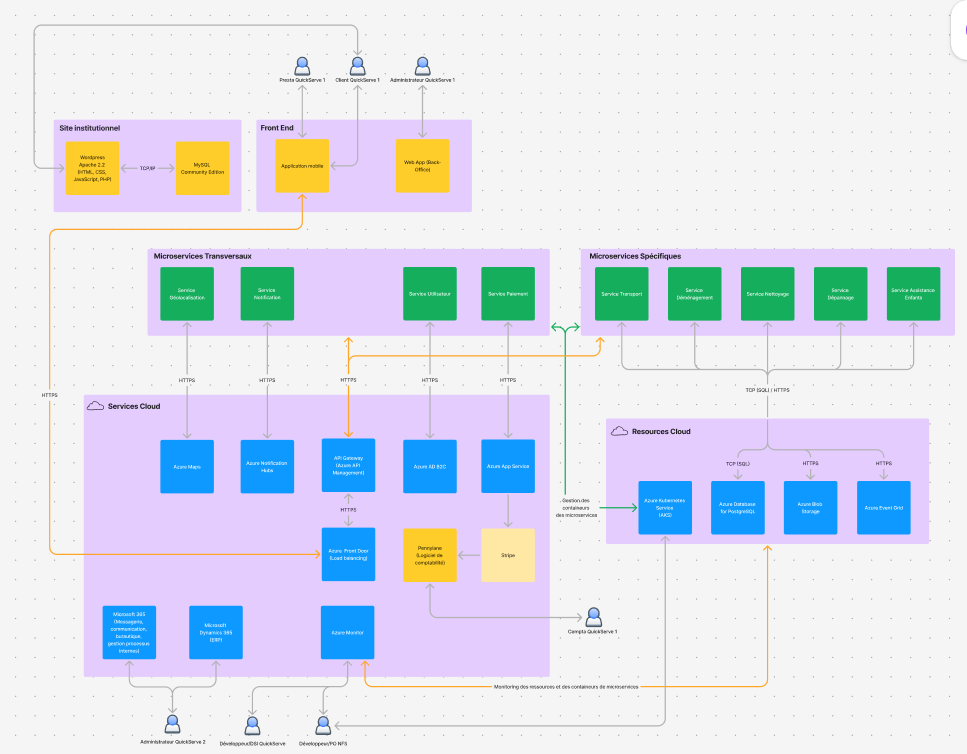
# Modélisation technique

## Schéma de l’architecture logicielle



## ****Modélisation des bases de données****

* **Objectif :** Structurer les données pour chaque microservice avec des schémas conceptuels et physiques.

### ****Services à modéliser :****

#### ****1. UserService****

* **Description** : Gère les utilisateurs (clients et prestataires).
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| name | VARCHAR(100) | Nom de l'utilisateur. |
| email | VARCHAR(100) | Email unique. |
| password | VARCHAR(255) | Mot de passe hashé. |
| role | ENUM('client', 'prestataire') | Rôle de l'utilisateur. |
| created\_at | TIMESTAMP | Date de création de l'utilisateur. |

#### ****2. OrderService****

* **Description** : Gère les commandes passées par les utilisateurs.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| user\_id | INT [FK > UserService.id] | Référence à l'utilisateur ayant passé la commande. |
| service\_type | ENUM('transport', 'nettoyage', 'dépannage', 'garde enfant', 'déménagement') | Type de service demandé. |
| status | ENUM('en cours', 'terminé', 'annulé') | Statut de la commande. |
| latitude | DECIMAL(9,6) | Latitude du lieu de service. |
| longitude | DECIMAL(9,6) | Longitude du lieu de service. |
| created\_at | TIMESTAMP | Date de création de la commande. |

#### ****3. PaymentService****

* **Description** : Gère les transactions financières liées aux commandes.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| order\_id | INT [FK > OrderService.id] | Référence à la commande associée. |
| amount | DECIMAL(10,2) | Montant de la transaction. |
| payment\_status | ENUM('en attente', 'validé', 'échoué') | Statut du paiement. |
| created\_at | TIMESTAMP | Date de la transaction. |

