Projet de fin d'études

R&D démonstrateur web : Conception et développement d'un démonstrateur web d'un flux de travail boutique-logistique-entrepôt et de système IoT

Encadré par : Bruno Le Fellic, Vincent Ricordel

Rédigé par : Zijun PAN

POLYTECH NANTES - ETN5
Mars - Août 2020

Sommaire

Abstract

La première phase de ce stage de fin d'études était consacrée au développement d'un démonstrateur d'un flux de travail boutique-logistique-entrepôt. Le travail comprend le développement de pages, la modification d'API et le déploiement de pages Web.

Dans la partie back, on utilise MongoDB, une base de données distribuée, universelle et basée sur des documents, pour stocker les données fictives générées par un script Python. L'API de ce projet est construite avec Python Flask et Swagger, qui nous permet de récupérer les données persistantes.

Dans la partie front, on utilise Angular, le framework front-end de Google, pour construire un site web structuré et évolutif. Typescript(un sur-ensemble de Javascript), CSS et HTML sont à la base de la construction d'un composant Angular. Angular Material, une bibliothèque de components UI, est utilisée pour améliorer l'interface utilisateur.

Le deuxième projet du stage consiste à transformer un système de recharge de télécommunications distribué à une solution de gestion de big data distribuée pour les piles de recharge de véhicules électriques On intègre les données des utilisateurs et les données collectées, et publie les données intégrées sur Kafka.

Le logiciel Git est utilisé pour la synchronisation de projet avec le dépôt Git distant. Le front et le back se communiquent en JSON. Docker et docker-compose sont utilisés pour le déploiement des microservices sur une machine Linux.

1. Introduction

Ce rapport est une présentation de mon stage de six mois chez SpikeeLabs à Rennes. SpikeeLabs est une entreprise qui propose à ses clients les concepts et les réalisations des systèmes de service d'information.

J'ai travaillé sur la solution BillingLabs de l'entreprise, le but étant de réaliser la fonction et l'affichage de l'ensemble du processus du système de service, y compris API, Analytics et IHM.

1.1. Contexte

Mon stage se compose de deux projets: le but des deux projets est de migrer le projet original de l'entreprise vers la nouvelle scène pour la réalisation, afin que notre fonction d'origine puisse être itérée, mise à jour et étendue dans la nouvelle scène.

Le premier projet est un système de gestion des télécommunications basé sur l'entreprise. Nous espérons migrer le système vers le scénario entrepôt-logistique-magasin pour construire un système de gestion ETL entrepôt-logistique-magasin.

Le deuxième projet est basé sur le système d'architecture distribuée de l'entreprise. Nous espérons faire migrer un système de télécom billing vers le scénario de piles de recharge de véhicules électriques. Grâce à la collecte d'informations du système distribué, nous collectons la file d'attente des messages du projet et effectuons certaines analyses big data sur les données.

Le temps consacré à chaque projet représente la moitié de l'ensemble du cycle de stage.

1.2. Objectives

Le but de mon stage est de fournir un soutien aux entreprises dans le processus de migration de nouveaux projets, et de conduire la recherche et le développement de certains projets. Dans ce processus, je connais les différentes technologies utilisées dans l'entreprise et les opérations quotidiennes de l'entreprise.

1.3. À propos de SpikeeLabs

Spikeelabs est immatriculée en 2016. C'est une entreprise ESN basée à Rennes mais sert pour les client autour de la France. Depuis sa création, SpikeeLabs connaît une forte croissance en termes de chiffre d'affaires.

Le logo de SpikeeLabs

Comme une entreprise ESN typique, Spikeelabs fournit les services sur 4 activités:

- Conseil: Analyse les existences ainsi que les besoins. Donner la conception, Créer le cahier des charges pour le produit du client
- Réalisation: Développement du architecture et projet par rapport aux besoins du client. migration pour la base de données, amélioration et test de fonctionnements.
- Intégration: Appliquer le produit de Spikeelabs au projet existé
- Support: Maintenance, évaluation et adapation pour un projet du client.

Les clients de SpikeeLabs

Aujourd'hui le chiffre d'affaire est arrivé à 4 million euro et il y a plus de 50 salariés qui travaillent dans le bureau à Rennes, à Paris et à Nantes.

Cartes des 3 sites SpikeeLabs en France

Plus d'informations peuvent être trouvées sur ce lien: https://www.spikeelabs.fr/ .

2. Technologies

2.1. Développement

Dans nos projets, nous utilisons des technologies auxiliaires pour faciliter notre développement.

2.1.1. Angular

Afin de simplifier à la fois le développement et les tests de SPA (application d'une seule page), Angular peut nous fournir des architectures MVC (model-view-controller) et MVVM (model-view-viewmodel) côté client.

Le front-end de notre projet télécom original a été développé en utilisant Angular.

Logo d'Angular

Plus d'informations sur Angular peuvent être trouvées ici : https://angular.io/ .

2.1.2. Vue

Vue est un framework évolutif pour construire des interfaces utilisateur. Il s'agit d'un framework MVVM léger qui fournit une liaison de données efficace et un système de composants flexible via une API simple.

Afin de réaliser un développement rapide, nous utilisons le framework Vue léger et simple pour développer le Dashboard du projet IoT.

Logo de Vue.js

Plus d'informations sur Angular peuvent être trouvées ici : https://vuejs.org/ .

2.1.3. Swagger

Swagger est une suite d'outils de développement d'API à la fois puissants et faciles à utiliser pour les équipes et les individus, permettant le développement sur l'ensemble du cycle de vie de l'API, de la conception et de la documentation aux tests et au déploiement.

Afin de faciliter la réalisation des fonctionalités de l'API, nous utilisons le Swagger Codegen pour la génération de code et Swagger UI pour la visualisation.

Logo de Swagger

Plus d'informations sur Swagger peuvent être trouvées ici : https://swagger.io/ .

2.1.4. Flask

Flask est un framework d'application web WSGI (Web Server Gateway Interface) léger. Il est conçu pour rendre la mise en route rapide et facile, avec la possibilité de s'adapter à des applications complexes.

Afin de réaliser un hébergement de services Web back-end, nous utilisons Flask pour nous aider à créer le serveur.

Plus d'informations sur Flask peuvent être trouvées ici : https://palletsprojects.com/p/flask/ .

2.1.5. MongoDB

Afin de rendre l'intégration des données plus facile et plus rapide, nous utilisons MongoDB comme base de données pour développer des documents de type JSON avec des schémas dynamiques.

Logo de MongoDB

Plus d'informations sur MongoDB peuvent être trouvées ici : https://www.mongodb.com/ .

2.1.6. SQL Server

SQL Server est un système de gestion de base de données (SGBD) en langage SQL incorporant entre autres un SGBDR (SGBD relationnel) développé et commercialisé par la société Microsoft.

Nous utilisons une base de données SQL Server pour stocker les informations relatives au compte.

Logo de SQL Server

Plus d'informations sur SQL Server : https://www.microsoft.com/sql-server

2.1.7. Cassandra

Cassandra est un SGBD de type NoSQL conçu pour gérer des quantités massives de données sur un grand nombre de serveurs. C'est une solution de stockage de données structurée distribuée populaire.

Dans notre projet IoT, nous l'utilisons pour stocker les données EDR (Event Detail Record).

Logo de Cassandra

Plus d'informations sur Cassandra peuvent être trouvées ici : https://cassandra.apache.org/

2.1.8. RabbitMQ

RabbitMQ est un logiciel d'agent de messages open source qui implémente le protocole Advanced Message Queuing (AMQP). Afin de permettre au projet de faire face à un volume plus élevé de demandes, nous avons utilisé RabbitMQ pour traiter les messages de demande.

Logo de RabbitMQ

Plus d'informations sur RabbitMQ peuvent être trouvées ici : https://www.rabbitmq.com/

2.1.9. Kafka

Kafka peut fournir un mécanisme de file d'attente de messages pour améliorer la fiabilité des données de log.

Pour transmettre les données de log au réseau distribué Kafka afin de garantir la fiabilité, nous utilisons lensio/fast-data-dev, une configuration Kafka à part entière.

Logo de Kafka

Plus d'informations sur Kafka :https://kafka.apache.org/

2.1.10. Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs

Afin d'implémenter le déploiement de microservices pour ces deux projets, nous avons utilisé Docker dans le serveur pour déployer les microservices dans le conteneur Docker.

Logo de Docker

Plus d'informations sur Docker peuvent être trouvées ici : https://www.docker.com/

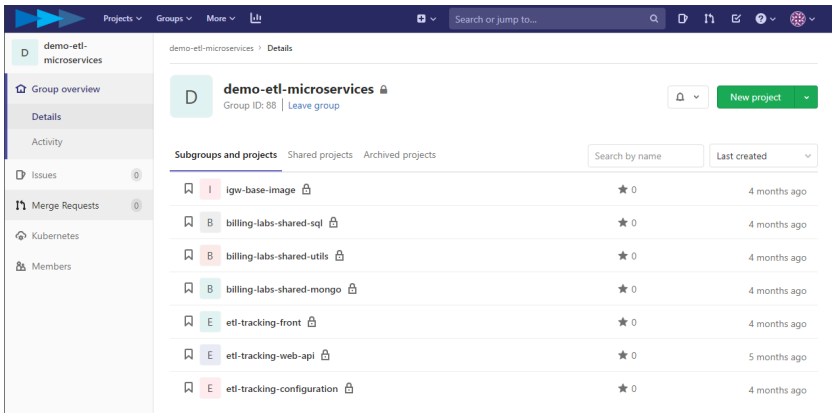
2.2. Outils de coopération

Bien que le projet de stage soit un projet personnel, , j'ai utilisé également de nombreux outils de projets collaboratifs dans l'entreprise, tels que GitLab, Teams, Trello et One Drive.

2.2.1. GitLab

GitLab est un service d'hébergement de référentiel Git basé sur le Web. Il offre toutes les fonctionnalités de contrôle des révisions distribuées et de gestion du code source (SCM) de Git ainsi que l'ajout de ses propres fonctionnalités. Contrairement à Git, qui est strictement un outil de ligne de commande, GitLab fournit une interface graphique Web.

Il fournit également un contrôle d'accès et plusieurs fonctionnalités de collaboration comme le suivi des bogues, les demandes de fonctionnalités, la gestion des tâches et les wikis pour chaque projet.



Les projets ETL dans le Git de SpikeeLabs

GitLab intègre également des fonctions telles que CI (Continuous Integration).

2.2.2. Teams

Microsoft Teams est une application de collaboration qui permet à l'équipe de rester organisée et d'avoir des conversations au même endroit.

- Équipes: rechercher des canaux pour en faire partie ou réserver notre propre canal. Dans les canaux, nous pouvons organiser une réunion immédiate, discuter et partager des fichiers.
- Réunions: afficher toutes les réunions prévues de la journée ou de la semaine. Ou bien, planifier une réunion. Ce calendrier se synchronise avec le calendrier Outlook.
- Appels: dans certains cas, si l'organisation a installé cette fonctionnalité, nous pouvons appeler qui nous voulons à partir de Teams, même si ces personnes ne l'utilisent pas.
- Activité: consulter tous les messages non lus, @mentions, réponses, etc.

