

📁 Structure de Projet AgroDeep – Application Full-Stack PostgreSQL

Voici la structure complète adaptée à votre application **AgroDeep** avec **PostgreSQL** comme base de données principale, optimisée pour le développement local avec **Cursor** et respectant les principes **SOLID/DRY**.

Voici la structure mise à jour du projet AgroDeep avec les nouvelles fonctionnalités intégrées, tout en conservant l'architecture PostgreSQL existante :

...

```
agrodep-platform/
├── docs/
├── src/
│   ├── app/
│   │   ├── (auth)/
│   │   ├── (admin)/
│   │   │   ├── overview/
│   │   │   ├── users/
│   │   │   ├── products/
│   │   │   ├── categories/
│   │   │   ├── orders/
│   │   │   ├── trackings/
│   │   │   ├── settings/
│   │   │   ├── analytics/      # NOUVEAU: Analytics avancés
│   │   │   ├── inventory/    # NOUVEAU: Gestion des stocks
│   │   │   └── notifications/ # NOUVEAU: Centre de notifications
│   │   ├── (market)/
│   │   ├── (customer)/
│   │   ├── (chat)/           # NOUVEAU: Chat en direct
│   │   │   ├── [roomId]/
│   │   │   │   ├── page.tsx
│   │   │   └── page.tsx
│   │   ├── (notifications)/  # NOUVEAU: Notifications utilisateur
│   │   │   └── page.tsx
│   │   ├── (reports)/       # NOUVEAU: Rapports générés
│   │   │   └── page.tsx
│   │   └── landing/
│   ├── components/
│   │   ├── common/
│   │   ├── auth/
│   │   ├── admin/
│   │   ├── AnalyticsDashboard.tsx # NOUVEAU
│   │   ├── InventoryManager.tsx  # NOUVEAU
│   │   └── NotificationCenter.tsx # NOUVEAU
```

```

├── ReportGenerator.tsx    # NOUVEAU
├── customer/
├── market/
├── chat/                  # NOUVEAU: Composants chat
│   ├── ChatWindow.tsx
│   ├── MessageBubble.tsx
│   └── ChatSidebar.tsx
├── notifications/        # NOUVEAU: Système de notifications
│   ├── NotificationBell.tsx
│   ├── NotificationList.tsx
│   └── NotificationItem.tsx
├── reports/              # NOUVEAU: Composants rapports
│   ├── SalesReport.tsx
│   ├── InventoryReport.tsx
│   └── DeliveryReport.tsx
├── configs/
├── guards/
├── hooks/
│   ├── useChat.ts        # NOUVEAU: Gestion chat
│   ├── useNotifications.ts # NOUVEAU: Gestion notifications
│   ├── useInventory.ts    # NOUVEAU: Gestion stocks
│   └── useAnalytics.ts    # NOUVEAU: Analytics
├── stores/
│   ├── useChatStore.ts    # NOUVEAU: Store Zustand chat
│   ├── useNotificationStore.ts # NOUVEAU: Store notifications
│   └── useInventoryStore.ts # NOUVEAU: Store inventaire
├── services/
│   ├── api/
│   │   ├── chat.ts        # NOUVEAU: Endpoints chat
│   │   ├── notifications.ts # NOUVEAU: Endpoints notifications
│   │   ├── inventory.ts    # NOUVEAU: Gestion stocks
│   │   └── analytics.ts    # NOUVEAU: Endpoints analytics
│   ├── database/
│   ├── storage/
│   ├── messaging/
│   │   ├── websocket.ts    # NOUVEAU: Service WebSocket
│   │   └── notification.ts # NOUVEAU: Service notifications
│   └── monitoring/
├── types/
│   ├── database/
│   │   ├── chat.ts        # NOUVEAU: Types chat
│   │   ├── notifications.ts # NOUVEAU: Types notifications
│   │   └── inventory.ts    # NOUVEAU: Types inventaire
│   └── api/
├── utils/
├── seed/
├── migrations/
└── 005_chat_notifications.ts # NOUVEAU

```

```

| | | | 006_inventory_analytics.ts # NOUVEAU
| | | | 007_reports_features.ts   # NOUVEAU
| | | | data/
| | | | | chat.ts                 # NOUVEAU: Seed chat
| | | | | notifications.ts        # NOUVEAU: Seed notifications
| | | | | inventory.ts            # NOUVEAU: Seed inventaire
| | | | logs/
| | | test/
| | | scripts/
| | | docker-compose.yml
| | | Dockerfile
| | | package.json
| | | README.md
| ...

```

Nouvelles Fonctionnalités Ajoutées :

1. Système de Chat en Temps Réel

- **WebSocket** avec Socket.IO pour communication bidirectionnelle
- **Salles de chat** privées et groupées
- **Historique des messages** persistant dans PostgreSQL
- **Notifications** de nouveaux messages

2. Centre de Notifications

- **Notifications push** en temps réel
- **Multi-canaux** : email, in-app, SMS
- **Système de préférences** par utilisateur
- **Historique** et marquage comme lu/non lu

3. Gestion des Stocks (Inventory)

- **Suivi en temps réel** des quantités
- **Alertes automatiques** de stock bas
- **Gestion des fournisseurs** et réapprovisionnement
- **Historique des mouvements** de stock

4. Analytics Avancés

- **Dashboard analytique** avec métriques en temps réel
- **Rapports personnalisables** (ventes, stocks, livraisons)
- **Export PDF/Excel** des rapports
- **Visualisations graphiques** interactives

5. Génération de Rapports

- **Rapports automatisés** programmés
- **Templates personnalisables**
- **Distribution automatique** par email
- **Archivage** des rapports historiques

**Migrations PostgreSQL Ajoutées :

```

```sql
-- Table chat_rooms
CREATE TABLE chat_rooms (
 id SERIAL PRIMARY KEY,
 name VARCHAR(255),
 type VARCHAR(50) DEFAULT 'private',
 participants INTEGER[],
 last_message_at TIMESTAMP WITH TIME ZONE,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Table chat_messages
CREATE TABLE chat_messages (
 id SERIAL PRIMARY KEY,
 room_id INTEGER REFERENCES chat_rooms(id) ON DELETE CASCADE,
 user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
 content TEXT NOT NULL,
 message_type VARCHAR(50) DEFAULT 'text',
 is_read BOOLEAN DEFAULT FALSE,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Table notifications
CREATE TABLE notifications (
 id SERIAL PRIMARY KEY,
 user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
 title VARCHAR(255) NOT NULL,
 message TEXT NOT NULL,
 type VARCHAR(50) NOT NULL,
 is_read BOOLEAN DEFAULT FALSE,
 metadata JSONB,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Table inventory
CREATE TABLE inventory (
 id SERIAL PRIMARY KEY,
 product_id INTEGER UNIQUE REFERENCES products(id) ON DELETE CASCADE,
 quantity INTEGER NOT NULL DEFAULT 0,
 reserved_quantity INTEGER DEFAULT 0,
 reorder_point INTEGER DEFAULT 10,
 last_updated TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 supplier_id INTEGER,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Table inventory_logs

```

```

CREATE TABLE inventory_logs (
 id SERIAL PRIMARY KEY,
 inventory_id INTEGER REFERENCES inventory(id) ON DELETE CASCADE,
 user_id INTEGER REFERENCES users(id),
 change_type VARCHAR(50),
 quantity_change INTEGER,
 previous_quantity INTEGER,
 new_quantity INTEGER,
 reason TEXT,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
...

```

## \*\*Services Ajoutés :\*\*

### \*\*Service WebSocket :\*\*

```

``typescript
// src/services/messaging/websocket.ts
import { Server } from 'socket.io';
import { createAdapter } from '@socket.io/redis-adapter';
import { redis } from './redis';

export class WebSocketService {
 private io: Server;

 constructor(server: any) {
 this.io = new Server(server, {
 cors: { origin: process.env.CLIENT_URL },
 adapter: createAdapter(redis, redis.duplicate())
 });

 this.setupEvents();
 }

 private setupEvents() {
 this.io.on('connection', (socket) => {
 // Gestion des salles de chat
 socket.on('join-room', (roomId) => {
 socket.join(roomId);
 });

