Rapport SOA : Rendu final

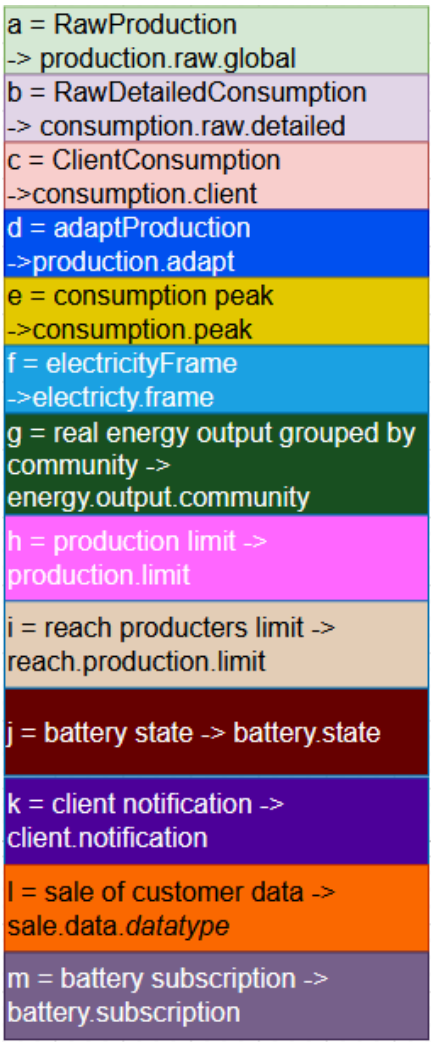
Valentin Campello, Yann Martin d'Escrienne, Lucie Morant, Yohann Tognetti, Tigran Nerssisian

Etat actuel de notre architecture

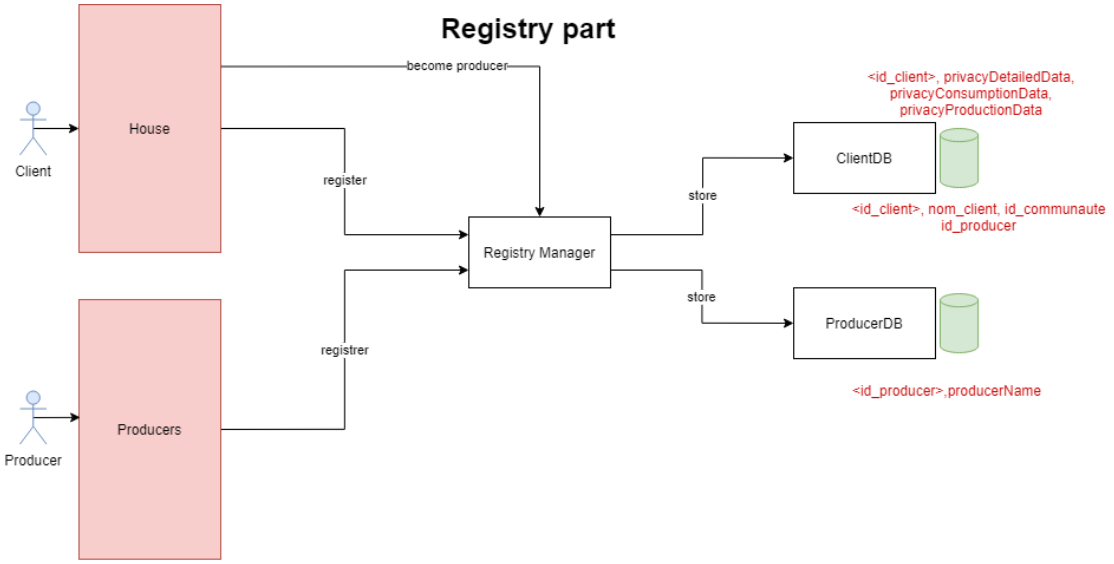
Pour représenter notre architecture, nous avons décidé de découper nos schémas par secteur pour les rendre plus compréhensibles. Tout d'abord, voici une liste des événements présents dans notre architecture.

Nos événements

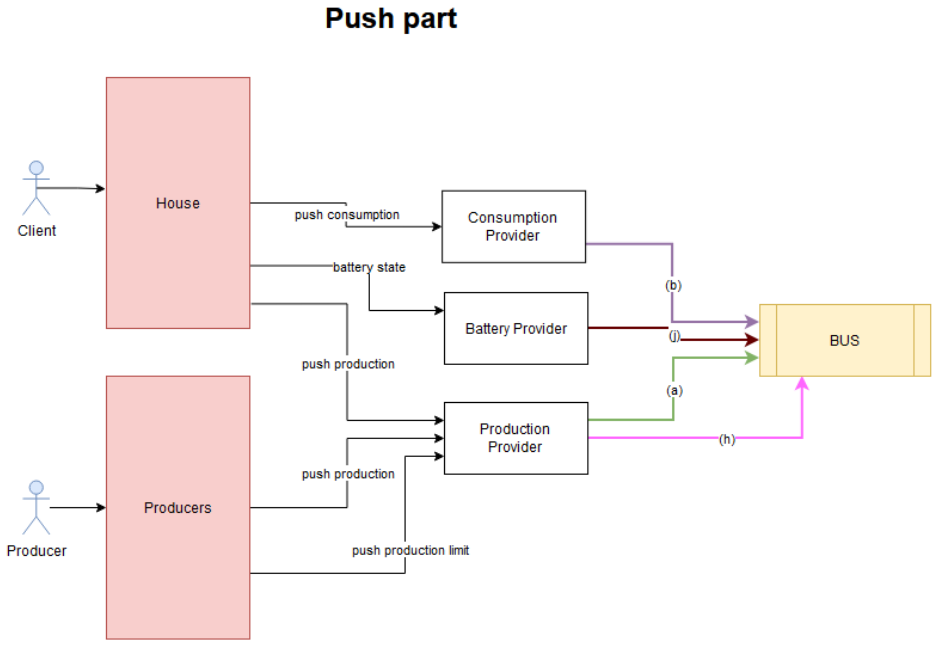
- a = production.raw.global : l'envoi de production d'un producteur pour une date donnée.
- b = consumption.raw.detailed : l'envoi des consommations détaillées de chaque objet des clients pour une date donnée.
- c = consumption.client : l'envoi de la consommation totale d'un client pour une date donnée.
- d = production.adapt : correspond au fait de devoir adapter la consommation côté producteurs. Contient la marge à adapter.
- e = consumption.peak : correspond au fait qu'un pic de consommation a été détecté. Contient la communauté où il y a le pic et la valeur du pic.
- f = electricity.frame : la frame de données consommation/production ainsi que les ID des clients et producteurs pour une période donnée.
- g = energy.output.community : la consommation d'énergie réel de chaque maison d'une communauté.
- h = production.limit : la limite de production d'un producteur.
- i = reach.production.limit : envoyé lorsque tous les producteurs ont atteints leurs limites et qu'il n'est pas possible d'adapter.
- j = battery.state : état de la batterie à une date donnée.
- k = client.notification : notification envoyée vers le client.
- l = sale.data.datatype : est envoyé à chaque achat de données par un partenaire.
- m = battery.subscription : inscription d'une batterie.



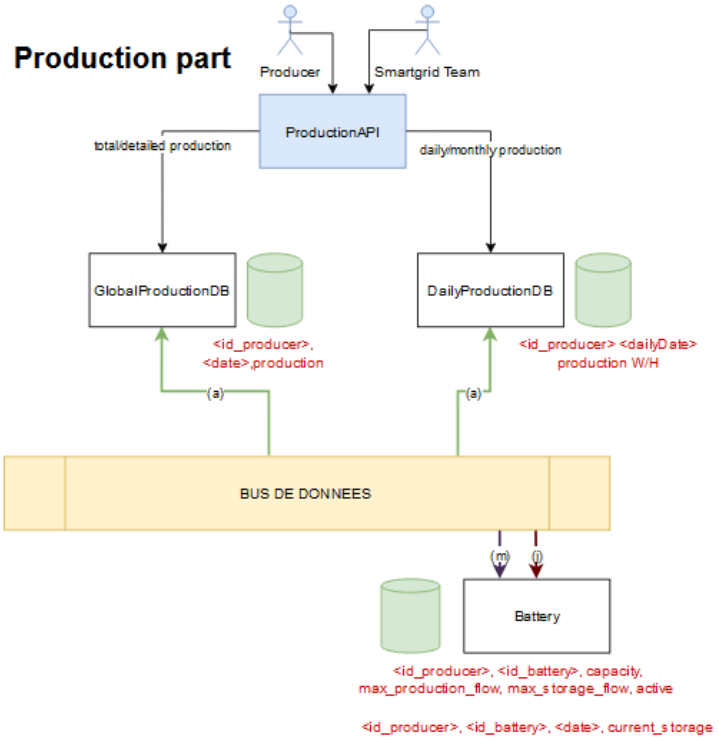
Nos diagrammes d'architectures



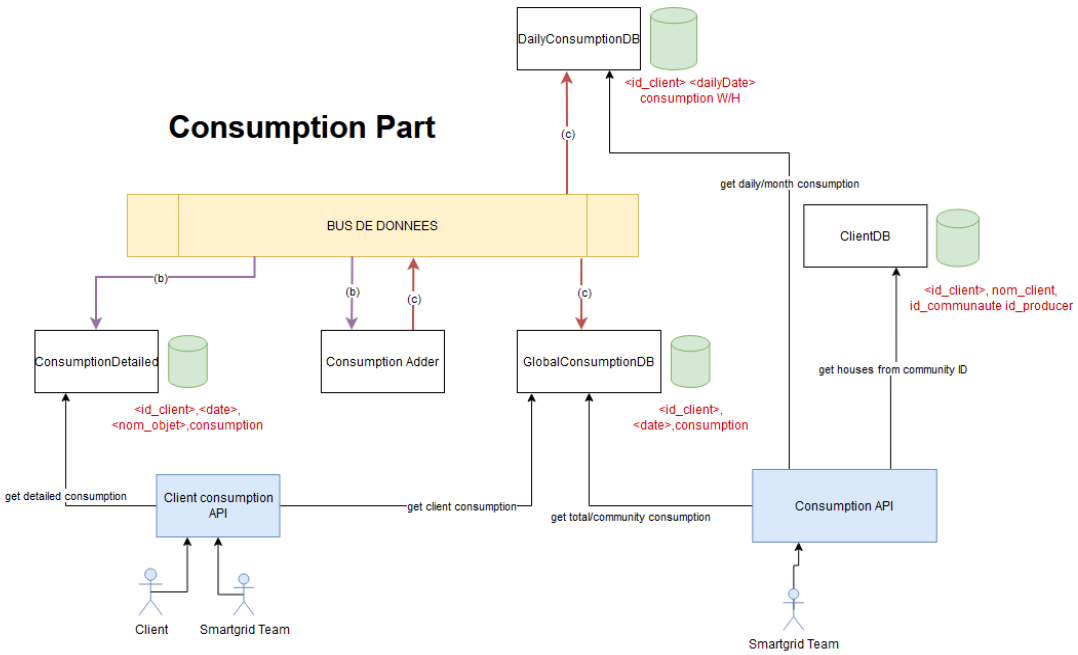
Cette partie correspond à l'inscription des maisons à Smartrix Grid. Les maisons peuvent s'enregistrer en tant que producteur ou consommateur. Tout est sauvegardé dans des bases de données. La base de données cliente contient deux tables : une avec leurs informations tels que leur nom, ID de communauté… et une autre avec les paramètres de confidentialité que le client a accepté lors de son inscription.



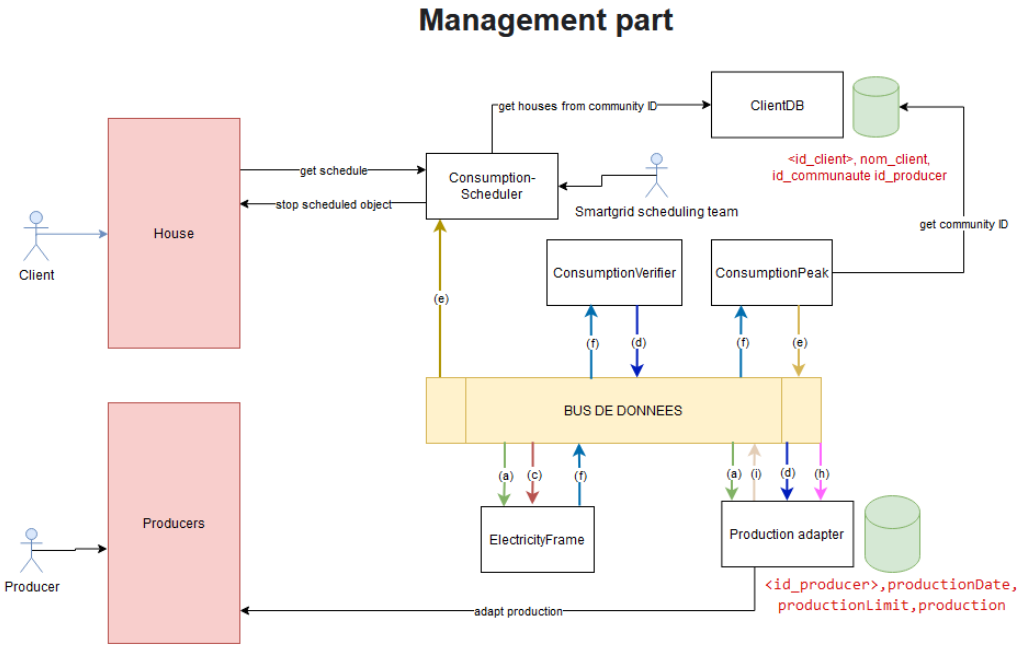
Cette partie correspond à la mise à jour des informations sur la consommation, la production et l'état de la batterie des clients et des fournisseurs d'électricité. Ceux-ci envoient ces informations à trois services différents : Consumption Provider, Battery Provider, Production Provider, qui les font passer à notre bus de données. Nous avons choisi de faire ainsi afin que nos clients et producteurs ne puissent pas directement accéder au bus de données et assurer la sécurité de notre système. De cette façon, ils accèdent à une interface spécialement créée pour qu'ils nous transmettent les données les concernant.



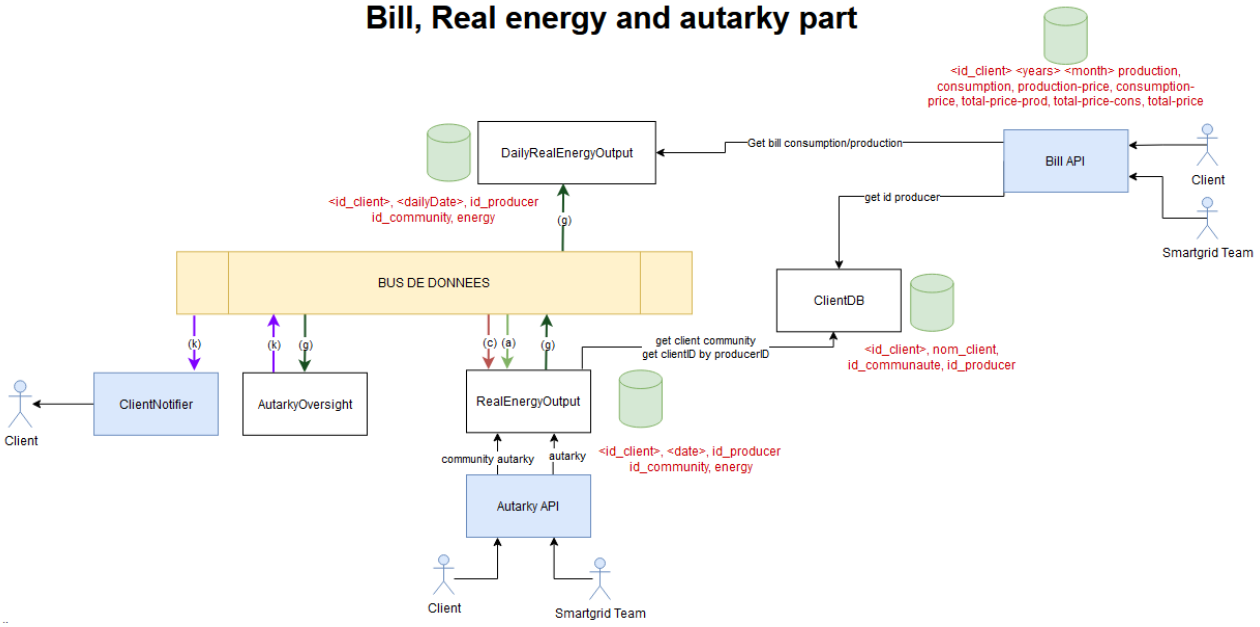
Cette partie correspond à notre gestion de la production et des batteries, GlobalProductionDB et DailyProductionDB consomment l'événement de production, pour le sauvegarder à différent grain (chaque événement pour global et par jour pour daily). ProductionAPI récupère des informations dans ces services. Battery récupère l'événement d'inscription d'une batterie pour inscrire la batterie dans sa base de données ainsi que l'événement du statut de la batterie pour le stocker.



