

## Portfolio Project - Technical Documentation (Stage 3)

### 1. User Stories

#### 1.1 Client (Utilisateur Final)

##### Must Have (M)

- US001: En tant que client, je veux saisir mon adresse de départ et d'arrivée, afin de pouvoir réserver un VTC.
- US002: En tant que client, je veux voir une estimation du prix avant de confirmer, afin de connaître le coût de ma course.
- US003: En tant que client, je veux recevoir une confirmation par email une fois ma réservation validée, afin d'avoir une preuve de ma réservation.
- US004: En tant que client, je veux suivre le statut de ma commande en temps réel, afin de savoir où en est ma réservation.
- US005: En tant que client, je veux créer un compte et m'authentifier, afin de gérer mes réservations.

##### Should Have (S)

- US006: En tant que client, je veux consulter l'historique de mes courses, afin de retrouver mes trajets précédents.
- US007: En tant que client, je veux télécharger mes factures en PDF, afin d'avoir une trace comptable de mes frais.
- US008: En tant que client, je veux utiliser la géolocalisation pour remplir automatiquement mon adresse de départ, afin de gagner du temps lors de la réservation.
- US009: En tant que client, je veux voir une estimation du temps d'attente, afin de planifier mon départ.

##### Could Have (C)

- US010: En tant que client, je veux évaluer ma course et le chauffeur, afin de contribuer à l'amélioration du service.
- US011: En tant que client, je veux modifier ou annuler ma réservation, afin de m'adapter aux imprévus.
- US012: En tant que client, je veux sauvegarder mes adresses favorites, afin de réserver plus rapidement.

#### 1.2 Administrateur/Dispatcher (Centrale)

##### Must Have (M)

- US013: En tant qu'administrateur, je veux recevoir les demandes de courses en temps réel, afin de pouvoir les traiter rapidement.

- US014: En tant qu'administrateur, je veux un calcul automatique du prix d'une course, afin que le client a une estimation précise.
- US015: En tant qu'administrateur, je veux suivre les statuts des courses, afin de montrer l'activité en temps réel.

#### **Should Have (S)**

- US016: En tant qu'administrateur, je veux voir un tableau de bord avec les statistiques d'activité, afin de suivre les performances de la plateforme.
- US017: En tant qu'administrateur, je veux gérer les cas particuliers manuellement, afin de résoudre les problèmes complexes.
- US018: En tant qu'administrateur, je veux générer des rapports d'activité, afin d'analyser les tendances.

#### **Could Have (C)**

- US019: En tant qu'administrateur, je veux configurer des règles de dispatch automatique, afin d'optimiser l'attribution des courses. → *Cette fonctionnalité est envisagée comme évolution future. Aucun algorithme de dispatch automatique n'est défini dans le périmètre actuel.*
- US020: En tant qu'administrateur, je veux recevoir des alertes en cas de problème, afin de réagir rapidement.

### **1.3 Chauffeur VTC**

#### **Must Have (M)**

- US021: En tant que chauffeur, je veux recevoir un mail (ou Telegram) récapitulatif de la commande après acceptation de la course.

#### **Should Have (S)**

- US022: En tant que chauffeur, je veux indiquer ma disponibilité, afin que la centrale sache quand me contacter.
- US023: En tant que chauffeur, je veux voir les détails de la course (départ, arrivée, prix), afin de prendre une décision éclairée.

## **2. Priorisation MoSCoW**

#### **Must Have (Essentiels pour le MVP)**

**Total : 9 User Stories**

#### **Client (5 stories) :**

- Saisie d'adresses départ/arrivée (US001)
- Estimation de prix avant confirmation (US002)
- Confirmation par email (US003)
- Suivi de statut en temps réel (US004)
- Authentification et gestion de compte (US005)

**Administrateur (3 stories) :**

- Réception des demandes en temps réel (US013)
- Calcul automatique des prix (US014)
- Suivi des statuts des courses (US015)

**Chauffeur (1 story) :**

- Réception d'email récapitulatif de commande (US021)

**Should Have (Importantes pour une bonne expérience)****Total : 9 User Stories****Client (4 stories) :**

- Historique des courses (US006)
- Téléchargement factures PDF (US007)
- Géolocalisation automatique (US008)
- Estimation temps d'attente (US009)

**Administrateur (3 stories) :**

- Tableau de bord avec statistiques (US016)
- Gestion manuelle des cas particuliers (US017)
- Génération de rapports d'activité (US018)

**Chauffeur (2 stories) :**

- Indication de disponibilité (US022)
- Consultation détails de course (US023)

**Could Have (Améliorations futures)****Total : 5 User Stories****Client (3 stories) :**

- Système d'évaluation course/chauffeur (US010)
- Modification/annulation de réservation (US011)
- Sauvegarde d'adresses favorites (US012)

**Administrateur (2 stories) :**

- Configuration règles dispatch automatique (US019)
- Alertes en cas de problème (US020)

**Won't Have (Hors scope MVP)****Fonctionnalités explicitement exclues selon la charte de projet :**

- Application mobile native (iOS/Android)
- Paiement en ligne intégré
- Programme de fidélité
- Intelligence artificielle
- Intégrations tierces étendues (ERP, CRM...)
- Notifications et suivi par SMS

### 3. Mockups Principaux

#### 3.1 Interface Client - Page de Réservation

The mockup shows a user interface for reserving a VTC (Taxi, Cab, or Limousine). It includes fields for 'Départ' (Departure) and 'Arrivée' (Arrival), date and time inputs, and vehicle type selection (ECO, BERLINE, VAN). Estimated cost and wait time are also displayed.

Vehicle Type	Estimation	Temps d'attente
ECO	25€ – 30€	~5 min
BERLINE		
VAN		

**Réserver maintenant**

### 3.2 Interface Client - Suivi de Commande

The screenshot shows a mobile application interface for a VTC platform. At the top, there's a header with a car icon and the text "VTC Platform" next to a "Mon compte" link. Below the header, a section titled "Suivi de votre course" displays a reservation with the identifier "Réservation #VTC001234". A list of status updates is shown with icons and timestamps:

- Demande reçue 14:30
- Chauffeur assigné 14:32
- En route vers vous 14:35
- Course en cours ---:--
- Course terminée ---:--

Below the status list, a card provides details about the driver: "Chauffeur: Ahmed M.", "Peugeot 508 - AB-123-CD", and a phone number "Contacter: 06.xx.xx.xx". At the bottom of the screen is a blue button labeled "Annuler la réservation".

### 3.3 Interface Admin - Dashboard de Dispatch

The screenshot shows the Central Dispatch admin dashboard. At the top, it displays the name "Central Dispach" and "Admin: Gabriel". Below the header, a section titled "Tableau de bord – Temps réel" shows real-time statistics for requests:

Demandes	En attente	En cours
42	9	31

Under the statistics, there's a section titled "Demandes en attente" (Pending Requests) listing three entries:

#001	15:42	République → CDG	45€	
#002	15:47	Châtelet → Orly	38€	
#003	15:43	Bastille → La Défense	22€	

Finally, the dashboard shows a section titled "Chauffeurs disponibles: 12" (Available Drivers) with a list of drivers and their assigned zones:

Ahmed M.	Zone C	[Assigner]
Sophie L.	Zone V	[Assigner]
Marc D.	Zone S	

### 3.4 Interface Admin - Détail d'une Demande

Détail Demande #VTC001234

Jean.Dupont  
 Email.jean.dupont@email.com  
 Tél: 06.xx.xx.xx.xx  
 Trajet:  
 123 Rue de République  
Aéroport CDG Terminal 2  
 Heure demandée: 16:30  
 Prix estimate: 45 €  
 Distance: 32 km

Chauffeurs contactés:

Ahmed M. - Refusé (15:43 )  
 Sophie L. - En attente...

