

YOUR CAR YOUR WAY

ARCHITECTURE DEFINITION DOCUMENT

Cahier des charges technique

Version	2.0
Date	01 février 2026
Auteur	Arnaud DERISBOURG
Statut	Version révisée

Historique des versions

Version	Date	Modifications
1.0	30/01/2026	Version initiale
2.0	01/02/2026	Suppression réf. hexagonale, ajout diagrammes UML, traçabilité BR, 3NF, WebSocket

Sommaire

Sommaire	2
Objet du document	3
Objectifs du projet.....	3
Principes d'architecture.....	3
Architecture métier.....	4
Acteurs du système.....	4
Diagramme de classes UML (approche DDD)	4
Architecture de données	6
Conformité aux formes normales (3NF)	6
Modèle Physique de Données (MPD)	6
Relations entre les entités.....	7
Détail des tables.....	7
Table users.....	8
Table conversations	8
Table messages	8
Table rentals (hors PoC).....	8
Table agencys (hors PoC)	9
Architecture technique	10
Stack technologique.....	10
Diagramme de composants UML	10
Architecture en couches (Backend).....	11
Flux d'authentification JWT (BR-AUTH-02)	12
Architecture WebSocket — Tchat temps réel (BR-SUP-02).....	12
Architecture Visioconférence WebRTC (BR-SUP-03) — V2	13
Infrastructure et déploiement	14
Diagramme de déploiement UML.....	14
Configuration Docker Compose	14
Traçabilité : Contraintes client → Choix techniques	16
Justification de l'architecture.....	17
Choix de l'architecture en couches.....	17
Choix de WebSocket + STOMP pour le temps réel.....	17
Choix de JWT pour l'authentification	17
Faisabilité du projet	17

Objet du document

Ce document décrit l'architecture technique de la solution Your Car Your Way. Il définit les choix technologiques, les composants logiciels, le modèle de données et l'infrastructure de déploiement nécessaires à la réalisation du projet.

Ce document complète le Business Requirements (BR) et sert de référence pour l'implémentation. Chaque choix technique est tracé vers une exigence du BR (identifiants BR-xxx) afin d'assurer la cohérence entre les livrables.

Périmètre du PoC : la preuve de concept se concentre sur le système de support client (authentification + messagerie + tchat temps réel + chatbot IA). Les fonctionnalités hors PoC (locations, paiement) sont décrites pour la cohérence globale mais ne sont pas implémentées.

Objectifs du projet

Les objectifs de l'application Your Car Your Way sont définis dans le BR. L'architecture doit répondre aux besoins suivants :

Réf. BR	Objectif	Périmètre
BR-AUTH-01/02	Permettre l'inscription, la connexion et la déconnexion sécurisée	PoC
BR-SUP-01	Fournir une messagerie asynchrone client-support	PoC
BR-SUP-02	Offrir un tchat en temps réel (WebSocket) sans rechargement de page	PoC
BR-SUP-03	Prévoir une visioconférence client-support (WebRTC)	Décriv — V2
BR-SUP-04	Intégrer un chatbot IA pour l'assistance automatisée	PoC
BR-LOC-01 à 07	Gestion des locations (recherche, réservation, paiement, historique)	Hors PoC
BR-PROF-01 à 03	Gestion du profil utilisateur	Hors PoC

Principes d'architecture

L'architecture de la solution repose sur les principes suivants. Chaque principe est justifié par une contrainte issue du BR.

Principe	Description	Réf. BR
Architecture en couches	Séparation stricte des responsabilités : Controller (exposition REST) → Service (logique métier) → Repository (accès données). Ce pattern garantit la maintenabilité et la testabilité du code.	Toutes
API REST stateless	Communication frontend-backend en HTTP/JSON. Chaque requête est autonome (pas de session serveur). Permet l'accès par les applications en agence (exigence API du BR).	BR — Exigences API

Authentification JWT	Tokens JSON Web Token pour l'authentification stateless. Compatible avec l'architecture REST et la scalabilité horizontale.	BR-AUTH-02
WebSocket (STOMP)	Communication bidirectionnelle temps réel pour le tchat. Le protocole STOMP sur WebSocket permet le push serveur sans polling, garantissant la réactivité < 2 secondes exigée.	BR-SUP-02
BDD relationnelle	PostgreSQL avec intégrité référentielle (FK), conformité 3NF. Garantit la cohérence des données (utilisateurs, conversations, messages).	Toutes
Sécurité en profondeur	Mots de passe hashés BCrypt, tokens JWT signés, CORS configuré, endpoints publics limités, validation des comptes employés par admin.	BR-AUTH-01, RM-05
Conteneurisation	Docker Compose pour la reproductibilité de l'environnement (base de données, outils d'administration). Déploiement en une commande.	Exigence technique
Paiement externalisé	Externalisation du paiement auprès de Stripe (conformité PCI-DSS déléguée). Aucune donnée bancaire stockée côté serveur.	BR-LOC-04, RM-07