 // Envoi de messages
 socket.on('send-message', async (data) => {
 // Sauvegarde en base de données
 // Diffusion aux participants
 socket.to(data.roomId).emit('new-message', data);
 });
 });
 }
}

```

```
}
}
...
```

### \*\*Service de Notifications :\*\*

```
```typescript  
// src/services/messaging/notification.ts  
export class NotificationService {  
  static async send(userId: number, notification: {  
    title: string;  
    message: string;  
    type: string;  
    metadata?: any;  
  }) {  
    // Enregistrement en base  
    // Envoi WebSocket  
    // Envoi email si configuré  
  }  
  
  static async sendBulk(userIds: number[], notification: any) {  
    // Notifications groupées  
  }  
}  
...
```

Package.json - Dépendances Ajoutées :

```
```json  
{
 "dependencies": {
 "socket.io": "^4.7.4",
 "socket.io-client": "^4.7.4",
 "@socket.io/redis-adapter": "^8.2.1",
 "pdfkit": "^0.14.0",
 "exceljs": "^4.4.0",
 "chart.js": "^4.4.1",
 "react-chartjs-2": "^5.2.0",
 "recharts": "^2.10.3",
 "date-fns": "^3.3.1",
 "nodemailer": "^6.9.9",
 "twilio": "^4.19.4"
 }
}
...
```

## \*\*Configuration Docker-Compose Mise à Jour :\*\*

```
```yaml
```

```

services:
  # Services existants...
  websocket:
    image: node:18-alpine
    container_name: agrodeep-websocket
    restart: unless-stopped
    working_dir: /app
    ports:
      - "3001:3001"
    volumes:
      - ./src/services/messaging/websocket.ts:/app/websocket.ts
    depends_on:
      - redis
    command: ["node", "websocket.ts"]
  ...

```

Scripts Ajoutés .

```

```json
{
 "scripts": {
 "dev:ws": "nodemon src/services/messaging/websocket.ts",
 "generate:report": "tsx src/scripts/generate-report.ts",
 "backup:inventory": "tsx src/scripts/backup-inventory.ts",
 "notify:stock": "tsx src/scripts/notify-low-stock.ts"
 }
}

```

## ## 📝 DESCRIPTION DÉTAILLÉE DES PARTIES CLÉS

#### 1. \*\*Configuration de la Base de Données PostgreSQL\*\*

```

📄 `src/configs/database.ts`
```typescript
import { Pool, PoolConfig } from 'pg';
import { config } from './env';

const poolConfig: PoolConfig = {
  host: config.db.host,
  port: config.db.port,
  database: config.db.name,
  user: config.db.user,
  password: config.db.password,
  max: 20, // Maximum de connexions dans le pool
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
  ssl: config.env === 'production' ? { rejectUnauthorized: false } : false,

```

```

};

export const db = new Pool(poolConfig);

// Gestion des erreurs de connexion
db.on('error', (err) => {
  console.error('❌ PostgreSQL Pool Error:', err);
  process.exit(-1);
});

// Logging des connexions (pour debug)
db.on('connect', () => {
  if (config.env === 'development') {
    console.log('📦 PostgreSQL Connected');
  }
});

export default db;
``

```

📄 `docker-compose.yml`

``yaml

version: '3.8'

services:

postgres:

image: postgres:16-alpine

container_name: agrodeep-db

restart: unless-stopped

environment:

POSTGRES_USER: agrodeep_admin

POSTGRES_PASSWORD: \${DB_PASSWORD}

POSTGRES_DB: agrodeep_db

ports:

- "5432:5432"

volumes:

- postgres_data:/var/lib/postgresql/data

- ./scripts/init.sql:/docker-entrypoint-initdb.d/init.sql

healthcheck:

test: ["CMD-SHELL", "pg_isready -U agrodeep_admin"]

interval: 10s

timeout: 5s

retries: 5

redis:

image: redis:7-alpine

container_name: agrodeep-redis

restart: unless-stopped

```
ports:
  - "6379:6379"
volumes:
  - redis_data:/data
command: redis-server --appendonly yes
```

```
minio:
  image: minio/minio
  container_name: agrodeep-storage
  restart: unless-stopped
  environment:
    MINIO_ROOT_USER: ${MINIO_ROOT_USER}
    MINIO_ROOT_PASSWORD: ${MINIO_ROOT_PASSWORD}
  ports:
    - "9000:9000"
    - "9001:9001"
  volumes:
    - minio_data:/data
  command: server /data --console-address ":9001"
```

```
volumes:
  postgres_data:
  redis_data:
  minio_data:
  ...
```

2. **Migrations PostgreSQL**

```
#####  `src/seed/migrations/001_init_schema.ts`
```

```
```typescript
```

```
import { db } from '@services/database';
```

```
import { logger } from '@services/monitoring/logger';
```

```
export async function up(): Promise<void> {
 const client = await db.connect();
 try {
 await client.query(`
 -- Table: settings (doit être créée en premier)
 CREATE TABLE settings (
 id SERIAL PRIMARY KEY,
 app_name VARCHAR(255) NOT NULL DEFAULT 'AgroDeep',
 app_primary_color VARCHAR(7) NOT NULL DEFAULT '#2563eb',
 app_logo_url VARCHAR(500),
 timezone VARCHAR(50) NOT NULL DEFAULT 'UTC',
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
 `);
 } catch (error) {
 logger.error('Error during migration: ', error);
 }
}
```

```
);
```

```
-- Table: roles
```

```
CREATE TABLE roles (
 id SERIAL PRIMARY KEY,
 name VARCHAR(100) UNIQUE NOT NULL,
 permissions JSONB NOT NULL DEFAULT '[]',
 fields JSONB,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_by INTEGER
);
```

```
-- Table: users
```

```
CREATE TABLE users (
 id SERIAL PRIMARY KEY,
 email VARCHAR(255) UNIQUE NOT NULL,
 password_hash VARCHAR(255) NOT NULL,
 image_profile_url VARCHAR(500),
 is_email_verified BOOLEAN DEFAULT FALSE,
 role_id INTEGER REFERENCES roles(id) ON DELETE SET NULL,
 billing_info JSONB,
 fields JSONB,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_by INTEGER REFERENCES users(id) ON DELETE SET NULL
);
```

```
-- Table: categories
```

```
CREATE TABLE categories (
 id SERIAL PRIMARY KEY,
 name VARCHAR(255) NOT NULL,
 description TEXT,
 parent_id INTEGER REFERENCES categories(id) ON DELETE CASCADE,
 fields JSONB,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_by INTEGER REFERENCES users(id) ON DELETE SET NULL
);
```

```
-- Table: tags
```

```
CREATE TABLE tags (
 id SERIAL PRIMARY KEY,
 name VARCHAR(100) UNIQUE NOT NULL,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_by INTEGER REFERENCES users(id) ON DELETE SET NULL
);
```

-- Table: products

```
CREATE TABLE products (
 id SERIAL PRIMARY KEY,
 name VARCHAR(255) NOT NULL,
 description TEXT,
 price DECIMAL(10, 2) NOT NULL CHECK (price >= 0),
 category_id INTEGER REFERENCES categories(id) ON DELETE SET NULL,
 tags INTEGER[] DEFAULT '{}',
 fields JSONB,
 image_urls VARCHAR(500[]),
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_by INTEGER REFERENCES users(id) ON DELETE SET NULL
);
```

-- Table: payments

```
CREATE TABLE payments (
 id SERIAL PRIMARY KEY,
 user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
 stripe_payment_method_id VARCHAR(255) UNIQUE NOT NULL,
 card_last4 VARCHAR(4),
 card_brand VARCHAR(50),
 is_preferred_card BOOLEAN DEFAULT FALSE,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

