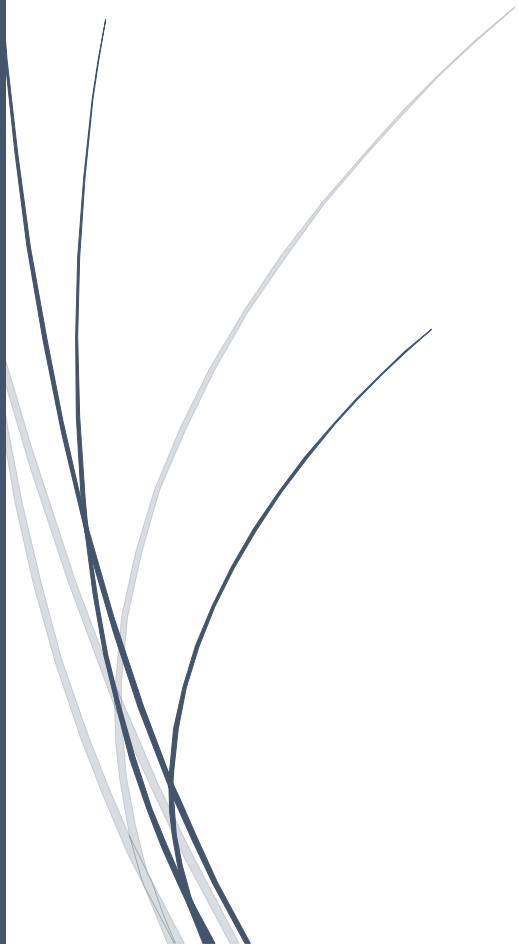
A dark blue vertical bar runs along the left edge of the page. A blue arrow points to the right from the bar, containing the text '2021-2022'.

2021-2022

# Rapport Projet Service oriented Architecture

Smatrix Grid

An abstract line drawing in the bottom left corner, consisting of several thin, curved lines in dark blue and light grey, resembling stylized reeds or grass.

Anas FRANCIS  
Jonas ANIGLO  
Soulaïman ZABOURDINE  
Patrick ANAGONOU

POLYTECH'NICE SOPHIA

## Table des matières

I.	Introduction .....	2
II.	Architecture du système .....	2
A.	Hypothèses .....	2
B.	Vue globale des services .....	3
C.	Les services .....	4
	Customer Care Service .....	4
	Local Production Monitor .....	6
	Energy Orchestrator Service .....	8
	Energy Monitor Service.....	10
	Reserve Energy Service .....	12
	Energy Supplier Service.....	13
	Accounting Service.....	14
	Bank Service .....	16
	Email Service .....	17
III.	Scénarios .....	18
IV.	Limites de l'architecture .....	36
V.	Bilan .....	37

## I. Introduction

Dans le cadre du cours Service Oriented Architecture (SOA), nous avons eu à implémenter une application appelée SmatrixGrid qui permet de fournir de l'énergie électrique. Ce projet consiste à gérer la distribution d'énergie à travers une grille, en suivant le même principe que les géants du marché tel que Amazon, Uber ou encore AirBnB.

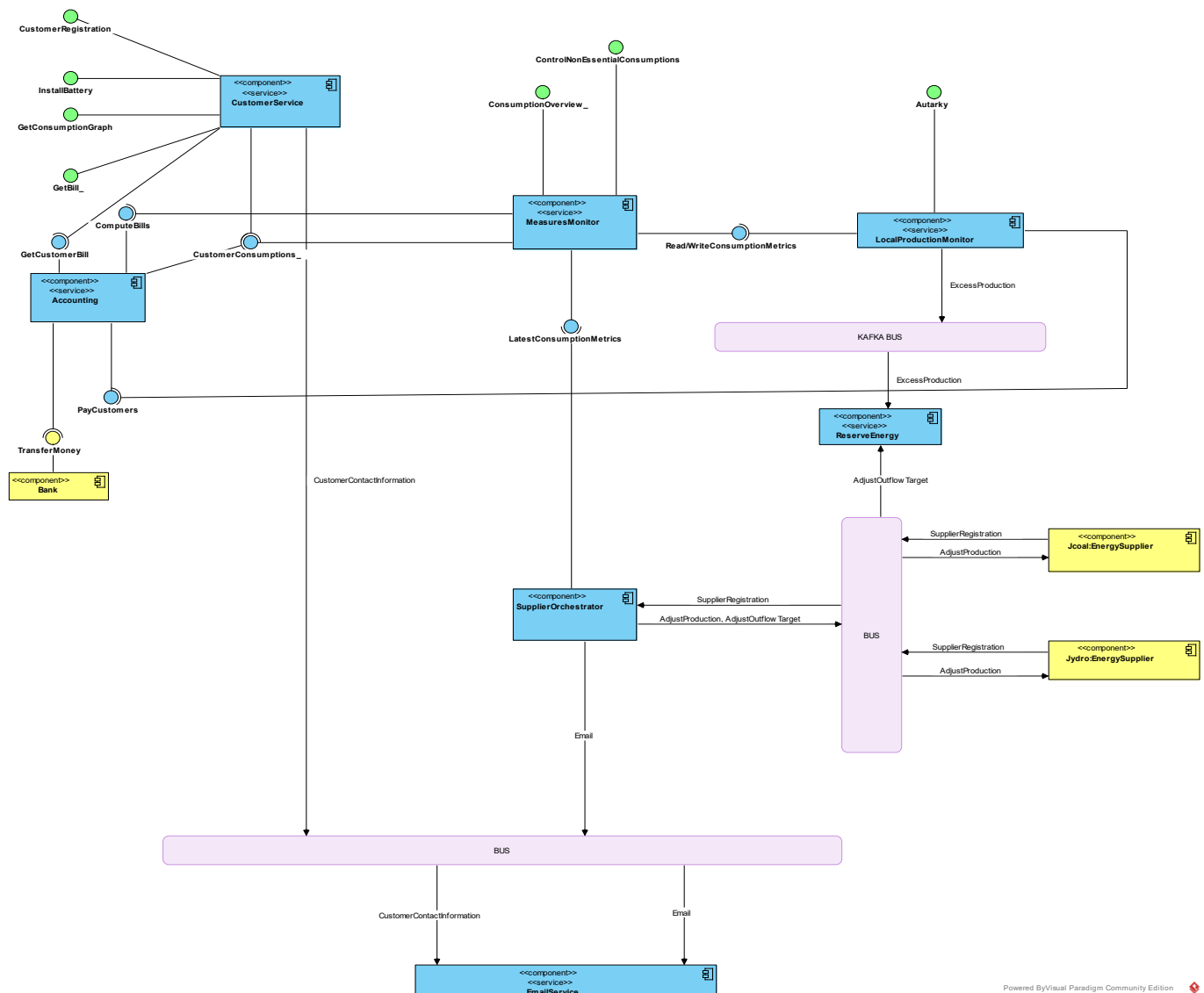
N'étant pas producteur, ces sociétés se concentrent principalement sur la mise en relation des fournisseurs avec les clients. SmatrixGrid se base sur le même principe, c'est à dire mettre directement en relation les fournisseurs d'énergies électriques avec les clients.

## II. Architecture du système

### A. Hypothèses

- Pour les fournisseurs d'énergie nous avons considéré qu'il s'agit de services externes à notre application. Ils doivent s'enregistrer pour être connus du système.
- Les clients peuvent prendre connaissances de leurs appareils qui consomment le plus d'énergie en se basant sur les consommations aperçues pendant les différentes périodes de la journée. Par exemple, un client ayant un pic de consommation le matin à l'heure du réveil peut estimer la production des appareils qu'il utilise à son réveil (machine à café, micro-onde etc.).
- Nikola est un employé de SmatrixGrid, mais son travail est en réalité automatisé et déclenché par l'horloge.
- L'entreprise d'Elon est un client de SmatrixGrid

## B. Vue globale des services



Légende :

Les rectangles **bleus** représentent nos services

En **vert** nous avons les interfaces qu'on expose aux clients et aux employés / administration de Smatrix Grid

En **jaune** nous avons les services et interfaces des partenaires de Smatrix Grid

En **rose** nous avons les bus de message Kafka ou RabbitMQ

Les ronds **bleus** qui connectent les services représentent des routes http POST ou GET permettant à nos services de communiquer.

## C. Les services

### Customer Care Service

Ce service sert de point d'entrées aux clients de SmatrixGrid car il permet de stocker toutes les informations contractuelles de nos clients lors de leurs inscriptions. Il permet également de communiquer avec d'autres services essentiels à notre application comme le service de comptabilité 'AccountingService' ou le service de courriel 'EmailService'.

Vous retrouverez ci-dessous des diagrammes permettant de comprendre les données manipulées par le service ainsi que les besoins auxquels ce service répond.

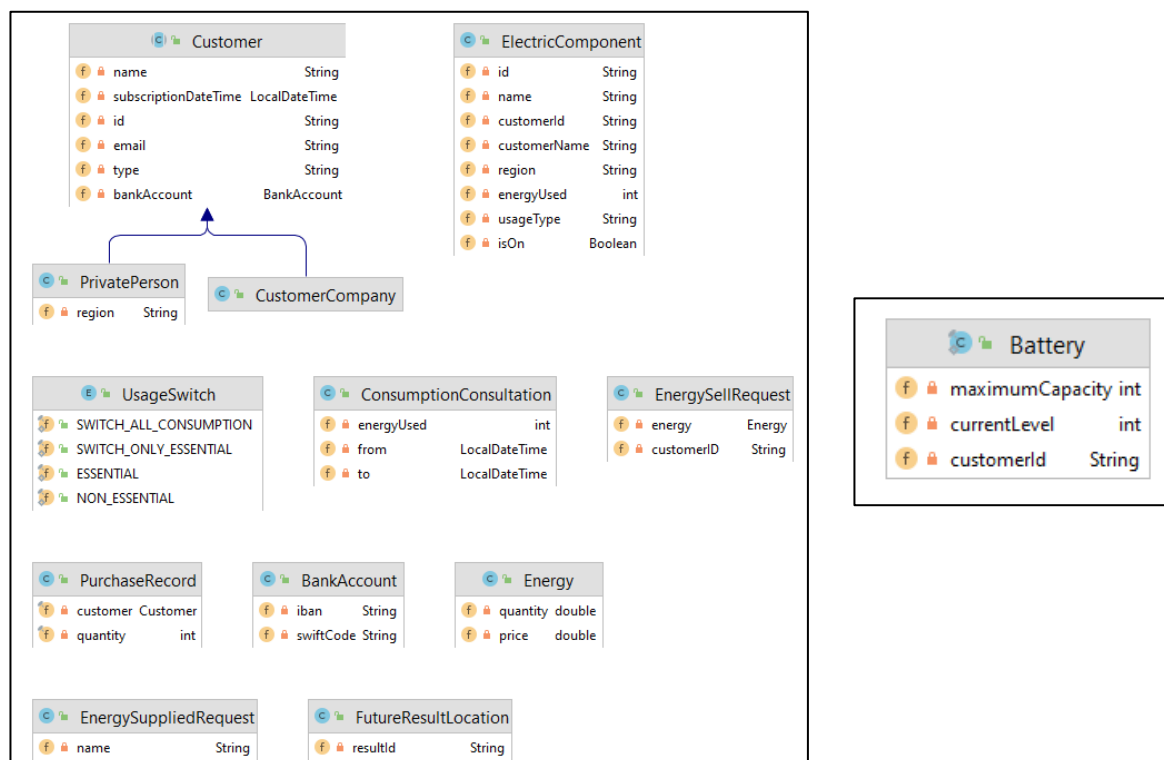
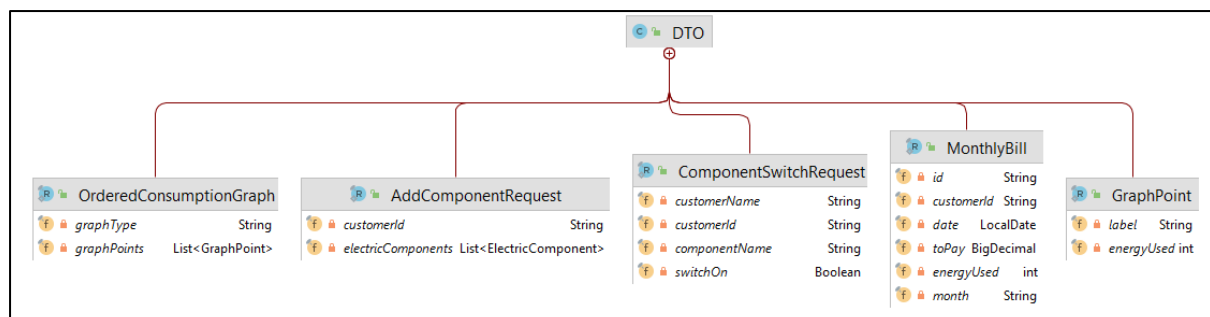
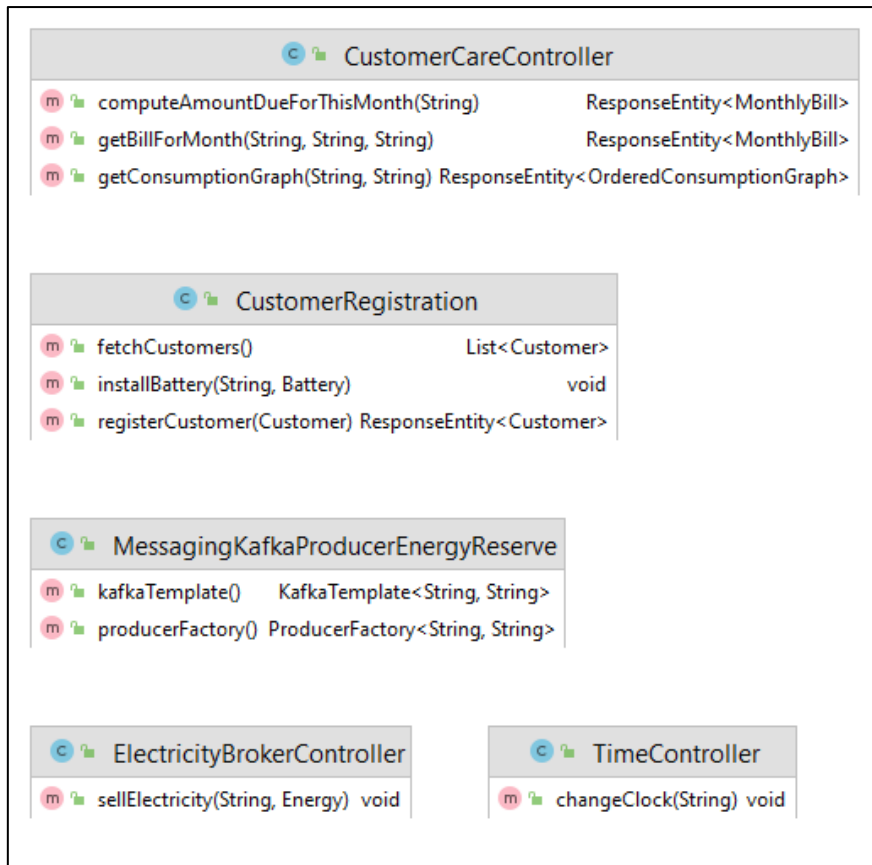


Diagramme de classe



Objets de transfert de données



*Interfaces REST/Kafka*

## Local Production Monitor

Le LocalProductionMonitor, va server comme un intermédiaire entre la consommation / production du client et le measureMonitorService.