Architecture métier

Acteurs du système

Acteur	Responsabilités	Réf. BR
Client (USER)	Crée un compte, s'authentifie, envoie des messages au support, utilise le tchat temps réel, interagit avec le chatbot. Hors PoC : recherche et réserve des véhicules.	BR-AUTH-*, BR-SUP-*, BR-LOC-*, BR-PROF-*
Employé (EMPLOYEE)	Répond aux conversations clients via messagerie et tchat temps réel. Son compte doit être validé par un administrateur avant de pouvoir se connecter.	BR-SUP-01, BR-SUP-02
Admin (ADMIN)	Valide ou rejette les demandes de comptes employés. Administration générale du système.	BR-AUTH-01 (RM-05)

Diagramme de classes UML (approche DDD)

Le diagramme de classes ci-dessous modélise le domaine métier selon une approche Domain-Driven Design. Les entités du périmètre PoC sont en vert, les entités hors PoC en gris.

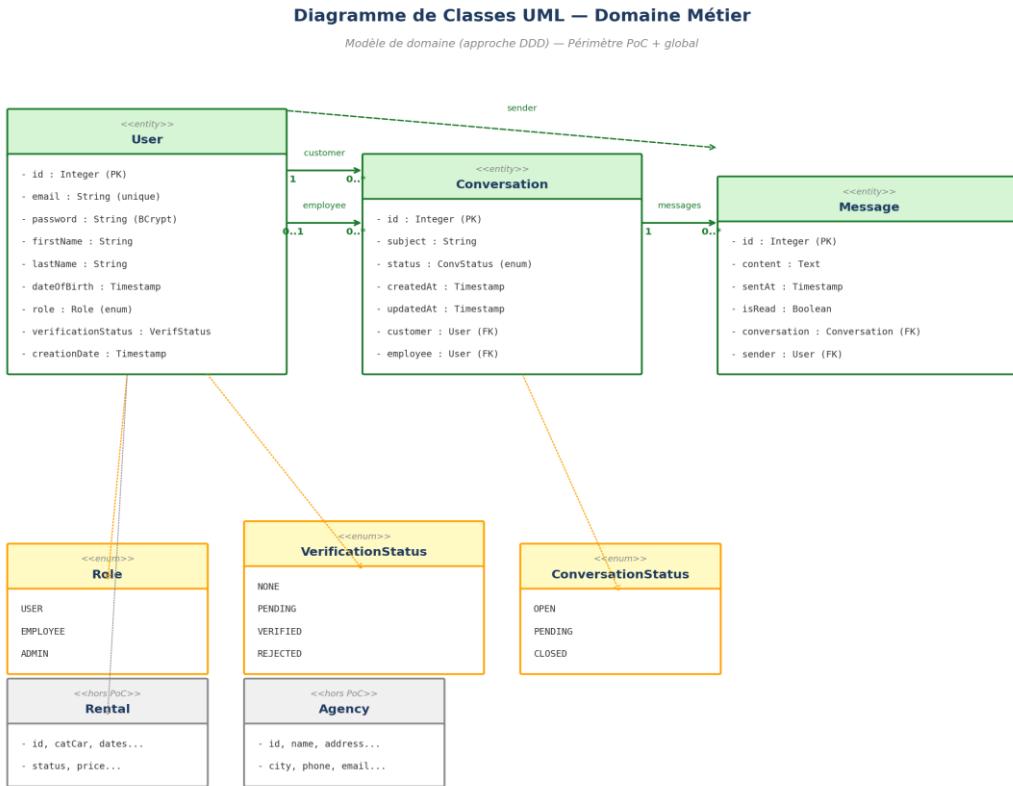


Figure 1 : Diagramme de classes UML — Modèle de domaine (approche DDD)

Les entités du domaine et leurs responsabilités :

- User : représente un utilisateur du système (client, employé ou administrateur). Porte l'identité, l'authentification et le rôle. [PoC]
- Conversation : échange entre un client et un employé du support. Porte le sujet, le statut et les timestamps. [PoC]
- Message : unité de communication au sein d'une conversation. Porte le contenu, l'horodatage et l'état de lecture. [PoC]
- Rental : représente une location de véhicule avec dates, prix et statut. [Hors PoC]
- Agency : agence physique de location. [Hors PoC]