#### ****4. NotificationService****

* **Description** : Envoie des notifications aux utilisateurs.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| user\_id | INT [FK > UserService.id] | Référence à l'utilisateur concerné. |
| message | TEXT | Contenu de la notification. |
| is\_read | BOOLEAN | Statut de lecture (non lu/lu). |
| created\_at | TIMESTAMP | Date d'envoi de la notification. |

#### ****5. ProviderService****

* **Description** : Gère les informations des prestataires de services.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| name | VARCHAR(100) | Nom du prestataire. |
| email | VARCHAR(100) [UNIQUE] | Email unique du prestataire. |
| availability | BOOLEAN | Disponibilité (disponible/non). |
| latitude | DECIMAL(9,6) | Latitude de la position actuelle. |
| longitude | DECIMAL(9,6) | Longitude de la position actuelle. |
| created\_at | TIMESTAMP | Date de création du prestataire. |

#### ****6. RepairService****

* **Description** : Gère les réparations pour les utilisateurs.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| order\_id | INT [FK > OrderService.id] | Référence à la commande associée. |
| issue\_type | ENUM('batterie', 'pneu', 'moteur', 'autre') | Type de panne. |
| technician\_name | VARCHAR(100) | Nom du technicien. |
| status | ENUM('en route', 'en cours', 'terminé', 'annulé') | Statut de l’intervention. |
| created\_at | TIMESTAMP | Date de création de l’intervention. |

#### ****7. ChildAssistanceService****

* **Description** : Gère les services d’assistance pour enfants.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| order\_id | INT [FK > OrderService.id] | Référence à la commande associée. |
| guardian\_name | VARCHAR(100) | Nom du gardien. |
| child\_count | INT | Nombre d’enfants pris en charge. |
| destination | VARCHAR(255) | Destination prévue. |
| status | ENUM('en cours', 'terminé', 'annulé') | Statut de la prestation. |
| created\_at | TIMESTAMP | Date de création de la prestation. |

#### ****8. MovingService****

* **Description** : Gère les déménagements pour les utilisateurs.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| order\_id | INT [FK > OrderService.id] | Référence à la commande associée. |
| team\_size | INT | Nombre de déménageurs. |
| truck\_size | ENUM('petit', 'moyen', 'grand') | Taille du camion. |
| status | ENUM('préparation', 'en cours', 'terminé', 'annulé') | Statut du déménagement. |
| created\_at | TIMESTAMP | Date de création du déménagement. |

#### ****9. CleaningService****

* **Description** : Gère les services de nettoyage.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| order\_id | INT [FK > OrderService.id] | Référence à la commande associée. |
| location\_type | ENUM('maison', 'bureau', 'véhicule') | Type de lieu nettoyé. |
| cleaning\_duration | INT | Durée estimée du nettoyage (minutes). |
| status | ENUM('préparation', 'en cours', 'terminé', 'annulé') | Statut du nettoyage. |
| created\_at | TIMESTAMP | Date de création de la prestation. |

#### ****10. TransportService****

* **Description** : Gère les services de transport.
* **Entités et colonnes** :

| ****Colonne**** | ****Type**** | ****Description**** |
| --- | --- | --- |
| id | INT [PK] | Identifiant unique. |
| order\_id | INT [FK > OrderService.id] | Référence à la commande associée. |
| vehicle\_type | ENUM('voiture', 'camion', 'moto') | Type de véhicule. |
| driver\_name | VARCHAR(100) | Nom du chauffeur. |
| driver\_contact | VARCHAR(15) | Contact du chauffeur. |
| status | ENUM('en route', 'livré', 'annulé') | Statut du transport. |
| created\_at | TIMESTAMP | Date de création de la commande. |