Ce service va transférer au measureMonitorService les mesures de consommation du client (peut-être positive ou null) et il va enregistrer dans la batterie du client sa production.

Les clients envoient donc la production et la consommation au LocalProductionMonitor.

Plus tard quand il y aura une nouvelle demande de consommation, les clients envoient la consommation et la production au LocalProductionMonitor. On déduit de la batterie (si la batterie contient de l'énergie) la quantité que le client veut consommer. Et enregistrer la production dans sa batterie et/ou envoyer l'excès de production au service reserveenergy.

Si la demande de consommation est supérieure à la quantité produite par le client et ce qu'il a dans la batterie, on demande à smartrix grid de fournir l'énergie.

Ce service va nous permettre de savoir si un client (et donc une région) est en autarky ou pas.

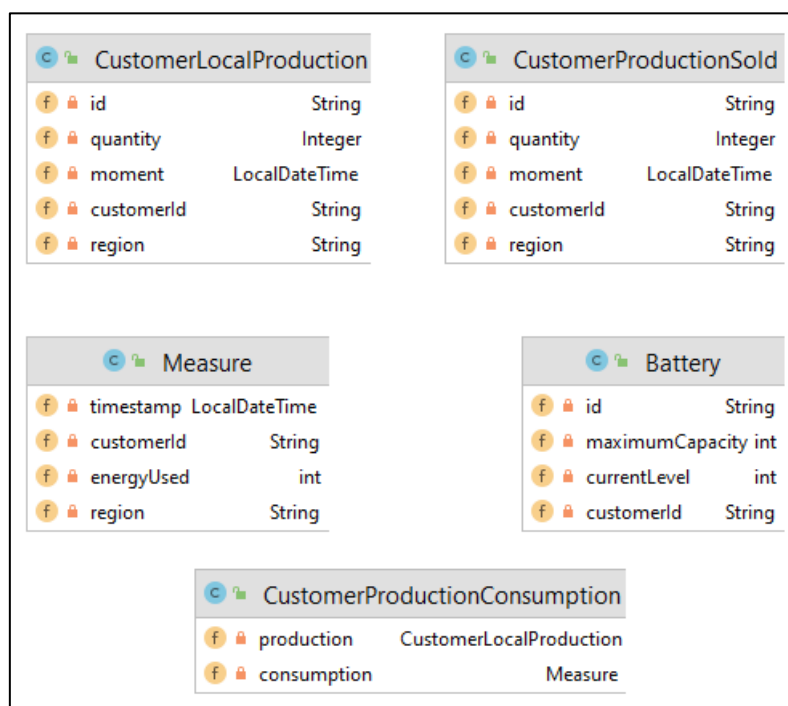
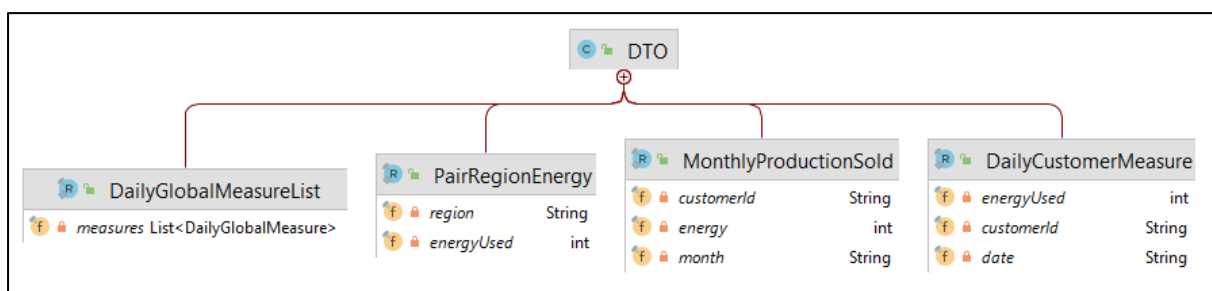
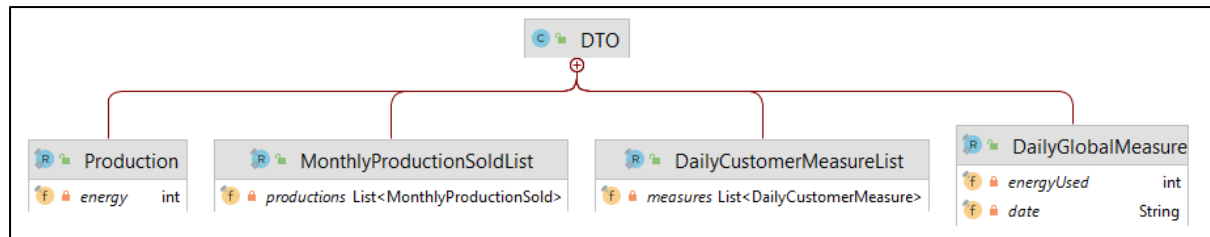
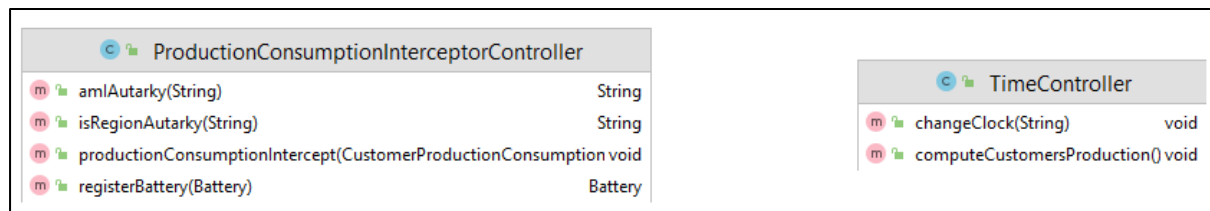


Diagramme de classe

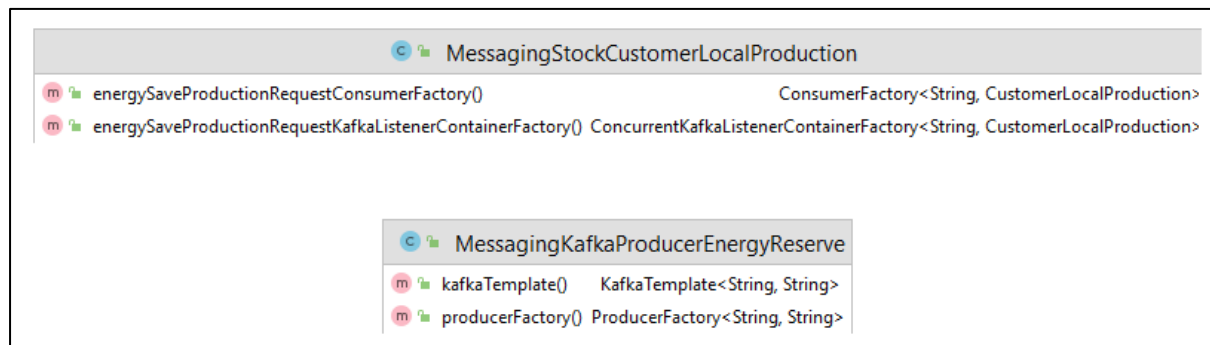




*Objets de transfert de données*



*Interfaces REST*



*Interfaces Kafka*



## Energy Orchestrator Service

Ce service un rôle d'orchestrateur à la gestion de l'énergie de SmatrixGrid. Il permet notamment de consulter la charge globale au sein de la grille et de réguler la charge à laquelle est soumise la grille. De plus, il permet notamment de faire le lien avec les différents fournisseurs de l'entreprise en offrant la possibilité aux personnes concerner de faire des demandes d'ajustements de production.

Vous retrouverez ci-dessous des diagrammes permettant de comprendre les données manipulées par le service ainsi que les besoins auxquels ce service répond.

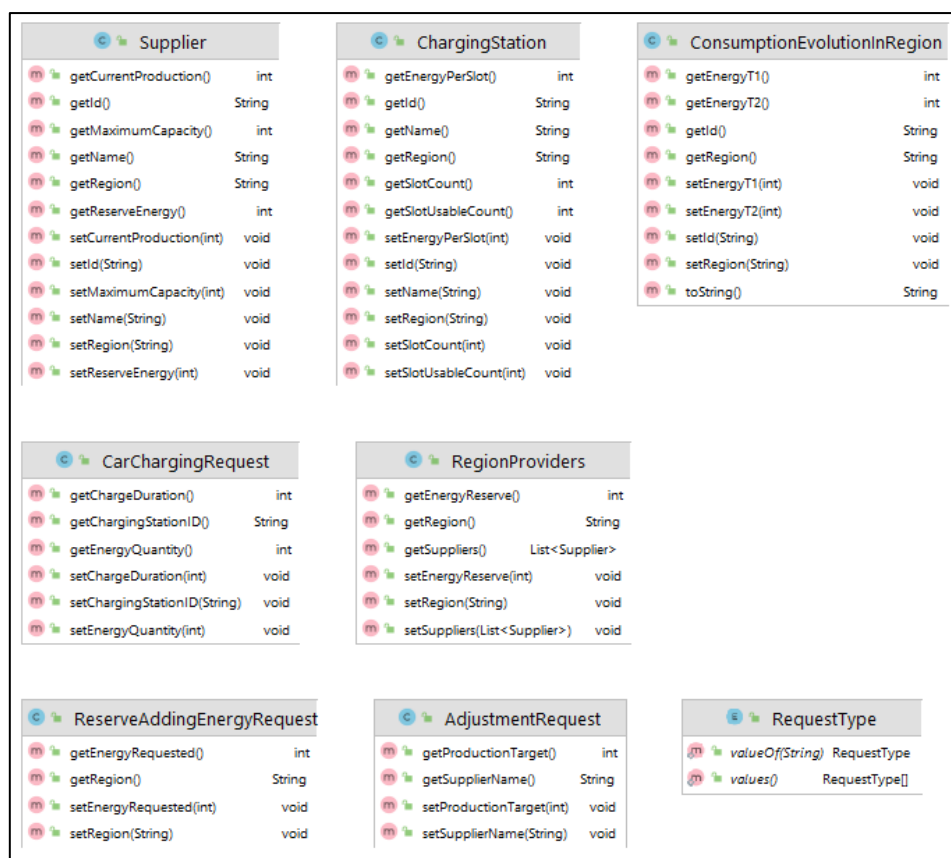
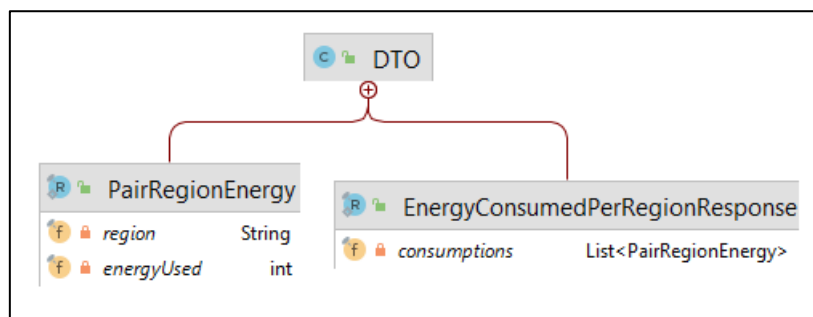
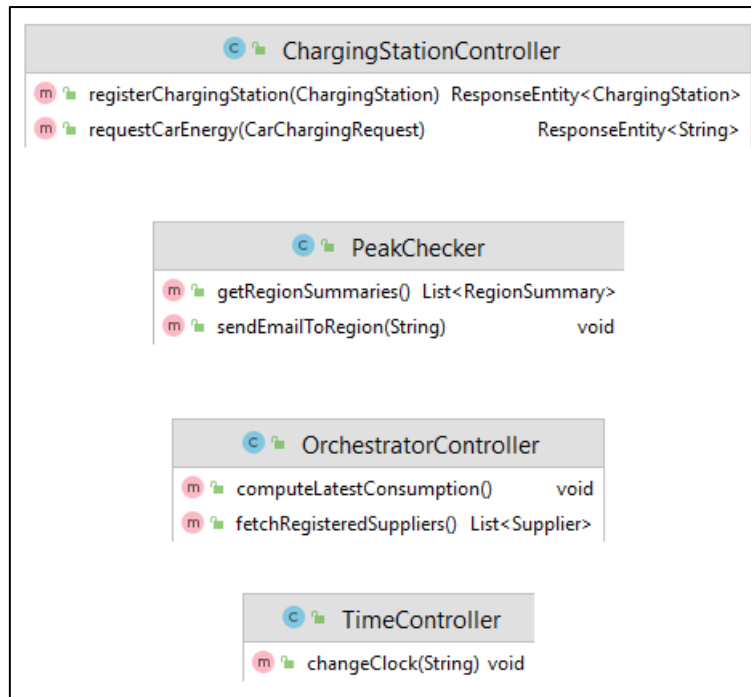


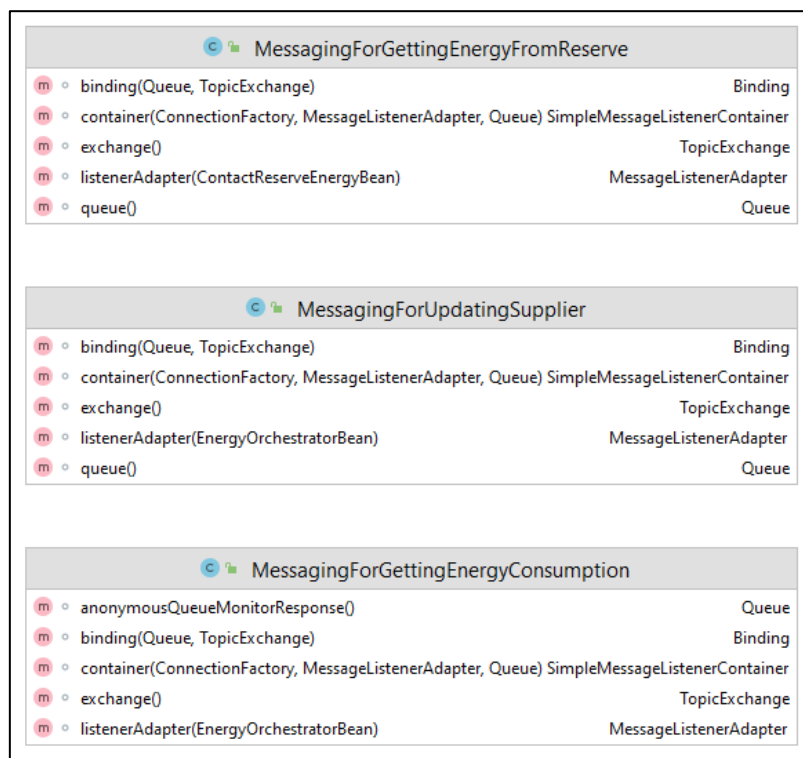
Diagramme de classe



Objets de transfert de données



*Interfaces REST*



*Interfaces RabbitMQ*

## Energy Monitor Service

Ce service permet de manipuler toutes les transactions liées aux consommations des clients. De ce fait, c'est ici que seront répertoriés tous les composants électriques de nos clients. Ce service est également utilisé par d'autres personas de l'application notamment par Charles, le CEO de l'entreprise. En effet ce service permettra de réguler toutes les consommations de toutes les grilles de l'application.

