



GUIDE DE DÉMARRAGE - TITAN EMR V31.4 BACKEND

Guide complet pour démarrer le backend TITAN V31.4 en développement et production.

PRÉREQUIS

Logiciels requis

```
bash

Node.js >= 18.x
PostgreSQL >= 15.x
pnpm >= 8.x (ou npm/yarn)
```

Extensions PostgreSQL

```
sql

-- Vérifier que ces extensions sont activées
SELECT * FROM pg_available_extensions
WHERE name IN ('uuid-ossp', 'pgcrypto', 'pg_trgm', 'unaccent', 'btree_gin');
```

INSTALLATION

1. Cloner et installer

```
bash

# Cloner le projet
git clone <repo-url>
cd titan-emr

# Installer les dépendances
pnpm install
```

2. Configuration environnement

Créer `.env` à la racine :

```
bash
```

```
# Database
DATABASE_URL=postgresql://titan:password@localhost:5432/titan_emr

# Auth
JWT_SECRET=your-super-secret-jwt-key-change-in-production-min-32-chars
COOKIE_SECRET=your-cookie-secret-key-change-in-production

# OAuth Manus (si utilisé)
OAUTH_SERVER_URL=https://oauth.example.com
APP_ID=your-app-id

# Optionnel
NODE_ENV=development
PORT=3000
```

3. Créer la base de données

```
bash

# Créer la DB
createdb titan_emr

# Ou via psql
psql -U postgres
CREATE DATABASE titan_emr;
\q
```

4. Exécuter le schéma SQL

```
bash

# Exécuter le schéma V31.4
psql -U postgres -d titan_emr -f "EMR TITAN V31.4 FINAL - PRODUCTION ENTERPRISE.sql"

# Vérifier que tout est OK
# Devrait afficher : "✓ EMR TITAN V31.4 FINAL - DÉPLOIEMENT PRODUCTION 100% VÉRIFIÉ"
```

5. Configurer Drizzle

```
bash
```

```
# Push le schéma Drizzle (si modifications)
```

```
pnpm db:push
```

```
# Ou générer les migrations
```

```
pnpm db:generate
```

```
pnpm db:migrate
```

🔒 CONFIGURATION SÉCURITÉ (OBLIGATOIRE)

1. Changer la clé de chiffrement PGP

```
sql
```

```
-- ! CRITIQUE : Changer cette clé AVANT de créer des données
```

```
UPDATE system_settings
```

```
SET value = jsonb_set(
```

```
    value,
```

```
    '{encryption_key}',
```

```
    to_jsonb('VOTRE_CLE_256_BITS_ALEATOIRE_GENEREE_ICI'::text)
```

```
)
```

```
WHERE key = 'encryption';
```

Générer une clé sécurisée :

```
bash
```

```
# Option 1 : OpenSSL
```

```
openssl rand -base64 32
```

```
# Option 2 : Node.js
```

```
node -e "console.log(require('crypto').randomBytes(32).toString('base64'))"
```

2. Changer le mot de passe admin

```
sql
```

```
-- Changer le mot de passe par défaut
```

```
UPDATE users
```

```
SET hashed_password = crypt('VotreNouveauMotDePasseComplexe!2025', gen_salt('bf'))
```

```
WHERE username = 'admin';
```

3. Créer un centre

```
sql
```

-- Créez votre premier centre

```
INSERT INTO centers (name, code, timezone, is_active)
VALUES ('Clinique Centrale', 'CC001', 'Africa/Tunis', true)
RETURNING id;
```

-- Noter l'ID retourné pour l'étape suivante

4. Assigner l'admin au centre

```
sql
```

-- Remplacer <CENTER_ID> par l'UUID du centre créé

```
UPDATE users
SET center_id = '<CENTER_ID>'
WHERE username = 'admin';
```

DÉMARRAGE

Mode développement

```
bash
```

Démarrer le serveur avec hot-reload

```
pnpm dev
```

Le serveur démarre sur http://localhost:3000

```
# Swagger docs : http://localhost:3000/docs
```

Mode production

```
bash
```

Build

```
pnpm build
```

Démarrer

```
pnpm start
```

Avec Docker

```
bash

# Build l'image
docker-compose up -d --build

# Voir les logs
docker-compose logs -f api

# Arrêter
docker-compose down
```

TESTS

1. Test connexion DB

```
bash

curl http://localhost:3000/api/system/health
# ✓ Devrait retourner : { "status": "ok", "database": "connected" }
```

2. Test authentication

```
bash

# Login
curl -X POST http://localhost:3000/api/auth/login \
-H "Content-Type: application/json" \
-d '{
  "username": "admin",
  "password": "Admin123!"'
}

# Devrait retourner un JWT
```

