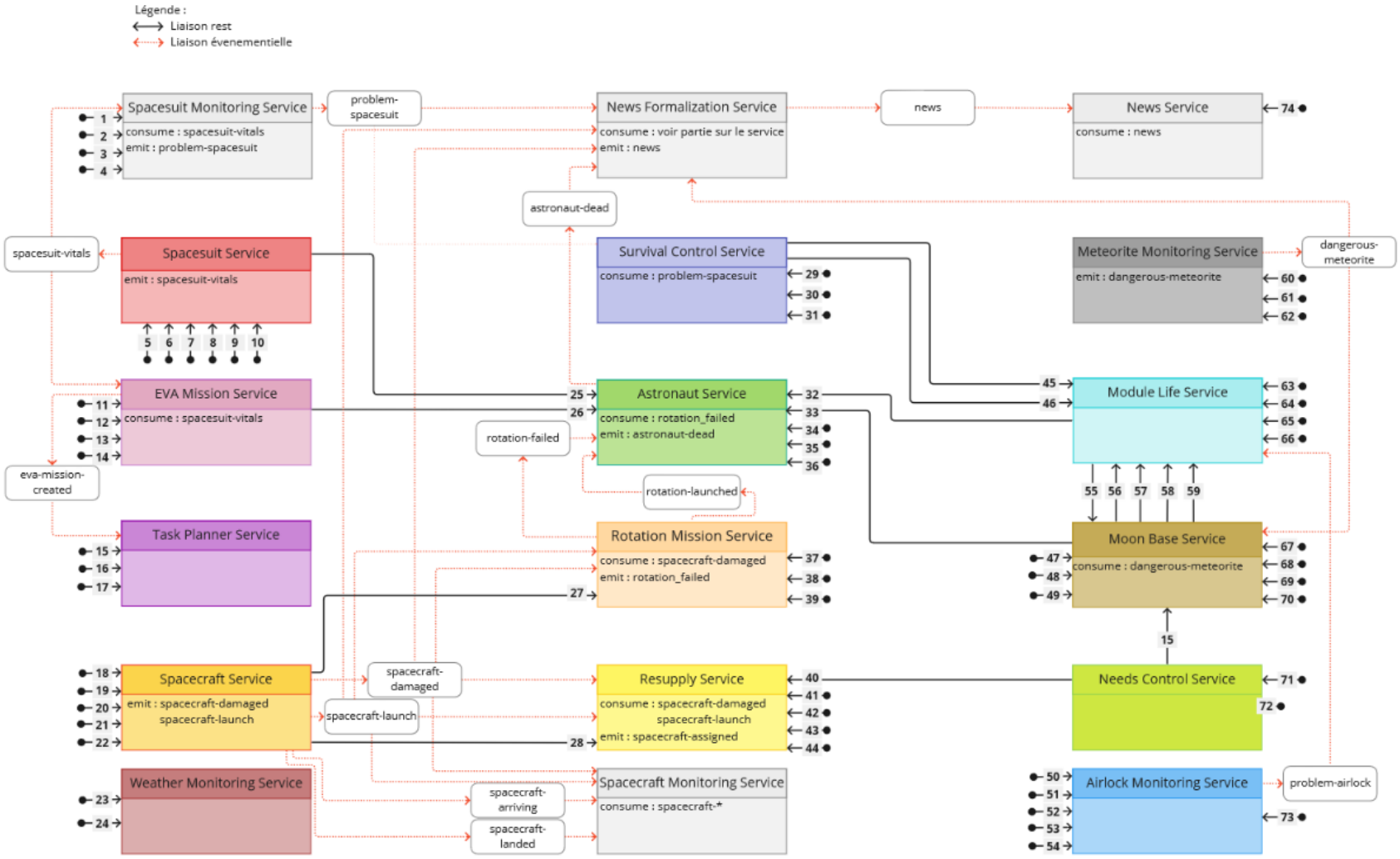
HYPOTHESES FAITES PAR L'EQUIPE

Un certain nombre d'hypothèses ont été faites par l'équipe lors de la lecture des user stories, en voici la liste :

- On peut rajouter des commandes à une mission de réapprovisionnement tant que celle-ci a un statut 'En cours de préparation' « PREPARING »
- Si aucune mission de réapprovisionnement n'a un statut "En cours de préparation" alors une nouvelle mission est créée
- Certaines opérations ne sont pas automatisées pour conserver l'aspect vérification par un humain comme ce serait le cas dans la vraie vie. C'est le cas de l'envoi de commande qui devra être réalisé par Buzz et non pas par Needs Control Service dès la réception de la liste des besoins. C'est également le cas du ravitaillement des modules qui devra être fait par un humain et non pas par un service dès la réception de la mission de ravitaillement pour en valider la préparation.
- Les fusées peuvent contenir une mission de réapprovisionnement, une mission de rotation d'astronautes ou les deux à la fois.
- Les vues sur l'inventaire de la base sont faites depuis le stock global de la base.
- Lorsqu'une mission échoue, la fusée, son ravitaillement et ses occupants sont tous trois considérés comme perdus.
- Nous disposons d'un système d'alarme global sur la base, géré seulement par la base elle-même, une alerte ne fera donc sonner que celle-ci.
- Les secteurs sont une simple liste de zones prédéfinies sur la Lune, les coordonnées GPS ne pouvant y être appliquées.
- Lors d'une alerte, les astronautes sont tous redirigés vers le module sécurisé, puis les modules lunaires isolés afin qu'aucun astronaute ne se retrouve enfermé dans un module non sécurisé en cas de chute de météorite prévue sur la base.
- La création d'une tâche de type EVA mission se fait par le biais de la création d'une EVA Mission.
- Dans le cas d'un dysfonctionnement dans les métriques d'un astronaute relevées par sa combinaison, une nouvelle EVA Mission de type « HELP » serait créée par un astronaute. Les reste des EVA Mission classiques d'exploration sont quant à elles de type « SCIENTIFIC ».
- Une EVA mission est une tâche à part entière, elle apparaitra donc elle aussi dans le registre des tâches.
- Afin de ne pas renvoyer sur Terre un astronaute qui vient tout juste d'arriver sur la Lune, on doit déclarer à l'avance à la fois les astronautes se rendant sur la Lune et ceux en revenant d'après les listes d'astronautes sur la Terre et ceux sur la Lune. La fusée amenant des astronautes sur le Lune est celle qui ramènera le groupe revenant sur Terre.
- Un astronaute dont on isole le module dans lequel il se trouvait est automatiquement redirigé vers le module sécurisé
- Lors de la création d'une EVA mission, on définit les combinaisons qui y seront utilisées
- Une fusée lorsqu'elle est lancée suit une trajectoire perpendiculaire au sol depuis son point de départ et sera ensuite orientée une fois qu'elle aura quitté l'atmosphère, une seule zone météorologique est donc à analyser pour lancer une fusée, celle depuis laquelle on la lance.
- Un module peut posséder entre o et plusieurs sas de pressurisation.

Diagramme d'architecture complet

DIAGRAMME D'ARCHITECTURE



INTERFACES

1. GET /spacecraft/landed
2. GET /spacecraft/arriving
3. GET /spacecraft/launched
4. GET/spacecraft/crashed

5. GET /spacesuit
6. GET /spacesuit/{spacesuitId}
7. POST /spacesuit
   Body :
   ```
   {
     "id_spacesuit": number
   }
   ```
8. POST /spacesuit/{spacesuitId}/update-vitals
   Body :
   ```
   {
     "cardiac_rythm": number,
     "o2_rate": number,
     "temperature": number,
     "pressure": number,
     "power": number
   }
   ```
9. PUT /spacesuit/{spacesuitId}/affect-astronaut
   Body :
   ```
   {
     "id_astronaut": number
   }
   ```
10. PUT /spacesuit/{spacesuitId}/remove-astronaut

11. GET /eva-mission
12. GET /eva-mission/{evaMissionId}/metrics ou
    GET /eva-mission/metrics
13. POST /eva-mission
    Body :
    ```
    {
      "id_mission": 0,
      "type": "string",
      "date_begin": "2022-11-13T16:58:29.373Z",
      "date_end": "2022-11-13T16:58:29.373Z",
      "supervisor": "string",
      "notes": "string",
      "spacesuits": [number]
    }
    ```
14. PUT /eva-mission/{evaMissionId}
    Body :
    ```
    {
      "id_mission": 0,
      "type": "string",
      "date_begin": "2022-11-13T16:58:29.373Z",
      "date_end": "2022-11-13T16:58:29.373Z",
      "supervisor": "string",
      "notes": "string",
      "spacesuits": [number]
    }
    ```

15. GET /task-planner
16. POST /task-planner
    Body :
    ```
    {
      "id_task": 0,
      "type": "string",
      "date_begin": "2022-11-13T17:00:43.112Z",
      "date_end": "2022-11-13T17:00:43.112Z",
      "astronauts": [number],
      "description": "string"
    }
    ```
17. PUT /task-planner/{taskPlannedId}
    Body :
    ```
    {
      "id_task": 0,
      "type": "string",
      "date_begin": "2022-11-13T17:00:43.112Z",
      "date_end": "2022-11-13T17:00:43.112Z",
      "astronauts": [number],
      "description": "string"
    }
    ```

18. GET /spacecraft ou GET /ava
19. POST /spacecraft
  Body :
  ```
  {
    "id_spacecraft": 0,
    "vitals": true,
    "status": "string",
    "id_resupplyMission": "string",
    "id_rotationMissions": ["string"]
  }
  ```
20. PUT /spacecraft/{spacecraftId}
    Body :
    ```
    {
      "id_spacecraft": number,
      "vitals": true,
      "status": "string",
      "id_resupplyMission": "string",
      "id_rotationMissions": ["string"]
    }
    ```

21. PUT /spacecraft/{spacecraftId}/launch
22. PUT /spacecraft/{spacecraftId}/affectSpaceCraftToResupplyMission ou PUT/spacecraft/{spacecraftId}/affectSpaceCraftToRotationMission
    Body :
    ```
    {
      "id_Mission": "string"
    }
    ```

23. GET /weather/current
24. GET /weather/prevision/{dateStart}/{dateEnd}

25. GET /astronaut
36. GET /astronaut
27. PUT /rotation-mission/{rotationMissionId}/affectSpacecraft
    Body :
    ```
    {
      "spacecraft_id": number;
    }
    ```
28. PUT /resupply/{resupplyMissionId}/affectSpacecraft
    Body:
    ```
    {
        "spacecraft_id": string;
    }
    ```

29. GET /survival-control/supervision
30. GET /survival-control/spacesuit-with-problem
31. PUT /survival-control/{moduleId}/isolate

32. 33. PUT /astronaut/{astronautId}/secure
34. GET /onMoonAstronauts ou GET /onEarthAstronaut
35. POST /astronaut
    Body :
    ```
    {
      "id_astronaut": 0,
      "name": "string",
      "isDead": true,
      "job": "string",
      "planet": "string",
      "location": "string"
    }
    ```

36. PUT /astronaut/{astronautId}
    Body :
```
{
  "id_astronaut": 0,
  "name": "string",
  "isDead": true,
  "job": "string",
  "planet": "string",
  "location": "string"
}
```

37. GET /rotation-mission
38. POST /rotation-mission
    Body :
```
{
  "spacecraft_id": "string",
  "date": "2022-11-13T17:21:53.615Z",
  "astronauts": [number]
}
```
39. PUT /rotation-mission/{rotationMissionId}
    Body :
```
{
  "spacecraft_id": "string",
  "date": "2022-11-13T17:21:53.615Z",
  "astronauts": [number]
}
```

40. POST /resupply/supply
    Body :
```
{
  "supplies": [
    {
      "label":"string"
      "quantity":0
    }
  ]
}
```
41. GET /resupply/resupplyMission
42. GET /resupply/supplyOrders
43. PUT /resupply/{supplyOrderId}/validate
44. PUT /resupply/{resupplyMissionId}/send

45. GET /module/vitals
46. et 59. PUT /module/{moduleId}/isolate

47. GET /moon-base/needs
48. GET /moon-base/inventory
49. GET /moon-base/{moonBaseId}

50. GET /airlock
51. GET /airlock/{moduleId}
52. POST /airlock
    Body :
```
{
  "id_airlock": 0,
  "id_module": 0,
  "external_door_open": true,
  "internal_door_open": true,
  "pressurized": true
}
```
53. PUT /airlock/{airlockId}/pressurize
54. PUT /airlock/{airlockId}/depressurize

55. POST /moon-base/pick
    Body :
```
[
  {
    "label":"string"
    "quantity":0
  }
]
```
56. GET /module/{moduleId}
57. GET /module/needs
58. GET /module/inventory
59. PUT /module/{moduleId}/isolate

60. GET /meteorite
61. GET /meteorite/danger
62. POST /meteorite
    Body :
```
{
  "id_meteorite": 0,
  "dangerous": true
}
```

