

YOUR CAR
YOUR WAY

Business Requirements

1. Objet du document 2
2. Contexte 2
3. Périmètre 2
4. Liste des fonctionnalités 3
5. Exigences particulières 5

1. Objet du document

Le document “Business Requirements” liste les fonctionnalités à implémenter pour le projet Your Car Your Way. Ces fonctionnalités sont exprimées du point de vue métier sous la forme d’actions que l’utilisateur peut effectuer sur l’application.

2. Contexte

Your Car Your Way est une entreprise de location de voitures. Les clients utilisent actuellement des applications web qui ne correspondent plus aux besoins fonctionnels ni aux contraintes techniques. Une nouvelle application centralisée pour tous les clients doit être créée.

3. Périmètre

Les fonctionnalités décrites dans ce document concernent la première version de la nouvelle application Your Car Your Way. Cette application sera déployée à l'international, et utilisée par tous les clients de l’entreprise. Cette application est à destination des clients, et ne concerne pas les actions que les employés de Your Car Your Way doivent faire en agence.

4. Liste des fonctionnalités

Gestion du profil

- Connexion via la page de connexion.
- Inscription via la page d'inscription.
- Modifier ses informations personnelles (nom, prénom, date de naissance, adresse) via la page de profil.
- Supprimer son compte.

Gestion des agences

- Consulter la liste des agences de location.
- Affichage des détails d'une agence (adresse, horaires d'ouverture, services disponibles, véhicules disponibles).
- Rechercher / filtrer les agences avec une barre de recherche

Gestion d’une location de voitures

- Afficher les offres de location après avoir rempli un formulaire de recherche avec les critères suivants :
    - Ville de départ ;
    - Ville de retour ;
    - Date et heure de début ;
    - Date et heure de retour ;
    - Catégorie du véhicule.
- Consulter le détail d’une offre de location.
- Réserver une location correspondant à une offre, ce qui inclut :
    - Fournir ses informations personnelles (récupérables depuis le profil si présentes) ;
    - Effectuer le paiement.
- Consulter l’historique de ses réservations (passées et en cours).

- Consulter une réservation.
- Modifier et annuler une réservation.

Autres

- Envoyer un message au support (communication synchrone)
- Visio avec le support (communication synchrone)
- Envoyer un message à un chatbot si vous avez une question (communication asynchrone)

5. Exigences particulières

- La modification d’une réservation est possible jusqu’à 48 h avant le début de cette dernière.

- À moins d'une semaine du début de la réservation, YourCarYourWay ne rembourse que 25 % du montant total de la réservation.

- Les catégories du véhicule reprennent la norme ACRISS : https://www.acriss.org/car-codes/.

- La suppression du compte implique de saisir le mot de passe du compte.

- Les applications utilisées en agence doivent avoir accès à une API pour consulter et modifier les données traitées par l’application à destination des clients. Les opérations CRUD standard sont requises pour chaque domaine (exemple : utilisateur, réservation, etc.).

- La gestion du paiement doit être externalisée auprès d’un fournisseur de service de paiement en ligne (par exemple : Stripe).



YOUR CAR
YOUR WAY

Architecture Definition Document

1. Objet du document 2
2. Objectifs du projet 3
3. Principes d'architecture 4
4. Architecture existante 5
5. Architectures 6
6. Justification de l'approche architecturale 12
7. Architectures de transition 16

1. 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.

2. Objectifs du projet

Améliorer l’expérience client :

- Créer une application centralisée, intuitive et accessible à l’international.
- Simplifier la gestion des profils, recherches et réservations.

Faciliter la location de voitures :

- Centraliser les informations des agences et des offres des véhicules.
- Optimiser la recherche et la modification des réservations.

Optimiser les revenus :

- Appliquer une politique claire d’annulation et limiter les remboursements tardifs.
- Intégrer un système de paiement sécurisé via Stripe.

Renforcer la collaboration agences/clients :

- Fournir une API permettant aux agences de mettre à jour leurs données en temps réel.

Améliorer le support client :

- Proposer un chatbot pour les requêtes simples.
- Offrir un support synchrone via messagerie et visioconférence.

Garantir la sécurité et la conformité :

- Respecter les normes de sécurité (PCI DSS et RGPD) pour les paiements et les données personnelles.

3. Principes d’architecture

Modularité :

L’architecture sera divisée en modules distincts (gestion des profils, réservations, paiements, etc.) pour une meilleure maintenabilité.

Scalabilité :

Les composants doivent être conçus pour supporter une montée en charge et des pics de trafic.

Interopérabilité :

L’application doit utiliser une d’API RESTful pour les interactions entre applications client et agence.

Sécurité :

Intégration de JWT et OAuth2 pour l’authentification, chiffrement des données sensibles.

Performance :
Garantir un temps de réponse moyen inférieur à 200ms pour les principales opérations.

Portabilité :
Utiliser Docker pour garantir des déploiements consistants dans différents environnements.

4. 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.

