**Inkspot – Manuel d’utilisation technique**

Version: 1.0

Généré automatiquement à partir du code source.

# **Sommaire**

# **1. Introduction**

Présentation technique d’Inkspot.

Inkspot est une application destinée aux professionnels qui souhaitent présenter leur travail, échanger avec des clients et gérer des rendez-vous avec un parcours de paiement intégré. L’objectif technique est double: offrir une expérience fluide côté utilisateur (navigation rapide, interactions en temps réel, sécurité par défaut) et fournir une plateforme d’exploitation robuste côté opérateurs (observabilité, déploiements reproductibles, résilience).

Le système est construit sur Next.js (App Router), avec des routes d’API qui encapsulent toutes les fonctions critiques: authentification, gestion des utilisateurs, messagerie, rendez-vous, paiements Stripe, notifications, et export/visualisation de métrriques. Prisma orchestre l’accès à PostgreSQL avec un schéma riche en relations (utilisateurs, conversations, messages, rendez-vous, paiements, factures, notifications…). L’écosystème comprend Redis (cache et coordination légère), AWS S3 pour le stockage d’objets et un reverse proxy Nginx en frontal pour la terminaison TLS, la montée en charge horizontale et la gestion des websockets.

Du point de vue de la sécurité, Inkspot applique un middleware spécialisé (limitation de débit, assainissement et en-têtes de protection) et une authentification via NextAuth (JWT) avec possibilité d’OAuth (Google). Les communications sensibles sont chiffrées en transit (HTTPS), et le modèle de données est conçu pour minimiser les surfaces d’attaque (ex: pas d’accès direct au stockage d’objets, politique de permissions côté API).

Côté exploitation, une route "/api/metrics" expose des indicateurs métiers et techniques en format Prometheus. Un stack Prometheus + Grafana récupère ces métriques (récupérations applicatives, exporteurs système et base de données) afin d’alimenter des tableaux de bord et des alertes. Le tout est prévu pour tourner localement (Docker Compose dev) et en production (Docker Compose prod), en conservant des principes identiques: déclaratif, traçable et observable. Pour les détails exhaustifs et extraits de configuration, voir les annexes D, E et F.

## **Architecture globale de l’application**

Voir Diagramme D1 — Architecture globale (Annexe H). 🦊 Ajoutez un signet Word nommé D1 sur le diagramme en Annexe H et créez un lien depuis cette section si souhaité.

**Image (URL):** https://miro.medium.com/v2/resize:fit:1400/1\*WcXghWz8nRwl0FdrVv5f9g.png

**📌 Screen:** Dashboard principal de l’app en local (http://localhost:3000) — page d’accueil ou tableau de bord PRO

# **2. Stack Technique**

## **Stack locale (frameworks, libs, environnement dev)**

### **Frameworks principaux**

**Next:** 15.4.6

**React:** ^18

**TypeScript:** ^5

**Prisma:** ^6.14.0

### **Librairies clés**

**next-auth:** latest

**stripe:** ^18.4.0

**@stripe/stripe-js:** ^7.8.0

**@aws-sdk/client-s3:** latest

**web-push:** latest

**zustand:** ^5.0.7

Côté développement, l’environnement recommandé repose sur Node.js 18+, npm/pnpm, Prisma (CLI) et Docker Desktop pour simuler les services dépendants (PostgreSQL, Redis, Prometheus, Grafana, MailHog). Les scripts npm permettent de lancer le serveur en mode développement, d’exécuter les tests, de pousser le schéma Prisma et de peupler la base. La convention est de regrouper les routes API sous "app/api/\*" et de centraliser les services horizontaux (Stripe, monitoring, stockage) dans "lib/\*".

