DOCUMENT DE CONCEPTION

GIA Vehicle Booking System

Architecture · Diagrammes · Justifications techniques

Auteur	Version	Contexte
Epanti Awoum Franc Junior	**v1.0.0 · Février 2026**	**Test Technique GIA Group**

1. Contexte et Justification du Projet

1.1 Contexte du recrutement

GIA Group recherche un Développeur Full Stack. Parmi les deux projets proposés dans le test technique (TaskFlow — gestion de tâches, ou Système de Réservation), le choix s'est porté sur le Système de Réservation de Véhicules pour des raisons stratégiques directement liées à la mission du poste.

> Alignement stratégique : GIA Group est un groupe actif dans la logistique, le transport et la gestion de flottes au Cameroun. Un système de réservation de véhicules répond directement à leurs problématiques métier quotidiennes, démontrant une compréhension de leur secteur d'activité.

1.2 Problématique métier résolue

La gestion manuelle des réservations de véhicules (appels téléphoniques, tableurs Excel, carnets papier) génère :

- Des conflits de disponibilité (double réservation d'un même véhicule)
- Un manque de visibilité sur l'état de la flotte en temps réel
- Une traçabilité financière incomplète (paiements non enregistrés)
- Une expérience client dégradée (pas de confirmation automatique)

GIA Vehicle Booking System résout ces problèmes avec :

- Vérification automatique des conflits de dates à la création
- Tableau de bord temps réel avec statuts et revenus
- Module de paiement avec ticket et historique complet
- Notifications email automatiques à chaque étape

1.3 Public cible

Acteur	Description
Client particulier	Loue un véhicule pour un usage personnel ou professionnel via l'interface web
Administrateur GIA	Gère la flotte, valide/modifie les réservations, consulte les paiements
Responsable financier	Consulte les paiements filtrés par date, statut, mode de paiement

2. Cahier des Charges Fonctionnel

2.1 Objectifs du projet

1. Permettre la réservation en ligne de véhicules avec vérification de disponibilité en temps réel
2. Offrir un tableau de bord administrateur complet pour la gestion de la flotte et des réservations
3. Automatiser les notifications (email) à chaque étape du cycle de vie d'une réservation
4. Fournir un module de paiement avec ticket imprimable et téléchargeable
5. Respecter la charte graphique GIA Group et offrir une interface bilingue FR/EN

2.2 Fonctionnalités principales (MVP)

Module	Fonctionnalités
Authentification	Inscription, connexion JWT, réinitialisation mot de passe (token TTL 1h), anti-énumération
Catalogue	Filtres multi-critères (catégorie, carburant, transmission, prix), vue détail, disponibilité
Réservation	Sélection de dates, calcul prix automatique, détection de conflits, confirmation immédiate
Paiement simulé	Formulaire carte, simulation gateway (1.5s), ticket PDF, impression navigateur
Dashboard client	Historique, statuts colorés, bouton Payer (CONFIRMED), bouton Ticket (payé)
Dashboard admin	Stats temps réel, CRUD véhicules, gestion réservations/utilisateurs/paiements
Emails	Bienvenue, confirmation réservation, reset mot de passe — templates HTML personnalisés
i18n	Basculement FR/EN instantané sur toute l'interface, persisté en localStorage

2.3 Fonctionnalités secondaires

- Optimistic updates : actions admin instantanées (0ms de latence perçue)
- Onglet Paiements admin : filtres par dates, statut, moyen de paiement
- Logging structuré (Winston) avec niveaux INFO/WARN/ERROR
- Gestion des rôles (USER/ADMIN) avec protections middleware
- Session unique : redirection silencieuse si déjà connecté