5. Architectures

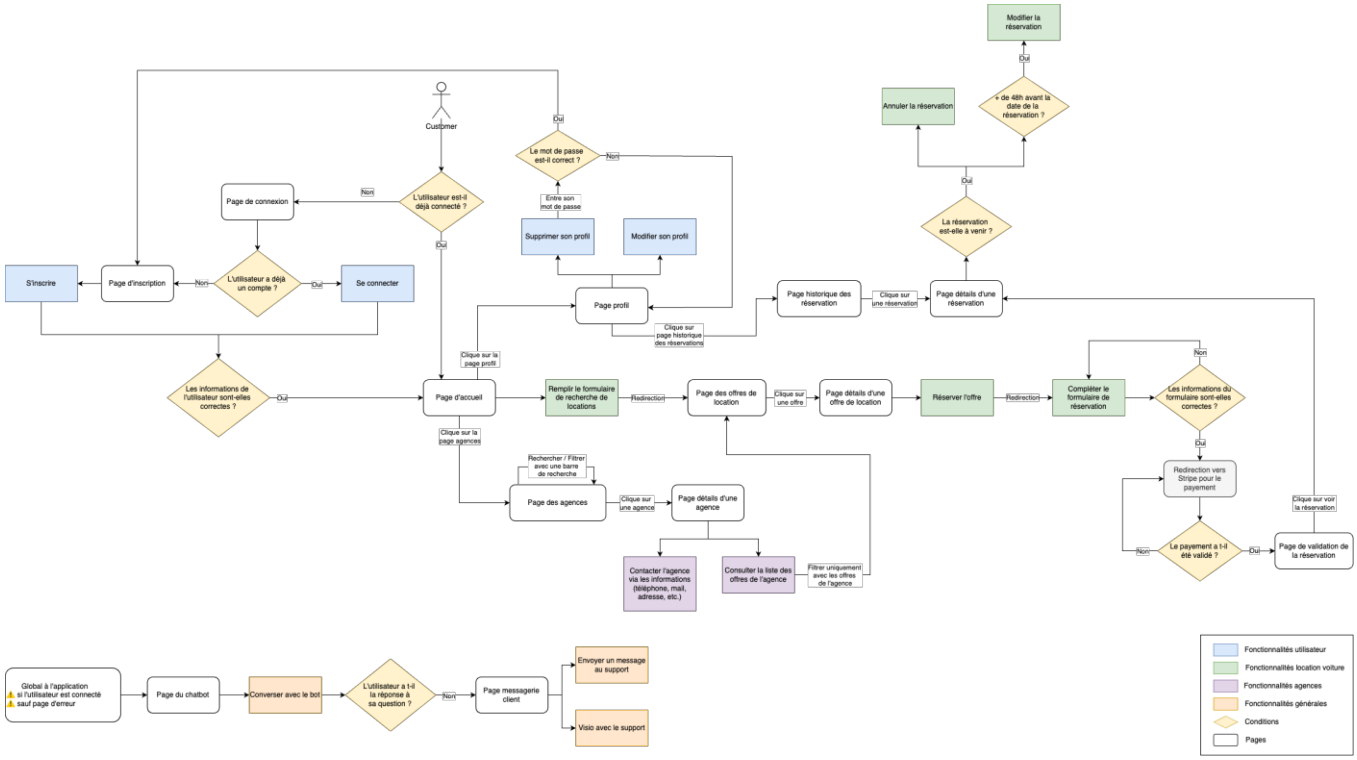
a. Métier

L’application permet les actions suivantes :

- **Gestion des profils utilisateurs :**
    - Inscription et connexion sécurisée (email et mot de passe, ou authentification sociale).
    - Modification et consultation des informations personnelles (nom, adresse, numéro de téléphone, etc.).
- **Recherche et réservation de véhicules :**
    - Recherche avancée avec des critères comme la ville de départ, la date et l’heure de prise en charge/retour, et la catégorie du véhicule.
    - Consultation des offres détaillées, incluant les tarifs, les caractéristiques des véhicules, et la disponibilité.
    - Réservation rapide avec un processus fluide, incluant la récupération des données du profil utilisateur et le paiement en ligne.
- **Gestion des réservations :**
    - Visualisation des réservations en cours et passées dans un historique clair.
    - Modification ou annulation des réservations, avec des règles d’annulation adaptées (par exemple, remboursement partiel en cas d’annulation tardive).
- **Support client :**
    - Chatbot intégré pour des réponses immédiates aux questions fréquentes.
    - Messagerie directe et visioconférence pour des interactions personnalisées avec le support.
- **Paiements sécurisés en ligne :**

- Intégration de Stripe pour gérer les transactions financières de manière fiable et conforme aux normes de sécurité.

Schéma fonctionnel :



---

b. De données

Les entités principales :

- **Utilisateurs :**
  - Informations personnelles (nom, prénom, email, adresse, numéro de téléphone).
- **Agences :**
  - Informations sur les agences (nom, adresse, horaires d'ouverture, coordonnées).
  - Relation avec les véhicules disponibles.
- **Véhicules :**

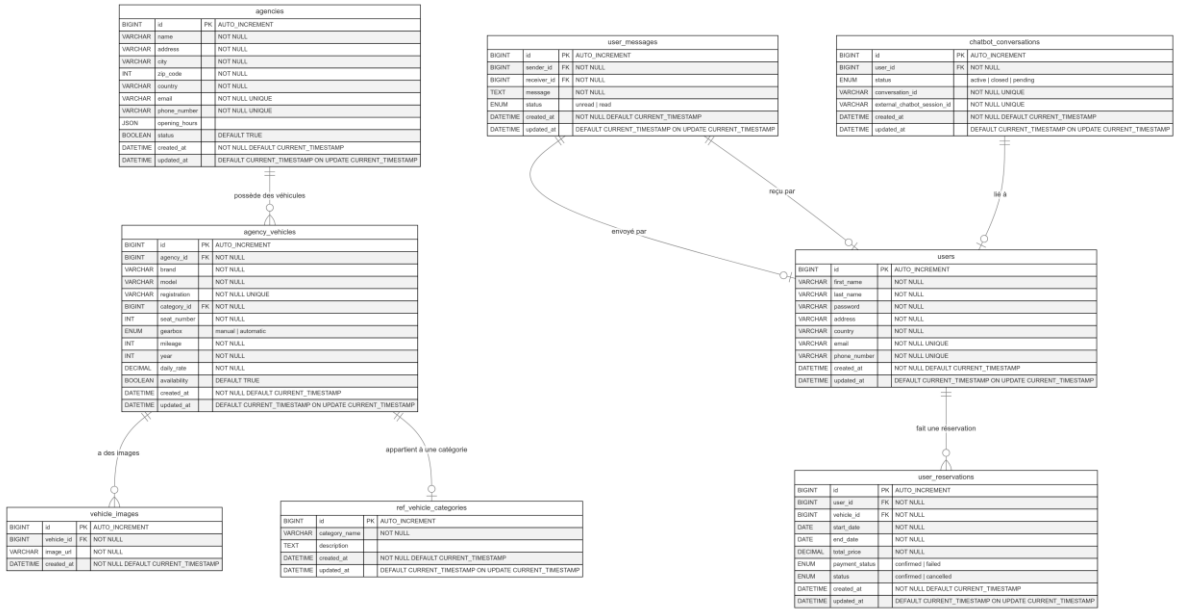
    - Caractéristiques des véhicules (marque, modèle, année, kilométrage, catégorie, tarif journalier).
    - Disponibilité et images associées.
- **Réservations :**
    - Détails des réservations (utilisateur, véhicule, agence, dates de début et de fin, état de la réservation).
    - Historique des modifications ou annulations.
- **Paiements :**
    - Informations sur les transactions (montant, méthode de paiement, statut).
- **Messages :**
    - Gestion des communications entre utilisateurs et support (asynchrones ou synchrones).

