Simeis

Simeis est un jeu par API (inspiré de SpaceTraders), dont le but est de faire fructifier votre empire économique dans toute la galaxie.

Dans ce manuel, vous trouverez les mécaniques de base du jeu, c'est à vous de porter ces mécaniques de bases vers l'excellence intergalactique !

Ce manuel contient les URL de l'API brute (en query GET standard), mais aussi la fonction associée dans le fichier `api.rs` si vous travaillez en Rust.

Dans les descriptions des appels à l'API, il est admis que vous avez utilisé l'ID attendu dans les arguments de la query GET, ainsi ce manuel ne couvrira pas les cas d'erreurs où l'ID attendu n'est pas le bon

1. Le Joueur

1.1. La structure de donnée `Player`

- `id` : L'ID du joueur
- `key` : La clé **privée** d'authentification du joueur sur le jeu
- `lost` : Si le joueur a déjà perdu
- `name` : Le nom de ce joueur
- `money` : Argent du joueur
- `costs` : Frais que le joueur perdra par secondes (salaires de l'équipage, voir Section 4)
- `stations` : Coordonnées et ID de chacune des stations qu'un joueur possède
- `ships` : ID et données des vaisseaux qu'un joueur possède (voir Section 2.1)

1.2. Créer un joueur

La création d'un joueur requiert un *nom unique*, qui vous donnera accès à un **ID** de joueur, ainsi qu'à une **clé** d'authentification unique.

Pour intéragir avec ce joueur, tous les appels à l'API devront comporter un paramètre `key=<votre clé>` dans l'URL de votre query GET.

- Endpoint: `/player/new/{name}`
- Client: `ApiClient::new_player`

Retournera une erreur si le nom du joueur existe déjà

À sa création, le joueur n'aura aucun vaisseau, et sera doté d'un montant fixe d'argent.

Il sera propriétaire d'une station (Section 3)

1.3. Obtenir des informations d'un joueur

Les informations obtenues seront complètes s'il s'agit de votre joueur, ou partielles s'il s'agit d'un autre joueur.

- Endpoint: `/player/{id}`
- Client: `ApiClient::get_player`

Retournera une erreur si le joueur avec l'ID correspondant n'existe pas

Les données retournées comprendront:

- `id` : L'ID du joueur
- `name` : Le nom du joueur
- `stations` : Liste des coordonnées de toutes les stations que possède le joueur, mappées par leur IDs,

Si la clé passée dans la query correspond à la clé du joueur, les données retournées comprendront aussi:

- `ships` : Une liste des vaisseaux du joueur et leurs métadonnées
- `money` : Combien d'argent a ce joueur
- `costs` : Combien d'argent par second ce joueur perds en frais

2. Les vaisseaux

2.1. La structure de données `Ship`

- `id` : ID du vaisseau
- `position` : Coordonnées du vaisseau dans l'espace

Caractéristiques du vaisseau

- `reactor_power` : Puissance du moteur, détermine la vitesse du vaisseau
- `fuel_tank_capacity` : Contenance du réservoir de carburant
- `hull_decay_capacity` : Quantité d'usure que la coque peut subir avant destruction
- `modules` : Tous les modules du vaisseau et leur ID

Cargo

- `cargo.usage` : Volume utilisé par les resources à bord
- `cargo.capacity` : Volume total que peut contenir le cargo
- `cargo.resources` : Pour chaque resource contenue, quelle quantité est en stock

État

- `state` : Commande actuelle du vaisseau (en vol, extraction, inactif...)
- `fuel_tank` : Niveau de carburant restant, à `0` le vaisseau s'immobilisera
- `hull_decay` : Usure de la coque, à `hull_decay_capacity`, le vaisseau se détruira
- `stats` : Performances du vaisseau, calculées à partir des caractéristiques du vaisseau

Équipage

- `crew` : Équipage de ce vaisseau (voir Section 4)
- `pilot` : ID du membre d'équipage assigné en tant que pilote de ce vaisseau

2.2. Obtenir de informations sur un vaisseau

Récupère toutes les données d'un vaisseau (voir Section 2.1)

- Endpoint: `/ship/{ship_id}`
- Client: `ApiClient::get_ship`

2.3. Lister les vaisseaux à l'achat

Retourne la liste de tous les vaisseaux pouvant être achetés dans une station, ainsi que leurs caractéristiques (voir Section 2.1)

- Endpoint: `/station/{station_id}/shipyard/list`
- Client: `ApiClient::list_shipyard_ships`

Les données reçues, en plus des caractéristiques du vaisseau en question, contient le prix à payer pour ce vaisseau (clé `price`).