C'est notamment le service qui est soumis à la charge la plus importante. Vous retrouverez ci-dessous des diagrammes permettant de comprendre les données manipulées par le service ainsi que les besoins auxquels ce service répond.

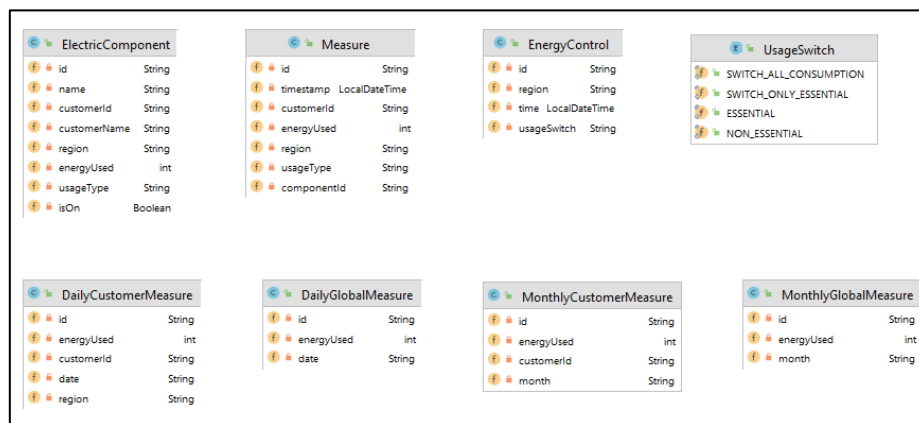
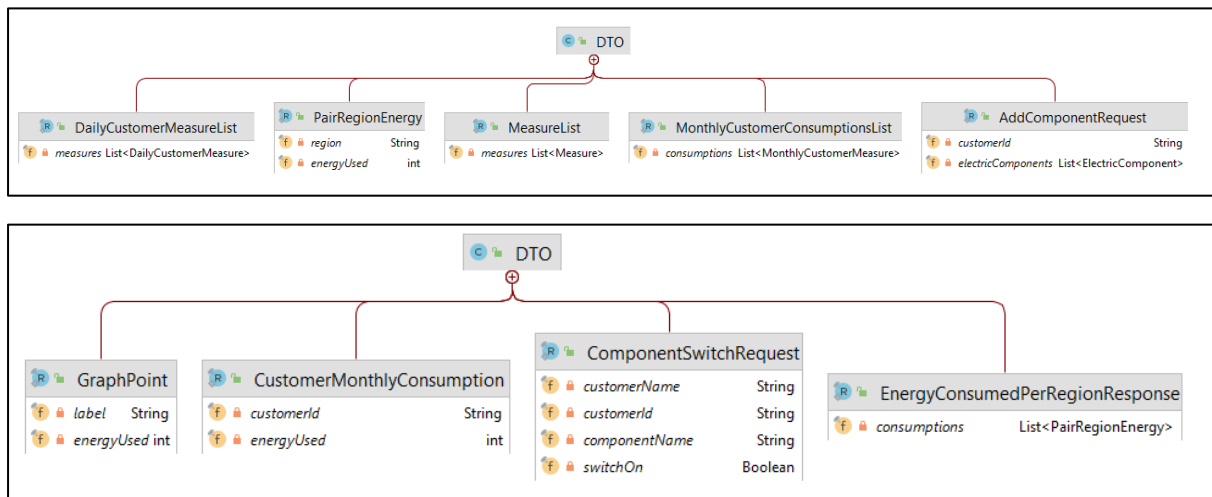


Diagramme de classe



Objets de transfert de données

<b>ElectricMeterController</b> <ul style="list-style-type: none"> <li>addCustomerComponent(AddComponentRequest) ResponseEntity&lt;AddComponentRequest&gt;</li> <li>customerSendMeasures(String) void</li> <li>getLatestConsumptionsOfRegions() EnergyConsumedPerRegionResponse</li> <li>getMeasuresByCustomerId(String, String, String) ResponseEntity&lt;DailyCustomerMeasureList&gt;</li> <li>getMeasuresByRegion(String, String, String) ResponseEntity&lt;PairRegionEnergy&gt;</li> <li>receiveMeasure(Measure) ResponseEntity&lt;Measure&gt;</li> <li>switchCustomerComponent(ComponentSwitchRequest) ResponseEntity&lt;ComponentSwitchRequest&gt;</li> <li>viewMeasures() void</li> </ul>	<b>CEOController</b> <ul style="list-style-type: none"> <li>overviewConsumption(String) ResponseEntity&lt;OrderedConsumptionGraph&gt;</li> <li>regionTotalConsumption(String) int</li> <li>switchUsageType(EnergyControl) ResponseEntity&lt;EnergyControl&gt;</li> <li>viewSwitchUsage() void</li> </ul>
<b>AccountingInterfaceController</b> <ul style="list-style-type: none"> <li>computeMonthlyConsumption(String) CustomersMonthlyConsumptionsList</li> <li>computeThisMonthConsumptionForCustomer(String) CustomerMonthlyConsumption</li> </ul>	<b>MeasureAggregatorController</b> <ul style="list-style-type: none"> <li>dailyAggregate() void</li> <li>monthlyAggregate() void</li> </ul>
<b>CustomerInterface</b> <ul style="list-style-type: none"> <li>computeConsumptionForConsumer(String, String) ResponseEntity&lt;OrderedConsumptionGraph&gt;</li> </ul>	<b>TimeController</b> <ul style="list-style-type: none"> <li>changeClock(String) void</li> </ul>

## Interfaces REST

<b>MessagingForCustomerConsumption</b> <ul style="list-style-type: none"> <li>binding(Queue, TopicExchange) Binding</li> <li>container(ConnectionFactory, MessageListenerAdapter) SimpleMessageListenerContainer</li> <li>exchange() TopicExchange</li> <li>listenerAdapter(CustomerInterface) MessageListenerAdapter</li> <li>queue() Queue</li> </ul>
<b>MessagingStockCustomerConsumption</b> <ul style="list-style-type: none"> <li>consumerFactory(String) ConsumerFactory&lt;String, String&gt;</li> <li>energyConsumptionKafkaListenerContainerFactory() ConcurrentKafkaListenerContainerFactory&lt;String, Measure&gt;</li> <li>energyConsumptionRequestConsumerFactory() ConsumerFactory&lt;String, Measure&gt;</li> </ul>

## Interfaces RabbitMQ / Kafka

## Reserve Energy Service

Ce service représente le stockage d'énergie que les client vont produire (notamment si un client possède un panneau solaire et a un excès d'énergie).

Ce service sera appelé par le EnergyOrchestratorSupplier, si les fournisseurs ont déjà atteints leur capacité de production maximale.

Il peut-être utiliser par Charles (user story 18) pour vendre l'énergie à d'autres systèmes (ou tiers)

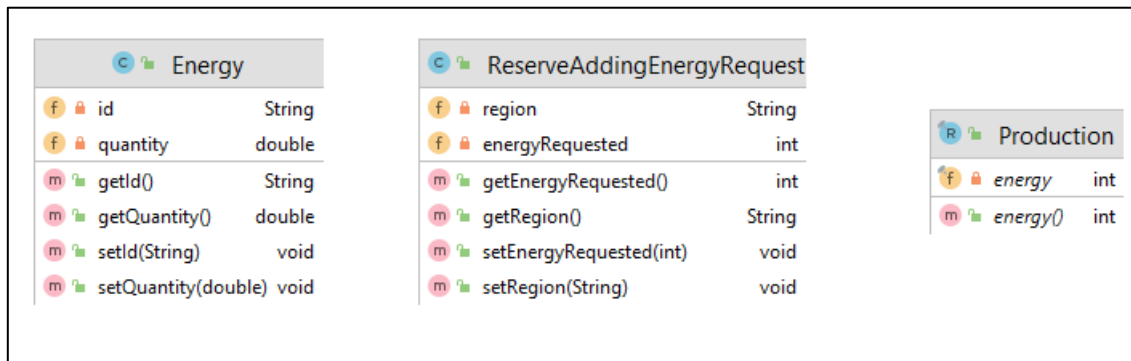


Diagramme de classe

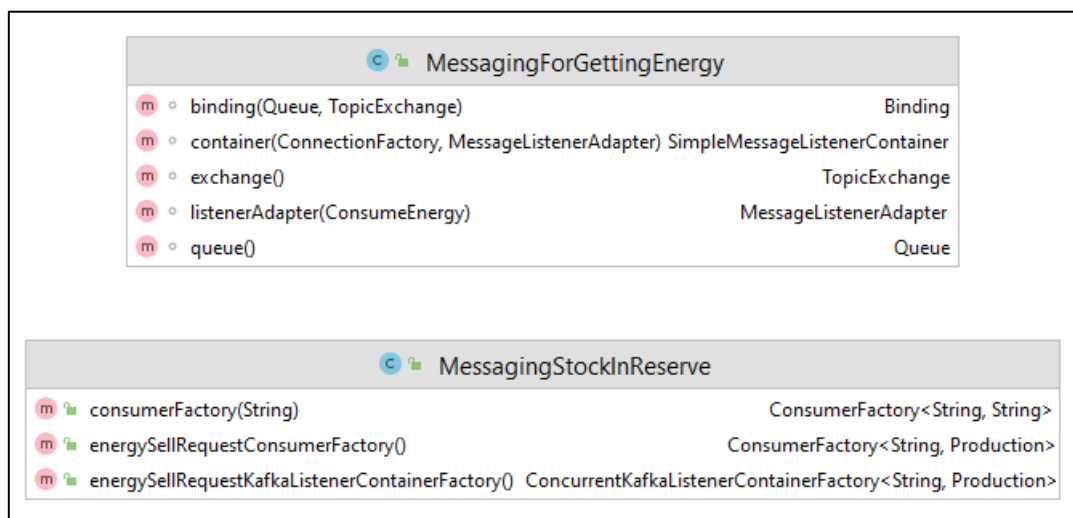


Diagramme de classe Interfaces RabbitMQ / Kafka

## Energy Supplier Service

Ce service représente les fournisseurs externe de smartrix grid, qui vont ajouter la production selon la demande (si les client consomme plus ou moins).

Ils ont une capacité maximale de production, c'est-à-dire ils peuvent renoncer à une demande d'augmentation de production s'ils ont déjà atteint leur capacité maximale.

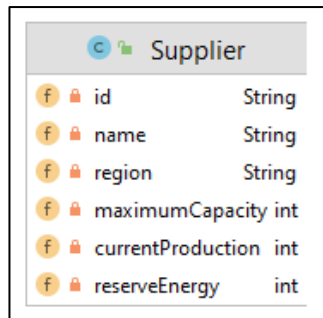
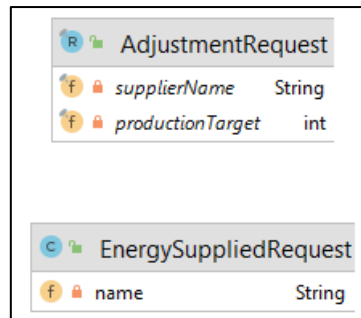
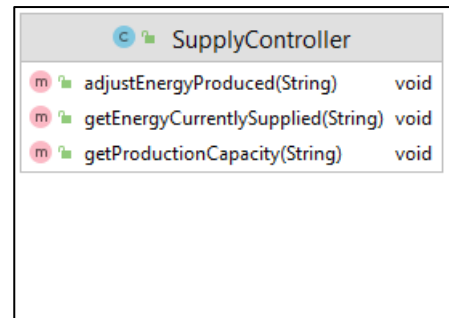


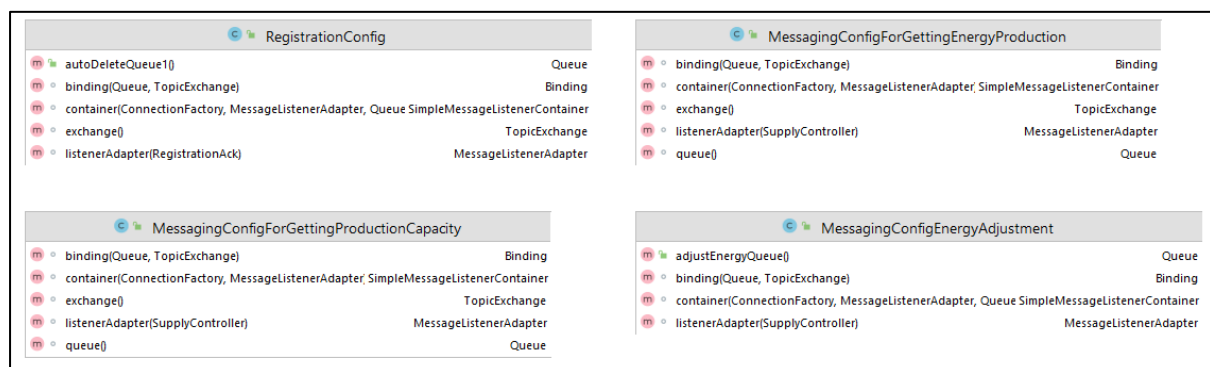
Diagramme de classe



Objets de requêtes



Interfaces REST



Interfaces RabbitMQ

## Accounting Service

L'AccountingService est le composant de comptabilité de notre application. Toutes les informations liées aux transactions d'argent y sont centralisée, et ce service permet notamment de générer des factures régulièrement afin que nos clients puissent consulter le coût de leurs consommations.