Contacter chauffeur Annuler

Spécifications des Mockups

### 3.5 Responsive Design

Desktop : Interface complète avec tous les éléments visibles

Tablet : Adaptation en 2 colonnes, navigation simplifiée

Mobile : Interface empilée, boutons plus larges, navigation par onglets

### 3.6 Couleurs et Branding

Couleur principale : Noir

Couleur secondaire : Gris moderne

Couleur d'accent : Vert pour les confirmations

Couleur d'alerte : Orange pour les attentes

Couleur d'erreur : Rouge pour les erreurs

### 3.7 Éléments d'Interface

Icônes : Utilisation d'une bibliothèque cohérente (Lucide React)

Typographie : Police claire et professionnelle (Inter, Arial)

Boutons : Style moderne avec coins arrondis

Formulaires : Champs clairement étiquetés avec validation visuelle

### **3.8 Navigation**

Client : Navigation simple (Accueil, Réserver, Mes courses, Mon compte)

Admin : Sidebar avec sections (Dashboard, Demandes, Chauffeurs)

États de navigation : Indication claire de la page active

## **4. User Flow Principal**

### **4.1 Parcours Client Standard**

1. Accueil → Clic "Réserver"
2. Réservation → Saisie adresses → Estimation → Confirmation
3. Attente → Page de suivi → Notifications
4. Course → Suivi temps réel → Fin de course
5. Post-course → Évaluation → Historique

### **4.2 Parcours Admin Standard**

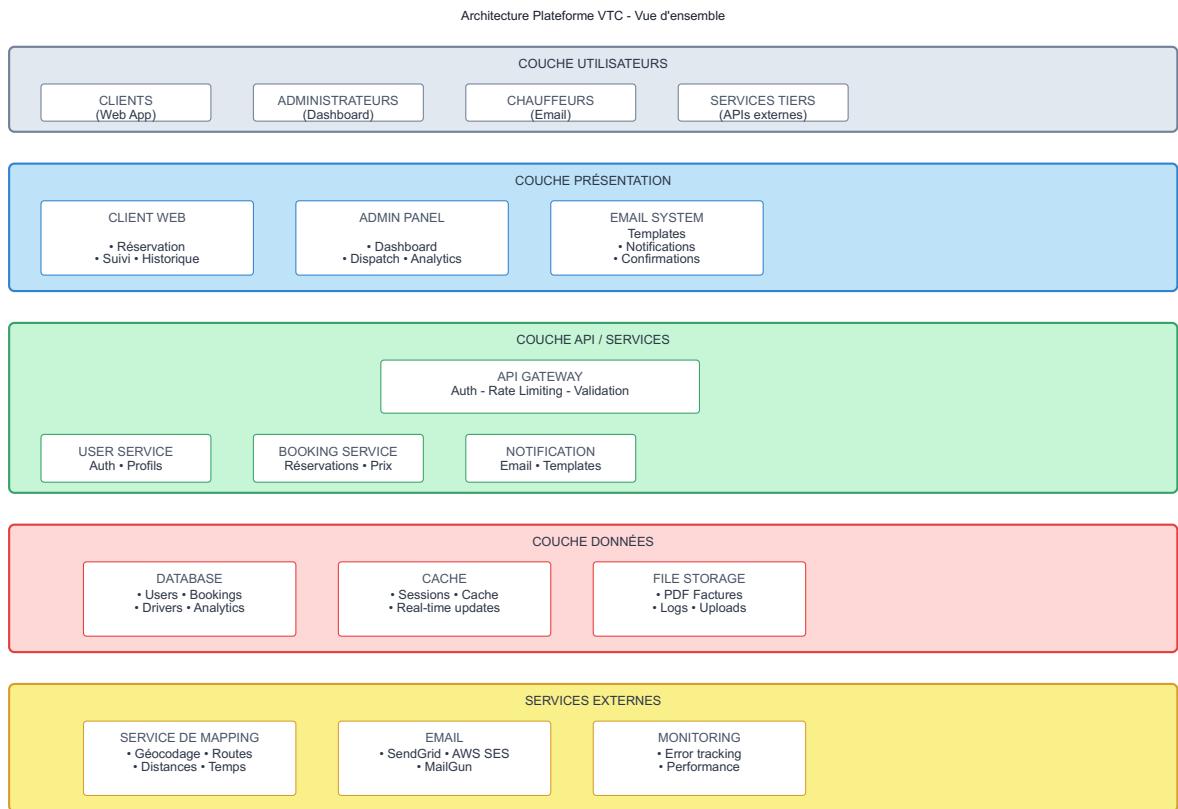
1. Login → Dashboard
2. Réception demande → Évaluation → Contact chauffeur
3. Gestion réponses → Assignation → Suivi
4. Clôture course → Facturation → Archivage

Cette documentation des User Stories et Mockups établit les bases fonctionnelles et visuelles du MVP, en priorisant les fonctionnalités essentielles pour valider le concept de plateforme VTC.

## 6. System Architecture - Plateforme VTC

## A. Vue d'ensemble de l'architecture

L'architecture de la plateforme VTC suit un modèle en couches avec séparation claire des responsabilités et scalabilité horizontale.



#### B. Flux de données détaillées

## B.1 Flux de réservation client

```
sequenceDiagram
    participant Client
    participant APIGateway
    participant AuthService
    participant BookingService
    participant MapAPI
    participant Validation
    participant DataBase
    participant NotificationService
    participant Email
    participant AdminDashboard

    Client->>APIGateway: 
    activate APIGateway
    APIGateway->>AuthService: 
    activate AuthService
    AuthService->>BookingService: 
    activate BookingService
    BookingService->>MapAPI: 
    deactivate BookingService
    deactivate MapAPI
    MapAPI-->>Validation: Calcul prix, Distance/Route
    deactivate MapAPI
    Validation-->>DataBase: Création booking
    DataBase-->>NotificationService: Trigger email
    NotificationService-->>Email: 
    activate Email
    Email-->>Client: confirmation
    deactivate Email
    Client-->>AdminDashboard: 
```

The diagram illustrates a sequential flow of requests and responses between various components. It starts with a client sending a request to an API Gateway. The API Gateway then sends a request to an Auth Service. The Auth Service sends a request to a Booking Service. The Booking Service sends a request to a Map API. The Map API returns calculated prices and distances/route information to the Booking Service. The Booking Service then sends a request to a Database to create a booking. The Database sends a request to a Notification Service to trigger an email. Finally, the Notification Service sends a confirmation email back to the client, which is then received by an Admin Dashboard.

## B.2 Flux de dispatch administrateur

```
ADMIN DASHBOARD → API Gateway → Booking Service
    ↓                               ↓
    Sélection course → → → → → → Mise à jour statut
    ↓                               ↓
    DATABASE ← ← ← ← ← ← ← ← ← ← Sauvegarde
    ↓                               ↓
    Notification Service ← ← ← ← ← ← ← Trigger email
    ↓                               ↓
    Email → → → → → → → → → → CHAUFFEUR (commande)
    ↓                               ↓
    → → → → → → → → → → CLIENT WEB (mise à jour)
```

## C. Composants détaillés

### C.1 Frontend (React + Tailwind CSS)

#### CLIENT WEB APP

Components:

- BookingForm
- StatusTracker
- UserDashboard
- HistoryList
- InvoiceDownloader

Services:

- API Client
- Auth Manager
- WebSocket Client
- LocalStorage Manager

#### ADMIN DASHBOARD

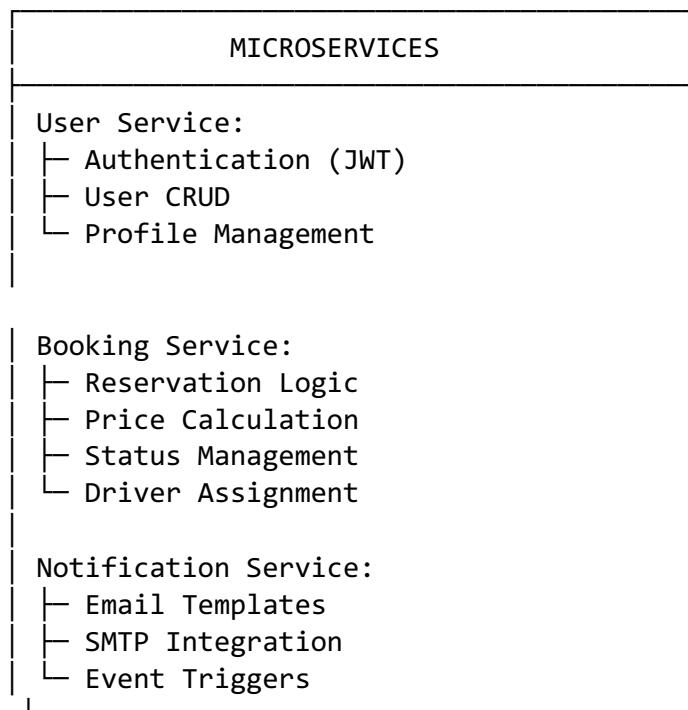
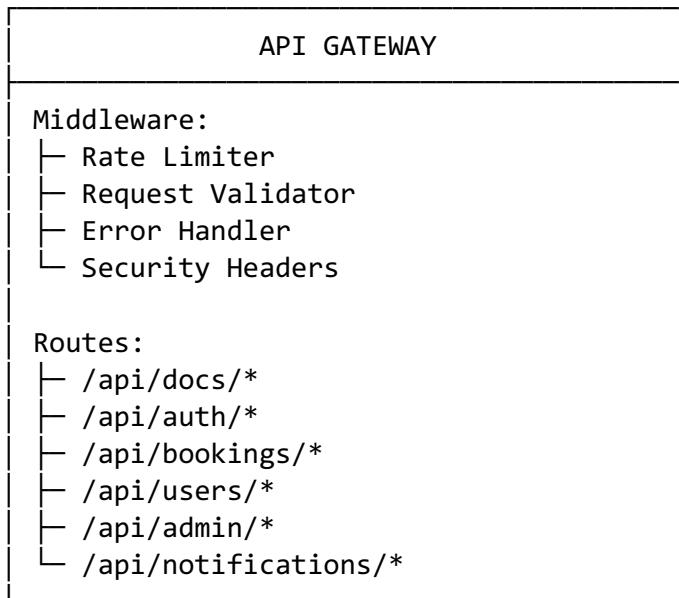
Components:

- RequestsQueue
- DriversList
- AnalyticsDashboard
- ReportsGenerator
- ManualDispatch

Real-time Features:

- WebSocket Connection
- Live Updates
- Notifications

## C.2 Backend (Django - Python)



### C.3 Base de données (PostgreSQL)

POSTGRESQL
Tables principales: └── users └── bookings └── drivers └── invoices └── analytics_events
Contraintes: └── Foreign Keys └── Check Constraints └── Unique Constraints

## D. Technologies sélectionnées

### D.1 Stack technique

- **Frontend** : React.js avec Tailwind CSS
- **Backend** : Python avec Django
- **Base de données**: PostgreSQL
- **Email** : Gmail
- **Maps** : Services de Maps

### D.2 Justifications techniques

#### Frontend : React + Tailwind CSS

##### React.js

- **Écosystème mature** : Large communauté, documentation complète, nombreuses bibliothèques tierces
- **Composants réutilisables** : Architecture modulaire permettant de créer des composants métier réutilisables dans différentes vues
- **Performance optimisée**
- **Développement rapide** : Outils de développement intégrés, syntaxe JSX intuitive

## Tailwind CSS

- **Développement accéléré** : Classes utilitaires prêtes à l'emploi, pas besoin d'écrire de CSS custom
- **Design system cohérent** : Palette de couleurs, espacements et typographies standardisés
- **Responsive design intégré** : Classes responsive natives pour adaptation multi-écrans
- **Bundle optimisé** : Supprime automatiquement le CSS non utilisé
- **Maintenance facilitée**

## Alternatives écartées

- **Vue.js** : Moins d'écosystème pour les bibliothèques métier VTC
- **Angular** : Plus lourd, courbe d'apprentissage plus élevée
- **Bootstrap** : Moins flexible que Tailwind, composants génériques

## Backend : Django + Swagger

### Django (Python)

- **Rapidité de développement** : ORM intégré, admin interface automatique
- **Batteries incluses** : Authentification, validation, gestion des formulaires, internationalisation native
- **Sécurité robuste** : Protection CSRF, SQL injection, XSS intégrées par défaut
- **Scalabilité éprouvée** : Utilisé par Instagram, Pinterest, Spotify pour des millions d'utilisateurs
- **Écosystème riche** : Django REST Framework pour les APIs

### Django REST Framework (DRF)

- **Sérialisation avancée** : Validation automatique des données, transformation JSON native
- **Authentification flexible**
- **Vues basées sur les classes**
- **Documentation automatique** : Génération de documentation API native

### Swagger/OpenAPI 3.0

- **Documentation interactive** : Interface utilisateur pour tester les endpoints en temps réel

- **Génération automatique** : Documentation synchronisée avec le code, pas de décalage
- **Standards industriels** : Format OpenAPI reconnu, compatible avec de nombreux outils
- **Intégration client**

#### Alternatives écartées

- **Node.js + Express** : Moins structuré
- **Laravel (PHP)** : Écosystème Python plus adapté aux calculs géographiques et intégrations API
- **Spring Boot (Java)** : Plus lourd, développement plus lent pour un MVP

#### Base de données : PostgreSQL

#### Propriétés ACID

- **Atomicité** : Garantit l'intégrité des transactions (réservation + notification + facturation)
- **Cohérence** : Contraintes d'intégrité strictes (foreign keys, check constraints)
- **Isolation** : Prévient les conditions de course lors de l'attribution des chauffeurs
- **Durabilité** : Persistance garantie des données critiques (réservations, paiements)

#### Relations complexes

- **Jointures efficaces** : Requêtes complexes entre utilisateurs, réservations, chauffeurs optimisées
- **Contraintes référentielles** : Intégrité des données garantie (pas de réservation sans utilisateur)
- **Triggers et procédures** : Logique métier complexe directement en base si nécessaire
- **Support JSON** : Stockage flexible pour métadonnées (préférences utilisateurs, logs détaillés)

#### Performance

- **Indexation avancée**
- **Optimisateur de requêtes** : Analyse statistique et optimisation automatique des plans d'exécution
- **Partitioning** : Division des tables par date/région pour de gros volumes
- **Parallel queries** : Exécution parallèle pour requêtes analytiques

## Alternatives écartées

- **MySQL** : Moins de fonctionnalités géospatiales, ACID moins strict historiquement
- **MongoDB** : Pas de relations strictes, moins adapté aux données transactionnelles critiques
- **SQLite** : Pas de concurrence multi-utilisateurs, pas d'extensions géospatiales

## Cohérence de la stack

### Avantages de l'intégration

- **Langage unifié** : JavaScript côté client, Python côté serveur (syntaxes modernes)
- **Outils de développement** : Docker compose pour environnement unifié
- **Déploiement simplifié** : Stack bien documentée, nombreux exemples de déploiement
- **Équipe développement** : Compétences transférables entre frontend et backend

### Performance globale

- **API REST optimisée** : Srialisation JSON native, pagination automatique
- **Monitoring intégré** : Logs structurés, métriques de performance natives

Cette stack technique garantit un développement rapide tout en maintenant la qualité, la sécurité et les performances nécessaires pour une plateforme VTC professionnelle.