Cette partie concerne tout ce qui est lié à la consommation. A l'aide du bus de données et des différents topics liés à la consommation, nos services vont pouvoir stocker la consommation détaillée d'un client (ConsumptionDetailed), la consommation globale d'un client (GlobalConsumptionDB) et la consommation mensuelle d'un client (DailyConsumptionDB). Les deux API permettent aux clients et aux personnels de Smartrix Grid d'accéder à ses données et de les gérer.

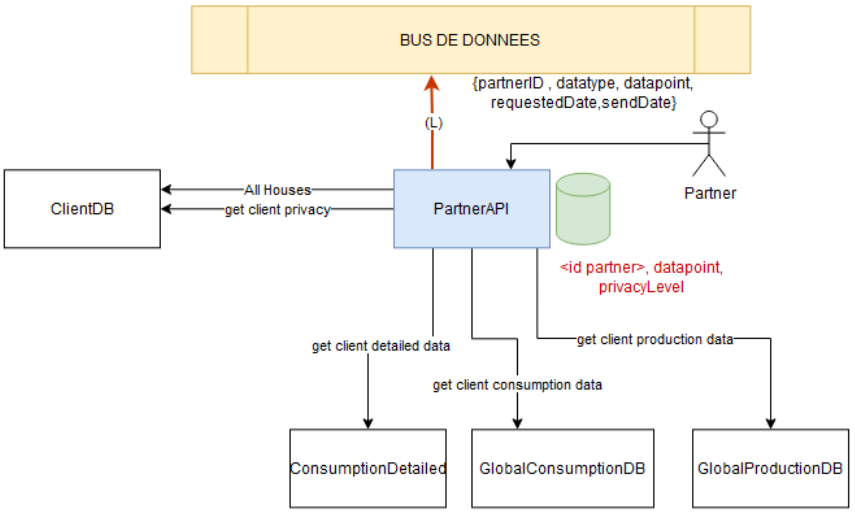


Cette partie s'occupe de gérer la Smartrix Grid. ElectricityFrame écoute les événements de production et consommation et met tout en forme pour créer un événement exploitable par ConsumptionVerifier et ConsumptionPeak. ConsumptionVerifier écoute cet événement et envoie un événement d'adaptation de production quand nécessaire. Production adapter écoute cet événement et demande aux producteurs de s'adapter tout en vérifiant sur les événements de production et de limite de production pour vérifier que cette limite n'est pas dépassée. Dans le cas où la limite est atteinte, il envoie un événement de limite de production atteinte.



Cette partie gère tout ce qui est auto-suffisante et factures. RealEnergyOutput va transformer la consommation et la production du client la même journée afin d'obtenir une énergie, qui est une soustraction entre les deux données. Si l'énergie est positive, cela signifie que le client a plus produit que consommé, et inversement. Cette donnée est ensuite passée via le topic (g) à DailyEnergyOutput, qui va trier les données selon leur date, et à AutarkyOversight, qui vérifie si un client est en auto-suffisance. Si c'est le cas, il envoie un message sur le topic (k), que ClientNotifier réceptionne afin de transmettre cette information au client. A travers les deux autres API, les clients et les employés de Smartrix Grid peuvent savoir si un client ou une communauté est auto-suffisante ou demander une facture.

Client Data



Cette partie s'occupe de la vente de données à des partenaires. Les partenaires ayant un nombre de datapoints suffisants et un niveau de confidentialité suffisant peuvent accéder à des informations dans ClientDB, ConsumptionDetailed, GlobalConsmptionDB et GlobalProductionDB depuis PartnerAPI. Un événement de vente est envoyé quand un partenaire achète des données.

Explications de nos services et interfaces

Comme vu dans notre architecture précédemment, nous avons un total de 30 services (en comptant les interfaces clientes). Chacun apporte une fonctionnalité différente à notre architecture et tous communiquent ensemble afin de couvrir notre MVP d'une manière efficace. Nous allons les présenter un par un.

- **autarky-api :** Ce service sert de point d'entrée à toutes les requêtes concernant l'autarcie d'un client ou d'une communauté, il récupère les informations de **real-energy-output** puis les renvoie.

- **autarky-oversight :** Le service **autarky-oversight** sert à monitorer les changements d'état d'autarcie des maisons et des communautés, pour qu'il soit possible de prévenir les clients lors de ces changements. Il consomme le topic energy.output.community alimenté par **real-energy-output**, et envoie des messages sur le topic client.notification à chaque fois qu'il détecte un changement d'autarcie.

- **battery :** Ce service représente la batterie et, à l'aide du topic battery.subscription qu'il consomme, permet à un client producteur d'énergie d'en inscrire une auprès de Smartrix Grid et d'y stocker ses informations. Elle est ensuite stockée dans une base de données pour que l'on puisse savoir quelle batterie appartient à quel client. Il est également possible de mettre à jour l'état d'une certaine batterie à l'aide du topic battery.state. Il expose également deux endpoints HTTP permettant d'obtenir l'état d'une certaine batterie et les informations la concernant.

- **battery-provider :** Celui-ci est le service approvisionnant les états des batteries des clients. A l'aide de deux méthodes POST, il va envoyer un message soit sur le topic battery.subscription soit sur le topic battery.state afin de permettre d'inscrire une batterie ou de mettre à jour l'état (c'est-à-dire son stockage) de celle-ci. Il communique donc étroitement avec **battery**.

- **bill-api :** Cette API permet aux clients ainsi qu'aux personnels de Smartrix Grid de générer les factures d'électricité des clients. Il leur est possible de soit générer toutes les factures d'une maison depuis son inscription chez Smartrix Grid, soit obtenir la facture d'un mois en particulier, soit générer une facture temporaire du mois courant. Lorsqu'une facture est générée, elle est stockée dans la base de données et il est possible d'y accéder de nouveau librement. En prenant l'exemple d'une facture mensuelle, le calcul se fait ainsi : on récupère via le service **daily-real-energy-output** l'énergie du client pendant le mois. On la parcourt ensuite : si l'énergie d'un jour est positive alors cela signifie que le client a plus produit qu'il a consommé, et inversement. On additionne donc la production et la consommation totale du mois dans deux variables différentes et on les multiplie par le prix de vente et d'achat du W/h. On soustrait ensuite le prix de vente au prix d'achat du client et on obtient de cette façon ce qu'il doit payer (ou ce qu'il a gagné s'il a réussi à vendre plus qu'il a acheté d'électricité).

- **client-consumption-api :** Cette interface permet à un client d'obtenir la consommation totale et la consommation détaillée d'un objet de la maison à une date donnée. Elle appelle le service **global-consumption-database** et **consumption-detailed** afin de récupérer ces informations.

- **client-database :** Comme son nom l'indique, ce service stocke tous les clients de notre système. Il comporte différents endpoints HTTP permettant de récupérer l'ID d'un client, sa communauté, son ID de producteur s'il en a un, quelles politiques de confidentialité il a accepté et tous les clients stockés dans la base de données. Il est également possible à travers les requêtes POST d'enregistrer un nouveau client dans la base de données, de mettre à jour son nom et d'initialiser son ID de producteur puisque celui-ci est vide lorsque le client s'inscrit une première fois.

- **client-notifier :** Le but de ce service est de notifier les clients lors d'un changement de l'état d'autarcie de leur maison (passage en autarcie ou arrêt d'autarcie). Il consomme le topic client.notification alimenté par **autarky-oversight**, et pourrait dans une version future contacter les clients pour les informer, par mail par exemple (actuellement les messages sont juste logués).

- **consumption-adder :** Ce service consomme le topic consumption.raw.detailed, qui nous permet de de récupérer la consommation détaillée d'un utilisateur à une date, avec tous les objets ayant consommés de l'électricité (et combien ceux-ci ont consommé) à cette date. Lors de sa réception, **consumption-adder** va faire la somme de l'électricité qu'a consommé chacun de ces objets et renvoie cette consommation totale ainsi que la date et l'ID du client sur le topic consumption.client.

- **consumption-api :** Cette interface a une utilité similaire que **client-consumption-api**, sauf qu'elle est plutôt dédiée aux employés de Smartrix Grid. En effet, il y est possible de récupérer la consommation totale d'une communauté, de la grille ou d'une maison durant une certaine période. Pour cela, elle fait appel à nos services **global-consumption-database**, **daily-consumption-db**, ainsi que **client-database** afin de récupérer les ID des clients dans une communauté.

- **consumption-detailed :** Dans celui-ci, le topic consumption.raw.detailed est écouté (une explication de ce topic a été faite lors de **consumption-adder**) afin de stocker dans une base de données la consommation détaillée d'un utilisateur durant un certain jour. Grâce à deux requêtes GET, il est possible de récupérer la consommation détaillée d'un client à une date donnée, ainsi que récupérer toutes les consommations détaillées d'un client depuis son inscription à Smartrix Grid.

- **consumption-peak :** Le but de ce service est de vérifier s'il existe un pic de consommation dans une des communautés de Smartrix Grid. Elle est abonnée au topic electricity.frame, qui contient le détail de la consommation totale sur le temps donné (une liste de clients avec leur consommation), la production totale (avec les

producteur et leur production) sur le temps donné et la date de début et de fin de cette frame. A la réception d'un message, une vérification de consommation est faite sur toutes les communautés présentes avec l'aide de **client-database** pour récupérer les communauté des clients (une utilisation de cache se fait également pour limiter les appels). Si un pic est détecté, le service envoie un message sur le topic consumption.peak.

- **consumption-provider :** Il nous sert d'interface entre le client et notre système (point d'entrée du bus). Le client peut, grâce à une requête POST, ajouter sa consommation détaillée pour un certain jour. Le service reçoit cette information et la renvoie dans le bus à travers le topic consumption.raw.detailed.

- **consumption-scheduler :** Bien qu'il ait comme fonction la planification des objets régulables branchés dans une maison, ce service nous sert également à réguler la consommation des clients lorsque nous recevons qu'il y a un pic de consommation. En effet, le topic consumption.peak y est écouté et, lorsqu'un message y est reçu, tous les objets planifiables dans la communauté ciblée sont coupés afin d'éviter au maximum une surtension.

- **consumption-verifier :** Ce service a pour but de vérifier si la production est bien égale à la consommation. Le topic electricity.frame (cf **consumption-peak**) est consommé afin de faire cette comparaison. S'il s'avère qu'elles ne sont pas égales, un message va être envoyé sur le topic production.adapt pour signaler que la production doit s'adapter.

- **daily-consumption-db :** Cette base de données stocke la consommation totale d'un client pendant un jour. Le topic consumption.client est écouté : on y récupère l'ID du client, la consommation et la date où celle-ci a été enregistrée. Avant d'être stockée, la consommation est convertie en W/h. Ensuite, s'il existe déjà une consommation pour la date et le client donnés dans la base de données, on met à jour cette ligne en sommant cette consommation et celle reçue. Sinon, une nouvelle ligne est créée. Deux endpoints HTTP permettent de récupérer la consommation journalière à une date donnée d'un client ou de récupérer sa consommation sur une certaine période de temps.

- **daily-production-db :** Le fonctionnement est le même que pour **daily-consumption-db**. Les données sont récupérées via le topic production.raw.global et on stocke ensuite la production totale d'un fournisseur durant un jour. Il y est possible de récupérer la production journalière d'un fournisseur, ainsi que sa production sur une certaine période de temps.

- **daily-real-energy-output :** Le fonctionnement est encore similaire à **daily-consumption-db** et **daily-production-db**, vu que nous raisonnons sur quelque chose de journalier. Le topic energy.output.community est consommé : celui-ci comporte la liste des clients d'une même communauté avec leur énergie (si elle est

positive, cela signifie que le client a plus produit que consommé durant la journée, et inversement) à une certaine date. Cela est ensuite stocké dans la base de données et il est possible de récupérer les valeurs d'un jour ou d'une période de temps à travers des endpoints HTTP.

- **electricity-frame :** Le but de ce service est de récupérer les production et les consommation d'un temps t à un temps t-5 (afin de s'assurer que les valeurs ont bien été réceptionnées) pour ensuite les envoyer aux services écoutant le topic electricity.frame. Il est donc inscrit aux topics consumption.client et production.raw.global pour obtenir ces valeurs.

- **global-consumption-database :** Il y est stocké la consommation d'un client à une certaine date. Il reçoit via le topic consumption.client l'ID d'un client, sa date de consommation et sa consommation. Il la stocke ensuite directement dans la base de données. Trois requêtes GET permettent de récupérer la consommation totale de la grille, la consommation totale d'une communauté ou d'un client.

- **global-production-database :** Même fonctionnement que pour **global-consumption-database**. On stocke la production d'un fournisseur à une certaine date. Les données sont reçues via le topic production.raw.detailed puis stockées dans la base de données. Le service ne comporte que deux requêtes GET, retournant soit la production totale des fournisseurs électriques soit la production d'un fournisseur.

- **house :** Ce service représente le client à travers sa maison. Celle-ci peut avoir une batterie, ajouter de nouveaux objets, les gérer, communiquer sa consommation, sa production (si elle existe), l'état de sa batterie… ainsi que demander à devenir un fournisseur local ou à s'inscrire à Smartrix Grid si elle n'est pas déjà cliente. Elle joue donc un rôle majeur dans notre architecture. Régulièrement, elle communique sa consommation, sa production si elle a un statut de fournisseur et l'état de sa batterie à nos services **consumption-provider**, **production-provider** et **battery-provider** respectivement. Par simplification de gestion, toutes les maisons sont regroupées dans ce "service".

- **partner-api :** Cette interface peut être accédé par les partenaires commerciaux de Smartrix Grid souhaitant acheter des données utilisateurs à celui-ci. Les différents endpoints HTTP permettent d'ajouter un nouveau partenaire, avec son ID, ses data points et son niveau de confiance dans la base de données, d'obtenir les informations d'un partenaire, d'ajouter des data points et de demander d'acheter la consommation détaillée des utilisateur, leur consommation globale ou leur production globale. L'achat de données repose sur le niveau de confiance du partenaire et si les clients ont accepté la police de confidentialité. Nous avons trois niveaux de confiance : le premier autorise le partenaire à acheter la consommation globale, la deuxième la production globale et la troisième les données détaillées d'un client. Cela signifie donc qu'un partenaire doit avoir un grand niveau de confiance

afin de pouvoir accéder aux données plus sensibles. Il doit également avoir le nombre de data point nécessaire pour l'achat des données des utilisateurs. Il ne récupérera que les données des utilisateurs ayant accepté la politique de confidentialité de ce niveau donné.