Vous retrouverez ci-dessous des diagrammes permettant de comprendre les données manipulées par le service ainsi que les besoins auxquels ce service répond.

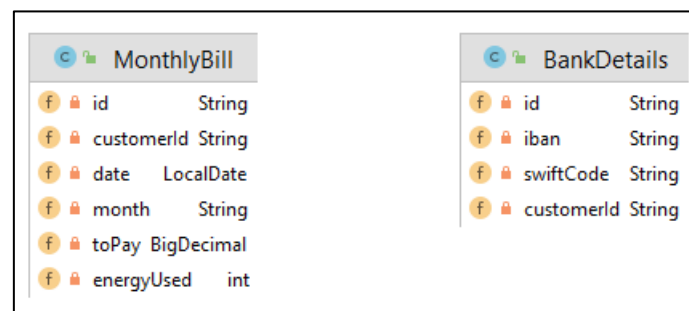
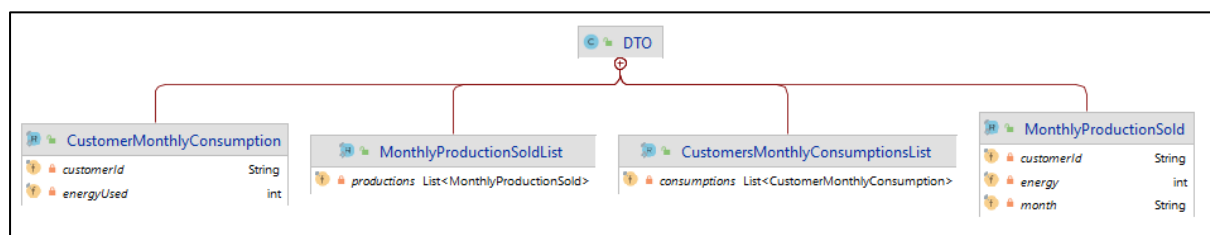
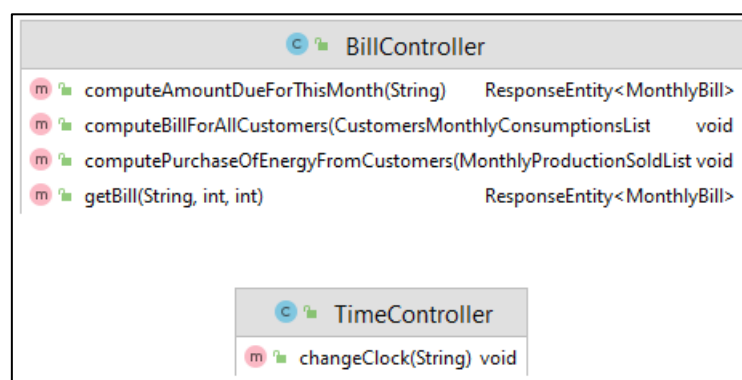


Diagramme de classe



Objets de transfert de données



Interfaces REST

MessagingForBankAccountInformation		
m	anonymousQueueCustomer()	Queue
m	bindingCustomerDuplication(Queue, TopicExchange)	Binding
m	containerDupContainer(ConnectionFactory, MessageListenerAdapter, Queue)	SimpleMessageListenerContainer
m	exchange()	TopicExchange
m	listenerAdapterCustomerDup(RegisterBankDetails)	MessageListenerAdapter

MessagingFromElectricityBroker		
m	binding(Queue, TopicExchange)	Binding
m	container(ConnectionFactory, MessageListenerAdapter)	SimpleMessageListenerContainer
m	exchange()	TopicExchange
m	listenerAdapter(ElectricityBrokerInterface)	MessageListenerAdapter
m	queue()	Queue

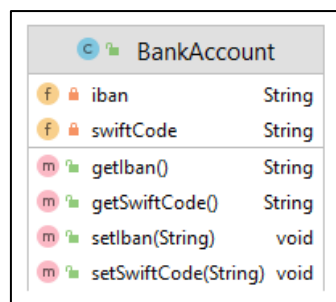
### *Interfaces RabbitMQ*



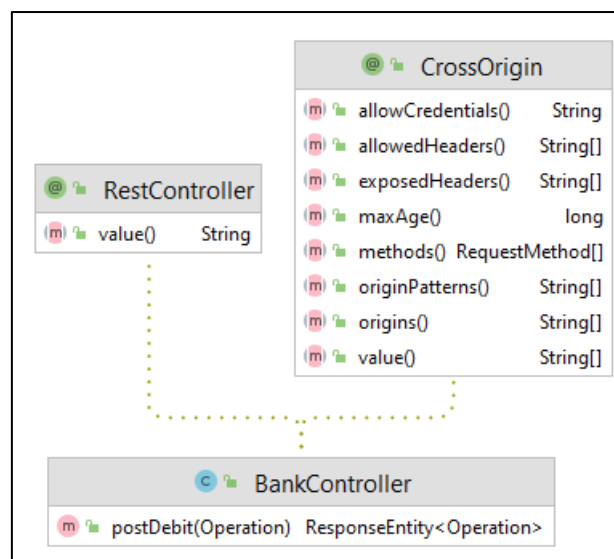
## Bank Service

Ce service représente une banque externe et permet de représenter toutes les transactions monétaires au sein de l'application. Il permet notamment de stocker les informations liées au compte bancaire des clients ainsi qu'à débiter/créditer les clients suite à leurs ventes/consommation.

Vous retrouverez ci-dessous des diagrammes permettant de comprendre les données manipulées par le service ainsi que les besoins auxquels ce service répond.



*Diagramme de classe*

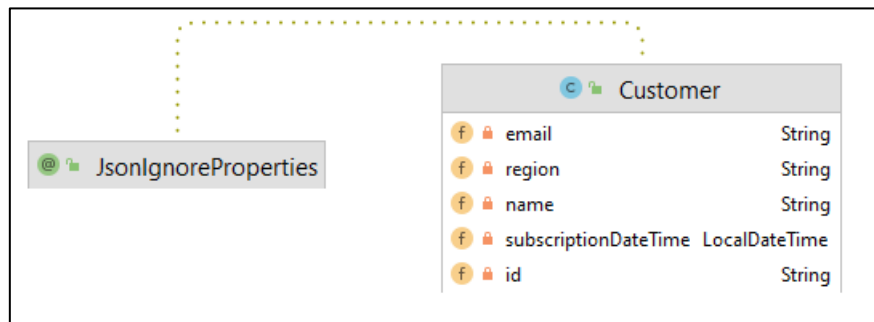


*Interfaces REST*

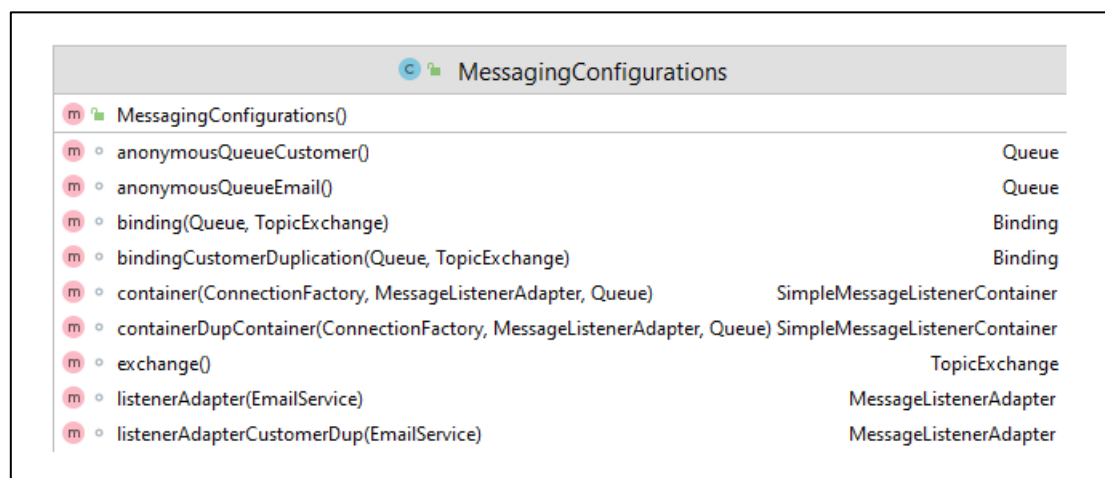
## Email Service

Lors de leurs inscriptions, les clients doivent spécifier leurs adresses emails, ces adresses seront plus tard dans certaines opérations de l'application. Ce service nous permet alors de communiquer directement avec les clients de l'application en offrant la possibilité d'envoyer des courriels.

Vous retrouverez ci-dessous des diagrammes permettant de comprendre les données manipulées par le service ainsi que les besoins auxquels ce service répond.



*Diagramme de classe*



*Interface RabbitMQ*

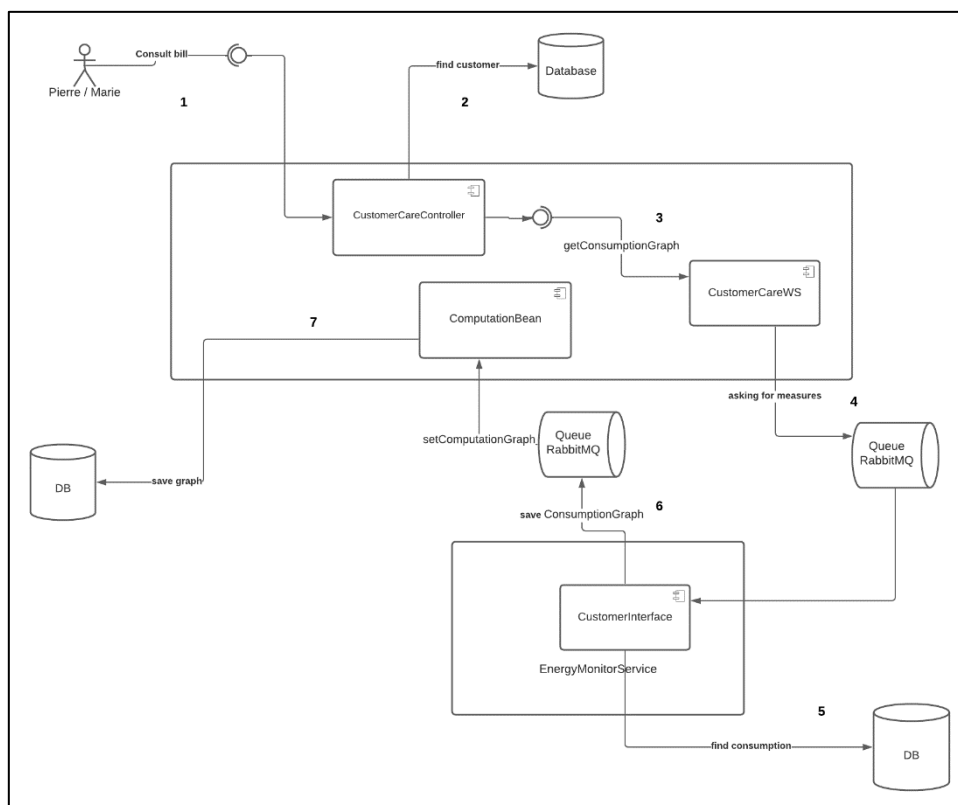
### III. Scénarios

#### User story 1

**En tant que** client nommé 'Pierre DUBOIS' **je souhaite** voir ma consommation **pour** connaître les jours où je consomme le plus d'énergie. Pour cela je vais sur mon interface je renseigne mon nom et demande à voir ma consommation pour les 30 derniers jours.

#### Scenario :

Après avoir renseigné ses informations et validé, une requête HTTP/GET est envoyée au service **CustomerService** qui contient un Endpoint qui va traiter cette requête. Le service **CustomerService** n'ayant pas accès aux informations de mesures des compteurs électriques, Il transfère donc la requête au service **EnergyMonitorService**. Au niveau du service **EnergyMonitorService**, la consommation de Pierre sur les 30 derniers jours dans une collection **dailyCustomerMeasure**, on y accède via l'interface **DailyAggregateRepository**. En effet lorsqu'on reçoit les consommations toutes les secondes on les sauvegarde dans une collection *measure* via l'interface **MeasureRepository**. Tous les jours à une heure du matin(01h00) on fait l'agrégation des consommations reçues la veille pour tous les clients et on stocke le résultat dans **dailyCustomerMeasure**. Le diagramme ci-dessous retrace ce scénario ainsi que les interactions nécessaires entre les différents composants.