2.4. Acheter un vaisseau

Achète le vaisseau d'ID `ship_id` sur la station `station_id`

- Endpoint: `/station/{station_id}/shipyard/buy/{ship_id}`
- Client: `ApiClient::buy_ship`

Retournera une erreur si le prix est au dessus de vos moyens financiers

Le nouveau vaisseau aura pour position celle de la station, avec un plein de carburant et une coque toute neuve.

Attention: Sans équipage, le vaisseau ne peut pas fonctionner (voir Section 4)

2.5. Lister les modules de vaisseau à l'achat

Liste tous les modules de vaisseaux disponibles à l'achat dans cette station, ainsi que leur prix

- Endpoint: `/station/{station_id}/shop/modules`
- Client: `ApiClient::list_shop_ship_module`

2.6. Acheter un module de vaisseau

Achète un module de type `module` et l'installe sur le vaisseau `ship_id`

Types de module possible:

- `Miner`: Permet d'extraire des resources depuis les planètes solides
- `GasSucker`: Permet d'extraire du gaz depuis les planète gaseuses

- Endpoint: `/station/{station_id}/shop/modules/{ship_id}/buy/{module}`
- Client: `ApiClient::buy_ship_module`

Retournera une erreur si le type de module n'est pas reconnu, si le vaisseau n'est pas à la station, le prix est trop élevé

2.7. Lister les améliorations de modules

Retourne la liste de tous les modules installés sur le vaisseau et leur ID, type et le prix à payer pour augmenter leur rang.

Chaque rang de module booste son efficacité, et permet d'extraire des resources nécessitant un rang élevé

- Endpoint: `/station/{station_id}/shop/modules/{ship_id}/upgrade`
- Client: `ApiClient::list_ship_module_upgrade`

Retournera une erreur si le vaisseau n'est pas à la station

2.8. Acheter une amélioration de module

Achète une amélioration de l'un des modules installés sur le vaisseau

- Endpoint: `/station/{station_id}/shop/modules/{ship_id}/upgrade/{module}`
- Client: `ApiClient::buy_ship_module_upgrade`

Retournera une erreur si le prix est élevé pour vos moyens, ou le vaisseau n'est pas à la station

2.9. Extraire des resources

Attention: Votre vaisseau doit être positionné à la même coordonée qu'une planète pour lancer une extraction de resources.

En fonction des resources disponibles sur la planète (solide ou gazeuse), utilisera les modules du vaisseau installé pour extraire des resources.

Le montant de resources obtenues dépendent de:
- La richesse de cette planète en cette resource
- La difficulté d'extraction de cette resource
- Le rang de votre opérateur
- Le rang de votre module d'extraction

Le vaisseau extraiera des resources jusqu'à ce que son cargo soit totalement plein.
Une fois le cargo plein, le vaisseau retrouvera un `state` inactif (`Idle`)

L'appel à cet endpoint retournera, pour chacune des resources extraites, la quantité extraite par seconde.

Attention: Un module n'ayant pas d'opérateur assigné ne produira rien.

- Endpoint: `/ship/{ship_id}/extraction/start`
- Client: `ApiClient::start_extraction`

Retournera une erreur si le vaisseau est déjà occupé ou n'est pas positionné sur une planète

Les planètes **solides**, nécessitant un module `Miner` possèdent les resources:
- `Stone`, de la pierre, peu chère mais facile à extraire
- `Iron`, du fer, plus cher mais plus rare et difficile à extraire

Les planètes **gazeuses**, nécessitant un module `GasSucker` possèdent les resources:
- `Helium`, peu cher et simple à extraire
- `Ozone`, plus cher mais plus rare et difficile à extraire

2.10. Arrêter l'extraction

Arrête l'extraction en cours, restaure l'état du vaisseau à `Idle` (inactif)

- Endpoint: `/ship/{ship_id}/extraction/stop`
- Client: `ApiClient::stop_extraction`

Retournera une erreur si le vaisseau n'était pas en train d'extraire

3. Stations

3.1. La structure de données `Station`

- `id` : L'ID de la station
- `position` : Coordonnées de la station dans l'espace
- `shipyard` : Liste des vaisseaux à l'achat sur cette station

Cargo

- `cargo.usage` : Volume utilisé par les resources à bord
- `cargo.capacity` : Volume total que peut contenir le cargo
- `cargo.resources` : Pour chaque resource contenue, quelle quantité est en stock

Équipage

- `idle_crew` : Équipage inactif de cette station (voir Section 4)
- `crew` : Équipage de cette station (voir Section 4)
- `trader` : ID du membre d'équipage assigné en tant que Trader

3.2. Obtenir le status de la station

Retourne les données de la station (voir Section 3.1 pour le détail)

- `id`
- `position`
- `crew`
- `idle_crew`
- `cargo`
- `trader`

- Endpoint: `/station/{station_id}`
- Client: `ApiClient::get_station_status`

3.3. Faire le plein de carburant

Transfert du carburant (`Resource::Fuel`) depuis le **cargo de la station** vers le réservoir de carburant du vaisseau.