-- Table: orders

```
CREATE TABLE orders (
 id SERIAL PRIMARY KEY,
 user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
 products JSONB NOT NULL, -- [{product_id, quantity, price}]
 amount DECIMAL(10, 2) NOT NULL,
 status VARCHAR(50) DEFAULT 'pending',
 stripe_payment_intent_id VARCHAR(255),
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

-- Table: trackings

```
CREATE TABLE trackings (
 id SERIAL PRIMARY KEY,
 order_id INTEGER UNIQUE REFERENCES orders(id) ON DELETE CASCADE,
 status VARCHAR(50) NOT NULL,
 is_cancelled BOOLEAN DEFAULT FALSE,
 estimated_delivery TIMESTAMP WITH TIME ZONE,
 fields JSONB,
```

```
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_by INTEGER REFERENCES users(id) ON DELETE SET NULL
);
```

-- Table: comments

```
CREATE TABLE comments (
 id SERIAL PRIMARY KEY,
 product_id INTEGER REFERENCES products(id) ON DELETE CASCADE,
 user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
 message TEXT NOT NULL,
 rating INTEGER CHECK (rating >= 1 AND rating <= 5),
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

-- Table: logs (centralisée pour audit)

```
CREATE TABLE logs (
 id SERIAL PRIMARY KEY,
 level VARCHAR(50) NOT NULL,
 service VARCHAR(100) NOT NULL,
 tenant_id INTEGER,
 user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
 message TEXT NOT NULL,
 metadata JSONB,
 timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

-- Index pour performance

```
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_products_tags ON products USING GIN(tags);
CREATE INDEX idx_orders_user ON orders(user_id);
CREATE INDEX idx_trackings_order ON trackings(order_id);
CREATE INDEX idx_comments_product ON comments(product_id);
CREATE INDEX idx_logs_timestamp ON logs(timestamp DESC);
CREATE INDEX idx_logs_tenant ON logs(tenant_id);
`);
```

```
 logger.info('✅ Schema migration completed');
 } finally {
 client.release();
 }
}
```

```
export async function down(): Promise<void> {
 const client = await db.connect();
 try {
```

```

await client.query(`
 DROP TABLE IF EXISTS logs;
 DROP TABLE IF EXISTS comments;
 DROP TABLE IF EXISTS trackings;
 DROP TABLE IF EXISTS orders;
 DROP TABLE IF EXISTS payments;
 DROP TABLE IF EXISTS products;
 DROP TABLE IF EXISTS tags;
 DROP TABLE IF EXISTS categories;
 DROP TABLE IF EXISTS users;
 DROP TABLE IF EXISTS roles;
 DROP TABLE IF EXISTS settings;
`);

logger.info('✅ Schema rollback completed');
} finally {
 client.release();
}
}
...

📄 `src/seed/migrations/002_create_indexes.ts`
```typescript
// Indexes supplémentaires pour les requêtes fréquentes
// Cache indexes, full-text search, etc.
...

---

### 3. **Services PostgreSQL**

#### 📄 `src/services/api/client.ts`
```typescript
import axios, { AxiosInstance, AxiosError } from 'axios';
import { config } from '@configs/env';
import { useAuthStore } from '@stores/useAuthStore';
import { logger } from '@services/monitoring/logger';

const apiClient: AxiosInstance = axios.create({
 baseURL: config.api.baseURL,
 timeout: 10000,
 headers: {
 'Content-Type': 'application/json',
 },
});

// Request interceptor: ajoute auth token
apiClient.interceptors.request.use(

```

```

(config) => {
 const token = useAuthStore.getState().token;
 if (token) {
 config.headers.Authorization = `Bearer ${token}`;
 }

 // Log requête
 logger.info('API Request', {
 method: config.method,
 url: config.url,
 timestamp: new Date().toISOString(),
 });

 return config;
},
(error) => {
 logger.error('API Request Error', error);
 return Promise.reject(error);
}
);

// Response interceptor: gestion erreurs
apiClient.interceptors.response.use(
 (response) => {
 logger.info('API Response', {
 status: response.status,
 url: response.config.url,
 duration: response.headers['x-response-time'],
 });
 return response;
 },
 (error: AxiosError) => {
 logger.error('API Response Error', {
 message: error.message,
 status: error.response?.status,
 data: error.response?.data,
 });

 // Gestion 401 Unauthorized
 if (error.response?.status === 401) {
 useAuthStore.getState().logout();
 window.location.href = '/login';
 }

 return Promise.reject(error);
 }
);

```

```
export default apiClient;
```

```
...
```

```
 `src/services/database/logs.ts`
```

```
``typescript
```

```
import { db } from '@services/database';
```

```
import { LogEntry } from '@types/database/logs';
```

```
export class LogService {
```

```
 static async create(log: Omit<LogEntry, 'id' | 'timestamp'>): Promise<void> {
```

```
 await db.query(
```

```
 `INSERT INTO logs (level, service, tenant_id, user_id, message, metadata)
```

```
 VALUES ($1, $2, $3, $4, $5, $6)`,
```

```
 [log.level, log.service, log.tenant_id, log.user_id, log.message, log.metadata]
```

```
);
```

```
 }
```

```
 static async getByTenant(tenantId: number, limit: number = 100): Promise<LogEntry[]> {
```

```
 const result = await db.query(
```

```
 `SELECT * FROM logs WHERE tenant_id = $1 ORDER BY timestamp DESC LIMIT $2`,
```

```
 [tenantId, limit]
```

```
);
```

```
 return result.rows;
```

```
 }
```

```
 static async cleanup(days: number = 30): Promise<void> {
```

```
 await db.query(
```

```
 `DELETE FROM logs WHERE timestamp < NOW() - INTERVAL '${days} days'`
```

```
);
```

```
 }
```


```
}
```

```
...
```

```

```

```
4. **Composants Clés avec Thème Bleu**
```

```
 `src/components/common/ThemeSwitcher.tsx`
```

```
``typescript
```

```
'use client';
```

```
import { useThemeStore } from '@stores/useThemeStore';
```

```
import { Sun, Moon } from 'lucide-react';
```

```
import { Button } from '@components/ui/button';
```

```
import { useEffect, useState } from 'react';
```

```
export function ThemeSwitcher() {
```

```
 const [mounted, setMounted] = useState(false);
```

```

const { theme, toggleTheme } = useThemeStore();

useEffect(() => {
 setMounted(true);
}, []);

if (!mounted) return null;

return (
 <Button
 variant="outline"
 size="icon"
 onClick={toggleTheme}
 className="fixed top-4 right-4 z-50"
 aria-label="Toggle theme"
 >
 {theme === 'dark' ? (
 <Sun className="h-[1.2rem] w-[1.2rem] text-blue-500" />
) : (
 <Moon className="h-[1.2rem] w-[1.2rem] text-blue-600" />
)}
 </Button>
);
}
...

```

```

📄 `src/stores/useThemeStore.ts`
``typescript
import { create } from 'zustand';
import { persist } from 'zustand/middleware';
import { logger } from '@services/monitoring/logger';

```

```

interface ThemeState {
 theme: 'light' | 'dark';
 primaryColor: string;
 setTheme: (theme: 'light' | 'dark') => void;
 setPrimaryColor: (color: string) => void;
 toggleTheme: () => void;
}

```

```

export const useThemeStore = create<ThemeState>()(
 persist(
 (set, get) => ({
 theme: 'light',
 primaryColor: '#2563eb', // Bleu par défaut

 setTheme: (theme) => {
 set({ theme });
 }
 })
)
);

```

```

 document.documentElement.classList.toggle('dark', theme === 'dark');
 logger.info('Theme changed', { theme });
 },


 setPrimaryColor: (color) => {
 set({ primaryColor: color });
 document.documentElement.style.setProperty('--primary', color);
 logger.info('Primary color changed', { color });
 },

 toggleTheme: () => {
 const newTheme = get().theme === 'light' ? 'dark' : 'light';
 get().setTheme(newTheme);
 },
}),
{
 name: 'agrodeep-theme',
 partialize: (state) => ({
 theme: state.theme,
 primaryColor: state.primaryColor,
 }),
}
)
);
...

```

---

#### ### 5. \*\*Guards pour Protection des Routes\*\*

####  `src/guards/AdminGuard.tsx`