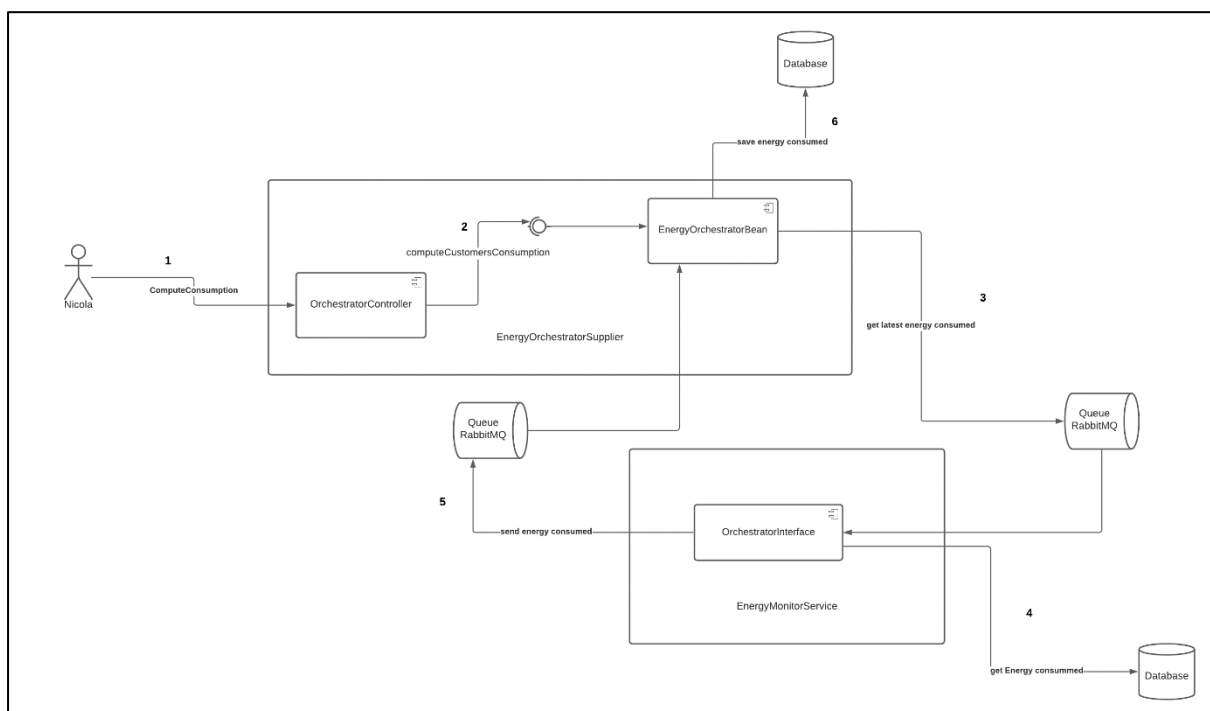


## User story 2 et user story 5

**En tant que** Nikola REED, COO de Smatrix Grid **je souhaite** m'assurer que la production reste toujours supérieure ou égale à la consommation, **afin que** le réseau électrique ne se surcharge pas.

### Scenario :

Le 24 décembre 2020 à 22h00 il décide de calculer la consommation de l'ensemble des **régions** pour les dernières 100 secondes. La requête suivante HTTP GET <http://localhost:4200/orchestrator/compute> est envoyée vers le service **EnergyOrchestratorSupplier**. Ce dernier va demander l'information sur les dernières mesures dans toutes les régions à **EnergyMonitorService** à travers la file de message. Lorsque le résultat lui parvient plus tard via une autre file, **Orchestrator** peut envoyer des **demandes d'ajustement de la production** de l'énergie aux services des fournisseurs en cas de besoin.



## User story 3

**En tant que** Charles DONNEL, CEO de Smatrix Grid **je souhaite** avoir une vue générale de la consommation de mes clients, **afin d'**adapter le besoin de consommation en énergie de mes clients.

### Scénario :

Pour cela je me connecte sur mon interface et je demande à voir la consommation mensuelle de mes clients pour ces 12 derniers mois. L'interface lance une requête HTTP/GET /ceo/{graphType} sur le service '**EnergyMonitorService**'. La variable '*graphtype*' prenant la valeur 'LAST\_DAY\_BY\_HOUR' pour avoir la consommation horaire sur les dernières 24 heures, 'LAST\_MONTH\_BY\_DAY' pour avoir la consommation journalière sur les derniers 30 jours et 'LAST\_YEAR\_BY\_MONTH' pour avoir la consommation par mois sur 1 an. Pour construire le graphe nous lisons Ces informations sont

disponibles dans les collections *measure* ou dans les collections annexes qui font l'agrégat des consommations.

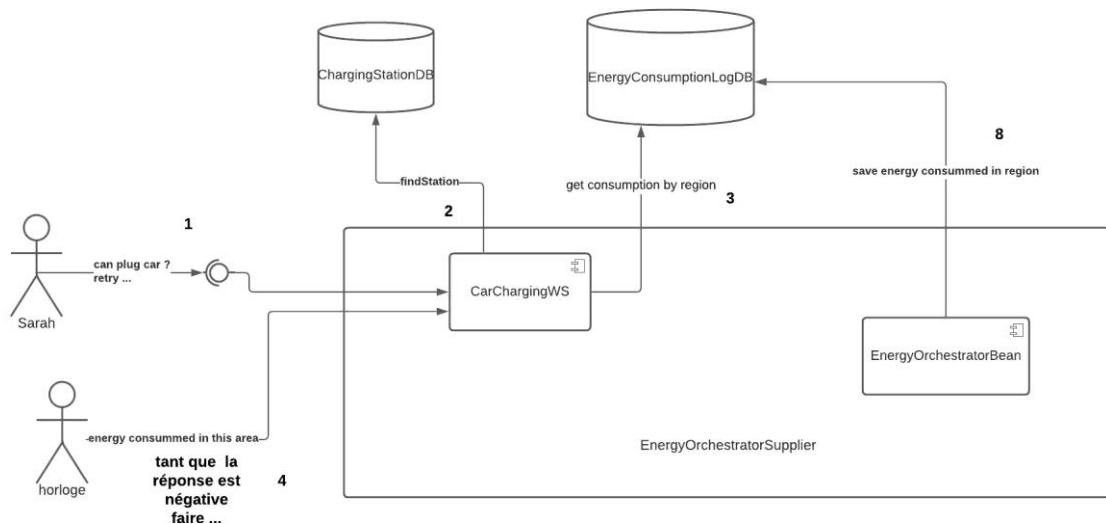
## User story 4 et user story 7

**En tant que** Sarah McBean, propriétaire d'une voiture électrique **je souhaite** recharger mon véhicule **pour** pouvoir l'utiliser le lendemain.

### Scenario :

Etant donné une grille surchargée à 18h, Lorsque Sarah branche son véhicule La station envoie au système de SmatrixGrid une requête (http post, avec le contenu de requête est un json contenant l'**id** de la station ainsi que la **durée** de rechargement et la **quantité** de l'énergie) pour demander l'autorisation de recharger le véhicule. On vérifie que région est surchargée ou pas. Comme la grille est en effet surchargée alors le véhicule de Sarah ne peut se recharger immédiatement. La station continue à envoyer des demande (**front**) toutes les heures afin d'obtenir l'autorisation (i.e. une fois la grille moins chargée)

À 01h00 du matin la grille est moins chargée, donc SmatrixGrid donne enfin son autorisation pour commencer le rechargement, ainsi le matin la voiture de Sarah sera chargée. Le diagramme ci-dessous retrace ce scénario ainsi que les interactions nécessaires entre les différents composants.



## User story 6

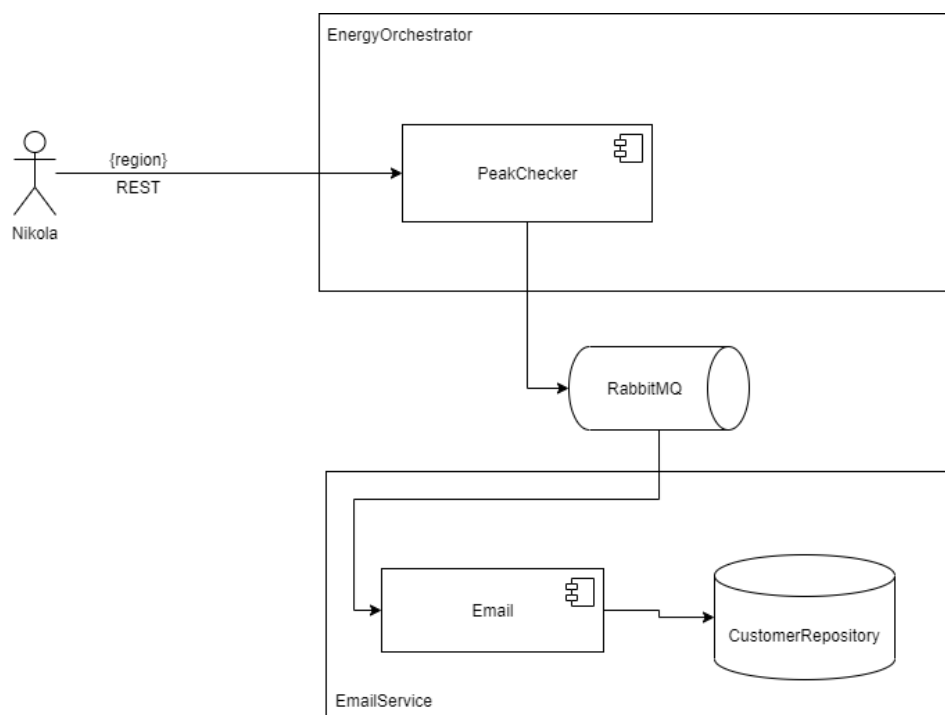
**En tant que** Nikola, **je souhaite** je veux garantir qu'il n'y a jamais de pic de consommation trop élevé dans une région **pour** éviter qu'une partie de la grille soit surchargée.

### Hypothèse :

Nikola peut d'ores et déjà consulter la consommation globale d'une région grâce à l'implémentation actuelle de l'application. Nikola souhaite éviter un pic dans sa grille et pour cela, il peut envoyer un mail à tous les habitants d'une région afin de les avertir de baisser leurs consommations lorsqu'une hausse est remarquée.

### Scénario et implémentation :

Nikola consulte la consommation actuelle de la région 'A'. Nikola réalise que la consommation de cette région est élevée et qu'elle est notamment sur une tendance grandissante. Souhaitant de pas surcharger cette région, il décide d'avertir tous les habitants de cette région par mail qu'une baisse en consommation est nécessaire. Par le biais du service 'EnergyOrchestrator', il déclenche un appel permettant de solliciter le service 'EmailService', sur une routing key dédiée en RabbitMQ. Ce service permet notamment de récupérer tous les 'Customers', répertorié à leurs créations dans le système, et d'envoyer un message par mail à ces derniers.



## User story 8, 9 & 15

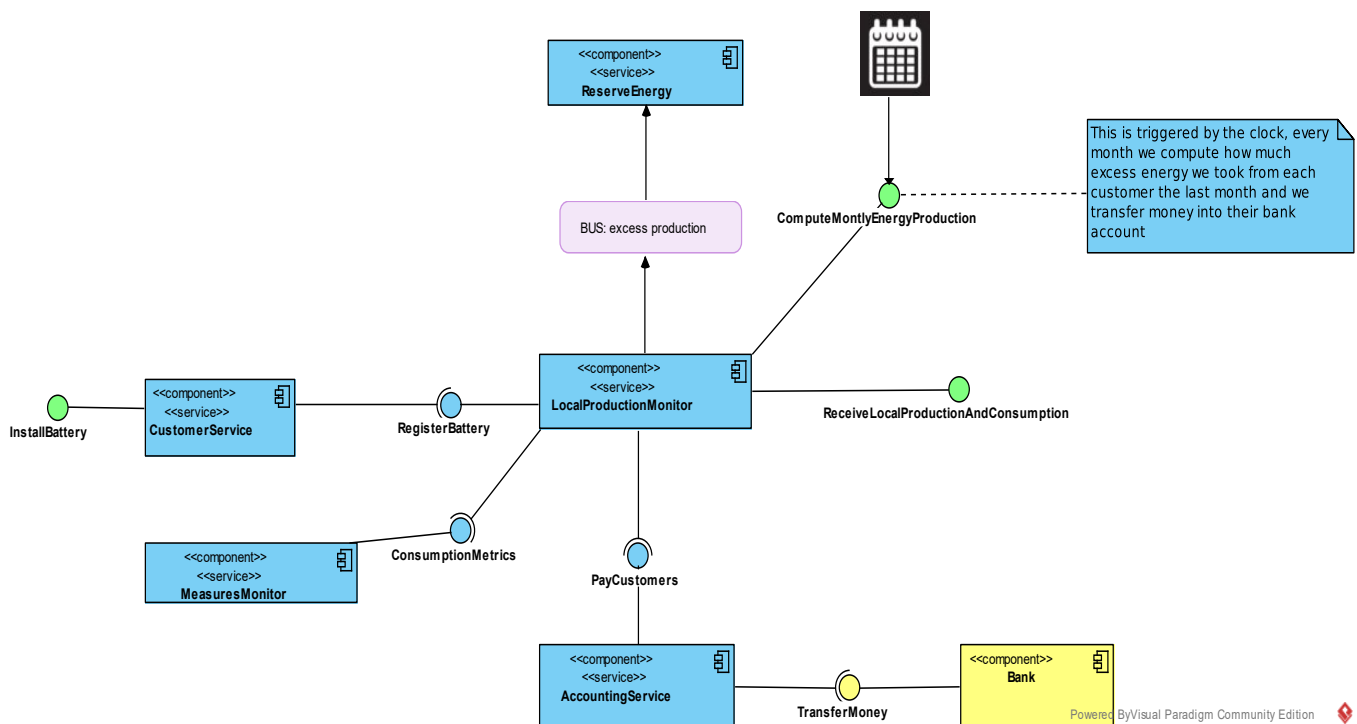
Pour ces fonctionnalités nous avons supposé que chaque client ayant un panneau solaire a aussi une batterie.

Le compteur électrique d'un client envoie les données de consommation et de production sur LocalProductionMonitor avec des requêtes POST, Si le client a une batterie LocalProductionMonitor utilise sa batterie et la production pour satisfaire la consommation. Si ceux-ci ne sont pas suffisants il transfère l'excès de consommation vers MeasuresMonitor. Si la batterie et la production sont suffisantes pour satisfaire la consommation, nous rechargeons la batterie du client. Si la batterie du client est pleine et qu'il y a un excès de production on l'envoie dans ReserveEnergy pour la mettre dans la batterie de SmartrixGrid, et on marque l'excès dans la base de données.

Afin de répondre au besoin 15 nous avons considéré que chaque client peut avoir une batterie, ainsi Thomas fait une requête HTTP/GET au CustomerService pour demander d'ajouter une batterie à un client en passant son nom, la demande est envoyée au CustomerProductionMonitorService qui va ajouter la batterie pour ce client et mettre à jour la base de données BatterieDB.

A la fin de chaque mois nous calculons pour chaque client le total d'excès qu'on a collecté. Et on utilise AccountingService pour transférer l'argent dans leur compte bancaire.