3. Test RLS

```
bash
```

```
# Créer 2 centres
psql -d titan_emr -c "INSERT INTO centers (name, code) VALUES ('Centre A', 'CA'), ('Centre B', 'CB');"

# Créer 2 users dans des centres différents
# User 1 → Centre A
# User 2 → Centre B

# Créer des patients dans chaque centre
# Vérifier qu'un user ne voit que les patients de son centre
```

4. Test PGP

```
bash

# Créer un appointment avec notes
curl -X POST http://localhost:3000/api/appointments \
-H "Authorization: Bearer <JWT>" \
-H "Content-Type: application/json" \
-d '{
    "patientId": "<UUID>",
    "scheduledFrom": "2025-12-01T10:00:00Z",
    "scheduledTo": "2025-12-01T11:00:00Z",
    "notes": "Notes confidentielles"
}'

# Vérifier en DB que notes_encrypted est BYTEA
psql -d titan_emr -c "SELECT id, notes_encrypted FROM appointments LIMIT 1;"

# Récupérer via API et vérifier que notes sont déchiffrées
curl http://localhost:3000/api/appointments/<ID> \
-H "Authorization: Bearer <JWT>"
```

5. Test triggers

```
bash
```

```
# Test trigger invoice totals  
# 1. Créer une facture  
# 2. Ajouter des items  
# 3. Vérifier que total_amount est calculé automatiquement
```

```
curl -X POST http://localhost:3000/api/invoices/<ID>/items \  
-H "Authorization: Bearer <JWT>" \  
-H "Content-Type: application/json" \  
-d '{  
    "invoiceId": "<UUID>",  
    "description": "Consultation",  
    "quantity": 1,  
    "unitPrice": 50  
}'
```

Le total_amount de la facture doit être mis à jour automatiquement

TESTS UNITAIRES (À IMPLÉMENTER)

```
bash  
  
# Installer jest  
pnpm add -D jest @types/jest ts-jest  
  
# Créer tests/__tests__/rls.test.ts  
# Créer tests/__tests__/pgp.test.ts  
# Créer tests/__tests__/auth.test.ts  
  
# Lancer les tests  
pnpm test
```

Exemple de test RLS :

```
typescript
```

```
// tests/_tests_/rls.test.ts
import { testRLSIsolation } from '../server/db/rls';

describe('RLS Isolation', () => {
  it('should isolate data between centers', async () => {
    const result = await testRLSIsolation(
      { id: 'user1', centerId: 'center1' },
      { id: 'user2', centerId: 'center2' }
    );

    expect(result.isolated).toBe(true);
  });
});
```

COMMANDES UTILES

Database

```
bash

# Voir les logs PostgreSQL
sudo tail -f /var/log/postgresql/postgresql-15-main.log

# Voir les connexions actives
psql -d titan_emr -c "SELECT * FROM pg_stat_activity WHERE datname='titan_emr';"

# Voir la taille de la DB
psql -d titan_emr -c "SELECT pg_size_pretty(pg_database_size('titan_emr'));""

# Backup
pg_dump -U postgres titan_emr > backup_$(date +%Y%m%d).sql

# Restore
psql -U postgres -d titan_emr < backup_20251122.sql
```

Drizzle

```
bash
```

```
# Voir le schéma actuel  
pnpm drizzle-kit introspect:pg
```

```
# Générer les types TypeScript  
pnpm drizzle-kit generate:pg
```

```
# Push les changements  
pnpm drizzle-kit push:pg
```

Debug

```
bash  
  
# Activer les logs SQL Drizzle  
# Dans .env  
DEBUG=drizzle:  
  
# Logs détaillés tRPC  
# Dans .env  
TRPC_LOG_LEVEL=debug
```

TROUBLESHOOTING

Erreur : "Database not available"

```
bash  
  
# Vérifier que PostgreSQL est démarré  
sudo systemctl status postgresql  
  
# Vérifier la connexion  
psql -U postgres -d titan_emr -c "SELECT 1;"  
  
# Vérifier DATABASE_URL dans .env  
echo $DATABASE_URL
```

Erreur : "RLS context not set"

```
bash
```

```

# Vérifier que les fonctions RLS existent
psql -d titan_emr -c "\df get_user_center_id"
psql -d titan_emr -c "\df get_current_user_id"

# Vérifier que RLS est activé
psql -d titan_emr -c "
    SELECT tablename, rowsecurity
    FROM pg_tables
    WHERE schemaname = 'public'
    AND tablename IN ('patients', 'appointments', 'invoices');
"

```

Erreur : "Encryption key not found"

```

bash

# Vérifier que system_settings existe
psql -d titan_emr -c "SELECT * FROM system_settings WHERE key='encryption';"

# Si absent, réexécuter le schéma SQL
psql -d titan_emr -f "EMR TITAN V31.4 FINAL - PRODUCTION ENTERPRISE.sql"