## 7. Composants, les classes et la conception de la base de données - Plateforme VTC

### 7.1 Classes du Backend (Django)

#### 7.1.1 User Management Classes

##### User (Django AbstractUser extended)

```
class User(AbstractUser):  
    # Attributes  
    - phone_number: CharField(max_length=15)  
    - user_type: CharField(choices=['CLIENT', 'ADMIN'])  
    - created_at: DateTimeField(auto_now_add=True)  
    - updated_at: DateTimeField(auto_now=True)  
  
    # Methods  
    + create_user()
```

```
+ get_user_bookings()  
+ update_profile()
```

## Driver

```
class Driver:  
    # Attributes  
    - id: AutoField(primary_key=True)  
    - name: CharField(max_length=100)  
    - phone_number: CharField(max_length=15)  
    - email: EmailField()  
    - license_number: CharField(max_length=50)  
    - vehicle_info: TextField()  
    - created_at: DateTimeField(auto_now_add=True)  
  
    # Methods  
    + create_driver()
```

### 7.1.2 Booking Management Classes

```
class Booking:  
    # Attributes  
    - id: AutoField(primary_key=True)  
    - user: ForeignKey(User, on_delete=CASCADE)  
    - driver: ForeignKey(Driver, null=True, blank=True)  
    - pickup_address: CharField(max_length=255)  
    - pickup_latitude: DecimalField(max_digits=10, decimal_places=8)  
    - pickup_longitude: DecimalField(max_digits=11, decimal_places=8)  
    - destination_address: CharField(max_length=255)  
    - destination_latitude: DecimalField(max_digits=10, decimal_places=8)  
    - destination_longitude: DecimalField(max_digits=11, decimal_places=8)  
    - estimated_price: DecimalField(max_digits=8, decimal_places=2)  
    - final_price: DecimalField(max_digits=8, decimal_places=2, null=True, blank=True,  
help_text="Prix final facturé. Pour l'instant égal à l'estimated_price.")
```

*Note : Le champ `final\_price` est égal à `estimated\_price` pour l'instant. Ce champ est prévu pour de futures évolutions permettant d'ajuster le prix final (frais supplémentaires,*

*réductions, etc.). Aucun ajustement n'est appliqué par la suite. Il est utilisé pour générer la facture.*

- status: CharField(choices=BOOKING\_STATUS\_CHOICES)
- scheduled\_time: DateTimeField()
- created\_at: DateTimeField(auto\_now\_add=True)
- completed\_at: DateTimeField(null=True, blank=True)

#### # Methods

- + create\_booking()
- + calculate\_estimated\_price()
- + assign\_driver()
- + update\_status()
- + complete\_booking()
- + cancel\_booking()

### **BookingStatusEnum**

```
class BookingStatus:  
    PENDING = 'PENDING'  
    CONFIRMED = 'CONFIRMED'  
    DRIVER_ASSIGNED = 'DRIVER_ASSIGNED'  
    IN_PROGRESS = 'IN_PROGRESS'  
    COMPLETED = 'COMPLETED'  
    CANCELLED = 'CANCELLED'
```

### **7.1.3 Service Classes**

#### **PricingService**

```
class PricingService:  
    # Attributes  
    - base_price: float = 5.0  
    - price_per_km: float = 1.5  
    - price_per_minute: float = 0.3
```

#### # Methods

```
+ calculate_distance(pickup_coords, destination_coords)
+ estimate_duration(pickup_coords, destination_coords)
+ calculate_price(distance, duration)
```

## NotificationService

```
class NotificationService:
```

```
    # Attributes
```

```
    - email_backend: EmailBackend
    - templates_path: str
    - telegram_client: TelegramBotClient # <-- Dépendance injectée
```

```
    # Methods
```

```
    + send_booking_confirmation(user, booking)
    + send_driver_assignment(user, booking, driver)
    + send_booking_completion(user, booking)
    + send_cancellation_notice(user, booking)
    + notify_driver_new_booking(driver, booking)
    + notify_via_telegram(driver, booking) # <-- méthode dédiée (optionnelle)
```

```
class TelegramBotClient:
```

```
    # Attributes
```

```
    - bot_token: str
    - base_url: str = "https://api.telegram.org"
```

```
    # Methods
```

```
    + send_message(chat_id: str, text: str)
    + send_inline_keyboard(chat_id: str, options: List[str])
```

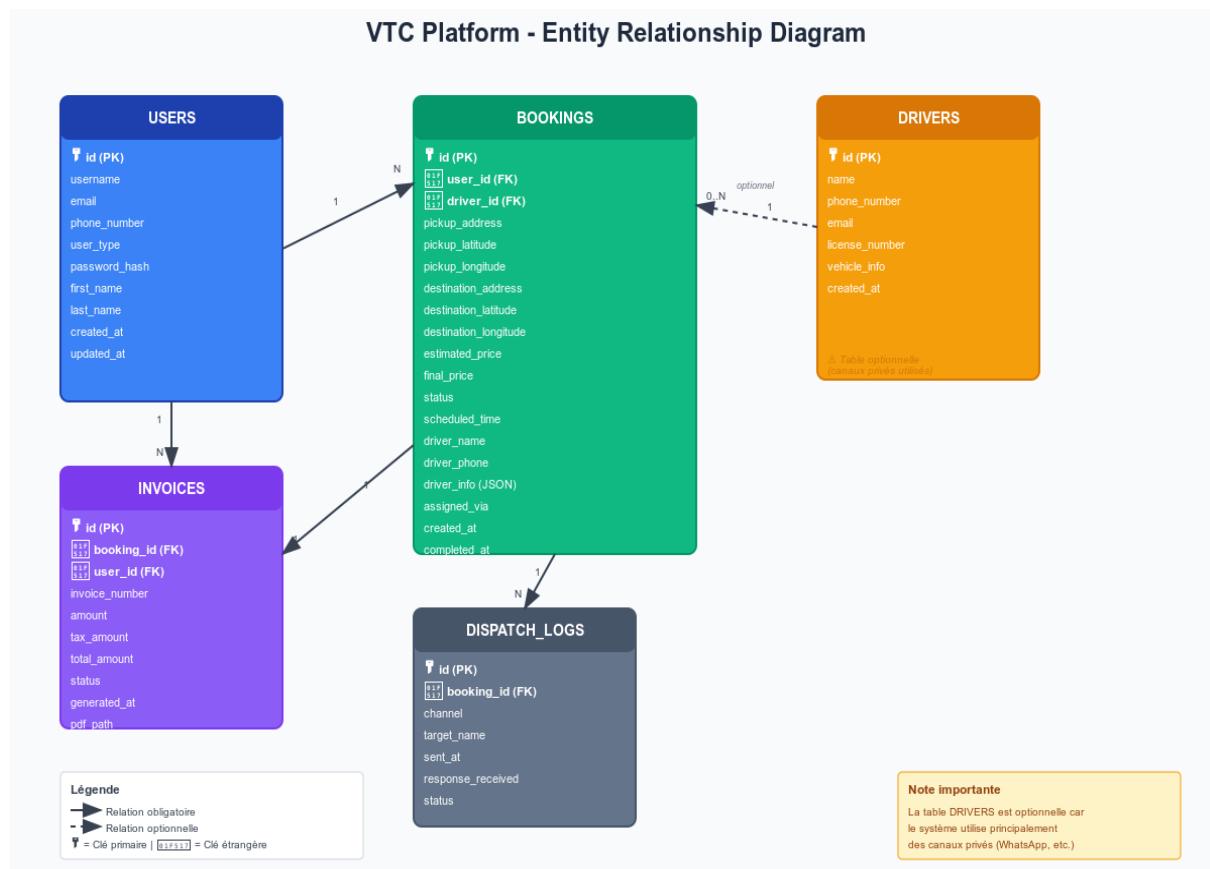
```
+ handle_webhook(payload: dict)

class DispatchService:
    # Description
        # Le DispatchService est responsable de la diffusion des demandes de courses aux chauffeurs via différents canaux (email, Telegram, etc.).
        # Dans la version actuelle, le système n'effectue pas de traitement automatique des réponses reçues.
        # La décision finale d'assigner un chauffeur à une course est prise manuellement par un administrateur,
        # après avoir consulté la réponse du chauffeur sur l'interface de communication (ex: Telegram).

    # Methods
    + create_dispatch_request(booking)
    + broadcast_to_channels(booking, channels_config)
    + manage_dispatch_queue()
```

## 7.2 Conception de la Base de données (PostgreSQL)

### Diagramme des relations internes



### Definition des Tables :

*USERS Table*

**CREATE TABLE** users (

```

    id SERIAL PRIMARY KEY,
    username VARCHAR(150) UNIQUE NOT NULL,
    email VARCHAR(254) UNIQUE NOT NULL,
    phone_number VARCHAR(15),
    user_type VARCHAR(10) CHECK (user_type IN ('CLIENT', 'ADMIN')),
    password_hash VARCHAR(128) NOT NULL,
    first_name VARCHAR(30),
    last_name VARCHAR(150),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

*DRIVERS Table*

```
CREATE TABLE drivers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(15) UNIQUE NOT NULL,
    email VARCHAR(254) UNIQUE NOT NULL,
    license_number VARCHAR(50) UNIQUE NOT NULL,
    vehicle_info TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

