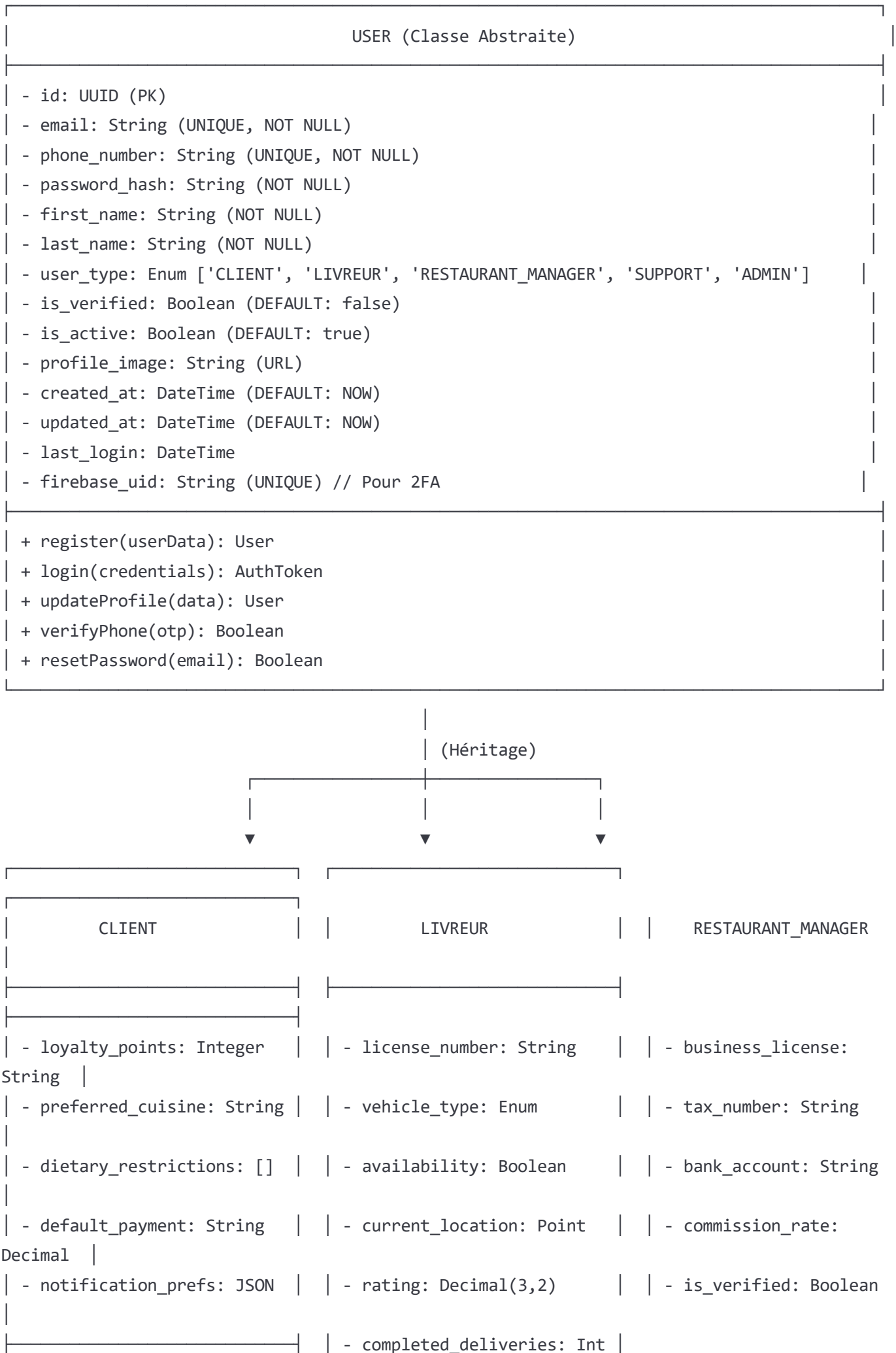
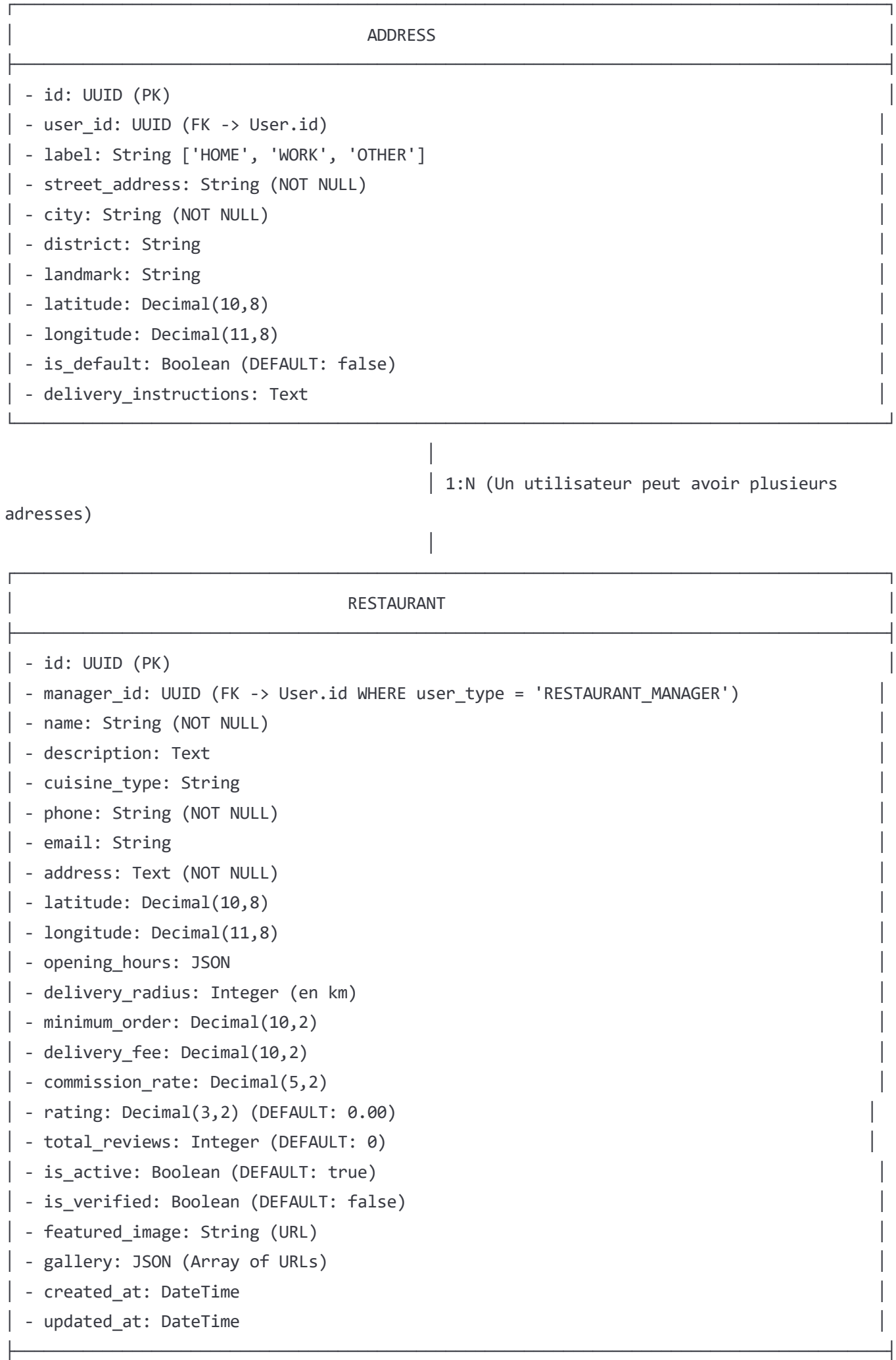
Eat Fast - Diagramme de Classes et Architecture Backend

