

Table des matières

[.. Objet du document 2](#_Toc206581906)

[.. Objectifs du projet 2](#_Toc206581907)

[.. Principes d’architecture 2](#_Toc206581908)

[.. Architecture existante 2](#_Toc206581909)

[.. Architectures 2](#_Toc206581910)

[fonctionnel. 2](#_Toc206581911)

[De données 3](#_Toc206581912)

[Technologiques 3](#_Toc206581913)

[.. Justification de l’approche architecturale 3](#_Toc206581914)

[.. Architectures de transition 3](#_Toc206581915)

# .. Objet du document

Ce document permet de modéliser l’architecture qui sera réalisée par les équipes de développement. Au-delà de la modélisation graphique, il s’agit également d’énoncer les principes sur lesquels l’architecture s'appuie, de justifier cette approche architecturale, mais également d’indiquer des transitions si nécessaire.

# .. Objectifs du projet

L’application Your Car Your Way a pour but de permettre à un utilisateur de pouvoir, à la suite d’une authentification, choisir un véhicule de location dans le but de se rendre d’un emplacement et moment X à un emplacement et instant Y. Pour cela, l’utilisateur cherchera dans une interface des véhicules correspondant à sa demande et pourra en réserver un. A l’heure actuelle, plusieurs applicatifs dans plusieurs langues permettent déjà la résolution de cette problématique. Cependant, dans un soucis d’uniformiser l’application, une nouvelle application multilingue centralisera l’ensemble des demande des utilisateurs, peu importe leur langue et emplacement géographique.

# .. Principes d’architecture

Pour réaliser l’application, il va nous falloir deux sections majeures :

* Un backend réalisé avec **Java / Spring Boot** disposant d’un ensemble d’APIs servant à la résolution des requêtes :
  + Une API permettant l’authentification des utilisateurs
  + Une API permettant la recherche et la sélection d’un véhicule
  + Une API servant à la résolution d’une réservation et au référencement du paiement final
  + Une API pour l’assistance litige asynchrone (par mail / messages sur la plateforme)
  + Une API pour le chat lors d’une demande d’assistance immédiate via conseiller
* Un frontend réalisé avec **Angular** servant à l’affichage d’une interface homme-machine adaptée à la langue du navigateur comportant plusieurs modules de sorte à permettre du lazy loading :
  + Module d’authentification
    - Page de connexion / enregistrement
  + Module de visualisation des véhicules
    - Page de référencement des véhicules disponible avec filtres et barre de recherche
  + Module de réservation
    - Page de réalisation d’une réservation
    - Page de paiement
  + Module de gestion utilisateur
    - Page de profil utilisateur servant à en modifier les informations ou à supprimer le compte
    - Page de réinitialisation du mot de passe
  + Module d’assistance litige
    - Page d’assistance regroupant des tickets d’assistance par message / mail et la capacité à demander une assistance instantanée par chat ou via un envoi de mail
    - Page de suivi des messages du chat

En réalisant une séparation des fonctionnalités et leur regroupement dans des sous-ensemble, on peut permettre la mise en place progressive d’une architecture de type micro-service au niveau du backend. Cette architecture permettra un développement facilité des fonctionnalités et leur mise en service plus rapide. Pour éviter d’avoir une sécurité faillible et d’avoir besoin d’un certificat TLS pour chaque micro-service, on procèdera à la mise en place d’une Gateway qui se chargera de rediriger les requêtes HTTPS provenant des clients (extra-serveur) vers les microservices par le protocole HTTP (intra-serveur).

Au niveau de l’application frontend, l’utilisation de plusieurs modules devrait permettre également un développement plus aisé et aussi un chargement plus rapide de l’applicatif au niveau du navigateur. Il serait de bon ton de mettre également en place du SSR (Server-Side Rendering) de sorte à obtenir un référencement plus rigoureux de notre application. L’ajout de SSR permettrait également de traiter en interne la récupération de la langue utilisée par le client dans le but de lui fournir des pages web adaptées à ses besoins directement.