*BOOKINGS Table*

```
CREATE TABLE bookings (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    driver_id INTEGER REFERENCES drivers(id) ON DELETE SET NULL,
    pickup_address VARCHAR(255) NOT NULL,
    pickup_latitude DECIMAL(10,8) NOT NULL,
    pickup_longitude DECIMAL(11,8) NOT NULL,
    destination_address VARCHAR(255) NOT NULL,
    destination_latitude DECIMAL(10,8) NOT NULL,
    destination_longitude DECIMAL(11,8) NOT NULL,
    estimated_price DECIMAL(8,2) NOT NULL,
    final_price DECIMAL(8,2),
    status VARCHAR(20) CHECK (status IN ('PENDING', 'CONFIRMED', 'DRIVER_ASSIGNED',
    'IN_PROGRESS', 'COMPLETED', 'CANCELLED')),
    scheduled_time TIMESTAMP WITH TIME ZONE NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    completed_at TIMESTAMP WITH TIME ZONE
);
```

*INVOICES Table*

```
CREATE TABLE invoices (
    id SERIAL PRIMARY KEY,
    booking_id INTEGER REFERENCES bookings(id) ON DELETE CASCADE,
    invoice_number VARCHAR(50) UNIQUE NOT NULL,
    amount DECIMAL(8,2) NOT NULL,
    tax_amount DECIMAL(8,2) DEFAULT 0.00,
    total_amount DECIMAL(8,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'GENERATED' CHECK (status IN ('GENERATED', 'SENT',
    'PAID')),
    generated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
```

```
pdf_path VARCHAR(255)
);
```

## Database Constraints

- users.user\_type must be ‘CLIENT’ or ‘ADMIN’
- bookings.status must be valid booking status
- invoices.status must be valid invoice status

## 7.3 Composantes du Frontend (React)

### 7.3.1 Principales composantes de l’application

#### App Component

```
const App = () => {
  // State Management
  - user: User | null
  - isAuthenticated: boolean
  - loading: boolean

  // Methods
  + handleLogin()
  + handleLogout()
  + checkAuthStatus()
}
```

### 7.3.2 Composantes de l’Authentification

#### LoginForm

```
const LoginForm = () => {
  // State
  - email: string
  - password: string
  - errors: object
  - isSubmitting: boolean

  // Methods
  + validateForm()
  + handleSubmit()
  + handleInputChange()
}
```

#### RegisterForm

```
const RegisterForm = () => {
  // State
  - formData: UserRegistrationData
  - errors: ValidationErrors
  - step: number (multi-step form)

  // Methods
  + validateStep()
  + nextStep()
  + previousStep()
  + handleSubmit()
}
```

### 7.3.3 Composantes de la Réservation

#### BookingForm

```
const BookingForm = () => {
  // State
  - pickupAddress: string
  - destinationAddress: string
  - scheduledTime: Date
  - estimatedPrice: number
  - isCalculating: boolean

  // Methods
  + handleAddressChange()
  + calculatePrice()
  + submitBooking()
  + validateAddresses()
}
```

#### BookingStatusTracker

```
const BookingStatusTracker = ({ bookingId }) => {
  // State
  - booking: Booking
  - status: BookingStatus
  - driverInfo: Driver | null
  - updates: StatusUpdate[]

  // Methods
  + fetchBookingStatus()
  + subscribeToUpdates()
  + handleStatusChange()
```

```
}
```

### 7.3.4 Composantes du Dashboard de l'utilisateur

#### UserDashboard

```
const UserDashboard = () => {
  // State
  - recentBookings: Booking[]
  - upcomingBookings: Booking[]
  - userStats: UserStatistics

  // Methods
  + fetchUserData()
  + refreshDashboard()
```

```
}
```

#### BookingHistory

```
const BookingHistory = () => {
  // State
  - bookings: Booking[]
  - filters: FilterOptions
  - pagination: PaginationState

  // Methods
  + fetchBookings()
  + applyFilters()
  + handlePagination()
  + downloadInvoice()
```

```
}
```

### 7.3.5 Composantes du Dashboard de l'administrateur

#### AdminDashboard

```
const AdminDashboard = () => {
  // State
  - pendingBookings: Booking[]
  - activeBookings: Booking[]
  - drivers: Driver[]

  // Methods
  + fetchDashboardData()
  + assignDriver()
```

```
+ handleBookingUpdate()  
}  
}
```

### **BookingQueue**

```
const BookingQueue = () => {  
  // State  
  - pendingBookings: Booking[]  
  - selectedBooking: Booking | null  
  - availableDrivers: Driver[]  
  
  // Methods  
  + selectBooking()  
  + assignDriver()  
  + refreshQueue()  
  + handleBookingAction()  
}  
}
```

### **7.3.6 Composantes de Fonctionnalités**

#### **AddressAutocomplete**

```
const AddressAutocomplete = ({ onAddressSelect, placeholder }) => {  
  // State  
  - query: string  
  - suggestions: AddressSuggestion[]  
  - isLoading: boolean  
  
  // Methods  
  + handleInputChange()  
  + fetchSuggestions()  
  + selectAddress()  
  + getCurrentLocation()  
}  
}
```

#### **Map Component**

```
const Map = ({ pickup, destination, driver }) => {  
  // State  
  - mapInstance: MapInstance  
  - markers: Marker[]  
  - route: Route | null  
  
  // Methods  
  + initializeMap()
```

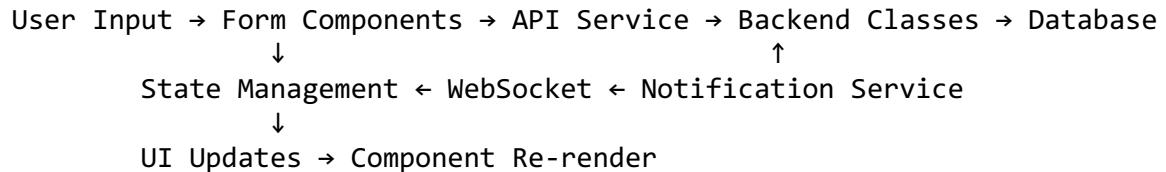
```
+ addMarkers()  
+ showRoute()  
+ updateDriverLocation()  
}
```

### NotificationToast

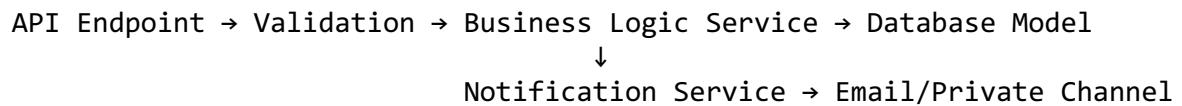
```
const NotificationToast = () => {  
    // State  
    - notifications: Notification[]  
    - isVisible: boolean  
  
    // Methods  
    + showNotification()  
    + hideNotification()  
    + handleAction()  
}
```