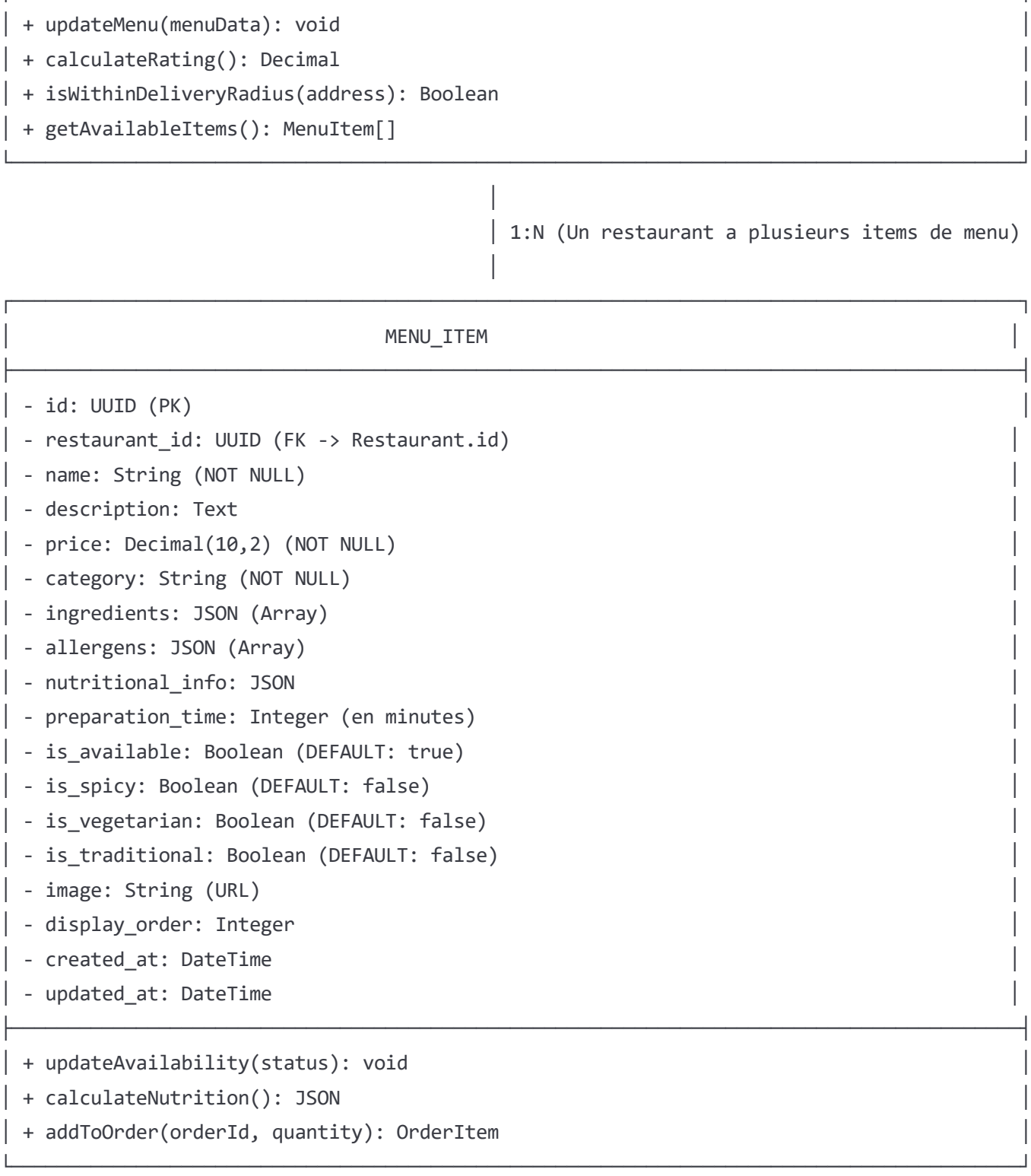
1. Diagramme de Classes Détaillé

Classes Principales avec Relations et Cardinalités

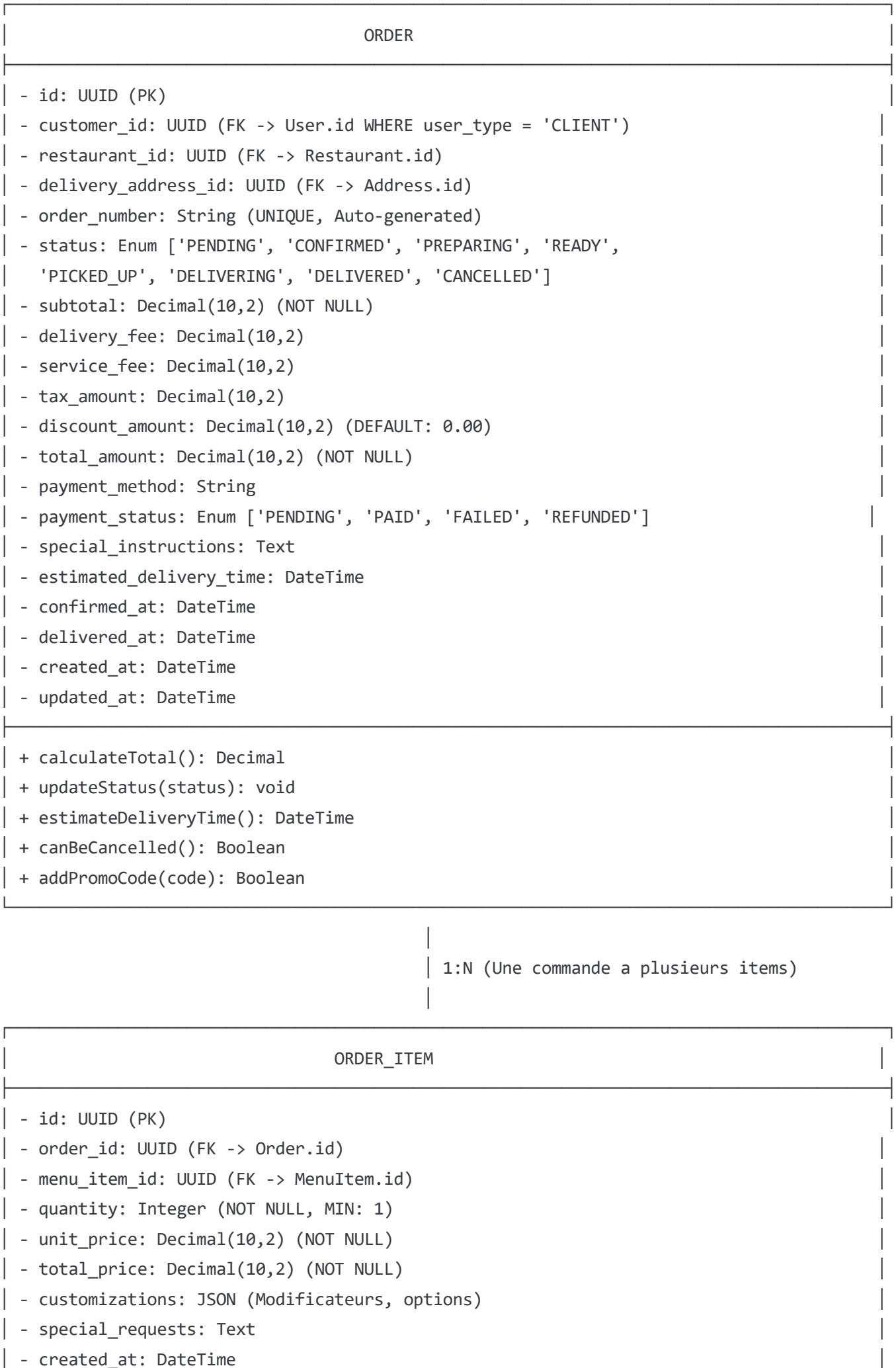

USER (Classe Abstraite)
- id: UUID (PK)
- email: String (UNIQUE, NOT NULL)
- phone_number: String (UNIQUE, NOT NULL)
- password_hash: String (NOT NULL)
- first_name: String (NOT NULL)
- last_name: String (NOT NULL)
- user_type: Enum ['CLIENT', 'LIVREUR', 'RESTAURANT_MANAGER', 'SUPPORT', 'ADMIN']
- is_verified: Boolean (DEFAULT: false)
- is_active: Boolean (DEFAULT: true)
- profile_image: String (URL)
- created_at: DateTime (DEFAULT: NOW)
- updated_at: DateTime (DEFAULT: NOW)
- last_login: DateTime
- firebase_uid: String (UNIQUE) // Pour 2FA
+ register(userData): User
+ login(credentials): AuthToken
+ updateProfile(data): User
+ verifyPhone(otp): Boolean
+ resetPassword(email): Boolean
(Héritage)
CLIENT
- loyalty_points: Integer
- preferred_cuisine: String
- dietary_restrictions: []
- default_payment: String
- notification_prefs: JSON
LIVREUR
- license_number: String
- vehicle_type: Enum
- availability: Boolean
- current_location: Point
- rating: Decimal(3,2)
- completed_deliveries: Int
RESTAURANT_MANAGER
- business_license: String