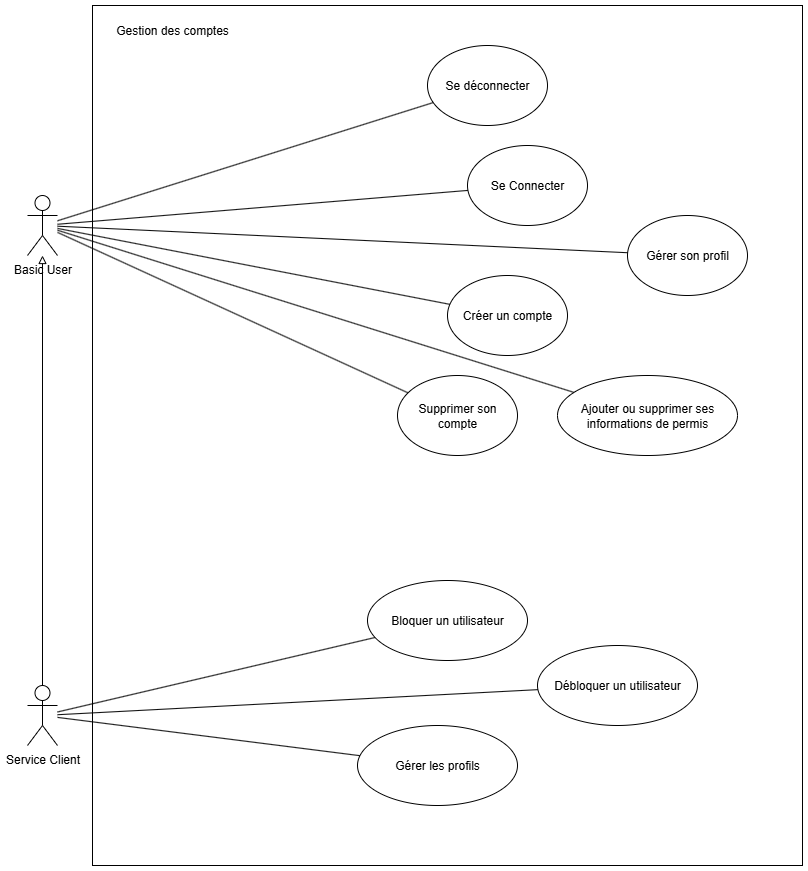
# .. Architecture existante

Le projet ne s'appuie pas sur un produit existant qui serait repris puis mis à jour. Il n’y a donc pas d’architecture existante à définir.