## 7.4 Intéractions des composantes

### 7.4.1 Frontend Data Flow



### 7.4.2 Backend Service Interactions



### 7 .4.4 Key Integration Points

#### Authentication Flow

LoginForm → AuthService → JWT Token → Protected Routes

#### Booking Creation Flow

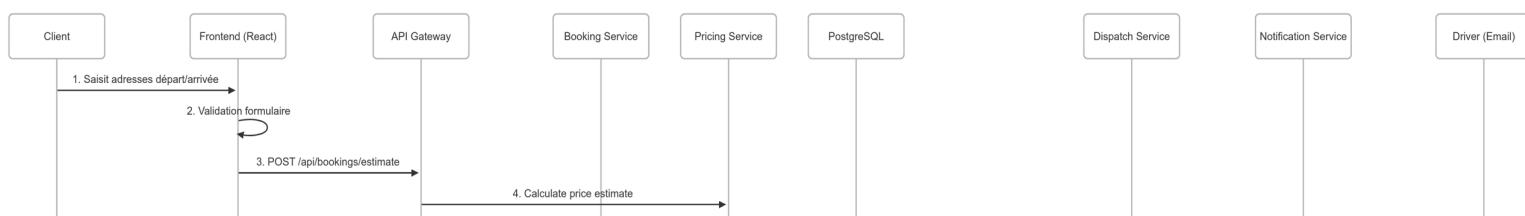
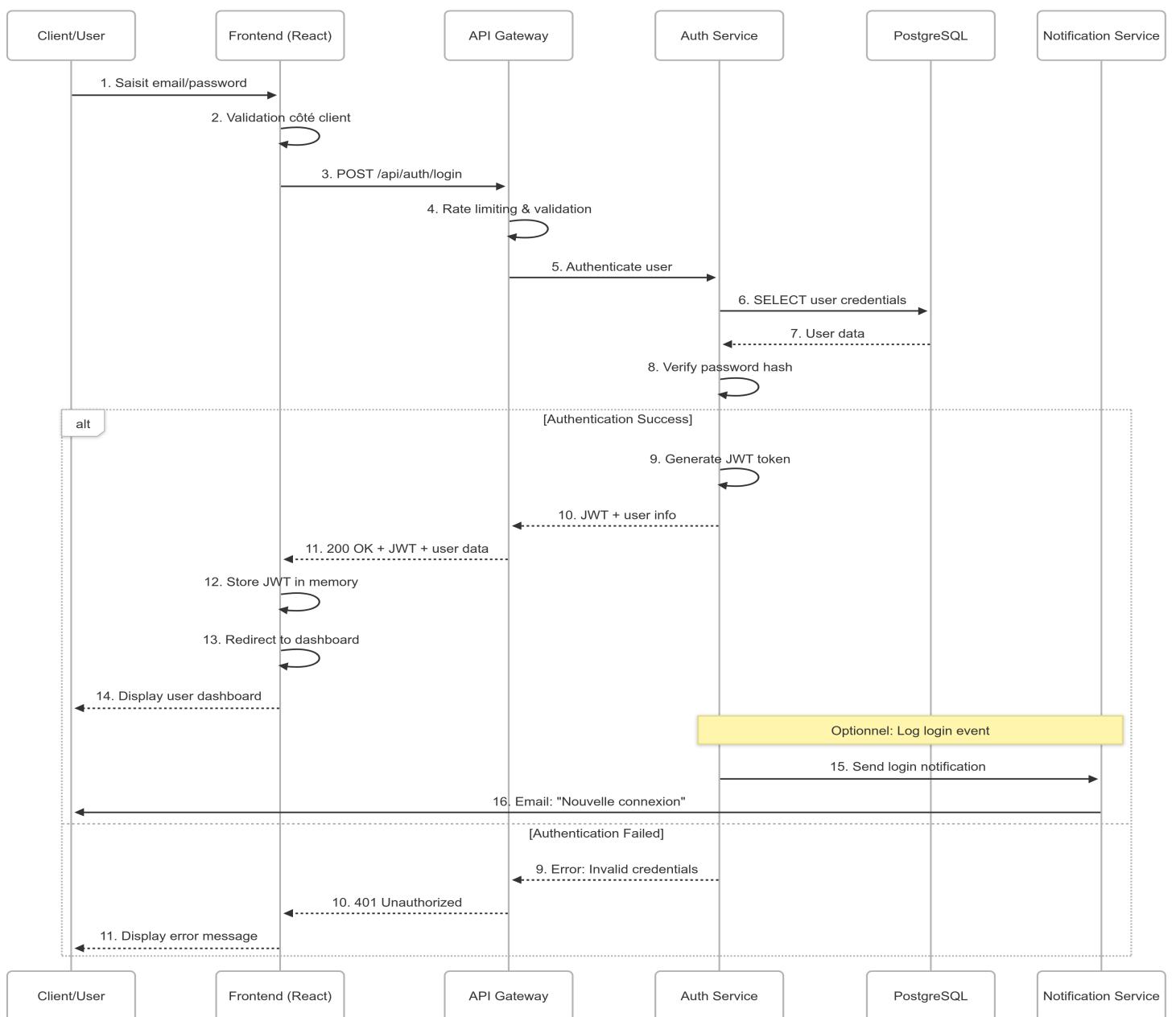
BookingForm → PricingService → BookingService → DispatchService → NotificationService

### Admin Dashboard Flow

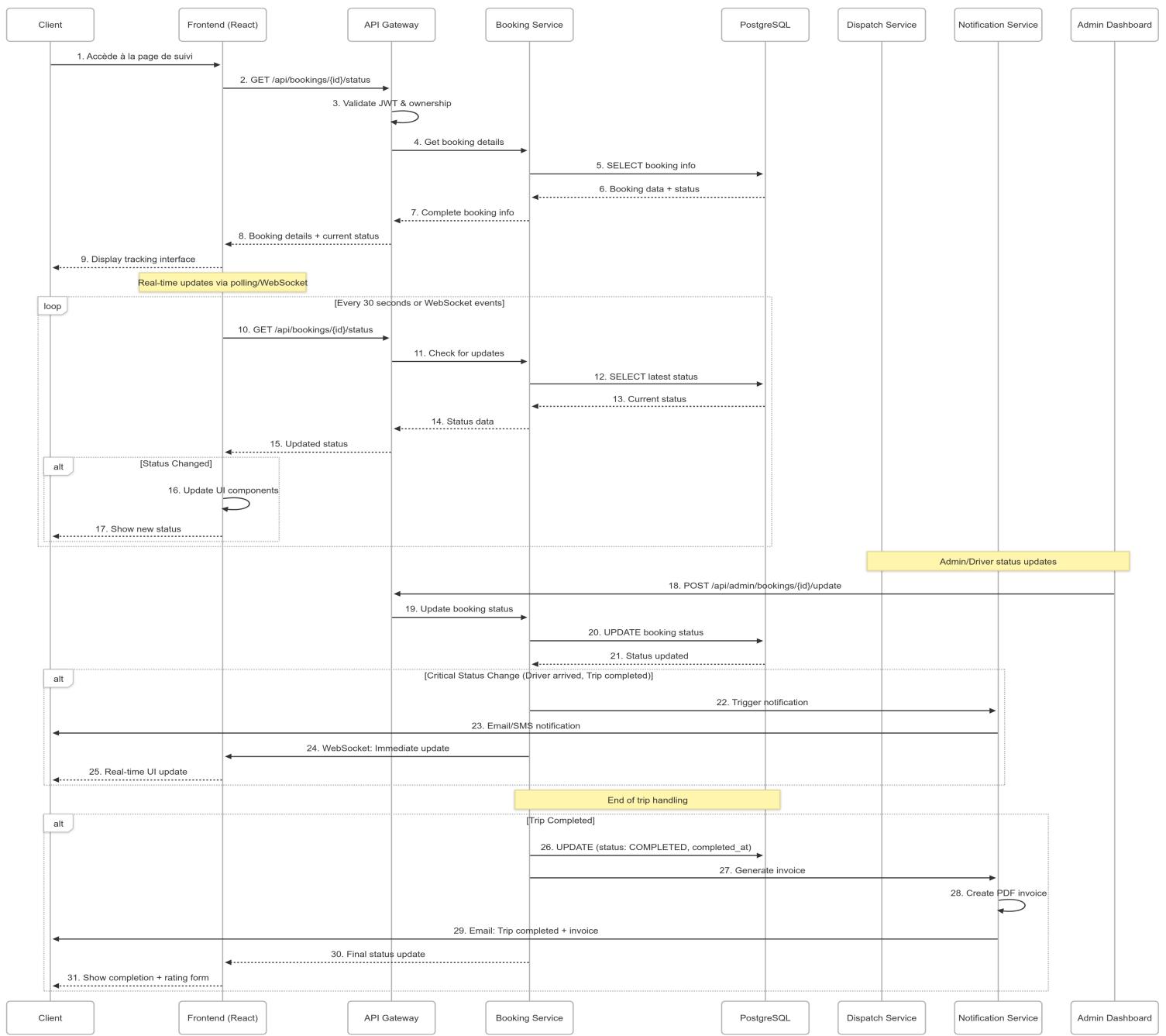
BookingQueue → DispatchService → Driver Assignment

## 8. Diagrammes de Séquence - Plateforme VTC

### 8.1. Authentification Utilisateur (User Login)



### 8.3. Suivi de Commande en Temps Réel (Order Tracking)



## **8.4 Légende des Composants**

### **8.4.1 Frontend (React)**

- Interface utilisateur responsive
- Gestion d'état local et validation
- Communication API REST