- tax_number: String
- bank_account: String
- commission_rate: Decimal
- is_verified: Boolean

Relations entre Classes


ADDRESS
- id: UUID (PK)
- user_id: UUID (FK -> User.id)
- label: String ['HOME', 'WORK', 'OTHER']
- street_address: String (NOT NULL)
- city: String (NOT NULL)
- district: String
- landmark: String
- latitude: Decimal(10,8)
- longitude: Decimal(11,8)
- is_default: Boolean (DEFAULT: false)
- delivery_instructions: Text
1:N (Un utilisateur peut avoir plusieurs adresses)
RESTAURANT
- id: UUID (PK)
- manager_id: UUID (FK -> User.id WHERE user_type = 'RESTAURANT_MANAGER')
- name: String (NOT NULL)
- description: Text
- cuisine_type: String
- phone: String (NOT NULL)
- email: String
- address: Text (NOT NULL)
- latitude: Decimal(10,8)
- longitude: Decimal(11,8)
- opening_hours: JSON
- delivery_radius: Integer (en km)
- minimum_order: Decimal(10,2)
- delivery_fee: Decimal(10,2)
- commission_rate: Decimal(5,2)
- rating: Decimal(3,2) (DEFAULT: 0.00)
- total_reviews: Integer (DEFAULT: 0)
- is_active: Boolean (DEFAULT: true)
- is_verified: Boolean (DEFAULT: false)
- featured_image: String (URL)
- gallery: JSON (Array of URLs)
- created_at: DateTime
- updated_at: DateTime