Attention: Cette opération nécessite d'avoir du carburant de stocké dans le cargo de la station. Voir Section 6.3 pour savoir comment acheter du carburant

- Endpoint: `/station/{station_id}/refuel/{ship_id}`
- Client: `ApiClient::refuel_ship`

Retournera une erreur si il n'y a pas de carburant dans le cargo de la station

Ce endpoint retourne quelle quantité de carburant a été ajoutée au réservoir du vaisseau.

3.4. Réparer la coque du vaisseau

Utilise des plaques de réparation (`Resource::HullPlate`) depuis le **cargo de la station** pour diminuer l'usure de la coque du vaisseau (Voir `hull_decay` à Section 2)

- Endpoint: `/station/{station_id}/repair/{ship_id}`
- Client: `ApiClient::repair_ship`

Retournera une erreur si il n'y a pas de plaques de réparations dans le cargo de la station

Ce endpoint retourne de combien d'unités l'usure du vaisseau a été restaurée

3.5. Décharger un vaisseau

Retire du cargo du vaisseau une quantité `amnt` de la resource `resource`, et la place dans le cargo de la station où le vaisseau est stationné.

Si le vaisseau n'a pas autant de quantité, déchargera le maximum

Si la station n'a pas assez de place pour accueillir les resources, en chargera un maximum et replacera le reste dans le cargo du vaisseau

L'appel à cet endpoint retournera la quantité de resource qui a été déchargée

- Endpoint: `/ship/{ship_id}/unload/{resource}/{amnt}`
- Client: `ApiClient::unload_cargo`

Retournera une erreur si le vaisseau n'est pas dans une station

3.6. Lister le prix des améliorations de vaisseau

Retourne la liste de toutes les améliorations possibles sur les vaisseaux sur cette station, ainsi que leur prix.

- Endpoint: `/station/{station_id}/shipyard/upgrade`
- Client: `ApiClient::list_ship_upgrades`

Chacune des amélioration aura son prix (clé `price`) et une description (clé `description`)

3.7. Améliorer le vaisseau

Achète une amélioration de type `upgrade_type` sur le vaisseau `ship_id` s'il est stationné sur la station `station_id`

- Endpoint: `/station/{station_id}/shipyard/upgrade/{ship_id}/{upgrade_type}`
- Client: `ApiClient::buy_ship_upgrade`

Retournera une erreur si le type d'upgrade n'est pas reconnu, le vaisseau n'est pas sur la station, ou le joueur n'a pas assez d'argent pour acheter cet upgrade

3.8. Lister les améliorations de la station

Retourne le prix à payer pour chaque amélioration de la station:

- `cargo-expansion`: Prix par extension de cargo (voir Section 3.9)
- `trader-upgrade`: Prix pour augmenter le rang du trader (voir Section 4.8)

Attention La fonction de `ApiClient` nécessite de passer un paramètre `key` correspondant au type d'amélioration à avoir

- Endpoint: `/station/{station_id}/upgrades`
- Client: `ApiClient::get_station_upgrade_price`

3.9. Acheter une extension de cargo à la station

Améliore le cargo de la station en y ajoutant une quantité `amount` de capacité de cargo.

Le prix augmentera à chaque extension achetée

- Endpoint: `/station/{station_id}/shop/cargo/buy/{amount}`
- Client: `ApiClient::buy_station_cargo`

Retournera une erreur si vous n'avez pas assez d'argent

3.10. Scanner les planètes aux alentours

Scanne tous les objets dans ce secteur de la galaxie, et retourne leur liste:

- Planètes
- Stations

Peut être utilisé pour obtenir la position d'une planète à aller exploiter, ses caractéristiques (si elle est *solide* ou *gazeuse*, voir Section 2.9)