### **8.4.2 API Gateway**

- Point d'entrée unique pour toutes les requêtes
- Rate limiting et validation des requêtes
- Authentification JWT

### **8.4.3 Services Backend (Django)**

- Auth Service : Gestion authentification et autorisation
- Booking Service: Logique métier des réservations
- Pricing Service: Calcul des tarifs et estimations
- Dispatch Service: Attribution des chauffeurs
- Notification Service: Emails et communications

### **8.4.4 PostgreSQL**

- Base de données principale
- Tables: users, bookings, drivers, invoices

### **8.4.5 External Actors**

- Driver: Communication via email
- Admin Dashboard: Interface d'administration

### **8.4.6 Points Clés des Interactions**

1. Sécurité: Validation JWT sur toutes les requêtes authentifiées
2. Résilience: Gestion des erreurs et timeouts à chaque étape

3. Temps Réel: Combinaison WebSocket + polling pour le suivi
4. Notifications: Communications multi-canal (email prioritaire)
5. Audit: Logging des actions critiques pour traçabilité

## 9.1 APIs Externes

### 9.1.1 OpenStreetMap & Services Associés

**Services utilisés :**

- Nominatim (Geocoding)
- OpenRouteService (Distance Matrix, calculs d'itinéraires)
- Photon (Autocomplétion d'adresses)
- Leaflet.js (Affichage cartographique)

**Justification du choix :**

Coût réduit : OpenStreetMap est une solution libre et gratuite, adaptée à un usage commercial via auto-hébergement ou prestataires tiers à faible coût.

Couverture mondiale : Données cartographiques contributives très riches, notamment en Europe (France bien couverte).

Modularité : Chaque service (geocoding, routing, autocomplétion) est indépendant, permettant une architecture souple et personnalisée.

Open Source : Code ouvert, personnalisable et auto-hébergeable.

Communauté active : Développement continu et documentation abondante.

**Fonctionnalités utilisées :**

Nominatim (Geocoding API) :

Conversion adresses ↔ coordonnées géographiques

Utilisé pour localiser les adresses de départ et d'arrivée

OpenRouteService (Distance Matrix / Routing) :

Calcul des distances et durées de trajet en voiture

Génération d'itinéraires optimisés entre deux points

**Photon (Places API - Autocomplétion) :**

Suggestion d'adresses pendant la saisie utilisateur

Fonctionnalité de recherche rapide et conviviale

Leaflet.js (Cartographie interactive) :

Affichage dynamique des cartes

Ajout de marqueurs, trajets, zones, etc.

### **9.1.2 SMTP (Gmail/Google Workspace)**

**Service utilisé:** Gmail SMTP ou Google Workspace SMTP

**Justification du choix:**

- **Fiabilité:** Infrastructure email robuste de Google
- **Délivrabilité:** Excellente réputation des serveurs Google
- **Sécurité:** Chiffrement TLS et authentification OAuth2 - **Intégration Django:** Support natif avec django.core.mail
- **Coût:** Gratuit pour Gmail, peu coûteux pour Google Workspace

**Configuration:**

- **Serveur SMTP:** smtp.gmail.com
- **Port:** 587 (TLS) - **Authentification:** OAuth2 ou App Passwords
- **Limite:** 500 emails/jour (Gmail), 2000/jour (Google Workspace)

### **9.1.3 Telegram Bot API**

**Service utilisé:** Telegram Bot API

**Justification du choix:**

- **Communication directe:** Canal de dispatch direct avec les chauffeurs
- **Instantané:** Notifications en temps réel via mobile
- **Gratuit:** API gratuite sans limitation stricte
- **Fiabilité:** Infrastructure Telegram stable
- **Interface simple:** Messages texte et boutons interactifs

**Fonctionnalités utilisées:**

- **Envoi de messages:** Notifications de nouvelles courses
- **Inline keyboards:** Boutons Accepter/Refuser
- **Webhooks:** Réception des réponses des chauffeurs

- **File uploads:** Envoi de détails de course (optionnel)

**Configuration:**

- **URL de base:** [https://api.telegram.org/bot{BOT\\_TOKEN}/](https://api.telegram.org/bot{BOT_TOKEN}/) - **Rate limit:** 30 messages/seconde - **Webhooks:** Configuration pour réponses temps réel

## 9.2. API Interne - Plateforme VTC

### 9.2.1 Authentification

*POST /api/auth/login*

**Description:** Authentification utilisateur

```
{  
    "method": "POST",  
    "path": "/api/auth/login",  
    "input": {  
        "email": "string",  
        "password": "string"  
    },  
    "output": {  
        "success": true,  
        "data": {  
            "access_token": "string",  
            "refresh_token": "string",  
            "user": {  
                "id": "integer",  
                "email": "string",  
                "first_name": "string",  
                "last_name": "string",  
                "user_type": "CLIENT|ADMIN"  
            }  
        }  
    }  
}
```

*POST /api/auth/register*

**Description:** Création de compte utilisateur

```
{  
    "method": "POST",  
    "path": "/api/auth/register",  
    "input": {  
        "email": "string",  
        "password": "string",  
        "first_name": "string",  
    }  
}
```

```

    "last_name": "string",
    "phone_number": "string"
},
"output": {
    "success": true,
    "data": {
        "user_id": "integer",
        "message": "User created successfully"
    }
}
}

```

*POST /api/auth/refresh*

**Description:** Renouvellement du token d'accès

```

{
    "method": "POST",
    "path": "/api/auth/refresh",
    "input": {
        "refresh_token": "string"
    },
    "output": {
        "success": true,
        "data": {
            "access_token": "string"
        }
    }
}

```

## 2.2 Gestion des Réservations

*POST /api/bookings/estimate*

**Description:** Estimation de prix avant réservation

```

{
    "method": "POST",
    "path": "/api/bookings/estimate",
    "headers": {
        "Authorization": "Bearer {access_token}"
    },
    "input": {
        "pickup_address": "string",

```

```

    "destination_address": "string",
    "scheduled_time": "ISO 8601 datetime"
},
"output": {
    "success": true,
    "data": {
        "estimated_price": "decimal",
        "distance_km": "decimal",
        "estimated_duration_minutes": "integer",
        "pickup_coordinates": {
            "latitude": "decimal",
            "longitude": "decimal"
        },
        "destination_coordinates": {
            "latitude": "decimal",
            "longitude": "decimal"
        }
    }
}
}
}

```

*POST /api/bookings/create*

**Description:** Création d'une nouvelle réservation

```
{
    "method": "POST",
    "path": "/api/bookings/create",
    "headers": {
        "Authorization": "Bearer {access_token}"
    },
    "input": {
        "pickup_address": "string",
        "pickup_latitude": "decimal",
        "pickup_longitude": "decimal",
        "destination_address": "string",
        "destination_latitude": "decimal",
        "destination_longitude": "decimal",
        "scheduled_time": "ISO 8601 datetime",
        "estimated_price": "decimal"
    },
    "output": {
        "success": true,
        "data": {
            "booking_id": "integer",

```

```

        "status": "PENDING",
        "confirmation_number": "string",
        "estimated_price": "decimal"
    }
}
}
```

*GET /api/bookings/{booking\_id}*

**Description:** Récupération des détails d'une réservation

```
{
  "method": "GET",
  "path": "/api/bookings/{booking_id}",
  "headers": {
    "Authorization": "Bearer {access_token}"
  },
  "output": {
    "success": true,
    "data": {
      "id": "integer",
      "status":
        "PENDING|CONFIRMED|DRIVER_ASSIGNED|IN_PROGRESS|COMPLETED|CANCELLED",
      "pickup_address": "string",
      "destination_address": "string",
      "scheduled_time": "ISO 8601 datetime",
      "estimated_price": "decimal",
      "final_price": "decimal|null",
      "driver": {
        "name": "string",
        "phone_number": "string",
        "vehicle_info": "string"
      } || null,
      "created_at": "ISO 8601 datetime",
      "completed_at": "ISO 8601 datetime|null"
    }
  }
}
```