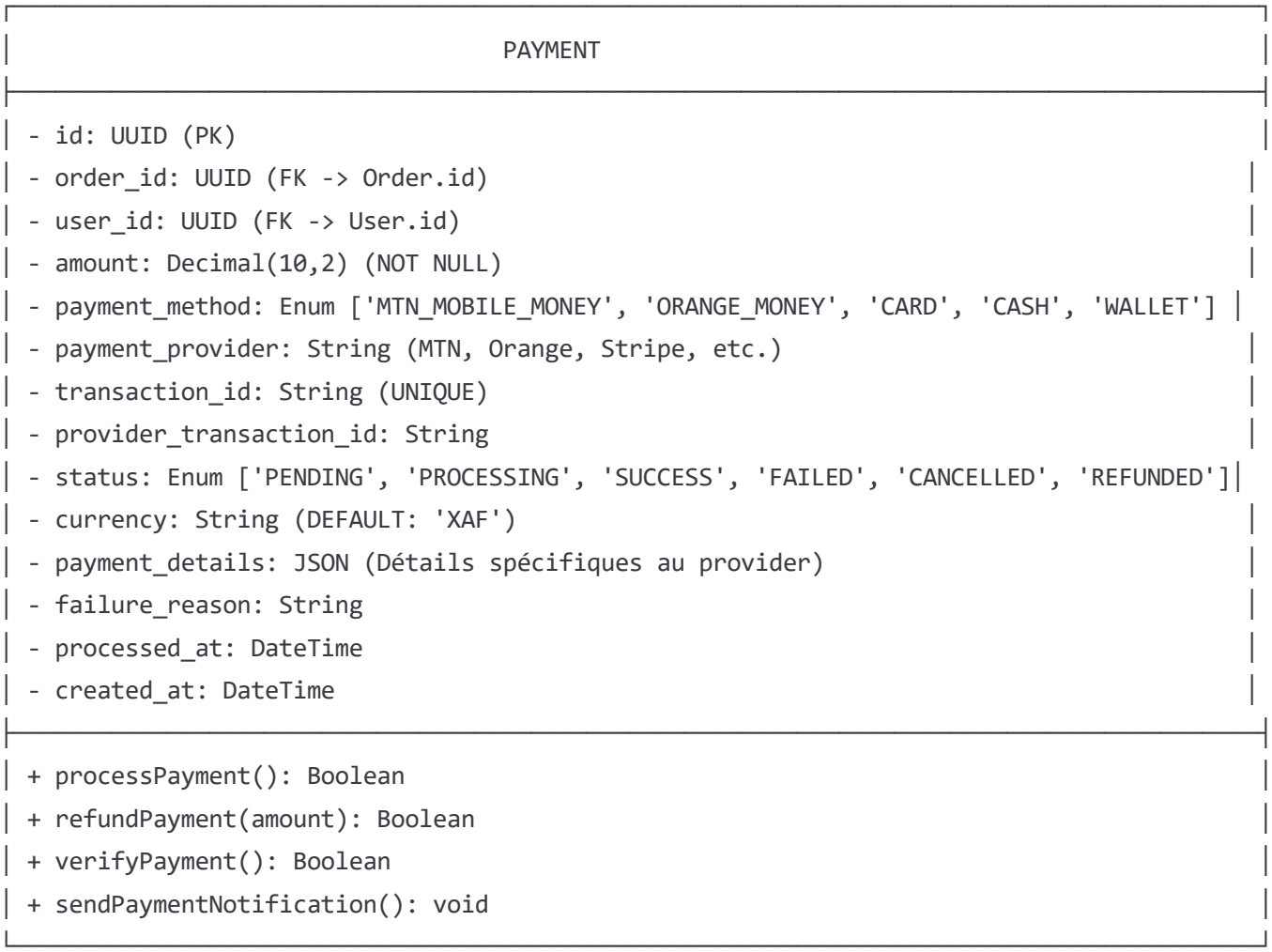
Classes de Commande et Livraison

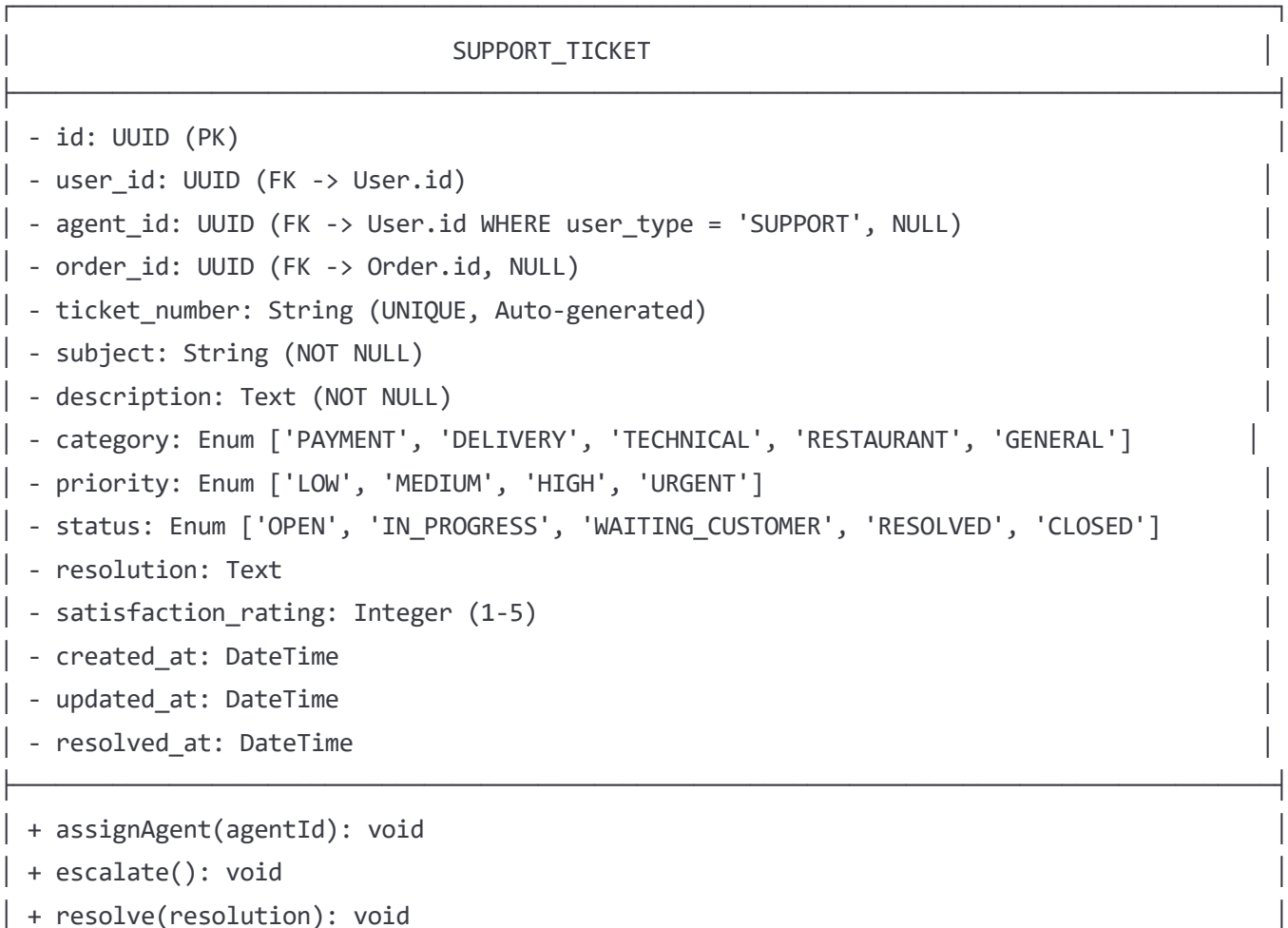

ORDER
- id: UUID (PK)
- customer_id: UUID (FK -> User.id WHERE user_type = 'CLIENT')
- restaurant_id: UUID (FK -> Restaurant.id)
- delivery_address_id: UUID (FK -> Address.id)
- order_number: String (UNIQUE, Auto-generated)
- status: Enum ['PENDING', 'CONFIRMED', 'PREPARING', 'READY',
'PICKED_UP', 'DELIVERING', 'DELIVERED', 'CANCELLED']
- subtotal: Decimal(10,2) (NOT NULL)
- delivery_fee: Decimal(10,2)
- service_fee: Decimal(10,2)
- tax_amount: Decimal(10,2)
- discount_amount: Decimal(10,2) (DEFAULT: 0.00)
- total_amount: Decimal(10,2) (NOT NULL)
- payment_method: String
- payment_status: Enum ['PENDING', 'PAID', 'FAILED', 'REFUNDED']
- special_instructions: Text
- estimated_delivery_time: DateTime
- confirmed_at: DateTime
- delivered_at: DateTime
- created_at: DateTime
- updated_at: DateTime
+ calculateTotal(): Decimal
+ updateStatus(status): void
+ estimateDeliveryTime(): DateTime
+ canBeCancelled(): Boolean
+ addPromoCode(code): Boolean
1:N (Une commande a plusieurs items)
ORDER_ITEM
- id: UUID (PK)
- order_id: UUID (FK -> Order.id)
- menu_item_id: UUID (FK -> MenuItem.id)
- quantity: Integer (NOT NULL, MIN: 1)
- unit_price: Decimal(10,2) (NOT NULL)
- total_price: Decimal(10,2) (NOT NULL)
- customizations: JSON (Modificateurs, options)
- special_requests: Text
- created_at: DateTime