### ****Justification des choix pour les bases de données****

#### ****Pourquoi PostgreSQL pour les données relationnelles ?****

PostgreSQL a été retenu pour sa robustesse et ses fonctionnalités avancées qui répondent aux besoins spécifiques du projet **QuickServe** :

1. **Performance** :
   * **Requêtes complexes** : PostgreSQL excelle dans l'exécution de requêtes SQL complexes grâce à son moteur de requêtes optimisé.
   * **Transactions** : Supporte les transactions ACID, garantissant la cohérence des données même en cas de panne.
2. **Scalabilité** :
   * **Verticale** : PostgreSQL permet d'augmenter les ressources matérielles (CPU, mémoire) pour améliorer les performances.
   * **Horizontale** : Possibilité d'implémenter un sharding natif avec des extensions comme Citus pour gérer des bases de données massives.
3. **Fonctionnalités avancées** :
   * **Support des données géospatiales** : Avec PostGIS, il devient possible de traiter efficacement des données géographiques (ex. : coordonnées pour le suivi en temps réel des services).
   * **JSON/JSONB** : Permet le stockage et la requête de données semi-structurées pour les cas nécessitant de la flexibilité.
4. **Fiabilité** :
   * PostgreSQL est réputé pour sa stabilité et sa communauté active, garantissant un support et des mises à jour régulières.
5. **Open-source et extensibilité** :
   * Gratuit et facilement extensible grâce à des modules pour répondre aux besoins spécifiques du projet.

#### ****Pourquoi Redis pour le cache ?****

Redis a été choisi pour sa rapidité et sa capacité à répondre efficacement aux besoins de mise en cache et de traitement en temps réel.

1. **Performance** :
   * Redis est une base de données **in-memory**, ce qui la rend extrêmement rapide pour les opérations en lecture et en écriture.
   * Idéal pour des cas d'usage comme les notifications en temps réel ou le stockage des sessions utilisateurs.
2. **Faible latence** :
   * Redis garantit une latence quasi nulle, essentielle pour améliorer l'expérience utilisateur dans des services nécessitant une réactivité immédiate.
3. **Simplicité d’intégration** :
   * Redis peut être facilement intégré avec Symfony ou d'autres frameworks backend, grâce à des bibliothèques prêtes à l'emploi.
4. **Fonctionnalités avancées** :
   * **Pub/Sub** : Permet la communication en temps réel entre les services (par exemple, envoyer une notification immédiatement après une action utilisateur).
   * **TTL (Time-to-Live)** : Gère les données temporaires comme les sessions ou les caches de réponses API en définissant une durée de vie.
5. **Scalabilité** :
   * Redis Cluster permet une scalabilité horizontale pour gérer un grand nombre de requêtes en parallèle.
6. **Soutien d'Azure** :
   * Redis est disponible en tant que service managé sur Azure (Azure Cache for Redis), ce qui simplifie la gestion, la maintenance, et les mises à jour.

| ****Critère**** | ****PostgreSQL (Relationnel)**** | ****Redis (Cache)**** |
| --- | --- | --- |
| **Performance** | Excellente pour les transactions et requêtes complexes. | Très rapide grâce au stockage en mémoire. |
| **Latence** | Faible, mais dépend du volume de données. | Latence quasi nulle pour les lectures et écritures. |
| **Scalabilité** | Verticale et horizontale via sharding ou extensions. | Scalabilité horizontale via Redis Cluster. |
| **Cas d’usage principal** | Données transactionnelles (Commandes, Paiements). | Notifications en temps réel, sessions utilisateurs. |
| **Soutien sur Azure** | Disponible avec des options managées (PostgreSQL Flexible Server). | Azure Cache for Redis simplifie l’intégration. |

Ces deux bases de données se complètent parfaitement pour répondre aux exigences du projet **QuickServe** : PostgreSQL pour les données critiques et Redis pour les opérations nécessitant une vitesse immédiate.