```typescript

'use client';

```

import { useRouter } from 'next/navigation';
import { useEffect, useState } from 'react';
import { useAuthStore } from '@stores/useAuthStore';
import { logger } from '@services/monitoring/logger';
import { Spinner } from '@components/ui/spinner';

```

```

interface AdminGuardProps {
  children: React.ReactNode;
}

```

```

export function AdminGuard({ children }: AdminGuardProps) {
  const router = useRouter();
  const { user, isLoading } = useAuthStore();
  const [authorized, setAuthorized] = useState(false);

```

```

useEffect(() => {
  if (!isLoading) {
    if (!user) {
      logger.warn('Unauthorized access attempt to admin', {
        path: window.location.pathname,
      });
      router.push('/login');
    } else if (user.role !== 'admin') {
      logger.warn('Non-admin user attempted to access admin', {
        userId: user.id,
        role: user.role,
      });
      router.push('/home/overview');
    } else {
      setAuthorized(true);
    }
  }
}, [user, isLoading, router]);

if (isLoading || !authorized) {
  return (
    <div className="flex items-center justify-center min-h-screen">
      <Spinner size="lg" />
    </div>
  );
}

return <>{children}</>;
}
...

```

6. **Scripts de Seeding pour PostgreSQL**

 `src/seed/data/users.ts`

``typescript

```

import bcrypt from 'bcryptjs';
import { config } from '@configs/env';
import { logger } from '@services/monitoring/logger';
import { db } from '@services/database';

```

```

export async function seedUsers(): Promise<void> {
  const hashedPassword = await bcrypt.hash('Admin123', 12);

```

```

  const users = [
    {

```

```

    email: 'admin@agrodeep.com',
    password_hash: hashedPassword,
    role_id: 1, // Admin role
    is_email_verified: true,
    billing_info: {
      address: '123 Admin Street, Paris',
      postalCode: '75001',
      country: 'France',
    },
  },
  {
    email: 'user1@agrodeep.com',
    password_hash: await bcrypt.hash('User123', 12),
    role_id: 2, // Customer role
    is_email_verified: true,
    billing_info: {
      address: '456 User Avenue, Lyon',
      postalCode: '69001',
      country: 'France',
    },
  },
];

for (const user of users) {
  try {
    await db.query(
      `INSERT INTO users (email, password_hash, role_id, is_email_verified, billing_info)
      VALUES ($1, $2, $3, $4, $5)
      ON CONFLICT (email) DO NOTHING`,
      [user.email, user.password_hash, user.role_id, user.is_email_verified,
JSON.stringify(user.billing_info)]
    );
    logger.info('User seeded', { email: user.email });
  } catch (error) {
    logger.error('Failed to seed user', { error, email: user.email });
  }
}
}
...

```

```

#### 📄 `src/seed/runner.ts`
``typescript
import { logger } from '@services/monitoring/logger';
import { seedSettings } from './data/settings';
import { seedRoles } from './data/roles';
import { seedUsers } from './data/users';
import { seedCategories } from './data/categories';
import { seedTags } from './data/tags';

```

```

import { seedProducts } from './data/products';
import { seedOrders } from './data/orders';
import { seedComments } from './data/comments';

export async function runSeeds(): Promise<void> {
  logger.info('🌱 Starting database seeds...');

  try {
    await seedSettings();
    await seedRoles();
    await seedUsers();
    await seedCategories();
    await seedTags();
    await seedProducts();
    await seedOrders();
    await seedComments();

    logger.info('✅ All seeds completed successfully');
  } catch (error) {
    logger.error('❌ Seed process failed', { error });
    process.exit(1);
  }
}

```

7. **Configuration du Thème (Bleu Dynamique)**

```

##### 📄 `tailwind.config.ts`
``typescript
import type { Config } from 'tailwindcss';
import { fontFamily } from 'tailwindcss/defaultTheme';

const config: Config = {
  darkMode: ['class'],
  content: [
    './src/pages/**/*.{js,ts,jsx,tsx,mdx}',
    './src/components/**/*.{js,ts,jsx,tsx,mdx}',
    './src/app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    container: {
      center: true,
      padding: '2rem',
      screens: {
        '2xl': '1400px',
      },
    },
  },
}

```

```

},
extend: {
  colors: {
    border: 'hsl(var(--border))',
    input: 'hsl(var(--input))',
    ring: 'hsl(var(--ring))',
    background: 'hsl(var(--background))',
    foreground: 'hsl(var(--foreground))',
    primary: {
      DEFAULT: 'var(--primary)', // Bleu dynamique depuis settings
      foreground: 'hsl(var(--primary-foreground))',
    },
    secondary: {
      DEFAULT: 'hsl(var(--secondary))',
      foreground: 'hsl(var(--secondary-foreground))',
    },
    destructive: {
      DEFAULT: 'hsl(var(--destructive))',
      foreground: 'hsl(var(--destructive-foreground))',
    },
    muted: {
      DEFAULT: 'hsl(var(--muted))',
      foreground: 'hsl(var(--muted-foreground))',
    },
    accent: {
      DEFAULT: 'hsl(var(--accent))',
      foreground: 'hsl(var(--accent-foreground))',
    },
    popover: {
      DEFAULT: 'hsl(var(--popover))',
      foreground: 'hsl(var(--popover-foreground))',
    },
    card: {
      DEFAULT: 'hsl(var(--card))',
      foreground: 'hsl(var(--card-foreground))',
    },
  },
  borderRadius: {
    lg: 'var(--radius)',
    md: 'calc(var(--radius) - 2px)',
    sm: 'calc(var(--radius) - 4px)',
  },
  fontFamily: {
    sans: ['var(--font-sans)', ...fontFamily.sans],
  },
},
plugins: [require('tailwindcss-animate')],

```

```
};
```

```
export default config;
```

```
...
```

```
####  `src/app/globals.css`
```

```
```css
```

```
@tailwind base;
```

```
@tailwind components;
```

```
@tailwind utilities;
```

```
:root {
```

```
 --background: 0 0% 100%;
```

```
 --foreground: 222.2 84% 4.9%;
```

```
 --card: 0 0% 100%;
```

```
 --card-foreground: 222.2 84% 4.9%;
```

```
 --popover: 0 0% 100%;
```

```
 --popover-foreground: 222.2 84% 4.9%;
```

```
 --primary: 217.2 91.2% 59.8%; /* Bleu #2563eb */
```

```
 --primary-foreground: 210 40% 98%;
```

```
 --secondary: 210 40% 96.1%;
```

```
 --secondary-foreground: 222.2 84% 4.9%;
```

```
 --muted: 210 40% 96.1%;
```

```
 --muted-foreground: 215.4 16.3% 46.9%;
```

```
 --accent: 210 40% 96.1%;
```

```
 --accent-foreground: 222.2 84% 4.9%;
```

```
 --destructive: 0 84.2% 60.2%;
```

```
 --destructive-foreground: 210 40% 98%;
```

```
 --border: 214.3 31.8% 91.4%;
```

```
 --input: 214.3 31.8% 91.4%;
```

```
 --ring: 217.2 91.2% 59.8%;
```

```
 --radius: 0.5rem;
```

```
}
```

```
.dark {
```

```
 --background: 222.2 84% 4.9%;
```

```
 --foreground: 210 40% 98%;
```

```
 --card: 222.2 84% 4.9%;
```

```
 --card-foreground: 210 40% 98%;
```

```
 --popover: 222.2 84% 4.9%;
```

```
 --popover-foreground: 210 40% 98%;
```

```
 --primary: 217.2 91.2% 59.8%;
```

```
 --primary-foreground: 222.2 84% 4.9%;
```

```
 --secondary: 217.2 32.6% 17.5%;
```

```
 --secondary-foreground: 210 40% 98%;
```

```
 --muted: 217.2 32.6% 17.5%;
```

```
 --muted-foreground: 215 20.2% 65.1%;
```

```
 --accent: 217.2 32.6% 17.5%;
```

```

--accent-foreground: 210 40% 98%;
--destructive: 0 62.8% 30.6%;
--destructive-foreground: 210 40% 98%;
--border: 217.2 32.6% 17.5%;
--input: 217.2 32.6% 17.5%;
--ring: 217.2 91.2% 59.8%;
}

```