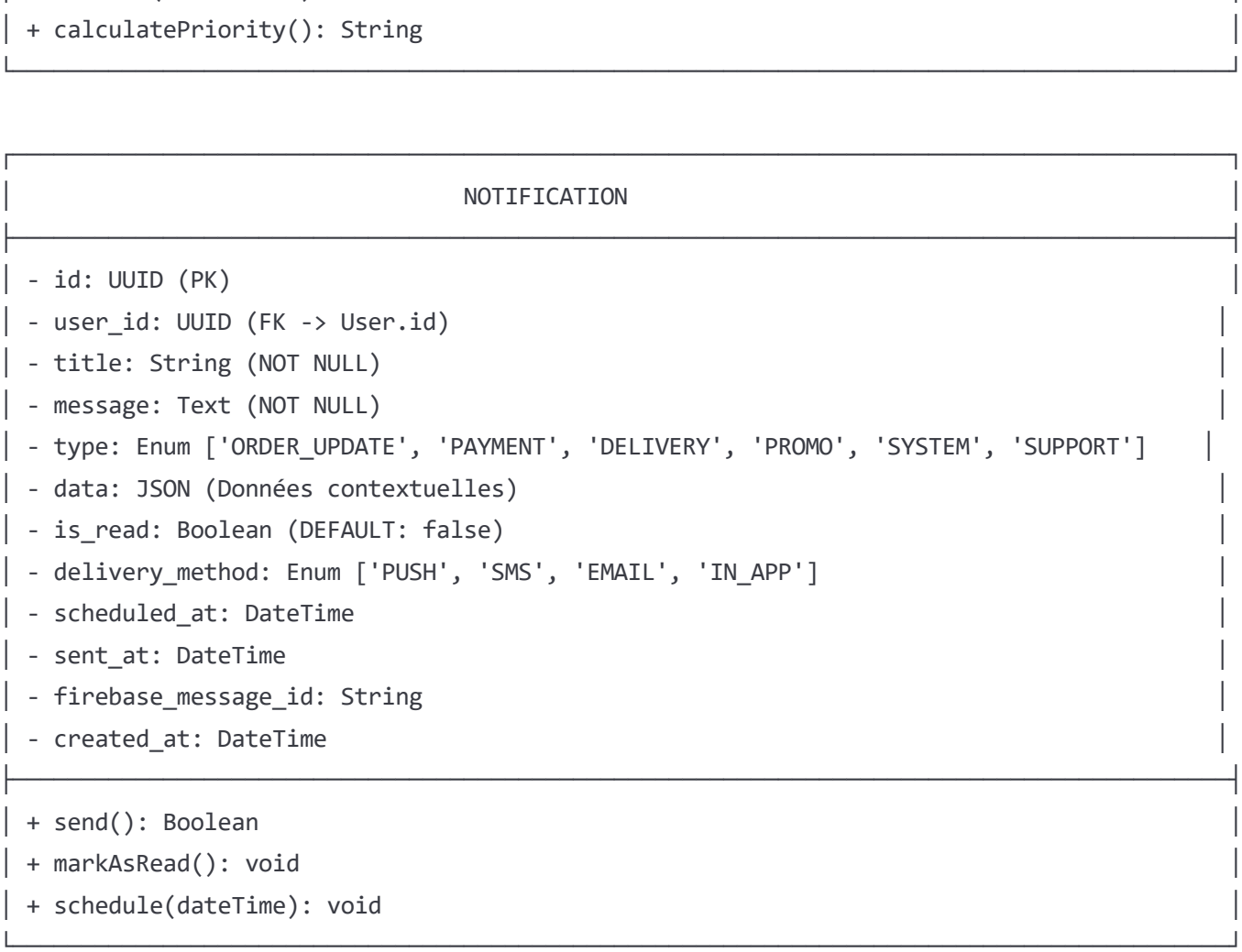
```
├──────────────────────────────────────────────────────────────────────────────┤
│ + calculateTotalPrice(): Decimal                                             │
│ + applyCustomizations(): void                                                │
└──────────────────────────────────────────────────────────────────────────────┘


┌──────────────────────────────────────────────────────────────────────────────┐
│                                   DELIVERY                                   │
├──────────────────────────────────────────────────────────────────────────────┤
│ - id: UUID (PK)                                                              │
│ - order_id: UUID (FK -> Order.id, UNIQUE)                                    │
│ - driver_id: UUID (FK -> User.id WHERE user_type = 'LIVREUR')                │
│ - pickup_address: Text (Restaurant address)                                  │
│ - delivery_address: Text                                                     │
│ - distance: Decimal(8,2) (en km)                                             │
│ - estimated_duration: Integer (en minutes)                                   │
│ - status: Enum ['ASSIGNED', 'HEADING_TO_RESTAURANT', 'AT_RESTAURANT',        │
│   'PICKED_UP', 'HEADING_TO_CUSTOMER', 'DELIVERED', 'FAILED']                 │
│ - pickup_time: DateTime                                                      │
│ - delivery_time: DateTime                                                    │
│ - driver_earnings: Decimal(10,2)                                             │
│ - delivery_route: JSON (Coordonnées GPS)                                     │
│ - delivery_proof: String (URL de l'image)                                    │
│ - customer_rating: Integer (1-5)                                             │
│ - customer_feedback: Text                                                    │
│ - created_at: DateTime                                                       │
│ - updated_at: DateTime                                                       │
├──────────────────────────────────────────────────────────────────────────────┤
│ + assignDriver(driverId): void                                               │
│ + updateLocation(coordinates): void                                          │
│ + calculateEarnings(): Decimal                                               │
│ + completeDelivery(): void                                                   │
│ + trackDelivery(): JSON                                                      │
└──────────────────────────────────────────────────────────────────────────────┘
```

Classes de Paiement et Support


PAYMENT
- id: UUID (PK)
- order_id: UUID (FK -> Order.id)
- user_id: UUID (FK -> User.id)
- amount: Decimal(10,2) (NOT NULL)
- payment_method: Enum ['MTN_MOBILE_MONEY', 'ORANGE_MONEY', 'CARD', 'CASH', 'WALLET']
- payment_provider: String (MTN, Orange, Stripe, etc.)
- transaction_id: String (UNIQUE)
- provider_transaction_id: String
- status: Enum ['PENDING', 'PROCESSING', 'SUCCESS', 'FAILED', 'CANCELLED', 'REFUNDED']
- currency: String (DEFAULT: 'XAF')
- payment_details: JSON (Détails spécifiques au provider)
- failure_reason: String
- processed_at: DateTime
- created_at: DateTime
+ processPayment(): Boolean
+ refundPayment(amount): Boolean
+ verifyPayment(): Boolean
+ sendPaymentNotification(): void


SUPPORT_TICKET
- id: UUID (PK)
- user_id: UUID (FK -> User.id)
- agent_id: UUID (FK -> User.id WHERE user_type = 'SUPPORT', NULL)
- order_id: UUID (FK -> Order.id, NULL)
- ticket_number: String (UNIQUE, Auto-generated)
- subject: String (NOT NULL)
- description: Text (NOT NULL)
- category: Enum ['PAYMENT', 'DELIVERY', 'TECHNICAL', 'RESTAURANT', 'GENERAL']
- priority: Enum ['LOW', 'MEDIUM', 'HIGH', 'URGENT']
- status: Enum ['OPEN', 'IN_PROGRESS', 'WAITING_CUSTOMER', 'RESOLVED', 'CLOSED']
- resolution: Text
- satisfaction_rating: Integer (1-5)
- created_at: DateTime
- updated_at: DateTime
- resolved_at: DateTime
+ assignAgent(agentId): void
+ escalate(): void
+ resolve(resolution): void