- Endpoint: `/station/{station_id}/scan`
- Client: `ApiClient::station_scan`

4. Équipage

4.1. La structure de donnée `CrewMember`

- `rank` : Rang de l'équipier, ses performances dépenderont de ce niveau (voir Section 4.7)
- `member_type` : Quel type d'équipage ce membre fait partie
  - `CrewMemberType::Pilot` : Pilote de vaisseau (voir Section 4.4 et Section 2.1)
  - `CrewMemberType::Operator` : Opérateur de module (voir Section 4.5 et Section 2.1)
  - `CrewMemberType::Trader` : Trader sur la station (voir Section 4.3 et Section 3.1)
  - `CrewMemberType::Soldier` : Soldat (développement en cours)

4.2. Engager un membre d'équipage

Engager un nouveau member d'équipage, qui rejoindra la station en status **inactif** (voir Section 3.2)

Les types de membres incluent:

- `pilot` : Permet de déplacer un vaisseau à travers la galaxie
- `operator` : Permet d'utiliser un module d'extraction de resource sur un vaisseau
- `trader` : Permet d'acheter ou vendre des resources dans une station (voir Section 6.4)

Chaque membre d'équipage pourra ensuite être améliorer (voir Section 4.7)

- Endpoint: `/station/{station_id}/crew/hire/{crew_type}`
- Client: `ApiClient::hire_crew`

Retournera une erreur si le type de member n'est pas reconnu

L'ID du membre de l'équipage sera retourné.

4.3. Assigner un trader

Assigne un trader à une station, permet alors d'utiliser les fonctionalités d'achat et de vente de resources (voir Section 6.4 et Section 6.3)

Les frais appliqués à chaque transaction dépenderont du rang du trader

- Endpoint: `/station/{station_id}/crew/assign/{crew_id}/trading`
- Client: `ApiClient::assign_trader`

Retournera une erreur si un trader est déjà assigné, le membre d'équipage n'est pas inactif ou n'est pas un trader

4.4. Assigner un pilote

Assigne un membre d'équipage (de type pilote) inactif en tant que pilote (voir Section 2.1).

La vitesse du vaisseau, ainsi que la consommation de carburant dépenderont du rang du pilote

- Endpoint: `/station/{station_id}/crew/assign/{crew_id}/{ship_id}/pilot`
- Client: `ApiClient::assign_pilot`

Retournera une erreur si le vaisseau n'est pas sur la station, un pilote est déjà assigné, le membre d'équipage n'est pas inactif ou n'est pas un pilote

4.5. Assigner un opérateur

Assigne un opérateur sur l'un des modules d'un vaisseau.

Ce module pourra alors fonctionner lorsqu'il sera activé, et ses performances dépenderont du rang de l'opérateur

- Endpoint: `/station/{station_id}/crew/assign/{crew_id}/{ship_id}/{mod_id}`
- Client: `ApiClient::assign_operator`

Retournera une erreur si le vaisseau n'est pas sur la station, un opérateur est déjà assigné à ce module, le membre d'équipage n'est pas inactif ou n'est pas un opérateur

4.6. Lister les améliorations de l'équipage

L'appel à cet endpoint retourne, pour chaque membre d'équipage du vaisseau, le rang suivant et le prix pour l'atteindre

- Endpoint: `/station/{station_id}/crew/upgrade/ship/{ship_id}`
- Client: `ApiClient::get_crew_upgrades`

Retournera une erreur si le vaisseau n'est pas à une station

4.7. Augmenter le rang d'un membre d'équipage

Améliore le rang du membre `crew_id` de l'équipage.

Ses performances, mais aussi son salaire, seront beaucoup augmentés.

Selon le type de membre, cela aura différents effets:

- `Pilot`: Réduis la consommation de carburant, augmente la vitesse
- `Opérator`: Augmente la quantité de resources extraites, débloque certaines resources
- `Trader`: Réduis les frais de trading (voir Section 4.8)

- Endpoint: `/station/{station_id}/crew/upgrade/ship/{ship_id}/{crew_id}`
- Client: `ApiClient::buy_crew_upgrade`

Retournera une erreur si le vaisseau n'est pas dans une station ou vous n'avez pas assez d'argent

4.8. Augmenter le rang du trader d'une station

Améliore le rang du `trader` assigné à une station.