```
/* Custom scrollbar */
```

```

::-webkit-scrollbar {
 width: 8px;
 height: 8px;
}

```

```

::-webkit-scrollbar-track {
 background: hsl(var(--muted));
}

```

```

::-webkit-scrollbar-thumb {
 background: hsl(var(--primary));
 border-radius: var(--radius);
}

```

```
/* Mobile-first responsive */
```

```

@media (max-width: 640px) {
 .container {
 padding-left: 1rem;
 padding-right: 1rem;
 }
}
...

```

```

```

### 8. \*\*Package.json – Scripts pour Cursor\*\*

```
📄 `package.json`
```

```

```json
{
  "name": "agrodeep-platform",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "concurrently \"npm run dev:db\" \"next dev\"",
    "dev:db": "docker-compose up -d",
    "build": "next build",
    "start": "next start",
    "lint": "eslint . --ext .ts,.tsx",
  }
}

```

```

"lint:fix": "eslint . --ext .ts,.tsx --fix",
"format": "prettier --write \"src/**/*.ts,tsx\"",
"test": "vitest run",
"test:watch": "vitest",
"test:e2e": "playwright test",
"seed": "tsx src/seed/runner.ts",
"migrate": "tsx src/seed/migrations/runner.ts",
"logs": "tail -f src/logs/app.log",
"backup": "./scripts/backup.sh",
"cursor:setup": "npm install && npm run dev:db && npm run migrate && npm run seed"
},
"dependencies": {
  "@hookform/resolvers": "^3.3.4",
  "@radix-ui/react-dialog": "^1.0.5",
  "@radix-ui/react-dropdown-menu": "^2.0.6",
  "@radix-ui/react-label": "^2.0.2",
  "@radix-ui/react-select": "^2.0.0",
  "@radix-ui/react-slot": "^1.0.2",
  "@radix-ui/react-switch": "^1.0.3",
  "@radix-ui/react-tabs": "^1.0.4",
  "@radix-ui/react-toast": "^1.1.5",
  "@stripe/react-stripe-js": "^2.5.0",
  "@stripe/stripe-js": "^3.0.5",
  "axios": "^1.6.7",
  "bcryptjs": "^2.4.3",
  "class-variance-authority": "^0.7.0",
  "clsx": "^2.1.0",
  "date-fns": "^3.3.1",
  "lucide-react": "^0.323.0",
  "next": "14.1.0",
  "pg": "^8.11.3",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-hook-form": "^7.50.1",
  "redis": "^4.6.12",
  "socket.io-client": "^4.7.4",
  "tailwind-merge": "^2.2.1",
  "tailwindcss-animate": "^1.0.7",
  "winston": "^3.11.0",
  "zod": "^3.22.4",
  "zustand": "^4.5.0"
},
"devDependencies": {
  "@playwright/test": "^1.41.2",
  "@types/bcryptjs": "^2.4.6",
  "@types/node": "^20.11.16",
  "@types/pg": "^8.11.0",
  "@types/react": "^18.2.55",

```

```

"@types/react-dom": "^18.2.19",
"@typescript-eslint/eslint-plugin": "^6.21.0",
"@typescript-eslint/parser": "^6.21.0",
"@vitejs/plugin-react": "^4.2.1",
"@vitest/ui": "^1.2.2",
"autoprefixer": "^10.4.17",
"concurrently": "^8.2.2",
"eslint": "^8.56.0",
"eslint-config-next": "14.1.0",
"eslint-config-prettier": "^9.1.0",
"eslint-plugin-react-hooks": "^4.6.0",
"eslint-plugin-unicorn": "^50.0.1",
"postcss": "^8.4.35",
"prettier": "^3.2.5",
"prettier-plugin-tailwindcss": "^0.5.11",
"tailwindcss": "^3.4.1",
"tsx": "^4.7.1",
"typescript": "^5.3.3",
"unplugin-auto-import": "^0.17.5",
"vitest": "^1.2.2"
},
"engines": {
  "node": ">=18.17.0",
  "npm": ">=9.0.0"
}
}
...

```

9. **Guides pour Cursor – Comment démarrer**

📄 `README.md`

```markdown

# 🌱 AgroDeep Platform

Plateforme SaaS modulaire pour l'écosystème agro-logistique – 100% Open-Source.

## 🚀 Démarrage Rapide avec Cursor

#### Prérequis

- Node.js 18.17+
- Docker & Docker Compose
- Cursor IDE

#### Installation

1. \*\*Cloner le dépôt\*\*

```
```bash
git clone https://github.com/agrodeep/platform.git
cd agrodeep-platform
```
```

## 2. **\*\*Configurer l'environnement\*\***

```
```bash
cp .env.local.example .env.local
# Modifier les variables dans Cursor
```
```

## 3. **\*\*Installer les dépendances\*\***

```
```bash
npm run cursor:setup
```
```

## 4. **\*\*Démarrer l'environnement de développement\*\***

```
```bash
npm run dev
```
```

## 5. **\*\*Ouvrir dans Cursor\*\***

- `Cmd+Shift+P` → `Developer: Reload Window`
- Les recommandations d'extensions s'installeront automatiquement

## ### Scripts Principaux

| Commande          | Description                    |
|-------------------|--------------------------------|
| `npm run dev`     | Démarre Next.js + DB           |
| `npm run build`   | Build production               |
| `npm run seed`    | Seed la base avec données test |
| `npm run migrate` | Exécute les migrations         |
| `npm run logs`    | Voir les logs en temps réel    |
| `npm test`        | Lancer les tests unitaires     |

## ### 🖥️ Accès aux Services Locaux

- **\*\*App\*\*** : http://localhost:3000
- **\*\*PostgreSQL\*\*** : localhost:5432
- **\*\*Redis\*\*** : localhost:6379
- **\*\*MinIO Console\*\*** : http://localhost:9001

## ### 👤 Comptes de Test

| Email              | Mot de passe | Rôle  |
|--------------------|--------------|-------|
| admin@agrodeep.com | Admin123     | Admin |

| user1@agrodeep.com | User123 | Customer |

### ### 🎯 Bonnes Pratiques pour Cursor

1. **Auto-imports** : Configurés dans `./.cursor/settings.json`
2. **Snippets** : Utiliser `agro-` prefix pour snippets custom
3. **Types** : Toujours importer depuis `@/types`
4. **Composants** : Respecter pattern `Component.tsx` + `Component.test.tsx`
5. **Logs** : Utiliser `logger.info/error` pour tout événement métier

### ### 📁 Structure à respecter

```
...
src/
├── app/ # Routes Next.js 14 (App Router)
├── components/ # Composants réutilisables
├── services/ # Logique métier + API
├── types/ # Types TypeScript
├── stores/ # Zustand stores
├── hooks/ # Hooks custom
└── seed/ # Migrations & seeds
...
```

### ### 🛠️ Configuration Cursor

Créer `./.cursor/settings.json` :

```
```json
{
  "typescript.preferences.importModuleSpecifier": "non-relative",
  "typescript.suggest.autoImports": true,
  "editor.codeActionsOnSave": {
    "source.organizeImports": true,
    "source.fixAll.eslint": true
  }
}
```
```

### ### 📊 Monitoring Local

- **Grafana** : `http://localhost:3001` (metrics)
- **Loki** : Logs structurés dans `./src/logs/app.log`

### ### 🤝 Contribuer

1. Créer une branche `feature/nom-feature`
2. Respecter les commits conventions (`feat:`, `fix:`, `docs:`)
3. Tests > 80% coverage obligatoire
4. PR vers `develop` → Review → Merge