En production, l’application est conteneurisée. Un reverse proxy Nginx reçoit le trafic, gère TLS et relaie vers l’application (port 3000) et le service WebSocket (port 3001). Les variables d’environnement (ex: NEXTAUTH\_SECRET, clés Stripe, identifiants SMTP, configuration S3) sont injectées côté conteneur. Les volumes assurent la persistance (ex: uploads) et les dépendances (PostgreSQL, Redis) tournent en services dédiés. Pour des raisons d’observabilité, l’application expose /api/metrics; les exporteurs système (Node Exporter) et base (Postgres Exporter) complètent la vue d’ensemble.

Distinctions entre environnements: en local, priorité à la rapidité (hot reload, data de test, services dockérisés). En staging, on se rapproche de la prod (env vars réalistes, builds optimisés) pour vérifier les parcours complets. En prod, le focus est la sécurité, l’isolation et l’élasticité: secrets gérés par l’environnement, reverse proxy durci, monitoring actif et procédure de rollback disponible. Les écarts de configuration sont explicités dans la section Build & Déploiement et illustrés en Annexe D.

```mermaid

graph LR

Dev[Dev Local] -- push --> Repo[Git]

Repo -- CI --> Build[CI Build + Tests]

Build -- deploy --> Prod[Production]

Prod -->|/api/metrics| Prometheus --> Grafana

```

```mermaid

flowchart TD

Commit --> CI[CI/CD Pipeline]

CI --> Test[Unit/E2E]

Test --> Build[Next build]

Build --> Deploy[Deploy Docker/Vercel]

Deploy --> Notify[Notifications]

```

```mermaid

graph TD

User --> LB[Reverse Proxy/Nginx]

LB --> Next[Next.js Runtime]

Next --> DB[(PostgreSQL)]

Next --> S3[AWS S3]

Next --> Stripe[Stripe]

Next --> Metrics[/api/metrics/]

```

🦊 Insérez en Annexe F une capture du pipeline CI/CD (ex: GitHub Actions/Vercel) et une capture de la base de données (Prisma Studio/pgAdmin). Créez des signets F1 (pipeline) et F2 (DB) et ajoutez des liens depuis cette section si nécessaire.

# **3. Dépendances**

La gestion des dépendances se fait via npm (lockfile), avec un soin particulier apporté aux bibliothèques critiques: next, react, prisma, next-auth, stripe, @aws-sdk. Les mises à jour suivent le SemVer. Une politique pragmatique est recommandée: sécurités et correctifs en priorité (patch), nouvelles fonctionnalités par opportunité (minor), et upgrades majeurs (major) soigneusement planifiés (tests E2E, canary, rollback).

Pour éviter la stagnation, exécuter régulièrement un "npm outdated" et qualifier l’impact sur l’application (breaking changes connus, notes de versions). Les dépendances transverses (UI, charting, états) doivent être stabilisées avant d’introduire d’autres briques. En particulier, Stripe, Prisma et NextAuth doivent rester proches de versions supportées, car elles structurent les flux de paiement, de données et d’accès.

Les dépendances de développement (ESLint, TypeScript, Jest/Playwright) garantissent qualité et cohérence. Un budget de maintenance régulier est nécessaire pour maintenir l’outillage (linters, types) et conserver un feedback rapide pendant le cycle de dev. Les sections Maintenance/Évolution et l’Annexe E proposent des checklists concrètes.

🦊 Ne pas coller ici le package.json. À la place, ajoutez en Annexe C une capture d’écran des sections scripts et dépendances de votre package.json (signets C1 pour scripts, C2 pour deps).

```mermaid

graph TD

Next --> React

Next --> NextAuth

API --> Prisma

API --> Stripe

Uploads --> AWS\_S3

```

**Image (URL):** https://raw.githubusercontent.com/semver/semver.org/master/static/img/semver.png

**📌 Screen:** Résultat de la commande: npm outdated

**📌 Screen:** Extrait du package.json (dépendances et scripts)

# **4. Monitoring**

L’observabilité repose sur trois piliers: métriques, logs et (optionnellement) traces. Les métriques sont exposées en texte (Prometheus exposition format) par /api/metrics: compteurs métiers (utilisateurs, posts, réservations, paiements), indicateurs de performance (temps de réponse), et santé (mémoire, uptime, connectivité DB/Redis). Ces métriques alimentent Prometheus qui conserve des séries temporelles interrogeables par Grafana.