63. GET /module
64. PUT /module/{moduleId}
    Body :
```
{
  "id_module": 0,
  "type": "string",
  "vitals": {
    "pressure": 0,
    "co2_rate": 0,
    "co2_scrubbers_activated": true
  },
  "supplies": [
     {
      "label":"string"
      "quantity":0
    }
  ],
   "astronauts": [number],
  "isolated": true
}
```
65. POST /module
    Body:
```
{
  "id_module": 0,
  "type": "string",
  "vitals": {
    "pressure": 0,
    "co2_rate": 0,
    "co2_scrubbers_activated": true
  },
  "supplies": [
     {
      "label":"string"
      "quantity":0
    }
  ],
  "astronauts": [number],
  "isolated": true
}
```

66. POST /module/{moduleId}/supply
    Body :
```
[
  {
      "label":"string"
      "quantity":0
    }
]
```

67. POST /moon-base
    Body :
```
{
  "initialStock": [
    {
      "label":"string"
      "quantity":0
    }
  ],
  "id_base": 0,
  "alarm_on": true,
  "listOfModuleIds": number[]
  }
```
68. PUT /moon-base/{moonBaseId}
    Body :
```
{
  "initialStock": [
    {
      "label":"string"
      "quantity":0
    }
  ],
  "id_base": 0,
  "alarm_on": true,
  "listOfModuleIds": number[]
}
```

```
69. POST /moon-base/{moonBaseId}/supply
    Body :
    [
     {
        "label":"string"
        "quantity":0
      }
    ]
70. PUT /moon-base/{moonBaseId}/isolate
  Body :
  {
    "id_base": 0,
    "stock": [
      {
        "label":"string"
        "quantity":0
      }
    ],
    "alarm_on": true,
    "listOfModuleIds": number[]
  }
```

```
71. GET /needs-control/moduleNeeds
72. POST /needs-control/sendOrder
    Body :
    {
      "supplies": [
        {
          "label":"string"
          "quantity":0
        }
      ]
    }
73. PUT /airlock/{airlockId}
    Body :
    {
      "id_airlock": 0,
      "id_module": 0,
      "external_door_open": true,
      "internal_door_open": true,
      "pressurized": true
    }

74. GET /news
```

Détail des services

Dans cette partie, nous allons décrire les micro-services que nous avons créé au cours du projet. Chaque description sera séparée en 4 parties :

- Rôle : Quel est l'objectif du micro-service ?
- Utilité : Pourquoi avons-nous fait le choix de créer un micro-service pour ce cas d'usage ?
- Données manipulées : Quelles sont les données qui transitent dans le micro-service et quelles manipulations fait-on sur la donnée ?
- Communications par bus : Si ce micro-service émet ou écoute un message d'un bus quelconque, lesquels et à quelle fin ?

ASTRONAUT-SERVICE

Rôle

Le rôle de ce micro-service est de gérer les astronautes. Un astronaute est une personne qui a été mandaté pour partir faire une mission sur la Lune. Un astronaute, à la base est sur la Terre, il va ensuite prendre une fusée pour rejoindre sa base sur la Lune (cf. AstronautPlanet).

Un astronaute peut avoir plusieurs rôles :

- Il peut être un astronaute classique (cf. AstronautJob.ASTRONAUT), ce sera donc un habitant "classique" sur la Lune, il aura quelques missions, notamment des missions d'exploration.
- Il peut être un astronaute commandant de bases lunaire (cf. AstronautJob.COMMANDER), dans ce cas-là, il aura des missions de commandement au sein de la base, par exemple il aura pour but de commander le réapprovisionnement de certains modules

Un astronaute peut évoluer dans plusieurs secteurs. On peut en distinguer 3 catégories de secteurs :

- Les modules (cf. Sector.*_MODULE) qui sont les endroits relatifs au bon fonctionnement de la base lunaire et à la vie sur celle-ci (exemple : les modules d'urgence - hôpitaux -, les modules scientifiques – modules de recherche -, les modules de vie …)
- Les secteurs d'exploration (cf. Sector.EXPLORATION_*) : ce sont les endroits dans lesquels les astronautes qui ont des sorties extravéhiculaires (EVA missions) évoluent. Ces missions sont généralement faites dans un but scientifique.
- L'espace d'embarquement et débarquement des fusées (cf. Sector/BOARDING_AREA) : cet espace sera utilisé par tous les astronautes à un moment durant leur séjour sur la base, lors des roulements de personnel sur la base ils devront se rendre dans cet espace pour embarquer dans une fusée en direction de la Terre et les astronautes arrivant y débarqueront. Les astronautes en charge du ravitaillement, **Jim** dans nos user stories, s'y rendra pour récupérer les produits acheminés par fusée pour ravitaillement du stock de la base.

Sur la Lune, à certains moments, un astronaute peut être soumis à certaines radiations. Et, pour que celui-ci n'ait pas de problèmes de santé, nous devons nous assurer que la durée d'exposition à ces radiations soit inférieure à un certain seuil. Nous pouvons traquer ce temps d'exposition grâce à l'attribut 'expositionTime'

L'astronaute à un attribut 'is_dead' qui décrit l'état de vie d'un astronaute. Si un astronaute venait à mourir, on passerait cet attribue à False. On peut modifier cet attribut par l'intermédiaire de Put("/astronaut/:astronautId") où Post("/astronaut")

Utilité

Ce micro-service, qui a pour but de représenter les entités astronautes, contient des attributs relatifs au comportement d'un véritable astronaute, comme son rôle sur la base lunaire, son nom, son temps d'exposition aux radiations, sa planète et dans le cas où il serait sur la Lune, le secteur sur lequel il se trouve. Il sert principalement à lier un astronaute à des combinaisons, tâches et missions.

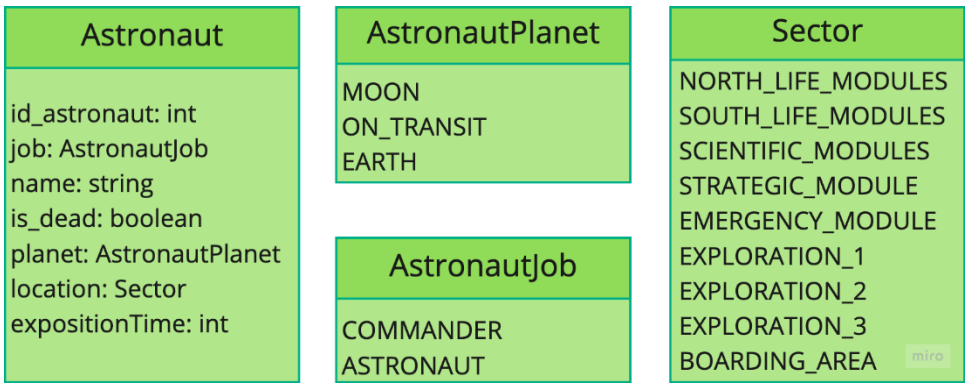
Cependant, nous avons fait un choix fort qui est d'y intégrer sa localisation, plus globalement, dans le modèle de donnée 'astronaut', nous avons inclut où il se trouve dans l'espace (est-t-il sur Terre ? Sur la Lune ? Dans la fusée ? Evolue-t-il dans une mission d'exploration ? …) . Tout ceci, nous aurions pu le mettre dans un autre micro-service qui aurait pu s'intituler 'astronaut-position'.

Cependant, nous ne trouvions pas de réel intérêt à créer cet autre micro-service, dans la mesure où nous ne faisons pas vraiment de réels calculs ni statistiques sur ces positions. Et également dans la mesure où ces positions font partie intégrante de la description d'un astronaute.

Interface REST

- GET /astronaut : Récupération de tous les astronautes créés
- POST /astronaut : Création d'un nouvel astronaute
- PUT /astronaut/{astronautId} : Mise à jour d'un astronaute
- PUT /astronaut/{astronautId}/secure : Déplacement d'un astronaute dans un module sécurisé
- GET /onMoonAstronauts : Récupération des astronautes qui sont actuellement sur la Lune
- GET /onEarthAstronauts : Récupération des astronautes qui sont actuellement sur la Terre

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service interagit avec le bus Kafka sous le clientId « astronaut ». Ce micro-service émet un message « astronaut-dead » lorsque l'on détecte que l'astronaute est mort (quand l'attribut 'is_dead' est à True). Dans le message, on intègre l'id_astronaut de l'astronaute qui est mort. Il écoute le topic « rotation-failed » et émet un message « astronaut-dead » pour tous les astronautes dont la mission a échoué. Il écoute également le topic « rotation-launched » afin que les astronautes passent en « ON_TRANSIT ».

SPACESUIT-SERVICE

Rôle

Ce micro-service est dédié à la surveillance du bon fonctionnement de l'équipement des combinaisons spatiales et aux métriques de l'astronaute qui la porte. Une combinaison ou « spacesuit » a son identifiant ainsi que l'identifiant de l'astronaute qui l'utilise. En plus de ces identifiants, la combinaison possède également une information « current_vitals » correspondant aux informations en temps réel relevées par les différents capteurs qu'elle possède.

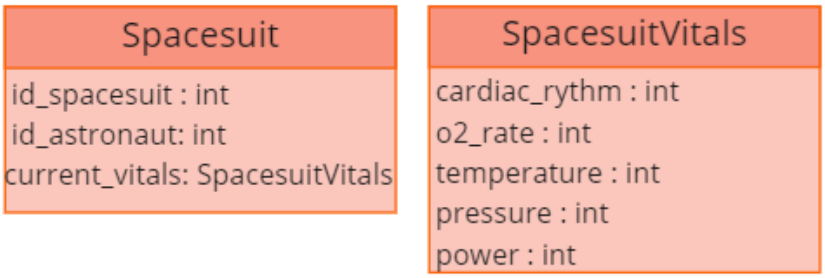
Utilité

Ce service va principalement servir à **Jim** pour la surveillance de ses signes vitaux de sa combinaison que nous avons imaginés comme directement consultables sur celle-ci.

Interface REST

- GET /spacesuit : Récupération des données d'une combinaison
- POST /spacesuit : Création d'une nouvelle combinaison
- PUT /spacesuit/{spacesuitId} : Modification des données de la combinaison d'id_spacesuit spacesuitId
- GET /spacesuit/{spacesuitId} : Récupération de la combinaison d'id_spacesuit spacesuitId
- POST /spacesuit/{spacesuitId}/o2sensor : Ajout de nouvelles données de taux d'oxygène de la combinaison d'id_spacesuit spacesuitId
- RECEIVE captors : Réception d'évènements Kafka des capteurs de combinaison qui serviront à la mise à jour des données de SpacesuitVitals
- EMIT vitals : Emission d'évènement Kafka des métriques SpacesuitVitals d'une combinaison

Modèle de données relatif à ce micro-service



Communications par bus

La communication par bus ici se fait par l'émission d'évènements du bus Kafka de topic « spacesuit-vitals » afin de mettre à jour les données collectées des capteurs listés dans le champ current_vitals de la combinaison.

SPACESUIT-MONITORING-SERVICE

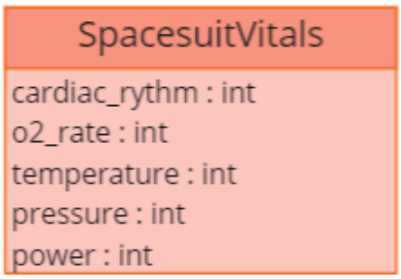
Rôle

Ce micro-service est destiné à écouter sur le topic « spacesuit-vitals ». Il captera un évènement toutes les secondes comprenant l'état actuel de la combinaison spacesuit. A chaque réception, il vérifiera si les vitals de la spacesuit sont correctes. Dans le cas ou elles ne le seraient pas, un évènement sera émis sur le topic « problem-spacesuit» avec uniquement les vitals problématique comme corp du message.

Utilité

Ce service sera utile dans le cadre de l'user story de **Jim** qui consiste à la surveillance des signes vitaux de la combinaison. En effet, si la combinaison pose un problème alors un évènement sera envoyé pour le préciser. Cet évènement contiendra l'id de la spacesuit qui a un problème et donc mettra au courant sont porteur. Cet évènement sera également utile à **Marie** afin qu'elle puisse tenir informés les personnes sur Terre en direct des nouvelles de la base lunaire.

Modèle de données relatif à ce micro-service



Communications par bus

Ce service écoute sur le topic « spacesuit-vitals » d'où il captera un évènement contenant les vitals d'une spacesuit. Si les vitals change de bonnes à mauvaise alors ce service émettra un évènement sur le topic « problem-spacesuit » avec un message contenant uniquement les champs mauvais de la spacesuit.

TASK-PLANNER-SERVICE

Rôle

Ce micro-service est relatif au besoin de garder un agenda de l'ensemble des tâches attribuées aux occupants de la base lunaire dans le but de planifier des diffusions de ces activités. Une tâche est identifiée par son identifiant, elle comporte également des données de type qui peuvent correspondre à 5 catégories de tâches dont 3 sont visibles dans les user stories développées : les tâche d'exploration « EXPLORATION » et d'aide « HELP » ainsi que les tâches de réapprovisionnement « RESUPPLY ». Une tâche possède également une date de début et une date de fin auxquelles vient s'ajouter la liste des astronautes en charge d'effectuer ladite tâche. Une description peut également être ajoutée à la définition de notre tâche.