```
{
 "typescript.preferences.importModuleSpecifier": "non-relative",
 "typescript.suggest.autoImports": true,
 "typescript.updateImportsOnFileMove.enabled": "always",
 "editor.formatOnSave": true,
 "editor.codeActionsOnSave": {
 "source.organizeImports": true,
 "source.fixAll.eslint": true,
 "source.addMissingImports": true
 },
 "emmet.includeLanguages": {
 "typescriptreact": "typescript",
 "javascriptreact": "javascript"
 },
 "files.associations": {
 "*.css": "tailwindcss"
 },
 "tailwindCSS.experimental.classRegex": [
 ["cva\\(((\\^)*))\\)", "[\\\"'"]([\\^\\\"'"])*.*?[\\\"'"]"]
]
}
```

...

####  `.cursor/snippets/typescript.json`

```
```json
{
  "AgroDeep Component": {
    "prefix": "agro-component",
    "body": [
      "import { FC } from 'react';",
      "",
      "interface ${1:ComponentName}Props {",
      "  className?: string;",
      "}",
      "",
      "export const ${1:ComponentName}: FC<${1:ComponentName}Props> = ({ className",
    ] => {",
      "  return (",
      "    <div className={cn('${2:baseClasses}', className)}>",
      "      ${3:content}",
      "    </div>",
      "  );",
      "};",
    ],
    "description": "Crée un composant React TypeScript"
  }
}
...
---
```

11. **Logs & Monitoring Centralisés**

 `src/services/monitoring/logger.ts`

```
```typescript
import winston from 'winston';
import path from 'path';
import { LogService } from '@services/database/logs';

// Format structuré pour Loki/PostgreSQL
const logFormat = winston.format.combine(
 winston.format.timestamp(),
 winston.format.errors({ stack: true }),
 winston.format.json({
 remplacer: (key, value) => {
 // Masquer les données sensibles
 if (key === 'password' || key === 'cardNumber') {
 return '[REDACTED]';
 }
 }
 })
);
```

```

 return value;
 },
 })
);

// Logger principal
export const logger = winston.createLogger({
 level: process.env.LOG_LEVEL || 'info',
 format: logFormat,
 defaultMeta: {
 service: 'agrodeep-platform',
 version: '1.0.0',
 },
 transports: [
 // Console (dev)
 new winston.transports.Console({
 format: winston.format.combine(
 winston.format.colorize(),
 winston.format.simple()
),
 }),
 // Fichier local
 new winston.transports.File({
 filename: path.join(process.cwd(), 'src/logs/app.log'),
 maxsize: 10485760, // 10MB
 maxFiles: 5,
 tailable: true,
 }),
],
});

// Transport custom pour PostgreSQL
class PostgreSQLTransport extends winston.Transport {
 async log(info: any, callback: Function) {
 try {
 await LogService.create({
 level: info.level,
 service: info.service,
 tenant_id: info.tenant_id,
 user_id: info.user_id,
 message: info.message,
 metadata: info.metadata || {},
 });
 } catch (error) {
 console.error('Failed to write log to PostgreSQL:', error);
 }
 callback();
 }
}

```


```
}
```

```
// Ajouter transport PostgreSQL en production
if (process.env.NODE_ENV === 'production') {
 logger.add(new PostgreSQLTransport());
}
...
```

```

```

### 12. \*\*Schémas de Validation Zod (SOLID/DRY)\*\*

```
 `src/types/schema/users.schema.ts`
```typescript
import { z } from 'zod';
```

```
export const userSchema = z.object({
  id: z.number().optional(),
  email: z.string().email('Email invalide'),
  password: z.string().min(8, '8 caractères minimum'),
  roleId: z.number().optional(),
  billingInfo: z.object({
    address: z.string().min(5, 'Adresse requise'),
    postalCode: z.string().min(5, 'Code postal requis'),
    country: z.string().min(2, 'Pays requis'),
  }).optional(),
});
```

```
export const loginSchema = z.object({
  email: z.string().email(),
  password: z.string().min(1, 'Mot de passe requis'),
});
```

```
export const commentSchema = z.object({
  productId: z.number(),
  message: z.string().min(3, 'Commentaire trop court').max(500, 'Commentaire trop long'),
  rating: z.number().min(1).max(5),
});
...
```

```
---
```

 **Instructions de Développement pour Cursor**

Workflow Recommandé

1. **Créer un composant** :
 - Utiliser snippet `agro-component`

- Placer dans `src/components/[domain]/`
- Créer le fichier `.test.tsx` correspondant
- Exporter depuis `index.ts`

2. **Créer une route API** :

- Créer dossier dans `src/app/api/[entity]/`
- Implémenter `route.ts` (GET/POST/PUT/DELETE)
- Valider avec Zod schema
- Logger toutes les actions

3. **Modifier la base de données** :

- Créer migration dans `src/seed/migrations/`
- Exécuter `npm run migrate`
- Mettre à jour types dans `src/types/database/`

Raccourcis Cursor Essentiels

```
| Raccourci | Action |
|-----|-----|
| `Cmd+K` | Command Palette |
| `Cmd+Shift+L` | Sélectionner toutes les occurrences |
| `Cmd+.` | Quick Fix (TypeScript) |
| `Cmd+Click` | Aller à la définition |
| `Cmd+/' | Commenter/Décommenter |
```

Vérifications Pré-commit (Git Hooks)

Installer Husky pour exécuter :

```
```bash
npm run lint
npm run format
npm test
```
```

Architecture des Données PostgreSQL Améliorée

Tables Principales avec Relations

```
```sql
-- Relations
ALTER TABLE users ADD CONSTRAINT fk_user_role FOREIGN KEY (role_id)
REFERENCES roles(id);
ALTER TABLE products ADD CONSTRAINT fk_product_category FOREIGN KEY
(category_id) REFERENCES categories(id);
ALTER TABLE products ADD CONSTRAINT fk_product_updated_by FOREIGN KEY
(updated_by) REFERENCES users(id);
```

```

ALTER TABLE orders ADD CONSTRAINT fk_order_user FOREIGN KEY (user_id)
REFERENCES users(id);
ALTER TABLE trackings ADD CONSTRAINT fk_tracking_order FOREIGN KEY (order_id)
REFERENCES orders(id);
ALTER TABLE trackings ADD CONSTRAINT fk_tracking_updated_by FOREIGN KEY
(updated_by) REFERENCES users(id);
ALTER TABLE comments ADD CONSTRAINT fk_comment_product FOREIGN KEY
(product_id) REFERENCES products(id);
ALTER TABLE comments ADD CONSTRAINT fk_comment_user FOREIGN KEY (user_id)
REFERENCES users(id);

```

-- Indexes pour performance

```

CREATE INDEX idx_orders_status ON orders(status);
CREATE INDEX idx_trackings_status ON trackings(status);
CREATE INDEX idx_products_price ON products(price);
CREATE INDEX idx_users_role ON users(role_id);

```

-- Trigger pour updated\_at

```

CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
 NEW.updated_at = NOW();
 NEW.updated_by = COALESCE(NEW.updated_by, NULL);
 RETURN NEW;
END;
$$ language 'plpgsql';

```

```

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

```

---

## 🎨 \*\*Thème Bleu Dynamique (Admin Panel)\*\*

#### 📄 `src/components/admin/SettingsPanel.tsx`

```

``typescript
'use client';

```

```

import { useState, useEffect } from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { Button } from '@components/ui/button';
import { Input } from '@components/ui/input';

```