Architecture de données

Conformité aux formes normales (3NF)

Le schéma de base de données respecte la troisième forme normale (3NF). Voici la démonstration pour chaque table :

Table	1NF — Valeurs atomiques	2NF — Dépendance totale de la PK	3NF — Pas de dépendance transitive
users	✓ Chaque colonne contient une valeur atomique (email, firstName, lastName séparés)	✓ Tous les attributs dépendent entièrement de la PK (id)	✓ Aucun attribut non-clé ne dépend d'un autre attribut non-clé
conversations	✓ Valeurs atomiques (subject, status sont des scalaires)	✓ Tous les attributs dépendent de id. customer_id et employee_id sont des FK	✓ Pas de transitivité : status dépend de la conversation, pas du client
messages	✓ content est de type TEXT (atomique), pas de tableau	✓ Tous les attributs dépendent de id. conversation_id et sender_id sont des FK	✓ Pas de transitivité : sentAt dépend du message, pas de la conversation
rentals	✓ Chaque colonne est scalaire (prix en INTEGER, dates en TIMESTAMP)	✓ Tous les attributs dépendent de id. Les FK (user_id, agency_id) pointent vers d'autres tables	✓ Pas de transitivité : le prix dépend de la location, pas de l'agence
agency	✓ Adresse décomposée (address, city, country, postalCode)	✓ Tous les attributs dépendent de id	✓ Pas de transitivité : city ne dépend pas de postalCode (au niveau international)

Modèle Physique de Données (MPD)

Le MPD ci-dessous illustre les tables et leurs relations. Il a été généré depuis la base de données PostgreSQL 15 via pgAdmin.

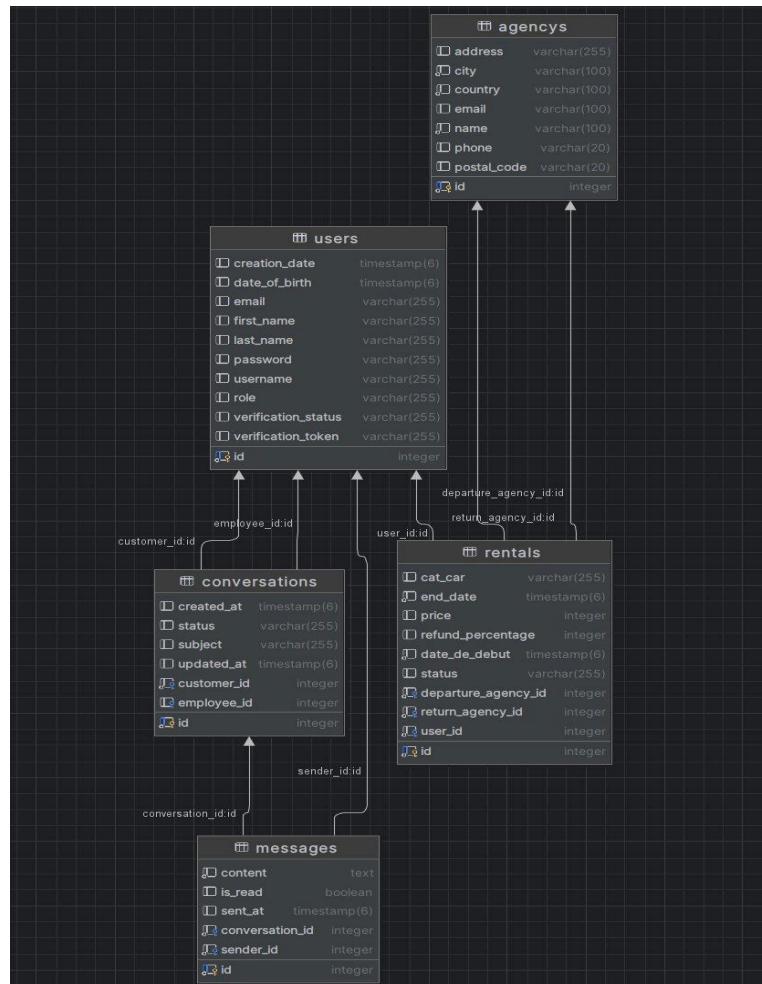


Figure 2 : Modèle Physique de Données — Your Car Your Way

Relations entre les entités

Relation	Cardinalité	Cle étrangère
users → rentals	1..* (un user, plusieurs locations)	rentals.user_id → users.id
agencys → rentals (départ)	1..* (une agence, plusieurs départs)	rentals.departure_agency_id → agencys.id
agencys → rentals (retour)	1..* (une agence, plusieurs retours)	rentals.return_agency_id → agencys.id
users → conversations (client)	1..* (un client, plusieurs conversations)	conversations.customer_id → users.id
users → conversations (employé)	0..1..* (un employé, plusieurs conversations)	conversations.employee_id → users.id
conversations → messages	1..* (une conversation, plusieurs messages)	messages.conversation_id → conversations.id
users → messages (sender)	1..* (un user, plusieurs messages envoyés)	messages.sender_id → users.id