Côté logs, on recommande PM2 pour capturer les journaux de l’application en production, et la journalisation Nginx côté reverse proxy. Pour isoler et résoudre des incidents, le couplage des métriques (alerte) et des logs (contexte) est déterminant. La mise en place de panels Grafana (disponibles dans "monitoring/grafana") aide à visualiser disponibilité, charge, latence et erreurs, et à repérer les régressions.

Pour l’alerte, privilégier des seuils simples au début (availability, taux d’erreur, latence P95/P99) puis affiner par composant (DB, WebSocket, Webhooks Stripe). L’objectif est d’obtenir des alertes utiles (actionnables) sans bruit. Les représentations Mermaid de cette section illustrent le flux logique (détection → notification → intervention). Les détails de configuration sont fournis en Annexe D.

## **Workflow Monitoring**

```mermaid

sequenceDiagram

participant App as Inkspot API

participant Prom as Prometheus

participant Graf as Grafana

App->>Prom: Expose /api/metrics

Prom->>Graf: Alimente panneaux

Graf-->>Ops: Alertes visuelles/notifications

```

```mermaid

graph TD

Error[Erreur App] --> Collect[Logs/Metrics]

Collect --> Detect[Détection seuils]

Detect --> Notify[Notification Slack/Email]

Notify --> Respond[Intervention]

```

🦊 En Annexe F, ajoutez deux captures: Grafana (signet F3) et logs temps réel (signet F4). Reliez ces captures à cette section si nécessaire.

🦊 Cette section renvoie vers l’Annexe D (D2). Placez un signet D2 au bloc Prometheus et créez un lien depuis ici si souhaité.

# **5. Traitement des anomalies**

La gestion d’incident suit un cycle standard: détection → triage → diagnostic → remédiation → post-mortem. Le triage distingue rapidement l’impact (client vs interne), l’urgence (SLA, production vs staging) et le périmètre (paiements, auth, messagerie, rendez-vous, uploads). Chaque incident documenté doit aboutir à des mesures concrètes: correctif, test d’intégration, ou amélioration d’observabilité.

Un exemple typique: échec du webhook Stripe. Symptômes: paiements en "pending" malgré un succès côté Stripe. Diagnostic: signature manquante, latence réseau ou parsing JSON. Remédiation: vérifier la variable STRIPE\_WEBHOOK\_SECRET, logs de /api/stripe/webhook, activer une alerte spécifique, et ajouter un test de non-régression (simulant l’événement). Post-mortem: checklist de déploiement, alerte dédiée, documentation opérateur.

Pour les incidents de performance, s’appuyer sur les métriques P95/P99, la saturation DB (connections) et le monitoring des erreurs 5xx. Les arbres de décision fournis aident à garder un raisonnement simple: est-ce une panne externe (Stripe, S3), une contrainte interne (DB/Redis), ou une régression applicative? Voir Annexe E pour des procédures et scripts de vérification.

```mermaid

flowchart LR

Detect --> Triage --> Fix --> PostMortem

Triage -->|Impact élevé| Critical

Triage -->|Impact faible| Minor

```

```mermaid

graph TD

Start --> IsCritical{Impact/SLA Critique ?}

IsCritical -- Oui --> Escalate[Escalade N2/N3]

IsCritical -- Non --> NormalFlow[Résolution standard]

Escalate --> RCA[Analyse cause racine]

NormalFlow --> RCA

RCA --> Lessons[Améliorations]

```

🦊 En Annexe F, ajoutez une capture d’un ticket d’incident résolu (signet F5) avec statut, timeline et résumé de résolution.