Relations clés :

- Une agence peut posséder plusieurs véhicules (relation 1-N*).
- Un utilisateur peut effectuer plusieurs réservations (relation 1-N*).
- Chaque réservation relie un utilisateur, un véhicule et une agence.
- Les messages peuvent relier plusieurs utilisateurs au support (relation N-N**).

*relation 1-N : One-to-Many
**relation N-N : Many-to-Many
***Respect la normalisation 3NF (Troisième Forme Normale) : Utilisée en conception de bases de données relationnelles pour structurer les données afin de minimiser la redondance et garantir l'intégrité des données.

Schéma des tables de données et de leurs relations :




```
CREATE TABLE users ( id BIGINT PRIMARY KEY AUTO_INCREMENT, first_name VARCHAR(255)
NOT NULL, last_name VARCHAR(255) NOT NULL, password VARCHAR(255) NOT NULL, address
VARCHAR(255) NOT NULL, country VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL
UNIQUE, phone_number VARCHAR(255) NOT NULL UNIQUE, created_at DATETIME NOT NULL
DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP );

CREATE TABLE agencies ( id BIGINT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255) NOT
NULL, address VARCHAR(255) NOT NULL, city VARCHAR(255) NOT NULL, zip_code INT NOT
NULL, country VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL UNIQUE, phone_number
VARCHAR(255) NOT NULL UNIQUE, opening_hours JSON, status BOOLEAN DEFAULT TRUE,
created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP );

CREATE TABLE ref_vehicle_categories ( id BIGINT PRIMARY KEY AUTO_INCREMENT,
category_name VARCHAR(255) NOT NULL, description TEXT, created_at DATETIME NOT NULL
DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP );

CREATE TABLE agency_vehicles ( id BIGINT PRIMARY KEY AUTO_INCREMENT, agency_id BIGINT
NOT NULL, brand VARCHAR(255) NOT NULL, model VARCHAR(255) NOT NULL, registration
VARCHAR(255) NOT NULL UNIQUE, category_id BIGINT NOT NULL, seat_number INT NOT NULL,
gearbox ENUM('manual', 'automatic'), mileage INT NOT NULL, year INT NOT NULL,
daily_rate DECIMAL NOT NULL, availability BOOLEAN DEFAULT TRUE, created_at DATETIME
NOT NULL DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP, FOREIGN KEY (agency_id) REFERENCES agencies(id), FOREIGN
KEY (category_id) REFERENCES ref_vehicle_categories(id) );

CREATE TABLE vehicle_images ( id BIGINT PRIMARY KEY AUTO_INCREMENT, vehicle_id BIGINT
NOT NULL, image_url VARCHAR(255) NOT NULL, created_at DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP, FOREIGN KEY (vehicle_id) REFERENCES agency_vehicles(id) );

CREATE TABLE user_reservations ( id BIGINT PRIMARY KEY AUTO_INCREMENT, user_id BIGINT
NOT NULL, vehicle_id BIGINT NOT NULL, start_date DATE NOT NULL, end_date DATE NOT
NULL, total_price DECIMAL NOT NULL, payment_status ENUM('confirmed', 'failed'),
status ENUM('confirmed', 'cancelled'), created_at DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP, updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
```

```
CURRENT_TIMESTAMP, FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY
(vehicle_id) REFERENCES agency_vehicles(id) );

CREATE TABLE user_messages ( id BIGINT PRIMARY KEY AUTO_INCREMENT, sender_id BIGINT
NOT NULL, receiver_id BIGINT NOT NULL, message TEXT NOT NULL, status ENUM('unread',
'read'), created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME
DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, FOREIGN KEY (sender_id)
REFERENCES users(id), FOREIGN KEY (receiver_id) REFERENCES users(id) );

CREATE TABLE chatbot_conversations ( id BIGINT PRIMARY KEY AUTO_INCREMENT, user_id
BIGINT NOT NULL, status ENUM('active', 'closed', 'pending'), conversation_id
VARCHAR(255) NOT NULL UNIQUE, external_chatbot_session_id VARCHAR(255) NOT NULL
UNIQUE, created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME
DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, FOREIGN KEY (user_id)
REFERENCES users(id) );
```

---

c. Technologique

Frontend :
Angular 16 pour une interface utilisateur réactive et moderne.

Backend :
Spring Boot 2.7.3 pour les API REST de manière performante et sécurisée.

Base de données :
MySQL 8.1 pour le stockage relationnel des données.

Communication :

