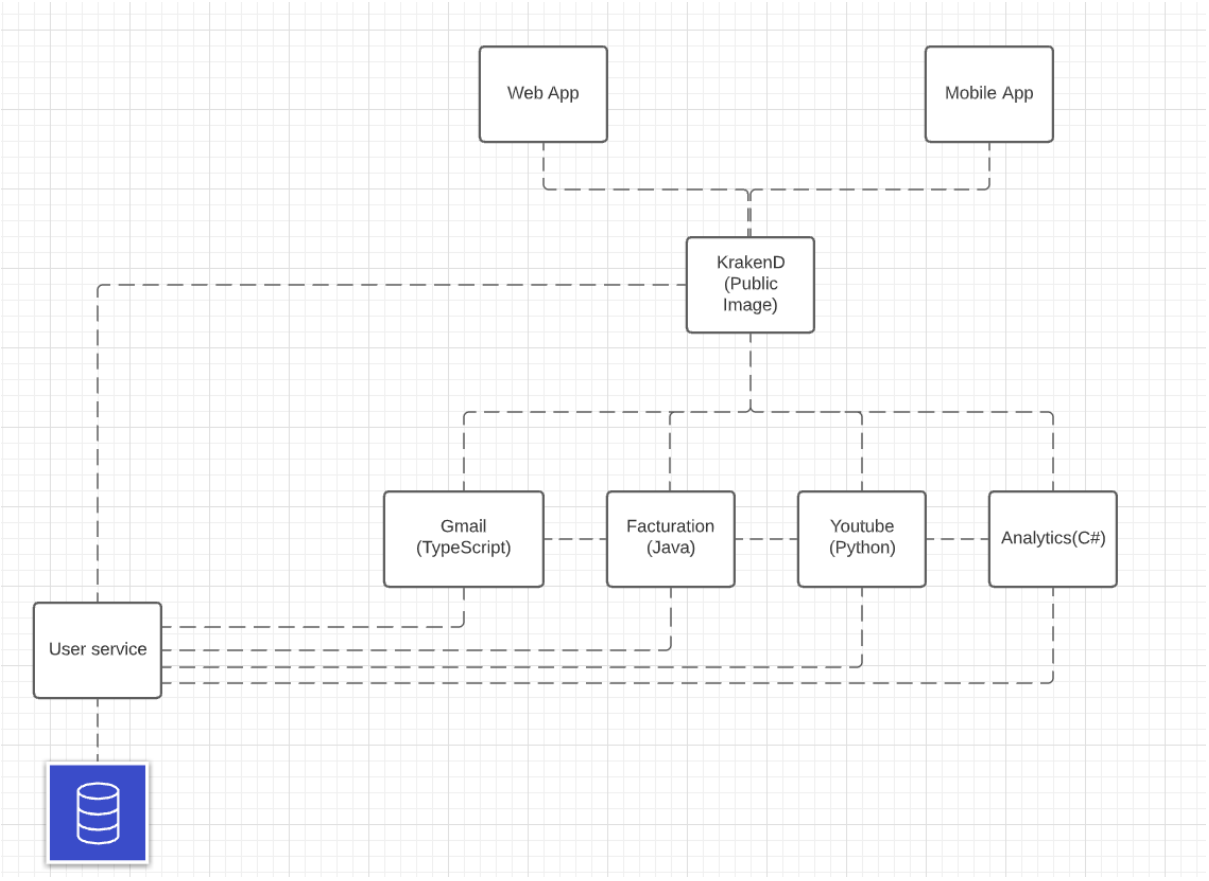
Architecture Microservice

Les microservices sont un style d'architecture utilisé par de nombreuses organisations pour le développement de logiciels. Par le passé, l'industrie informatique utilisait des solutions monolithiques (une seule application) ou basées sur l'architecture orientée services (ou SOA pour Service-Oriented Architecture) comme standard.

Cependant, le manque d'évolutivité dynamique de ce genre de système architectural n'était plus adapté à la complexité croissante des infrastructures actuelles. C'est là qu'intervient le microservice qui est finalement est une évolution logique du système SOA conçu pour atteindre un degré élevé d'agilité, de distribution rapide et d'évolutivité.

Concrètement, les microservices sont une méthode développement logiciel utilisée pour concevoir une application comme un ensemble de services modulaires. Chaque module répond à un objectif métier spécifique et communique avec les autres modules.
Il est intéressant de noter que chaque module peut être développé dans un langage différent en fonction des besoins puisqu'ils sont tous indépendants les uns des autres. Nous pouvons aussi imaginer par la suite, que chaque service est disponible sur un serveur différent afin de permettre une plus grande scalabilité lorsque l'affluence sur le site augmente par exemple.

Voici ci-dessous un schéma qui illustre bien cela :



Avant toute explication, il est intéressant d’introduire la notion d’api gateway : Une **API Gateway** (également appelée passerelle **API**) est le point d'entrée unique pour les **API** et microservices back-end définis (qui peuvent être à la fois internes et externes). Intervenant avant les **API**, l'**API Gateway** agit en tant que protecteur, renforçant la sécurité et assurant l'évolutivité et la haute disponibilité. Elle permet aux différents clients de requêter un seul et unique même point d’entrée, cela renforce la sécurité des systèmes. L’api gateway se charge ensuite de rediriger les requêtes sur les différents services concernés.