2.4 Contraintes techniques

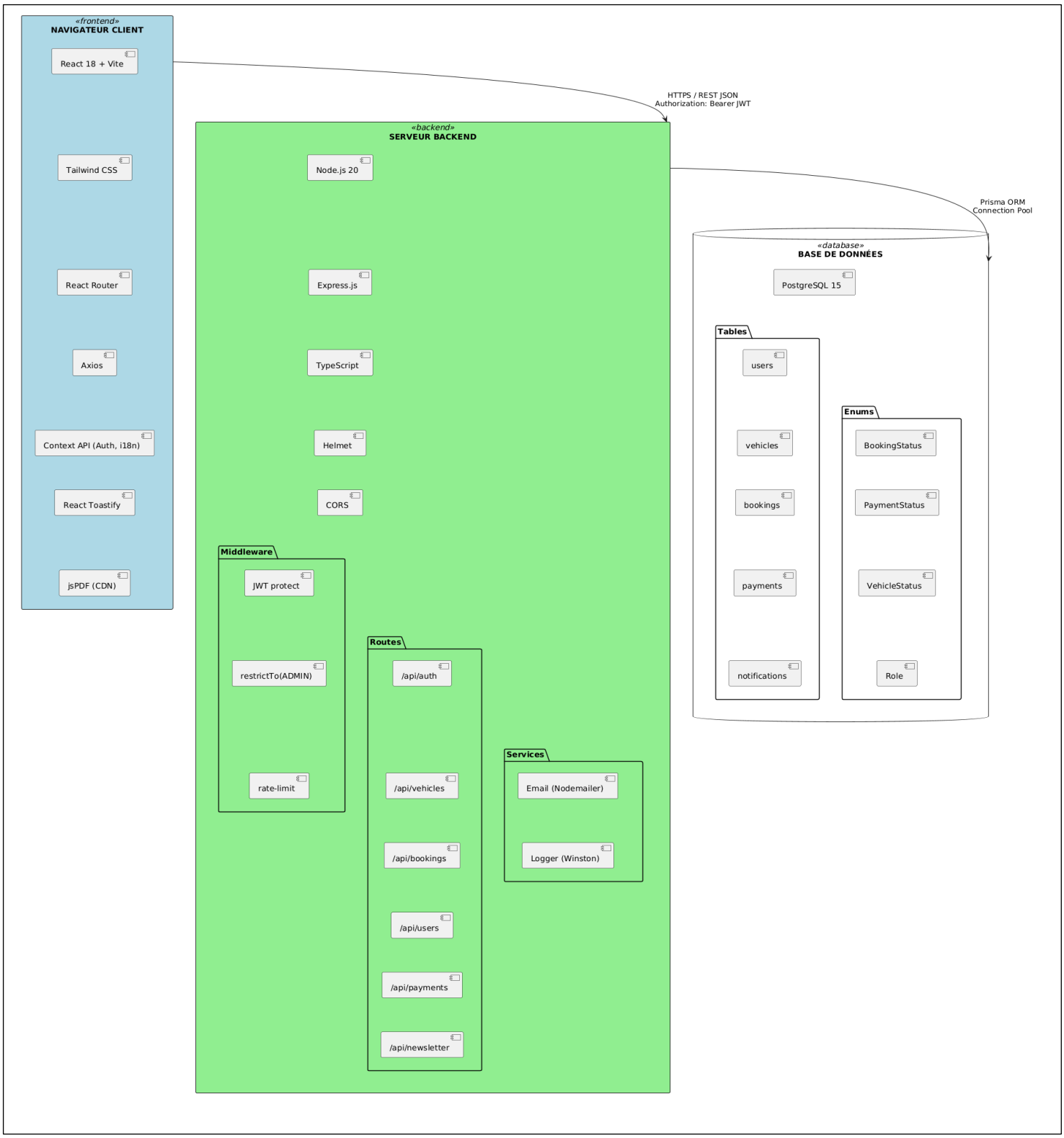
Contrainte	Détail
Performance	Optimistic updates — toutes les actions admin sont reflétées < 50ms côté UI
Sécurité	Mots de passe bcrypt (12 rounds), JWT expiry 7j, tokens reset TTL 1h
Typage	TypeScript strict sur frontend et backend — zéro any non justifié
Compatibilité	Chrome 120+, Firefox 121+, Safari 17+, responsive mobile
Conventions	Enums UPPERCASE en base et API, snake_case en base, camelCase en API JSON