Nous utilisons un topic **Kafka** pour transférer les métriques d'excès de production dans ReserveEnergy.





## User story 10

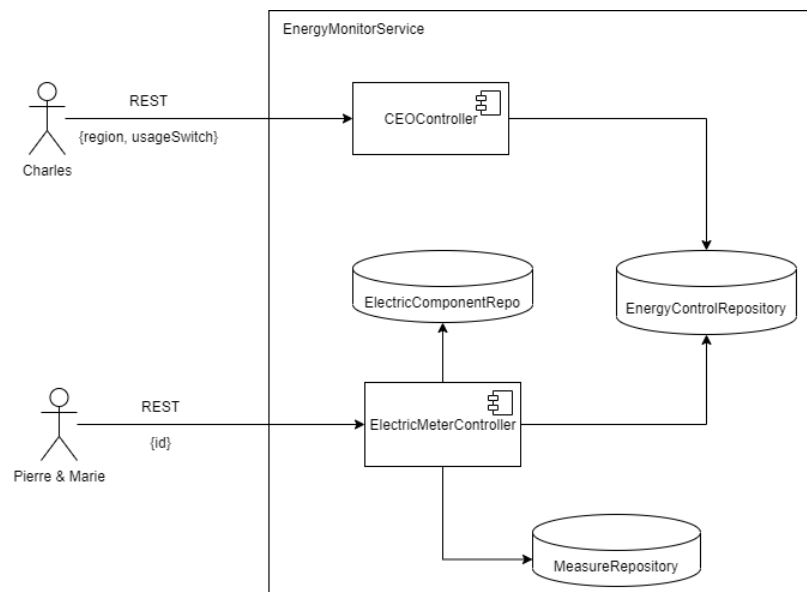
**En tant que** Charles, **je souhaite** avoir le control sur les consommations non-essentiels des régions **pour** afin de réduire la consommation globale de la grille.

### Implémentation :

Tous les composants électriques des utilisateurs de SmartrixGrid sont répertoriés, dans 'EnergyMonitor' comme étant 'Essentiel' ou 'Non-essentiel'. De plus, chacun de ces composants ont un état 'ON/OFF' qui peut être modifié par les clients. En temps normal, une horloge permettrait d'enregistrer en continue des consommations pour leurs composants allumé. Dans le cadre de ce projet, pour faciliter les démonstrations, les clients enverrons une requête 'saveMeasure' afin d'enregistrer des mesures.

### Hypothèse :

Charles a la possibilité de contrôler toutes les consommations non essentielles d'une région en bloquant/autorisant les consommations non-essentiels de la région. Cela permettra alors d'éteindre automatiquement tous les composants concernés de la région.



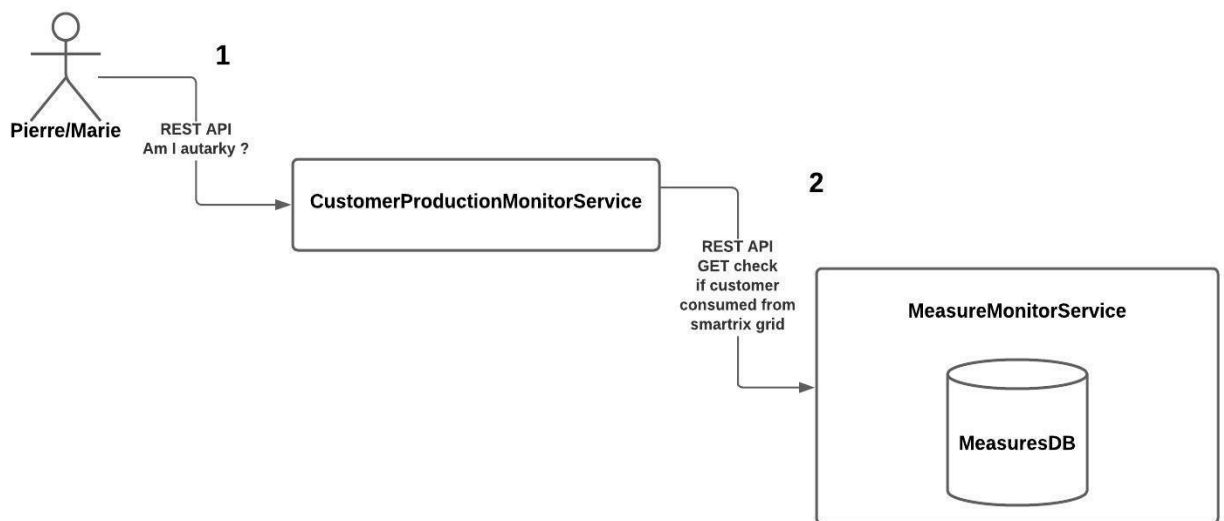
## User story 11

**En tant que** Pierre / Marie, un(e) client(e) de smartrix grid, **je veux** savoir si je suis auto-suffisant au niveau de ma consommation **afin de** réduire ma consommation si je ne le suis pas.

Afin de répondre à ce besoin notre équipe a compris ce besoin comme une comparaison au niveau de production de Pierre / Marie et de sa consommation.

C'est-à-dire si la capacité de production chez Pierre / Marie est  $\geq$  à sa consommation, alors notre système va lui répondre "Vous êtes en auto-suffisance". Pour cela, nous avons créé un service qui va permettre d'enregistrer la production des clients. C'est justifier par le fait qu'on ne voulait pas surcharger le service MeasureMonitorService.

Voici un dessin montrant la traversée de nos services afin de répondre à Pierre / Marie s'il(elle) est en auto-suffisance ou pas.



Donc ici Pierre / Marie va demander à travers une requête HTTP/GET s'il (elle) est autarky.

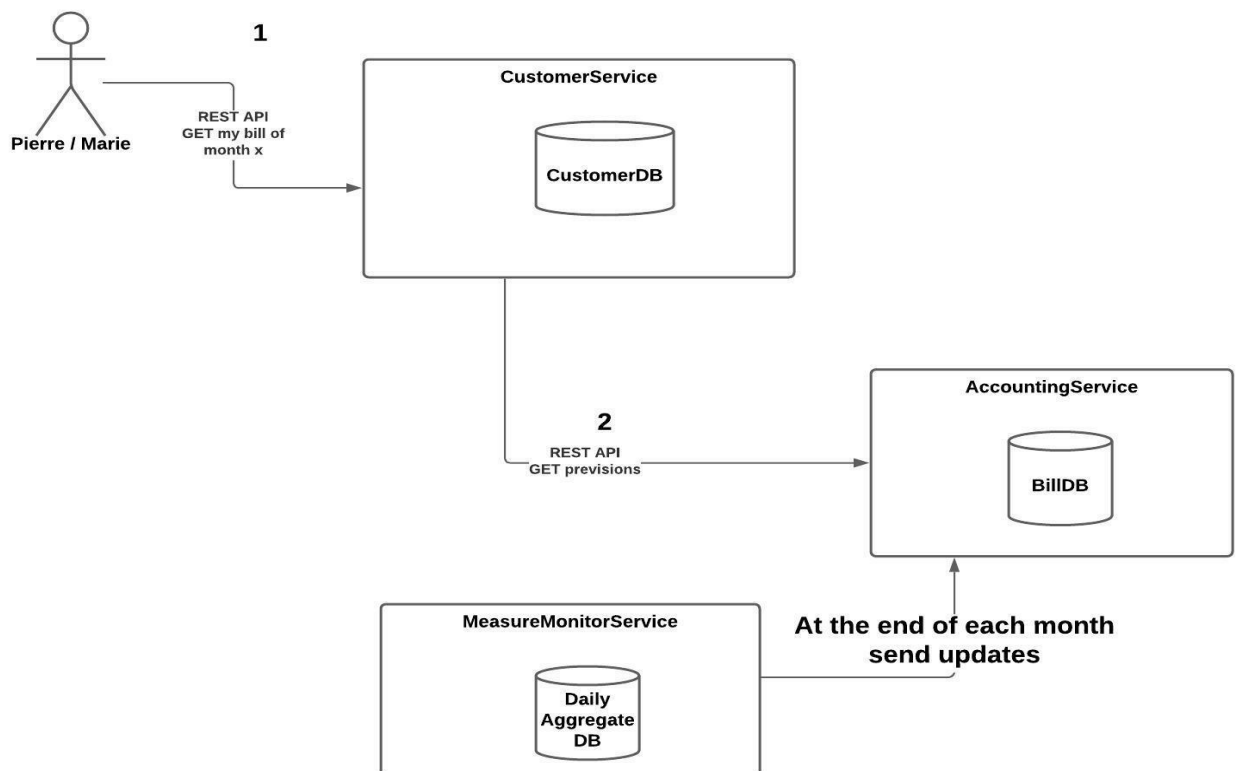
Le service de CustomerProductionMonitorService va vérifier s'il contient les logs de la production de ce(cette) client(e), si oui il va chercher sa production sur les 7 (codée en dur) derniers jours ainsi que sa consommation sur la même période et va faire une comparaison entre la production et la consommation et va répondre le client oui ou non s'il est autarky.

## User story 12

**En tant que** Pierre / Marie, un(e) client(e) de smartrix grid, **je veux** connaître ma consommation sur la dernière année, **afin de** savoir combien j'ai dû payer au total sur cette année.

Afin de répondre à ce besoin notre équipe avait déjà implémenté un service qui va enregistrer la consommation des clients dans une base de données (c'est le MeasureMonitorService), afin de faire des requêtes sur cette dépôt de stockage de consommation et récupérer la consommation des clients sur une période donnée.

Donc nous avons proposé l'architecture suivante



Ici on peut voir que lorsque le client effectue une requête en HTTP/GET pour récupérer sa consommation. Cette requête sera traitée par le service customerService qui va vérifier que ce client existe dans la base de données. Et ensuite il va demander à travers une API REST (aussi) la consommation de ce(tte) client(e) sur une période donnée (donc 1 an) au service MeasureMonitorService, et retourner la réponse au client.

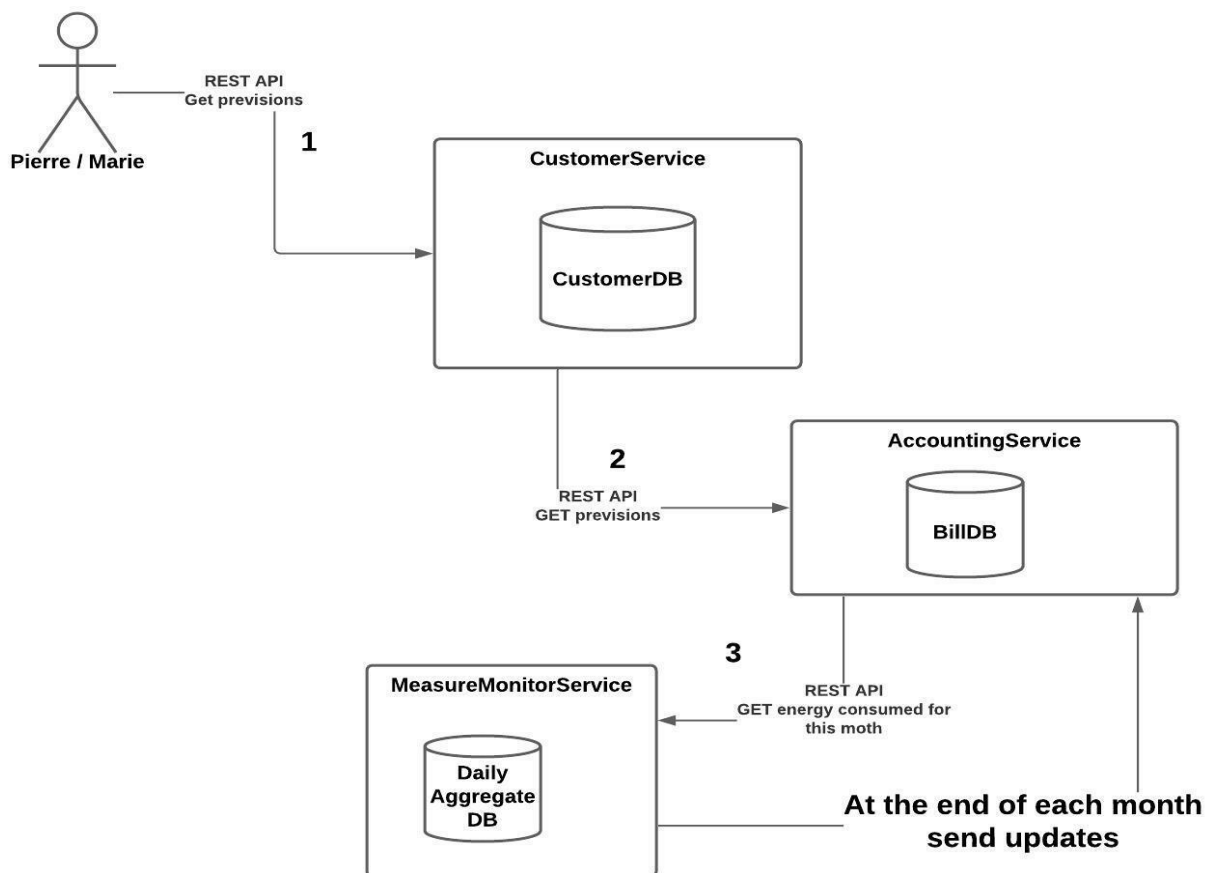
Ici le CustomerService va simplement interroger l'AccountingService pour récupérer la consommation du client sur un mois donné. Car le MeasureMonitorService envoie la consommation des clients à l'AccountingService à la fin de chaque mois, pour débiter les clients depuis la banque.

### User story 13

**En tant que** Pierre / Marie, **Je veux** savoir combien devrais-je payer ce mois à smartrix grid, **afin** d'ajuster mon budget.

Comme précédemment, pour répondre à ce besoin, le client va devoir passer par l'AccountingService qui va à son tour interroger le MeasurMonitorService, afin de récupérer combien ce(tte) client(e) a consommé sur ce mois et calculer le total que le client doit payer sur le mois en cours.

Donc l'AccountingService va représenter un la facturation entre les clients et smartrix grid. Pour éviter un refactoring conséquent nous avons pensé qu'il sera nécessaire de rajouter un tel service, pour deviser les responsabilités entre nos services, tout en évitant de faire l'over-engineering.