2.2.3. Trello

Trello est un outil de gestion de projet gratuit en ligne. Inspiré par la méthode Kanban de Toyota, Trello repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement.

Trello permet ainsi de travailler sur des projets collaboratifs depuis n'importe où dans le monde, tout en étant notifié sur les différents apports réalisés par les membres.

2.2.4. One Drive

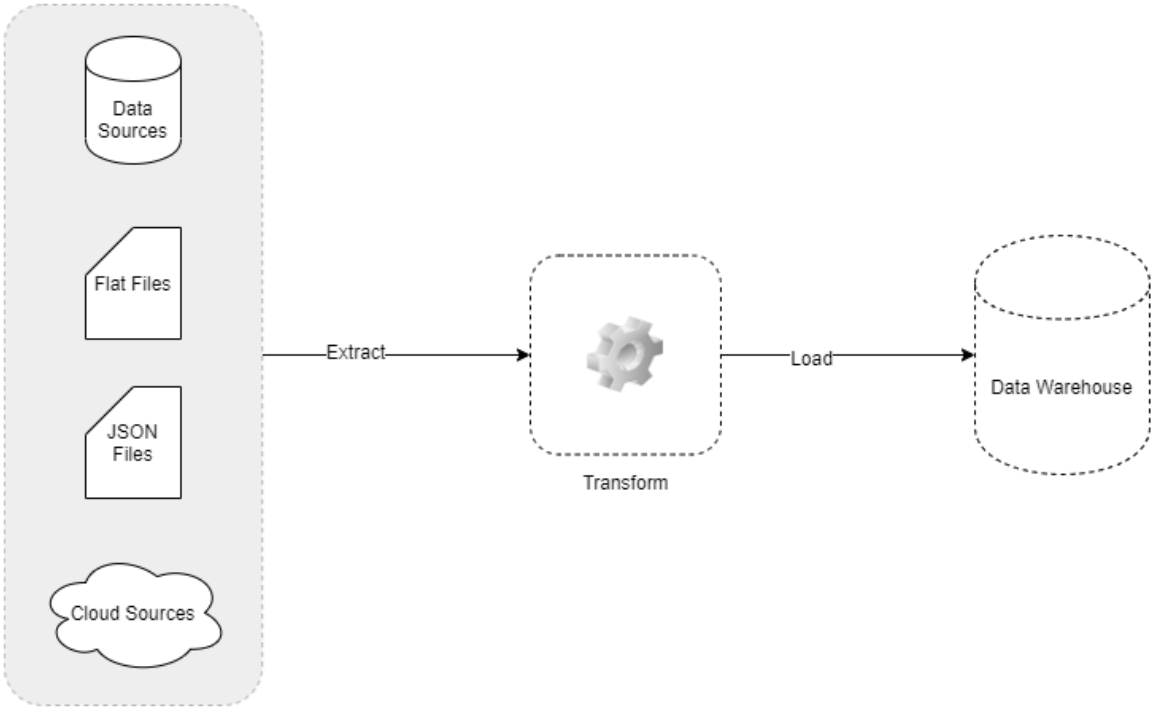
OneDrive est un service d'hébergement de fichiers qui permet aux utilisateurs de synchroniser des fichiers et d'y accéder ultérieurement à partir d'un navigateur Web ou d'un appareil mobile. Les utilisateurs peuvent partager des fichiers publiquement ou avec leurs contacts.

2.2.5. Gantt

3. Projet ETL

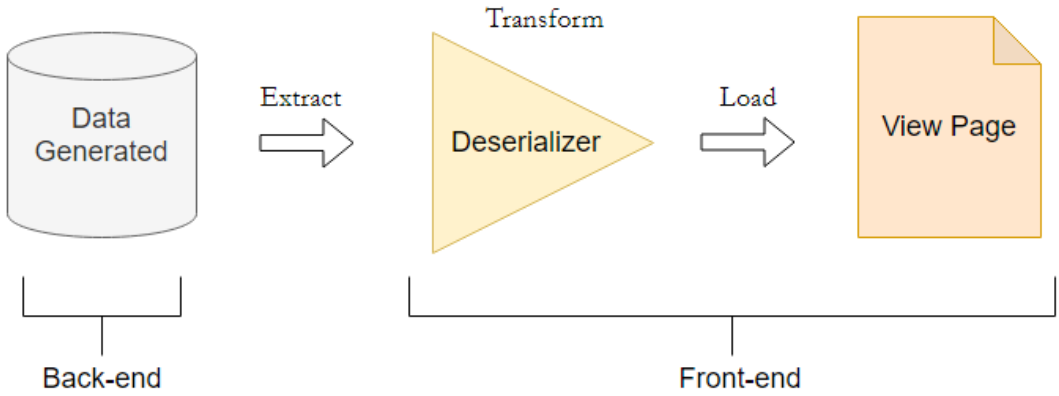
3.1. Organisation

ETL, l'abréviation de Extract-Transform-Load en anglais, est utilisé pour décrire le processus d'extraction, de transformation et de chargement de données de la source à la destination. Le terme ETL est plus couramment utilisé dans le stockage de données, mais son objet ne se limite pas au stockage de données.



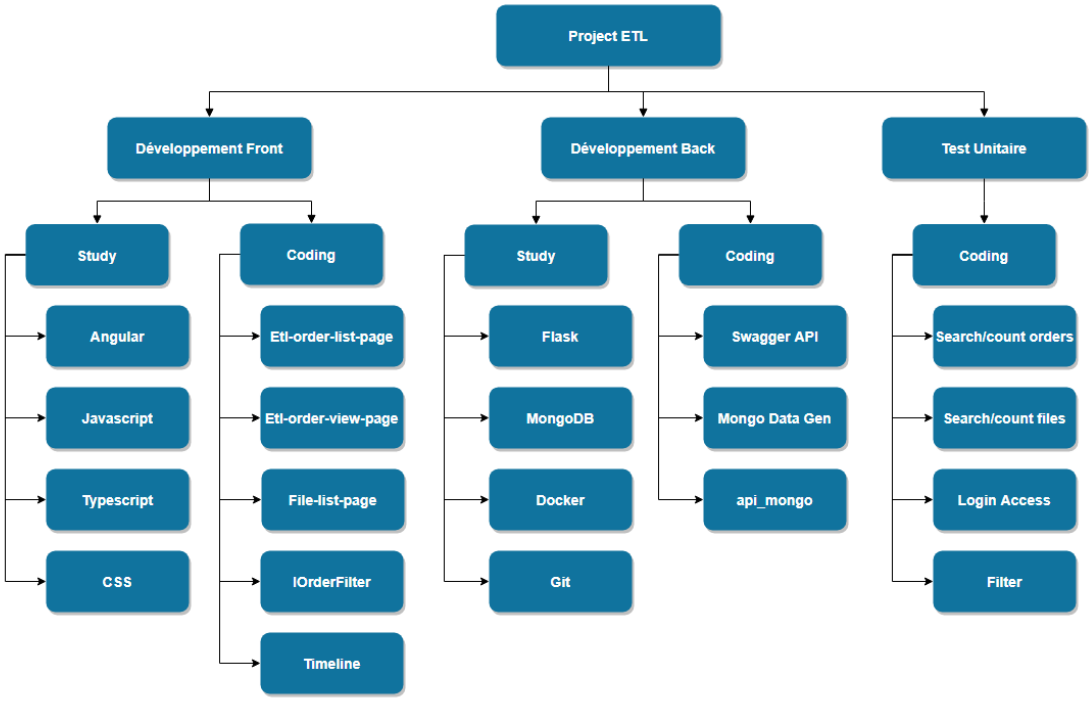
Les principes de modèle ETL

Dans notre projet, nous utilisons le modèle ETL pour nous aider à développer et intégrer front-end et back-end.



Le modèle ETL adapté

Nous divisons le projet ETL en les parties suivantes pour organiser l'exécution des tâches, afin de faciliter la compréhension et la gestion de l'ensemble de notre projet.

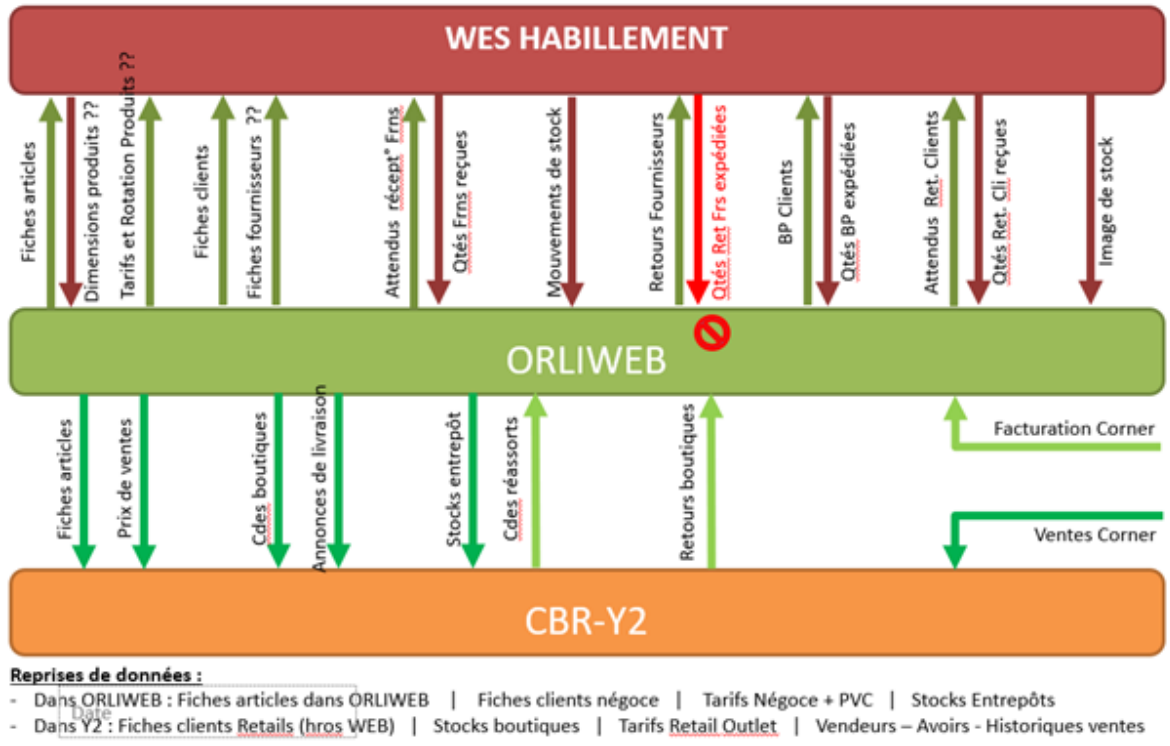


Les tâches détaillées de projet ETL

Le développement de chaque étape est divisé en deux étapes : l'apprentissage et le codage.

