Architecture Logicielle: Construction
Rendu final

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*
- **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

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.*

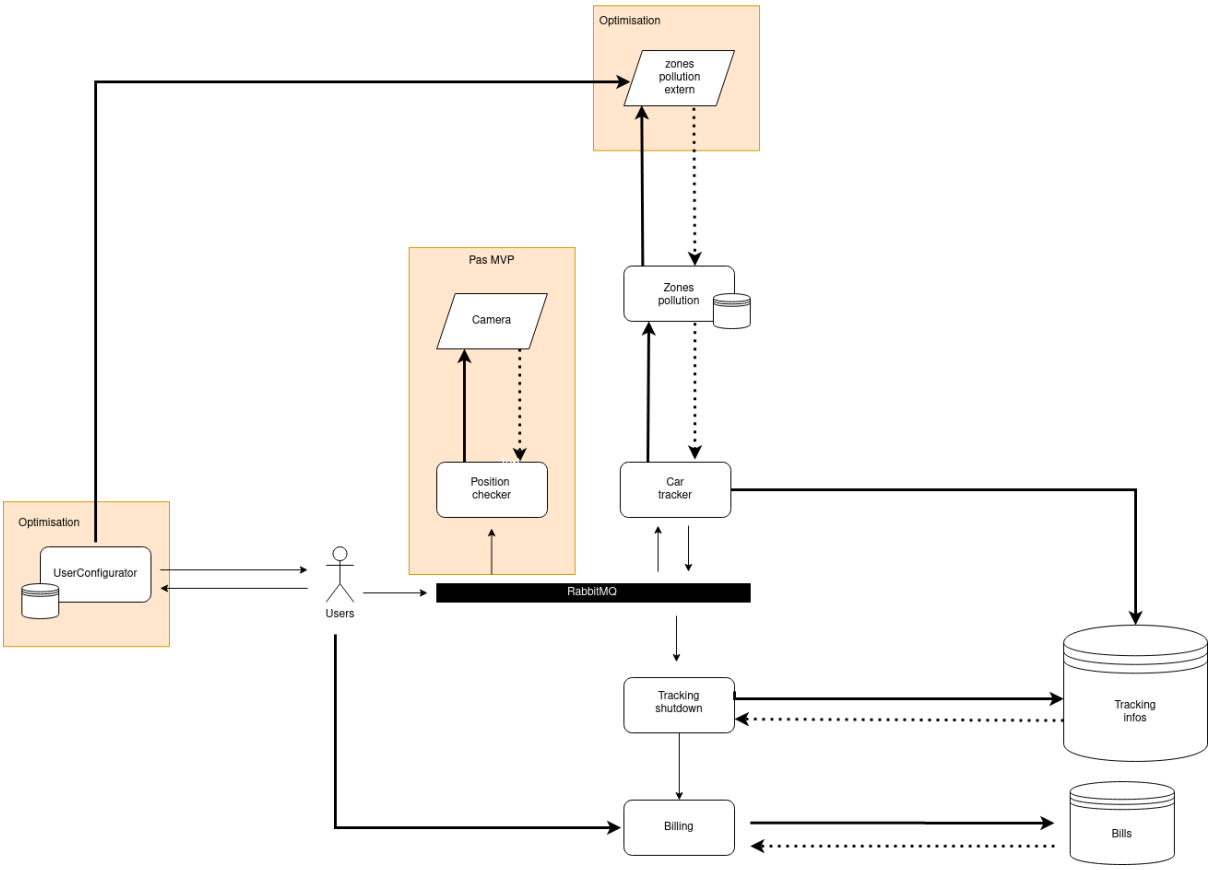
- Le service *Billing* avec les infos qu'il a reçues va calculer la facture puis il va la stocker dans la base de données *Bills.*

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

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 :



Auto-évaluation du travail :

L’investissement personnel de tous les membres du groupe est équivalent. Nous considérons que nous avons tous travaillé de manière équivalente sur le projet mais sur des aspects différents.
William, Nicolas et Guillaume ont travaillé sur le code du projet. William a également travaillé sur la partie des tests de charge. Enfin tous les membres du groupe ont travaillé de manière équivalente tout au long du projet sur les réflexions quant à l’architecture, les solutions et les problématiques. Les participations étaient en général plutôt pertinentes. La dernière version de ce rapport ainsi que les slides de la présentation ont été fait par nous quatre également.
Nous pensons que la qualité de notre travail est plutôt bonne. Nous avons réussi à identifier le métier principal et à l’implémenter dans globalité, nous avons aussi été en capacité de déterminer que notre problème principal est la résilience de l’architecture et il se caractérise à travers un single point of failure qui pour nous est notre base de données des logs de tracking. Nous avons aussi pu déterminer les points critiques de notre architecture avec des injections de charges et nous avons des idées pour la suite du projet.

Répartition des points:

- Guillaume Piccina : 100 points
- William d’Andrea : 100 points
- Nicolas Fernandez : 100 points
- Yann Brault : 100 points