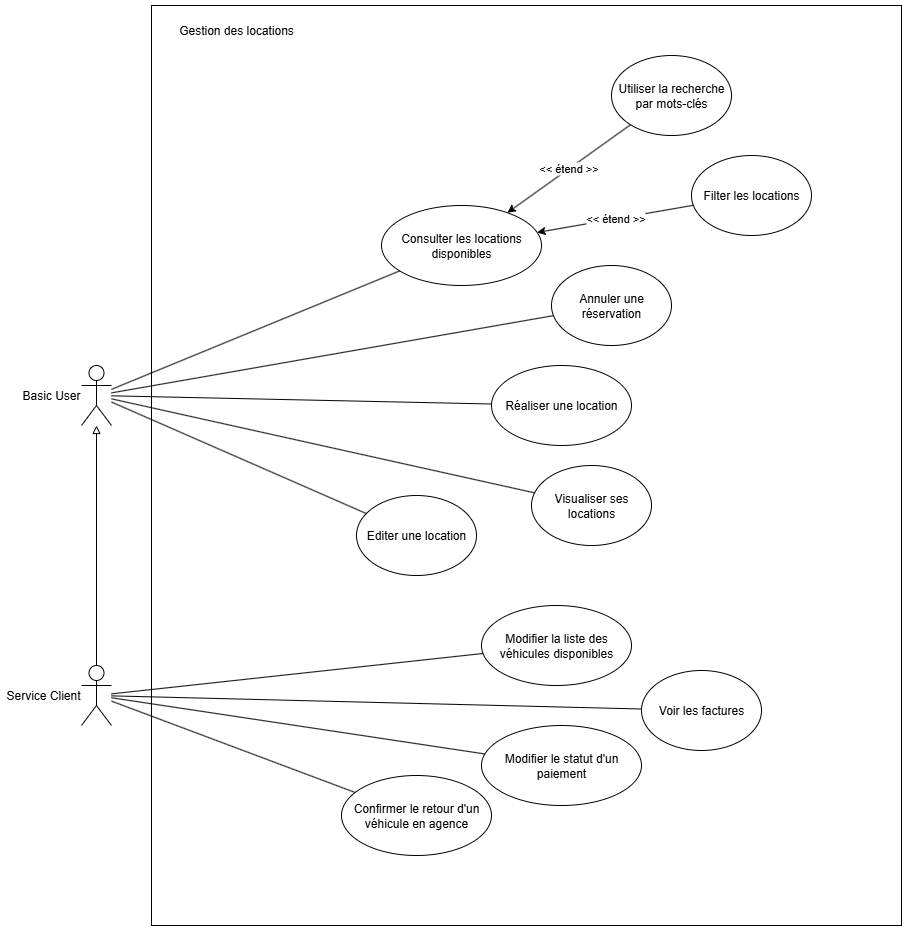
# .. Architectures

## Métier

### Authentification et Gestion des comptes



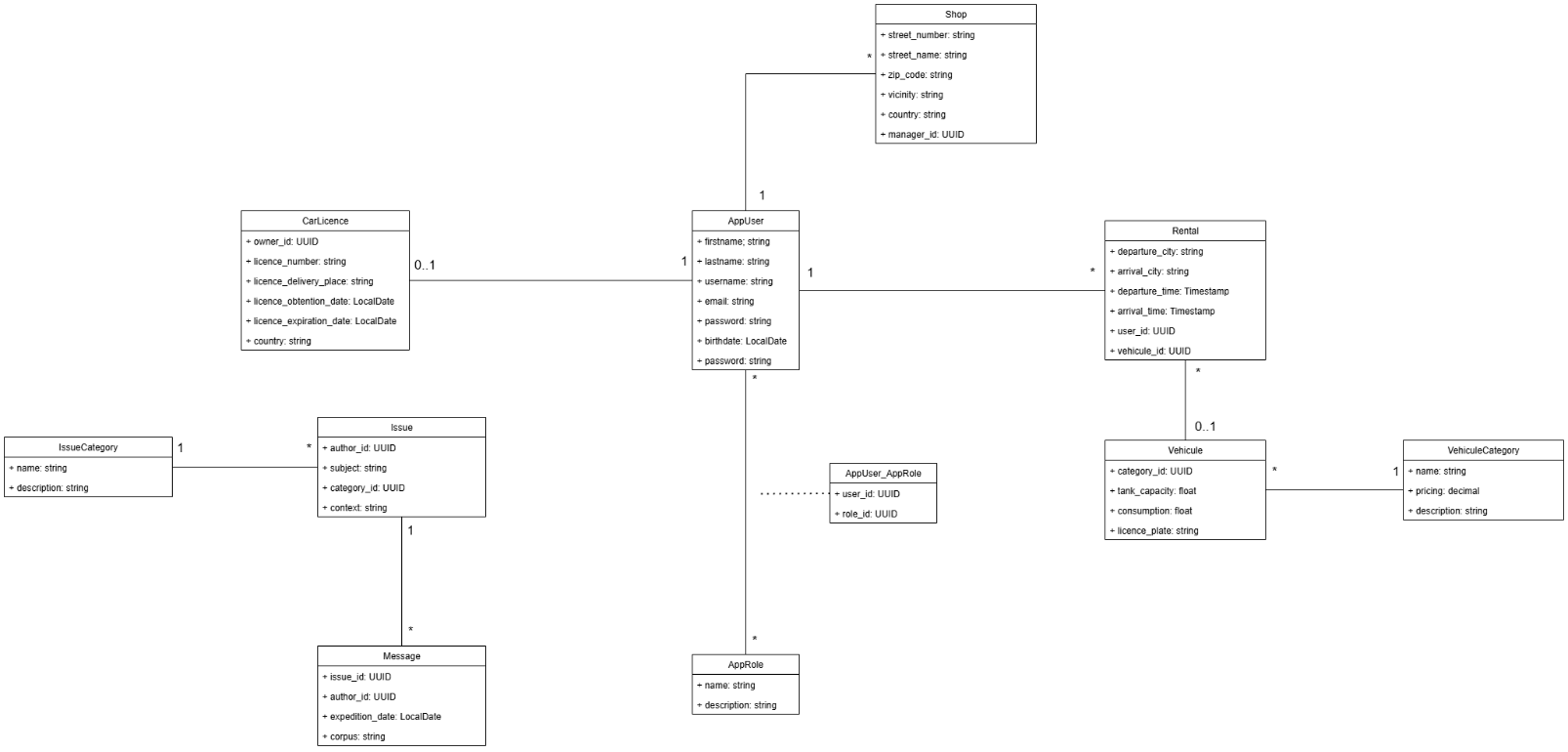
### Gestion des locations



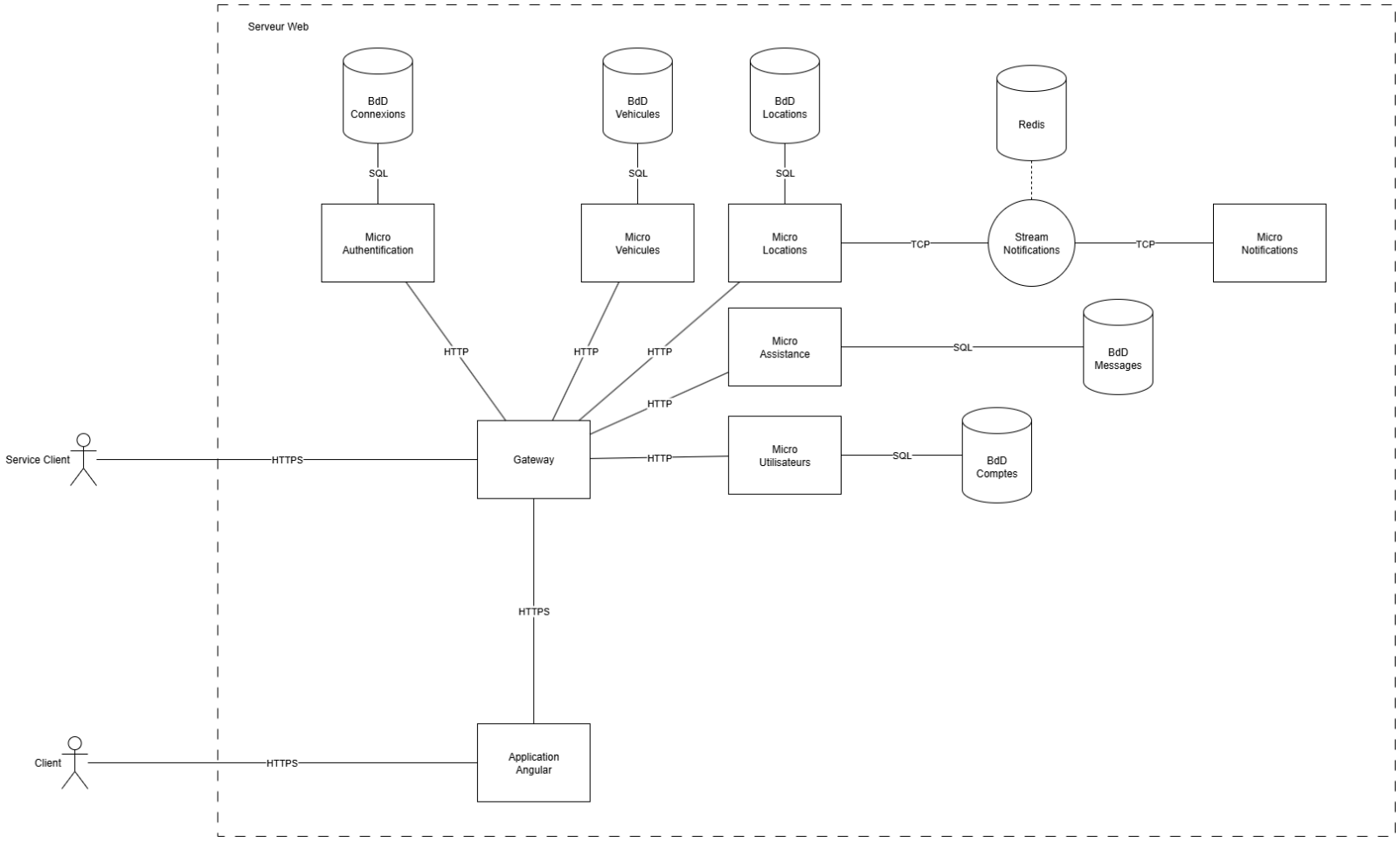
### Assistance et litiges



## De données



## Technologiques



# .. Justification de l’approche architecturale

L’architecture proposée se trouve être, pour le backend, une architecture de type micro-services. Dans le cadre de cette architecture, chaque service se verra conteneurisé et déployé dans un environnement Cloud via Kubernetes.

* L’emplacement de notre cluster va être l’une des clés du respect de la RGPD, cette dernière demandant le positionnement de nos ressources en France / Europe. Via Azure France, il serra possible de déployer l’ensemble des applicatifs sur le territoire français. Les bases de données seront également hébergées sur Azure France. Dans le cadre d’une forte demande, il sera possible