2.5 Critères de succès

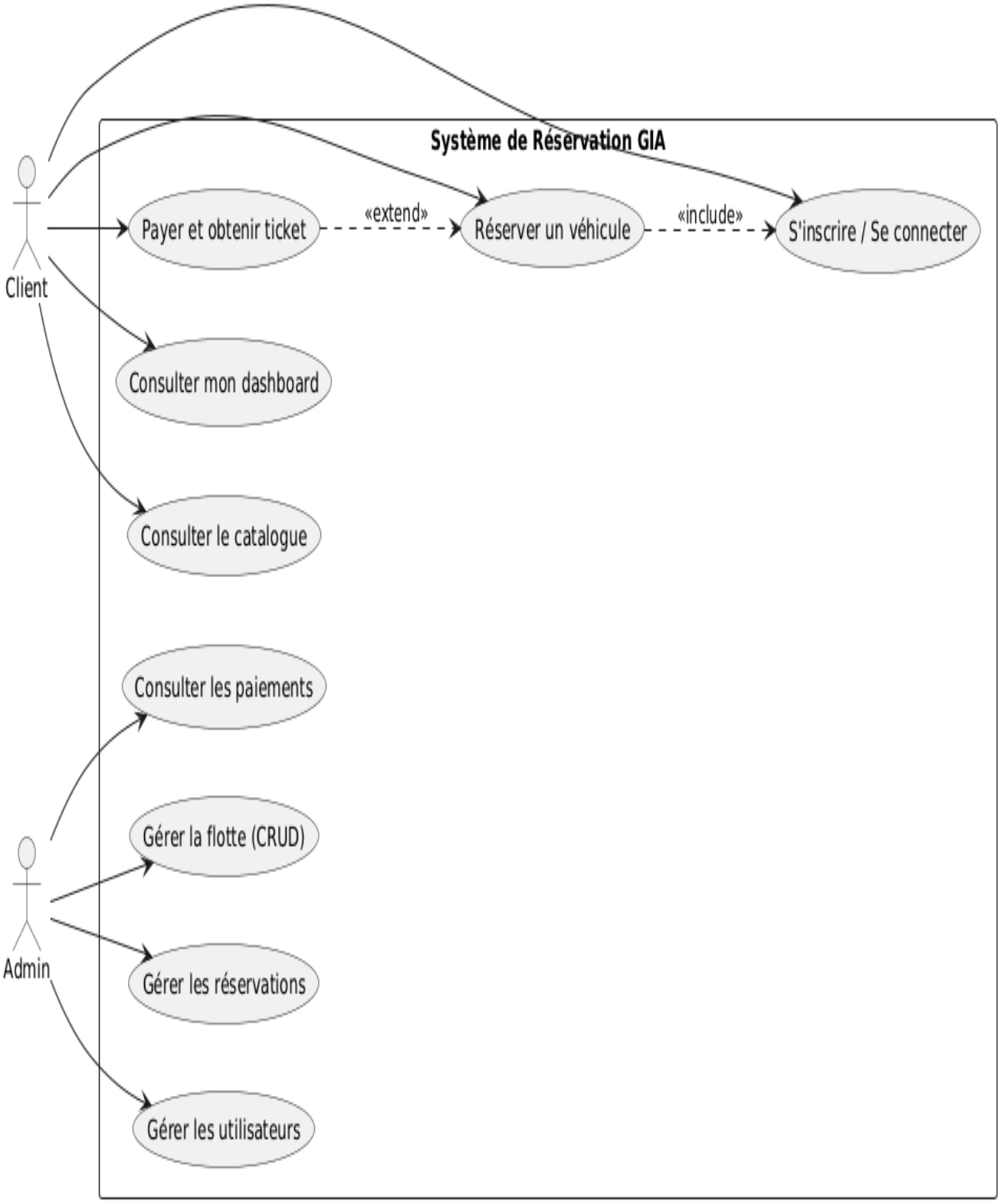
- Zéro conflit de réservation possible (vérification atomique en base)
- 100% des parcours utilisateur couverts par le cahier de recette (62 cas de test PASS)
- Interface bilingue cohérente sur toutes les pages et états
- Ticket de paiement imprimable et téléchargeable en PDF