Détail des tables

Table users

Colonne	Type	Description
id	INTEGER (PK)	Identifiant auto-incrémenté
email	VARCHAR(255)	Email unique — identifiant de connexion
password	VARCHAR(255)	Mot de passe hashé BCrypt (jamais en clair — RM-05)
first_name	VARCHAR(255)	Prénom
last_name	VARCHAR(255)	Nom
username	VARCHAR(255)	Nom d'affichage
role	VARCHAR(255)	Enum : USER, EMPLOYEE, ADMIN
verification_status	VARCHAR(255)	Enum : NONE, PENDING, VERIFIED, REJECTED
verification_token	VARCHAR(255)	Token de vérification email (usage unique)
date_of_birth	TIMESTAMP(6)	Date de naissance
creation_date	TIMESTAMP(6)	Date de création du compte

Table conversations

Colonne	Type	Description
id	INTEGER (PK)	Identifiant auto-incrémenté
subject	VARCHAR(255)	Sujet de la conversation
status	VARCHAR(255)	Enum : OPEN, PENDING, CLOSED
created_at	TIMESTAMP(6)	Date de création
updated_at	TIMESTAMP(6)	Date de dernière mise à jour
customer_id	INTEGER (FK)	Référence → users.id (client)
employee_id	INTEGER (FK)	Référence → users.id (employé support)

Table messages

Colonne	Type	Description
id	INTEGER (PK)	Identifiant auto-incrémenté
content	TEXT	Contenu du message
sent_at	TIMESTAMP(6)	Date et heure d'envoi
is_read	BOOLEAN	Indicateur de lecture
conversation_id	INTEGER (FK)	Référence → conversations.id
sender_id	INTEGER (FK)	Référence → users.id (expéditeur)

Table rentals (hors PoC)

Colonne	Type	Description
id	INTEGER (PK)	Identifiant auto-incrémenté

cat_car	VARCHAR(255)	Code catégorie ACRISS (RM-03)
date_de_debut	TIMESTAMP(6)	Date/heure de début de location
end_date	TIMESTAMP(6)	Date/heure de fin de location
price	INTEGER	Prix en centimes d'euros
status	VARCHAR(255)	BOOKED, IN_PROGRESS, COMPLETED, CANCELLED
refund_percentage	INTEGER	Pourcentage de remboursement (RM-02)
user_id	INTEGER (FK)	Référence → users.id
departure_agency_id	INTEGER (FK)	Référence → agencys.id (départ)
return_agency_id	INTEGER (FK)	Référence → agencys.id (retour)

Table agencys (hors PoC)

Colonne	Type	Description
id	INTEGER (PK)	Identifiant auto-incrémenté
name	VARCHAR(100)	Nom de l'agence
address	VARCHAR(255)	Adresse postale
city	VARCHAR(100)	Ville
country	VARCHAR(100)	Pays
postal_code	VARCHAR(20)	Code postal
phone	VARCHAR(20)	Numéro de téléphone
email	VARCHAR(100)	Email de contact

Architecture technique

Stack technologique

Couche	Technologie	Version	Justification (réf. BR)
Frontend	Angular (Standalone)	19.x	SPA réactive, lazy loading (BR-SUP-02)
Backend	Spring Boot	4.0.0	Écosystème complet (security, websocket, mail)
Langage	Java	21 (LTS)	Support long terme, typage fort
Base de données	PostgreSQL	15	Intégrité référentielle 3NF
ORM	JPA / Hibernate	6.x	Mapping objet-relationnel automatisé
Sécurité	Spring Security + JWT	java-jwt 4.4.0	Auth stateless (BR-AUTH-02)
Temps réel	Spring WebSocket + STOMP	Spring 6.x	Chat sans refresh (BR-SUP-02)
Chatbot	OpenAI API	4.8.0	Assistance 24/7 (BR-SUP-04)
Paiement	Stripe API	24.10.0	Paiement externalisé (RM-07) — Hors PoC
Doc API	SpringDoc OpenAPI	2.8.14	API documentée Swagger (BR — Exig. API)
PDF	OpenPDF	2.0.2	Factures PDF (BR-LOC-04) — Hors PoC
Email	Spring Mail (SMTP)	Spring 6.x	Vérification compte (BR-AUTH-01)

Diagramme de composants UML

Le diagramme de composants ci-dessous montre l'organisation des composants logiciels et leurs interactions. Les composants du PoC sont en vert.