3.2. Génération des données fictives

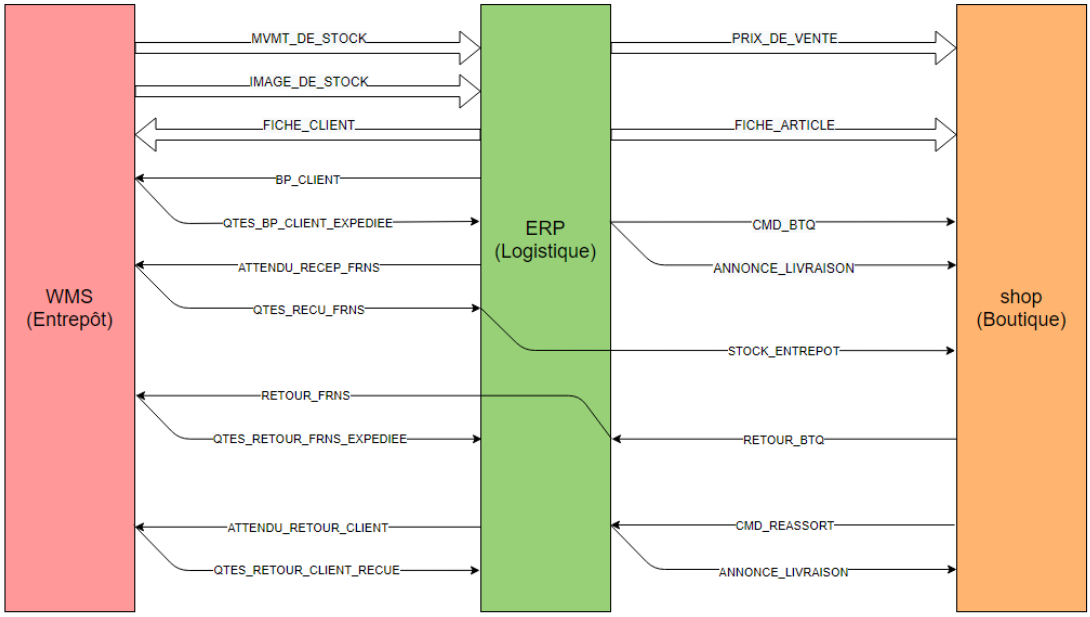
Voici un organigramme de notre projet de gestion logistique. Il s'agit notamment de magasin (CBR-Y2), de réseau logistique (ORLIWEB) et d'entrepôt (WES HABILLEMENT).



L'organigramme du projet ETL

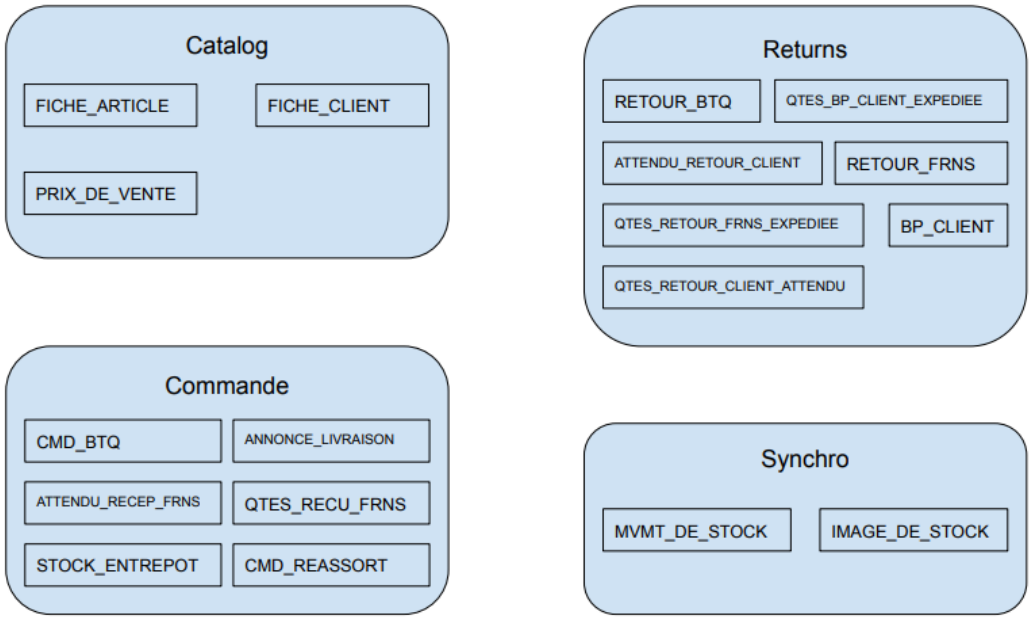
Nous pouvons voir qu'il existe de nombreux workflows ci-dessus, ce sont les données qui seront générées dans le travail réel. Mais comme notre projet est un projet de recherche, nous utiliserons plutôt des données virtuelles.

Selon cet organigramme, nous pouvons obtenir la relation entre différents flux de transmission. Nous résumons l'organigramme et obtenons le flux de commandes suivant.



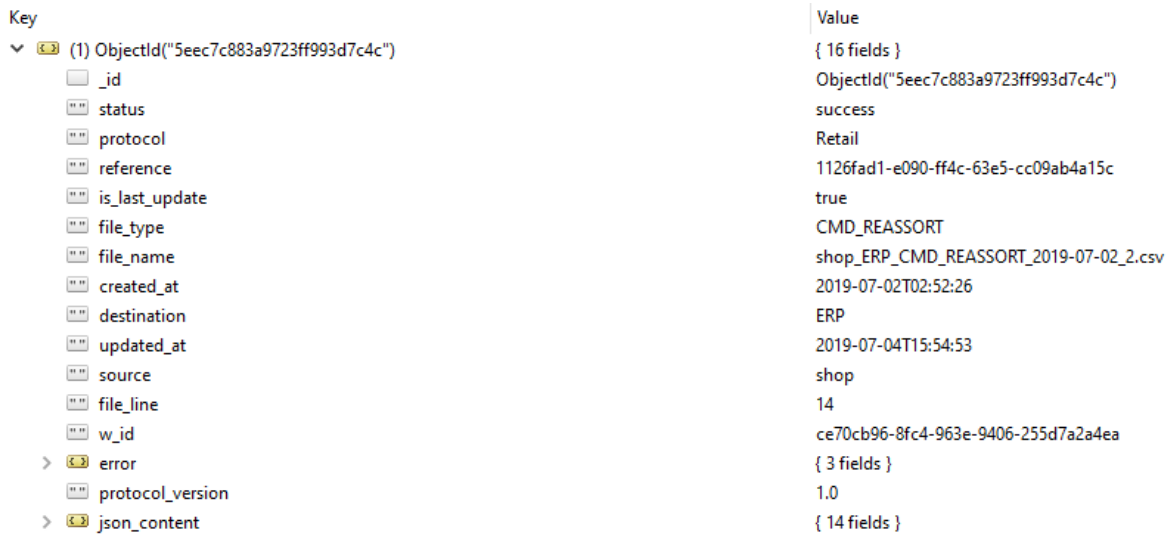
Les flux dans le système ETL

Nous pouvons diviser ces flux en quatre catégories, à savoir `Catalog`, `Commande`, `Returns` et `Synchro`.



Les quatre catégories de flux

Dans cette partie, nous avons terminé l'opération de génération de données de simulation. Nous avons spécifiquement utilisé Python comme langage pour générer des scripts et nous nous sommes connectés à la base de données MongoDB pour implémenter l'insertion de documents. Nous utilisons également JSON comme type de données pour réaliser une communication sans état entre les API.



Un exemple d'une commande générée dans MongoDB

3.3. Modifications des APIs Python

Le serveur principal de notre projet utilise le framework Flask et Swagger pour construire ce service. Notre ancien projet est un système typique de gestion de l'information logistique des télécommunications, donc notre travail consiste à modifier l'ancien projet pour l'adapter à notre nouveau type de données et système de type de service.

URI	Method	Result
/file	GET	Retrieve file list
	POST	Create/upload a file
/order	GET	Retrieve order list
/orders/{reference}/{operateur}	GET	Retrieve order view page for a given reference and operateur
/metrics/orders/{protocol}	GET	Retrieve all orders for a given protocol
/flux	GET	Retrieve order with the label of protocol

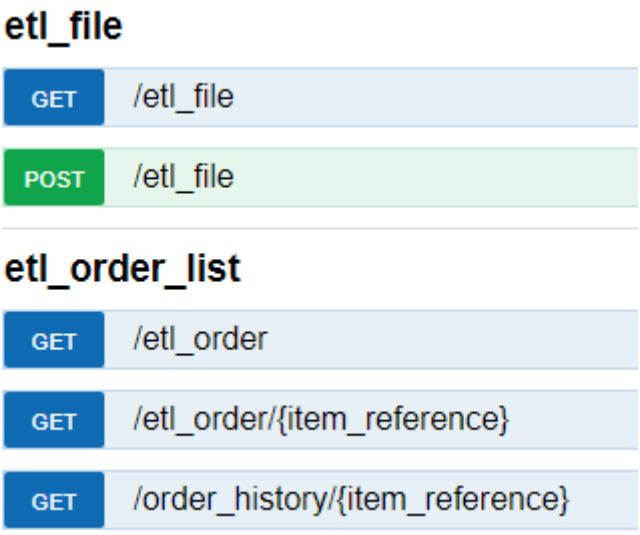
L'API du système d'origine

Nous avons défini un nouveau type de données pour notre projet et défini une nouvelle API basée sur l'API d'origine. Il comprend les opérations suivantes.

URI	Method	Result
/etl_file	GET	Retrieve etl file list
	POST	Create/upload etl file property
/etl_order	GET	Retrieve etl order list
/etl_order/{item_reference}	GET	Retrieve etl order view page with a given item_reference
/etl_history/{item_reference}	GET	Retrieve etl order history usage with a given item_reference

La conception d'API pour ETL

Sur la base du routeur défini ci-dessus, nous pourrons construire le fichier `swagger.yml` pour définir la documentation de l'API. Ensuite, nous utilisons swagger-codegen officiellement fourni par swagger pour générer automatiquement des classes d'API et des contrôleurs. Nous pouvons donc voir l'API suivante sur la page Web.