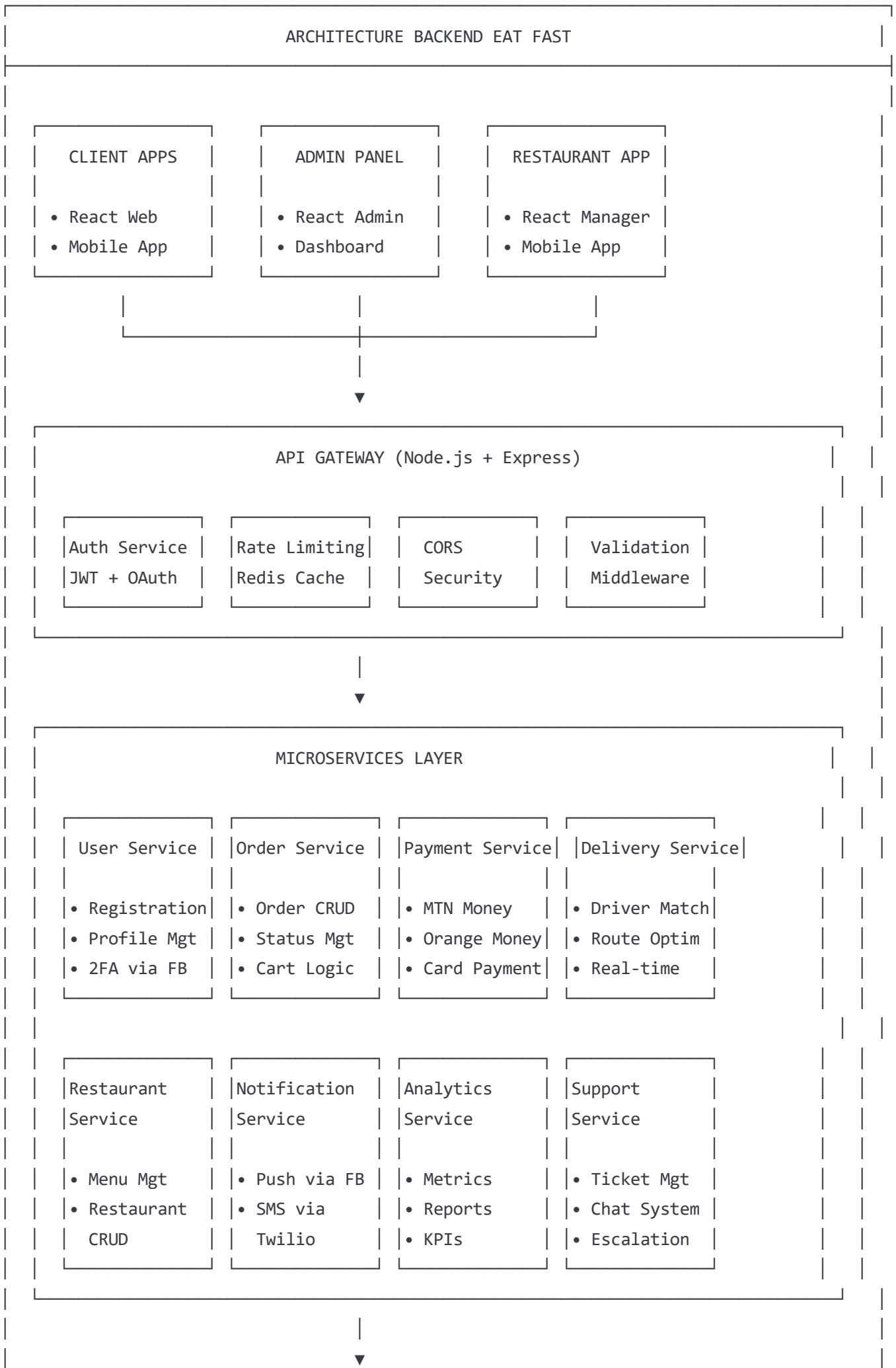
2. Cardinalités et Relations

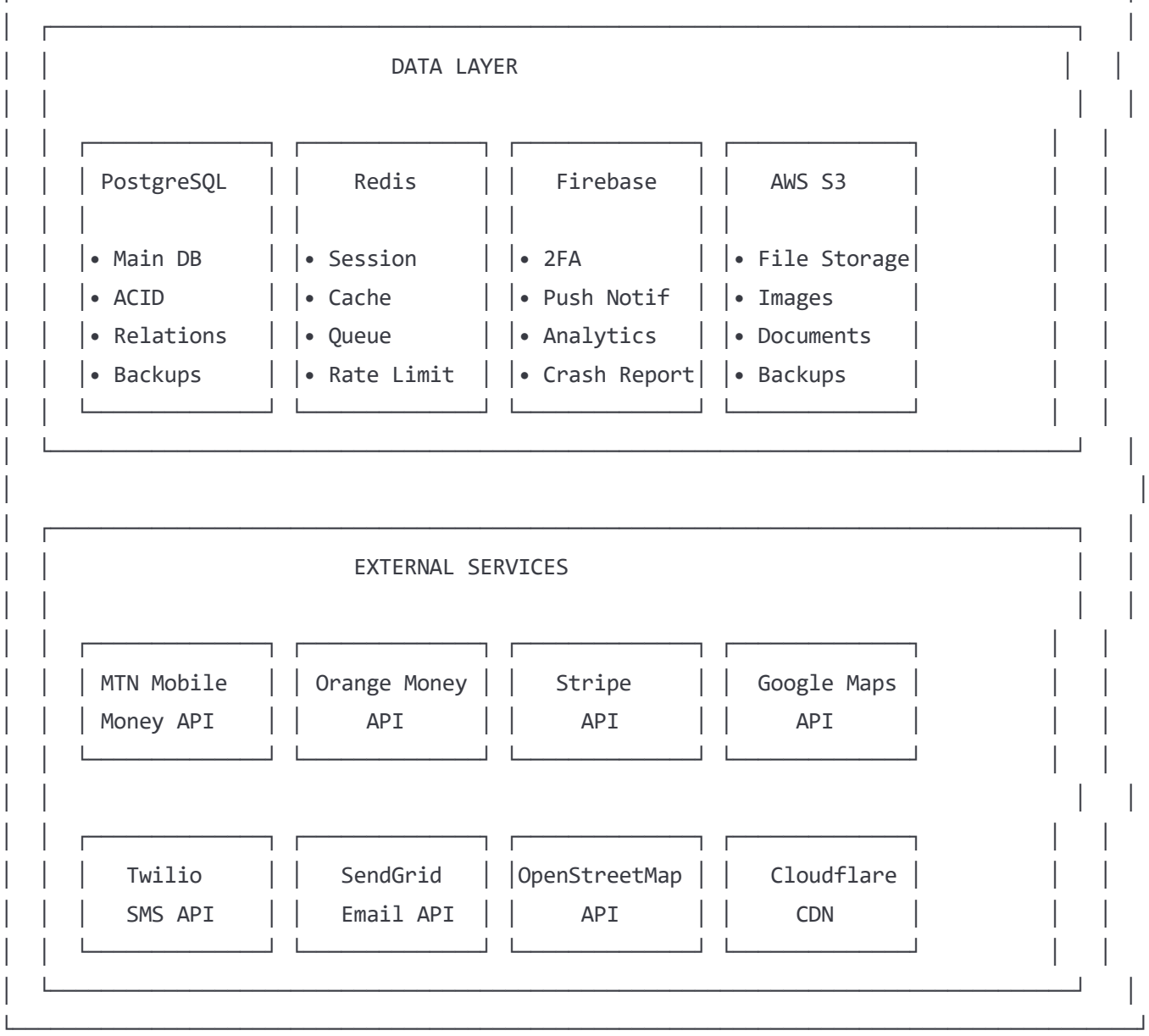
Relations Principales

1. **User** → **Address** (1:N)
   - Un utilisateur peut avoir plusieurs adresses
   - Une adresse appartient à un seul utilisateur
2. **User** → **Restaurant** (1:N)
   - Un gestionnaire peut gérer plusieurs restaurants
   - Un restaurant a un seul gestionnaire
3. **Restaurant** → **MenuItem** (1:N)
   - Un restaurant a plusieurs items de menu
   - Un item appartient à un seul restaurant
4. **User** → **Order** (1:N)
   - Un client peut passer plusieurs commandes
   - Une commande appartient à un seul client
5. **Order** → **OrderItem** (1:N)
   - Une commande contient plusieurs items
   - Un item appartient à une seule commande
6. **Order** → **Delivery** (1:1)
   - Une commande a une seule livraison
   - Une livraison concerne une seule commande
7. **User** → **Delivery** (1:N)
   - Un livreur peut effectuer plusieurs livraisons
   - Une livraison est assignée à un seul livreur
8. **Order** → **Payment** (1:N)
   - Une commande peut avoir plusieurs tentatives de paiement
   - Un paiement concerne une seule commande

3. Architecture Backend avec Node.js, Firebase et PostgreSQL

Vue d'ensemble de l'Architecture


ARCHITECTURE BACKEND EAT FAST
CLIENT APPS
• React Web
• Mobile App
ADMIN PANEL
• React Admin
• Dashboard
RESTAURANT APP
• React Manager
• Mobile App
API GATEWAY (Node.js + Express)
Auth Service
JWT + OAuth
Rate Limiting
Redis Cache
CORS
Security
Validation
Middleware
MICROSERVICES LAYER
User Service
• Registration
• Profile Mgt
• 2FA via FB
Order Service
• Order CRUD
• Status Mgt
• Cart Logic
Payment Service
• MTN Money
• Orange Money
• Card Payment
Delivery Service
• Driver Match
• Route Optim
• Real-time
Restaurant Service
• Menu Mgt
• Restaurant CRUD
Notification Service
• Push via FB
• SMS via Twilio
Analytics Service
• Metrics
• Reports
• KPIs
Support Service
• Ticket Mgt
• Chat System
• Escalation



Structure des Services Node.js

1. Service d'Authentification avec Firebase

javascript

```javascript
// services/auth.service.js
const admin = require('firebase-admin');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const { User } = require('../models');

class AuthService {
  constructor() {
    this.firebase = admin.initializeApp({
      credential: admin.credential.applicationDefault(),
      projectId: process.env.FIREBASE_PROJECT_ID
    });
  }

  async register(userData) {
    // 1. Créer l'utilisateur en base
    const hashedPassword = await bcrypt.hash(userData.password, 12);
    const user = await User.create({
      ...userData,
      password_hash: hashedPassword
    });

    // 2. Créer l'utilisateur Firebase pour 2FA
    const firebaseUser = await admin.auth().createUser({
      uid: user.id,
      email: user.email,
      phoneNumber: user.phone_number,
      displayName: `${user.first_name} ${user.last_name}`
    });

    // 3. Envoyer SMS de vérification
    await this.sendPhoneVerification(user.phone_number);

    return user;
  }

  async sendPhoneVerification(phoneNumber) {
    // Utilisation de Firebase Auth pour envoyer le code SMS
    const sessionInfo = await admin.auth().createSessionCookie(
      idToken, { expiresIn: 60 * 60 * 24 * 5 * 1000 }
    );
    return sessionInfo;
  }

  async verifyPhone(uid, verificationCode) {
```

```