Voir Section 4.7 pour comprendre les effets

- Endpoint: `/station/{station_id}/crew/upgrade/trader`
- Client: `ApiClient::upgrade_trader`

Retournera une erreur si aucun trader n'est assigné à cette station ou vous n'avez pas assez d'argent

5. Navigation

5.1. Calculer les coût d'un voyage

Calcule le coût d'un voyage vers les coordonnées `(x, y, z)` depuis la position du vaisseau

Les données retournées du calcul comprendront:

- `direction`: Vecteur de direction du vaisseau dans l'espace
- `distance`: Distance totale du voyage
- `duration`: Temps (en secondes) du voyage
- `fuel_consumption`: Consommation de carburant qu'un tel voyage nécessite
- `hull_usage`: Usure de la coque d'un tel voyage

- Endpoint: `/ship/{ship_id}/travelcost/{x}/{y}/{z}`
- Client: `ApiClient::travel_cost`

Retournera une erreur si aucun pilote n'est assigné au vaisseau ou la distance est nulle

Attention: Toujours penser au retour lorsque vous prévoyez un voyage

5.2. Voyager

Envoie le vaisseau voyager vers `(x, y, z)`.

Son `state` devient alors `InFlight`, et vous ne pouvez plus le commander jusqu'à son arrivée.

L'appel à cet endpoint retourne les coûts du voyage (voir Section 5.1)

- Endpoint: `/ship/{ship_id}/navigate/{x}/{y}/{z}`
- Client: `ApiClient::navigate`

Retournera une erreur si aucun pilote n'est assigné au vaisseau, la distance est nulle ou les coûts trop élevés (pas assez de carburant...)

6. Le marché

Chaque resource a un prix de base, qui évolue au fil du temps.

L'évolution s'effectue de manière aléatoire, mais est conçue pour être légèrement biaisée pour finalement retourner vers le prix de base.

En revanche le joueur *vends*, le prix baisse, lorsque le joueur *achète*, le prix augmente

Ces variations de prix peuvent être exploitées pour engendrer des bénéfices, mais **chaque opérations sur le marché engendrera des frais**.

Le montant de ces frais dépenderont du rang de votre trader assigné.

Chaque transaction retourne une donnée `MarketTx` comprenant:

- `added_cargo` : Quelle quantité de quelle resource a été ajoutée au cargo de la station
- `removed_cargo` : Quelle quantité de quelle resource a été retirée du cargo de la station
- `added_money` : Combien d'argent a été ajouté au joueur
- `removed_money` : Combien d'argent a été retiré au joueur
- `fees` : Combien d'argent a été soustrait lors de la transaction

6.1. Lister les prix des resources

Retourne, pour chacune des resources, le prix actuel de cette resource sur le marché

- Endpoint: `/market/prices`
- Client: `ApiClient::resource_prices`

6.2. Obtenir le pourcentage de frais

En fonction du rang du trader assigné à cette station, retourne le montant de frais (en pourcentage) qui sera appliqué à chacune des transactions.

- Endpoint: `/market/{station_id}/fee_rate`
- Client: `ApiClient::get_fee_rate`

Retournera une erreur si aucun trader n'est assigné à cette station

6.3. Acheter des resources

Achète une quantité `amnt` de resources sur le marché, et les place dans le cargo de la station où la transaction est effectuée.

- Endpoint: `/market/{station_id}/buy/{resource}/{amnt}`
- Client: `ApiClient::buy_resource`

Retournera une erreur si aucun trader n'est assigné à la station ou la quantité demandée est nulle

6.4. Vendre des resources

Décharge une quantité `amnt` de resources depuis le cargo de la station où la transaction est effectuée, et les vends sur le marché.

Vends une quantité `amnt` de resources sur le marché, et les place dans le cargo de la station où la transaction est effectuée.

- Endpoint: `/market/{station_id}/sell/{resource}/{amnt}`
- Client: `ApiClient::sell_resource`

Retournera une erreur si aucun trader n'est assigné à la station, la quantité demandée est nulle ou la quantité de cette resource dans le cargo est nulle

7. Système

7.1. Tester la connectivité

Permet de tester que la connection avec le serveur distant fonctionne bien.

Ne nécessite pas de clé de joueur pour être appelée.

En cas de succès, retournera `{"ping": "pong"}`

- Endpoint: `/ping`
- Client: `ApiClient::ping`

7.2. Récupérer les logs du système

Lorsque le jeu réalise une action automatiquement, ou si une alerte est lancée, cela sera visible dans les logs associés au joueur.

Ainsi, les logs montreront lorsque:
- Le jeu a commencé pour ce joueur
- Le joueur a perdu
- Un vaisseau a atteint sa destination
- Un vaisseau a arrêté d'extraire des resources car son cargo est plein
- Un vaisseau a été détruit

Et produiront des alertes lorsque:
- Le déchargement des resources d'un vaisseau n'est pas possible (voir Section 3.5)
- Il ne reste que 60 secondes avant que les frais n'épuisent les réserves d'argent

- Endpoint: `/syslogs`
- Client: `ApiClient::get_syslogs`