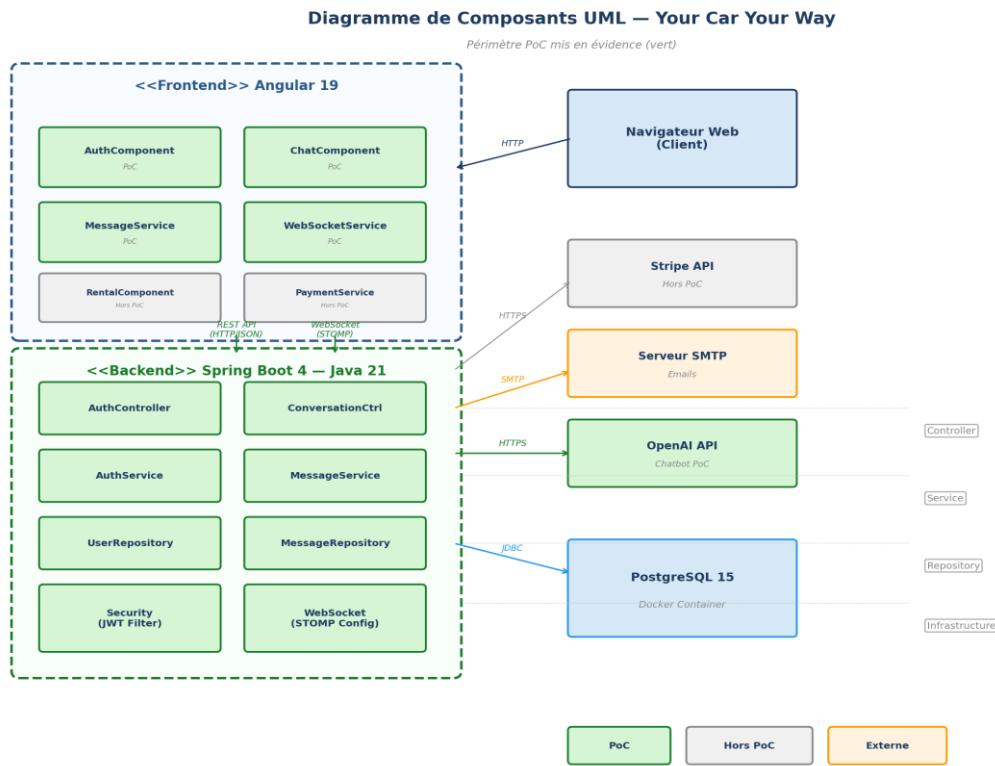


Figure 3 : Diagramme de composants UML — Organisation des composants logiciels

Architecture en couches (Backend)

Le backend suit une architecture en couches stricte. Chaque couche a une responsabilité unique et ne communique qu'avec la couche adjacente :

Couche	Responsabilité	Composants PoC
Controller	Expose les endpoints REST (@RestController). Valide les requêtes entrantes, délègue au Service et retourne les DTOs. Ne contient aucune logique métier.	AuthController, ConversationCtrl, MessageCtrl, ChatCtrl
Service	Implémente la logique métier. Orchestre les appels aux Repositories et applique les règles de gestion (RM-01 à RM-07).	AuthService, ConversationService, MessageService, ChatService
Repository	Accès aux données via Spring Data JPA. Chaque Repository correspond à une entité du domaine.	UserRepository, ConversationRepo, MessageRepo
Mapper	Conversion bidirectionnelle Entity ↔ DTO via MapStruct. Isole le modèle de domaine de l'API exposée.	UserMapper, ConversationMapper, MessageMapper
Security	Intercepte chaque requête HTTP pour valider le token JWT (JwtAuthenticationFilter). Configure CORS et les endpoints publics/protégés.	JwtFilter, SecurityConfig, CustomUserDetailsService
WebSocket	Configure le broker STOMP pour la communication temps réel. Gère les souscriptions par conversation et le broadcast des messages.	WebSocketConfig, MessageWebSocketCtrl

Exception	Gestion centralisée des erreurs via GlobalExceptionHandler. Retourne des réponses d'erreur structurées (code, message).	GlobalExceptionHandler, BusinessException
------------------	---	---

Flux d'authentification JWT (BR-AUTH-02)

Le diagramme de séquence ci-dessous détaille le flux complet d'authentification, de la saisie des identifiants jusqu'à l'accès aux ressources protégées.