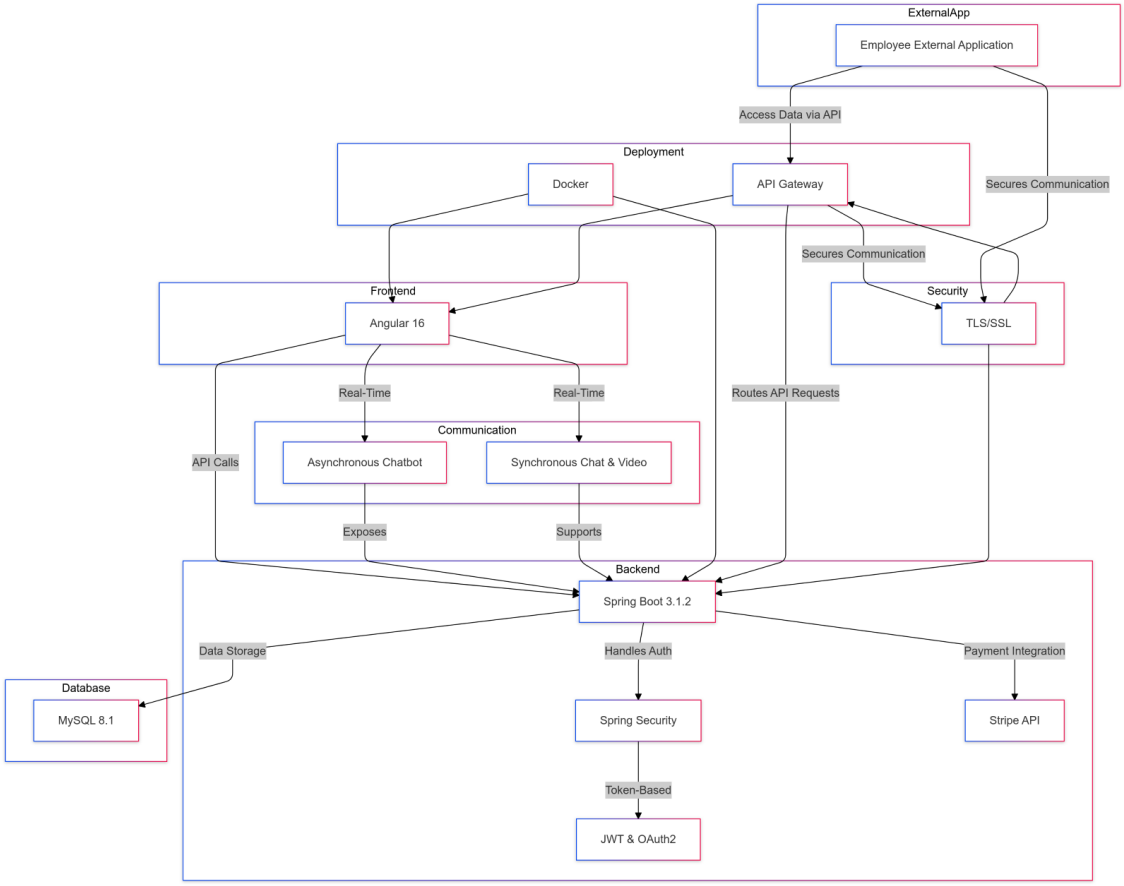
- **Asynchrone** : Chatbot (API REST).
- **Synchrone** : Messagerie (API REST, possibilité d'évolution avec les websockets) et visioconférence (WebRTC).

Authentification et sécurité :
Spring Security avec JWT et OAuth2 pour gérer l'authentification et l'autorisation de manière sécurisée.

Paiements :
Intégration de Stripe pour la gestion des paiements en ligne.

Déploiement :
Docker pour la conteneurisation et la portabilité des services.

Schéma de l’architecture logicielle (modulaire) :



6. Justification de l’approche architecturale

L’approche architecturale retenue répond aux objectifs business et techniques grâce aux éléments suivants :

Cohérence avec les objectifs business :

- Une architecture modulaire permet de centraliser efficacement la gestion des profils, des réservations et des paiements, répondant ainsi aux besoins de simplicité et d’efficacité pour les clients.

- L’interopérabilité offerte par les API RESTful garantit une intégration fluide entre les clients, agences, et systèmes tiers.

- Le support client est amélioré grâce à une gestion claire des communications asynchrones (chatbot) et synchrones (messagerie, visioconférence).

Respect des principes d’architecture :

- **Modularité**
  Chaque composant est conçu indépendamment, facilitant ainsi le développement, les tests, et la maintenance.

- **Scalabilité**
  Les technologies choisies (Spring Boot, Angular, Docker) permettent une montée en charge adaptée à une audience internationale.

- **Interopérabilité**
  Utilisation d’APIs standardisées pour garantir une intégration fluide avec les systèmes tiers et les outils des agences.

- **Expérience utilisateur**
  La rapidité de réponse et une interface intuitive sont des priorités pour garantir la satisfaction des clients.

Sécurité renforcée :

- L’utilisation de Spring Security avec JWT et OAuth2 assure une gestion robuste des droits d’accès, tout en protégeant les données sensibles des utilisateurs.

- L’intégration de Stripe pour les paiements respecte les normes PCI DSS, garantissant ainsi la sécurité des transactions.

Suivi des normes et bonnes pratiques reconnues :

- Respect des réglementations RGPD pour la gestion des données personnelles.
- Conformité avec les standards internationaux de sécurité et de paiement (ex. PCI DSS).
- Adoption des frameworks et technologies modernes largement supportés et documentés (Angular, Spring Boot, Docker).

Robustesse technique et performance :

- Angular 16 et Spring Boot 3.1.2 offrent une base moderne et performante, adaptée aux montées en charge et aux exigences de performance élevées.
- Docker permet une portabilité accrue et facilite le déploiement dans des environnements variés.

Évolutivité et maintenabilité :

- L’architecture modulaire facilite l’ajout de nouvelles fonctionnalités sans impact sur les services existants.
- Les technologies choisies, comme Angular et Spring Boot, sont bien supportées et documentées, ce qui réduit les coûts de maintenance et garantit une évolution continue.

Expérience utilisateur améliorée :

- Les temps de réponse rapides et les interfaces intuitives améliorent la satisfaction des clients.

- La prise en charge des communications temps réel (chat et visioconférence) renforce l'engagement des utilisateurs.

Comparaison entre différentes architectures :

- **Architecture monolithique classique :**
  Bien que simple à déployer et à maintenir, elle manque de flexibilité pour s'adapter aux évolutions futures. Tout changement nécessite un redéploiement complet.

- **Architecture microservices :**
  Idéale pour des systèmes complexes et distribués, elle introduit cependant une complexité importante, avec la gestion de multiples bases de données, la latence des appels entre services et l'orchestration (ex. Kubernetes).

- **Architecture modulaire (choisie) :**
  Combine les avantages du monolithique et des microservices. Les modules sont indépendants dans le code mais partagent une base de données et un déploiement commun, simplifiant ainsi la gestion tout en offrant une bonne évolutivité.

Pourquoi une architecture modulaire est adaptée à ce projet ?

- **Simplicité initiale :**
  Une architecture modulaire centralisée permet de gérer l'ensemble des fonctionnalités (authentification, réservations, paiements, etc.) dans une seule application. Cela réduit la complexité technique par rapport à une architecture microservices.

- **Coût réduit :**
  Avec une architecture monolithique modulaire, les coûts de développement, de déploiement et de maintenance sont plus faibles. Elle évite la surcharge liée à la gestion et à l'orchestration de multiples services.

- **Adaptée à la taille du projet :**
  Pour un projet en phase de développement initial ou avec des fonctionnalités centralisées, une architecture monolithique est suffisante. Une migration vers des microservices peut être envisagée si les besoins évoluent (ex. : augmentation du trafic ou nouvelles fonctionnalités majeures).