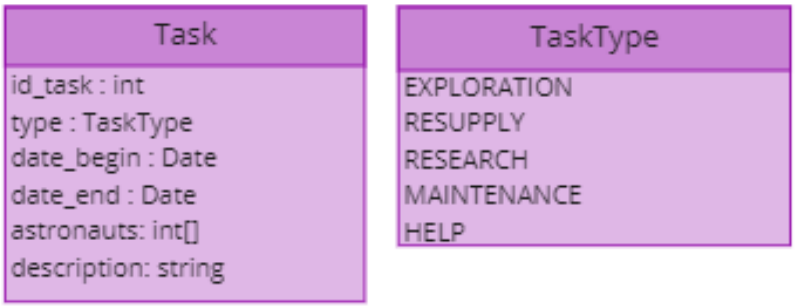
Utilité

Ce micro-service sera intéressant à utiliser dans le cas de la user story de **Buzz** selon laquelle un registre des tâches et missions de chaque astronaute présent sur la base doit lui être accessible. Comme vous le verrez plus loin, le format d'une tâche Task ressemble fortement au format d'une EVA Mission et pour cause, une EVA Mission correspond en effet à une tâche d'exploration « EXPLORATION » ou de secours à un astronaute en danger « HELP ».

Interface REST

- GET /task : Récupérer toutes les tâches définies
- POST /task : Créer une tâche
- PUT /task/{taskId} : Modification des données d'une tâche d'id_task taskId

Modèle de données relatif à ce micro-service



Communications par bus

Le micro-service Task écoute les évènements « eva-mission-created », lorsqu'un de ces messages est perçu, une tâche reprenant les informations de l'EVA mission envoyée est créée.

EVA-MISSION-SERVICE

Rôle

Ce micro-service a pour objectif l'organisation des missions EVA sur la Lune. Ce micro-service est en majorité utilisé par **Buzz**, **Marie** et **Jim**, il leur permet de consulter toutes les missions, ainsi que les détails de celles-ci. Une EVA mission est une mission qui est effectuée sur la Lune. Elle a un type (cf. EvaMissionType), qui peut être une mission d'analyse « SCIENTIFIC », ou une mission d'aide « HELP », dans le cas il faudrait aller envoyer des secours à un astronaute en danger.

Nous avons un attribut « supervisor », celui-ci doit contenir le nom de la personne qui supervise la mission. Dans notre cas, ce superviseur est généralement le commandant du village lunaire, Buzz.

En rapport à cette EVA mission, nous pouvons prendre des notes, la mission a également une date de début et une date de fin. Nous retrouvons aussi le secteur de la mission (cf. EVAMissionSector) qui permet aux superviseurs de voir dans quelles zones évoluent les astronautes. Ces attributs sont notamment utiles pour les superviseurs 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 EVA missions. En effet, même si une mission est toujours réalisée par un astronaute, nous ne pouvions pas stocker tous les détails de ces missions dans le micro-service relatif à l'astronaute, ni même au micro-service Spacesuit relatif aux combinaisons, car une mission est une sorte d'agrégat d'une combinaison. On ne peut pas avoir de combinaisons en dehors de la base sans qu'elle ne soit utilisée pour une mission. De plus, via la connexion avec le bus, les différents superviseurs pourront avoir des détails sur les missions sans pour autant connaître l'identité de l'astronaute qui l'a réalisée.

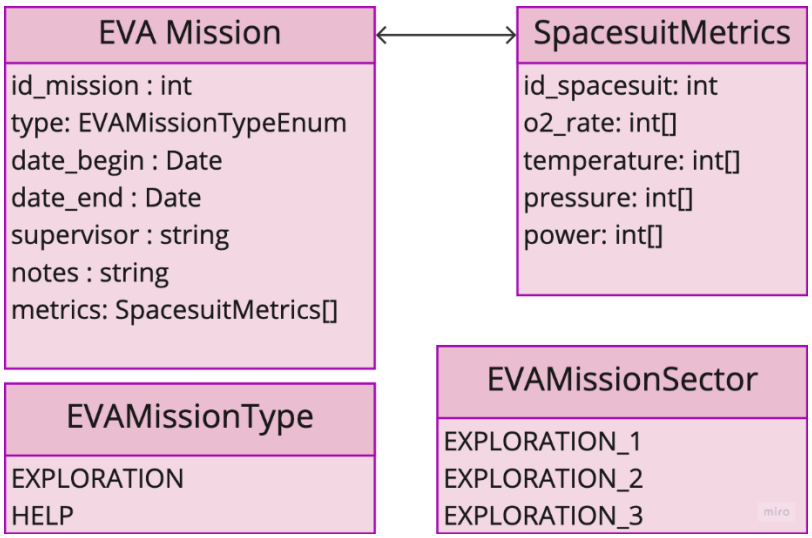
Nous avons également fait le choix de stocker les métriques de combinaisons spacesuitMetrics dans le service EVAMission et non pas dans le micro-service Spacesuit, car, les superviseurs analystes voudront analyser une mission, ce qui compose cette dernière, et plus particulièrement les métriques des combinaisons utilisées pendant les missions afin d'améliorer les combinaisons et leur utilisation.

Nous aurions pu rajouter 2 détails, qui peuvent être important, celui de l'heure exacte où l'on stocke une nouvelle métrique spacesuitMetric afin qu'au moment de l'analyse de la mission, on puisse savoir combien de temps s'est passé entre deux remontées d'information ainsi que la position en temps réel (dans spacesuitMetrics potentiellement), afin que l'analyse de la mission soit plus complète, cependant, comme il n'y a pas de GPS sur la Lune, nous avons supposé qu'il n'était pas réellement intéressant de l'intégrer.

Interface REST

- GET /eva-mission : Récupérer l'ensemble des données des EVA mission
- POST /eva-mission : Créer une EVA mission
- PUT /eva-mission/{evaMissionId} : Modifier les données d'une EVA mission d'id_mission evaMisssionId
- GET /eva-mission//{evaMissionId}/metrics : Récupérer les métriques des combinaisons utilisées pour une EVA mission d'id_mission evaMissionId
- GET /eva-mission/metrics : Récupérer les métriques de toutes les combinaisons pour les missions passées.

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service écoute les évènements inscrit dans le topic Kafka "spacesuit-vitals". Lorsqu'il y a des missions EVA, chaque spacesuit relève des données par l'intermédiaire de ses capteurs. Ces données doivent ensuite être transféré à la mission EVA, par l'intermédiaire de ce bus. A la réception de nouvelles valeurs, notre micro-service va chercher le spacesuit correspondant et va mettre à jour ses spacesuitMetrics.

MODULE-LIFE-SERVICE

Rôle

Le rôle de ce micro-service est de surveiller les besoins des modules lunaires et de la base lunaire en général. Globalement, un module est un bâtiment se trouvant sur la Lune et qui permet d'accueillir des astronautes dans des conditions de vie similaires à la Terre. Il possède donc une liste d'astronautes représentés par leur id, ainsi qu'un type parmi les quatre suivants définis par l'équipe :

- SAFETY : module sécurisé où les astronautes se trouvant sur la Lune viennent se réfugier en cas de problème de module ou d'alerte
- LIVING : module qui représente l'habitation des astronautes
- SCIENTIFIC : module de recherche où les astronautes font des expériences scientifiques (comme des laboratoires de recherche)
- WAREHOUSE : un module entrepôt dans lequel sont stockées les provisions

Afin d'assurer la sécurité du module, nous avons des capteurs qui nous fournissent les constantes du module, représenté par 'VitalsModule'. Nous y trouvons la pression du module, le niveau de CO_2, et si les épurateurs de CO_2 sont actifs ou non. Ces constantes peuvent être mises à jour par l'intermédiaire de POST /module/{moduleId}

Un module possède des ressources, représentées par 'Supply'. Une ressource du module a un libellé (par exemple "paquet de pate" ou "éprouvettes gradué") et une quantité (nombre de label en stock).

Dans la mesure où il agrège de nombreuses fonctionnalités, il interagit avec plusieurs micro-services et plusieurs persona. Il est notamment utilisé par **Deke** qui va surveiller l'état du module et par **Buzz** qui va vouloir observer les provisions restantes afin de potentiellement ordonner la création d'une mission de réapprovisionnement ResupplyMission.

Il peut arriver qu'il y ait un problème dans un module, c'est pourquoi nous avons la route '/isolate'. Cette route permet d'isoler un module, et de transférer les astronautes y vivant dans un autre module de type « SAFETY ».

Ce micro-service va être en relation avec le service MoonBase qui va agréger ce qu'on pourrait comparer à un village avec des une liste de modules.

Ce micro-service est également en relation avec le micro-service astronaute dans la mesure ou un module contient des astronautes, il est donc nécessaire d'avoir des informations sur ceux-ci (mais aussi de sécurisé les astronautes par l'intermédiaire du service MoonBase)

Utilité

Ce micro-service fut un des tout premier micro-service que nous avons conçu. Nous l'avons implémenté a la première itération du projet, au moment où nous n'avions pas encore le micro-service MoonBase, il a donc été assujetti a de nombreuses modifications, et comporte donc également certaines limitations, que nous avons essayé de combler, ou d'accepter par l'intermédiaire d'hypothèses.

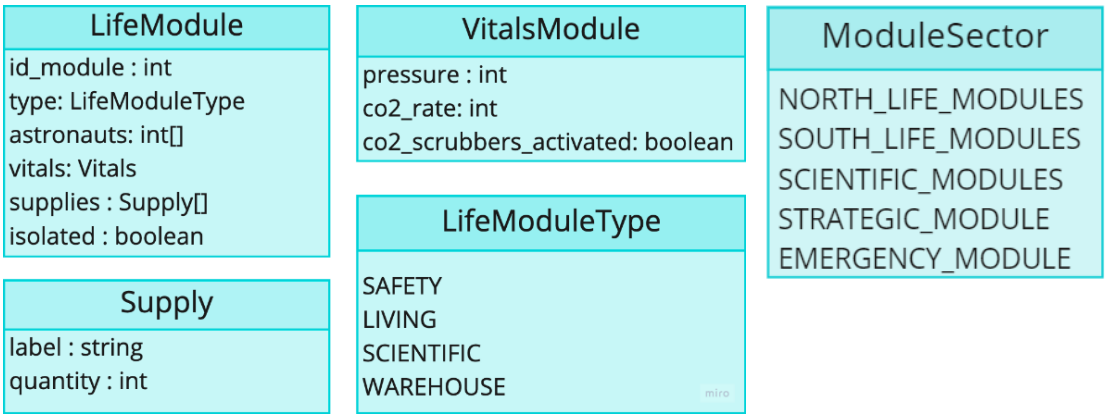
Par rapport aux ressources Supply, nous aurions pu créer une énumération, ou même une autre base de données qui contient les ressources potentiellement disponibles sur la Lune, pour cela nous aurions remplacé l'attribut "label" par cette valeur. Nous n'avons pas fait ce choix car nous le trouvions très restrictif, même s'il aurait pu être corrélé au métier. Par exemple, nous aurions pu créer un micro-service qui permet l'ajout de nouveaux items ou le retrait d'items des bases de données, et si jamais, pendant une mission d'exploration, un astronaute trouve un objet inconnu, a son arrivé, il appellerai ce micro-service et il donnerait un nom à cet objet, ce qui pourrait déclencher une procédure d'analyse dans un module scientifique ou autre. Mais dans la mesure où cette hypothèse n'était pas mentionnée dans les user stories, nous avons préféré conserver l'attribut "label" au format string, sans aucune contrainte.

Comme nous l'avons dit précédemment, nous pouvons mettre à jour les constantes 'VitalsModule' de la base par l'intermédiaire d'une route REST. Nous nous apercevons qu'il aurait été judicieux d'utiliser une approche événementielle à cette modification, dans la mesure où ces valeurs sont des capteurs, qui envoie des informations à intervalle de temps régulier. Nous avons cette implémentation car nous avons implémenté ce service avant l'implémentation du bus Kafka, et nous n'avions pas eu le temps d'implémenter l'approche évènementielle pour cette utilisation.

Interface REST

- GET /module : Récupérer les détails de tous les modules
- GET /module/vitals : Récupérer les constantes Vitals de tous les modules
- GET /module/needs : Récupérer les besoins de tous modules
- GET /module/inventory : Récupérer l'inventaire des ressources de tous les modules
- GET /module/{moduleId} : Récupérer toutes les informations relatives à un module
- POST /module : Créer un module
- POST /module/{moduleId}/supply : Ajouter de nouvelles ressources à un module
- PUT /module/{moduleId} : Modifier le contenu d'un module
- PUT /module/{moduleId}/isolate : Isoler un module (en cas de problème)

Modèle de données relatif à ce micro-service



Communications par bus

Nous n'avons aucune communication qu'il s'agisse d'émission ou de consommation de messages par bus dans ce micro-service.

SURVIVAL-CONTROL-SERVICE

Rôle

Ce micro-service a pour mission de contrôler (de manière passive ou active) les modules de vies. Il agit en tant qu'intermédiaire entre tous les modules sur la Lune, et quelqu'un qui souhaite surveiller et avoir une vue globale de l'état de ces modules.