# **6. Maintenance et évolution**

La maintenance régulière comprend: mises à jour de sécurité, rotation des secrets, nettoyage des logs, vérification des sauvegardes, réindexation ou migrations DB planifiées, et calibration des limites (rate limit, tailles d’upload). Côté front, il s’agit de maintenir la cohérence UI et d’éviter l’entropie (tokens de design, composants partagés).

La montée en charge s’envisage horizontalement (plusieurs instances derrière Nginx); les états brefs (sessions, queues légères) sont externalisés (Redis), et les uploads vont sur S3. La base de données demeure le point de vérité, et les migrations doivent être atomiques et réversibles. L’Annexe D récapitule les artefacts de déploiement; l’Annexe E liste les checklists; l’Annexe F précise les captures d’écran à produire pour un runbook visuel.

Évolutions futures: rationaliser le temps réel (pattern pub/sub), enrichir la télémétrie (business KPIs), formaliser un SLA par surface (auth, paiement, messagerie), et automatiser plus loin le CI/CD (tests de charge, smoke tests post-déploiement). La roadmap Mermaid illustre un séquencement réaliste et progressif.

```mermaid

gantt

title Plan de maintenance et releases

dateFormat YYYY-MM-DD

section Maintenance

Mises à jour deps :active, des1, 2025-01-01, 7d

Migration DB : des2, 2025-01-10, 5d

section Releases

v1.1 : rel1, 2025-01-20, 3d

v1.2 : rel2, 2025-02-05, 3d

```

```mermaid

graph TD

Roadmap[Roadmap technique]

Roadmap --> Perf[Optimisations perfs]

Roadmap --> Obs[Observabilité avancée]

Roadmap --> Features[Nouvelles fonctionnalités]

```

🦊 En Annexe F, insérez: un planning de maintenance (Trello/Jira) — signet F6, et un plan de migration DB (schéma, étapes) — signet F7.

## **Build & Déploiement — Guide pas-à-pas (détaillé)**

Ce chapitre fournit une procédure pas-à-pas claire, avec explications simples, pour lancer Inkspot en local, puis le déployer en production. Les commandes sont données pour Windows (PowerShell), macOS et Linux. Pour les commandes Git, utilisez de préférence Git Bash [[Préférence utilisateur]].

1) Pré-requis

- Windows 10/11: installez Node.js 18+ (node -v), Docker Desktop, et (optionnel) WSL2 pour de meilleures performances Docker.
- macOS: installez Homebrew, puis: brew install node; installez Docker Desktop for Mac.
- Linux (Debian/Ubuntu): sudo apt-get update && sudo apt-get install -y curl build-essential; installez Node 18 LTS via nvm; installez Docker Engine + docker compose.

2) Cloner le projet et installer les dépendances

Windows PowerShell:
git clone <repo> && cd INKSPOT-5z
npm install

macOS/Linux:
git clone <repo> && cd INKSPOT-5z
npm install

3) Configurer les variables d’environnement (.env.local en développement)

Créez un fichier .env.local à la racine et renseignez au minimum:
DATABASE\_URL=postgresql://postgres:password@localhost:5432/inkspot
NEXTAUTH\_SECRET=une\_chaîne\_aléatoire\_longue
NEXTAUTH\_URL=http://localhost:3000
NEXT\_PUBLIC\_APP\_URL=http://localhost:3000
GOOGLE\_CLIENT\_ID=...
GOOGLE\_CLIENT\_SECRET=...
STRIPE\_SECRET\_KEY=sk\_live\_ou\_test
STRIPE\_PUBLISHABLE\_KEY=pk\_live\_ou\_test
STRIPE\_WEBHOOK\_SECRET=whsec\_...
AWS\_ACCESS\_KEY\_ID=...
AWS\_SECRET\_ACCESS\_KEY=...
AWS\_REGION=eu-west-1
AWS\_S3\_BUCKET=inkspot-assets
SMTP\_HOST=smtp.votre\_domaine
SMTP\_PORT=587
SMTP\_USER=...
SMTP\_PASS=...
REDIS\_URL=redis://localhost:6379