Swagger UI pour visualiser l'API

3.4. Angular pour l'Interface Homme-Machine

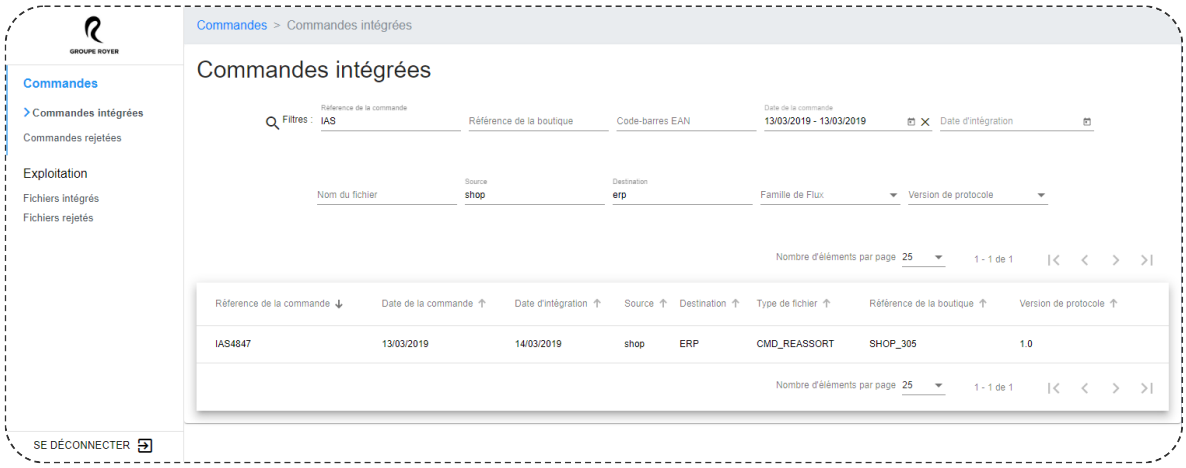
Angular est un framework côté client open source de Google, basé sur TypeScript. Nous avons également utilisé son interface utilisateur, Angular Material, pour développer une page front texturée.

Angular est utilisé pour faire beaucoup de conception fonctionnelle. Par exemple, nous pouvons ajouter une zone de recherche pour filtrer les résultats EAN, ou nous pouvons ajouter des informations supplémentaires sur le timeline.

On présente ci-dessous les filtres qu'on a créé. Selon les différents besoins, on peut filtrer la référence de la commande, la référence de la boutique, le code barre EAN, la date de la commande, la date d'intégration de la commande, le nom du fichier, la source de la commande, la destination de la commande, la catégorie de la commande et la version de protocole.

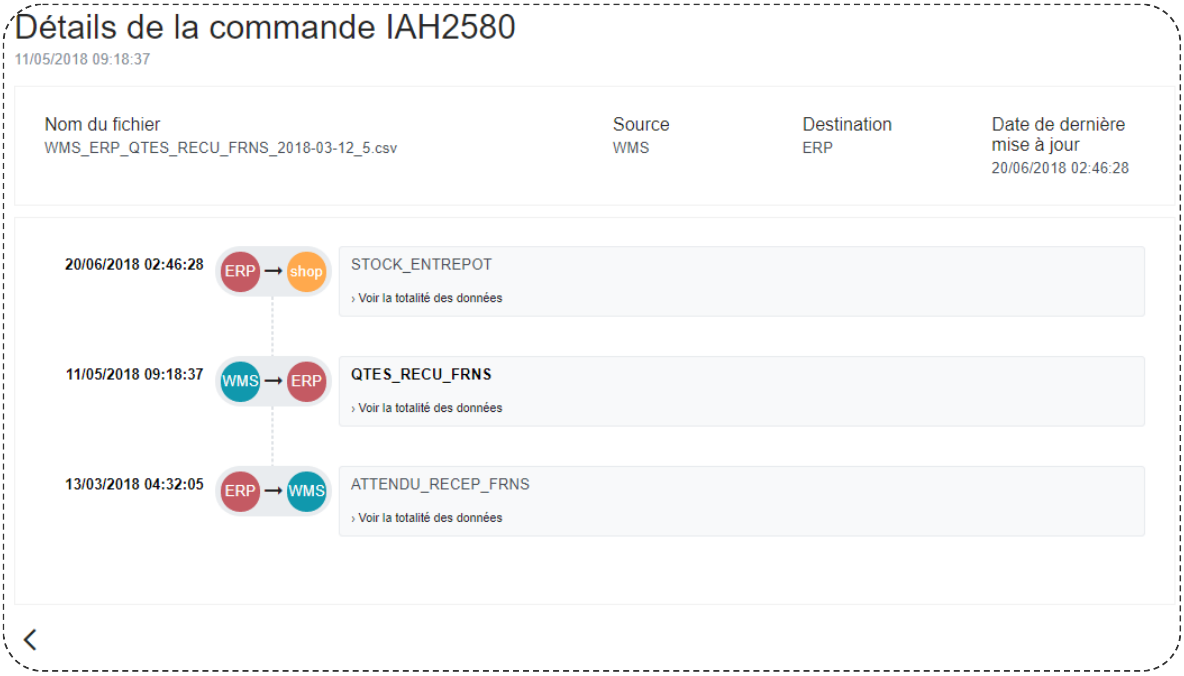
Les filtres pour filtrer les commandes

Le test d'une page peut être effectué. On précise que la référence de la commande commence par "IAS" et la commande passe de "boutique" à "ERP" (insensible à la casse). La date de la commande est fixée au 13/03/2019.



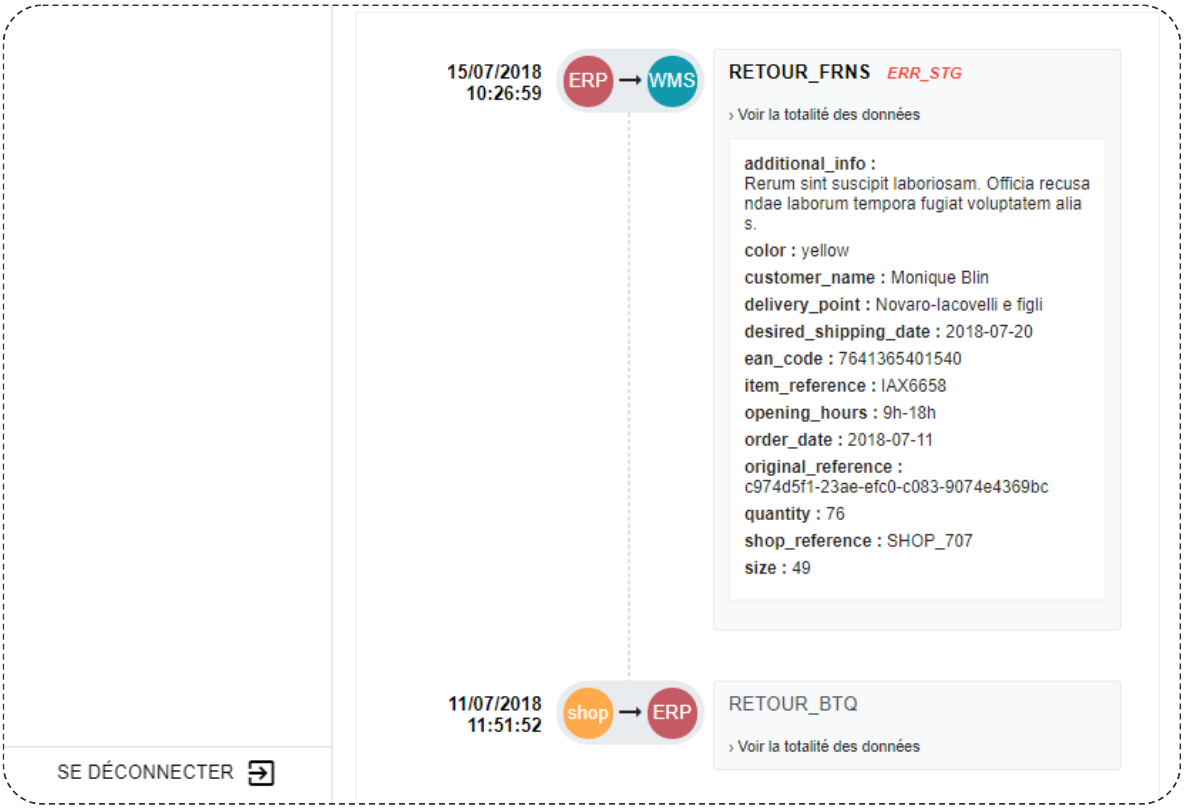
Les commandes filtrées

De plus, lorsque nous cliquons sur une page, celle-ci doit pouvoir afficher les informations détaillées de la commande et toutes les commandes qui y sont liées dans le même workflow. Ces commandes forment le timeline.



Le timeline est affiché dans la page détail

Lors de la génération de données, nous avons également généré certains types de données qui indiquent des erreurs pour identifier les problèmes pouvant survenir pendant le processus de commande, tels que des erreurs d'inventaire, des erreurs de transaction ou des erreurs de format.



Une commande erronée

3.5. Test Unitaire

Les tests unitaires sont utilisés pour détecter si le projet peut fonctionner normalement. C'est une pratique très importante pour le développement TDD (développement piloté par les tests).

Dans notre projet, nous testons l'API et le front-end. Parce que notre backend utilise le langage Python, nous avons choisi la bibliothèque unittest, une bibliothèque très couramment utilisée pour écrire des tests unitaires en Python, pour nous aider à implémenter le processus de test unitaire.

```
    # test count_files without param
    def test_count_files_none(self)

    # test count_files with param
    def test_count_files_param(self)

    # test search_files without param
    def test_search_files_none(self)

    # test search_files with param
    def test_search_files_param(self)

    # test count_orders without param
    def test_count_orders_none(self)

    # test count_orders with param
    def test_count_orders_param(self)

    # test search_orders without param
    def test_search_orders_none(self)

    # test search_orders with param
    def test_search_orders_param(self)
```

Les fonctions de test

Ce qui suit est le résultat de notre test unitaire, ce qui signifie que notre projet a réussi le test unitaire.

```
2020-07-09 16:40:15,112 - INFO - __main__ - test count_files without param
2020-07-09 16:40:15,112 - DEBUG - __main__ - count_documents [db=igw, collection=test_etl_demo_file, param_dict={}]
2020-07-09 16:40:15,156 - DEBUG - __main__ - result:3
2020-07-09 16:40:15,158 - INFO - __main__ - test count_files with param: {'item_reference': 'IAU4002', 'file_name': 'shop_ERP_RETOUR_BTQ_2018-12-20_5.csv'}
2020-07-09 16:40:15,158 - DEBUG - __main__ - count_documents [db=igw, collection=test_etl_demo_file, param_dict={'item_reference': 'IAU4002', 'file_name': 'shop_ERP_RETOUR_BTQ_2018-12-20_5.csv'}]
2020-07-09 16:40:15,161 - DEBUG - __main__ - result:1
2020-07-09 16:40:15,162 - INFO - __main__ - test count_orders without param
2020-07-09 16:40:15,162 - DEBUG - __main__ - count_documents [db=igw, collection=test_etl_demo_order, param_dict={}]
2020-07-09 16:40:15,165 - DEBUG - __main__ - result:2
2020-07-09 16:40:15,166 - INFO - __main__ - test count_orders with param: {'error.step_status': 'failed', 'json_content.shop_reference': 'SHOP_457'}
2020-07-09 16:40:15,166 - DEBUG - __main__ - count_documents [db=igw, collection=test_etl_demo_order, param_dict={'error.step_status': 'failed', 'json_content.shop_reference': 'SHOP_457'}]
2020-07-09 16:40:15,169 - DEBUG - __main__ - result:1
2020-07-09 16:40:15,170 - INFO - __main__ - test search_files without param
2020-07-09 16:40:15,171 - DEBUG - __main__ - get_documents [db=igw, collection=test_etl_demo_file, param_dict={}, limit=0, offset=0, sorts_list=None, projection=None]
2020-07-09 16:40:15,175 - INFO - __main__ - test search_files with param: {'item_reference': 'IAU4002', 'file_name': 'shop_ERP_RETOUR_BTQ_2018-12-20_5.csv'}
2020-07-09 16:40:15,175 - DEBUG - __main__ - get_documents [db=igw, collection=test_etl_demo_file, param_dict={'item_reference': 'IAU4002', 'file_name': 'shop_ERP_RETOUR_BTQ_2018-12-20_5.csv'}, limit=0, offset=0,
sorts_list=None, projection=None]
2020-07-09 16:40:15,179 - INFO - __main__ - test search_orders without param
2020-07-09 16:40:15,179 - DEBUG - __main__ - get_documents [db=igw, collection=test_etl_demo_order, param_dict={}, limit=0, offset=0, sorts_list=None, projection=None]
2020-07-09 16:40:15,185 - INFO - __main__ - test search_orders with param: {'updated_at': {'$gt': '2019-01-01T22:22:22'}, 'json_content.item_reference': {'$regex': 'IAK'}}
2020-07-09 16:40:15,185 - DEBUG - __main__ - get_documents [db=igw, collection=test_etl_demo_order, param_dict={'updated_at': {'$gt': '2019-01-01T22:22:22'}, 'json_content.item_reference': {'$regex': 'IAK'}}, limit=0,
offset=0, sorts_list=None, projection=None]

----------------------------------------------------------------------
Ran 8 tests in 0.078s

OK
```