Dans ce schémas le client va devoir passer par le customerService (API REST) afin de connaître le montant à payer pour ce mois.

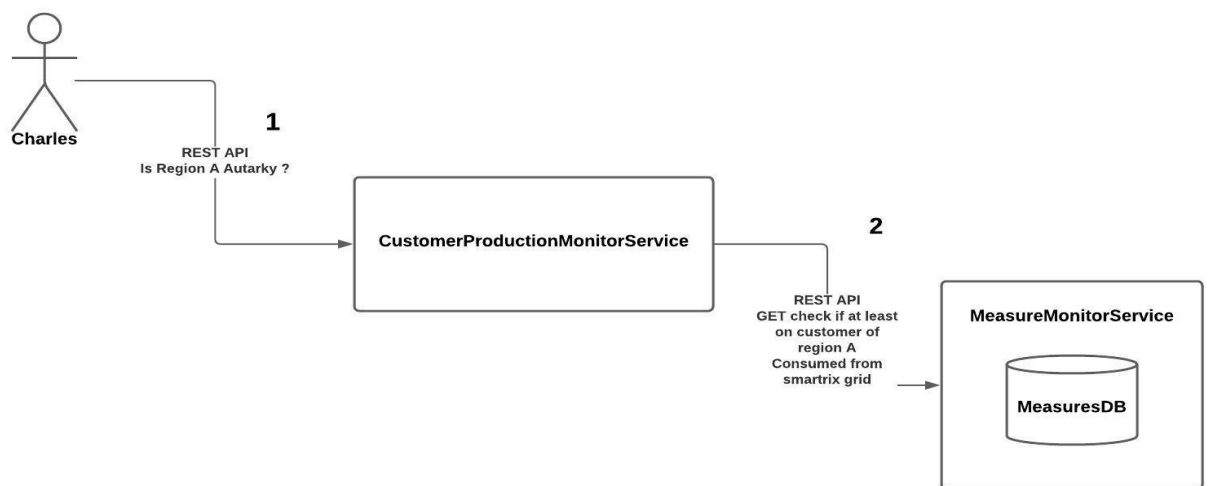
Ce dernier va solliciter l'AccountingService qui va à son tour récupérer la consommation du client depuis le début du mois jusqu'au jour de la consultation depuis le MeasureMonitorService. Ensuite, l'AccountingService va établir une facture qui sera retourner au customerService et donc au client.

Contrairement à l'user story 12, ici l'AccountingService va interroger directement le MeasureMonitorService pour récupérer la quantité consommé depuis le début du mois jusqu'au moment où Pierre / Marie a fait la demande.

## User story 14

**En tant que** Charles, **je veux** savoir si une région est autarky, **afin de** pouvoir mettre une stratégie de vente d'énergie en excès.

Afin de répondre à ce besoin, nous avons considéré que ce besoin ressemble au besoin de l'auto-suffisance de Pierre / Marie, que nous avons déjà expliqué plus haut. Comme les clients ils ont un label permettant de connaître à quelle région ils appartiennent, donc ce besoin est trivial.



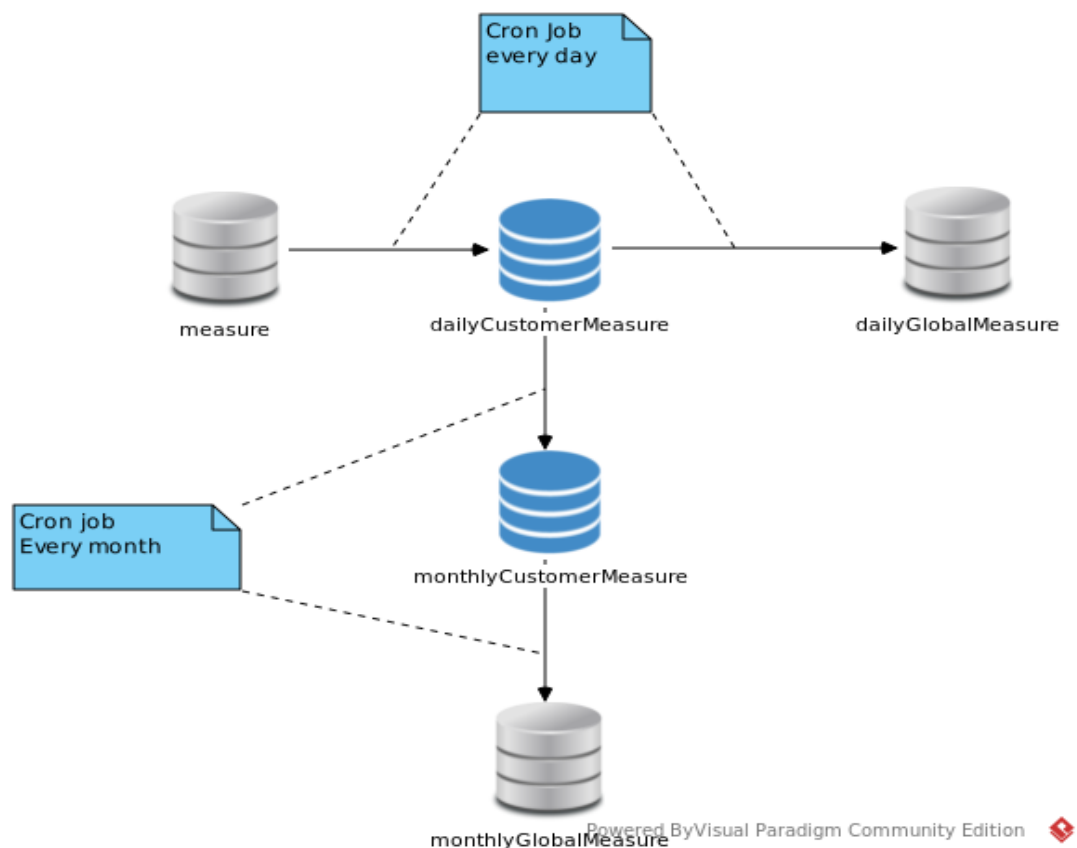
Chercher l'ensemble des clients d'une région afin de récupérer leurs productions. Ensuite, on récupère la consommation des clients de cette région depuis le service MeasureRepository, et comparer leur production et consommation sur 7 jours (codée en dur), avant de déduire si la région est autarky ou pas.

## User story 16

Notre système doit permettre de supporter des milliers d'utilisateurs. Nous avons donc commencé par identifier les services et les parties du code critiques. Il s'est avéré que l'un des services critiques est EnergyMonitorService. En effet, ce service reçoit le flux de consommation de tous les clients à intervalles réguliers. Ce flux de consommation est stocké dans la base de données et est utilisé pour un certain nombre de fonctionnalités comme l'orchestration des fournisseurs d'énergie, le calcul des factures à la fin du mois, la visualisation du graphe de consommation des clients. Au départ nous utilisons la même collection *measures* pour lire et écrire ces données et nous faisons les agrégations et les calculs au niveau du code Java. Pour rendre le système capable de gérer plusieurs milliers d'utilisateurs nous avons commencé par ajouter de nouvelles collections :

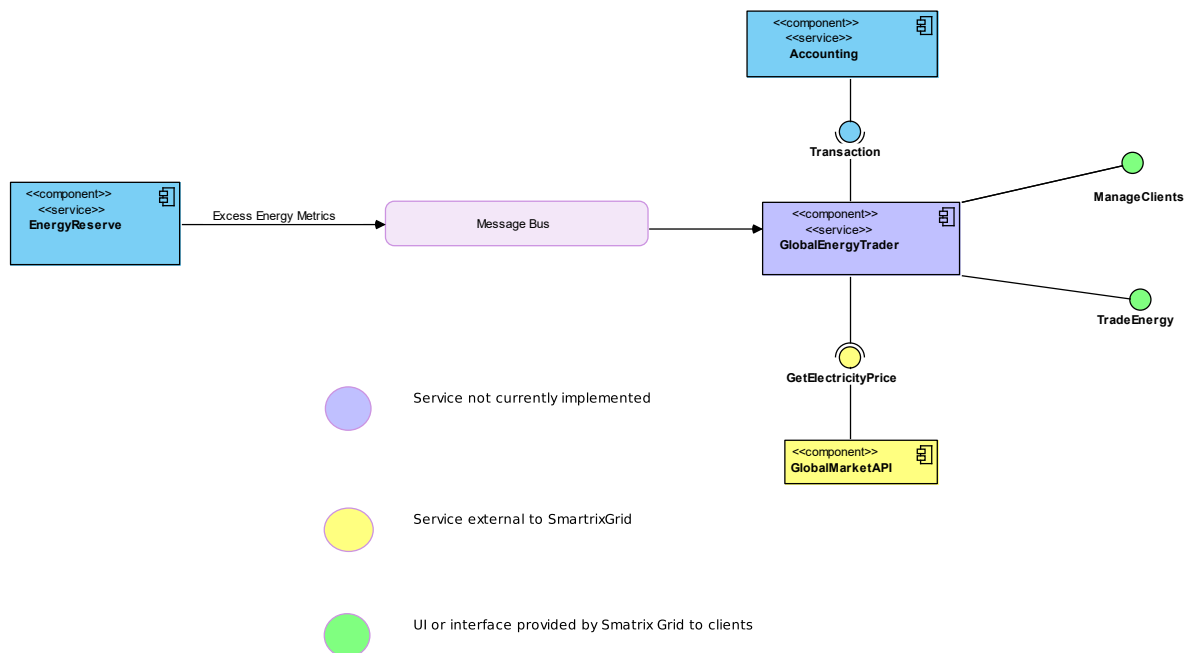
- **DailyCustomerMeasure** : contient l'agrégat des consommations regroupées par client et par jour
- **DailyGlobalMeasure**: contient l'agrégat des consommations regroupées par jour
- **MonthlyCustomerMeasure**: contient l'agrégat des consommations regroupées par client et par mois
- **monthlyGlobalMeasure** : contient l'agrégat des consommations regroupées uniquement par mois

Le schéma suivant illustre comment on agrège le résultat à partir d'une collection source et on le stocke dans une collection destination. Nous avons également utilisé l'annotation Java afin de faire les filtrages, regroupement et projection directement au niveau de la base de données MongoDB plutôt qu'au niveau des services Java. Ainsi lorsqu'une fonctionnalité qui nécessite les données journalières ou mensuelles doit être exécuté on lit juste dans une des collections secondaires ainsi les traitements des requêtes se font plus vite. On ne lit dans *measure* que lorsque qu'on veut des données avec une granularité horaire ou moins.



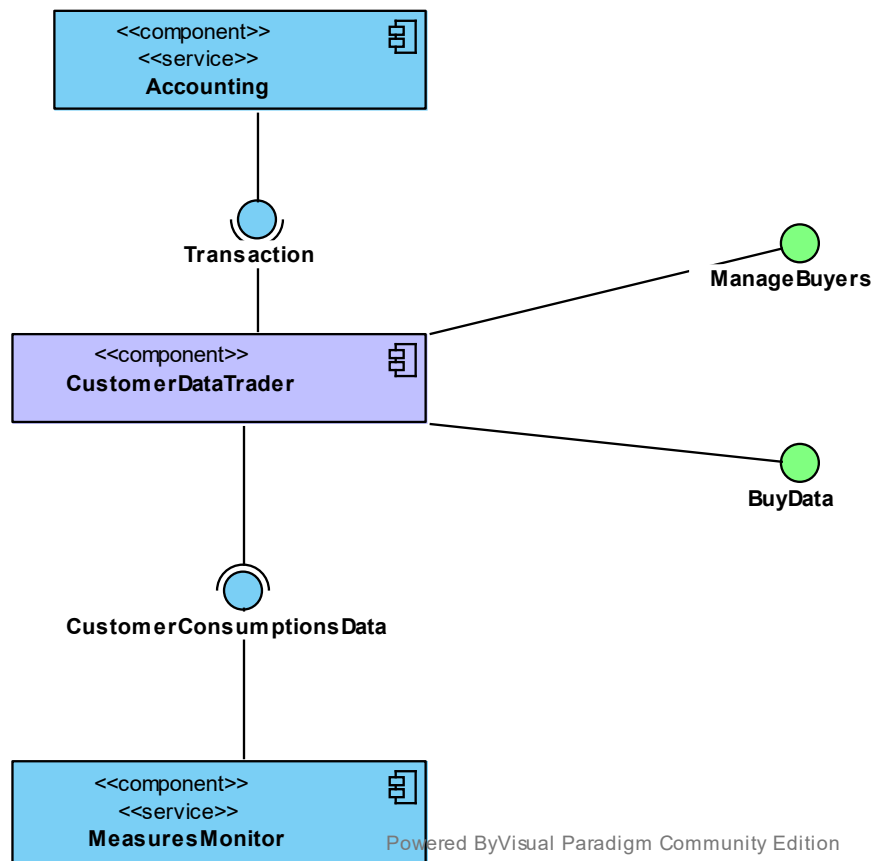
## User story 17

Il s'agit de rajouter la fonctionnalité de Virtual Power Plant. Nous ne l'avons pas implémenté dans notre version de rendu. Pour l'implémenter en se basant sur l'architecture existante nous rajouterions un service GlobalEnergyTrader qui sera chargé de gérer nos clients sur le marché de l'électricité globale et de gérer les ventes. Actuellement l'excès d'énergie produit par les clients (particuliers) de Smatrix Grid est stocké dans une batterie gérée par le service EnergyReserve. Si cette batterie est pleine, nous vendrons le reste de l'excès sur le marché global si le prix actuel de l'électricité nous convient. Le service Accounting va permettre de gérer les transactions financières et encaisser l'argent des ventes d'électricité.



### User story 18

Il s'agit de vendre les données des consommateurs à des partenaires. Pour cela nous rajouterions un service CustomerDataTrader pour gérer les partenaires et contrôler l'accès aux données de consommations de nos clients. CustomerDataTrader est connecté à MeasuresMonitor car les données de consommations sont stockées dans MeasureMonitor. Les données de consommations dans MeasureMonitor sont anonymisées car il ne contient pas le nom des clients ou leur adresse mais juste des customerId et une information de région générale, que les partenaires externes qui achètent les données ne pourront pas associer à l'identité de nos clients.