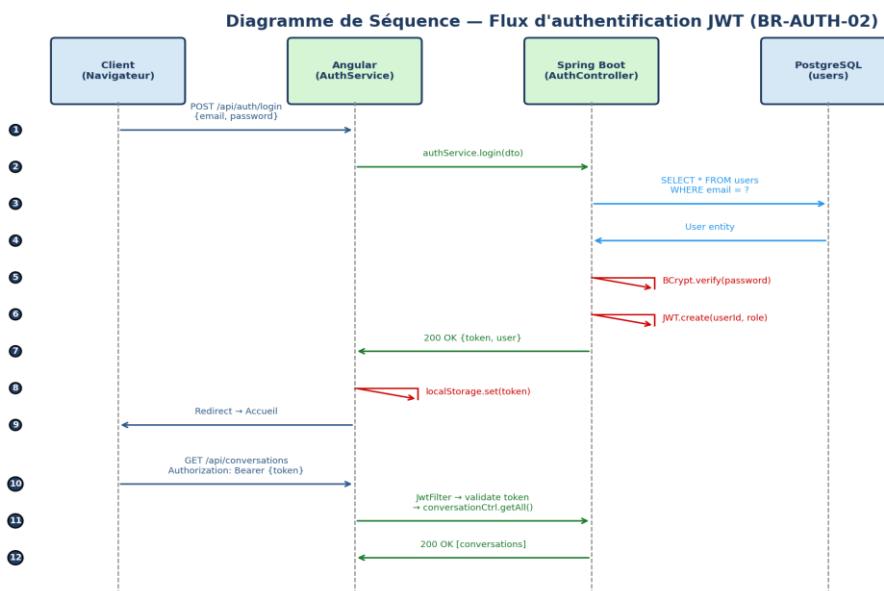


Figure 4 : Diagramme de séquence — Flux d'authentification JWT

Détail du flux :

- Le client soumet ses identifiants (email + mot de passe) via POST /api/auth/login
- Le AuthController délègue au AuthService qui interroge la base de données
- Le mot de passe est vérifié avec BCrypt.verify() (jamais de comparaison en clair)
- Si valide, un token JWT est généré avec les claims (userId, role, expiration)
- Le token est renvoyé au client qui le stocke localement
- Pour chaque requête suivante, le token est envoyé dans le header Authorization: Bearer {token}
- Le JwtAuthenticationFilter intercepte la requête, valide le token et autorise l'accès

Architecture WebSocket — Tchat temps réel (BR-SUP-02)

Le tchat temps réel est implémenté avec Spring WebSocket et le protocole STOMP (Simple Text Oriented Messaging Protocol). Ce choix permet le push serveur vers les clients sans polling, garantissant une latence inférieure à 2 secondes comme exigé par le BR-SUP-02.

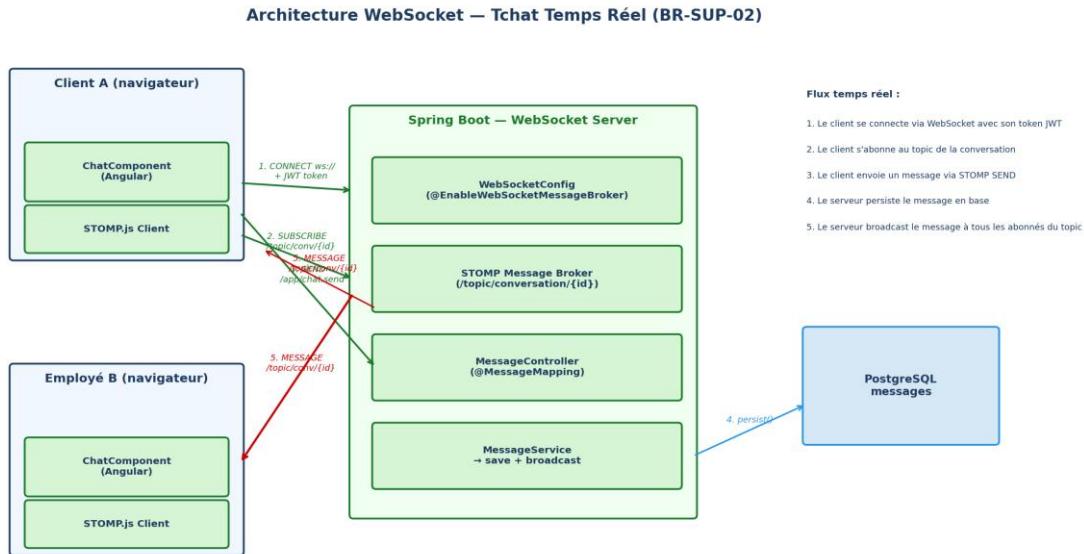


Figure 5 : Architecture WebSocket — Communication temps réel (BR-SUP-02)