L'affichage dans le console des tests unitaires

3.6. Déploiement en Docker

Après avoir terminé la conception front-end et back-end, notre projet peut s'exécuter normalement en local de VM.

```
start@dev-zpa:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                                       NAMES
259daac7f592        etl-tracking-front:develop     "nginx -g 'daemon of…"   7 days ago          Up 7 days           80/tcp, 0.0.0.0:8010->8000/tcp              etl-tracking-front
7a4a78156978        etl-tracking-web-api:develop   "bash /spikeelabs/ru…"   7 days ago          Up 7 days           0.0.0.0:8011->8000/tcp                      etl-tracking-web-api
f3d06b709cd5        mongo:latest                   "docker-entrypoint.s…"   3 months ago        Up 3 months         0.0.0.0:27017->27017/tcp                    keen_ganguly
```

Les services sont stockées dans des conteneurs docker

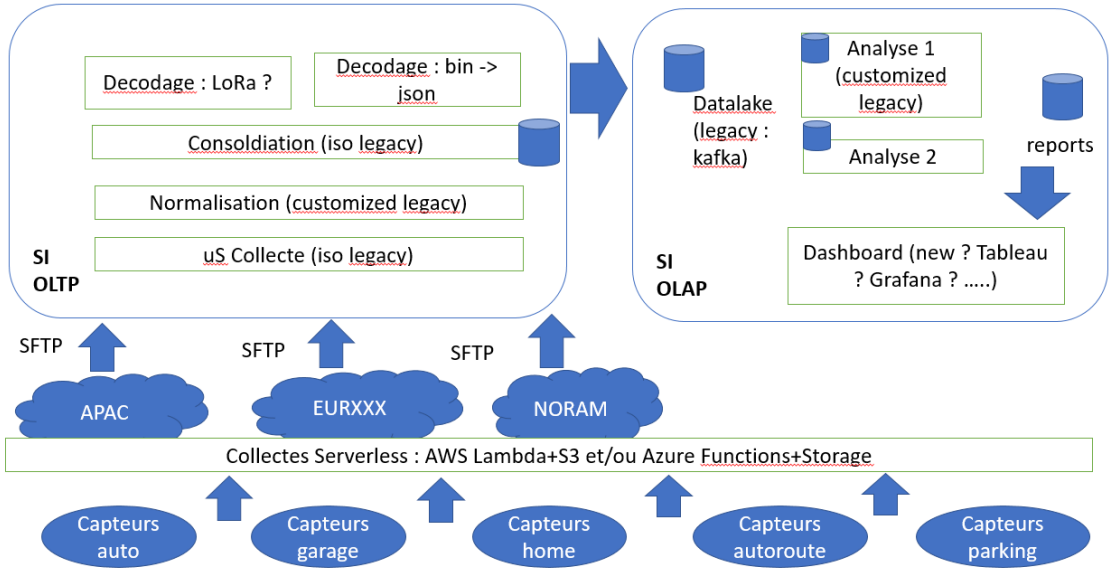
Mais si nous voulons déployer le projet afin que le projet puisse toujours fournir des services et être plus facilement accessible, nous devons utiliser Docker.

4. Projet IoT

4.1. Organisation

Dans notre projet IoT, nous commencerons par l'étape uS Collecte (nous utilisons les données générées à la place) .

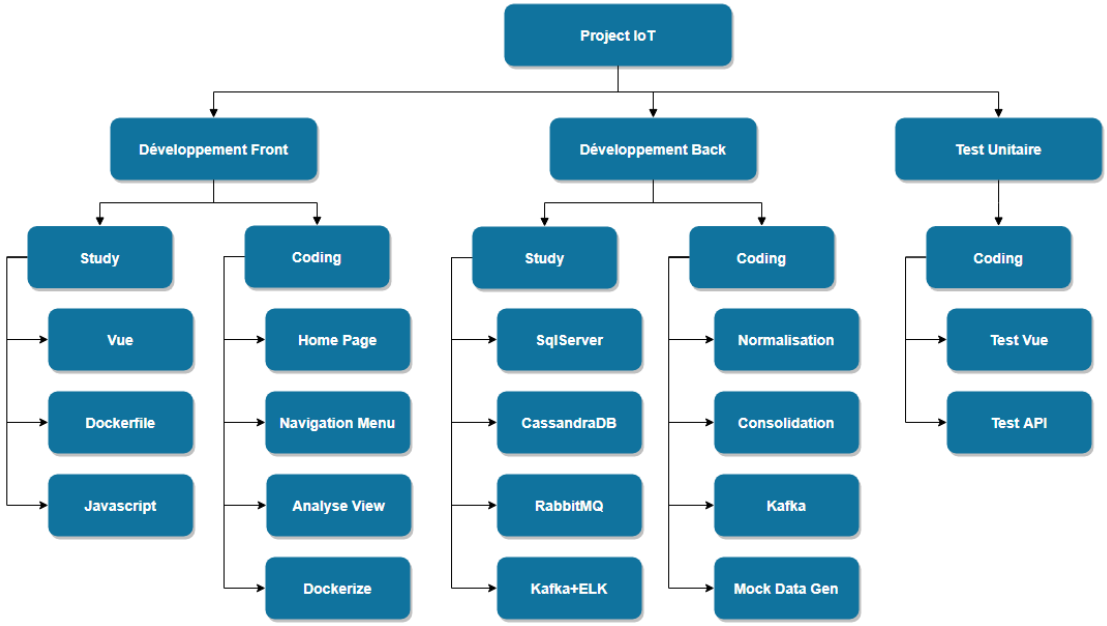
Les données entrantes dans différents formats seront intégrées en normaliastion dans un format unifié et stockées dans Cassandra.



Le processus global du projet

Ensuite, nous faisons correspondre les données normalisées dans Cassandra avec les données utilisateur stockées dans SQL Server, et sortons les données qui répondent aux conditions utilisateur dans le lac de données de Kafka.

Enfin, nous sortons les données de la file d'attente de messages Kafka, effectuons certains traitements de Big Data et les affichons sur le dashboard.

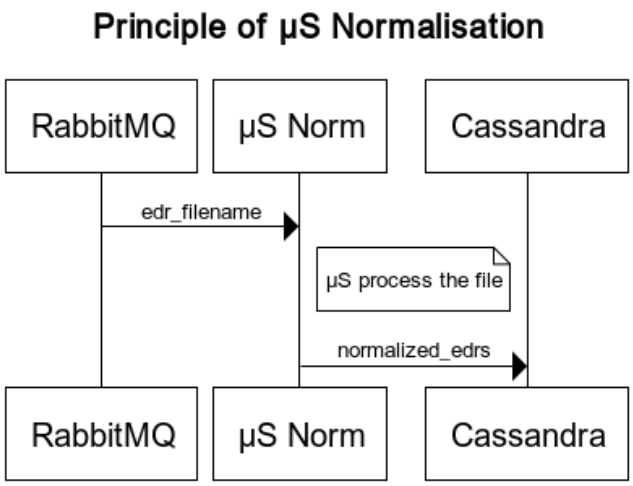


Les tâches détaillées de projet IOT

Tout comme ce que nous avons fait dans le projet ETL, nous pouvons diviser le projet IoT en quelques petites tâches.

4.2. Normalisation

À l'étape Normalisation, ce que nous devons faire est de traiter les données brutes collectées, afin que nous puissions obtenir les données de trois formats de fichiers différents (JSON, CSV et XML) à un format unifié.



Le principe du microservice normalisation

Les données normalisées seront transférées à l'étape suivante, la consolidation.

Voici comment la même donnée est représentée dans trois formats différents.

```
DeviceId,ArrivalTime,DepartureTime
"18,773",04/11/2017 07:24:35 AM,04/11/2017 07:30:00 AM
```

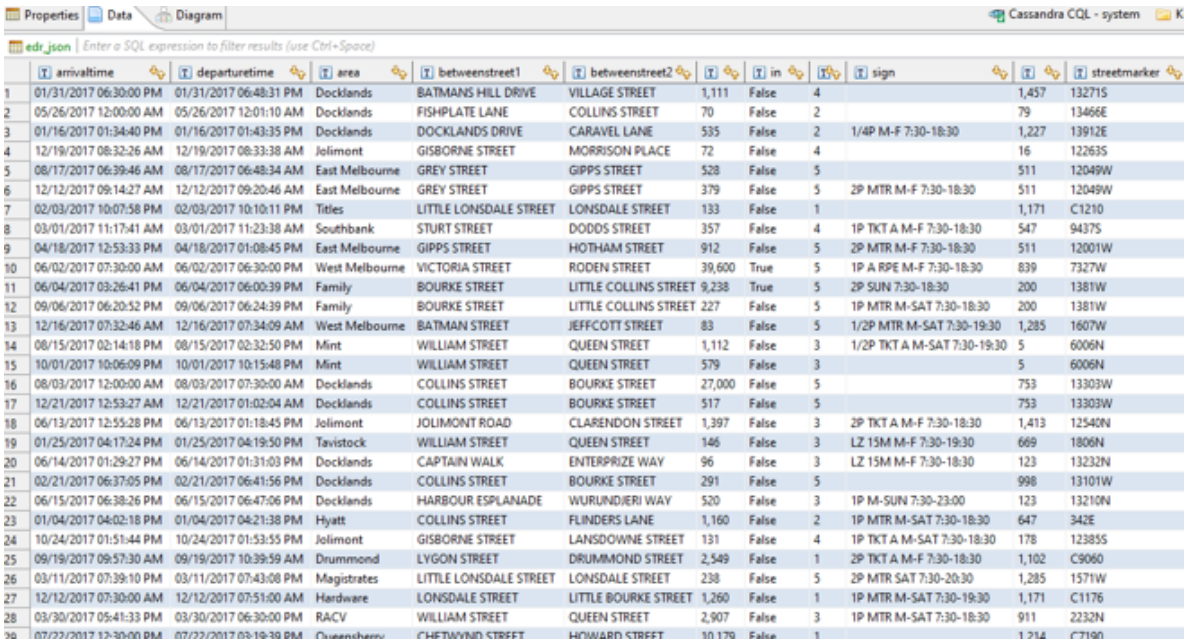
```
{
        "DeviceId": "18,773",
        "ArrivalTime": "04/11/2017 07:24:35 AM",
        "DepartureTime": "04/11/2017 07:30:00 AM",
}
```

```
<row>
    <deviceid>18773</deviceid>
    <arrivaltime>2017-04-11T07:24:35</arrivaltime>
    <departuretime>2017-04-11T07:30:00</departuretime>
</row>
```

Une exemple des trois formats de données : CSV, JSON, XML

Comme données de simulation, nous avons utilisé *On-street Car Parking Sensor Data-2017*, les données de détection des places de stationnement de la ville de Melbourne dans la zone CBD en 2017.