```

Erreur : "Double-booking" inattendu

```

bash

# Vérifier la contrainte unique_appt_schedule
psql -d titan_emr -c "
    SELECT conname, contype, pg_get_constraintdef(oid)
    FROM pg_constraint
    WHERE conname = 'unique_appt_schedule';
"

# Lister les RDV en conflit
psql -d titan_emr -c "
    SELECT patient_id, scheduled_from, scheduled_to, center_id
    FROM appointments
    WHERE patient_id = '<UUID>'
    AND scheduled_from < '<END_TIME>'
    AND scheduled_to > '<START_TIME>';
"

```

MONITORING PRODUCTION

Métriques PostgreSQL

```
sql

-- Voir les requêtes lentes
SELECT query, calls, total_time, mean_time
FROM pg_stat_statements
ORDER BY mean_time DESC
LIMIT 10;

-- Voir les index non utilisés
SELECT schemaname, tablename, indexname
FROM pg_stat_user_indexes
WHERE idx_scan = 0
AND schemaname = 'public';

-- Voir la taille des tables
SELECT
    schemaname,
    tablename,
    pg_size.pretty(pg_total_relation_size(schemaname||'.'||tablename)) AS size
FROM pg_tables
WHERE schemaname = 'public'
ORDER BY pg_total_relation_size(schemaname||'.'||tablename) DESC;
```

Logs application

```
bash

# Avec PM2
pm2 logs titan-api

# Avec systemd
sudo journalctl -u titan-api -f

# Avec Docker
docker logs -f titan-api
```

CHECKLIST AVANT PRODUCTION

Infrastructure

- [] PostgreSQL 15+ installé et configuré
- [] Backup automatique configuré (pg_dump quotidien)
- [] Monitoring activé (Prometheus/Grafana)
- [] Reverse proxy configuré (Nginx/Caddy)
- [] SSL/TLS activé (Let's Encrypt)
- [] Firewall configuré (ufw/iptables)

Database

- [] Schéma V31.4 exécuté sans erreurs
- [] Extensions activées (uuid-ossp, pgcrypto, pg_trgm, unaccent, btree_gin)
- [] RLS activé et testé sur toutes les tables sensibles
- [] Triggers fonctionnels (recalc, commissions, stock, search)
- [] Index créés et optimisés
- [] Vues statistiques accessibles

Sécurité

- [] Clé de chiffrement PGP changée
- [] Mot de passe admin changé (min 16 caractères)
- [] JWT_SECRET changé (min 32 caractères)
- [] COOKIE_SECRET changé
- [] Variables sensibles dans .env (pas dans git)
- [] RLS testé (isolation entre centres)
- [] PGP testé (chiffrement/déchiffrement)
- [] HTTPS forcé en production

Configuration

- [] Au moins un centre créé
- [] Admin assigné à un centre (ou center_id = NULL)
- [] Rôles créés (admin, doctor, nurse, receptionist)
- [] Permissions assignées
- [] File stores configuré (local ou S3)
- [] Timezone configuré (UTC ou local)

Tests

- [] Tests RLS passés (isolation centres)
- [] Tests PGP passés (chiffrement notes)
- [] Tests triggers passés (totaux, commissions, stock)
- [] Tests authentification passés (login, JWT)
- [] Tests charge passés (>100 requêtes/sec)
- [] Tests double-booking passés

Documentation

- [] API docs générées (Swagger)
- [] Guide déploiement écrit
- [] Procédures backup documentées
- [] Contacts support définis
- [] Plan de disaster recovery établi

RESSOURCES

Documentation

- tRPC : <https://trpc.io/docs>
- Drizzle ORM : <https://orm.drizzle.team/docs>
- PostgreSQL pgcrypto : <https://www.postgresql.org/docs/15/pgcrypto.html>
- PostgreSQL RLS : <https://www.postgresql.org/docs/15/ddl-rowsecurity.html>

Outils

- Drizzle Studio : `pnpm drizzle-kit studio`
- Postman/Insomnia : Pour tester l'API
- pgAdmin : Interface graphique PostgreSQL
- DBeaver : Client SQL universel

Support

- Issues GitHub : <repo-url>/issues
- Email : support@titan-emr.local
- Slack : #titan-dev

PROCHAINES ÉTAPES

Après avoir validé les modules critiques :

1. **Implémenter les routers secondaires** (users, centers, doctors, encounters...)
2. **Créer les tests unitaires** (Jest + Supertest)
3. **Optimiser les performances** (Redis cache, connection pooling)

4. Ajouter le monitoring (Prometheus + Grafana)

5. Déployer en staging puis en production

Version du guide : 1.0

Compatible avec : TITAN V31.4 GOLD MASTER

Date : 22/11/2025