Fonctionnement détaillé :

- Connexion : le client établit une connexion WebSocket (`ws://`) en fournissant son token JWT pour l'authentification
- Souscription : le client s'abonne au topic `/topic/conversation/{conversationId}` correspondant à sa conversation
- Envoi : le client envoie un message via STOMP SEND vers `/app/chat.send`
- Traitement : le MessageController reçoit le message, le Service le persiste en base via le Repository
- Broadcast : le broker STOMP diffuse le message à tous les clients abonnés au topic de la conversation
- Réception : le message apparaît instantanément dans les fenêtres de tchat de tous les participants (sans rechargement)

Architecture Visioconférence WebRTC (BR-SUP-03) — V2

Note : cette fonctionnalité est décrite architecturalement pour la version 2 de l'application. Elle n'est pas implémentée dans le PoC.

L'architecture prévue pour la visioconférence repose sur WebRTC (Web Real-Time Communication) :

- Signalisation : échange des offres SDP et candidats ICE via le canal WebSocket déjà en place (BR-SUP-02)
- Flux média : communication pair-à-pair (peer-to-peer) directe entre les navigateurs des participants
- Le serveur ne relaye pas les flux vidéo/audio — il sert uniquement de relais de signalisation
- Prérequis : les deux participants doivent autoriser l'accès caméra et microphone via l'API `navigator.mediaDevices`

Infrastructure et déploiement

Diagramme de déploiement UML

Le diagramme de déploiement ci-dessous montre les nœuds physiques et logiques de l'infrastructure, ainsi que les protocoles de communication entre eux.

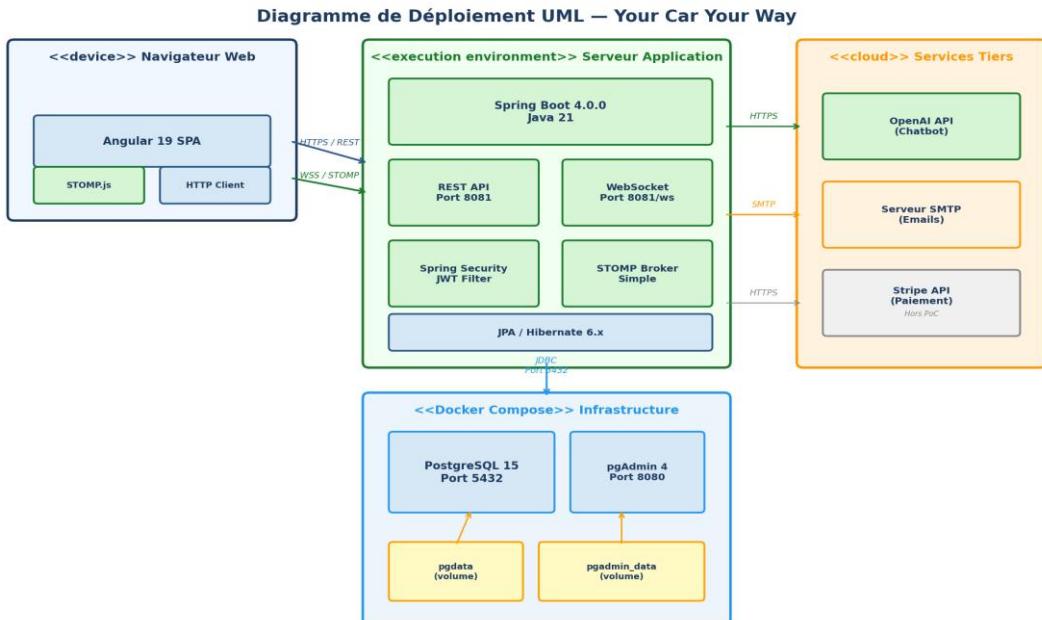


Figure 6 : Diagramme de déploiement UML — Infrastructure

Configuration Docker Compose

L'infrastructure de base de données est conteneurisée via Docker Compose. Cette approche garantit la reproductibilité de l'environnement et permet le lancement en une seule commande (docker-compose up).

Conteneur	Image	Port	Rôle
your_car_your_way	postgres:15	5432	Base de données principale
pgadmin4	dpage/pgadmin4	8080	Interface d'administration BDD

Volume	Conteneur	Rôle
pgdata	your_car_your_way	Persistance des données PostgreSQL
pgadmin_data	pgadmin4	Persistance de la configuration pgAdmin

Les deux conteneurs sont configurés avec restart: always pour un redémarrage automatique en cas de défaillance. La configuration sensible (identifiants BDD, clé JWT, clés API) est externalisée dans le fichier application.yaml et les variables d'environnement Docker.