- **producer-database :** Celui-ci est une base de données stockant les fournisseurs de Smartrix Grid. Il y est donc possible à travers les différentes requêtes HTTP de devenir un fournisseur, de mettre à jour son nom de fournisseur (dans le cas où une entreprise change de nom, tel que Engie) et récupérer tous les fournisseurs ou un fournisseur en particulier.

- **producers :** A travers ce service, nous pouvons ajouter un nouveau fournisseur, communiquer sa production à une certaine date ainsi que modifier la production (dans le cas où celle-ci est insuffisante ou en excès par rapport à la consommation) et récupérer sa valeur. Il sert donc d'interface entre les fournisseurs et notre système. Il regroupe ainsi plusieurs fournisseurs pour simplifier la gestion.

- **production-adapter :** Comme son nom l'indique, sa fonction est de communiquer au service **producers** que la production doit être modifiée. Pour être au courant de ceci, le topic production.adapt est consommé. Dès réception d'un message avec la quantité de W à ajouter, un appel vers **producers** est effectué afin que la production soit changée. La production se répartie ensuite chez les producteurs dans **producers.** Dans le cas où la limite de chaque producteur est atteinte, un événement est envoyé sur le topic reach.production.limit (non exploité). Pour ce faire, ce service consomme également le topic production.limit qui correspond au changement de limite de production d'un producteur.

- **production-api :** Cette API a été pensée pour les fournisseurs mais sert aussi à l'équipe de Smartrix Grid. Celle-ci leur permet de connaître la production totale dans la grille, la production d'un fournisseur en particulier ainsi que la production sur un jour ou sur une période de temps d'un fournisseur. Pour cela, elle fait appel à **global-production-database** et **daily-production-db**.

- **production-provider :** Ce service sert d'interface entre les fournisseurs et notre bus. Ceux-ci communiquent à travers la requête POST leur production à une date donnée et cette information est ensuite envoyée à travers le bus à l'aide du topic production.raw.global. Lors de leur inscription ou changement de limite de production, ce service envoie également l'évènement production.limit dans le bus. Il y est contenu l'ID du producteur et la nouvelle limite de production de celui-ci.

- **real-energy-output :** Le but de ce service est d'avoir une vue précise de la différence de production et de consommation de chaque client / communauté. Pour ce faire, il maintient dans sa base de données un score qu'il met à jour à chaque production et consommation. Il consomme donc les topics de production production.raw.global et consommation consumption.client et envoie des messages sur le topic

energy.output.community. Pour remplir ce topic, real-energy-output fait des messages par communauté, contenant l'état de production-consommation de la communauté ainsi que celui de chaque maison dans cette communauté à t-5, pour être sûr que toutes les informations soient arrivées. Il expose aussi des endpoints http permettant de récupérer ces "score" pour une date précise, pour une maison ou une communauté. Ces endpoints sont exploités par **autarky-api**.

- **registry-manager :** Son nom décrit sa fonction : il sert de registre d'inscription. A travers les différents endpoints HTTP, les clients ou les fournisseurs peuvent s'inscrire. Il est également possible pour un client de devenir un fournisseur local. Puis chacun peut changer leur nom s'il le souhaite. Le service va ensuite contacter **client-database** et **producer-database** pour les ajouter ou effectuer les modifications requises.

Notre compréhension du sujet et explication de nos choix de conception

Le principe du sujet est de se rapprocher le plus possible d'une réelle implémentation d'un fournisseur électrique. Notre architecture doit pouvoir répondre aux besoins de chaque user story d'une manière viable, minimaliste et réaliste.

Niveau client, ceux-ci doivent être capable, quand ils le souhaitent, de visualiser leur consommation, avec le détail de qu'est-ce qui a consommé et quand, ainsi qu'obtenir leurs factures mensuelles et une prévision de leur facture du mois. Ils peuvent également fournir leur propre électricité grâce à des panneaux solaires, leur donnant l'opportunité d'être auto-suffisants mais aussi de vendre leur surplus de consommation. S'ils sont auto-suffisants, nous devons le leur notifier, ainsi que s'ils perdent ce statut d'auto-suffisance.

Quant aux fournisseurs d'électricité, nous devons être capable de leur communiquer la consommation totale de la grille et leur notifier si un pic de consommation est aperçu dans une communauté afin qu'ils puissent adapter leur production en conséquence (bien que celle-ci soit limitée). Ils peuvent également acheter le surplus de l'énergie produite par les clients pour éviter du gaspillage d'électricité.

Ensuite, le PDG de Smartrix Grid doit être capable d'effectuer plusieurs actions sur la grille : avoir une vue d'ensemble sur la consommation d'énergie de ces clients (qui sont divisés dans des communautés), avoir un contrôle sur les objets électriques non essentiels (notamment pour les couper en cas de pic de consommation), planifier la charge des voitures électriques des clients, savoir si une certaine communauté est en autarcie, pouvoir vendre les données de consommation / production de ses clients…

Enfin, nous devons répondre aux besoins des autres acteurs. Il doit être possible de recharger les voitures électriques de nuit, ainsi que d'installer des panneaux solaires aux maisons des clients. Le surplus d'énergie de ces panneaux solaires peut être stocké dans des batteries fournies aux clients. Ces besoins sont étroitement liés à ceux des trois précédents acteurs cités puisque leur ajout permet à ces acteurs d'effectuer de nouvelles actions et de réfléchir à de nouvelles contraintes.

Nous allons présenter celles que nous avons identifiées.

Contraintes

- Notre architecture doit marcher avec une notion de temps pour correspondre à des besoins de temps réel,
- Besoin de précision sur la différence de production / consommation et de réactivité pour prévenir le producteur en cas de différence,
- Les maisons doivent toujours pouvoir accéder à l'électricité demandée, sauf lors de cas spéciaux où les objets branchés peuvent attendre et demander un planning de consommation (voitures électriques par exemple),
- Les clients doivent être capables d'accéder facilement à leur consommation et leurs factures sans forcément passer par des services de Smartrix Grid,
- Les maisons doivent être regroupées en communauté mais, lors de l'enregistrement d'une maison, celle-ci ne doit pas être bloquée par des problèmes internes à Smartrix Grid. Elle doit pouvoir être ajoutée rapidement dans une communauté, que ce soit transparent pour elle.

- Les communautés doivent être monitorées de façon à ce qu'il n'y ait pas de pic de consommation,
- Une batterie ne peut pas charger instantanément et possède donc un flux maximum de chargement et de déchargement,
- S'il y a un pic de consommation, les objets branchés non essentiels doivent pouvoir être arrêtée afin de ne pas court circuiter la grille,
- La production des fournisseurs est limitée,
- Un client doit avoir accepté la politique de confidentialité afin que Smartrix Grid puisse vendre ses données de consommation et production,
- Une batterie a un stockage limité.

Choix de simplifications

- La date des demandes est passée dans les requêtes, car implémenter une horloge émulée et commune nous permettant de tester différentes périodes en peu de temps était trop compliqué dans la limite de temps que nous avions.
- Les communautés sont créées de façon arbitraire par Smartrix Grid, et non en fonction de la localisation des maisons (car trop compliqué à implémenter en si peu de temps). Cela pourra éventuellement évoluer dans le futur.
- Les maisons commencent déjà avec une consommation de base. Cela correspond à des appareils électroniques consommant constamment.
- Nous avons créé plusieurs APIs, qui servent d'interfaces à nos différents clients, afin qu'ils puissent récupérer les informations qui les intéressent facilement et rapidement. Celles-ci sont aussi une sécurité pour notre système car, de cette façon, les clients ne touchent pas directement aux services de notre architecture.
- Une communauté est en autarcie si la consommation et la production de la communauté est égale. Cela signifie qu'il n'y a pas besoin que toutes les maisons soient auto-suffisantes pour que la communauté le soit également. Il suffit que la production d'une ou plusieurs maisons atteignent la consommation d'une ou plusieurs maisons.
- Les objets branchés non essentiels d'une maison sont, pour nous, tous les objets programmables (qui ont un planning). Lorsqu'il y a un pic de communauté, ce sont eux que nous débranchons sans distinction.
- Lorsqu'un client devient ou n'est plus autosuffisant, nous écrivons seulement un log. L'implémentation d'une notification réelle aux clients aurait été compliquée, étant donné le temps qui nous restait.
- Lorsque les producteurs ne peuvent pas suivre l'adaptation de production car ils ont tous atteint leur limite, un événement est lancé dans le bus mais n'est jamais traité.
- Les producteurs s'inscrivent avec une limite par défaut qu'il est possible de changer.
- Lorsqu'un client s'inscrit, il accepte toutes les politiques de confidentialité.
- Il n'y a actuellement que 3 maisons par communauté

Couverture des users stories par notre architecture

Notre architecture permet au système de couvrir toutes les stories selon notre compréhension du sujet. En effet, nous avons créé notre système à l'aide de celles-ci, il répond ainsi au mieux aux besoins évoqués par les clients.

En ce qui concerne les besoins de Pierre et Marie, il leur est possible de voir la consommation de leur maison (/*client-consumption-api/house-global-consumptionl*) mais également de chacun des objets de celle-ci (/*client-consumption-api/house-detailed-consumptionl:{name}*) en donnant la date où ils veulent voir la consommation du foyer. Leurs différentes factures sont accessibles (*/bill-api/bill-for-house*) en entrant l'année et le mois souhaités. Une facture temporaire peut être également générée (*/bill-api/generate-temporary-bill*) pour le mois en cours. S'ils installent des panneaux solaires, ceux-ci sont inscrits en tant que producteur local et peuvent produire de l'électricité. De plus, s'ils ont une batterie, il leur est possible de vérifier son stockage (*/battery/get-battery-state*). Au niveau de leur auto-suffisante, ils peuvent vérifier s'ils l'ont ou s'ils la perdent (*/client-notifier/get-house-message ou autarky-api/get-house-autarky*).

Pour Nikola, il peut demander à la production de s'adapter à la consommation totale de la grille (*/producers/change-production*) et, si un pic de consommation est détecté, un message est envoyé automatiquement afin que la consommation des clients diminue. Il lui est aussi possible d'acheter le surplus d'énergie produit par les clients.

Ensuite, Charles, quant à lui, peut obtenir la consommation de tous ses clients (*/consumption-api/total-consumption*), de la consommation de certaines communautés (*/consumption-api/community-consumption*) ou bien même d'une maison en particulier (*/consumption-api/daily-consumption*) sur une période de temps. Il lui est également possible d'envoyer des plannings de consommation pour les objets paramétrables de ses clients à l'aide de consumption-scheduler quand ceux-ci lui font une demande en envoyant l'ID de leur maison (/*schedule*) et d'avoir un contrôle sur les objets programmables. A travers une API, il peut consulter si une certaine communauté est auto-suffisante (*autarky-api/get-community-autarky*), ainsi que permettre la vente de données clients avec Partner API (*partner-api/request-**). Un service d'inscription et un autre de stockage des données des clients permet de faciliter tous les échanges de nos services avec les maisons ou producteurs et de répondre au mieux à des demandes de management des clients.

Enfin, nous avons les derniers acteurs tels qu'Elon, Albert et Thomas. Elon peut charger ses véhicules la nuit en affectant directement un planning de consommation à ses objets paramétrables. Il peut le faire sans passer directement par consumption-scheduler. Il est possible pour Albert d'installer des panneaux solaires chez nos clients afin qu'ils aient une production locale (cf paragraphe sur la couverture des besoins clients). Et Thomas peut vendre des batteries à nos clients ayant déjà des panneaux solaires afin qu'ils stockent leur surplus d'énergie (cf paragraphe sur la couverture des besoins clients).

Scénarios créés

Lors de ce rendu, nous avons créé plusieurs scénarios, présentés ci-dessous, en plus des user stories du sujet. Chaque semaine, nous les faisons évoluer afin d'intégrer les nouvelles fonctionnalités apportées par les user stories de la semaine. Ils nous semblent donc pertinents et en adéquation avec ce que l'on attend de notre projet; ce sont des actions "basiques" et essentielles au bon fonctionnement de Smartrix Grid. Nous pensons également que ce sont de bons scénarios car ils passent dans tous nos services et démontrent ainsi une bonne communication entre eux.

Scénario 1 : Inscription, objets paramétrables et adaptation de la production avec la consommation

1. Une nouvelle maison s'inscrit à Smartrix Grid,
2. Smartrix Grid attribue un ID à la maison,
3. Le client peut voir sa consommation depuis son boitier,
4. Un nouveau fournisseur d'électricité s'inscrit à Smartrix Grid,
5. Smartrix Grid attribue un ID au fournisseur,
6. On vérifie que la consommation est égale à la production,
7. Un objet paramétrable est branché à la maison,
8. On demande un planning de consommation pour cet objet,
9. On voit que l'objet consomme à son heure d'activité,
10. On constate que la production n'est plus égale à la consommation,
11. On vérifie que la production s'est bien adaptée,
12. Trois mois s'écoulent, le client veut voir sa facture du premier mois et du mois courant.

Ce scénario permet de, tout d'abord, vérifier l'inscription d'une maison et d'un fournisseur d'électricité auprès de notre **registry-manager**. Lorsqu'ils ont été inscrits, on s'assure qu'un ID unique leur a bien été attribué et que le client peut accéder à sa consommation de manière interne. Ensuite, après branchement d'un nouvel objet paramétrable à la maison et création d'un planning de consommation pour celui-ci, on vérifie que l'objet consomme bien de l'électricité à l'horaire transmise par le planning. Ce nouvel objet change la consommation globale de la grille et on s'assure donc que la production a bien pu s'adapter à cette nouvelle valeur. Notre dernière étape permet de générer une facture prévisionnelle pour le client (celle du mois courant) et une autre facture complète pour son premier mois chez Smartrix Grid.

Scénario 2 : Gestion de la consommation, achat des données, pic et autarky dans une communauté

1. On a des maisons dans une communauté et une autre dans une autre communauté,
2. On ajoute plein d'objets planifiables et non planifiables dans les maisons,
3. On regarde la consommation totale de toutes les maisons,
4. On regarde la consommation dans une communauté,
5. On détecte un pic de consommation dans cette communauté,
6. On demande aux objets planifiables d'arrêter de charger dans cette communauté,

7. On remarque qu'il n'y a maintenant plus de pic,
8. On regarde si la communauté est en autarcie et ce n'est pas le cas,
9. On ajoute la production suffisante dans une des maisons de la communauté pour passer en autarcie,
10. On ajoute un nouveau partenaire avec des datapoints, il récupère les données de production des clients,
11. Le partenaire récupère les données de consommation détaillée des clients mais n'a plus de datapoints, il en reprend et retente mais il n'a pas le niveau de confiance requis.

Ce scénario permet en premier lieu de vérifier que les maisons sont bien dans des communautés, et que si ces maisons consomment (en ajoutant des objets) on a bien une consommation dans leurs communautés. Par la suite, ce scénario montre qu'en cas de pic détecté dans une communauté, les objets planifiables des maisons de cette communauté arrêtent de se charger. Après ça, ce scénario vérifie l'état d'autarcie de la communauté. Si l'autarcie n'est pas atteinte, on ajoute des objets producteurs d'électricité dans les maisons de cette communauté jusqu'à ce que celle-ci soit en autarcie. Enfin, ce scénario montre qu'un partenaire peut acheter des données s'il a le niveau de confiance requis et des datapoints suffisants.

Scénario 3 : Production d'une maison, autarcie et notification

1. On inscrit une maison avec des objets consommant de l'électricité et on l'inscrit en tant que producteur,
2. On ajoute à la maison un objet produisant de l'électricité et une batterie,
3. On remarque que la consommation est réduite mais toujours positive,
4. On regarde la production de la maison depuis SmartGrid, celle-ci est accessible mais pas suffisante,
5. On voit que la maison n'est pas en autarcie et que la batterie n'a rien stocké,
6. On ajoute un deuxième objet producteur d'énergie dans la maison,
7. On voit que la production est maintenant plus grande que la consommation,
8. On voit maintenant que la maison est passée en autarcie et que le client a reçu une notification,
9. On constate que la batterie a stocké le surplus d'énergie,
10. On désactive le réacteur DIY pour que la consommation soit plus élevée que la production et que la batterie soit utilisée.