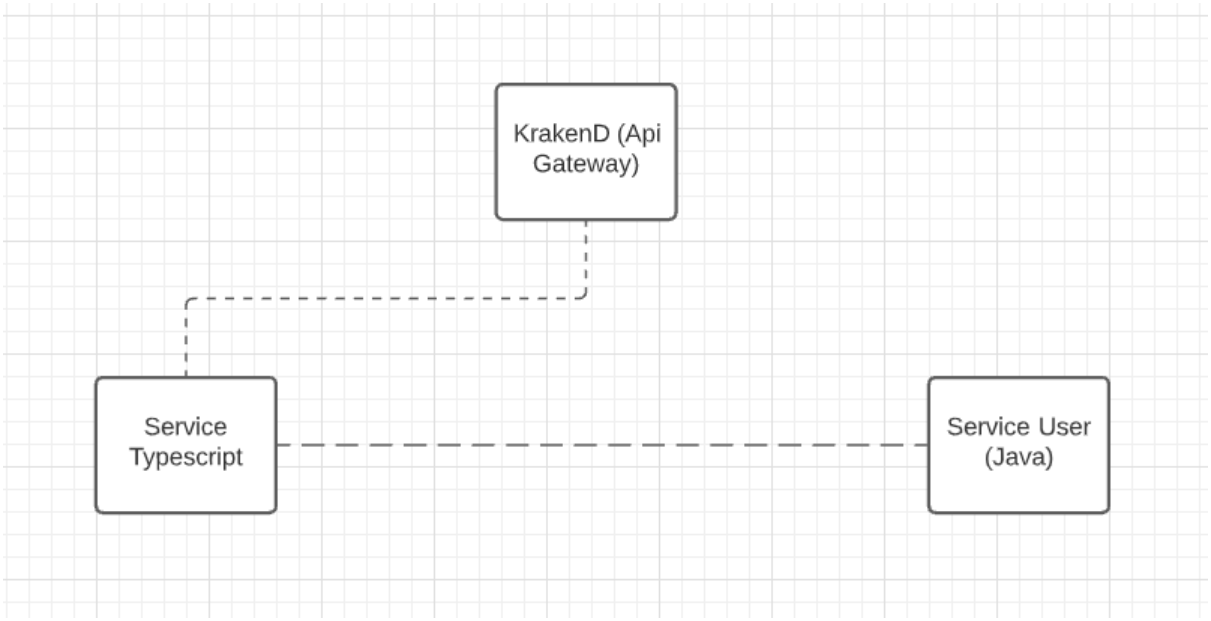
Sur le schéma ci-dessus, nous observons le fonctionnement d’une architecture microservice.

En premier lieu, nous avons une web app ainsi qu'une application mobile, ces dernières représentant les différentes applications utilisées par les utilisateurs. Ces deux applications pour communiquer avec le backend de la solution vont passer par l’api gateway qui va se charger de rediriger les requêtes sur les bons services et de renvoyer la réponse.

D’ailleurs, nous pouvons observer différents services, développé en différents langages (TypeScript, Java, Python, C#). Nous avons un service Gmail qui permet de gérer toutes les requêtes liées à la gestion de gmail dans notre application. Un service de facturation, qui lui permet la création/modification de facture. Un service Youtube qui nous permettra l’utilisation de youtube dans notre solution. Un service analytics qui lui se chargera de gérer le traitement des données des utilisateurs afin d’obtenir des informations intéressantes sur les données que nous fournissons aux utilisateurs. Et enfin un service User, qui lui se chargera de gérer tout ce qui est en lien avec la database de notre solution.

Tous ces services sont indépendants, et nous pouvons imaginer qu’ils sont hébergés sur des serveurs différents puisqu’ils offrent tous une api permettant l'intercommunication.

Étude de cas



Pour notre étude de cas, nous avons décidé de mettre en place une api gateway. Nous avons choisi l'utilisation de krakenD qui est un projet open source utilisé avec docker.

Nous avons réalisé un service en typescript offrant une api permettant de récupérer des données de météo à Paris. Et permettant la création d'un nouvel utilisateur (ici, aucune database n'a été utilisée, nous revoyons seulement des réponses json).

Notre api gateway est disponible sur le port 8080 en localhost et se chargera d'interpréter les requêtes pour les transmettre au bon service.

Voici les différentes routes de notre projet :

localhost:8080/telMeHello : requête qui renvoie "hello world" en json et qui sera interprété par le service typescript

localhost:8080/giveMeWeather : requête qui renvoie les informations météo de Paris à date en json et qui sera interprété par le service typescript

localhost:8080/createUser : requête qui demande la création d'un nouvel artiste et qui sera interprété par le service user

body :

```
{
 "email": "string",
 "name": "string",
 "password": "string"
}
```

localhost:8080/createUserWithService : requête qui demande la création d'un nouvel artiste et qui sera interprété par le service typescript, qui lui transmettra la requête au service user.

body :

```
{
 "email": "string",
 "name": "string",
 "password": "string"
}
```