4) Services locaux (PostgreSQL, Redis, Prometheus, Grafana) via Docker (optionnel mais recommandé)

Le fichier docker-compose-dev.yml lance:
- Postgres (port 5432)
- Redis (port 6379)
- MailHog (port 8025) pour tester l’email
- Prometheus (port 9090) et Grafana (port 3001) pour le monitoring

Commande unique:
docker compose -f docker-compose-dev.yml up -d

5) Préparer la base de données avec Prisma

npx prisma db push # crée les tables à partir de prisma/schema.prisma
npx prisma db seed # remplit des données de démo si disponible

6) Lancer l’application en développement

npm run dev
Ouvrez http://localhost:3000

Vérifications utiles:
- http://localhost:3000/api/health doit répondre OK
- http://localhost:3000/api/metrics expose des métriques Prometheus (utilisées par Grafana)
- http://localhost:9090 (Prometheus) et http://localhost:3001 (Grafana) si lancés via compose

7) Build de production local (sans Docker)

npm run build
npm start
# Le serveur écoute sur le port 3000 par défaut

8) Déploiement production avec Docker Compose

Le fichier docker-compose.prod.yml définit les services:
- app (Next.js) et websocket (port 3001) avec healthchecks
- postgres, redis
- nginx (reverse proxy) avec TLS
- grafana, prometheus (profil monitoring)

Étapes:
a. Créez un fichier .env (ou .env.prod) avec toutes les variables (voir section 3).
b. Fournissez des certificats TLS dans nginx/ssl (cert.pem, key.pem). Pour Let’s Encrypt, utilisez certbot sur le serveur.
c. Démarrez: docker compose -f docker-compose.prod.yml up -d
d. Vérifiez la santé: docker compose -f docker-compose.prod.yml ps
e. Consultez les logs: docker compose -f docker-compose.prod.yml logs -f app nginx

9) Nginx – ce que fait la configuration

- Redirige HTTP vers HTTPS
- Proxy / vers app:3000
- Proxy /socket.io/ vers websocket:3001 (temps réel)
- Applique du rate limiting sur /api et /api/auth
- Expose en interne Grafana (3002) et Prometheus (9090)

10) WebSocket – vérification

- Assurez-vous que le service websocket écoute sur 3001
- Sous Nginx, /socket.io/ doit établir une connexion WebSocket (Upgrade)

11) Monitoring – vérification

- curl http://<votre-domaine>/api/metrics renvoie des métriques custom (ex: inkspot\_user\_total)
- Prometheus scrape /api/metrics (voir monitoring/prometheus/prometheus.yml)
- Grafana lit Prometheus et affiche les dashboards fournis

12) Dépannage (les plus fréquents)

- Erreur DB: vérifiez DATABASE\_URL, réseau Docker, et que prisma db push a bien tourné.
- Erreur Stripe: clés invalides ou webhook\_secret manquant; reconfigurez et redémarrez.
- 502 via Nginx: vérifiez que le conteneur app est healthy et écoute 3000; examinez logs Nginx.
- WebSocket ne se connecte pas: port 3001 exposé? Bloc /socket.io/ présent et Upgrade activé?
- Images/Uploads: configurez AWS S3 (clé, secret, bucket) et vérifiez les permissions.

### **Extraits utiles**

Annexe D (signets D1 à D4):
- D1: Diagramme architecture globale (Mermaid)
- D2: Prometheus scrape
- D3: Nginx (proxy, WebSocket)
- D4: Docker Compose prod (services applicatifs)

# **7. Ticketing et report client**

La gestion des demandes client et des tickets d’incident passe par un tableau Kanban (Jira, GitHub Issues). Chaque ticket doit inclure: contexte, pas de reproduction, impact utilisateur, hypothèse, plan de test, critères d’acceptation. Le flux standard: création → tri → affectation → développement → QA → validation → mise en production → suivi post-release.