Ce scénario montre un cas où la maison produit sa propre électricité, jusqu'à atteindre l'auto-suffisance. Au début de notre scénario, le client n'a pas assez produit pour être considéré comme autosuffisant et n'a donc pas reçu une notification à ce sujet. L'ajout d'un deuxième objet de production d'énergie (panneaux solaires par exemple) permet au client de produire plus qu'il ne consomme et donc d'atteindre l'autarcie. Une notification lui est envoyée à ce sujet. Le surplus de production est ensuite stocké dans la batterie du client, qui est utilisée lorsque sa consommation est plus élevée que sa production afin qu'il conserve le plus longtemps son statut d'auto-suffisance.

Diagramme de séquence de chaque scénario

Afin d'apporter plus de clarté à notre rapport, nous avons décidé d'avoir tous nos diagrammes de séquence en annexe. Pour chaque étape de nos scénarios, nous avons donc mis en lien les diagrammes de séquence concernés.

Scénario 1 : Inscription, objets paramétrables et adaptation de la production avec la consommation

1. Une nouvelle maison s'inscrit à Smartrix Grid : Ajout d'une maison pour inscrire la maison et récupérer un id, ajout d'un objet pour générer de la consommation
2. Smartrix Grid attribue un ID à la maison : id récupéré dans l'étape précédente Ajout d'une maison
3. Le client peut voir sa consommation depuis son boitier : Vérification de la consommation sur le boîtier d'une maison, vérification depuis clientConsumptionAPI
4. Un nouveau fournisseur d'électricité s'inscrit à Smartrix Grid : Ajout d'un producteur
5. Smartrix Grid attribue un ID au fournisseur : id récupéré dans l'étape précédente Ajout d'un producteur
6. On vérifie que la consommation est égale à la production : Consommation globale depuis consumptionAPI, production globale depuis productionAPI
7. Un objet paramétrable est branché à la maison : Ajout d'un objet paramétrable et on vérifie les objets présents
8. On demande un planning de consommation pour cet objet : On demande un planning pour un objet paramétrable
9. On voit que l'objet consomme à son heure d'activité : On récupère la consommation de l'objet à l'heure où il est censé consommer
10. On constate que la production n'est plus égale à la consommation : On voit que la consommation globale depuis consumptionAPI n'est plus égale à la production globale depuis productionAPI
11. On vérifie que la production s'est bien adaptée : On redemande la production globale depuis productionAPI pour vérifier qu'elle s'est adaptée
12. Trois mois s'écoulent, le client veut voir sa facture du premier mois et du mois courant : On récupère la facture d'un mois précédent, puis celle partielle du mois en cours

Scénario 2 : Gestion de la consommation, achat des données, pic et autarky dans une communauté

1. On a des maisons dans une communauté et une autre dans une autre communauté : On regarde les maisons enregistrées
2. On ajoute plein d'objets planifiables et non planifiables dans les maisons : Ajout d'un objet, on récupère le planning des objets planifiables
3. On regarde la consommation totale de toutes les maisons : Obtention de la consommation totale
4. On regarde la consommation dans une communauté : Obtention de la consommation d'une communauté

5. On détecte un pic de consommation dans cette communauté : Identification d'un pic de consommation
6. On demande aux objets planifiables d'arrêter de charger dans cette communauté : Les objets planifiables sont arrêtés
7. On remarque qu'il n'y a maintenant plus de pic : Obtention de la consommation d'une communauté
8. On regarde si la communauté est en autarcie et ce n'est pas le cas : Vérification de l'auto-suffisance de la communauté, on regarde si la communauté a reçu une notification à ce sujet
9. On ajoute la production suffisante dans une des maisons de la communauté pour passer en autarcie : Un client devient un fournisseur, ajout d'un objet, vérification de l'auto-suffisance de la communauté, on regarde si la communauté a reçu une notification à ce sujet
10. On ajoute un nouveau partenaire avec des datapoints, il récupère les données de production des clients : Ajout d'un partenaire, récupération de ses informations, achat de données de production
11. Le partenaire récupère les données de consommation détaillée des clients mais n'a plus de datapoints, il en reprend et retente mais il n'a pas le niveau de confiance requis : Ajout de datapoints, récupération des informations du partenaire, tentative d'achat des données de consommation détaillée

Scénario 3 : Production d'une maison, autarcie et notification

1. On inscrit une maison avec des objets consommant de l'électricité et on l'inscrit en tant que producteur : Ajout d'une maison, le client devient un fournisseur, un objet est ajouté
2. On ajoute à la maison un objet produisant de l'électricité et une batterie : Ajout d'un objet, ajout d'une batterie, on récupère les informations de la batterie
3. On remarque que la consommation est réduite mais toujours positive : Vérification de la consommation de la maison
4. On regarde la production de la maison depuis SmartGrid, celle-ci est accessible mais pas suffisante : Vérification de la production de la maison
5. On voit que la maison n'est pas en autarcie et que la batterie n'a rien stocké : Vérification de l'auto-suffisance de la maison, on regarde si la maison a reçu une notification, on récupère les informations de la batterie
6. On ajoute un deuxième objet producteur d'énergie dans la maison : Ajout d'un objet
7. On voit que la production est maintenant plus grande que la consommation : Vérification de la consommation de la maison, vérification de la production de la maison
8. On voit maintenant que la maison est passée en autarcie et que le client a reçu une notification : Vérification de l'auto-suffisance de la maison, on regarde si la maison a reçu une notification
9. On constate que la batterie a stocké le surplus d'énergie : On récupère les informations de la batterie
10. On désactive le réacteur DIY pour que la consommation soit plus élevée que la production et que la batterie soit utilisée : On désactive un objet, on récupère les informations de la batterie

Limites et contraintes de notre architecture

Points of failure

Dans nos services, ElectricityFrame est un point of failure. Il agrège les données de consommation et de production pour créer un événement global pour chaque période de temps unitaire (une minute fictive ici). Il subit donc une lourde charge et la façon dont nous l'avons implémenté le rend difficile à dupliquer. Il faudrait donc revoir ce service qui est devenu un point of failure de notre architecture.

Points of high loads

Les bases de données de production et de consommation subissent un ralentissement lorsque de nombreuses maisons / producteurs interagissent avec le système. Il est possible que comme pour ce MVP, toutes nos bases de données soient en fait une seule et même base, que séparer les bases soit une solution. Un autre solution serait de dupliquer les bases comme vu en cours, avec une base d'écriture et une de lecture par exemple.

Problème des données

Dans notre architecture, les données de production des maisons et des producteurs ne sont pas séparées, ce qui peut causer des problèmes. Nous aimerions changer cela pour avoir des données différentes pour ces deux cas.

Si nous devions nous rapprocher le plus possible d'un cas réel, il nous manque des données. Il faudrait donc que nous étoffions nos données.

Simplification

Le ConsumptionScheduler donne actuellement un planning arbitraire aux objets, il conviendrait par la suite de récupérer des données de consommation de la communauté de la maison ayant fait la demande pour faire des prédictions et donner un planning plus juste aux objets.

Duplication

La plupart de nos services sont théoriquement duplicables et d'autres pourraient l'être avec quelques petites modifications, mais aucun test n'a été fait sur cette partie par manque de temps.

Recovery

Actuellement, rien dans notre architecture ne permet de relancer un service tombé sans le faire à la main. Il pourrait être intéressant de s'y pencher par la suite.

Bus

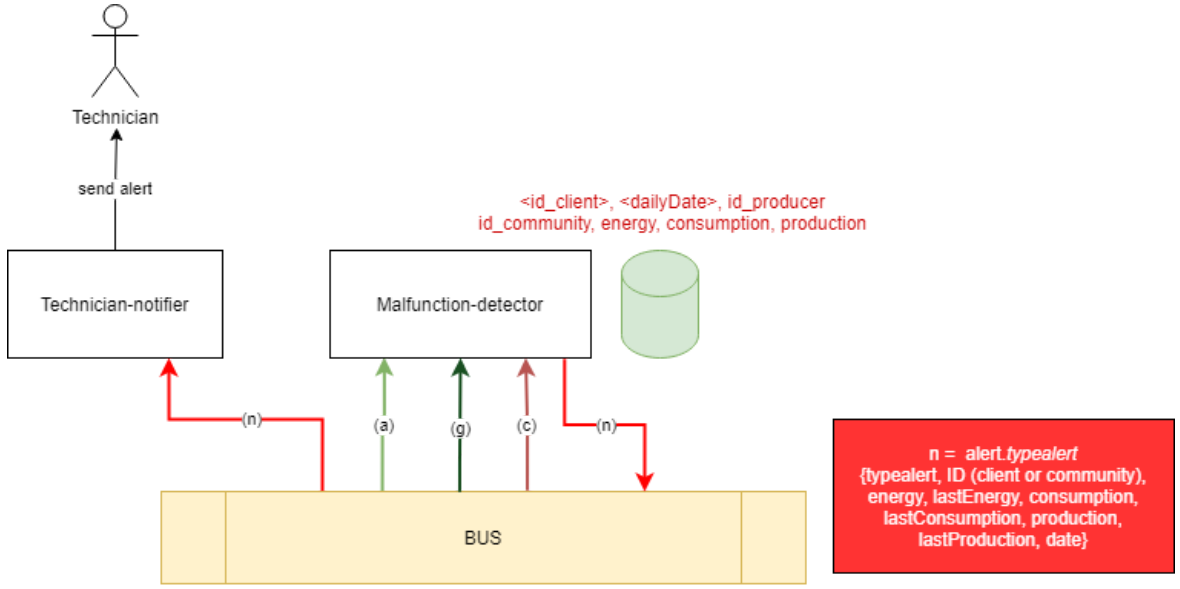
Dans un cas futur, où nous avons réglé nos problèmes de duplication, points of failure et points of high load, peut-être que le partitionnement des topics deviendra nécessaire. Pour l'instant, nous n'avons pas de problèmes avec cette partie de notre architecture.

Ce qui nous devrons modifier pour couvrir toutes les user stories

Par manque de temps, nous n'avons pas pu créer d'étapes dans les scénarios pour montrer que les producteurs possèdent une limite réelle et ainsi démontrer la user story 19. S'il n'est pas possible d'adapter la production car la limite serait atteinte, un événement (i) est envoyé dans le bus. Il n'est actuellement pas consommé. Ce cas est cependant géré dans le code, il n'est juste pas démontré. De plus, lorsqu'il est demandé aux producteurs de s'adapter, ceux-ci se partagent la charge : celui qui produit actuellement le moins prend le maximum de production possible modulo sa limite et ainsi de suite.

Pour ce qui concerne l'user story 16, nous avons un script permettant de créer 1000 maisons, qui peuvent push des informations à chaque tick. Cet user story n'est pas mis en valeur dans nos scénarios, il faudrait donc le faire dans le futur.

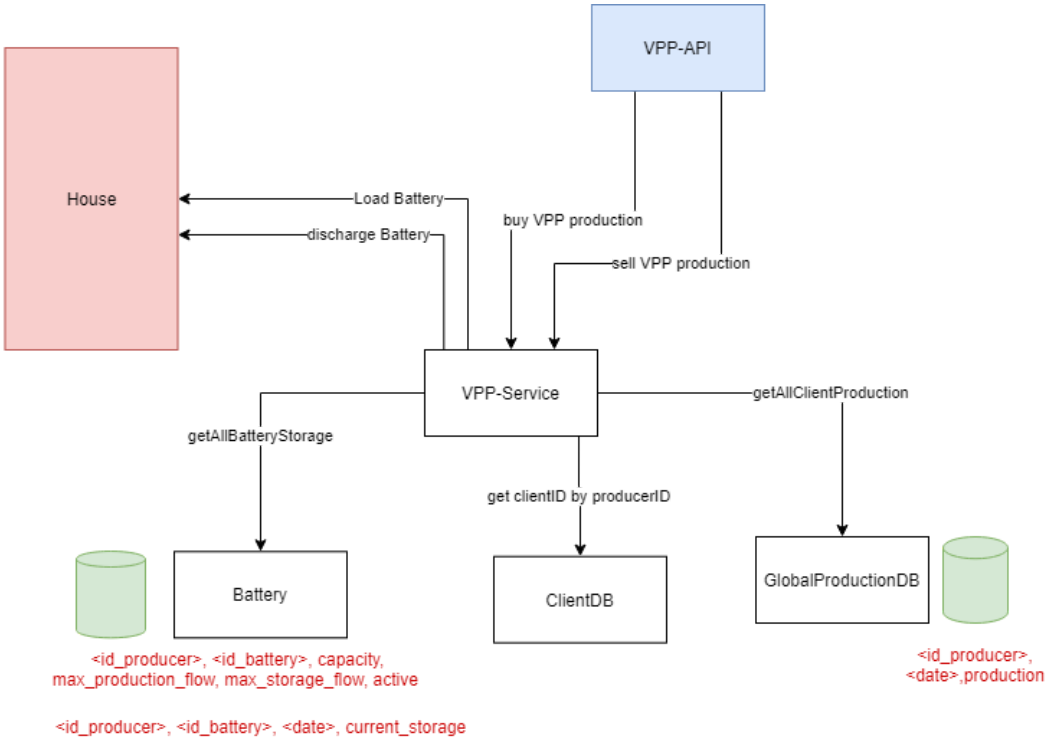
Implémentation de l'user story 21



Afin de détecter s'il y a un mauvais fonctionnement au niveau des données des maisons et des communautés du système, nous ajouterions deux nouveaux services, ayant qu'un nouveau topic alert.typealert où typealert correspond à un type d'alerte définis. Le principe de Malfunction-detector serait d'écouter les différentes informations sur la consommation, la production et la consommation réelle de nos clients et que, si celles-ci ne sont pas cohérentes (une maison ne renvoie plus de données pendant une certaine période de temps par exemple), un message est envoyé sur le topic alert.typealert avec les détails de l'anomalie. La détection se fera aussi au niveau des communautés en regardant la consommation réelle totale des communautés mais sans comparer une donnée antérieure. Cela permet de détecter lorsqu'une communauté entière n'envoie plus de données ou alors que celles-ci sont incohérentes. En cas de problème, l'événement sera envoyé sur le topic alert.typealert dans le bus avec l'ID de communauté et le type d'alerte voulu.

Ce message serait ensuite reçu par Technician-notifier, qui serait une interface entre un technicien de Smartrix Grid, et envoyé à un technicien sous forme de notification.