Comme son nom l'indique, il a comme rôle de contrôler la survie sur la Lune. Il a donc trois rôles majeurs :

- Le premier de superviser les modules en les interrogeant via le micro-service Module Life (GET /vitals)
- Le second est d'isoler un module qui a un problème via le micro-service ModuleLife (PUT /{moduleId}/isolate)
- Le troisième est de gérer les combinaisons Spacesuit qui ont un problème (détaillé dans la partie « communication par bus »)

Utilité

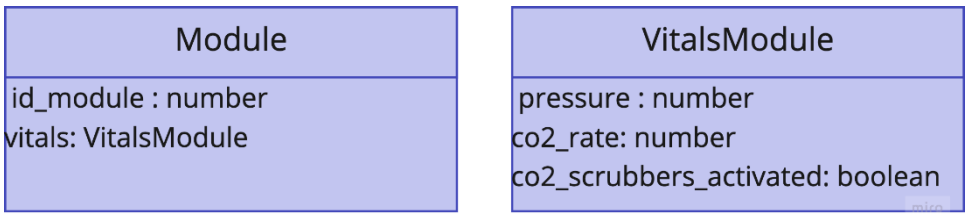
Ce micro-service est principalement utilisé par **Deke** et module-life-service. En effet, comme détaillé précédemment, Deke va vouloir surveiller l'état global de la base lunaire, elle va donc utiliser en majorité la route GET /vitals.

Ce micro-service à deux responsabilités majeures, la première est de surveiller les modules, et la seconde est de surveiller les spacesuits. Nous avons fait le choix de mettre ces deux responsabilités au sein du même micro-service car, dans la mesure ou dans les deux cas nous avons une supervision, il nous paraissait plus intéressant de les regrouper plutôt que de s'alourdir a tout séparer. Même pour Deke, c'est plus simple, car elle peut avoir une vision globale de ce qu'il se passe sur la Lune sans pour autant contacter plusieurs micro-services.

Interface REST

- GET /survival-control/supervision: Récupération du statut status de chacun des modules
- PUT /survival-control/{moduleId}/isolate : Isolation d'un module d'id_module moduleId
- GET /survival-control/spacesuit-with-problem: Récupération de la liste des combinaisons Spacesuit ayant un problème

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service écoute le topic « problem-spacesuit ». Lorsqu'il y a un problème avec une combinaisons (par exemple une dépressurisation non volontaire), le service Spacesuit Monitoring

Service écrit un message « problem-spacesuit » qui contient uniquement l'id de la spacesuit. A la réception dans notre micro-service, nous l'enregistrons, et ensuite, la liste de ces spacesuit qui ont un problème est accessible via la route GET /survival-control/spacesuit-with-problem/

MOON-BASE-SERVICE

Rôle

Ce micro-service a la charge du bon fonctionnement de l'ensemble de la base lunaire à travers la gestion de ses habitations, de ses ressources et de ses occupants. Ce micro-service va regarder nos besoins mais aussi isoler potentiellement un module à la suite d'un problème technique voire tous les modules en cas d'alerte. Il peut également regarder l'inventaire d'un module afin d'avoir un retour sur la quantité de provision restante au sein de notre module afin de garder un stock de ressources à jour.

Le micro-service est défini de manière à ce que l'on puisse mettre en place plus d'une base lunaire avec un identifiant de base lunaire, chaque base lunaire à une liste de modules et d'astronautes tous deux représentés par leurs identifiants, ainsi qu'un booléen alarm déterminant si l'alarme en cas de danger est en marche ou non et un stock de ressources correspondant à une liste de ressources Supply.

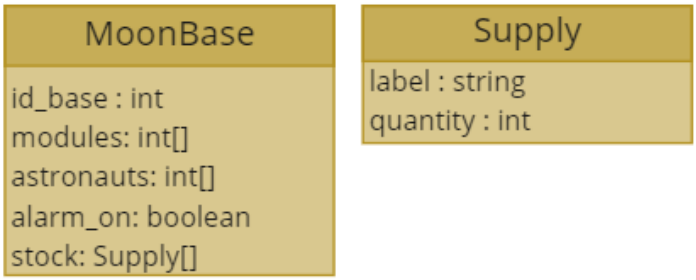
Utilité

Ce micro-service est principalement utilisé par **Buzz** et **Jim** dans le cadre des user stories de demande de création de missions de ravitaillement et de vérification des stocks. Ajoutons à cela l'isolation de l'ensemble des modules en cas d'alerte de risque de météorites détecté par le micro-service d'observation de ces dernières.

Interface REST

- GET /moonBase/{moonBaseId} : Récupération des données relatives à une base lunaire
- GET /moonBase/inventory : Récupération des données relatives aux stock de la base lunaire
- GET /moonBase/needs : Récupération de la liste des besoins générale en réunissant celles de chacun des modules de la base lunaire
- POST /moonBase : Création d'une base lunaire
- POST /moonBase/{moonBaseId}/supply : Création de nouvelles ressources à ajouter à la liste de ressources du stock de la base lunaire à l'arrivée d'une mission de réapprovisionnement
- POST /moonBase/{moonBaseId}/pick : Retirer un élément du stock d'une base lunaire MoonBase (par exemple quand un habitant vient chercher des pâtes à l'entrepôt, on retire un paquet de pâtes du stock de la base lunaire)
- PUT /moonBase/{moonBaseId} : Modification des données de la base lunaire
- PUT /moonBase/{moonBaseId}/isolate : Isolation de l'ensemble des modules en cas de d'alerte météorites

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service écoute le topic « dangerous-meteorite ». Lorsqu'une météorite dangereuse pour la MoonBase est détectée (ce message provient de Meteorite Monitoring Service), notre micro-service va isoler l'ensemble des bâtiments de la base, c'est-à-dire qu'elle va isoler tous les modules en appelant le service ModuleLife (PUT /{idModule}/isolate), et elle va mettre en marche l'alarme de la base en passant son attribut alarm_on à True.

NEEDS-CONTROL-SERVICE

Rôle

Ce micro-service a pour rôle d'afficher les besoins de la base lunaire et de passer des commandes pour envoyer du matériel ou des provisions. Il devra alors faire une requête vers le service MoonBase pour avoir l'information des besoins, et une autre requête vers Resupply Service pour passer commande.

Un besoin est défini par un NeedsDTO qui possédant une liste de SupplyDTO. Chaque SupplyDTO correspond à un label, l'objet manquant, et la quantité qu'il manque. De même, chaque commande sera définie par un SupplyOrderDTO identique à NeedsDTO.

Utilité

Ce micro-service offre une interface pour récupérer les besoins de la base lunaire. Il permet aussi de passer des commandes au service de ravitaillement.

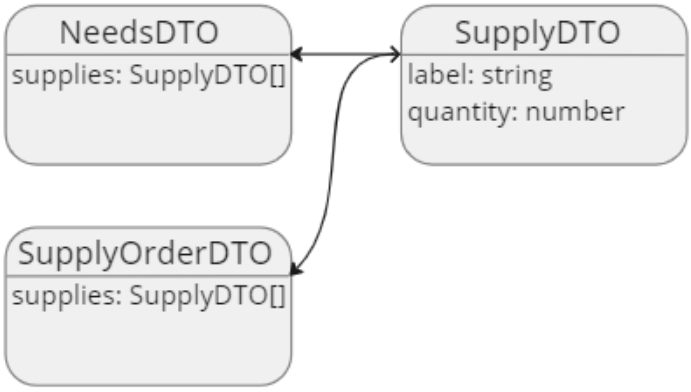
Il permet de répondre en partie au besoin de **Buzz** dans l'observation et le contrôle de l'état des modules de la base. Dans ce cas, l'états corresponds à l'états des provisions des modules de la base et donc au besoin de la base lunaire. De plus Buzz souhaitait pouvoir effectuer les changements nécessaires pour soutenir l'équipage, le fait de passer une commande réponds totalement à ce besoin, car Buzz pourra utiliser ce service pour passer une commande et permettre à l'équipage de ne pas y penser.

Interface REST

- GET /moduleNeeds : récupérer les besoins des modules de la base lunaire
- POST /sendOrder : envoyer une commande vers Resupply Service

Traitement des données relatives à ce micro-service

Ce micro-service ne stocke pas de données en base de données, mais vas gérer trois types de données : NeedsDTO, SupplyDTO et SupplyOrderDTO



Communications par bus

Nous n'avons aucune communication qu'il s'agisse d'émission ou de consommation de messages par bus dans ce micro-service.

RESUPPLY-SERVICE

Rôle

Ce micro-service sert à la création et gestion des missions de réapprovisionnement. On y trouve deux types d'informations, les missions de réapprovisionnement elles-mêmes et les demandes de ressources qui lui sont passées.

Une ressource Supply est un élément simple correspondant à un libellé de ressource label et une quantité demandée. Chaque demande en ressources SupplyOrder possède un identifiant afin de pouvoir être identifiée et liée à une mission de réapprovisionnement, ainsi que la liste des ressources qui y sont demandées. Enfin, elle contient un statut correspondant à une prise en compte, refus ou préparation de cette commande. Une demande de ressources ne peut passer en préparation qu'après avoir été préalablement acceptée.

Une mission de réapprovisionnement ResupplyMission possède un identifiant afin d'être liée à une fusée Spacecraft pour acheminement. Elle possède également des informations de statut ResupplyMissionStatus pour savoir où en est la livraison de nos demandes de ressources. Une liste de demandes de ressources SupplyOrder lui est enfin liée. Tant que la mission de réapprovisionnement a un statut à « PREPARING » on peut y ajouter de nouvelles demandes de ressources. Si aucune mission de réapprovisionnement n'a son statut à cette valeur car elles se trouvent soit en cours d'acheminement « TRAVELING » soit qu'elle est arrivée à destination « DONE », une nouvelle mission de réapprovisionnement est créée. Enfin un identifiant de fusée est passée à la mission de réapprovisionnement, en effet dans le cas d'un échec de l'acheminement de la fusée Spacecraft, on veut pouvoir relancer la mission de réapprovisionnement en repassant son identifiant de fusée à null et son statut à « PREPARING ».

Utilité

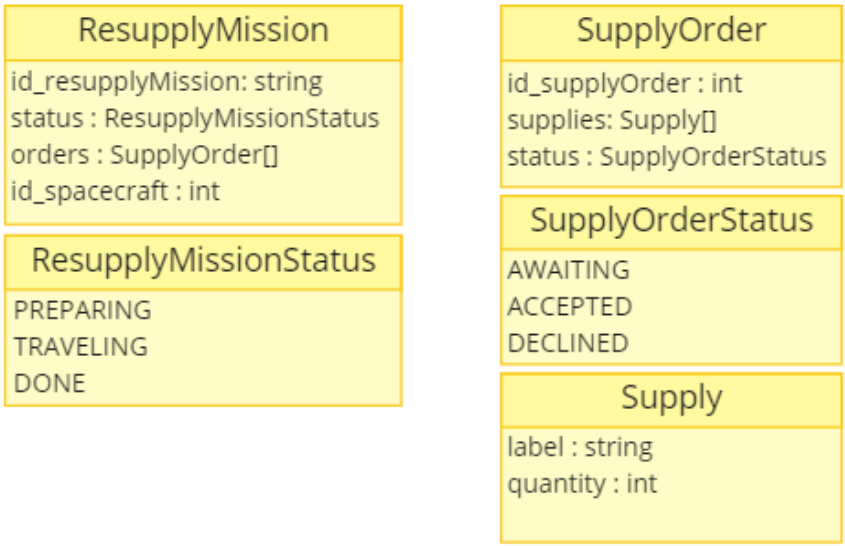
Ce micro-service permet de répondre à divers besoins :

- L'organisation et le suivi de missions de réapprovisionnement par **Dorothy**
- L'affectation d'une fusée à une mission de réapprovisionnement par **Gene** même s'il n'est pas fait par ce micro-service est visible ici à travers l'attribut id_spacecraft de la mission de réapprovisionnement
- La programmation de missions de ravitaillement dans le cas d'échec catastrophique lors de l'acheminement d'une fusée par **Gene**
- La notification d'un retard de mission de réapprovisionnement à **Jim**

Interface REST

- GET /resupply/supplyOrder : Récupération des demandes de ressources
- POST /resupply/supply : Création d'une demande de ressource
- PUT /resupply/{supplyOrderId}/validate : Modification du statut d'une demande de ressources pour la valider et en débuter la préparation
- GET /resupply/resupplyMission : Récupération des missions de réapprovisionnement
- PUT /resupply/{resupplyMissionId}/send : Modification du statut d'une mission de réapprovisionnement pour signifier son envoi

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service écoute le topic « spacecraft-damaged ». Lorsqu'une fusée est endommagée, la mission de réapprovisionnement repasse en statut « PREPARING » et son id de fusée à null. Il écoute également le topic « spacecraft-launch ». Lorsqu'une fusée est envoyée, le statut de la mission de réapprovisionnement passe en « TRAVELING »

ROTATION-MISSION-SERVICE

Rôle

Ce micro-service a la charge de la gestion des vols des astronautes depuis et vers la Lune des astronautes afin de mettre en place rotations de personnel de la base lunaire. Cette mission de rotation est liée à une fusée spacecraft pour assurer le transport des astronautes. Une mission de rotation est identifiée par son identifiant, elle possède également l'information de la fusée à laquelle elle est liée via l'identifiant de celle-ci. La mission de rotation possède une date afin que les astronautes puissent s'organiser pour leur changement de planète et finalement la liste des astronautes qui seront transférés à travers la liste de leurs identifiants.