- **Rapidité de mise en œuvre :**

Spring Boot permet de structurer facilement une architecture modulaire tout en gardant un déploiement unique. Cela accélère les cycles de développement et réduit les délais de livraison.

7. Architectures de transition

Phase 1 : Fonctionnalités essentielles

- Implémentation des fonctionnalités de base : inscription, connexion, gestion des profils, recherche de véhicules, réservation, et paiement sécurisé.

- Mise en place des API REST pour la communication entre le frontend et le backend.

Phase 2 : Amélioration des interactions client

- Intégration du chatbot pour les communications asynchrones avec les utilisateurs.

- Ajout de l'historique des réservations pour améliorer la transparence.

Phase 3 : Support avancé et scalabilité

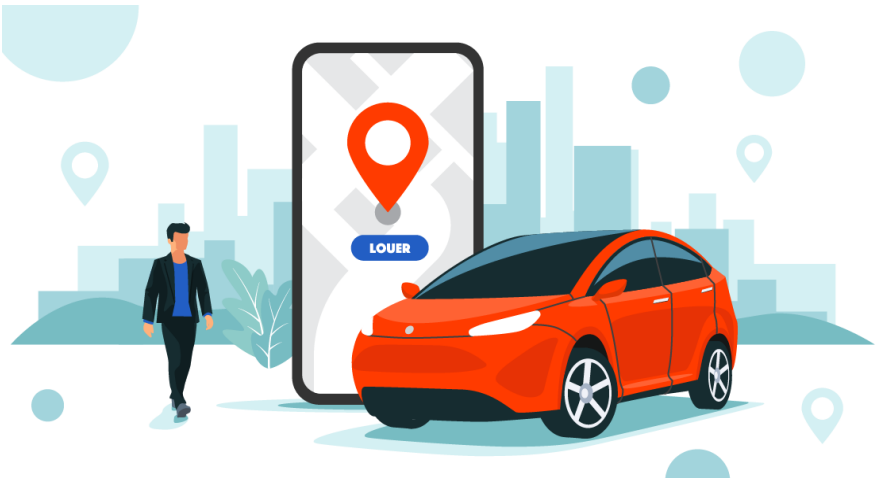
- Déploiement de la messagerie directe et des appels vidéo (visioconférence) pour les interactions synchrones avec le support client.

- Optimisation de la gestion des données avec des stratégies de cache pour améliorer les performances.

Phase 4 : Évolutivité et modularisation

- Préparation à une éventuelle migration vers des microservices si les besoins du projet évoluent (augmentation du trafic ou ajout de nouvelles fonctionnalités).

- Ajout de tests automatisés (unitaires, d'intégration, et E2E) pour renforcer la qualité et la fiabilité du système.



YOUR CAR
YOUR WAY

Compliance Assessment

1. Description générale de l’architecture 2
2. Liste de contrôle de l’architecture terminée 3
3. Services ou composants tiers logiciels 4
4. Gestion des données 5
5. Infrastructure 6
6. Sécurité 7

1. Description générale de l’architecture

L’architecture se repose sur une architecture modulaire centralisée, s’appuyant sur Angular 16 pour le front-end, Spring Boot 3.1.2 pour le back-end, et MySQL 8.1 pour la base de données relationnelle. Les interactions entre les modules sont basées sur des APIs RESTful. L’authentification est assurée par Spring Security avec JWT et OAuth2. Les paiements en ligne utilisent l’intégration de Stripe, et le déploiement est effectué via Docker.

2. Liste de contrôle de l’architecture terminée

Gestion des utilisateurs :

- **Contrôle :**
  Vérifier que les utilisateurs peuvent s’inscrire, se connecter, et modifier leur profil facilement.

- **Critère :**
  Les utilisateurs doivent pouvoir s’inscrire et se connecter sans erreur, avec un taux de succès supérieur à 99%.

Réservation de services :

- **Contrôle** :
  Vérifier que la recherche et la réservation de services (comme la location de véhicules) fonctionne correctement.

- **Critère :**
  La recherche doit afficher les bonnes options (ville, date, heure, catégorie) et se mettre à jour rapidement (en moins de 1 seconde).

Support client :

- **Contrôle :**
  Vérifier que le chatbot et le support en ligne répondent correctement aux demandes des utilisateurs.

- **Critère :**
  Le chatbot doit résoudre au moins 80% des demandes sans intervention humaine.

Performances :

- **Contrôle :**
  Vérifier que l’application répond rapidement, même en cas de forte utilisation.

- **Critère :**
  Le temps de réponse des services API ne doit pas dépasser 200 millisecondes.

3. Services ou composants tiers logiciels

Paiements avec Stripe :

- **Contrôle :**
  Vérifier l'intégration du système de paiement Stripe et qu'il respecte les normes de sécurité.

- **Critère :**
  Les paiements doivent être traités sans erreur et en toute sécurité.

APIs RESTful :

- **Contrôle :**
  Vérifier que les communications entre le frontend et le backend (via les APIs) sont fiables.

- **Critère :**
  Les appels API doivent être réussis dans 100% des cas.

4. Gestion des données

Base de données MySQL :

- **Contrôle :**
  Vérifier que la base de données est bien structurée et que les données sont stockées correctement.

- **Critère :**
  Les opérations (ajout, modification, suppression) doivent fonctionner sans erreur.

Respect de la confidentialité (RGPD) :

- **Contrôle :**
  Vérifier que les données personnelles des utilisateurs sont protégées et peuvent être supprimées sur demande.

- **Critère :**
  Les données sensibles doivent être chiffrées et supprimées en 24 heures si demandé par l'utilisateur.

5. Infrastructure

Docker :

- **Contrôle :**
  Vérifier que l'application peut être facilement déployée dans différents environnements (test et production).

- **Critère :**
  Les déploiements doivent réussir sans ajustement de configuration.

Scalabilité :

- **Contrôle :**
  Vérifier que l'application peut supporter une augmentation du nombre d'utilisateurs sans ralentir.

- **Critère :**
  L'application doit supporter deux fois plus d'utilisateurs sans perte de performance.

6. Sécurité

Authentification et sécurité des données :

- **Contrôle :**
  Vérifier que l'authentification avec JWT fonctionne correctement et que les données sensibles sont protégées.

- **Critère :**
  Tous les utilisateurs doivent être correctement authentifiés avec JWT, et les communications doivent être sécurisées avec HTTPS.