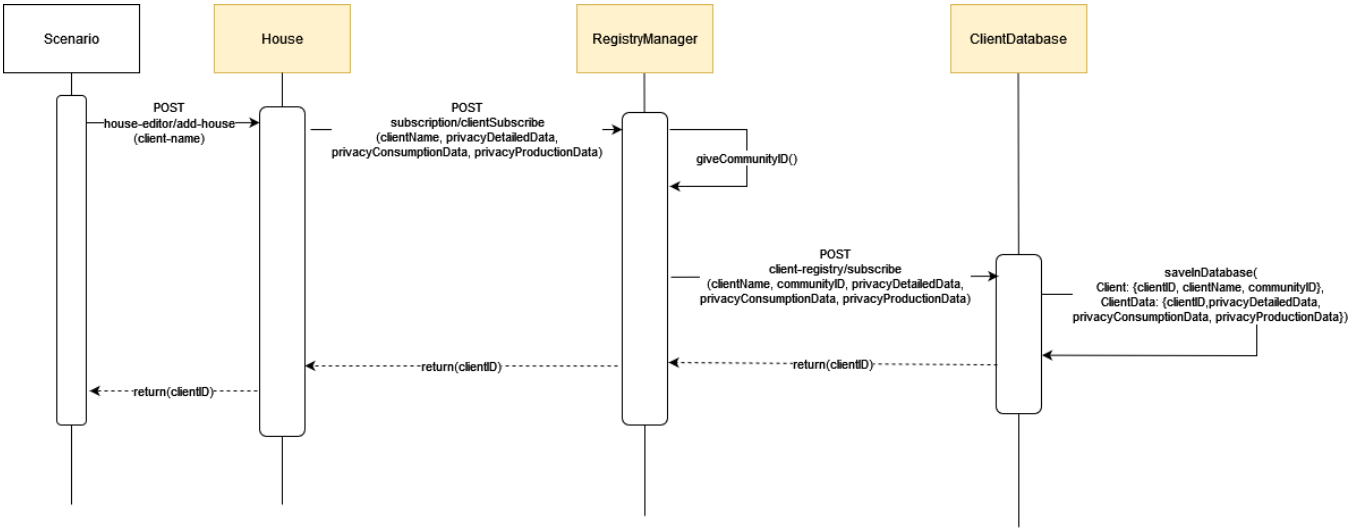
Implémentation de l'user story 17



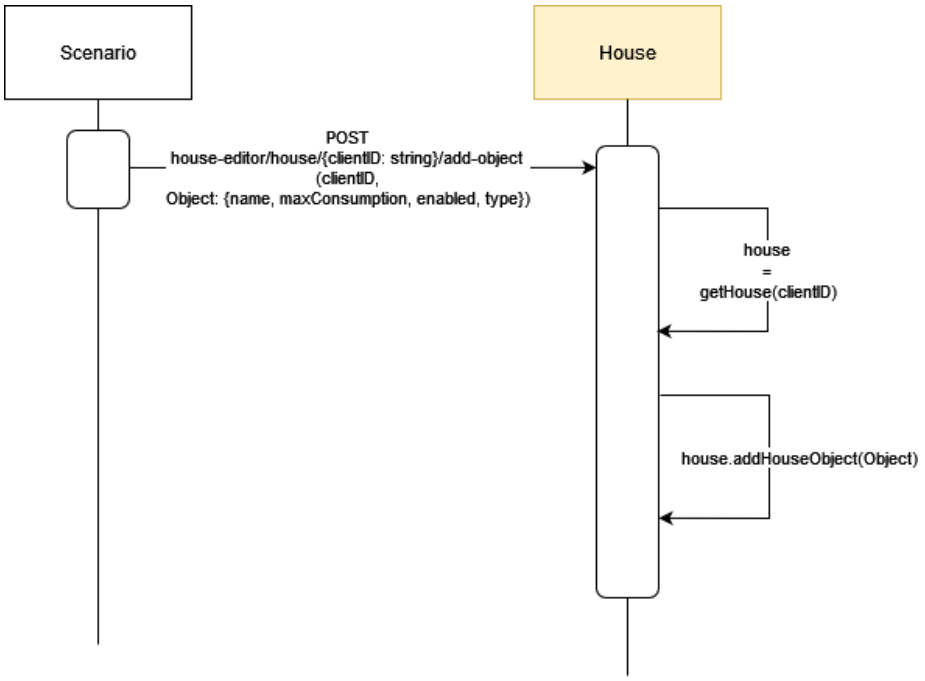
Pour permettre l'échange de VPP (la production en excès des fournisseurs et clients), nous avons pensé à la solution suivante. Si quelqu'un achète des VPP, la quantité achetée sera prélevée dans les batteries ou la production globale. Si quelqu'un vend des VPP, la quantité vendue sera stockée dans les batteries ou ajoutée à la production globale. Le VPP-Service permet de prélever et de charger les batteries selon les demandes de la VPP-API.

Annexe : Diagrammes de séquence

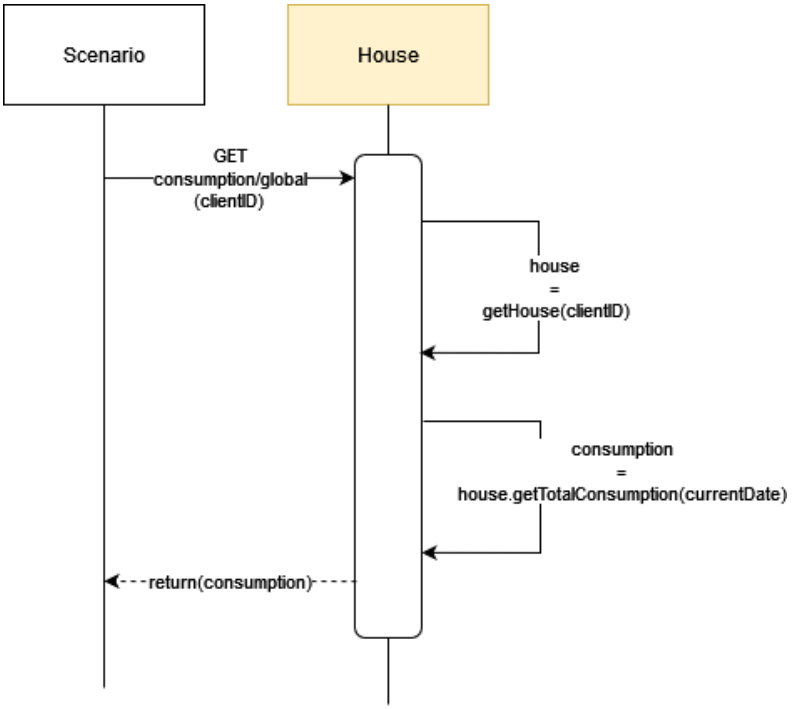
Ajout d'une maison



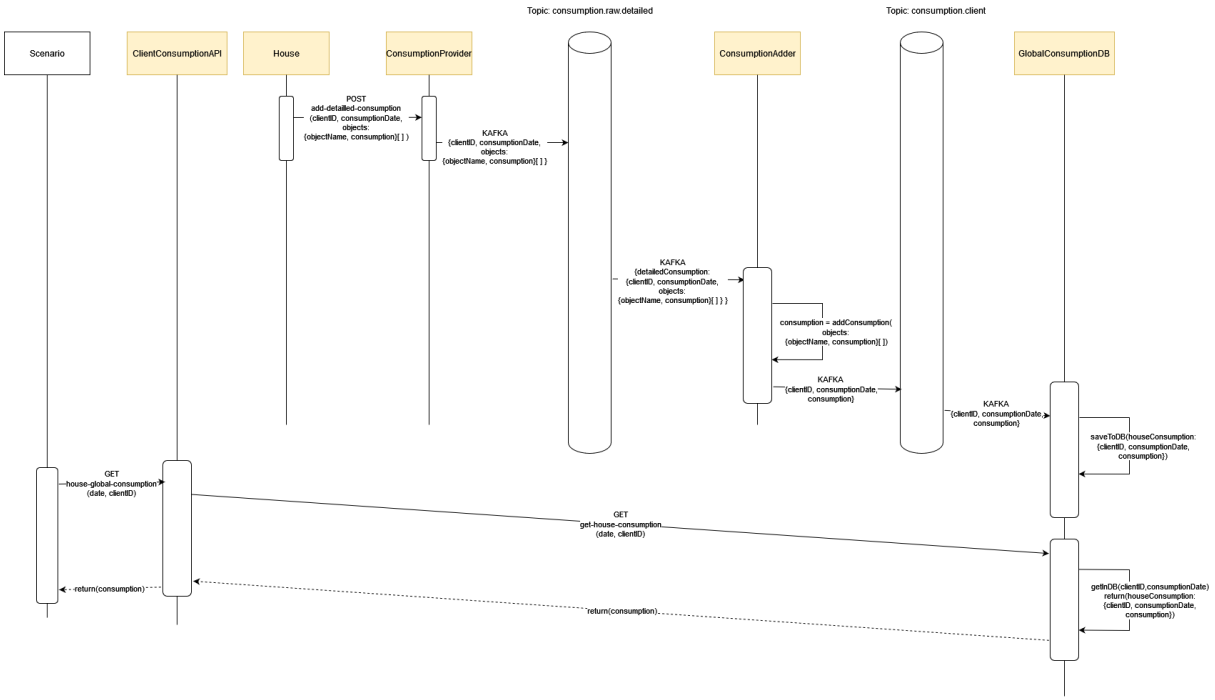
Ajout d'un objet



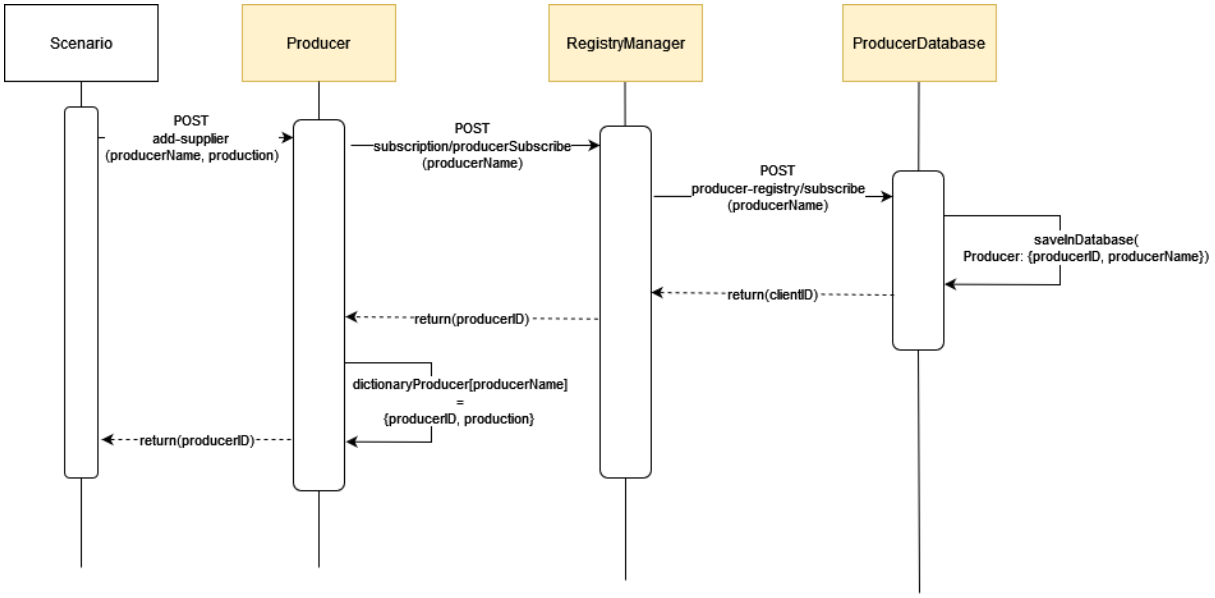
Consommation sur le boîtier d'une maison



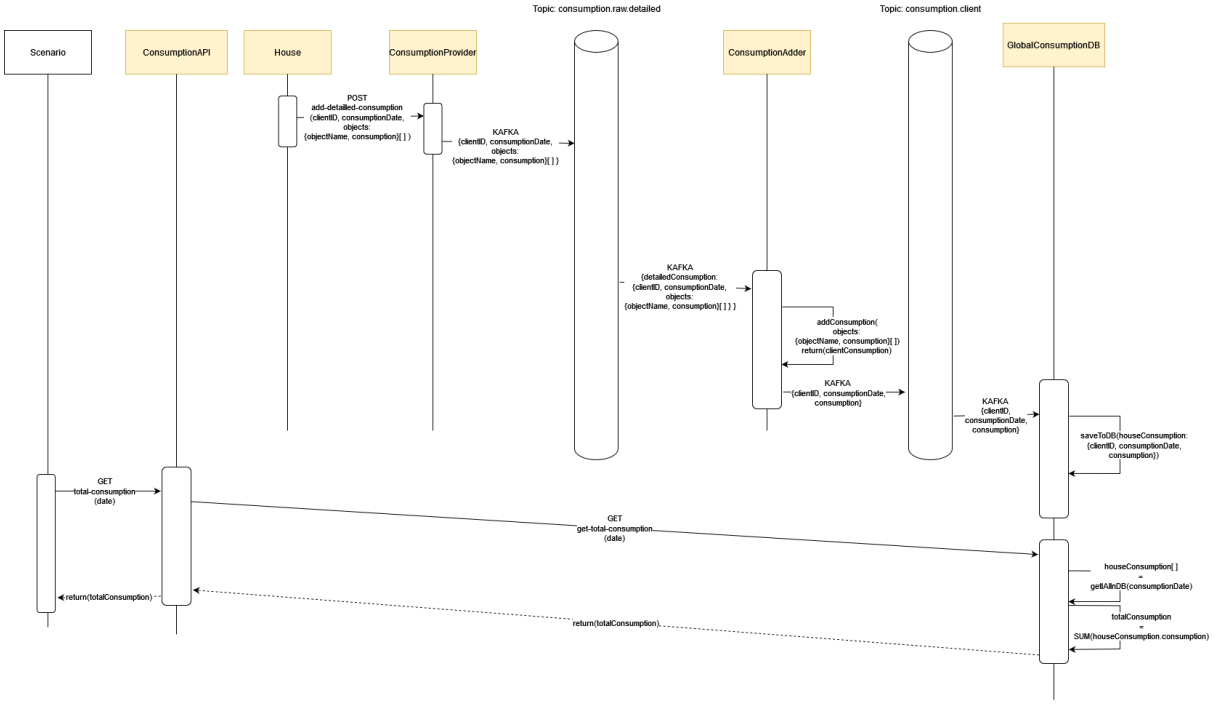
Consommation d'une maison depuis ClientConsumptionAPI