```

import { useThemeStore } from '@stores/useThemeStore';
import { api } from '@services/api';
import { logger } from '@services/monitoring/logger';
import { ColorPicker } from '@components/ui/color-picker'; // Composant custom

export function SettingsPanel() {
 const { primaryColor, setPrimaryColor } = useThemeStore();
 const [isSaving, setIsSaving] = useState(false);

 const form = useForm({
 resolver: zodResolver(settingsSchema),
 defaultValues: {
 appName: 'AgroDeep',
 primaryColor: '#2563eb',
 timezone: 'UTC',
 },
 });

 useEffect(() => {
 // Charger les settings depuis DB
 api.settings.get().then((settings) => {
 form.reset(settings);
 setPrimaryColor(settings.primaryColor);
 });
 }, []);

 const onSubmit = async (data: any) => {
 try {
 setIsSaving(true);
 await api.settings.update(data);
 setPrimaryColor(data.primaryColor);
 logger.info('Settings updated', { data });
 // Notification toast
 } catch (error) {
 logger.error('Failed to update settings', { error });
 } finally {
 setIsSaving(false);
 }
 };

 return (
 <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6 max-w-2xl">
 <div>
 <label>Nom de l'application</label>
 <Input {...form.register('appName')} />
 </div>

 <div>

```

```

 <label>Couleur primaire</label>
 <ColorPicker
 value={form.watch('primaryColor')}
 onChange={(color) => form.setValue('primaryColor', color)}
 />
 </div>

 <Button type="submit" disabled={isSaving} className="bg-blue-600
hover:bg-blue-700">
 {isSaving ? 'Sauvegarde...' : 'Sauvegarder'}
 </Button>
</form>
);
}
...

🛡️ **Sécurité & RGPD**

Password Hashing
```typescript
import bcrypt from 'bcryptjs';

export async function hashPassword(password: string): Promise<string> {
  const saltRounds = 12;
  return bcrypt.hash(password, saltRounds);
}

export async function verifyPassword(password: string, hash: string): Promise<boolean> {
  return bcrypt.compare(password, hash);
}
...

### **Injection SQL Protection**
Toutes les requêtes utilisent **paramétrisation** :
```typescript
// ✅ Sécurisé
await db.query('SELECT * FROM users WHERE email = $1', [email]);

// ❌ Non sécurisé (NE JAMAIS FAIRE)
await db.query('SELECT * FROM users WHERE email = '${email}');
...

Données Sensibles
- Variables `.env.local` **jamais committées**
- Configuration `.env.local.example` avec valeurs factices
- Logs ne jamais enregistrer mots de passe, numéros carte

```

---

##  \*\*Performance & Optimisations\*\*

### \*\*1. Caching Redis\*\*

```typescript

import { redis } from '@services/messaging/redis';

export async function getProductWithCache(productId: number) {
 const cacheKey = `product:\${productId}`;

// 1. Vérifier cache

const cached = await redis.get(cacheKey);

if (cached) return JSON.parse(cached);

// 2. Charger DB

const product = await db.query('SELECT * FROM products WHERE id = \$1', [productId]);

// 3. Mettre en cache (TTL 5 min)

await redis.setex(cacheKey, 300, JSON.stringify(product));

return product;

}

```

### \*\*2. Pagination Optimisée\*\*

```typescript

// Cursor-based pagination (plus performant que OFFSET)

export async function getProducts(cursor: number = 0, limit: number = 20) {

const result = await db.query(
 `SELECT * FROM products

WHERE id > \$1

ORDER BY id ASC

LIMIT \$2`,

[cursor, limit + 1] // +1 pour savoir s'il y a plus

);

const hasMore = result.rows.length > limit;

const products = hasMore ? result.rows.slice(0, -1) : result.rows;

return {

products,

nextCursor: hasMore ? products[products.length - 1].id : null,

};

}

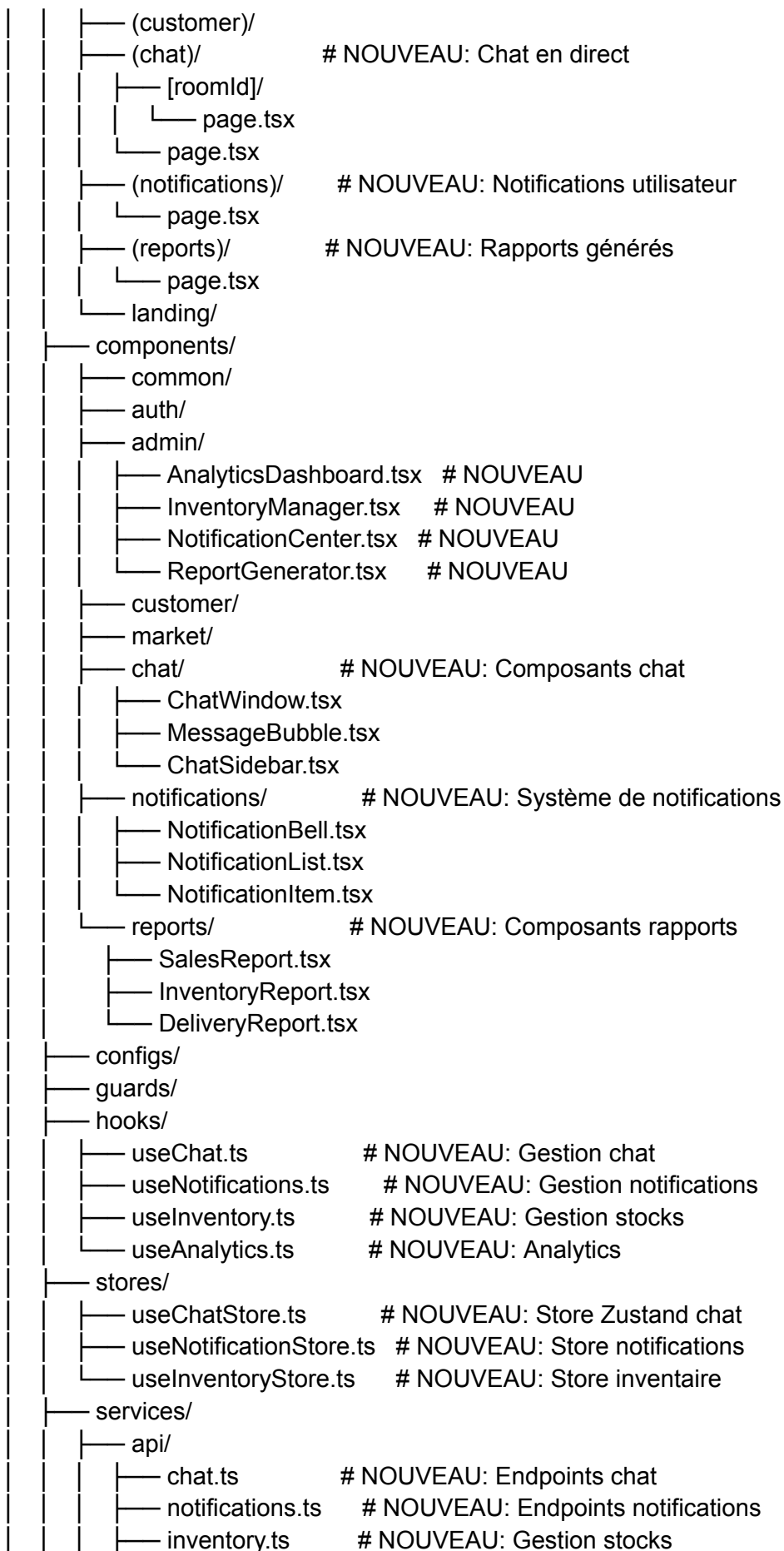
```

---  
**## 🎬 \*\*Prochaines Étapes pour Cursor\*\***

1. **\*\*Créer les migrations\*\*** : Exécuter `npm run migrate`
  2. **\*\*Seeder la DB\*\*** : Exécuter `npm run seed`
  3. **\*\*Démarrer le dev server\*\*** : `npm run dev`
  4. **\*\*Ouvrir `localhost:3000`\*\*** : Vérifier landing page
  5. **\*\*Se connecter en admin\*\*** : admin@agrodeep.com | Admin123
  6. **\*\*Configurer le thème\*\*** : Aller dans `/admin/settings`
- 

**Voici la structure mise à jour du projet AgroDeep avec les nouvelles fonctionnalités intégrées, tout en conservant l'architecture PostgreSQL existante :**

...  
agrodep-platform/  
├── docs/  
├── src/  
│ ├── app/  
│ │ ├── (auth)/  
│ │ ├── (admin)/  
│ │ │ ├── overview/  
│ │ │ ├── users/  
│ │ │ ├── products/  
│ │ │ ├── categories/  
│ │ │ ├── orders/  
│ │ │ ├── trackings/  
│ │ │ ├── settings/  
│ │ │ ├── analytics/ # NOUVEAU: Analytics avancés  
│ │ │ ├── inventory/ # NOUVEAU: Gestion des stocks  
│ │ │ ├── notifications/ # NOUVEAU: Centre de notifications  
│ │ └── (market)/