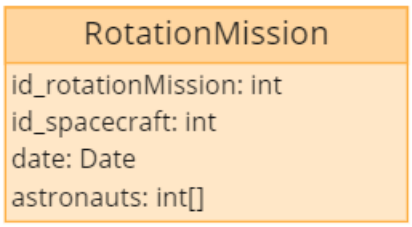
Utilité

Ce micro-service répond au besoin énoncé par **Gene** de mettre en place des rotations de personnel sur la base lunaire et du fait de faire voler des astronautes et non plus seulement des ressources Supply comme nous le faisions plus tôt dans le projet.

Interface REST

- GET /rotation/rotationMission : Récupérer les missions de rotations existantes
- POST /rotation/ rotationMission : Créer une mission de rotation

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service écoute le topic « spacecraft-damaged ». Lorsqu'une fusée est endommagée, son id de fusée passe à null et il émet un évènement « rotation-failed » indiquant que la mission a échoué. Il écoute également le topic « spacecraft-launch » afin d'émettre un event « rotation-launched » indiquant que la mission a été envoyé dans l'espace.

SPACECRAFT-SERVICE

Rôle

Ce micro-service a pour rôle la gestion des fusées aussi appelées « spacecrafts ». Une fusée contient différentes informations telles que son identifiant, sa destination, son statut et une date prévisionnelle de départ. Elle contient également la référence de la mission de ravitaillement à son bord et la liste des missions de rotations qui lui sont attribuées.

Une fusée en partance de la Lune à destination de la Terre sera toujours une mission de rotation.

Lorsqu'une fusée est chemin « TRAVELING », les astronautes de la première mission de rotation qui y est associée voient la valeur de leur champ planet passer à « ON_TRANSIT ».

Lorsqu'une fusée rencontre un problème en cours d'acheminement, son statut passe de « TRAVELING » à « DAMAGED ».

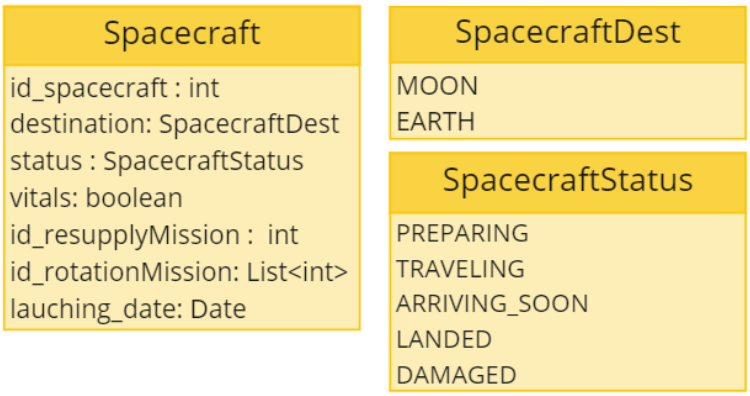
Utilité

Il permettra à **Marie**, **Buzz** et **Jim** d'être tenu au courant de l'avancée dans l'acheminement de missions de réapprovisionnement ou de rotation pour les statuts « PREPARING », « TRAVELING », et « LANDED », à **Jim** de se préparer à recevoir une livraison avec le statut « ARRIVING_SOON » . Dans le cas où la fusée rencontrerait un problème au point de la rendre irrécupérable, son statut serait passé à « DAMAGED » et servirait alors à **Gene** à relancer la préparation d'une mission de réapprovisionnement de même contenu que celle détruite.

Interface REST

- GET /spacecraft : Récupération des données des différentes fusées existantes
- POST /spacecraft : Création d'une nouvelle fusée
- PUT /spacecraft/launch : Changement de l'état d'une fusée lors de son lancement
- GET /spacecraft/{spacecraftId} : Récupération des données d'une fusée d'id_spacecraft spacecraftId

Modèle de données relatif à ce micro-service



Communications par bus

Un évènement est émis sur le bus Kafka lors du changement de statut SpacecraftStatus d'une fusée. Le but de notre communication asynchrone est d'aller avertir les différents intervenants du statut de la fusée pour suivre les missions qui y sont associées, ou si la fusée a été endommagée.

Lorsque nous détectons qu'il y a un problème d'ordre technique (i.e. qu'elle s'est détruite) sur la fusée passant son statut SpacecraftStatus à « DAMAGED », nous émettons un message sur le topic « spacecraft-damaged » avec comme valeur l'id_spacecraft de la fusée en charge de la resupply mission, ce message pourra donc être pris en compte par le service de Resupply qui devra recommencer la préparation de la mission de réapprovisionnement pour la Lune avec une autre fusée.

Lorsque la fusée est officiellement en route vers sa destination, elle émet un message sur le topic « spacecraft-launched», qui permet à Resupply Service de savoir qu'une fusée accueillant une mission de ravitaillement ou/et de rotation du personnel est bien été lancée. De plus, le micro-service News

Formalisation Service consomme également ce message, car ce micro-service a pour rôle de jouer d'interface entre les missions lunaires et le grand public, il doit donc savoir quand une fusée a été lancé afin de faire un communiqué de presse.

Notre micro-service émet des messages au sein du topic 'spacecraft-arriving' et 'spacecraft-landed' afin que le micro-service Spaceccaft Monitoring Service soit au courant du statut de la fusée, car ce micro-service a pour objectif de suivre toutes les fusées.

SPACECRAFT-MONITORING-SERVICE

Rôle

Ce service sert au suivi des fusées existantes, il permet de traiter les évènements relatifs aux fusées en écoutant sur les topics : « spacecraft-landed », « spacecraft-launch » et « spacecraft-arriving ». A chaque message, on garde en mémoire dans des listes les fusées qui arrive ou celles qui ont atterri.

Utilité

Ce micro-service sert pour la user story de **Jim** qui demande de savoir quand un vaisseau est sur le point d'atterrir car il va permettre de récupérer tous les vaisseaux arrivants ou atterrie.

Interface REST

- GET /landed : Récupération de l'ensemble des vaisseaux atterries.
- GET /arriving : Récupération de l'ensemble des vaisseaux arrivant.

Communications par bus

Ce service écoute sur les topics « spacecraft-landed », « spacecraft-launch » et « spacecraft-arriving ». Le but est de faire varier la liste dans laquelle se trouve un vaisseau à chaque événement capté. Ainsi on a une liste toujours à jours des vaisseaux arrivé et atterrie.

METEORITE-MONITORING-SERVICE

Rôle

Ce micro-service possède une implémentation assez simple car la user story y faisant allusion l'est aussi. Deux informations y sont liées, l'identifiant de météorite et un booléen signifiant si elle représente un danger pour la base lunaire ou non.

Il pourrait à l'avenir être complété avec une donnée de distance de la météorite à la base.

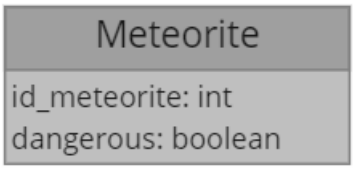
Utilité

Ce micro-service répond au besoin de suivi des météorites énoncé par **Deke** mais participe également indirectement à celui d'alerte en déplacement en lieu sûr des astronautes à travers les messages qu'il émet sur le bus Kafka.

Interface REST

- GET /meteorite : Récupération des données de l'ensemble des météorites existantes
- POST /meteorite : Création d'une nouvelle météorite
- PUT /meteorite/{meteoriteId} : Modification du champ « dangerous » d'une météorite d'id_meteorite meteoriteId

Modèle de données relatif à ce micro-service



Communications par bus

Lors du changement du passage de l'attribut booléen « dangerous » d'une météorite Meteorite à True, un event « dangerous-meteorite » est émit sur le bus Kafka. Il sera écouté par le service de la base lunaire MoonBase afin de déclencher l'alarme, transférer l'ensemble des astronautes sur la Lune vers le module sécurisé et isoler l'ensemble des modules.

NEWS-FORMALISATION-SERVICE

Rôle

Ce micro-service a pour rôle de formaliser les messages écoutés sur les topics « problem-spacesuit », « astronaut-dead », « spacecraft-assigned »,<« spacecraft-damaged », « spacecraft-launched » et « dangerous-meteorite ». Il permet de créer un évènement plus lisible pour un humain et émets cet évènement dans le topic « news ». Ce topic « news » corresponds à tous les évènements intéressant à remonter du système pour potentiellement les publier.

Utilité

Ce micro-service est utile pour **Marie**, car c'est lui qui va remonter les évènements intéressants dans un langage humainement compréhensible afin de pouvoir relayer l'information au public.

En effet, Marie demande à être informé et notifié des évènements importants afin de pouvoir les relayer au public. Ce service lui permet non seulement d'être au courant de chaque évènement mais en plus de les comprendre et de les traiter comme elle le souhaite.

Communications par bus

Ce service écoute sur les topics « problem-spacesuit », « astronaut-dead », « rotation-failed », « spacecraft-damaged », « spacecraft-launched » et « dangerous-meteorite ». A chaque fois, il récupèrera un évènement brut avec des données brutes du système. Après traitement de ces messages, il émettra sur le topic « news » un évènement de haut niveau contenant un message lisible pour un humain.

NEWS-SERVICE

Rôle

Ce micro-service a pour rôle de récupérer les données en écoutant les évènements sur le topic « news » et de les stocker dans une base de données.

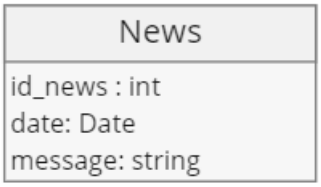
Utilité

Ce micro-service nous permets de stocker tous les messages envoyés sur le topic « news ». C'est utile pour **Marie** car c'est le micro-service qu'elle utilisera pour savoir quels évènements il y a eu et quand.

Interface REST

- GET /news : Récupération de l'ensemble des news stocker quand la base de données.

Modèle de données relatif à ce micro-service



Communications par bus

Ce micro-service écoute sur le topic « news », il va recevoir des évènements donc le message contient toujours du texte et ensuite stocker ce message dans la base de données avec la date de stockage.

WEATHER-MONITORING-SERVICE

Rôle

Ce micro-service n'a pas été implémenté dans notre code car il appartient aux user-stories additionnelles. Cependant, nous allons détailler comment nous l'aurions fait si nous voulions l'implémenter.

Ce micro-service a pour principal rôle de regarder les conditions météorologiques afin de repousser un lancement de fusée. Afin de réaliser ce micro-service, nous aurions implémenter un système de capteurs qui permet de prévoir la météo, ainsi que d'observer la météo courante, et, par l'intermédiaire de routes (décrites dans la partie Interface REST), les différents intervenants peuvent observer cette météo.

Utilité

Ce micro-service serait utilisé en majorité par **Gene**, qui de manière globale gère le lancement des fusées. Si nous souhaitons pousser les choses un peu plus loin, il pourrait également être utilisé par Spacecraft avant un vol, pour potentiellement avertir les astronautes au sein de la fusée des conditions météorologiques durant le vol.

Interface REST

- GET /weather/current : Récupérer la météo courante dans une zone particulière
- GET /weather/prevision/{dateStart}/{dateEnd} : Récupérer la prévision météorologique pour des dates spécifiques

Communications par bus

Afin d'avoir la météo en temps réelle, nous pouvons supposer que les capteurs émettront des événements dans un bus, et qu'ensuite, le micro-service stockerait ces données afin de faire des prédictions futures.

Ce micro-service pourrait avoir un système de météo en temps réel par l'intermédiaire d'une approche événementielle. Par exemple, nous pourrions faire en sorte que les prévisions météorologiques pour la prochaine heure soient postées dans ce bus, afin que Gene n'ait pas a faire une requête toutes les secondes pour connaître la météo.

AIRLOCK-CONTROL-SERVICE

Rôle

Ce micro-service, comme le précédent, n'a pas été implémenté car il ne fait pas partie de la liste des user stories que nous avons choisi de traiter.