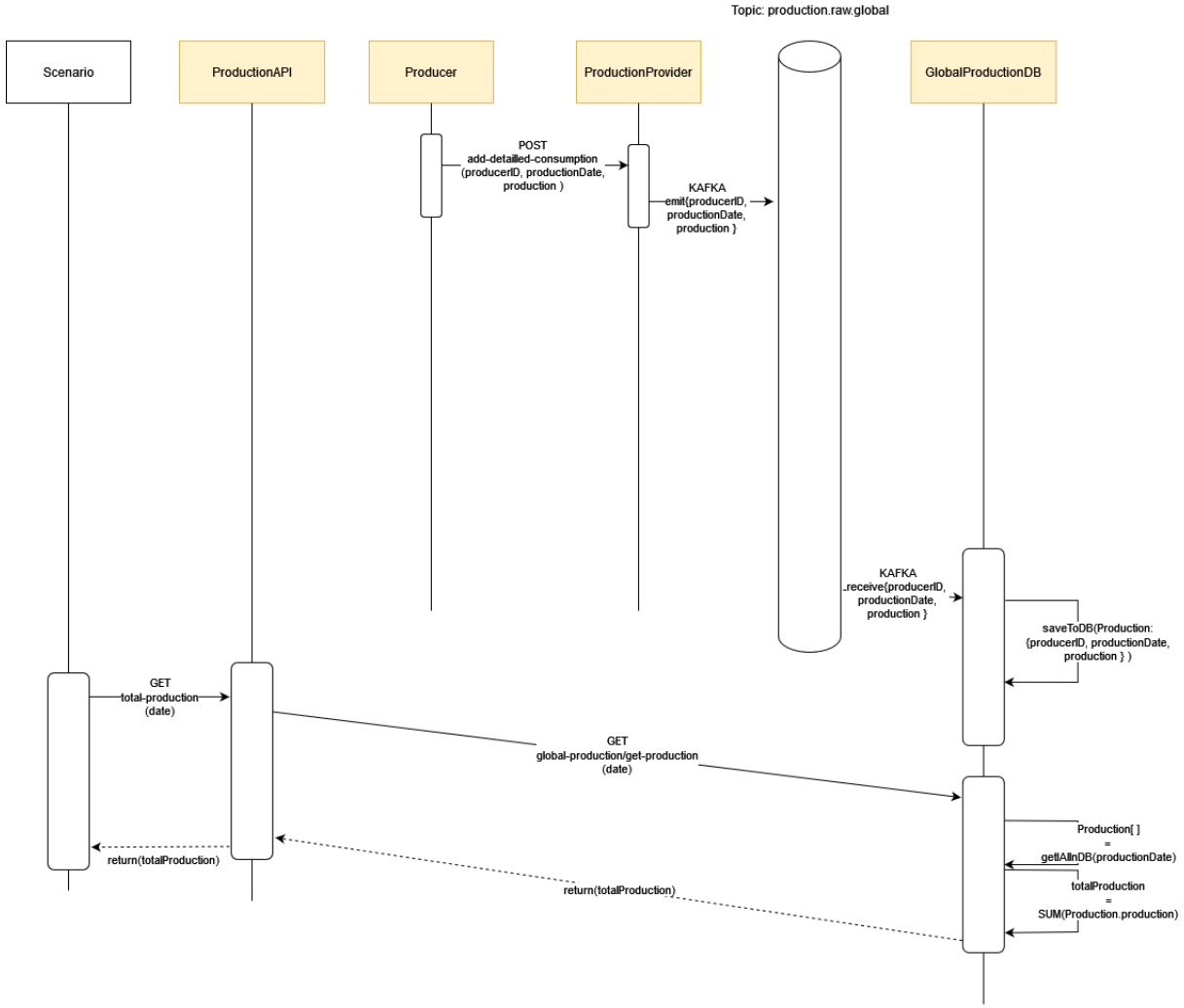
Ajout d'un producteur



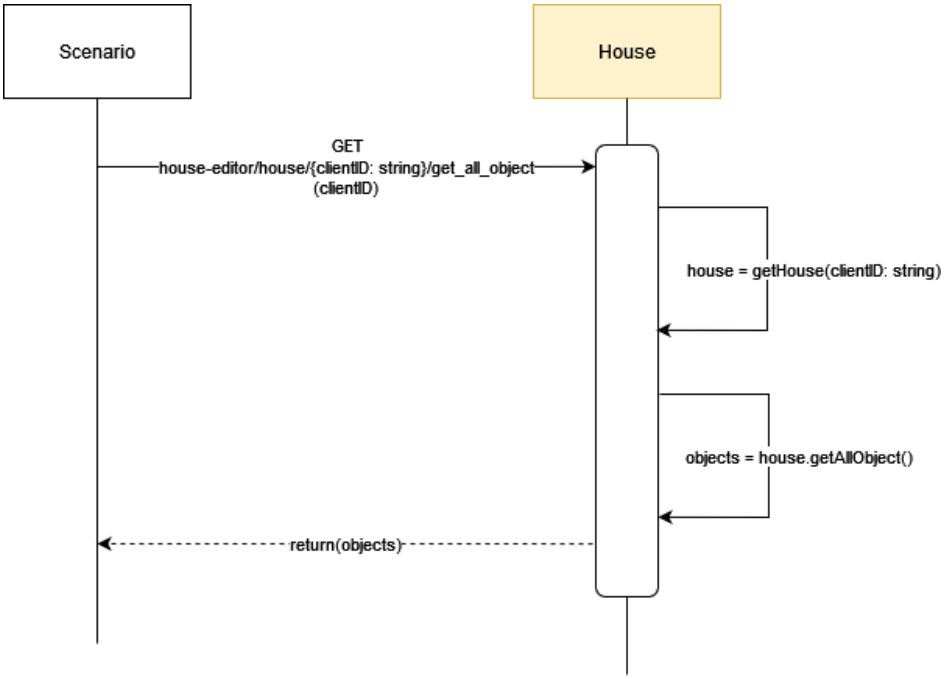
Consommation totale depuis consumptionAPI



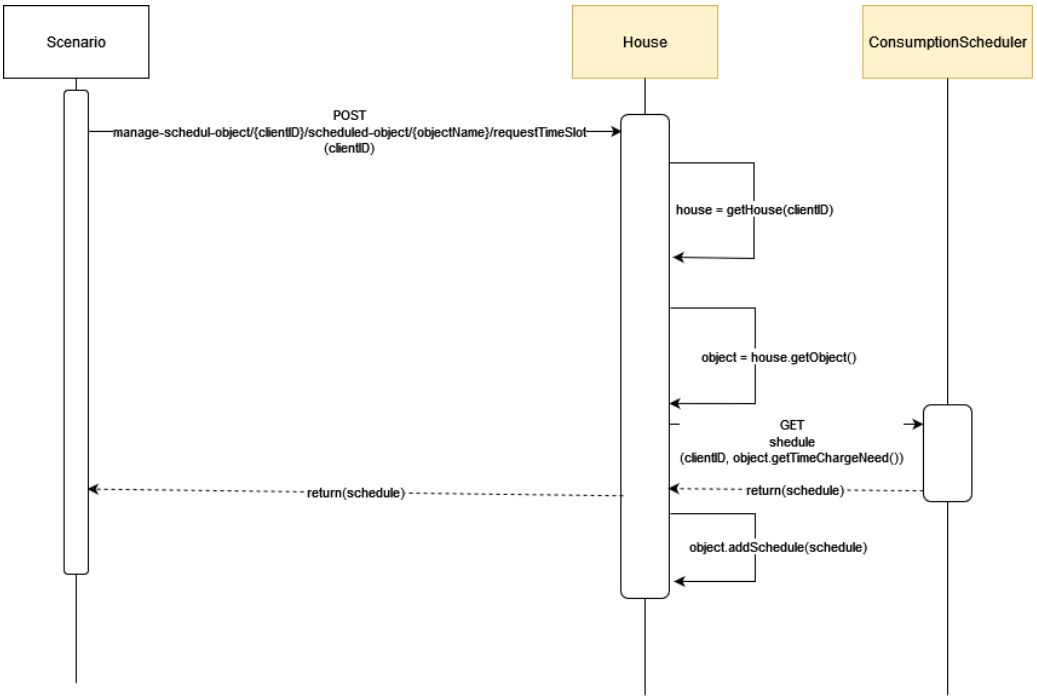
Production totale depuis productionAPI



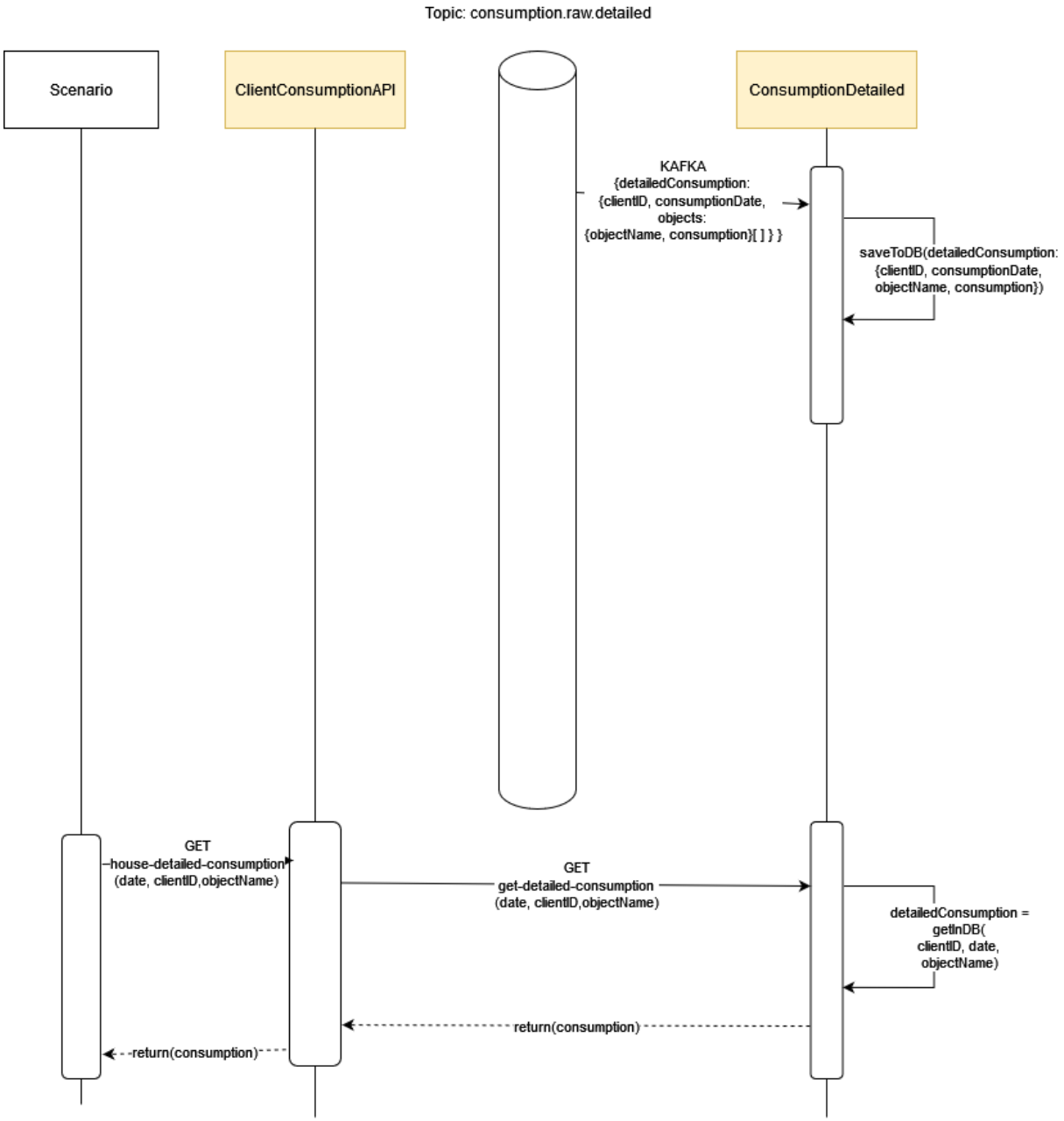
Récupérer tous les objets d'une maison



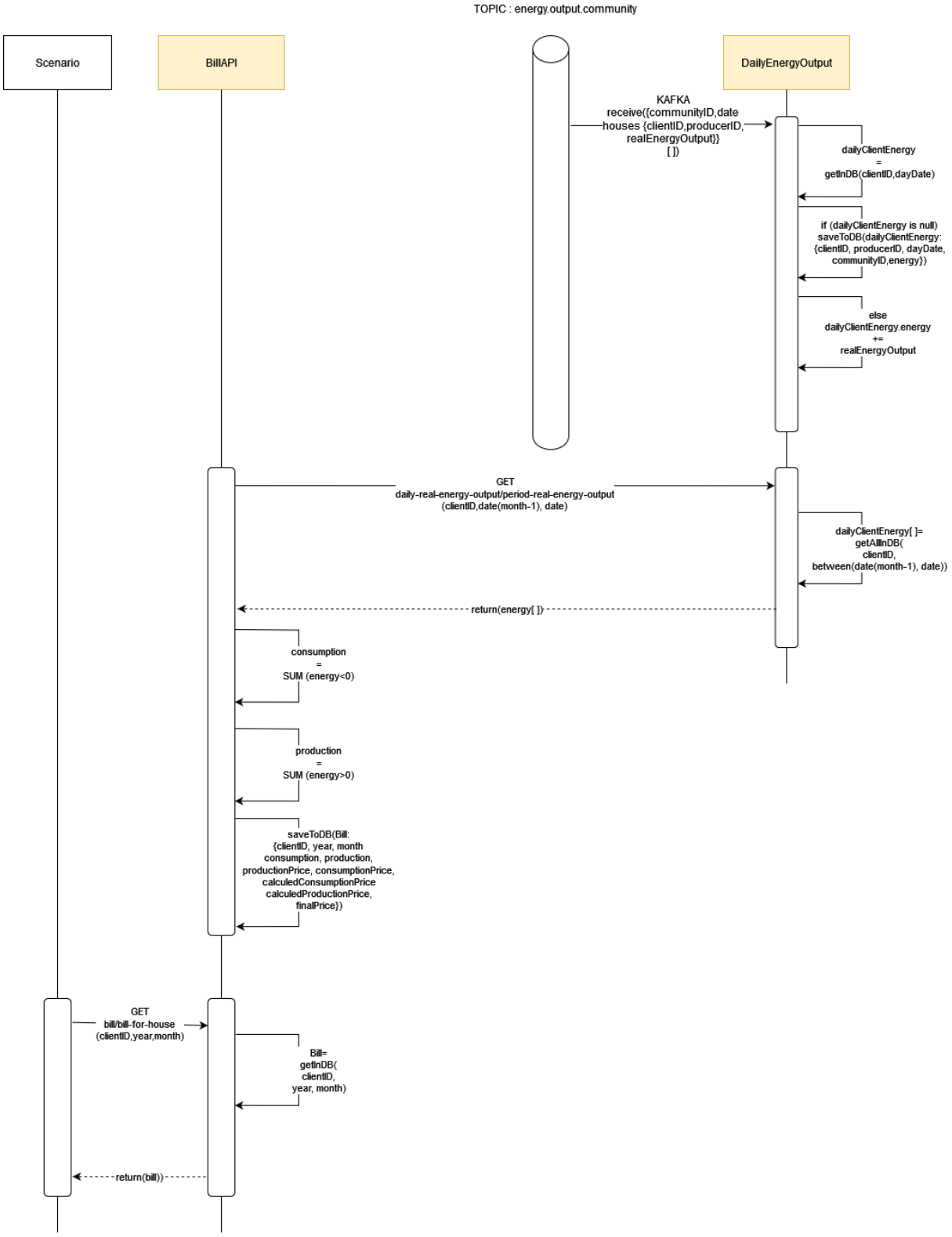
Récupérer le planning d'un objet



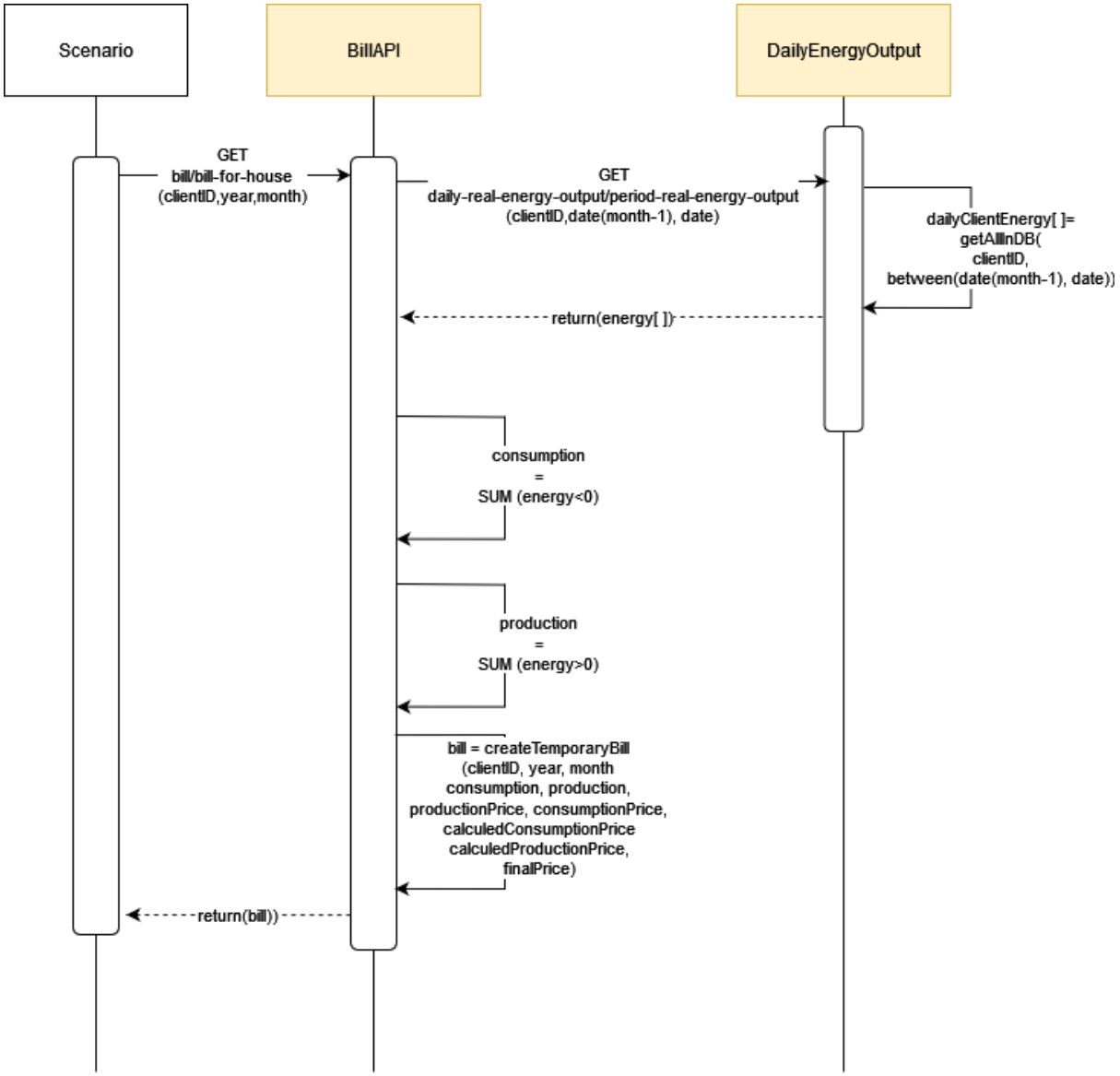
Récupérer la consommation détaillée d'un objet