3. Diagrammes

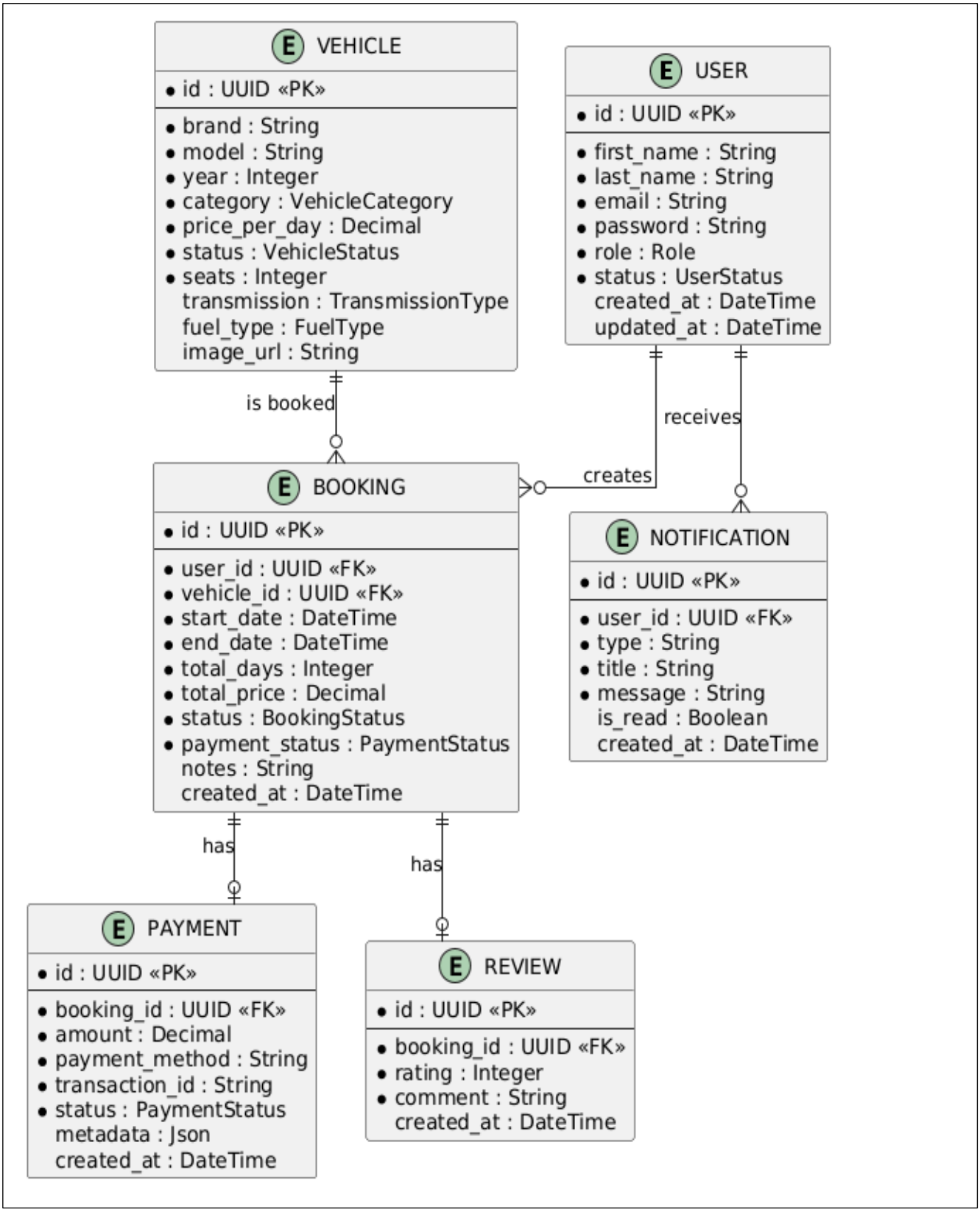
3.1 Architecture Technique (Client/serveur)