*GET /api/bookings/user/{user\_id}*

**Description:** Historique des réservations d'un utilisateur

```
{
  "method": "GET",
  "path": "/api/bookings/user/{user_id}",
  "headers": {
```

```

    "Authorization": "Bearer {access_token}"
},
"query_parameters": {
    "page": "integer (default: 1)",
    "limit": "integer (default: 10)",
    "status": "string (optional)"
},
"output": {
    "success": true,
    "data": {
        "bookings": [
            {
                "id": "integer",
                "status": "string",
                "pickup_address": "string",
                "destination_address": "string",
                "scheduled_time": "ISO 8601 datetime",
                "final_price": "decimal",
                "created_at": "ISO 8601 datetime"
            }
        ],
        "pagination": {
            "page": "integer",
            "limit": "integer",
            "total": "integer",
            "total_pages": "integer"
        }
    }
}
}

```

*PATCH /api/bookings/{booking\_id}/status*

**Description:** Mise à jour du statut d'une réservation (Admin)

```

{
    "method": "PATCH",
    "path": "/api/bookings/{booking_id}/status",
    "headers": {
        "Authorization": "Bearer {access_token}"
    },
    "input": {
        "status": "CONFIRMED|DRIVER_ASSIGNED|IN_PROGRESS|COMPLETED|CANCELLED",
        "driver_id": "integer (optional, for DRIVER_ASSIGNED)",
        "final_price": "decimal (optional, for COMPLETED)"
    }
}

```

```

},
"output": {
  "success": true,
  "data": {
    "booking_id": "integer",
    "new_status": "string",
    "updated_at": "ISO 8601 datetime"
  }
}
}
}

```

### 9.2.3 Gestion des Chauffeurs

*GET /api/drivers/*

**Description:** Liste des chauffeurs (Admin)

```

{
  "method": "GET",
  "path": "/api/drivers/",
  "headers": {
    "Authorization": "Bearer {access_token}"
  },
  "query_parameters": {
    "available": "boolean (optional)"
  },
  "output": {
    "success": true,
    "data": {
      "drivers": [
        {
          "id": "integer",
          "name": "string",
          "phone_number": "string",
          "email": "string",
          "license_number": "string",
          "vehicle_info": "string",
          "created_at": "ISO 8601 datetime"
        }
      ]
    }
  }
}
```

*POST /api/drivers/create*

**Description:** Ajout d'un nouveau chauffeur (Admin)

```
{
  "method": "POST",
  "path": "/api/drivers/create",
  "headers": {
    "Authorization": "Bearer {access_token}"
  },
  "input": {
    "name": "string",
    "phone_number": "string",
    "email": "string",
    "license_number": "string",
    "vehicle_info": "string"
  },
  "output": {
    "success": true,
    "data": {
      "driver_id": "integer",
      "message": "Driver created successfully"
    }
  }
}
```

#### 9.2.4 Dispatch et Notifications

*POST /api/dispatch/assign*

**Description:** Attribution manuelle d'un chauffeur (Admin)

```
{
  "method": "POST",
  "path": "/api/dispatch/assign",
  "headers": {
    "Authorization": "Bearer {access_token}"
  },
  "input": {
    "booking_id": "integer",
    "driver_id": "integer"
  },
  "output": {
    "success": true,
    "data": {
      "booking_id": "integer",
      "driver_id": "integer",
      "assignment_time": "ISO 8601 datetime"
    }
  }
}
```

```
}
```

*POST /api/dispatch/broadcast*

**Description:** Diffusion d'une course aux chauffeurs disponibles (Admin)

```
{
  "method": "POST",
  "path": "/api/dispatch/broadcast",
  "headers": {
    "Authorization": "Bearer {access_token}"
  },
  "input": {
    "booking_id": "integer",
    "channels": ["email", "telegram"]
  },
  "output": {
    "success": true,
    "data": {
      "booking_id": "integer",
      "broadcast_time": "ISO 8601 datetime",
      "channels_used": ["string"],
      "drivers_contacted": "integer"
    }
  }
}
```

## 9.2.5 Analytics et Tableaux de Bord

*GET /api/admin/dashboard*

**Description:** Données du tableau de bord administrateur

```
{
  "method": "GET",
  "path": "/api/admin/dashboard",
  "headers": {
    "Authorization": "Bearer {access_token}"
  },
  "query_parameters": {
    "period": "today|week|month (default: today)"
  },
  "output": {
    "success": true,
    "data": {

```

```

"stats": {
    "total_bookings": "integer",
    "pending_bookings": "integer",
    "completed_bookings": "integer",
    "cancelled_bookings": "integer",
    "total_revenue": "decimal",
    "average_response_time": "integer (minutes)"
},
"recent_bookings": [
    {
        "id": "integer",
        "status": "string",
        "pickup_address": "string",
        "created_at": "ISO 8601 datetime"
    }
]
}
}
}

```

### 9.2.6 Facturation

*GET /api/invoices/{booking\_id}*

**Description:** Récupération d'une facture

```

{
    "method": "GET",
    "path": "/api/invoices/{booking_id}",
    "headers": {
        "Authorization": "Bearer {access_token}"
    },
    "output": {
        "success": true,
        "data": {
            "invoice_id": "integer",
            "invoice_number": "string",
            "booking_id": "integer",
            "amount": "decimal",
            "tax_amount": "decimal",
            "total_amount": "decimal",
            "status": "GENERATED|SENT|PAID",
            "generated_at": "ISO 8601 datetime",
            "pdf_url": "string"
        }
    }
}

```

```
}
```

```
}
```

*GET /api/invoices/{invoice\_id}/download*

**Description:** Téléchargement PDF de la facture

```
{
```

```
    "method": "GET",
```

```
    "path": "/api/invoices/{invoice_id}/download",
```

```
    "headers": {
```

```
        "Authorization": "Bearer {access_token}"
```

```
    },
```

```
    "output": {
```

```
        "content_type": "application/pdf",
```

```
        "file": "PDF binary data"
```

```
    }
```

```
}
```

### 9.3. Gestion d'Erreurs Standardisée

#### 9.3.1 Format de Réponse d'Erreur

```
{
```

```
    "success": false,
```

```
    "error": {
```

```
        "code": "string",
```

```
        "message": "string",
```

```
        "details": "object (optional)"
```

```
    }
```

```
}
```

#### 9.3.2 Codes d'Erreur Principaux

- **400:** VALIDATION\_ERROR - Données d'entrée invalides
- **401:** UNAUTHORIZED - Token manquant ou invalide
- **403:** FORBIDDEN - Permissions insuffisantes
- **404:** NOT\_FOUND - Ressource introuvable
- **409:** CONFLICT - Conflit de données (ex: réservation déjà assignée)
- **429:** RATE\_LIMIT\_EXCEEDED - Limite de requêtes dépassée
- **500:** INTERNAL\_ERROR - Erreur serveur interne

### 9.4. Sécurité et Authentification

#### 9.4.1 Authentification JWT

- **Header:** Authorization: Bearer {access\_token}
- **Expiration:** 1 heure pour access\_token, 7 jours pour refresh\_token
- **Algorithme:** RS256

#### **9.4.2 Rate Limiting**

- **Général:** 1000 requêtes/heure par utilisateur incluant les estimations de prix
- **Authentification:** 5 tentatives/minute

#### **9.4.3 Validation des Données**

- Coordonnées GPS :

Vérification que les latitudes sont comprises entre -90 et +90 et les longitudes entre -180 et +180.

Empêche les coordonnées invalides ou corrompues.

- Adresses :

Validation via une requête Nominatim (OpenStreetMap) pour :

Confirmer l'existence de l'adresse saisie

Extraire les coordonnées correspondantes

Vérifier la cohérence avec les limites géographiques desservies (ex: rayon de service)

- Emails :

Validation syntaxique basée sur la norme RFC 5322 :

Regex stricte

Téléphones :

Format international conforme à la norme E.164 :

Exemple : +33612345678

Implémenté avec une librairie comme libphonenumbers (Google) ou PhoneNumber.js

### **9.5. Documentation API Interactive**

L'API sera documentée avec Swagger/OpenAPI 3.0 et accessible à l'adresse :

- **URL:** /api/docs/
- **Format:** Interface Swagger UI interactive
- **Authentification:** Support des tokens JWT dans l'interface
- **Tests:** Possibilité de tester les endpoints directement