```

├── analytics.ts # NOUVEAU: Endpoints analytics
├── database/
├── storage/
├── messaging/
├── websocket.ts # NOUVEAU: Service WebSocket
├── notification.ts # NOUVEAU: Service notifications
├── monitoring/
├── types/
│ ├── database/
│ │ ├── chat.ts # NOUVEAU: Types chat
│ │ ├── notifications.ts # NOUVEAU: Types notifications
│ │ └── inventory.ts # NOUVEAU: Types inventaire
│ └── api/
├── utils/
├── seed/
│ ├── migrations/
│ │ ├── 005_chat_notifications.ts # NOUVEAU
│ │ ├── 006_inventory_analytics.ts # NOUVEAU
│ │ └── 007_reports_features.ts # NOUVEAU
│ └── data/
│ ├── chat.ts # NOUVEAU: Seed chat
│ ├── notifications.ts # NOUVEAU: Seed notifications
│ └── inventory.ts # NOUVEAU: Seed inventaire
├── logs/
├── test/
├── scripts/
├── docker-compose.yml
├── Dockerfile
├── package.json
└── README.md

```

## \*\*Nouvelles Fonctionnalités Ajoutées :\*\*

### \*\*1. Système de Chat en Temps Réel\*\*

- \*\*WebSocket\*\* avec Socket.IO pour communication bidirectionnelle
- \*\*Salles de chat\*\* privées et groupées
- \*\*Historique des messages\*\* persistant dans PostgreSQL
- \*\*Notifications\*\* de nouveaux messages

### \*\*2. Centre de Notifications\*\*

- \*\*Notifications push\*\* en temps réel
- \*\*Multi-canaux\*\* : email, in-app, SMS
- \*\*Système de préférences\*\* par utilisateur
- \*\*Historique\*\* et marquage comme lu/non lu

### \*\*3. Gestion des Stocks (Inventory)\*\*

- \*\*Suivi en temps réel\*\* des quantités

- **Alertes automatiques** de stock bas
- **Gestion des fournisseurs** et réapprovisionnement
- **Historique des mouvements** de stock

#### ### **4. Analytics Avancés**

- **Dashboard analytique** avec métriques en temps réel
- **Rapports personnalisables** (ventes, stocks, livraisons)
- **Export PDF/Excel** des rapports
- **Visualisations graphiques** interactives

#### ### **5. Génération de Rapports**

- **Rapports automatisés** programmés
- **Templates personnalisables**
- **Distribution automatique** par email
- **Archivage** des rapports historiques

#### ## **Migrations PostgreSQL Ajoutées :**

```
```sql
```

```
-- Table chat_rooms
```

```
CREATE TABLE chat_rooms (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255),
  type VARCHAR(50) DEFAULT 'private',
  participants INTEGER[],
  last_message_at TIMESTAMP WITH TIME ZONE,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

```
-- Table chat_messages
```

```
CREATE TABLE chat_messages (
  id SERIAL PRIMARY KEY,
  room_id INTEGER REFERENCES chat_rooms(id) ON DELETE CASCADE,
  user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
  content TEXT NOT NULL,
  message_type VARCHAR(50) DEFAULT 'text',
  is_read BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

```
-- Table notifications
```

```
CREATE TABLE notifications (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  title VARCHAR(255) NOT NULL,
  message TEXT NOT NULL,
  type VARCHAR(50) NOT NULL,
  is_read BOOLEAN DEFAULT FALSE,
```

```

    metadata JSONB,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Table inventory
CREATE TABLE inventory (
    id SERIAL PRIMARY KEY,
    product_id INTEGER UNIQUE REFERENCES products(id) ON DELETE CASCADE,
    quantity INTEGER NOT NULL DEFAULT 0,
    reserved_quantity INTEGER DEFAULT 0,
    reorder_point INTEGER DEFAULT 10,
    last_updated TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    supplier_id INTEGER,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Table inventory_logs
CREATE TABLE inventory_logs (
    id SERIAL PRIMARY KEY,
    inventory_id INTEGER REFERENCES inventory(id) ON DELETE CASCADE,
    user_id INTEGER REFERENCES users(id),
    change_type VARCHAR(50),
    quantity_change INTEGER,
    previous_quantity INTEGER,
    new_quantity INTEGER,
    reason TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

```

Services Ajoutés :

Service WebSocket :

```

``typescript
// src/services/messaging/websocket.ts
import { Server } from 'socket.io';
import { createAdapter } from '@socket.io/redis-adapter';
import { redis } from './redis';

export class WebSocketService {
    private io: Server;

    constructor(server: any) {
        this.io = new Server(server, {
            cors: { origin: process.env.CLIENT_URL },
            adapter: createAdapter(redis, redis.duplicate())
        });
    }
}

```

```

    this.setupEvents();
  }

  private setupEvents() {
    this.io.on('connection', (socket) => {
      // Gestion des salles de chat
      socket.on('join-room', (roomId) => {
        socket.join(roomId);
      });

      // Envoi de messages
      socket.on('send-message', async (data) => {
        // Sauvegarde en base de données
        // Diffusion aux participants
        socket.to(data.roomId).emit('new-message', data);
      });
    });
  }
}
...

```

```

### **Service de Notifications :**
``typescript
// src/services/messaging/notification.ts
export class NotificationService {
  static async send(userId: number, notification: {
    title: string;
    message: string;
    type: string;
    metadata?: any;
  }) {
    // Enregistrement en base
    // Envoi WebSocket
    // Envoi email si configuré
  }

  static async sendBulk(userIds: number[], notification: any) {
    // Notifications groupées
  }
}
...

```

```

## **Package.json - Dépendances Ajoutées :**

``json
{
  "dependencies": {
    "socket.io": "^4.7.4",

```

```

    "socket.io-client": "^4.7.4",
    "@socket.io/redis-adapter": "^8.2.1",
    "pdfkit": "^0.14.0",
    "exceljs": "^4.4.0",
    "chart.js": "^4.4.1",
    "react-chartjs-2": "^5.2.0",
    "recharts": "^2.10.3",
    "date-fns": "^3.3.1",
    "nodemailer": "^6.9.9",
    "twilio": "^4.19.4"
  }
}
...

```

Configuration Docker-Compose Mise à Jour :

```

```yaml
services:
 # Services existants...
 websocket:
 image: node:18-alpine
 container_name: agrodeep-websocket
 restart: unless-stopped
 working_dir: /app
 ports:
 - "3001:3001"
 volumes:
 - ./src/services/messaging/websocket.ts:/app/websocket.ts
 depends_on:
 - redis
 command: ["node", "websocket.ts"]
...

```

## \*\*Scripts Ajoutés :

```

```json
{
  "scripts": {
    "dev:ws": "nodemon src/services/messaging/websocket.ts",
    "generate:report": "tsx src/scripts/generate-report.ts",
    "backup:inventory": "tsx src/scripts/backup-inventory.ts",
    "notify:stock": "tsx src/scripts/notify-low-stock.ts"
  }
}
...

```