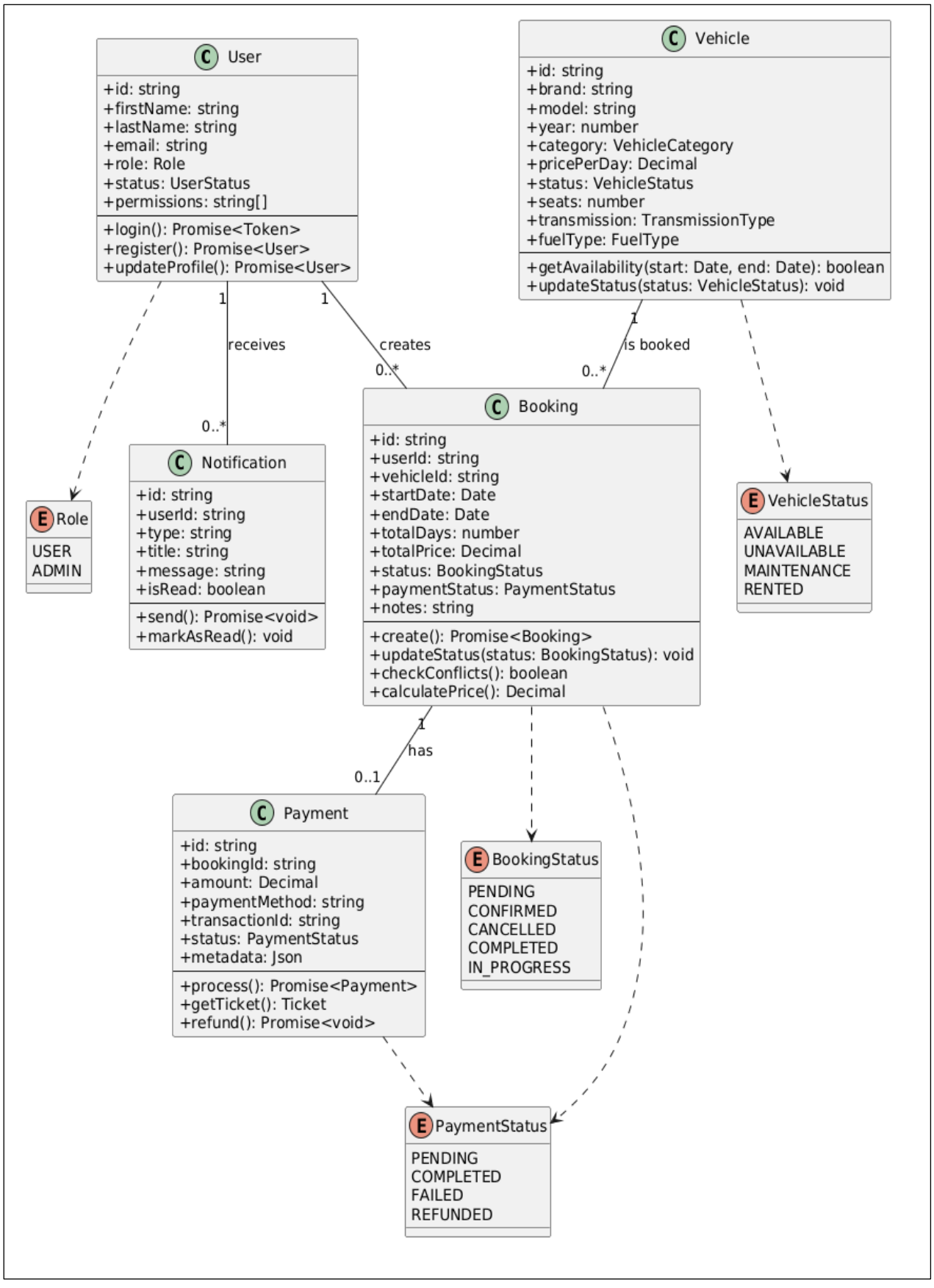
3.2 Diagramme de Cas d'Utilisation