Ce service correspond au sas de pressurisation des modules. Nous partons du principe qu'un module peut avoir plusieurs entrées et ainsi plusieurs sas de pressurisation. Ainsi, un airlock possède un identifiant et l'identifiant de module auquel il est associé. Un airlock possède deux portes distinctes :

- Une porte externe liant l'extérieur et l'intérieur du sas de pressurisation
- Une porte interne liant l'intérieur du sas de pressurisation et l'intérieur du module

Pour ces deux portes, le service possède des booléens signalant si elles sont ouvertes. Voici l'influence de l'ouverture des portes sur l'attribut « pressurized » de airlock définie par l'équipe et nous signifiant que le sas fonctionne correctement sur un exemple d'entrée dans le module depuis l'extérieur :

- Les deux portes sont fermées : le sas est pressurisé
- La porte externe est ouverte afin de faire rentrer l'astronaute, la porte interne reste fermée : le sas n'est alors pas pressurisé
- La porte externe est refermée afin de pouvoir pressuriser à nouveau le sas : on repasse dans la configuration des deux portes fermées et du sas pressurisé
- La porte interne est alors ouverte et la porte externe reste fermée : le sas est pressurisé, l'astronaute peut accéder au module, on referme la porte à son entrée dans le module

Utilité

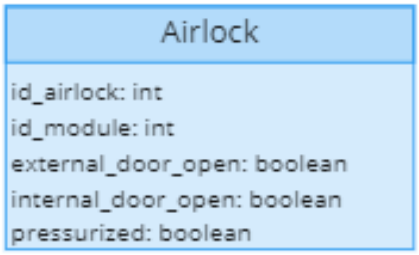
Ce micro-service répond au besoin énoncé par **Deke** de mise en place de sas de pressurisation et de leur gestion de la pressurisation et de la dépressurisation de ceux-ci. Ne considérant pas la base lunaire comme des modules tous reliés entre eux par des tunnels ou couloirs pressurisés et dans l'éventualité de modules isolés, nous avons préféré

Interface REST

Les interfaces REST imaginées pour ce service sont les suivantes :

- GET /airlock : Récupération des informations de tous les sas de pressurisation des modules
- GET /airlock/{moduleId} : Récupération des informations de tous les sas de pressurisation rattachés à un module d'id_module moduleId
- POST /airlock : Création d'un sas de pressurisation
- PUT /airlock/{airlockId} : Modification des informations d'un sas de pressurisation d'id_airlock airlockId principalement utilisé lors des ouvertures et fermetures de portes
- PUT /airlock/{airlockId}/pressurize : Modification des informations d'un sas de pressurisation d'id_airlock airlockId pour pressuriser le sas dans le cas d'une entrée dans le module depuis l'extérieur
- PUT /airlock/{airlockId}/depressurize : Modification des informations d'un sas de pressurisation d'id_airlock airlockId pour dépressuriser le sas dans le cas d'une sortie de module vers l'extérieur

Modèle de données relatif à ce micro-service



Communications par bus

Pour ce service, nous pouvons supposer que des capteurs existent pour rendre compte de l'état du airlock. Ces capteurs mettent à jour l'état du airlock du module. Ensuite, ce micro-service utilisera les données des capteurs du airlock pour détecter un problème au niveau des airlocks. Dans ce cas, on émettrait un évènement sur le topic "problem-airlock" afin de signaler un problème de pressurisation au niveau du airlock.

Scénarios

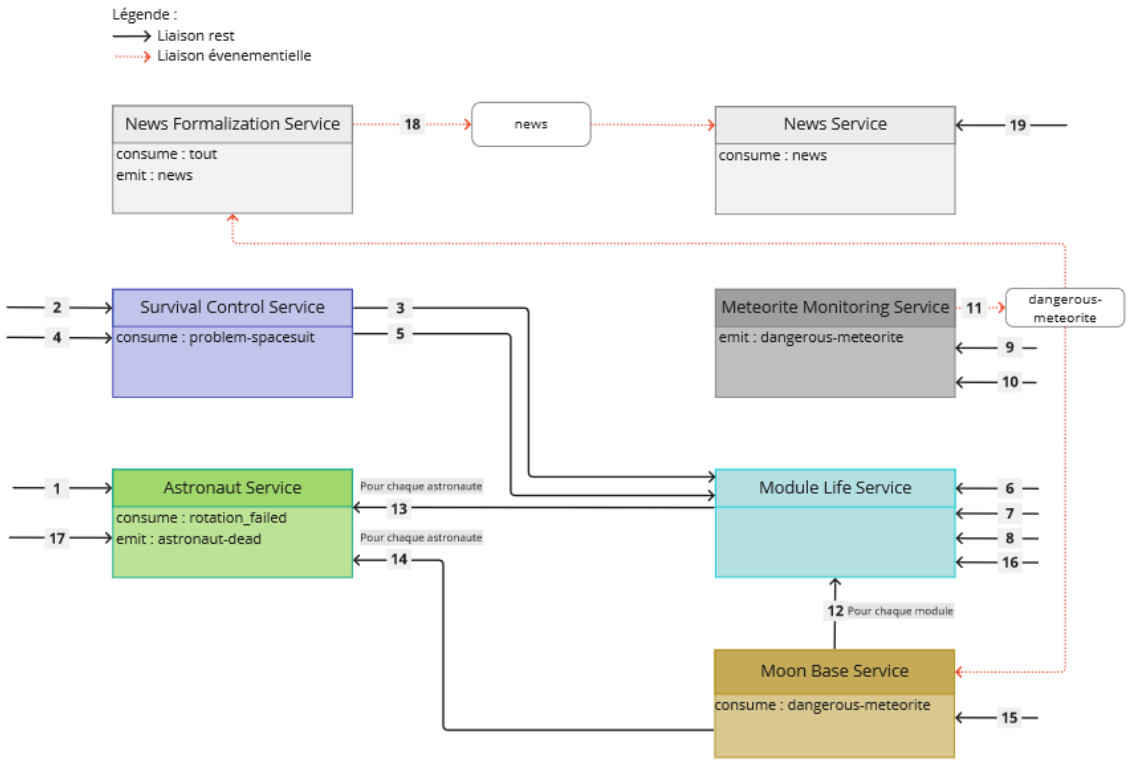
SCENARIO 1 – GESTION DE LA SECURITE DES MODULES ET DES ASTRONAUTES DE LA BASE LUNAIRE DANS LE CAS D'UNE METEORITE DANGEUREUSE EN APPROCHE

Tous les matins, en arrivant au centre de commandement, Deke demande à voir la position des astronautes et l'état des modules. Si tout est normal du côté des astronautes, il remarque cependant que le module 512 a une pression beaucoup trop faible et décide de l'isoler. Pendant ce temps, Jim a reçu une alerte et veut vérifier que les épurateurs de CO_2 du module 514 fonctionnent correctement. Deke observe ensuite les météorites autour de la base, mais une météorite dangereuse vient d'être détectée. Une alerte a déjà été lancée automatiquement pour que les astronautes se déplacent dans le module sécurisé et l'ensemble des modules est isolé. Deke demande à vérifier l'état des modules pour vérifier qu'ils sont en effet isolés et que tous les astronautes sont en sécurité. Finalement, Marie s'informe également des dernières nouvelles en provenance de la Lune et constate qu'une alerte générale a été lancée à cause d'un astéroïde.

User stories couvertes dans ce scénario

Les user stories 1, 4, 9, 13, 14, 15 et 17 sont traitées ici.

Diagramme de séquence



Déroulé du scénario

1. Requête **GET** sur /onMoonAstronauts de Astronaut Service de Deke afin de s'informer de la position des astronautes. Il récupère toutes les informations de tous les astronautes sur la Lune.
2. Requête **GET** sur /survival-control/supervision de Survival Control Service de Deke afin de s'informer des conditions de vie des modules.
3. Requête **GET** sur /module/vitals de Module Life Service de Survival Control Service. Module Life service renvoie uniquement les informations des conditions de vie de tous les modules de la base lunaire et Survival Control retransmet ces informations en réponse de la requête précédente.
4. Requête **PUT** sur /survival-control/{moduleId}/isolate de Survival Control Service de Deke pour isoler le module 512 après avoir constaté une pression insuffisante.
5. Requête PUT sur /module/{moduleId}/isolate de Module Life Service de Survival Control Service pour isoler le module 512.
6. Requête **GET** sur /module/{moduleId} de Deke afin de vérifier que le module est bien isolé.
7. Requête intermédiaire PUT sur /module/{moduleId} de Module Life Service afin de modifier le taux de co2 du module 514. Il s'agit de simuler la mise à jour de la valeur par un capteur.
8. Requête **GET** sur /module/{moduleId} de Module Life Service de Jim afin de vérifier l'activation des épurateurs de CO_2 du module 514.
9. Requête **GET** sur /meteorite/danger de Meteorite Monitoring Service de Deke afin de vérifier si une chute de météorite menace la base lunaire.
10. Requête intermédiaire **POST** sur /meteorite de Meteorite Monitoring Service afin de simuler l'action d'un scientifique ou d'un capteur ayant détecter une nouvelle météorite.
11. Emission d'un event *dangerous-meteorite* par Meteorite Monitoring Service à la suite de l'ajout d'une météorite dangereuse. Ecoute de l'event *dangerous-meteorite* par Moon Base Service et déclenchement de l'alarme générale dans Moon Base. Ecoute de l'event *dangerous-meteorite* par News Formalization Service.
12. Requête **PUT** sur /module/{moduleId}/isolate de Module Life Service de Moon Base Service pour chaque module de la base lunaire afin de l'isoler à cause du déclenchement de l'alarme générale.
13. Requête **PUT** sur /astronaut/{astronautId}/secure de Astronaut Service de Module Life Service afin de diriger les astronautes du module dans le module sécurisé.
14. Requête **PUT** sur /astronaut/{astronautId}/secure de Astronaut Service Moon Base Service afin de diriger les astronautes de la base lunaire dans le module sécurise à cause du déclenchement de l'alarme générale.
15. Requête **GET** sur /moon-base/{moonBaseId} de Moon Base Service de Deke afin de vérifier que l'alarme générale est en cours.
16. Requête **GET** sur /module de Module Life Service de Deke afin de vérifier que tous les modules sont isolés et que tous les astronautes se trouvent dans le module sécurisé.
17. Requête **GET** sur /onMoonAstronauts de Astronaut Service de Deke afin de vérifier que les astronautes sont bien dans le secteur EMERGENCY_MODULE.
18. Après l'écoute de l'event dangerous-meteorite par News Formalization Service, le service émet un event news contenant les raisons de l'isolement de la base lunaire. Ecoute de l'event news par News Service qui le sauvegardera.
19. Requête **GET** sur /news de News Service de Marie afin de s'informer des derniers évènements.