  async verifyPhone(uid, verificationCode) {
    try {
      // Vérifier le code avec Firebase
      const decodedToken = await admin.auth().verifyIdToken(verificationCode);

      // Mettre à jour le statut de vérification en base
      await User.update(
        { is_verified: true },
        { where: { id: uid } }
      );

      return true;
    } catch (error) {
      throw new Error('Code de vérification invalide');
    }
  }

  async login(email, password) {
    const user = await User.findOne({ where: { email } });
    if (!user) throw new Error('Utilisateur non trouvé');

    const isPasswordValid = await bcrypt.compare(password, user.password_hash);
    if (!isPasswordValid) throw new Error('Mot de passe incorrect');

    // Générer JWT token
    const token = jwt.sign(
      { userId: user.id, userType: user.user_type },
      process.env.JWT_SECRET,
      { expiresIn: '24h' }
    );

    return { user, token };
  }
}
```

2. Service de Notification avec Firebase

javascript

```javascript
// services/notification.service.js
const admin = require('firebase-admin');
const twilio = require('twilio');
const { Notification } = require('../models');

class NotificationService {
  constructor() {
    this.twilioClient = twilio(
      process.env.TWILIO_ACCOUNT_SID,
      process.env.TWILIO_AUTH_TOKEN
    );
  }

  async sendPushNotification(userId, title, message, data = {}) {
    try {
      // 1. Sauvegarder la notification en base
      const notification = await Notification.create({
        user_id: userId,
        title,
        message,
        type: data.type || 'GENERAL',
        data: JSON.stringify(data),
        delivery_method: 'PUSH'
      });

      // 2. Envoyer via Firebase Cloud Messaging
      const fcmMessage = {
        notification: { title, body: message },
        data: { ...data, notificationId: notification.id },
        token: await this.getUserFCMToken(userId)
      };

      const response = await admin.messaging().send(fcmMessage);

      // 3. Mettre à jour avec l'ID Firebase
      await notification.update({
        firebase_message_id: response,
        sent_at: new Date()
      });

      return response;
    } catch (error) {
      console.error('Erreur push notification:', error);
      throw error;
    }
```

```
    }
  }

  async sendSMS(phoneNumber, message) {
    try {
      const result = await this.twilioClient.messages.create({
        body: message,
        from: process.env.TWILIO_PHONE_NUMBER,
        to: phoneNumber
      });
      return result;
    } catch (error) {
      console.error('Erreur SMS:', error);
      throw error;
    }
  }

  async sendOrderUpdateNotification(orderId, status) {
    const order = await Order.findByPk(orderId, {
      include: [{ model: User, as: 'customer' }]
    });

    const messages = {
      'CONFIRMED': 'Votre commande a été confirmée par le restaurant',
      'PREPARING': 'Votre commande est en cours de préparation',
      'READY': 'Votre commande est prête pour la livraison',
      'PICKED_UP': 'Votre commande a été récupérée par le livreur',
      'DELIVERING': 'Votre commande est en cours de livraison',
      'DELIVERED': 'Votre commande a été livrée avec succès'
    };

    await this.sendPushNotification(
      order.customer_id,
      'Mise à jour de commande',
      messages[status],
      { type: 'ORDER_UPDATE', orderId, status }
    );
  }
}
```