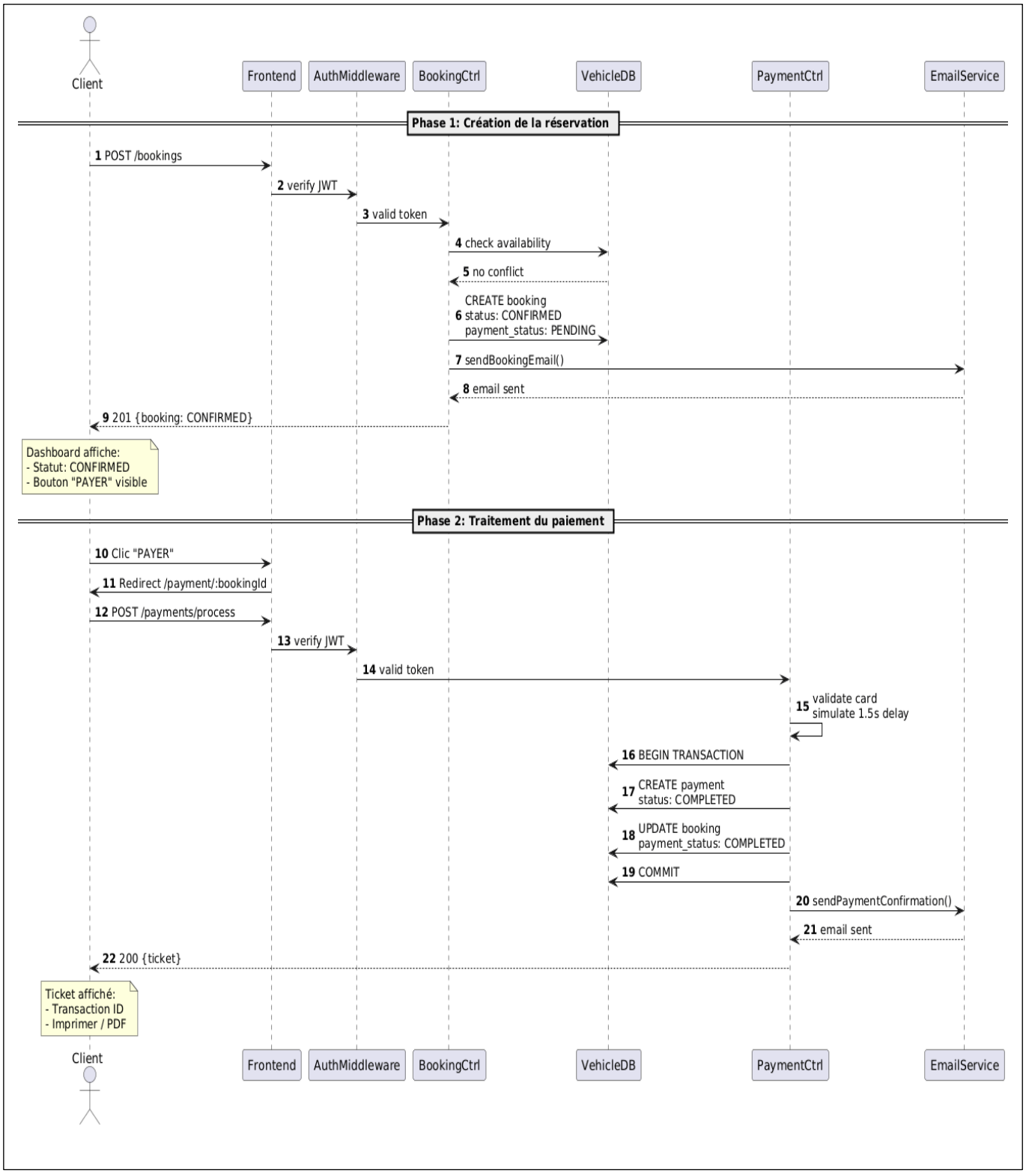
3.3 Modèle Conceptuel de Données (MCD)