Traçabilité : Contraintes client → Choix techniques

Le tableau ci-dessous relie chaque contrainte exprimée par le client (CEO) à une exigence du BR, puis au choix technique retenu dans cette architecture, avec la justification associée.

Contrainte client	Réf. BR	Choix technique	Justification
Application centralisée internationale	Contexte BR	SPA Angular + API REST	Une seule application web accessible depuis tous les pays, un seul backend API consommable par tout client HTTP
Inscription et connexion sécurisées	BR-AUTH-01/02	JWT + BCrypt + Spring Security	Auth stateless scalable, mots de passe non réversibles, protection des endpoints par rôle
Support client synchrone (tchat)	BR-SUP-02	Spring WebSocket + STOMP	Push serveur natif, latence < 2s sans polling. Spring intègre STOMP nativement, pas de serveur tiers.
Support client synchrone (visio)	BR-SUP-03	WebRTC (prévu V2)	Communication pair-à-pair, pas de relais serveur. Signalisation via WebSocket existant.
Support client asynchrone (message)	BR-SUP-01	REST API + PostgreSQL	Messages persistés en BDD, consultables à tout moment. Pas de dépendance à la connexion simultanée.
Assistance automatisée 24/7	BR-SUP-04	OpenAI API	Modèle de langage performant, intégration par API HTTPS simple, réponse contextuelle.
Paiement sécurisé externalisé	BR-LOC-04, RM-07	Stripe Checkout + Webhooks	Conformité PCI-DSS déléguée à Stripe. Aucune donnée bancaire côté serveur.
API pour les agences	BR — Exig. API	REST JSON + SpringDoc OpenAPI	API documentée Swagger, CRUD standard. Accessible par toute application en agence.
Reproductibilité environnement	Exig. technique	Docker Compose	Lancement en 1 commande. Scripts SQL de création inclus. README d'exécution fourni.
Base de données fiable et normée	Toutes	PostgreSQL 15, 3NF	Intégrité référentielle par FK, conformité 3NF démontrée (section 5.1), licence open-source.

Justification de l'architecture

Cette section justifie l'approche architecturale retenue en la reliant explicitement aux contraintes du projet et aux exigences du Business Requirements.

Choix de l'architecture en couches

L'architecture en couches (Controller → Service → Repository) a été retenue pour les raisons suivantes :

- Séparation des responsabilités : chaque couche a un rôle unique et clairement défini, ce qui facilite la maintenance et l'évolution du code
- Testabilité : chaque couche peut être testée indépendamment (tests unitaires sur les Services, tests d'intégration sur les Controllers)
- Convention Spring Boot : ce pattern est le standard de l'écosystème Spring, ce qui facilite l'onboarding de nouveaux développeurs et l'accès à la documentation
- Compatibilité avec le PoC : le scope du PoC (chat + authentification) ne nécessite pas une architecture plus complexe (hexagonale, microservices). L'architecture en couches offre le bon niveau d'abstraction pour ce périmètre

Choix de WebSocket + STOMP pour le temps réel

Le tchat en temps réel (BR-SUP-02) exige que les messages s'affichent sans recharge de page, avec une latence inférieure à 2 secondes. Trois options ont été évaluées :

Option	Description	Verdict
HTTP Polling	Le client interroge le serveur toutes les X secondes. Simple mais consomme de la bande passante et ne garantit pas la réactivité.	✗ Non retenu : latence imprévisible, charge serveur
Server-Sent Events	Flux unidirectionnel serveur → client. Adapté aux notifications mais pas au chat bidirectionnel.	✗ Non retenu : unidirectionnel
WebSocket + STOMP	Connexion bidirectionnelle persistante. STOMP fournit un système de topics (pub/sub) natif dans Spring. Latence minimale.	✓ Retenu

Choix de JWT pour l'authentification

L'authentification par token JWT a été retenue car elle est stateless (pas de session côté serveur), compatible avec l'architecture REST, et permet la scalabilité horizontale. Le token contient les informations nécessaires (userId, role) pour autoriser les requêtes sans interroger la base à chaque appel.

Faisabilité du projet

L'ensemble des choix techniques converge vers une architecture réalisable avec les technologies maîtrisées par l'équipe. Le PoC démontre la faisabilité du cœur fonctionnel (chat temps réel) avec les mêmes technologies que celles prévues pour la version complète. L'utilisation de Docker garantit la reproductibilité de l'environnement de développement et facilite le déploiement futur.

ANNEXE :