de bénéficier d’une série de replicas de nos bases de données en lecture seule de sorte à pouvoir accélérer le traitement des requêtes par les API.
* La sécurisation de notre applicatif étant essentielle, l’accès aux micro-service se ferra via le protocôle HTTPS uniquement. Il serra possible d’intéragir directement avec l’API en cas de besoin via un client http, mais la majorité du traitement des requêtes par les clients se feront via l’application frontend. Cette application sera également conteneurisée et disponible via un Ingress et un Load Balancer afin d’offrir un nom de domaine unique pour les visiteurs de l’application.
* L’utilisation du SSR dans l’application frontend permet d’offrir un meilleure référencement ainsi qu’une meilleure sécurité (la plupart des requêtes http vers l’api étant du coup traité dans le serveur et donc non visibles pour un client). De plus, le SSR devrait faciliter la mise en place d’une application avec une interface multilingue.
* Dans le cadre du déploiement en nuage, il est possible de mettre en place l’ensemble de notre architecture via des outils comme Terraform et Ansible. Ces outils permettent d’avoir un suivi de la version de notre architecture et devraient permettre un plan de restauration facilité.
* L’API Gateway possèdera un Rate limiter avec la réalisation de logs pour permettre d’éviter de potentielles surcharges de nos micro-services ainsi que l’identification d’une attaque de type déni de service (DoS). La plupart des cloud providers modernes permettent la mise en place de ce processus de façon aisée.
* L’ensemble des secrets de l’applicatifs, qu’il s’agisse de clés API pour les services tiers ou de nos propres applicatifs, devront être gérer via l’utilisation d’un gestionnaire tel qu’Azure Key Vault. Pour cela, la mise en place de notre applicatif dans un environnement en nuage est privilégiée.

# .. Architectures de transition