3. Structure de Base de Données PostgreSQL

sql

```sql
-- Script de création de la base de données
CREATE DATABASE eat_fast_db;

-- Extension pour UUID
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Fonction pour générer les timestamps
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ language 'plpgsql';

-- Table des utilisateurs
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    email VARCHAR(255) UNIQUE NOT NULL,
    phone_number VARCHAR(15) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    user_type VARCHAR(20) NOT NULL CHECK (user_type IN ('CLIENT', 'LIVREUR', 'RESTAURANT_MANAGE
    is_verified BOOLEAN DEFAULT FALSE,
    is_active BOOLEAN DEFAULT TRUE,
    profile_image TEXT,
    firebase_uid VARCHAR(255) UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP
);

CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

-- Index pour les recherches fréquentes
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_phone ON users(phone_number);
CREATE INDEX idx_users_type ON users(user_type);
CREATE INDEX idx_users_firebase_uid ON users(firebase_uid);

-- Table des profils clients
CREATE TABLE client_profiles (
    user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE
```

```
    user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
    loyalty_points INTEGER DEFAULT 0,
    preferred_cuisine VARCHAR(100),
    dietary_restrictions JSONB,
    default_payment_method VARCHAR(50),
    notification_preferences JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table des profils livreurs
CREATE TABLE driver_profiles (
    user_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,
    license_number VARCHAR(50) UNIQUE NOT NULL,
    vehicle_type VARCHAR(20) CHECK (vehicle_type IN ('BICYCLE', 'MOTORCYCLE', 'CAR')),
    is_available BOOLEAN DEFAULT FALSE,
    current_latitude DECIMAL(10,8),
    current_longitude DECIMAL(11,8),
    rating DECIMAL(3,2) DEFAULT 0.00,
    completed_deliveries INTEGER DEFAULT 0,
    earnings_today DECIMAL(10,2) DEFAULT 0.00,
    status VARCHAR(20) DEFAULT 'OFFLINE' CHECK (status IN ('ONLINE', 'OFFLINE', 'DELIVERING')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table des restaurants
CREATE TABLE restaurants (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    manager_id UUID NOT NULL REFERENCES users(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    cuisine_type VARCHAR(100),
    phone VARCHAR(15) NOT NULL,
    email VARCHAR(255),
    address TEXT NOT NULL,
    latitude DECIMAL(10,8),
    longitude DECIMAL(11,8),
    opening_hours JSONB,
    delivery_radius INTEGER DEFAULT 5,
    minimum_order DECIMAL(10,2) DEFAULT 0.00,
    delivery_fee DECIMAL(10,2) DEFAULT 0.00,
    commission_rate DECIMAL(5,2) DEFAULT 15.00,
    rating DECIMAL(3,2) DEFAULT 0.00,
    total_reviews INTEGER DEFAULT 0,
    is_active BOOLEAN DEFAULT TRUE,
    is_verified BOOLEAN DEFAULT FALSE,
```

```sql
    featured_image TEXT,
    gallery JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Index géospatial pour les recherches par localisation
CREATE INDEX idx_restaurants_location ON restaurants USING GIST (
    ll_to_earth(latitude, longitude)
);

-- Table des items de menu
CREATE TABLE menu_items (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    restaurant_id UUID NOT NULL REFERENCES restaurants(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL,
    category VARCHAR(100) NOT NULL,
    ingredients JSONB,
    allergens JSONB,
    nutritional_info JSONB,
    preparation_time INTEGER,
    is_available BOOLEAN DEFAULT TRUE,
    is_spicy BOOLEAN DEFAULT FALSE,
    is_vegetarian BOOLEAN DEFAULT FALSE,
    is_traditional BOOLEAN DEFAULT FALSE,
    image TEXT,
    display_order INTEGER,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Table des commandes
CREATE TABLE orders (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    order_number VARCHAR(20) UNIQUE NOT NULL,
    customer_id UUID NOT NULL REFERENCES users(id),
    restaurant_id UUID NOT NULL REFERENCES restaurants(id),
    delivery_address_id UUID REFERENCES addresses(id),
    status VARCHAR(20) DEFAULT 'PENDING' CHECK (status IN ('PENDING', 'CONFIRMED', 'PREPARING',
    subtotal DECIMAL(10,2) NOT NULL,
    delivery_fee DECIMAL(10,2) DEFAULT 0.00,
    service_fee DECIMAL(10,2) DEFAULT 0.00,
    tax_amount DECIMAL(10,2) DEFAULT 0.00,
    discount_amount DECIMAL(10,2) DEFAULT 0.00,
```

```
    total_amount DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(50),
    payment_status VARCHAR(20) DEFAULT 'PENDING' CHECK (payment_status IN ('PENDING', 'PAID', '
    special_instructions TEXT,
    estimated_delivery_time TIMESTAMP,
    confirmed_at TIMESTAMP,
    delivered_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Fonction pour générer le numéro de commande
CREATE OR REPLACE FUNCTION generate_order_number()
RETURNS TRIGGER AS $$
BEGIN
    NEW.order_number = 'EF' || TO_CHAR(CURRENT_DATE, 'YYYYMMDD') ||
                       LPAD(NEXTVAL('order_sequence'), 4, '0');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE SEQUENCE order_sequence START 1;
CREATE TRIGGER generate_order_number_trigger
    BEFORE INSERT ON orders
    FOR EACH ROW EXECUTE FUNCTION generate_order_number();
```

Configuration Docker pour le Déploiement

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  # Base de données PostgreSQL
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: eat_fast_db
      POSTGRES_USER: eat_fast_user
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"
    networks:
      - eat_fast_network

  # Cache Redis
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    networks:
      - eat_fast_network

  # API Principal Node.js
  api:
    build: .
    ports:
      - "3000:3000"
    environment:
      NODE_ENV: production
      DB_HOST: postgres
      DB_PORT: 5432
      DB_NAME: eat_fast_db
      DB_USER: eat_fast_user
      DB_PASSWORD: ${DB_PASSWORD}
      REDIS_URL: redis://redis:6379
      FIREBASE_PROJECT_ID: ${FIREBASE_PROJECT_ID}
      FIREBASE_PRIVATE_KEY: ${FIREBASE_PRIVATE_KEY}
      JWT_SECRET: ${JWT_SECRET}
```

```
      JWT_SECRET: ${JWT_SECRET}
      TWILIO_ACCOUNT_SID: ${TWILIO_ACCOUNT_SID}
      TWILIO_AUTH_TOKEN: ${TWILIO_AUTH_TOKEN}
    depends_on:
      - postgres
      - redis
    networks:
      - eat_fast_network
    volumes:
      - ./uploads:/app/uploads

  # Service de Worker pour les tâches async
  worker:
    build: .
    command: node worker.js
    environment:
      NODE_ENV: production
      DB_HOST: postgres
      REDIS_URL: redis://redis:6379
      FIREBASE_PROJECT_ID: ${FIREBASE_PROJECT_ID}
    depends_on:
      - postgres
      - redis
    networks:
      - eat_fast_network

networks:
  eat_fast_network:
    driver: bridge

volumes:
  postgres_data:
  redis_data:
```

Cette architecture fournit :

1. **Sécurité renforcée** avec Firebase Auth pour 2FA
2. **Notifications en temps réel** via Firebase Cloud Messaging
3. **Base de données relationnelle** PostgreSQL pour l'intégrité des données
4. **Cache Redis** pour les performances
5. **Architecture microservices** pour la scalabilité
6. **API REST** bien structurée
7. **Intégration des paiements** mobile money locaux
8. **Système de géolocalisation** pour les livraisons
9. **Monitoring et logs** complets
10. **Déploiement containerisé** avec Docker