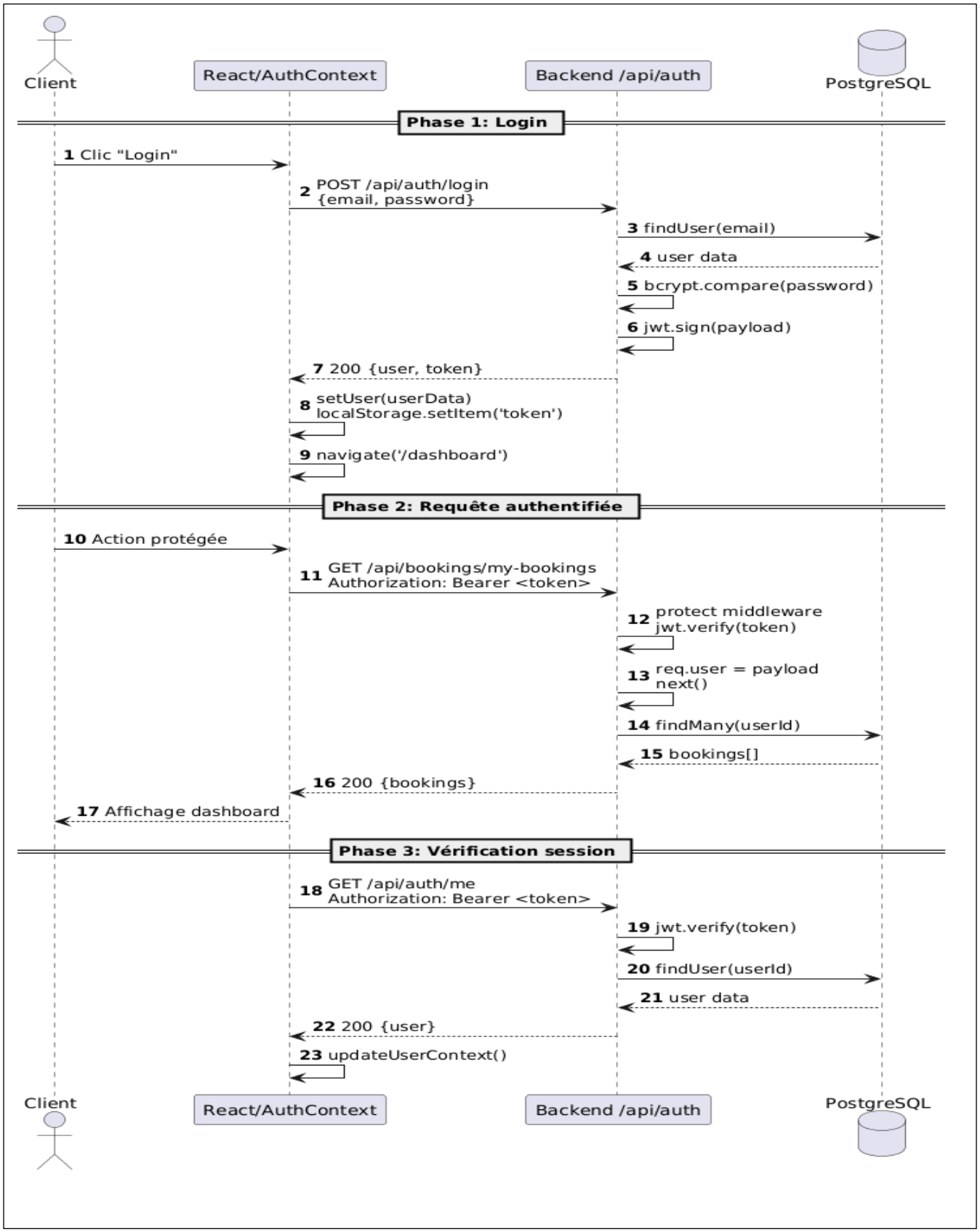
3.4 Diagramme de Classes (UML simplifié)



3.5 Diagramme de Séquence — Processus de Réservation & Paiement



3.6 Diagramme de Séquence — Authentification



4. Justification des Choix Techniques

4.1 Backend — Node.js + Express + TypeScript

Technologie	Justification
Node.js 20 LTS	Runtime JavaScript asynchrone — I/O non-bloquant idéal pour une API REST avec nombreuses requêtes DB et emails simultanés. LTS = stabilité en production.
Express.js	Framework minimaliste et éprouvé. Middleware chaînable (protect → restrictTo → handler). Pas de sur-ingénierie vs NestJS pour ce périmètre.
TypeScript strict	Typage statique : détection d'erreurs à la compilation, meilleure maintenabilité. Contrat de types partagé frontend/backend. Prisma génère des types depuis le schéma.
Pattern MVC	Séparation claire : routes (routing) → middleware (auth, validation) → controllers (logique) → Prisma (accès données). Lisibilité et testabilité maximales.

4.2 ORM & Base de données — Prisma + PostgreSQL

Technologie	Justification
Prisma 5	ORM type-safe : le schéma Prisma génère les types TypeScript automatiquement. Migrations déclaratives. Studio intégré pour l'inspection. Alternative à TypeORM jugée plus verbose.
PostgreSQL 15	SGBD relationnel robuste : intégrité référentielle, enums natifs (BookingStatus, PaymentStatus), index composites (start_date, end_date) pour la détection de conflits. ACID compliance critique pour les transactions de paiement.
Prisma.$transaction	Utilisé pour créer le Payment ET mettre à jour le Booking atomiquement — garantit la cohérence même en cas d'erreur réseau entre les deux opérations.
Supabase (prod)	PostgreSQL managé : backups automatiques, connexion SSL, dashboard web, plan gratuit suffisant pour ce test.