Il enregistre l'ID du capteur, l'heure d'arrivée de la voiture, l'heure de départ de la voiture, la durée du séjour, les panneaux de signalisation, l'emplacement géographique et d'autres données.



	arrivaltime	departuretime	area	betweenstreet1	betweenstreet2		in		sign		streetmarker
1	01/31/2017 06:30:00 PM	01/31/2017 06:48:31 PM	Docklands	BATMANS HILL DRIVE	VILLAGE STREET	1,111	False	4		1,457	13271S
2	05/26/2017 12:00:00 AM	05/26/2017 12:01:10 AM	Docklands	FISHPLATE LANE	COLLINS STREET	70	False	2		79	13466E
3	01/16/2017 01:34:40 PM	01/16/2017 01:43:35 PM	Docklands	DOCKLANDS DRIVE	CARAVEL LANE	535	False	2	1/4P M-F 7:30-18:30	1,227	13912E
4	12/19/2017 08:32:26 AM	12/19/2017 08:33:38 AM	Jolimont	GISBORNE STREET	MORRISON PLACE	72	False	4		16	12263S
5	08/17/2017 06:39:46 AM	08/17/2017 06:48:34 AM	East Melbourne	GREY STREET	GIPPS STREET	528	False	5		511	12049W
6	12/12/2017 09:14:27 AM	12/12/2017 09:20:46 AM	East Melbourne	GREY STREET	GIPPS STREET	379	False	5	2P MTR M-F 7:30-18:30	511	12049W
7	02/03/2017 10:07:58 PM	02/03/2017 10:10:11 PM	Titles	LITTLE LONSDALE STREET	LONSDALE STREET	133	False	1		1,171	C1210
8	03/01/2017 11:17:41 AM	03/01/2017 11:23:38 AM	Southbank	STURT STREET	DODDS STREET	357	False	4	1P TKT A M-F 7:30-18:30	547	9437S
9	04/18/2017 12:53:33 PM	04/18/2017 01:08:45 PM	East Melbourne	GIPPS STREET	HOTHAM STREET	912	False	5	2P MTR M-F 7:30-18:30	511	12001W
10	06/02/2017 07:30:00 AM	06/02/2017 06:30:00 PM	West Melbourne	VICTORIA STREET	RODEN STREET	39,600	True	5	1P A RPE M-F 7:30-18:30	839	7327W
11	06/04/2017 03:26:41 PM	06/04/2017 06:00:39 PM	Family	BOURKE STREET	LITTLE COLLINS STREET	9,238	True	5	2P SUN 7:30-18:30	200	1381W
12	09/06/2017 06:20:52 PM	09/06/2017 06:24:39 PM	Family	BOURKE STREET	LITTLE COLLINS STREET	227	False	5	1P MTR M-SAT 7:30-18:30	200	1381W
13	12/16/2017 07:32:46 AM	12/16/2017 07:34:09 AM	West Melbourne	BATMAN STREET	JEFFCOTT STREET	83	False	5	1/2P MTR M-SAT 7:30-19:30	1,285	1607W
14	08/15/2017 02:14:18 PM	08/15/2017 02:32:50 PM	Mint	WILLIAM STREET	QUEEN STREET	1,112	False	3	1/2P TKT A M-SAT 7:30-19:30	5	6006N
15	10/01/2017 10:06:09 PM	10/01/2017 10:15:48 PM	Mint	WILLIAM STREET	QUEEN STREET	579	False	3		5	6006N
16	08/03/2017 12:00:00 AM	08/03/2017 07:30:00 AM	Docklands	COLLINS STREET	BOURKE STREET	27,000	False	5		753	13303W
17	12/21/2017 12:53:27 AM	12/21/2017 01:02:04 AM	Docklands	COLLINS STREET	BOURKE STREET	517	False	5		753	13303W
18	06/13/2017 12:55:28 PM	06/13/2017 01:18:45 PM	Jolimont	JOLIMONT ROAD	CLARENDON STREET	1,397	False	3	2P TKT A M-F 7:30-18:30	1,413	12540N
19	01/25/2017 04:17:24 PM	01/25/2017 04:19:50 PM	Tavistock	WILLIAM STREET	QUEEN STREET	146	False	3	LZ 15M M-F 7:30-19:30	669	1806N
20	06/14/2017 01:29:27 PM	06/14/2017 01:31:03 PM	Docklands	CAPTAIN WALK	ENTERPRIZE WAY	96	False	3	LZ 15M M-F 7:30-18:30	123	13232N
21	02/21/2017 06:37:05 PM	02/21/2017 06:41:56 PM	Docklands	COLLINS STREET	BOURKE STREET	291	False	5		998	13101W
22	06/15/2017 06:38:26 PM	06/15/2017 06:47:06 PM	Docklands	HARBOUR ESPLANADE	WURUNDJERI WAY	520	False	3	1P M-SUN 7:30-23:00	123	13210N
23	01/04/2017 04:02:18 PM	01/04/2017 04:21:38 PM	Hyatt	COLLINS STREET	FLINDERS LANE	1,160	False	2	1P MTR M-SAT 7:30-18:30	647	342E
24	10/24/2017 01:51:44 PM	10/24/2017 01:53:55 PM	Jolimont	GISBORNE STREET	LANSDOWNE STREET	131	False	4	1P TKT A M-SAT 7:30-18:30	178	12385S
25	09/19/2017 09:57:30 AM	09/19/2017 10:39:59 AM	Drummond	LYGON STREET	DRUMMOND STREET	2,549	False	1	2P TKT A M-F 7:30-18:30	1,102	C9060
26	03/11/2017 07:39:10 PM	03/11/2017 07:43:08 PM	Magistrates	LITTLE LONSDALE STREET	LONSDALE STREET	238	False	5	2P MTR SAT 7:30-20:30	1,285	1571W
27	12/12/2017 07:30:00 AM	12/12/2017 07:51:00 AM	Hardware	LONSDALE STREET	LITTLE BOURKE STREET	1,260	False	1	1P MTR M-SAT 7:30-19:30	1,171	C1176
28	03/30/2017 05:41:33 PM	03/30/2017 06:30:00 PM	RACV	WILLIAM STREET	QUEEN STREET	2,907	False	3	1P MTR M-SAT 7:30-18:30	911	2232N

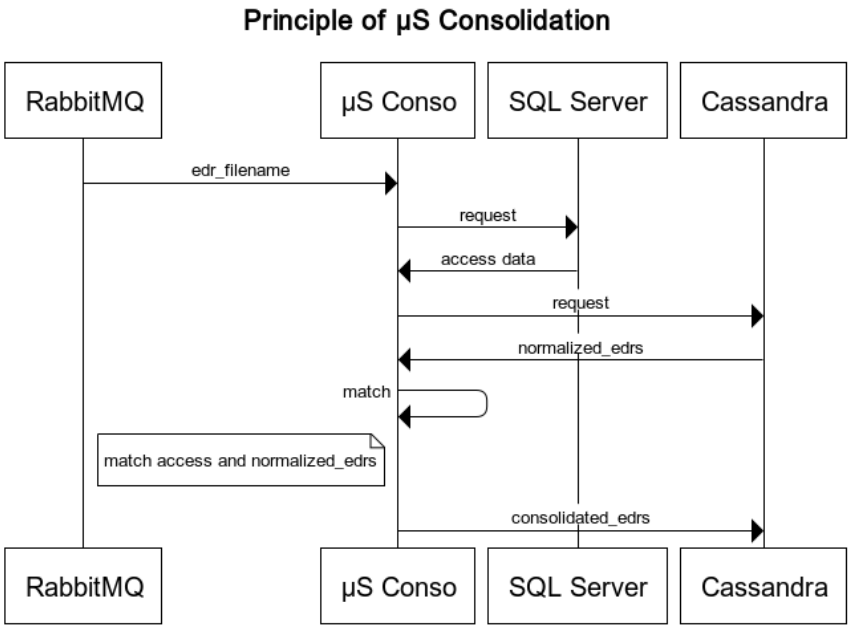
Les données normalisées sont stockées dans Cassandra

Les données normalisées stockées dans la base de données Cassandra après normalisation sont présentées ci-dessus.

4.3. Consolidation

Cette étape consiste à mapper toutes les données normalisées avec les données relatives à l'utilisateur dans SQL Server.

Selon la relation entre RabbitMQ, SQL Server et Cassandra et la relation entre les données utilisateur et les données normalisées, nous avons la conception suivante.



Le principe du microservice condolidation

On obtient le nom de fichier qui doit être normalisé en surveillant RabbitMQ. À ce stade, la normalisation traitera le fichier et stockera les données normalisées dans Cassandra.

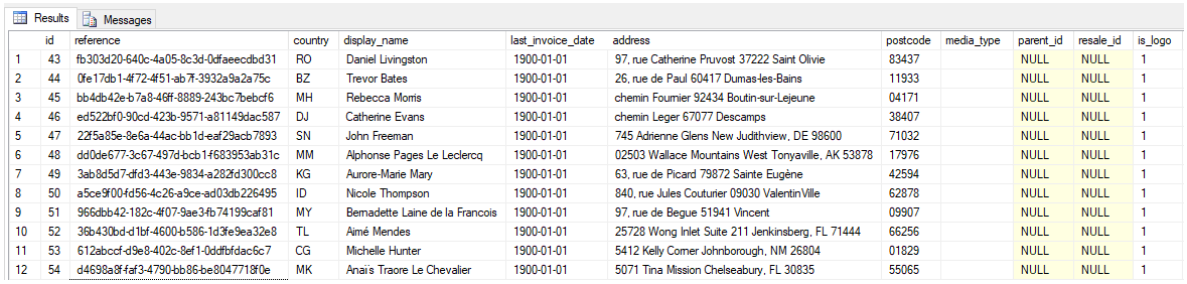
Ensuite, on envoie une demande à SQL Server pour obtenir des données d'accès utilisateur dans SQL Server. On envoie également une demande à Cassandra pour obtenir les données normalisées.