Respect des normes de sécurité (PCI DSS pour les paiements) :

- **Contrôle :**
  Vérifier que les paiements sont traités selon les normes de sécurité PCI DSS.

- **Critère :**
  Tous les paiements doivent respecter les règles de sécurité sans incident.

Your Car Your Way

Made with Gamma

Introduction

Made with Gamma

Présentation de Your Car Your Way

- Entreprise internationale de location de voitures, active depuis **20 ans**.
- Applications actuelles **obsolètes**, ne répondant plus aux exigences des clients et aux nouvelles contraintes techniques.

Objectif de la présentation

- Expliquer et **justifier les choix architecturaux** en fonction des besoins métiers et techniques.
- Présenter les fonctionnalités clés et la **démonstration du POC**.

Made with Gamma

Objectifs du projet

Made with Gamma

Pourquoi une refonte ?

1

Améliorer l’expérience utilisateur

Interface intuitive, fluide et accessible à l’international.

2

Faciliter la gestion des locations

Centralisation des informations des agences et véhicules.

3

Optimiser la rentabilité

Stratégie d’annulation claire et paiements sécurisés via Stripe.

4

Favoriser l’interaction avec les agences

API pour mise à jour des offres en temps réel.

5

Renforcer le support client

Chatbot et service de messagerie/visioconférence.

6

Garantir la sécurité et la conformité

Respect du **RGPD** et des normes **PCI DSS**.

Made with Gamma

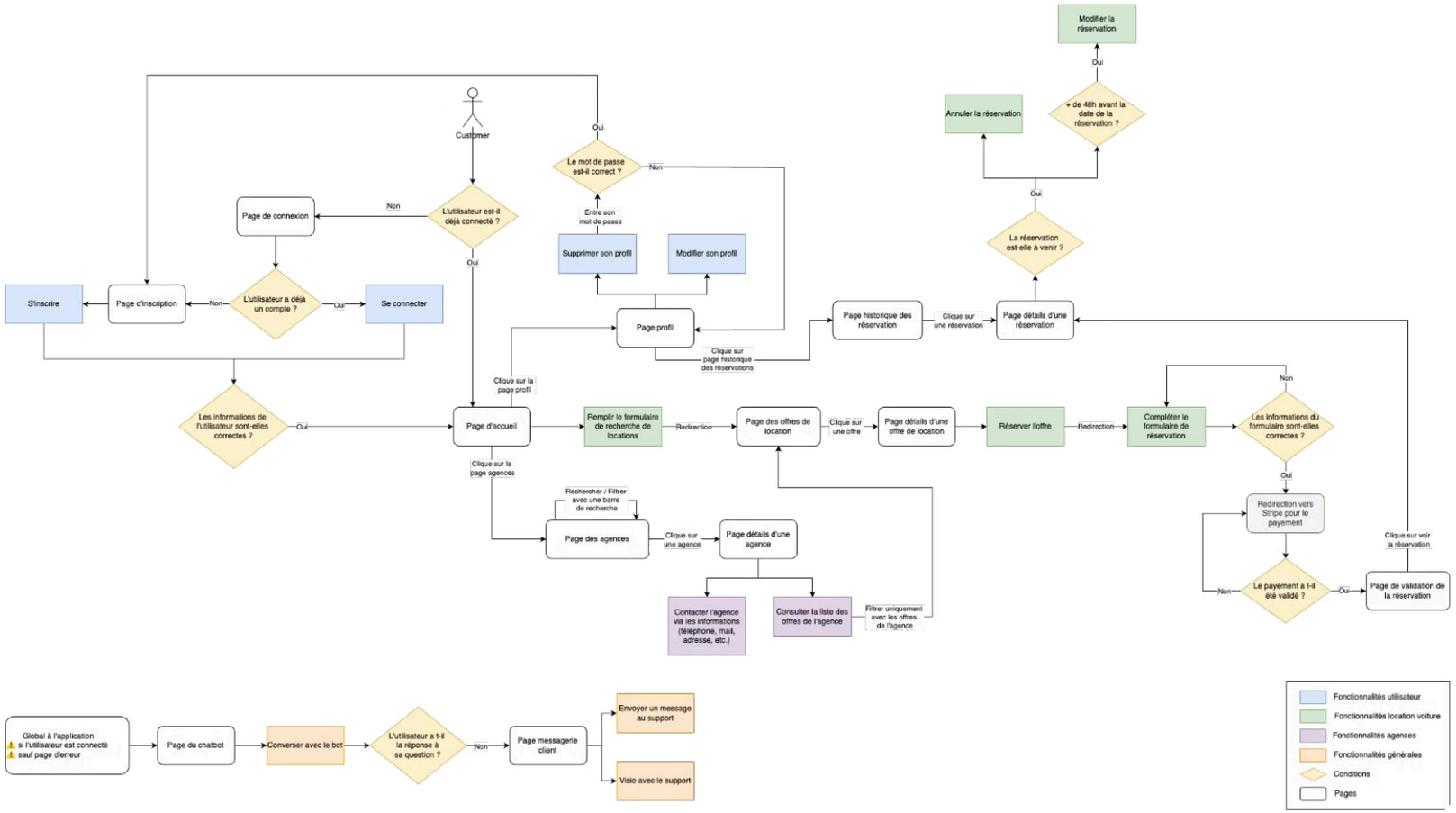
Exigences

- **Modification des réservations jusqu’à 48h avant le début**.
- **Remboursement limité à 25%** si annulation à moins d’une semaine.
- **Norme ACRISS** pour la classification des véhicules.
- **API en agence** pour gérer les réservations et utilisateurs via opérations CRUD.

Les **normes ACRISS** (**Association of Car Rental Industry Systems Standards**) sont un ensemble de critères utilisés dans l'industrie de la location de voitures pour **standardiser la classification des véhicules**. Elles attribuent des codes à différents types de voitures en fonction de leur **taille**, de leur **capacité**, de leur **type de transmission**, de leur **carburant** et d'autres caractéristiques spécifiques.

Made with Gamma

Fonctionnalités Clés



Gestion des profils

Connexion, inscription, modification, suppression.

Gestion des agences

Recherche, consultation des détails.

Location de véhicules

Recherche avancée, réservation et paiement.

Support client

Chatbot et assistance par messagerie/visioconférence.