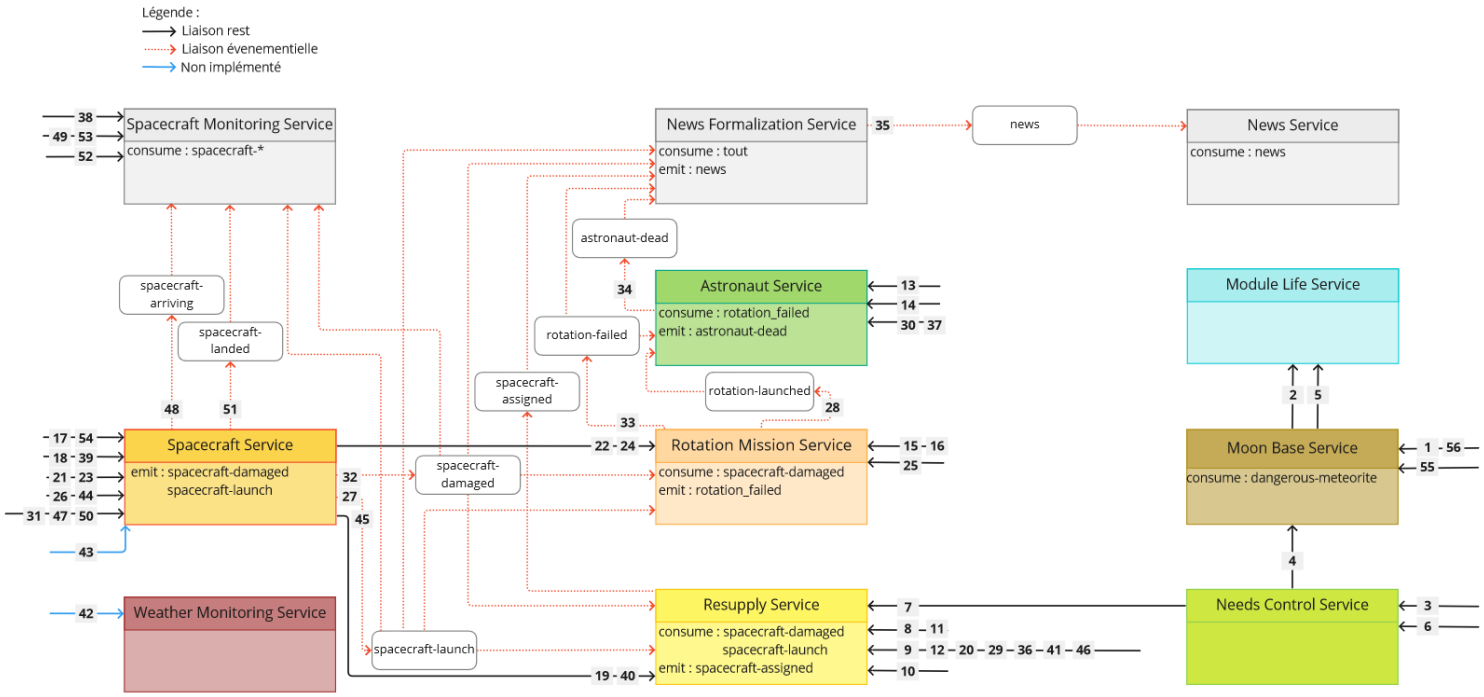
SCENARIO 2 – REAPPROVISIONNEMENT DE LA BASE ET ROULEMENT DES ASTRONAUTES

Jim veut vérifier l'inventaire de la station. Il constate que les stocks commencent à se faire bas et il en réfère à Buzz. Buzz vient alors demander la liste des besoins de la station. Il passe commande à la Terre. Dorothy reçoit sa commande et la valide, la faisant alors partir avec la prochaine mission de ravitaillement. Pendant ce temps, Gene organise la prochaine mission de renouvellement des astronautes. Il lie tout d'abord des astronautes sur Terre à une première mission, puis des astronautes de la Lune (sauf le commandant) à une deuxième mission. La construction d'un nouveau vaisseau spatial vient de se terminer. Gene attribue ce nouveau vaisseau spatial prêt à décoller à la mission de ravitaillement ainsi qu'aux deux missions destinées au roulement des astronautes et le fait décoller. Malheureusement, le lanceur s'appelait Ariane 5, le vaisseau explose en plein vol, les astronautes décèdent et la mission de ravitaillement est à recommencer. Jim est prévenu de l'échec de la mission en regardant le suivi des fusées et commence à s'organiser. Pendant ce temps, Gene assigne le nouveau vaisseau spatial disponible à la mission de ravitaillement. Manque de chance, le temps n'est pas propice à un lancement et Gene doit le reporter de quelques jours. Gene fait décoller le deuxième vaisseau. Cette fois-ci tout se passe bien, Jim reçoit l'information que le vaisseau est en approche toujours en consultant le suivi des fusées, puis Gene reçoit l'information que le vaisseau est arrivé. L'état du vaisseau change et Jim commence à décharger le matériel et à réapprovisionner la base et les modules. Finalement, Jim fait un nouveau contrôle de l'inventaire et constate qu'ils sont maintenant suffisants.

User stories couvertes dans ce scénario

Les user stories 2, 3, 5, 8, 12, 16, 18, 20, 24 et 25 sont traitées ici mais la 24 n'est pas implémentée.

Diagramme de séquence



Déroulé du scénario

Les requêtes bleues correspondent à celles que nous n'avons pas eu à implémenter car ne rentrant pas dans la liste des user stories obligatoires ou choisies par l'équipe.

1. Requête **GET** sur /moon-base/inventory de MoonBase Service de Jim pour vérifier l'inventaire.
2. Requête **GET** sur /module/inventory de Module Life Service de Moon Base Service afin de vérifier l'inventaire de chacun des modules et de l'additionner au stock général.
3. Requête **GET** sur /needs-control/module de Needs de Needs Control Service de Buzz afin de connaitre les besoins de la base lunaire.
4. Requête **GET** sur /moon-base/needs Moon Base Service de Needs Control Service afin de connaitre les besoins de la base lunaire.
5. Requête **GET** sur /module/needs de Module Life Service de Moon Base Service afin de connaitre les besoins de chacun des modules et de l'additionner aux besoins de la base.
6. Requête **POST** sur /needs-control/sendOrder de Needs Control Service de Buzz pour passer une nouvelle commande à la Terre.
7. Requête **POST** sur /resupply/supply de Resupply Service de Needs Control Service pour créer une nouvelle commande.
8. Requête **GET** sur /resupply/supplyOrders de Resupply Service de Dorothy afin de consulter les dernières commandes passées depuis la Lune.
9. Requête intermédiaire **GET** sur /resupplyMission de Resupply Service afin de vérifier qu'aucune mission de réapprovisionnement n'est prête.
10. Requête **PUT** sur /resupply/{supplyOrderId}/validate de Resupply Service de Marie afin de valider la commande de Buzz.
11. Requête intermédiaire **GET** sur /resupply/supplyOrders de Resupply Service afin de vérifier que la commande a bien été validée.
12. Requête **GET** sur /resupplyMission de Resupply Service de Marie afin de consulter les informations de la prochaine mission de ravitaillement.
13. Requête **GET** sur /onEarthAstronauts de Astronaut Service de Gene afin de connaitre les astronautes présents sur Terre.
14. Requête **GET** sur /onMoonAstronauts de Astronaut Service de Gene afin de connaitre les astronautes présents sur la Lune.
15. Requête **POST** sur /rotation-mission de Rotation Mission Service de Gene afin d'affecter les astronautes sur Terre à une nouvelle mission de renouvellement.
16. Requête **POST** sur /rotation-mission de Rotation Mission Service de Gene afin d'affecter les astronautes sur la Lune à une nouvelle mission de renouvellement venant équilibrer la précédente.
17. Requête **GET** sur /spacecraft de Spacecraft Service de Gene afin de connaitre les fusées disponibles.
18. Requête **PUT** sur /spacecraft/{spacecraftId}/affectSpaceCraftToResupplyMission de Spacecraft Service de Gene pour affecter la fusée 1 à la mission de réapprovisionnement.
19. Requête **PUT** sur /resupply/{resupplyMissionId}/affectSpacecraft de Resupply Service de Spacecraft Service afin d'affecter la fusée 1 à la mission de réapprovisionnement de Dorothy.
20. Requête intermédiaire **GET** sur /resupply/supplyOrders de Resupply Service afin de vérifier que la fusée a bien été affecté à la mission de réapprovisionnement.

21. Requête **PUT** sur /spacecraft/{spacecraftId}/affectSpaceCraftToRotationMission de Spacecraft Service de Gene afin d'affecter la fusée 1 à la première mission de rotation des astronautes.
22. Requête **PUT** sur /rotation-mission/{rotationMissionId}/affectSpacecraft de Rotation Mission Service de Spacecraft Service pour affecter la fusée à la mission de rotation des astronautes.
23. Requête **PUT** sur /spacecraft/{spacecraftId}/affectSpaceCraftToRotationMission de Spacecraft Service de Gene afin d'affecter la fusée 1 à la deuxième mission de rotation des astronautes.
24. Requête **PUT** sur /rotation-mission/{rotationMissionId}/affectSpacecraft de Rotation Mission Service de Spacecraft Service pour affecter la fusée à la mission de rotation des astronautes.
25. Requête intermédiaire **GET** sur /rotation-mission de Rotation Mission Service afin de vérifier que la fusée 1 est bien affectée aux deux missions.
26. Requête **PUT** sur /spacecraft/{spacecraftId}/launch de Spacecraft Service de Gene afin de lancer la fusée dans l'espace.
27. Emission d'un event *spacecraft-launch* par Spacecraft Service à la suite du lancement de la fusée 1. Ecoute de l'event *spacecraft-launch* par Resupply Service qui change le statut de la mission de réapprovisionnement. Ecoute de l'event *spacecraft-launch* Rotation Mission Service.
28. Emission d'un event *rotation-launched* par Rotation Mission Service à la suite de l'écoute de l'event *spacecraft-launch* si la fusée était affectée à une mission de renouvellement. Ecoute de l'event *rotation-launched* par Astronaut Service qui change le statut des astronautes de la mission de renouvellement à ON_TRANSIT.
29. Requête intermédiaire **GET** sur /resupply/supplyOrders de Resupply Service afin de vérifier que le statut de la mission de réapprovisionnement est « TRAVELING ».
30. Requête intermédiaire **GET** sur /astronaut de Astronaut Service afin de vérifier que les astronautes sont bien en train de voyager.
31. Requête intermédiaire **PUT** sur /spacecraft/{spacecraftId} de Spacecraft Service afin de modifier l'état de la fusée en endommagée. Il s'agit de simuler de l'explosion de la fusée qui serait détectée par un capteur et retransmis au service.
32. Emission d'un event *spacecraft-damaged* par Spacecraft Service à la suite de la modification du statut de la fusée en « DAMAGED ». Ecoute de l'event *spacecraft-damaged* par Resupply Service qui va remettre la mission de réapprovisionnement en « PREPARING ». Ecoute de event *spacecraft-damaged* par Rotation Mission Service.
33. Emission d'un event *rotation-failed* par Rotation Mission Service à la suite de l'écoute d'un event *spacecraft-damaged* si la fusée était affectée à une mission de renouvellement. Ecoute de l'event *rotation-failed* par Astronaut Service qui change déclare les astronautes comme décédés.
34. Emission d'un event *astronaut-dead* par Astronaut Service pour chaque astronaute décédé à la suite de l'écoute d'un event *rotation-failed*. Ecoute de l'event *astronaut-dead* News Formalization Service .
35. Emission d'un event *news* par News Formalization Service à la suite de l'écoute d'un event *astronaut-dead*. Ecoute de l'event news par News Service qui sauvegardera la triste nouvelle
36. Requête intermédiaire **GET** sur /resupply/supplyOrders de Resupply Service afin de vérifier que le statut de la mission de réapprovisionnement est « PREPARING ».
37. Requête intermédiaire **GET** sur /astronaut de Astronaut Service afin de vérifier que les astronautes sont décédés.

38. Requête **GET** sur /spacecraft/crashed de Spacecraft Monitoring Service de Jim afin de surveiller les fusées et de constater que la fusée contenant sa mission de réapprovisionnement est endommagée.
39. Requête **PUT** sur /spacecraft/{spacecraftId}/affectSpaceCraftToResupplyMission de Spacecraft Service de Gene afin d'affecter la fusée 2 à la mission de réapprovisionnement de Dorothy.
40. Requête **PUT** sur /resupply/{resupplyMissionId}/affectSpacecraft de Resupply Service de Spacecraft Service afin d'affecter la fusée 2 à la mission de réapprovisionnement de Dorothy.
41. Requête intermédiaire **GET** sur /resupply/supplyOrders de Resupply Service afin de vérifier que la fusée a bien été affecté à la mission de réapprovisionnement.
42. Requête **GET** sur /weather/prevision/{dateStart}/{dateEnd} de Weather Monitoring Service de Gene afin de s'informer du temps prévu pour le jour du lancement de la fusée.
43. Requête **PUT** sur /spacecraft/{spacecraftId} de Spacecraft Service de Gene afin de modifier le jour de lancement de la fusée.
44. Requête **PUT** sur /spacecraft/{spacecraftId}/launch de Spacecraft Service de Gene afin de lancer la fusée dans l'espace.
45. Emission d'un event *spacecraft-launch* par Spacecraft Service à la suite du lancement de la fusée 1. Ecoute de l'event *spacecraft-launch* par Resupply Service qui change le statut de la mission de réapprovisionnement. Ecoute de l'event *spacecraft-launch* Rotation Mission Service.
46. Requête intermédiaire **GET** sur /resupply/supplyOrders de Resupply Service afin de vérifier que le statut de la mission de réapprovisionnement est « TRAVELING ».
47. Requête intermédiaire **PUT** sur /spacecraft/{spacecraftId} de Spacecraft Service afin de modifier l'état de la fusée en « ARRIVING ». Il s'agit de simuler de l'arrivée imminente de la fusée qui serait détectée par un capteur et retransmis au service.
48. Emission d'un event *spacecraft-arriving* par Spacecraft Service à la suite de la modification de l'état de la fusée. Ecoute de l'event *spacecraft-arriving* par Spacecraft Monitoring Service.
49. Requête **GET** sur /spacecraft/arriving de Spacecraft Monitoring Service de Jim afin de surveiller les fusées dont l'arrivée est imminente.
50. Requête intermédiaire **PUT** sur /spacecraft/{spacecraftId} de Spacecraft Service afin de modifier l'état de la fusée en « LANDED ». Il s'agit de simuler de l'alunissage de la fusée qui serait détecté par un capteur et retransmis au service.
51. Emission d'un event *spacecraft-landed* par Spacecraft Service à la suite de la modification de l'état de la fusée. Ecoute de l'event *spacecraft-landed* par Spacecraft Monitoring Service.
52. Requête **GET** sur /spacecraft/landed de Spacecraft Monitoring Service afin de vérifier que la fusée a aluni.
53. Requête **GET** sur /spacecraft/arriving de Spacecraft Monitoring Service afin de vérifier que la fusée n'est plus en phase d'approche.
54. Requête intermédiaire **GET** sur /spacecraft de Spacecraft Service pour de vérifier l'état de la fusée.
55. Requête **POST** sur /moon-base/{moonBaseId}/supply de Moon Base Service de Jim afin d'indiquer le réapprovisionnement de la base lunaire.
56. Requête **GET** sur /moon-base/inventory de Moon Base Service de Jim pour vérifier que l'inventaire a bien été réapprovisionné.