Côté communication, un accusé réception rapide clarifie la priorité et l’ETA; une mise à jour régulière réduit l’incertitude pour le client. Les tickets techniques (dette, migrations, refactoring) sont planifiés pour éviter l’effet tunnel. L’Annexe F propose une liste de captures d’écran pour documenter les parcours clés; l’Annexe G fournit un glossaire afin d’aligner les termes.

```mermaid

flowchart LR

Client --> Support --> Ticket[Ticketing]

Ticket --> Dev[Dev] --> QA[QA] --> Done[Validé]

```

```mermaid

sequenceDiagram

participant U as Client

participant S as Support

participant D as Dev

U->>S: Ouvre un ticket

S->>D: Tri et attribution

D-->>U: Fix livré & feedback

```

**Image (URL):** https://www.atlassian.com/dam/jcr:7ac79c1f-ea54-4c8e-860d-b6c3c48fc4e8/Service-Level-Agreements-SLAs-hero.svg

**📌 Screen:** Tableau Kanban des tickets

**📌 Screen:** Ticket priorisé avec checklist

# **8. Conclusion**

Inkspot combine une base technique moderne (Next.js, Prisma, PostgreSQL) et des intégrations critiques (Stripe, S3), le tout encadré par une sécurité d’API et une observabilité de terrain (Prometheus/Grafana). Pour opérer au quotidien et faire évoluer la plateforme sereinement, deux principes guident la suite: visibilité (mesurer pour comprendre) et simplicité (privilégier des mécanismes clairs et auditables).

Les annexes rassemblent les détails opérationnels et techniques. Dans le corps du manuel, nous privilégions une narration orientée usage et exploitation: démarrer vite, diagnostiquer aisément, déployer avec confiance. Les sections peuvent être lues indépendamment; le sommaire vous permettra d’aller directement au sujet utile.

```mermaid

graph TD

Stack[Stack] --> Monitoring[Monitoring]

Monitoring --> Client[Qualité de service]

Client --> Feedback[Feedback Produit]

Feedback --> Stack

```

**📌 Screen:** Roadmap produit globale (vue marketing/produit)

# **Cheatsheet — Commandes utiles**

Commandes utiles (Windows, macOS, Linux) — Administration, déploiement et diagnostic:

1) Environnements & variables:
Windows PowerShell:
$env:NEXTAUTH\_URL
[System.Environment]::GetEnvironmentVariable("NEXTAUTH\_URL","Machine")
Get-ChildItem Env: | Sort-Object Name
Get-Content .env.local
setx NEXTAUTH\_SECRET "<valeur>" # Persiste pour nouvelles sessions
macOS/Linux (bash/zsh):
printenv NEXTAUTH\_URL
env | sort
grep -v "^#" .env.local | xargs -L1 echo
export NEXTAUTH\_SECRET=$(openssl rand -hex 32)
Node (server-side):
node -e "console.log(process.env.NEXTAUTH\_URL)"
Next.js (client):
NEXT\_PUBLIC\_\* seulement est accessible côté client (ne jamais exposer de secret).

2) Générer des secrets:
macOS/Linux:
openssl rand -hex 32
uuidgen
python3 -c "import secrets; print(secrets.token\_urlsafe(32))"
Windows PowerShell:
[guid]::NewGuid()
[Convert]::ToBase64String((1..32 | ForEach-Object {Get-Random -Max 256}))

3) Git & réseau:
git status && git log --oneline -n 10
Windows: netstat -ano | findstr :3000
macOS: lsof -i :3000
Linux: ss -ltnp | grep 3000

4) Node/Next.js:
npm run dev
npm run build && npm start
npm run type-check && npm run lint