Made with Gamma

Architecture Technique et Technologies

Made with Gamma

Technologies

Frontend 🎨

Angular 16 → Interface fluide et interactive.

Backend 👩‍💻

Spring Boot 2.7.3 → Performances élevées et scalabilité.

Base de données 💽

MySQL 8.1 → Stockage relationnel optimisé.

Sécurité 🔐

Spring Security, JWT, OAuth2 → Gestion sécurisée des sessions.

Déploiement 💻

Docker → Portabilité et homogénéité des environnements.

Made with Gamma

Principes d'architecture appliqués

Les **principes d’architecture** définissent les règles essentielles pour concevoir un système **efficace, évolutif et cohérent**. Ils guident les choix techniques afin d’assurer **performance, sécurité et facilité de maintenance**.

1

Modularité

Composants indépendants (profils, réservations, paiements).

2

Scalabilité

Support de montée en charge et pics de trafic.

3

Interopérabilité

APIs RESTful pour connecter agences et partenaires.

4

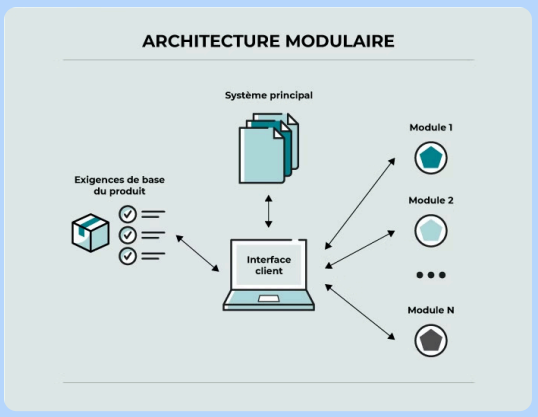
Sécurité

Cryptage des données et conformité PCI DSS/RGPD.

Le **PCI DSS** (**Payment Card Industry Data Security Standard**) est un ensemble de normes de sécurité pour **protéger les informations** des cartes de paiement.

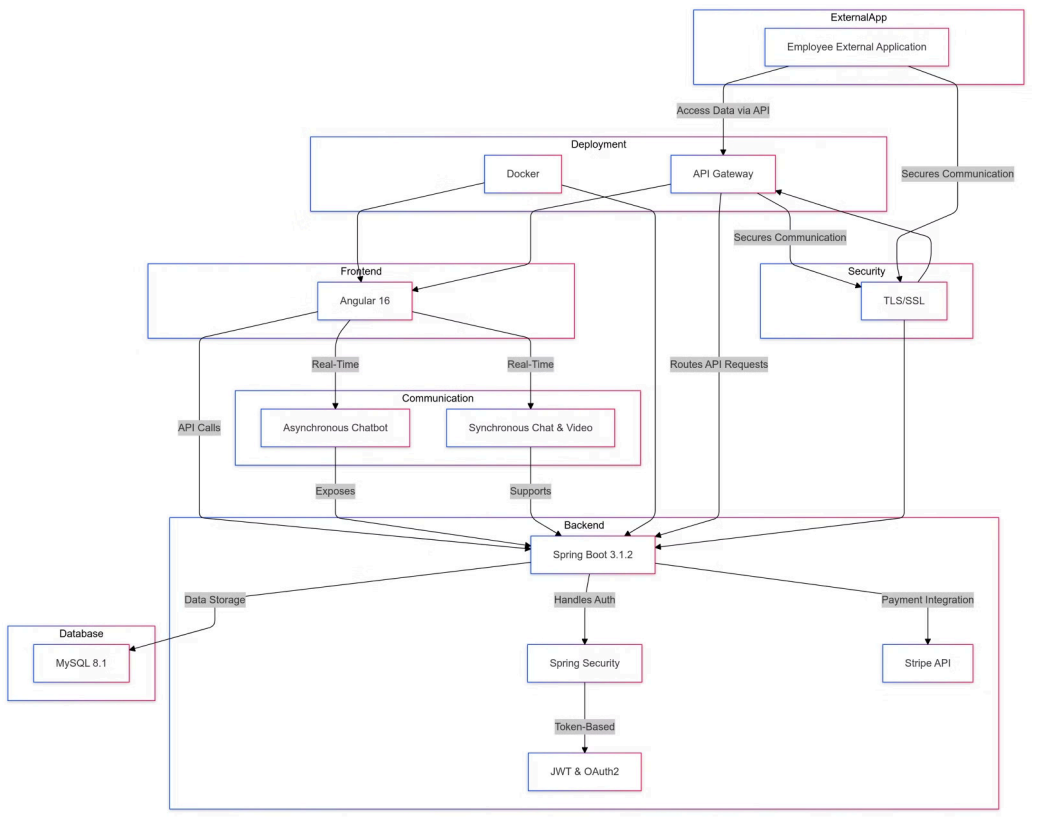
Made with Gamma

Justification du Choix de l’Architecture Modulaire



Une **architecture modulaire** divise l’application en plusieurs **blocs indépendants**, facilitant **le développement, la maintenance et l’évolution** du système. Chaque module remplit une fonction précise tout en restant connecté au système principal.

Notre architecture



Notre architecture se repose bien sur les **quatre parties essentielles** :

1. **Exigences de base** : Fonctions minimales essentielles (authentification, gestion des réservations, paiements sécurisés).
2. **Système principal** : Backend **Spring Boot** pour orchestrer les données, l’authentification et la communication entre les modules.
3. **Interface client** : Frontend **Angular**, garantissant une interaction fluide et intuitive pour l’utilisateur.
4. **Modules complémentaires** :
   - **Chatbot asynchrone** pour le support client.
   - **Chat & visioconférence** pour une assistance en temps réel.
   - **Paiement sécurisé** via Stripe API.

Pourquoi une architecture modulaire ?

Alignement avec les objectifs business

- Centralisation et accessibilité des profils, réservations et paiements.
- APIs RESTful garantissant l’interopérabilité entre agences et clients.
- Expérience client améliorée avec des services en temps réel (chat, vidéo).

Respect des principes d’architecture

- **Modularité** → Développement et maintenance facilités.
- **Scalabilité** → Technologies adaptées à la croissance.
- **Interopérabilité** → APIs standardisées pour une communication fluide.
- **Performance** → Temps de réponse inférieur à **200 ms**.