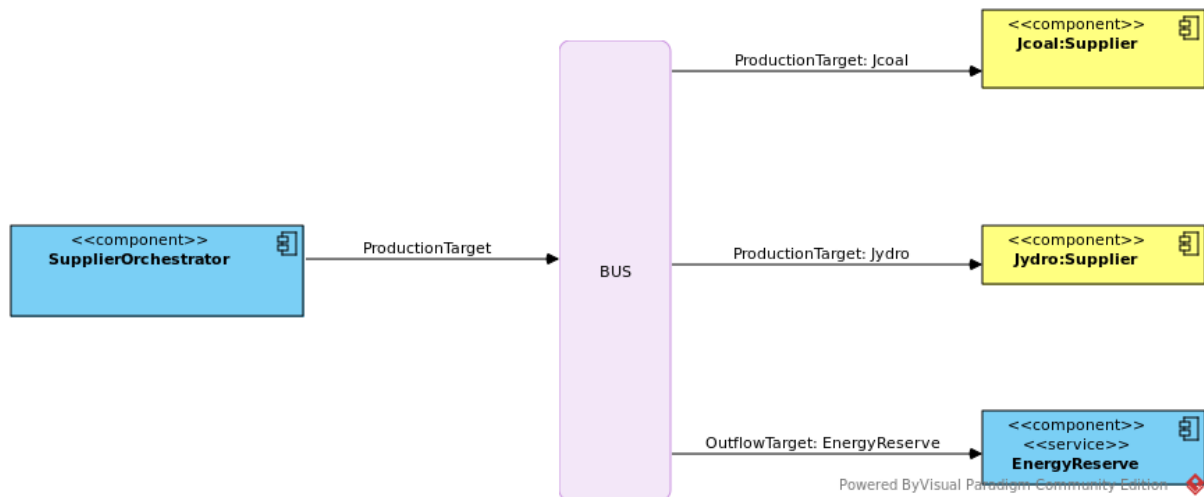
Powered By Visual Paradigm Community Edition





## User story 19

Cette fonctionnalité a été implémentée très tôt dans le projet. En effet nos fournisseurs d'énergie ont une limite de puissance qu'ils peuvent fournir à Smatrix Grid. Lors de l'orchestration nous prenons en compte cette valeur. Lorsque nos fournisseurs sont tous saturés nous pouvons demander l'énergie au stockage de EnergyReserve.

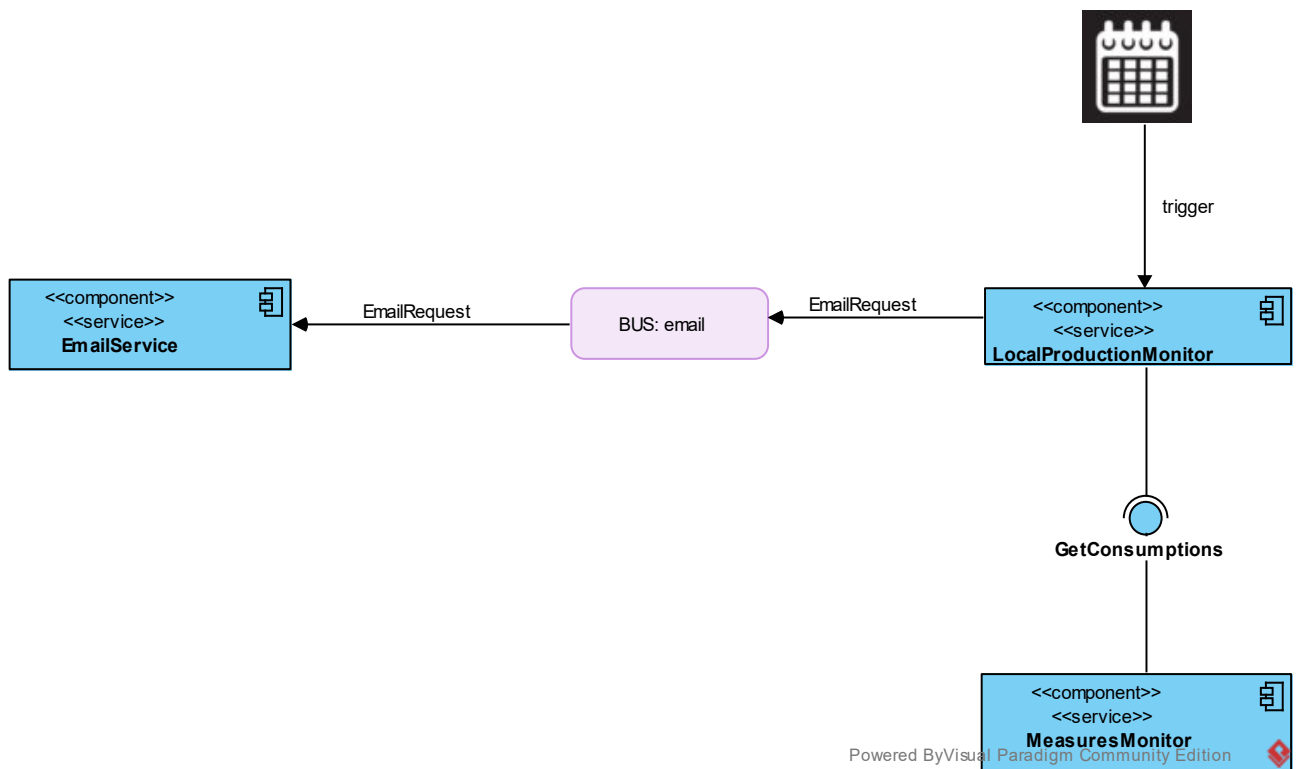


## User story 20

Il s'agit de notifier un utilisateur si son statut d'autonomie ou celui de sa région change. Cette fonctionnalité n'est pas implémentée dans la version de rendu. Pour l'implémenter on n'a pas besoin de rajouter un nouveau service. Il suffit de rajouter un cron job dans LocalProductionMonitor qui va tous les mois vérifier si tous les clients qui ont une batterie/panneau solaire et toutes les régions sont en autonomie. La première fois que ce cron job est lancé pour un client ou une région le résultat sera juste marquer dans la base de données. Les fois suivantes le résultat obtenu sera comparer avec le résultat déjà présent dans la base de données et si le résultat a changé on envoie un courriel au client concerné ou à tous les clients de la région.

```
public class RegionAutarky {  
    @Id  
    private String region;  
    private boolean isInAutarky;  
}
```

```
public class CustomerAutarky {  
    @Id  
    private String customerId;  
    private boolean isInAutarky;  
}
```



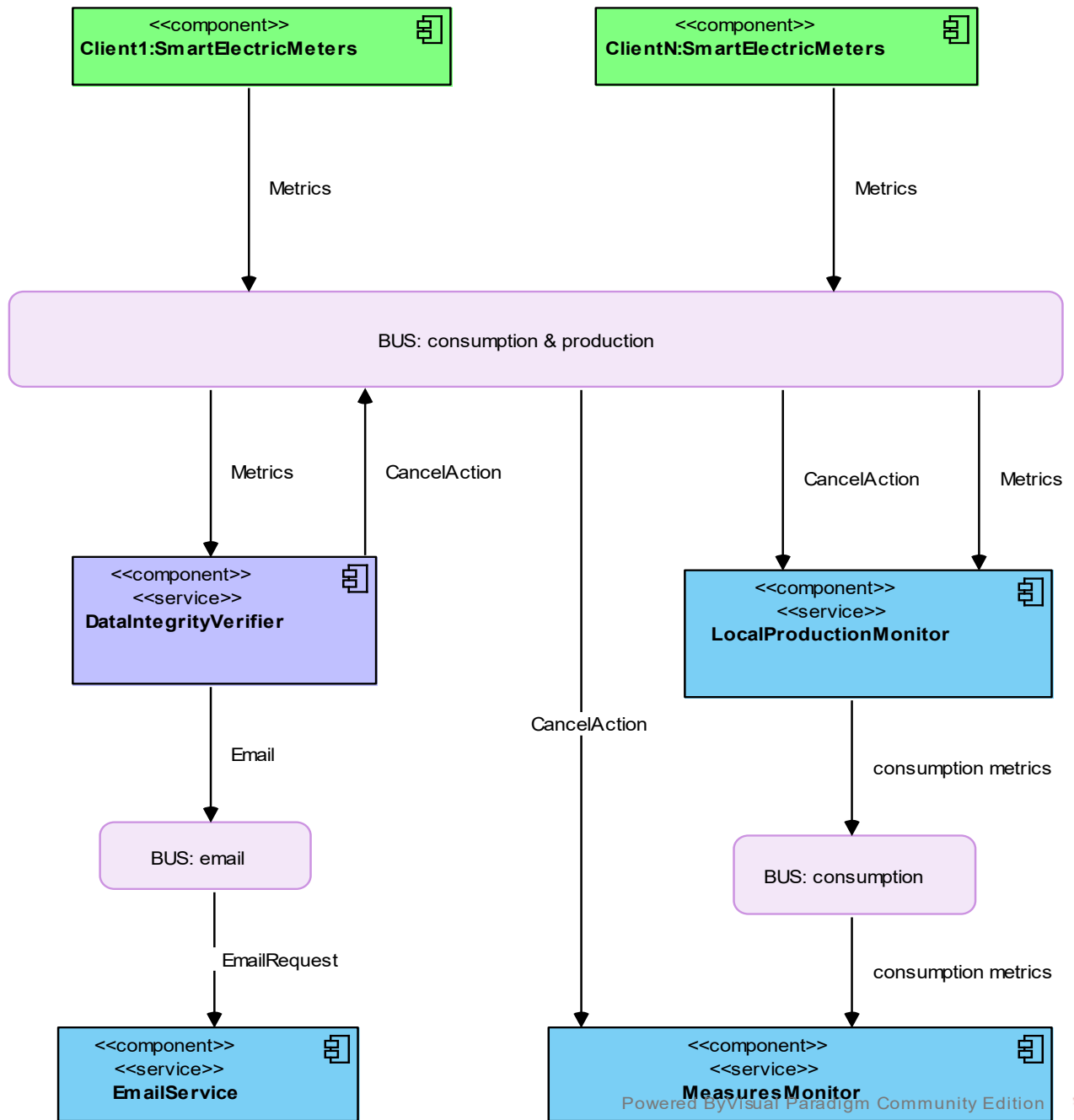
## **User story 21**

Il s'agit de monitorer les données de consommation et de production afin de détecter des aberrations (métriques négatives ou trop grandes) ou des pannes. Cette fonctionnalité n'a pas été implémentée dans la version rendue. Si on devait l'implémenter, ferions certaines modifications et ajouts sur notre architecture actuelle.

Actuellement les compteurs électriques envoient les données de production locale et de consommation via des requêtes POST. Nous allons remplacer cela par l'utilisation d'une file de message sur laquelle vont s'abonner le LocalProductionMonitor et un nouveau service DataIntegrityVerifier. Nous rajoutons remplaçons aussi l'utilisation d'un http/POST pour l'envoi des données de consommation de LocalProductionMonitor vers MeasuresMonitor par un bus de message. DataIntegrityVerifier va se charger de vérifier si des valeurs aberrantes sont envoyées par les compteurs électriques. Si c'est le cas il peut :

- envoyer un courriel à Charles via EmailService pour que celui-ci envoie un technicien hors du système.
- envoyer sur le bus une CancelAction pour annuler les effets des données aberrantes reçues. Cette CancelAction est consommée par MeasuresMonitor et LocalProductionMonitor.

La raison pour laquelle nous faisons consommer les métriques à DataIntegrityVerifier en même temps qu'elles sont traitées par LocalProductionMonitor et MeasuresMonitor est que les cas d'erreur et de panne seront rares par rapport au fonctionnement normal. Du coup nous préférons que LocalProductionMonitor et MeasuresMonitor commettent des erreurs que nous annulerons après coup. En effet une autre solution possible serait d'utiliser DataIntegrityVerifier comme pare-feu et vérifier l'intégrité de toutes les métriques avant de les transférer à LocalProductionMonitor mais cela aurait pour conséquence de ralentir le fonctionnement du système si la logique de vérification d'intégrité est complexe et prend du temps. Notre solution privilégie la vitesse par rapport aux fautes et utilise un mécanisme d'annulation des fautes.



## IV. Limites de l'architecture

Bien que nous ayons proposé une architecture complexe et qui permet de répondre aux précédents besoins (tout en évitant l'over-engineering) cette architecture a des limites.

- 1- Le service du reserveenergy peut stocker une quantité infinie d'énergie.
- 2- Certaines choses sont codées en dure, par exemple quand un client veut savoir s'il est en autarky, on regarde la consommation et la production de ce dernier sur les 7 dernières jours.
- 3- Comme nous avons une architecture complexe, il aurait été intéressant d'avoir une api gateway qui sera relier aux différents services. Ainsi les clients vont uniquement interroger une seul endpoint.
- 4- Concernant l'user story 16, on continue de lire régulièrement dans *measures* pour faire l'orchestration de l'offre d'énergie par rapport à la demande parce que nous avons besoins des dernières données pour avoir l'évolution de la consommation d'énergie.
- 5- Le service LocalProductionMonitor est le second service critique pour le passage à l'échelle. Il a un fonctionnement similaire à EnergyMonitorService, il stocke les métriques de productions locales d'électricité des clients de SmatrixGrid. Il est possible d'appliquer la solution utilisée dans EnergyMonitorService afin de réduire le temps de traitement de requêtes.

## V. Bilan

L'application comportait initialement 4 services internes qui pouvaient interagir entre eux afin de remplir un besoin donné. Au fil de l'ajout des nouvelles user stories, l'application était amenée à évoluer avec l'ajout de nouveaux services permettant de répondre à de nouveaux besoins. Un des enjeux majeurs a été de s'assurer de ne pas surcharger les services rendus par services, et de constamment remettre en question les tâches accomplies par ces dernières. De plus, il était nécessaire que ces tâches restent cohérentes aux limites imposées au service.

Finalement, nous voulions nous assurer de connaître les services les plus cruciaux de notre application et d'identifier les services soumis aux charges les plus importantes. Suite à ces études, nous pouvions conclure sur les méthodes les plus optimisées pour les échanges de données au sein de l'application en nous adaptant également aux nouvelles notions apprises en cours, comme l'utilisation de Kafka.

Bien que certaines user story n'étaient pas à implémenter pour ce rendu, nous sommes à présent confiant de nos capacités à étudier un besoin et de produire une architecture complète adaptée, qui permet ensuite de développer une solution cohérente et adaptable aux besoins futurs.