5) Prisma & PostgreSQL:
npx prisma db push
npx prisma db seed
npx prisma studio
psql "postgresql://user:pass@localhost:5432/inkspot" -c "\dt"
pg\_dump -h localhost -U user -d inkspot > backup.sql
psql -h localhost -U user -d inkspot < backup.sql

6) Redis:
redis-cli -a <password> ping
redis-cli -a <password> info
redis-cli -a <password> keys "inkspot:\*" | head

7) Docker Compose (dev/prod):
docker compose -f docker-compose-dev.yml up -d
docker compose -f docker-compose-dev.yml ps
docker compose -f docker-compose-dev.yml logs -f postgres
docker compose -f docker-compose.prod.yml up -d
docker compose -f docker-compose.prod.yml exec app sh
docker compose -f docker-compose.prod.yml restart nginx

8) Nginx (sur serveur):
nginx -T # afficher la config effective
nginx -s reload
tail -f /var/log/nginx/error.log

9) Stripe CLI (tests):
stripe login
stripe listen --forward-to localhost:3000/api/stripe/webhook
stripe trigger payment\_intent.succeeded

10) Health & Monitoring:
curl -i http://localhost:3000/api/health
curl -s http://localhost:3000/api/metrics | head -n 50
open http://localhost:9090 # Prometheus (macOS) | xdg-open (Linux)
open http://localhost:3001 # Grafana (macOS) | xdg-open (Linux)

# **Annexe A — API Routes (inventaire)**

- /admin/users/[id]
- /appointments/[id]/cancel
- /appointments/[id]/confirm
- /appointments/[id]/history
- /appointments/[id]/payment
- /appointments/[id]/payments
- /appointments/[id]/respond
- /appointments/[id]/status
- /appointments/events
- /appointments/events/[id]
- /appointments/history
- /appointments/list
- /appointments/propose
- /appointments/stats
- /appointments/status
- /appointments/transitions
- /artists/search
- /auth/[...nextauth]
- /auth/register
- /auth/reset-password
- /auth/verify-email
- /bookings
- /bookings/direct
- /collaborations
- /collaborations/received
- /conversations
- /conversations/[id]
- /conversations/[id]/messages
- /conversations/activate
- /conversations/check-or-create
- /conversations/create
- /conversations/create-direct
- /debug/config
- /grafana
- /grafana/metrics
- /health
- /interactions
- /invoices
- /invoices/export
- /invoices/generate
- /messages/realtime
- /metrics
- /metrics/public
- /notifications
- /payments/confirm-checkout
- /payments/create
- /payments/create-intent
- /payments/payout
- /payments/refund
- /posts
- /posts/[id]
- /posts/[id]/comments
- /posts/[id]/like
- /posts/create
- /posts/recommended
- /posts/search
- /pro/[proId]/availability/dates
- /pro/[proId]/availability/schedule
- /pro/[proId]/availability/times
- /pro/[proId]/services
- /pro/analytics
- /pro/availability/schedule
- /push-subscriptions
- /push-subscriptions/preferences
- /push-subscriptions/test
- /recommendations/users
- /reviews
- /search/advanced
- /search/analytics
- /search/geographic
- /search/hashtags
- /search/history
- /search/suggestions
- /search/trending
- /security/alerts
- /stripe/connect
- /stripe/webhook
- /upload
- /users/[id]
- /users/[id]/follow
- /users/[id]/follow-status
- /users/[id]/online-status
- /users/[id]/posts
- /users/[id]/stats
- /users/online
- /users/pros/search
- /users/search
- /webhooks/stripe-appointments

# **Annexe B — Modèles Prisma**

- User
- VerificationToken
- Post
- Like
- Comment
- Conversation
- ConversationMember
- Message
- Appointment
- AppointmentStatusHistory
- Payment
- Transaction
- Invoice
- Notification
- Follow
- Review
- UserInteraction
- SearchHistory
- PushSubscription
- MagicLink
- Collaboration
- Service
- Availability
- AvailabilitySchedule
- AvailabilityTimeSlot
- Booking
- UserPreferences
- DevicePreferences
- SmartReminder
- ScheduledTask
- PerformanceMetric