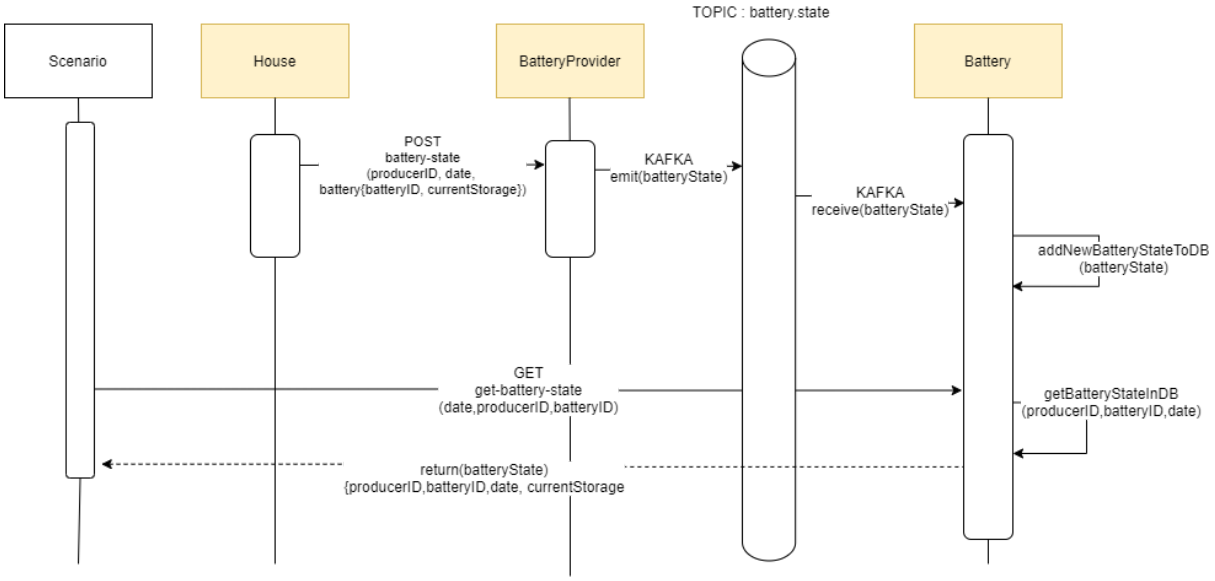
Récupérer la facture d'une maison pour un mois terminé



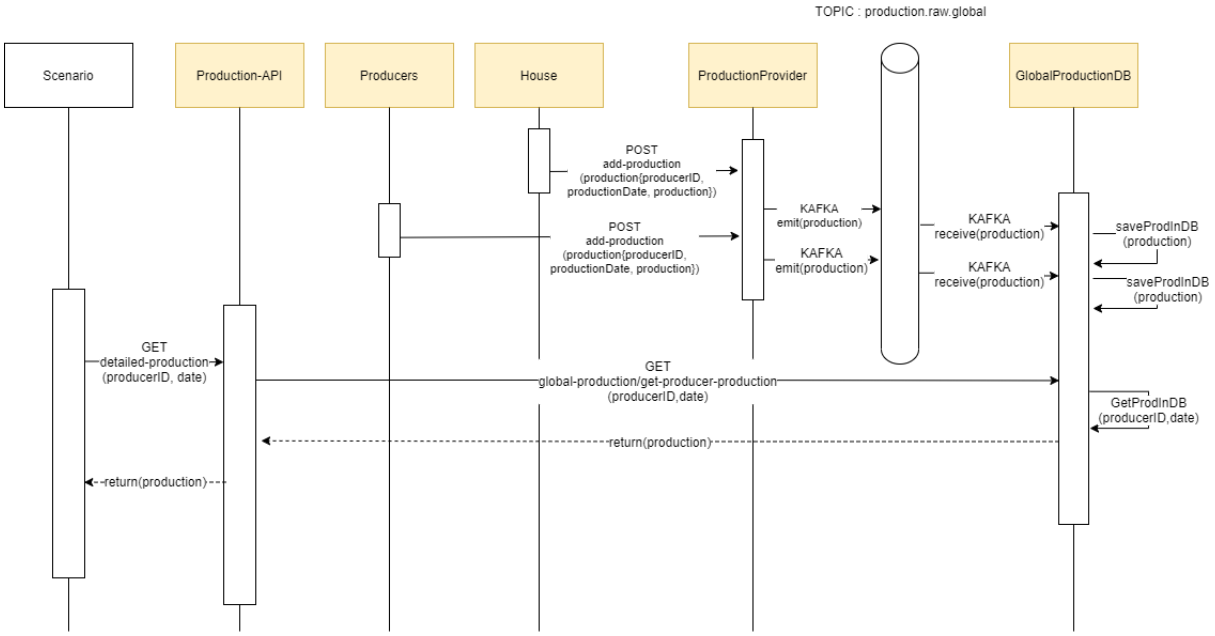
Récupérer la facture partielle d'une maison pour un mois en cours



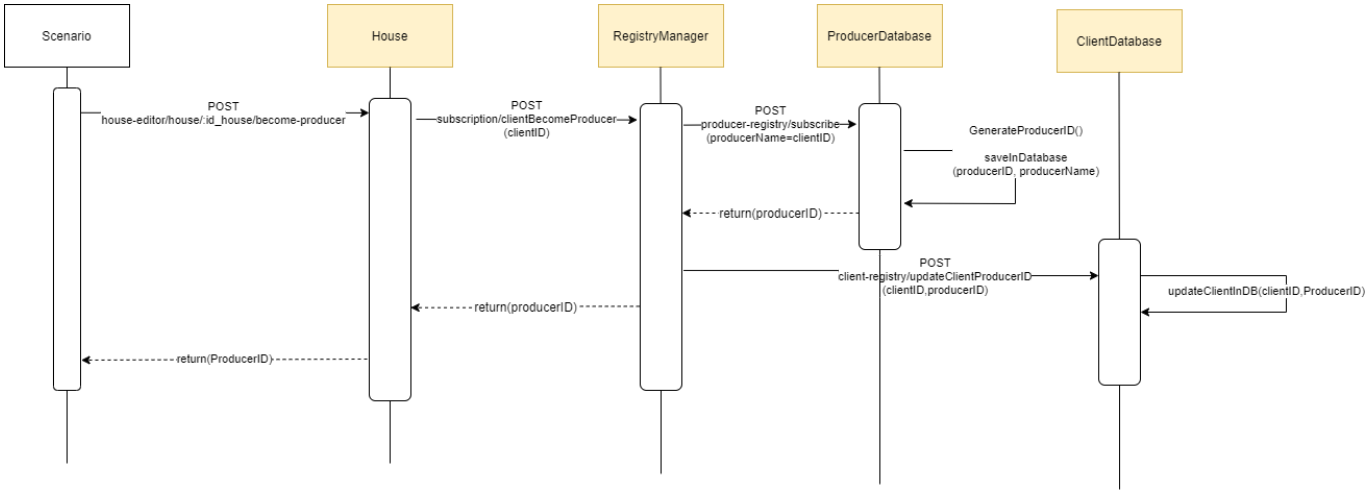
Récupérer l'état de la batterie



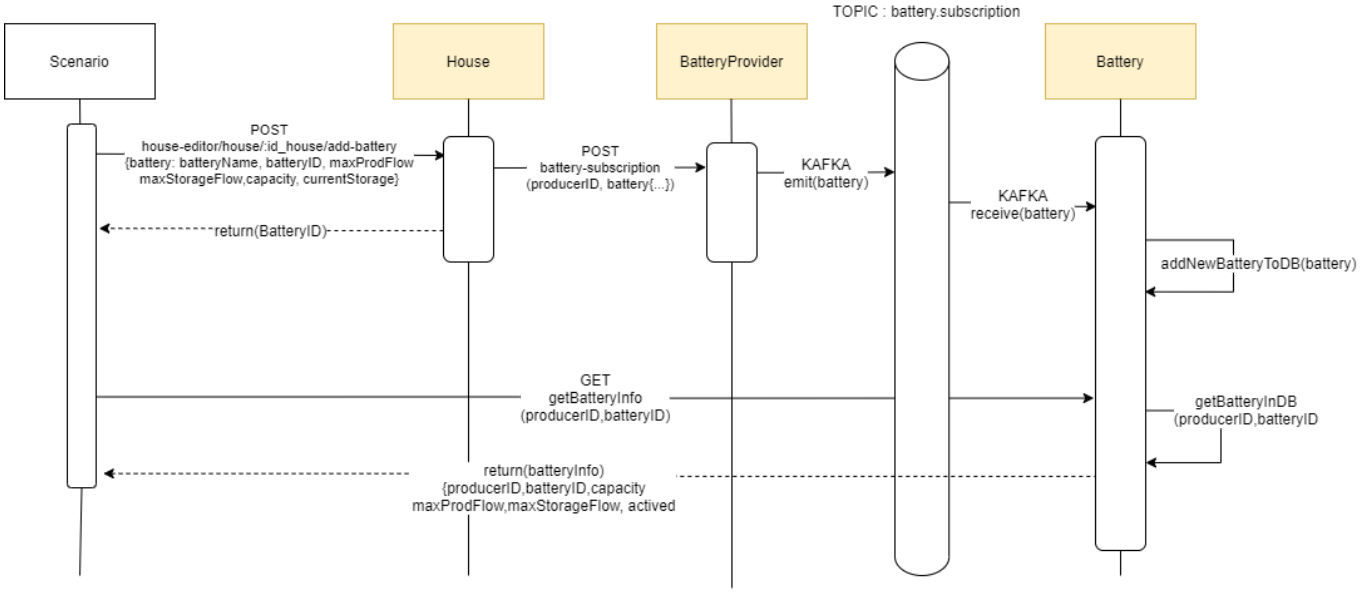
Obtenir la production détaillée d'un fournisseur



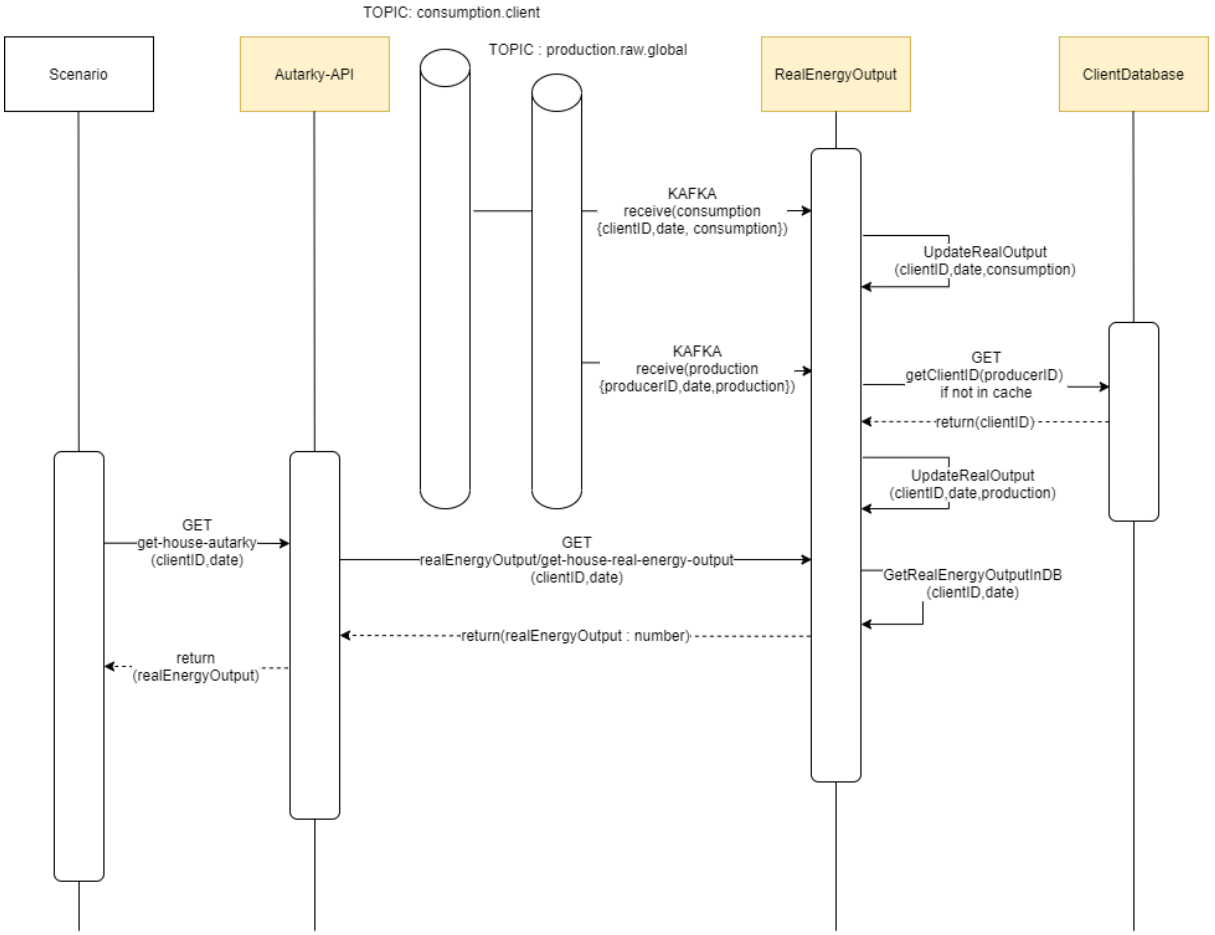
Devenir un fournisseur d'électricité local en tant que client