Sécurité et conformité

- Gestion des accès via **JWT & OAuth2**.
- Transactions Stripe sécurisées et conformes **PCI DSS**.
- Respect du **RGPD** pour la gestion des données personnelles.

Évolutivité et maintenabilité

- Possibilité **d’ajouter de nouvelles fonctionnalités** sans impacter l’existant.
- Technologies modernes et largement documentées.

Made with Gamma

Comparaison avec d’autres Architectures



Architecture Monolithique Classique

✅ Facile à **developper** et à **maintenir** au départ.

❌ Manque de **flexibilité** → tout changement nécessite un **redéploiement complet**.



Architecture Microservices

Excellente **évolutivité** et **indépendance** des services.

❌ Trop **complexe** pour ce projet → multiples bases de données, orchestration difficile.



Architecture Modulaire (choisie)

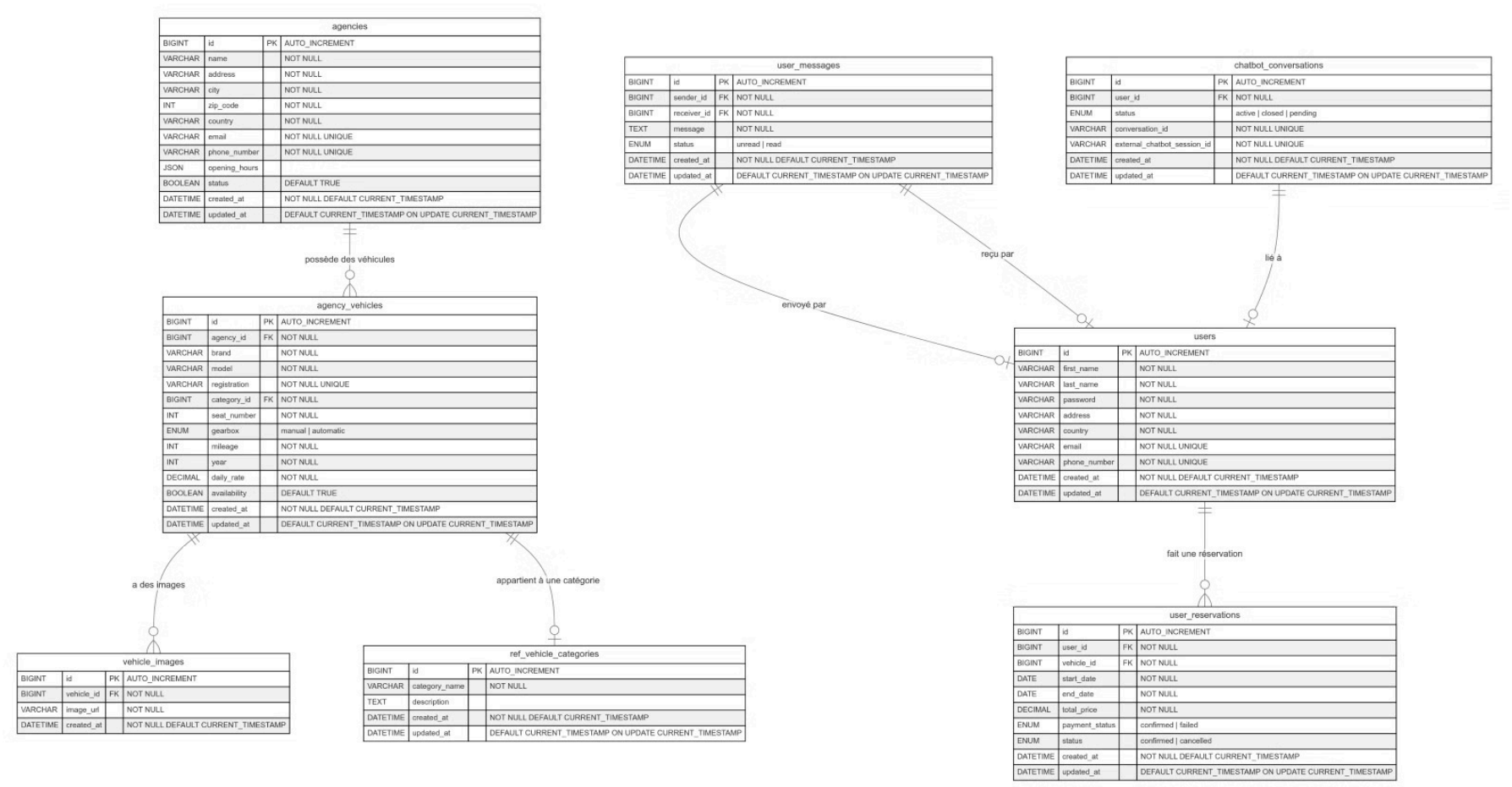
Simplicité du monolithique + **flexibilité** des microservices.

Modules **indépendants** partageant une base de données et un **déploiement unique.**

✅ **Évolutivité** garantie sans complexité excessive.

Made with Gamma

Structuration de la Base de Données



Pourquoi une base bien structurée ?

- Assure la **cohérence et l'intégrité** des données.
- Optimise **les requêtes et la scalabilité**.
- Permet une **intégration fluide** avec les agences via API.

Schéma de la base de données

- Tables principales : **Utilisateurs, Agences, Véhicules, Réservations, Paiements, Messages**.
- **Respect de la normalisation 3NF** → Évite la redondance et garantit des performances optimales.
- API RESTful permettant aux agences d'accéder aux données via **CRUD**.

La **normalisation 3NF (Troisième Forme Normale)** structure les bases de données pour **éviter la redondance** et garantir **l'intégrité des données**. Elle améliore la **cohérence**, **optimise** les requêtes et facilite la **maintenance**.

Made with Gamma

Présentation du POC

Made with Gamma

Conclusion

Pourquoi cette architecture est le meilleur choix ?

- **Répond aux besoins business et techniques** (centralisation, interopérabilité, sécurité).
- **Évolutive** → possibilité de transition vers une architecture microservices si nécessaire.
- **Optimisation des coûts et des délais de développement**.

Perspectives d'évolution

- Ajout de **nouvelles fonctionnalités** et optimisation des performances.
- Renforcement de la sécurité avec **tests unitaires et d'intégrations**.
- Migration potentielle vers **microservices** en fonction de la croissance.

Made with Gamma

Avez-vous des questions ?



Made with Gamma