# Architecture Logicielle: Évolution

Equipe B: Guillaume Piccina, William d'Andrea, Nicolas Fernandez, Yann Brault

Sujet V6 : Pay as you pollute

## Les services en détails:

- **Car tracker :**
  - Consomme dans le bus un topic de tracking
  - Transmettre à grande vitesse les infos de tracking vers la base de données (heure de pointes, bouchons)
  - Doit attendre le retour de *Zones Pollution* avant d'envoyer les data dans la base (il y a un lock ici)
- **Tracking shutdown :**
  - Consomme dans le bus un topic de shutdown
  - Récupère en base de données *Tracking Infos* les data du user
  - Déclenche la facturation pour un user en envoyant les data
  - Flush la base de données *Tracking Infos* de ces data
- **Position checker**
  - Consomme dans le bus un topic de redémarrage
  - Communique avec le système externe de caméra pour surveillance
  - Contacte *Billing* en cas de problème
- **Zones pollution :**
  - Doit pull à intervalle régulier les infos de pollution et les mettre en cache (solution de stockage clé valeur avec très gros débit de lecture)
  - Doit trouver une correspondance entre la position gps et la zone pour renvoyer la zone à *Car Tracker*
- **Billing :**
  - Reçoit l'event les infos de tracking depuis *Tracking Shutdown*
  - Calcule la facture finale du trajet
  - Stock la facture en dans la base de données *Bills*

## Nouveaux besoins:

- Retour et aide à l'utilisateur en temps réel ( le guider vers un parcours en zone moins chère )
- Détection des coupures réseaux avec détection de fraudes (par une carte des zones blanches connues) donc recalcul des montants après coup
- Pousser la résilience et le maintien des infos de passages
- Scénario de résilience maximale : pouvoir exploiter les données de passages / positionnement a posteriori

**Service pour répondre aux nouveaux besoins (en discussion)**

- **User configurator** :
    - Expose une route pour les users
    - Pull les infos de zones de pollutions : coordonnées des zones + prix associés aux zones (stock en cache et peut supporter une grosse charge en lecture)
    - Calcule des zones géographiques "de transition" autour des frontières des zones de pollution
    - Doit faire correspondre une position gps avec des zones
    - Retourne aux utilisateurs une politique de tracking (fréquent si proche d'une frontière de zones)
    - Renvoie aux utilisateurs le prix de la zone dans laquelle ils sont pour le calcul du prix estimé en real time
- **Route advisor**
    - Version MVP : doit accepter un trajet et retourner les zones composants ce trajet et donner un coût provisoire
    - Version étendue : doit prendre en entrée un trajet et retourner le coût provisoire du trajet + des trajets alternatifs moins chère
    - Version complète : comme la version étendue mais avec prise en compte des zones blanches
    - Doit pouvoir regarder en temps réel si un utilisateur ayant créer un itinéraire dévie de son trajet et lui renvoyer un nouveau trajet
- **User Proxy Client**
    - Joue le rôle d'intermédiaire entre toute notre architecture backend et notre vrai client
    - Il reçoit les positions des utilisateurs en temps réel et émet ces positions dans le bus de position
    - Il ouvre un websocket avec le client afin d'envoyer en temps réel des informations de changement d'itinéraire ou de changement de fréquence d'envoie de position
- **User Client**

- Application mobile du client (comportant le prix dépensé en temps réel, l'itinéraire, l'affichage de ses factures)
- Application développée sous framework front-end (angular ou react)
- Communique via REST pour l'envoie de position toutes les N secondes avec le Proxy
- Reçoit des informations du Proxy (changement d'itinéraire, changement de fréquence d'envoie de position)

## Scénario du flot de données:

- L'utilisateur envoie son id, sa position gps et un timestamp dans le bus.

- Le service *Car Tracker* consomme les infos disponibles dans le topic de tracking du bus et envoie la position gps au service *Zones Pollution* qui lui renvoie une zone de pollution puis *Car Tracker* stock les informations dans la base de données *Tracking Infos*.

- Quand l'utilisateur finit son trajet, il envoie les mêmes informations que d'habitude sauf que cette fois-ci le service *Tracking Shutdown* sera le consommateur.

- Le service *Tracking Shutdown*, en consommant les infos de son topic, reçoit l'info que l'utilisateur s'est arrêté, alors il va chercher en base de données les infos de tracking de l'utilisateur et les envoie vers *Billing.*

- Le service *Billing* avec les infos qu'il a reçues va calculer la facture puis il va la stocker dans la base de données *Bills.*

- L'utilisateur pourra accéder à ses factures et les payer via le service *Billing.*

## Scénario du flot de données V2

- L'utilisateur, quand il arrive dans sa voiture, appuie sur le bouton "se connecter" via UserClient, ce qui ouvre un socket avec UserProxyClient.
- UserProxyClient enregistre ce lien (cache, DB ? - en discussion de comment implémenter ce micro-service)
- L'utilisateur entre sa destination dans son application
- Une requête est envoyé à UserClientProxy qui est redirigé à RouteAdvisor et afin de générer un nouvel itinéraire
- RouteAdvisor envoie l'itinéraire (avec le montant potentiel à payer) à UserClientProxy qui envoie, via le socket, cet itinéraire au client

- Le client démarre son trajet, chaque N secondes, une requête (avec la position, timestamp, plaque d'immatriculation) est envoyé au proxy qui convertit cette information en message et la transfère dans le bus.
- CarTracker recoit le message, il le stocke en DB
- PositionChecker lit 1 message sur 10 dans le bus, regarde que la voiture se situe bien à l'endroit défini, si ce n'est pas le cas, génération d'une amende à travers le micro-service billing
- UserConfigurator lit chaque messages, regarde la distance entre la voiture et les frontières de zones, renvoie une information à Proxy (qui envoie a client) sur la fréquence d'envoie de position
- Si l'utilisateur ne suit plus son itinéraire, Route Advisor le détecte et envoie un nouvel itinéraire
- Quand un utilisateur a fini son trajet, il émet un "car-shutdown" qui va permettre au micro-service CarShutdown de mettre fin à la course, ce qui va entraîner une sauvegarde du trajet (pour avoir des statistiques) grâce à Tracking Analytics et une génération de la facture via Billing

## Choix de techno :

- Bus : RabbitMQ pour le moment (évolution possible en AL Evolution)
- BD principale : NoSQL
- BD zone pollution, tracking policy: SQL qui supporte une grosse charge de lecture
- Archi micro service : NestJS

## Schéma de l’architecture :