Le microservice consolidation peut faire correspondre les données d'accès de l'utilisateur avec les données normalisées et stocker les données qui répondent aux exigences de consolidation dans Cassandra.

Afin de simuler le processus ci-dessus, on génère des données fictives pour utilisateur et les services. Cela est lié aux cinq tables de SQL Server, elles sont `dbo.account`, `dbo.calling_access`, `dbo.invoice`, `dbo.billed_invoice_status` et `dbo.subscription`.

Le console indiquant les cinq tables générées

La figure suivante montre que les données générées ont été correctement stockées dans SQL Server.



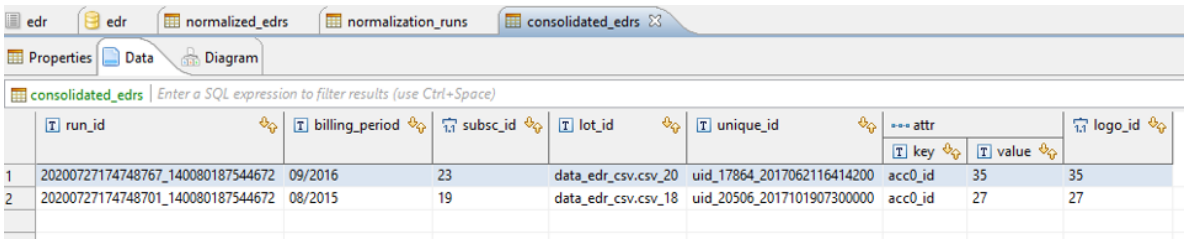
Results | Messages

	id	reference	country	display_name	last_invoice_date	address	postcode	media_type	parent_id	resale_id	is_logo
1	43	fb303d20-640c-4a05-8c3d-0dfaeecdbd31	RO	Daniel Livingston	1900-01-01	97, rue Catherine Pruvost 37222 Saint Olivie	83437		NULL	NULL	1
2	44	0fe17db1-4f72-4f51-ab7f-3932a9a2a75c	BZ	Trevor Bates	1900-01-01	26, rue de Paul 60417 Dumas-les-Bains	11933		NULL	NULL	1
3	45	bb4db42e-b7a8-46ff-8889-243bc7bebcf6	MH	Rebecca Morris	1900-01-01	chemin Fournier 92434 Boutin-sur-Lejeune	04171		NULL	NULL	1
4	46	ed522bf0-90cd-423b-9571-a81149dac587	DJ	Catherine Evans	1900-01-01	chemin Leger 67077 Descamps	38407		NULL	NULL	1
5	47	22f5a85e-8e6a-44ac-bb1d-eaf29acb7893	SN	John Freeman	1900-01-01	745 Adrienne Glens New Judithview, DE 98600	71032		NULL	NULL	1
6	48	dd0de677-3c67-497d-bcb1-f683953ab31c	MM	Alphonse Pages Le Leclercq	1900-01-01	02503 Wallace Mountains West Tonyaville, AK 53878	17976		NULL	NULL	1
7	49	3ab8d5d7-dfd3-443e-9834-a282fd300cc8	KG	Aurore-Marie Mary	1900-01-01	63, rue de Picard 79872 Sainte Eugène	42594		NULL	NULL	1
8	50	a5ce9f00-fd56-4c26-a9ce-ad03db226495	ID	Nicole Thompson	1900-01-01	840, rue Jules Couturier 09030 ValentinVille	62878		NULL	NULL	1
9	51	966dbb42-182c-4f07-9ae3-fb74199caf81	MY	Bernadette Laine de la Francois	1900-01-01	97, rue de Begue 51941 Vincent	09907		NULL	NULL	1
10	52	36b430bd-d1bf-4600-b586-1d3fe9ea32e8	TL	Aimé Mendes	1900-01-01	25728 Wong Inlet Suite 211 Jenkinsberg, FL 71444	66256		NULL	NULL	1
11	53	612abccf-d9e8-402c-8ef1-0ddfbfdac6c7	CG	Michelle Hunter	1900-01-01	5412 Kelly Corner Johnborough, NM 26804	01829		NULL	NULL	1
12	54	d4698a8f-faf3-4790-bb86-be8047718f0e	MK	Anais Traore Le Chevalier	1900-01-01	5071 Tina Mission Chelseabury, FL 30835	55065		NULL	NULL	1

Les données de table dbo.account dans SQL Server

On compare la référence des données en noramlisation avec la référence des données générées dans SQL Server pour confirmer si les deux données correspondent.

Si elles correspondent, les données sont intégrées dans un nouveau type de données `edr_consolidated` et enregistrées dans Cassandra.



edr | edr | normalized_edrs | normalization_runs | consolidated_edrs

Properties | Data | Diagram

consolidated_edrs | Enter a SQL expression to filter results (use Ctrl+Space)

	run_id	billing_period	subsc_id	lot_id	unique_id	attr		logo_id
						key	value	
1	20200727174748767_140080187544672	09/2016	23	data_edr_csv.csv_20	uid_17864_2017062116414200	acc0_id	35	35
2	20200727174748701_140080187544672	08/2015	19	data_edr_csv.csv_18	uid_20506_2017101907300000	acc0_id	27	27

Les données consolidées sont stockées dans Cassandra

On peut voir dans la figure ci-dessus que nos données consolidées ont été affichées avec succès dans Cassandra.

4.4. Kafka

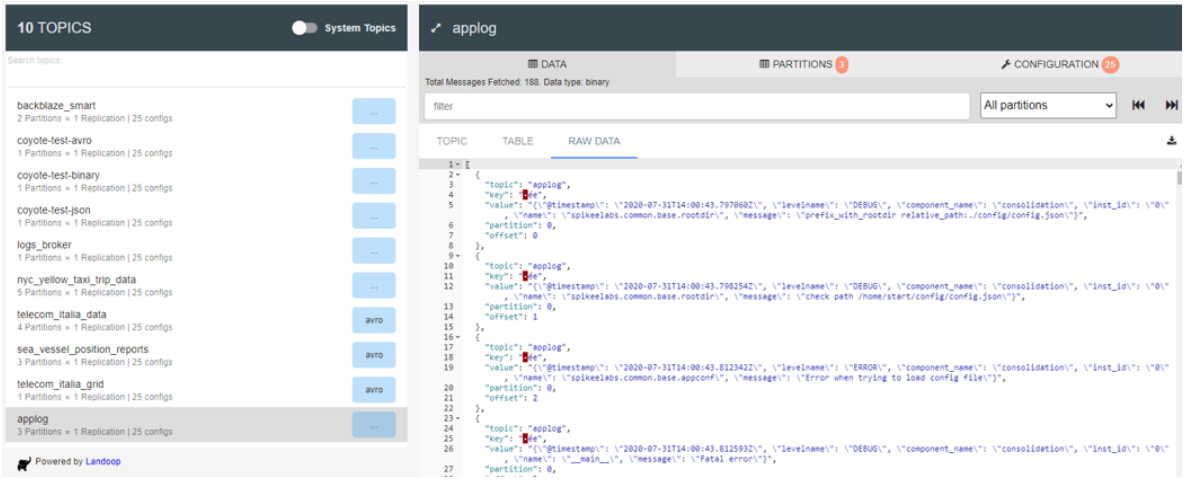
En tant que niveau de message distribué, kafka peut effectivement aider à améliorer la robustesse de la propagation des messages du système.

Afin de surveiller le log dans la phase de consolidation, nous avons introduit `KafkaHandler` dans le projet pour obtenir les données de la sortie du log par la consolidation. Pour ce faire, nous réécrivons le fichier `applog.py` dans `billing-labs-shared-utils` partagé par spikeelabs comme suit.

```
def configure(component_name, component_instance_id=None, is_threaded=False,
requestid_formatter=None):
    # ... code ... #
    mylogger = logging.getLogger()
    kh = KafkaHandler()
    formatter = CustomJsonFormatter('(@timestamp) (levelname) (component_name)
(inst_id) (name) (message)')
    kh.setFormatter(formatter)
    mylogger.addHandler(kh)
    # ... code ... #
```

Afin de surveiller le log de la phase de consolidation, nous avons introduit `KafkaHandler` dans le projet pour obtenir des données à partir de la sortie de la consolidation.

Grâce à fast-data-dev, nous pouvons facilement obtenir une interface utilisateur sur `http://localhost:3030`. Les données obtenues en modifiant le `applog.py` et en surveillant le log peuvent être affichées comme suit.



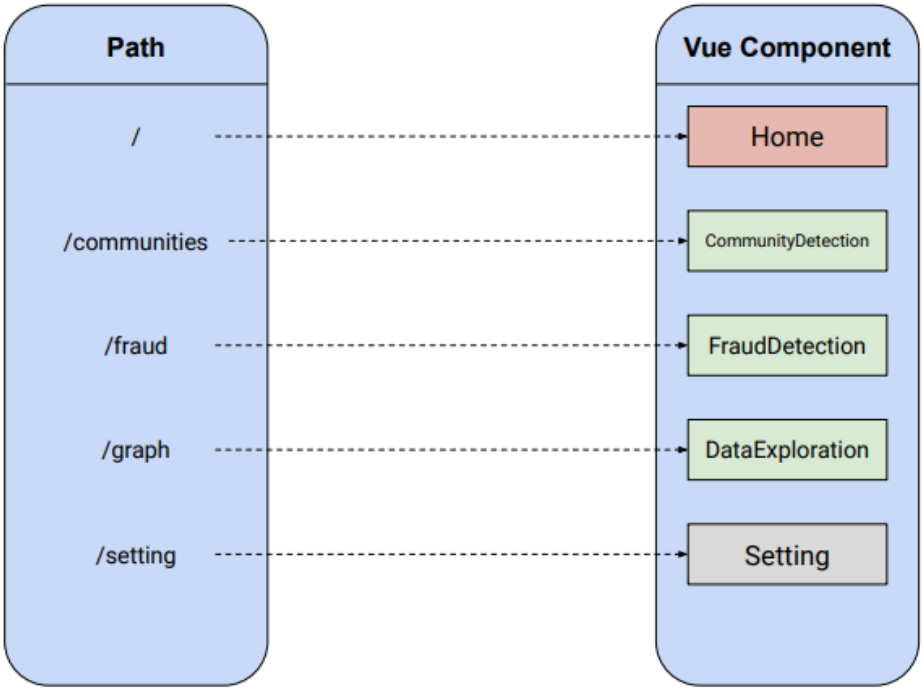
Ces données peuvent être enregistrées dans plusieurs sauvegardes dans Kafka pour assurer la redondance des données.

4.5. Dashboard

Un autre stagiaire Big Data et moi avons construit cette partie de démonstrateur de données.