4.3 Frontend — React 18 + Vite + Tailwind CSS

Technologie	Justification
React 18	Bibliothèque UI la plus adoptée dans l'écosystème. Hooks, composition de composants, JSX. Pas de sur-ingénierie avec un framework complet (Next.js non nécessaire ici).
Vite 5	Bundler nouvelle génération : démarrage < 300ms, HMR instantané. Remplace Create React App (déprécié). Build optimisé avec tree-shaking et code-splitting automatique.
Tailwind CSS 3	Utility-first : design system cohérent sans écrire de CSS. Classes conditionnelles lisibles (hover:, focus:, responsive:). Purge automatique en production — bundle CSS minimal.

Context API (pas Redux)	AuthContext + LangContext suffisent. Redux = sur-ingénierie pour 2 contextes globaux. Moins de boilerplate, plus de lisibilité pour un projet de cette taille.
Axios avec intercepteurs	Injection automatique du token JWT (Bearer) dans chaque requête. Gestion centralisée des erreurs 401 (déconnexion automatique).
Optimistic Updates	Actions admin mises à jour dans le state React immédiatement, sync silencieuse en arrière-plan. UX fluide même sur connexion lente.

4.4 Sécurité — Décisions architecturales

Mesure	Implémentation
Authentification JWT	Stateless : scalable horizontalement. Token signé HS256, expiry 7j. Vérification à chaque requête via middleware protect.
Hachage mots de passe	bcryptjs, 12 rounds de salt. Résistant aux attaques brute-force et aux rainbow tables.
Token reset sécurisé	crypto.randomBytes(32) en base64url — imprévisible. TTL 1h en base. Usage unique : invalidé après utilisation.
Anti-énumération	/api/auth/forgot-password retourne HTTP 200 même si l'email est inconnu — empêche de savoir quels emails sont enregistrés.
Rate limiting	express-rate-limit sur /api/ — protège contre les attaques par force brute et le DDoS applicatif.
Données carte	Numéro de carte jamais stocké. Seuls les 4 derniers chiffres masqués et le nom du titulaire sont conservés en metadata JSON.

5. Design Patterns Utilisés

Pattern	Application dans le projet
MVC (Model-View-Controller)	Prisma = Model, React = View, Express Controllers = Controller
Middleware Chain	protect() → restrictTo() → requirePermission() → handler()
Repository Pattern	Prisma abstrait l'accès aux données — les controllers ne connaissent pas SQL
Singleton (Prisma Client)	Une seule instance prisma partagée — évite la saturation des connexions DB
Observer (Context API)	AuthContext notifie tous les composants abonnés lors d'un changement de session
Optimistic UI	L'état local est mis à jour avant la confirmation serveur — puis rollback si erreur
Strategy (i18n)	Même interface t() quel que soit la langue — implémentation interchangeable FR/EN

Factory (generateTransactionId)	Génération d'identifiants transaction reproductibles et formatés (GIA-{ts}-{rand})

6. Synthèse et Bilan

Ce projet démontre la maîtrise d'une stack moderne full stack TypeScript, d'une architecture propre et maintenable, de la sécurité applicative, de l'expérience utilisateur, et de la capacité à livrer un projet fonctionnel complet dans les délais imposés.

Critère GIA	Réalisation
Qualité du code	TypeScript strict, séparation des responsabilités, commentaires JSDoc, logger structuré
Architecture	MVC + Middleware Chain + Repository Pattern + Context API — documentée avec diagrammes
Choix techniques	Justifiés et adaptés au périmètre — pas de sur-ingénierie
Ergonomie / Fluidité	Optimistic updates, i18n, feedback toast, étapes de paiement visuelles, responsive
Charte graphique	Couleurs GIA (#2A3180 / #189CD9), Inter, cartes arrondies, badges colorés
Documentation	Cahier de recette (62 cas), doc technique, doc de conception, DEPLOYMENT.md, README
Livraison fonctionnelle	Tous les modules livrés : auth, catalogue, réservation, paiement, admin, emails

— Fin du Document de Conception — GIA Group · Février 2026 —