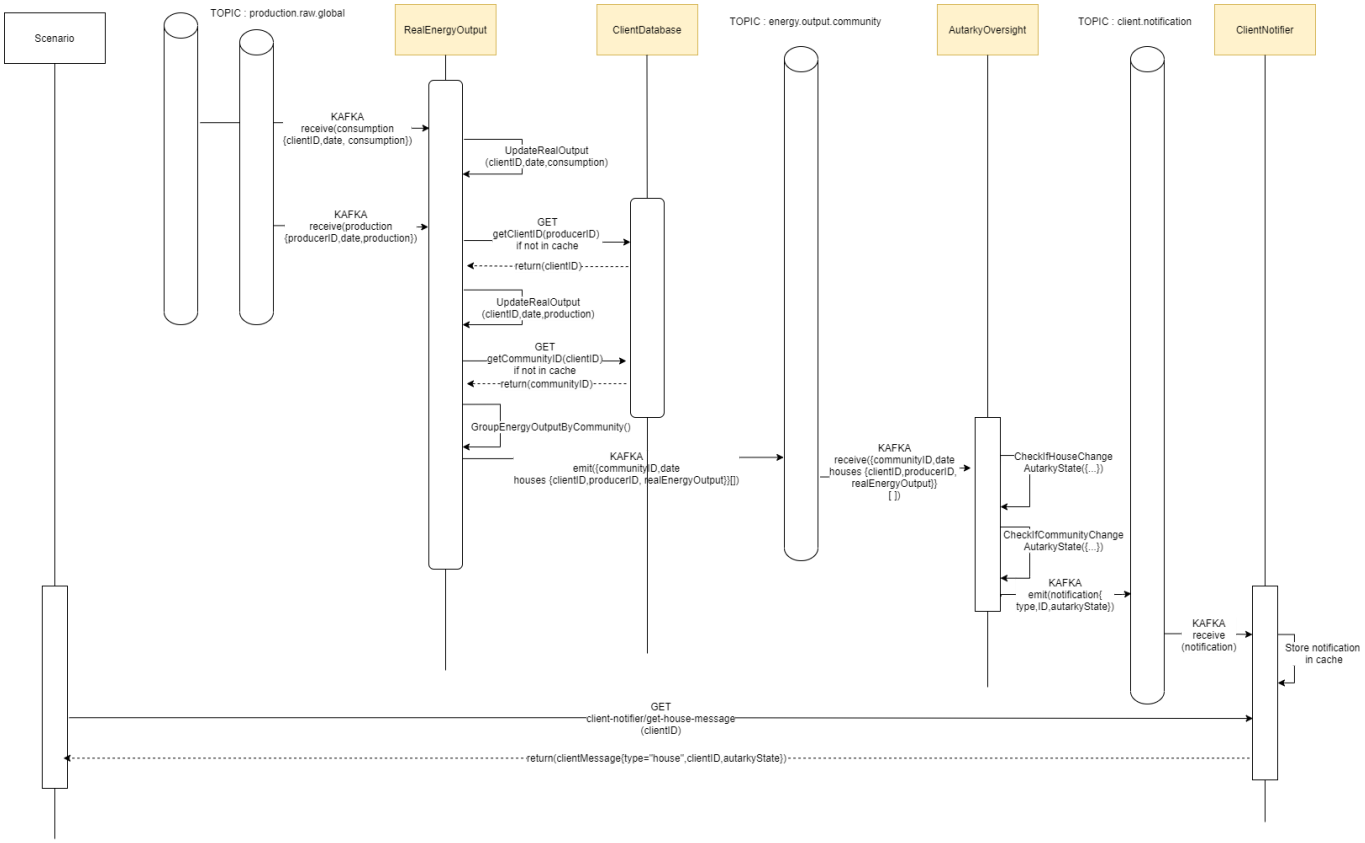
Ajout d'une batterie



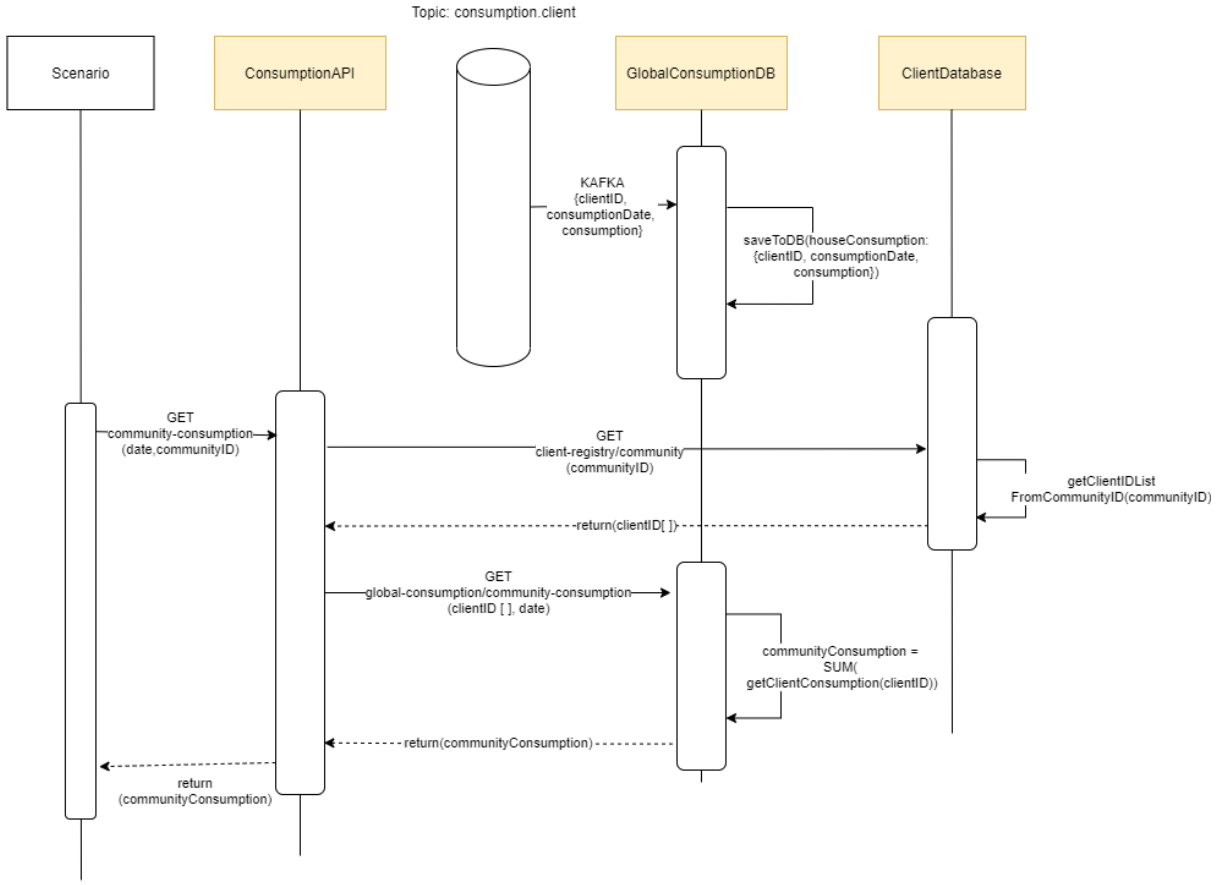
Obtenir l'énergie électrique d'un client



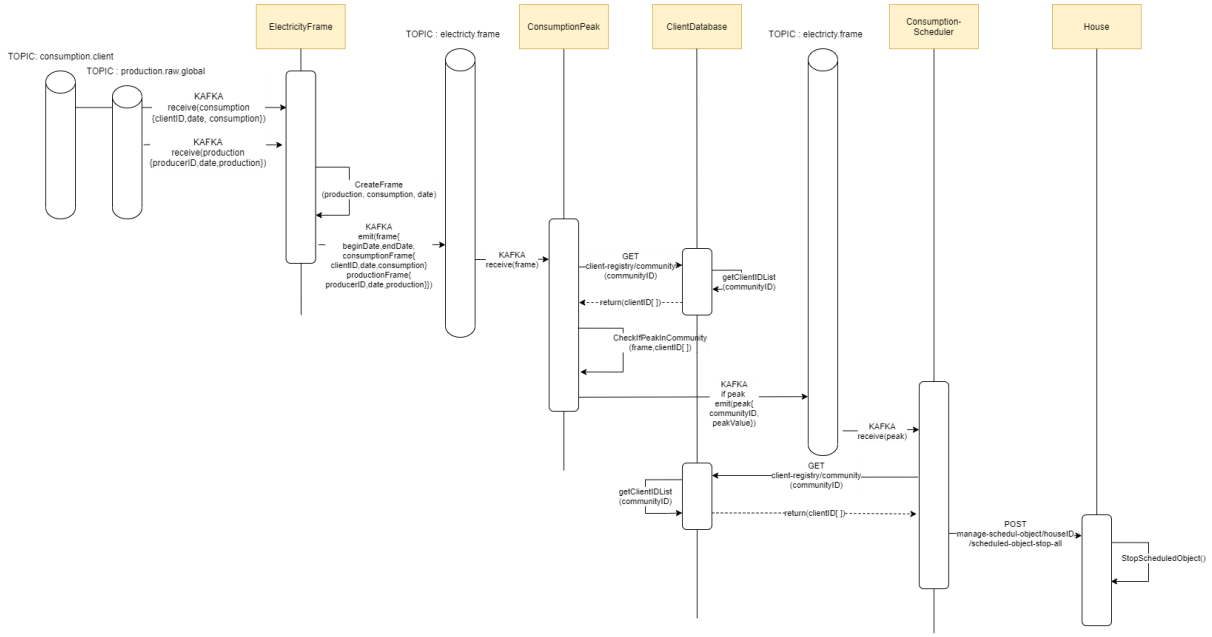
Obtenir la notification informant si le client est en auto-suffisance ou a perdu ce statut



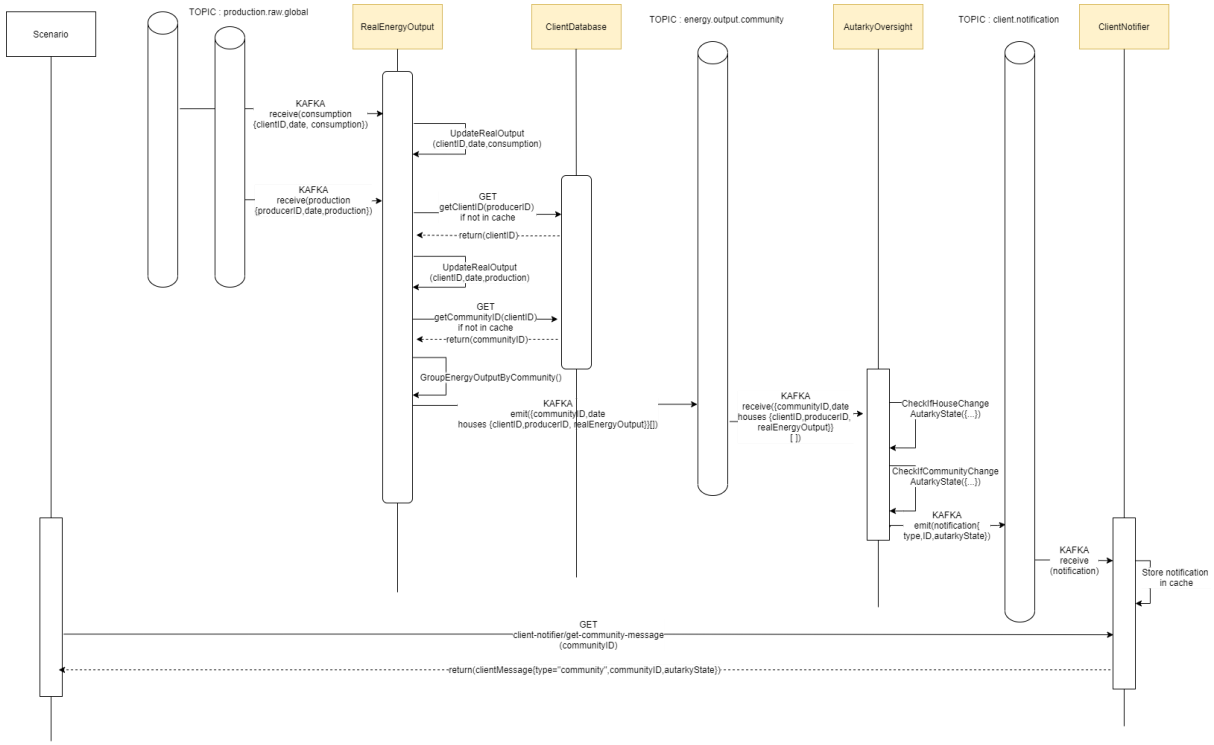
Récupérer la consommation d'une communauté de maisons



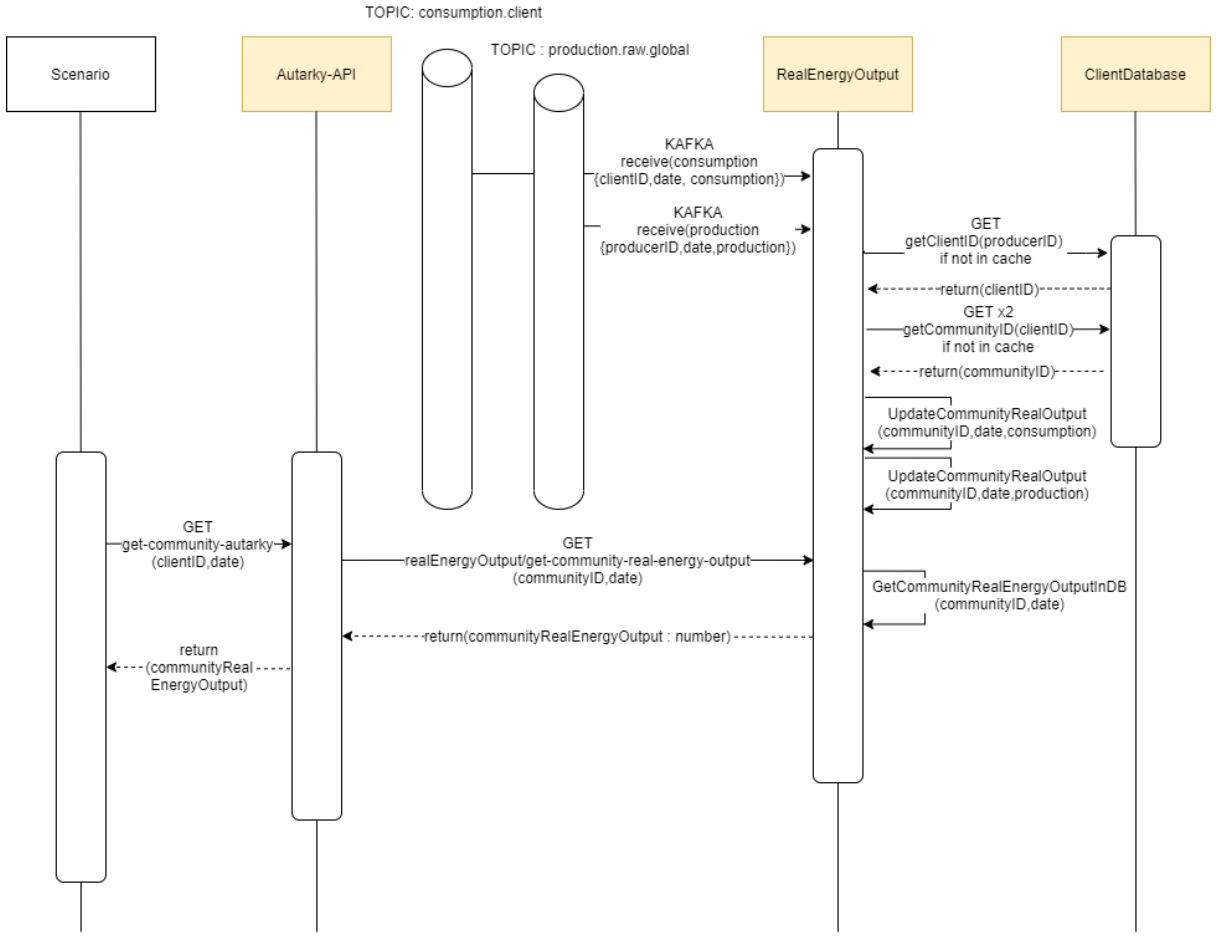
Identifier si un pic de consommation existe et réagir



Obtenir la notification informant si une communauté est auto-suffisante ou non



Obtenir l'énergie électrique totale d'une communauté



Fonctionnement de Partner API