Afin de réaliser un développement rapide et d'obtenir un bon effet d'affichage, on utilise le framework Vue pour réaliser le développement et utilise Element UI pour améloirer l'affichage. Les pages développées avec Element UI peuvent s'adapter à l'esthétique moderne, et il existe de nombreux composants pour Element UI.

Selon nos exigences en matière d'API back-end, on doit fournir des interfaces côté Web pour les trois processus BigData / MachinLearning. On a donc conçu le routage suivant.



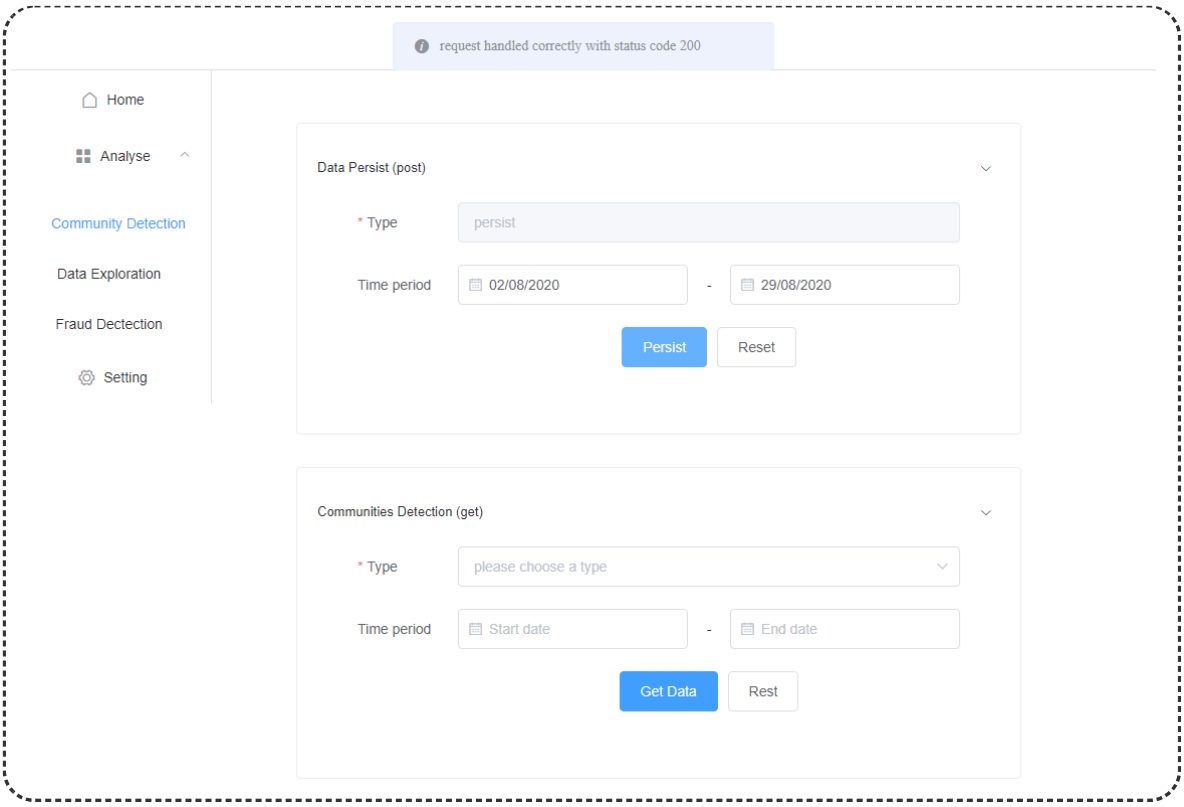
La conception de routeur en Vue

Afin de réaliser la communication entre la page front-end et la file d'attente de messages de Kafka, nous utilisons Axios pour partage des ressources entre origines multiples.

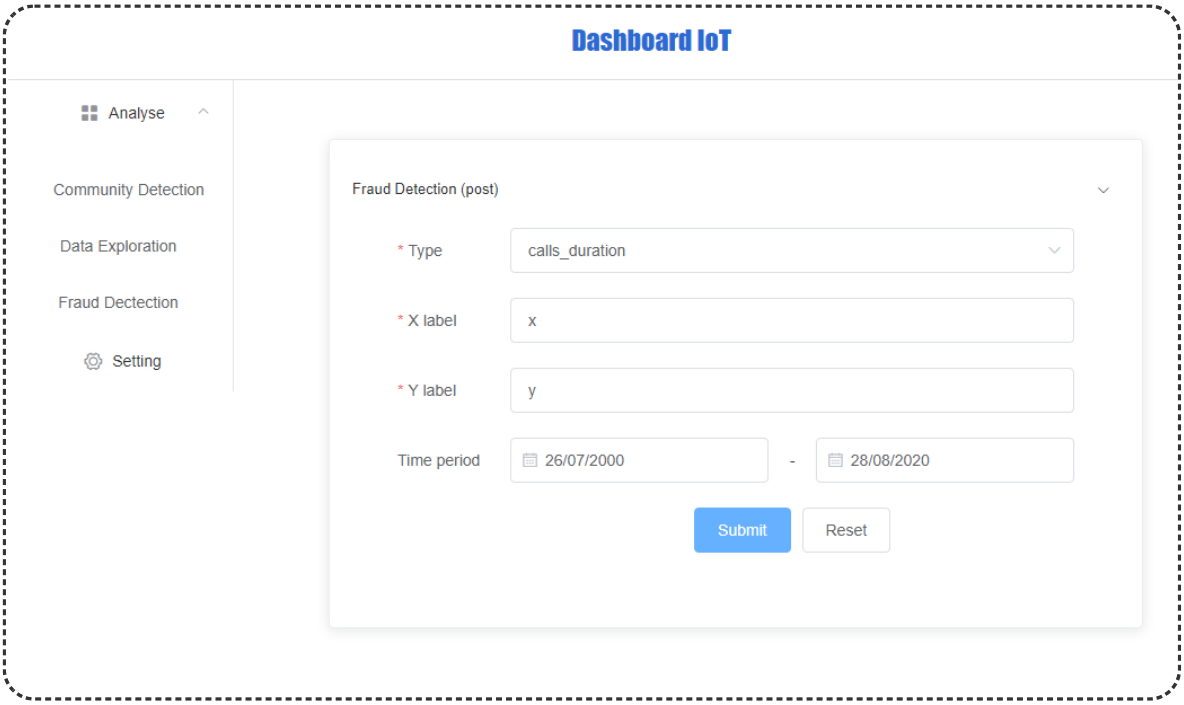
```
axios(config).then((response) => {
    this.msg = response.data['Message'] + ' with status code ' +
response.data['Status Code'];
    console.log(JSON.stringify(response.data));
}).catch((error)=>{
    this.msg = error;
    console.log(error);
});
```

Cela nous permet de faire des demandes de ressources interdomaines et de renvoyer un objet Promise en JavaScript pour indiquer que la ressource que nous avons demandée sera renvoyée de manière asynchrone.

Après avoir obtenu les données de Kafka, nous pouvons tester sur la page front.

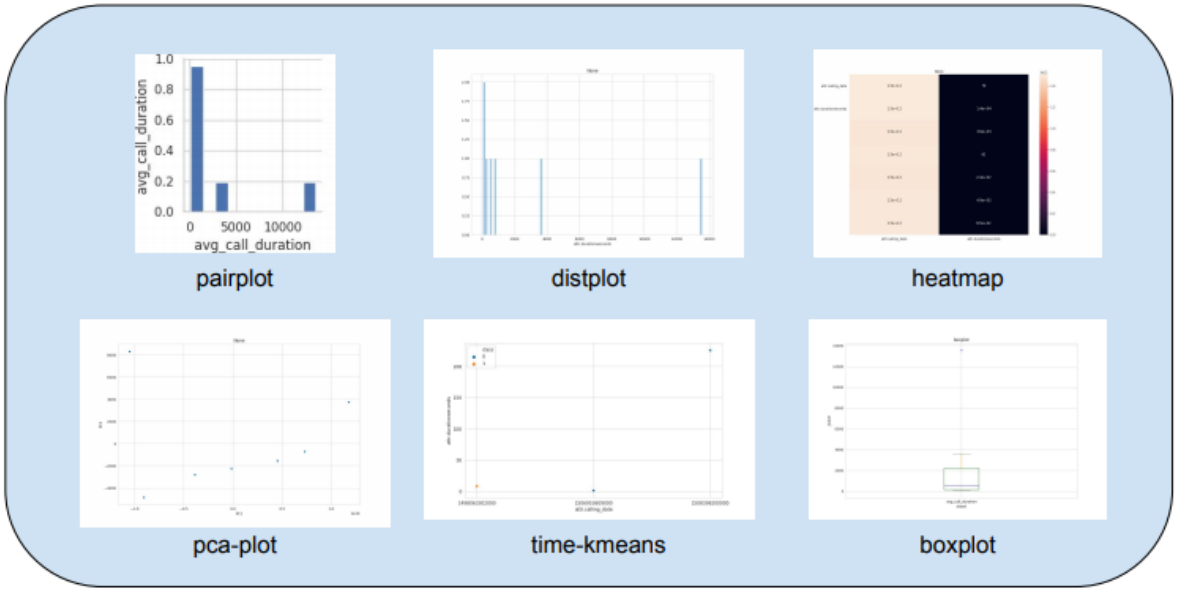


L'IHM pour faire des demandes Big Data vers serveur avec pop-up message



L'interface pour le Fraud Detection

Nous pouvons faire des demandes de big data sur la page et obtenir des analyses statistiques, telles que distplot, heatmap, etc.



Les analytics Big Data avec nos données venant de Kafka

5. Conclusion

Rappelant notre objectif initial, on peut dire que on a pratiquement terminé les deux projets du stage. Sous la direction de mon mentor d'entreprise, M. Le Fellic, j'ai réalisé ces deux projets étape par étape.

Mais il reste encore beaucoup de choses à considérer et à améliorer.

Par exemple, pour le deuxième projet, en plus de ce que on a fait, il est recommandé d'utiliser une conception de file d'attente plus spécifique et plus robuste. Je suggère donc que l'on puisse utiliser la coopération d'ELK + Kafka pour mettre en œuvre un cluster de système de collecte de log.

6. Références

- Fernando Monteiro – Learning Single-page Web Application Development
- Angular docs : https://angular.io/docs
- Le Guide Angular par Marmicode : https://guide-angular.wishtack.io/
- MongoDB documentation : https://docs.mongodb.com/manual/
- lensesio/fast-data-dev : https://github.com/lensesio/fast-data-dev
- lensesio/kafka-cheat-sheet : https://github.com/lensesio/kafka-cheat-sheet
- Docker docs : https://docs.docker.com/get-docker/
- On-street Car Parking Sensor Data - 2017 : https://data.melbourne.vic.gov.au/Transport/On-street-Car-Parking-Sensor-Data-2017/u9sa-j86i
- Cross-origin resource sharing (CORS) : https://developer.mozilla.org/fr/docs/Web/HTTP/CORS