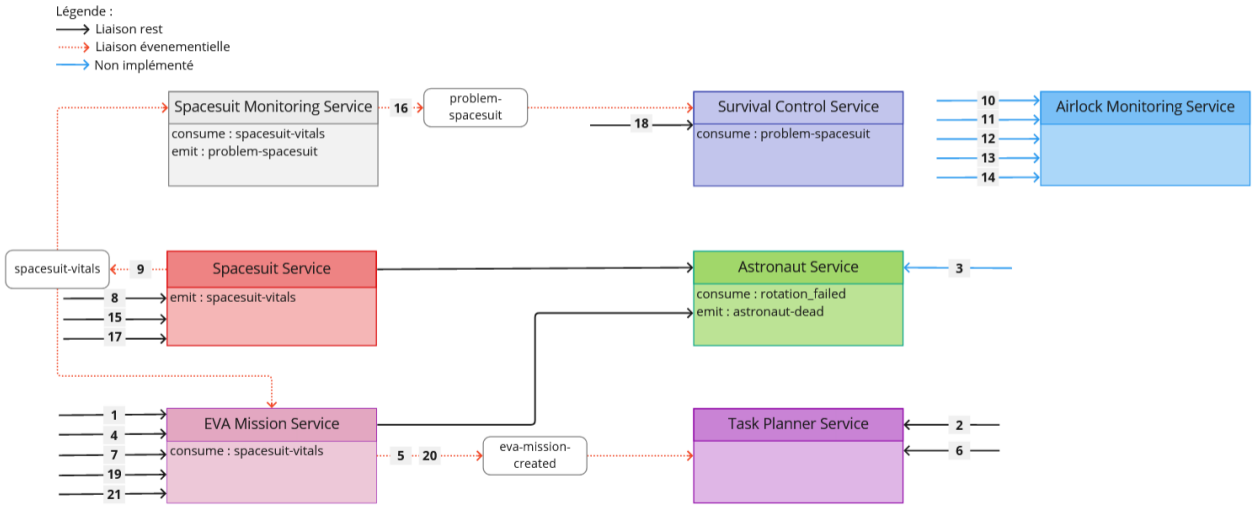
SCENARIO 3 – PROGRAMMATION D'UNE EVA MISSION ET SURVEILLANCE DU DEROULEMENT DE CELLE-CI

Buzz veut vérifier les missions EVA des prochains jours mais il constate qu'aucune mission n'est prévue dans la zone d'exploration 3. Il consulte tout d'abord toutes les tâches attribuées aux astronautes de la base puis le temps passé en extérieur de chaque personne afin de choisir le responsable de cette mission. Son choix se porte sur Jim et il créé une mission EVA en y affectant Jim, une tâche se créée automatiquement à la date voulue. Sur Terre, Marie s'informe des prochaines missions EVA pour programmer une diffusion en direct, elle choisira la mission de Jim. Le jour J, l'astronaute enfile sa combinaison et sort en passant par le sas de décompression. Quelques minutes plus tard, il ne se sent pas bien et vérifiant ses constantes et sa combinaison, il découvre que celles-ci sont mauvaises. Une alerte a déjà été envoyée et Buzz programme un EVA Mission de secours pour sauver Jim. Quelques jours plus tard, Deke consulte toutes les métriques de la mission de Jim pour comprendre ce qui a mal tourné et rassurer le public présent lors de la mission.

User stories couvertes dans ce scénario

Les user stories 6, 7, 10, 11, 19, 21, 22 et 23 sont traitées ici mais la 21 et la 23 ne sont pas implémentées.

Diagramme de séquence



Déroulé du scénario

Les requêtes bleues correspondent à celles que nous n'avons pas eu à implémenter car ne rentrant pas dans la liste des user stories obligatoires ou choisies par l'équipe.

1. Requête **GET** sur /eva-mission de EVA Mission Service de Buzz afin de consulter les EVA missions des prochains jours.
2. Requête **GET** sur /task-planner de Task Planner de Buzz afin de consulter de planning des tâches des astronautes présents sur la base lunaire.
3. Requête **GET** sur /onMoonAstronauts de Astronaut Service de Buzz afin de s'informer du temps d'exposition des astronautes
4. Requête **POST** sur /eva-mission de EVA Mission Service de Buzz pour créer une nouvelle mission d'exploration.

5. Emission d'un évènement eva-mission-created par EVA Mission Service. Ecoute de l'évènement par Task Planner qui créera une nouvelle tâche correspondant à cette mission d'exploration.
6. Requête **GET** sur /task-planner de Task Planner de Buzz afin de vérifier si la nouvelle tâche d'exploration affectée à Jim a bien été créée.
7. Requête **GET** sur /eva-mission de EVA Mission Service de Marie afin de consulter les prochaines EVA Missions et planifier ses diffusions en direct.
8. Requête **PUT** sur / spacesuit/{spacesuitId}/affect-astronaut de Spacesuit Service de Jim afin de notifier qu'il vient de s'équiper de sa combinaison.
9. Emission d'un évènement spacesuit-vitals par Spacesuit Service en boucle toutes les secondes contenant les informations sur l'état de santé de l'astronaute et son équipement. Ecoute de l'évènement spacesuit-vitals par EVA Mission Service qui ajoute ces données aux métriques de la mission que Jim effectue. Ecoute de l'évènement par Spacesuit Monitoring Service qui déclenche une alerte lorsque les données reçues sont inquiétantes.
10. Requête **PUT** sur /airlock/{airlockId} de Airlock Monitoring Service. Il s'agit de simuler l'action de Jim appuyant sur le bouton d'ouverture de la porte interne.
11. Requête **PUT** sur /airlock/{airlockId} de Airlock Monitoring Service. Il s'agit de simuler l'action de Jim appuyant sur le bouton de fermeture de la porte interne.
12. Requête **PUT** sur /airlock/{airlockId}/depressurize de Airlock Monitoring Service. Il s'agit de simuler l'action de Jim appuyant sur le bouton de dépressurisation du sas car il s'agit d'une sortie vers l'extérieur.
13. Requête **PUT** sur /airlock/{airlockId} de Airlock Monitoring Service. Il s'agit de simuler l'action de Jim appuyant sur le bouton d'ouverture de la porte externe.
14. Requête **PUT** sur /airlock/{airlockId} de Airlock Monitoring Service. Il s'agit de simuler l'action de Jim appuyant sur le bouton de fermeture de la porte externe.
15. Requête **POST** sur /spacesuit/{spacesuitId}/update-vitals de Spacesuit Service. Il s'agit de simuler la mise à jour des valeurs des conditions de l'astronaute et de son équipement par des capteurs
16. Ecoute de l'évènement par Spacesuit Monitoring Service qui émet un évènement problem-spacesuit car les nouvelles données sur la pression et le taux d'oxygène ne sont pas bons. Ecoute de l'évènement problem-spacesuit par Survival Control Service qui va sauvegardé le numéro de la combinaison en détresse.
17. Requête **GET** sur /spacesuit/{spacesuitId} de Spacesuit Service de Jim afin de connaitre ses constantes et comprendre son état précaire.
18. Requête **GET** sur /survival-control/spacesuit-with-problem de Survival Control de Buzz afin de se tenir informé si des combinaisons sont défectueuses.
19. Requête **POST** sur /eva-mission de EVA Mission Service de Buzz pour créer une nouvelle mission de secours afin d'aider Jim.
20. Emission d'un évènement eva-mission-created par EVA Mission Service. Ecoute de l'évènement par Task Planner qui créera une nouvelle tâche correspondant à cette mission de secours.
21. Requête **GET** sur /eva-mission/{evaMissionId}/metrics de EVA Mission Service par Deke afin de récolter les données collectées sur la combinaison durant la mission d'exploration.

Prise de recul

TESTS EFFECTUES

Pour valider notre implémentation des scénarios, notre démarche a été la suivante, dans un premier temps :

1. Des tests Swagger ont été effectuées pour vérifier la communication entre services
2. Un script d'intégration suivant les scénarios définis a été écrit dans un fichier integration.sh à l'aide d'opérations curl

Nos méthodes de test ont ensuite évolué vers la solution ci-dessous :

3. Un script Python permettant d'effectuer les différents appels au micro-services tels que le feraient nos persona a été réalisé selon les scénarios définis précédemment
4. Le script a été dockerisé afin de pouvoir le lancer dans un environnement contenant toutes les dépendances nécessaires
5. Le container Docker est lancé sur le même réseau que nos micros-services sans être détaché afin de pouvoir suivre les logs en direct

ÉVOLUTION DE L'ARCHITECTURE AU COURS DU PROJET

L'évolution de notre conception de solution et de son diagramme s'est faite à travers plusieurs points importants.

Dans un premier temps notre projet ne possédait qu'un ensemble de modules que l'on gérait dans son intégralité à travers le service ModuleLife. Avec le temps et la complexification du projet à travers les nouvelles user stories, il n'était plus cohérent de ne garder que ce service car nous avions à traiter de nouvelles données et de nouvelles demandes qui ne lui correspondaient que peu. Un second service a alors été ajouté à l'architecture : MoonBase. La mise en place de ce service a permis de faire un redécoupage du service de gestion des modules en ModuleLife et MoonBase afin que ModuleLife ne gère qu'un module, ses ressources personnelles été ses capteurs alors que MoonBase aura une vue sur l'ensemble des modules et le stock de ressources général. MoonBase est donc un agrégat des entités ModuleLife, on pourrait comparer une MoonBase a la mairie d'une ville quand les ModuleLife sont les habitations (avec tout ce qui les compose – stocks de nourriture, habitants ...). De cette manière-là, nous sommes plus proche du métier et plus proche du fonctionnement dans la vraie vie.

La seconde grande évolution de notre conception a pu être vue avec l'ajout d'un SpacecraftService afin de réellement différencier la fusée des commandes de ressources qu'elle transporte. L'action d'envoi launch et le statut associé « LAUNCHED » ont alors été attribués à la fusée plutôt qu'à la commande. Cela s'est avéré être un choix judicieux avec l'ajout de nouveaux besoins comme la gestion du roulement u personnel de la base lunaire que nous avons pu gérer de la même manière que des missions de réapprovisionnement ou encore les spécificités liées au lancement de fusées.

Le reste des évolutions de notre architecture a été fait en ajoutant différents nouveaux services pour les user stories avec pour but d'avoir un service par ensemble de feature ayant un métier similaire. Ainsi nous pouvons facilement scaler les fonctionnalités qui sont soumises à une charge élevée et dans le cas d'un service qui chute, une grande partie de l'application est toujours fonctionnelle.

De plus, chaque service est stateless ce qui encourage la scalabilité et la résilience de notre solution.

Certaines parties de notre application pourraient être améliorées avec par exemple une gestion des évènements Kafka sur des éléments tels que les changements de statuts qui dans certains cas sont encore fait à l'aide de requêtes PUT comme pour les changements des valeurs des capteurs des modules, fusées et de requêtes POST pour les capteurs des combinaisons pour les métriques de EVA missions par exemple.

On pourrait également mettre en place des mocks de ces capteurs.

RISQUES DE NOTRE ARCHITECTURE

Un des points rendant notre approche du projet Selene adaptable à une application réelle du problème de gestion de base lunaire est la manière dont nous avons pensé l'application. En effet, lors des discussions de l'équipe pour l'implémentation de notre solution, les user stories ont été vues d'un point de vue pratique et applicable dans la vraie vie en se posant la question « Si nous étions à la place de tel persona, comment ferions-nous ? Quelles sont les contraintes physiques et budgétaires à respecter dans ce cas ? Quels sont les critères essentiels à traiter ».

Tout ne pouvant néanmoins être traité dans les détails par contrainte de temps, voici les limitations que pourrait rencontrer notre solution. Dans notre cas, la principale limitation dans notre solution est la gestion des modifications d'EVA missions peut actuellement mener à des incohérences de données, en effet par manque de temps et complexité dans le code le planning des tâches verra la création d'une EVA mission mais la modification de celle-ci depuis l'EVAMissionService n'est pas visible par le TaskPlanningService. De la même manière, une modification d'une tâche de type « EXPLORATION » ou « HELP » dans le TaskPlanningService n'influera pas sur les données de la base gérée par EVAMissionService.

POINTS DE L'ARCHITECTURE DONT NOUS SOMMES LES PLUS FIERS

Les meilleurs points de notre architecture d'après nous sont les suivants.

La séparation du métier dans différents services qui ont tous un sens et sont applicables à des situations réelles rend notre solution facilement compréhensible et extensible.

Notre gestion des évènements nous permet de formaliser mais aussi de traiter d' autres évènements.

Enfin, le peu de base de données utilisées dans notre solution rend notre système plus fiable en cas de chute de l'un des services, chaque service gérant uniquement ses données qui lui sont relatives sans avoir connaissance de quoi que ce soit d'autre.

D'un point de vue plus macro, nous sommes contents d'avoir pu faire évoluer l'architecture au fil du temps, sans avoir à reconstruire toute l'architecture. Nous avons su adapter notre architecture, modifier certains micro-services, mais nous n'avons jamais eu à repartir de zéro. C'était un objectif compliqué à tenir, surtout quand nous avons eu à implémenter l'approche asynchrone avec Kafka, mais nous avons su nous adapter, adapter notre architecture. Bien sûr, nous avons eu à modifier certains micro-services et avec du recul, nous nous rendons compte que nous aurions pu ajouter plus d'événementiel, mais jamais à changer radicalement leur métier et ce pour quoi nous les avions conçus à la base.