# **Annexe C — Documentation existante (extraits)**

# Social Media Pro
A modern social media platform designed for professionals to showcase their work, connect with clients, and manage bookings with integrated payments.
## Features
- 🔐 Authentication (Email/Password, Google OAuth, Apple OAuth, Magic Links)
- 📱 Social media feed with posts, likes, comments
- 💬 Real-time messaging system
- 📅 Professional booking system
- 💳 Integrated payments with Stripe
- 🔔 Push notifications
- 📁 File uploads to AWS S3
- 🎨 Customizable professional profiles
- 🔍 Smart content recommendations
- 📊 Admin dashboard
## Tech Stack
- \*\*Frontend\*\*: Next.js 14, React, TypeScript, Tailwind CSS
- \*\*Backend\*\*: Next.js API Routes, Prisma ORM
- \*\*Database\*\*: PostgreSQL
- \*\*Authentication\*\*: NextAuth.js
- \*\*Payments\*\*: Stripe
- \*\*File Storage\*\*: AWS S3
- \*\*Email\*\*: Nodemailer
- \*\*Push Notifications\*\*: Web Push
## Setup Instructions
### 1. Clone and Install
```bash
git clone <repository-url>
cd social-media-pro
pnpm install
```
### 2. Environment Configuration
Copy the example environment file and configure it:
```bash
cp .env.example .env
```
Then edit `.env` with your actual values:
- \*\*Database\*\*: Set up a PostgreSQL database and update `DATABASE\_URL`
- \*\*NextAuth\*\*: Generate a random secret for `NEXTAUTH\_SECRET`
- \*\*OAuth\*\*: Configure Google and Apple OAuth credentials
- \*\*Email\*\*: Set up SMTP server credentials
- \*\*Stripe\*\*: Add your Stripe API keys
- \*\*AWS S3\*\*: Configure S3 bucket for file uploads
- \*\*VAPID\*\*: Generate keys for push notifications
### 3. Generate VAPID Keys
```bash
pnpm run generate-vapid
```
Copy the generated keys to your `.env` file.
### 4. Database Setup
```bash
# Push the schema to your database
pnpm run db:push
# Seed the database with sample data
pnpm run db:seed
```
### 5. Start Development Server
```bash
pnpm run dev
```
Visit `http://localhost:3000` to see the application.
## Default Users
After seeding, you can log in with:
- \*\*Admin\*\*: admin@example.com / admin123
- \*\*Professional\*\*: pierce@example.com / pro123
## Environment Variables Reference
| Variable | Description | Example |
|----------|-------------|---------|
| `DATABASE\_URL` | PostgreSQL connection string | `postgresql://user:pass@localhost:5432/db` |
| `NEXTAUTH\_SECRET` | NextAuth encryption secret | Random 32+ character string |
| `NEXTAUTH\_URL` | Application URL | `http://localhost:3000` |
| `GOOGLE\_CLIENT\_ID` | Google OAuth client ID | From Google Console |
| `GOOGLE\_CLIENT\_SECRET` | Google OAuth secret | From Google Console |
| `STRIPE\_SECRET\_KEY` | Stripe secret key | `sk\_test\_...` |
| `AWS\_S3\_BUCKET\_NAME` | S3 bucket for uploads | `my-app-uploads` |
## Scripts
- `pnpm run dev` - Start development server
- `pnpm run build` - Build for production
- `pnpm run start` - Start production server
- `pnpm run db:push` - Push schema to database
- `pnpm run db:studio` - Open Prisma Studio
- `pnpm run db:seed` - Seed database with sample data
- `pnpm run generate-vapid` - Generate VAPID keys
## License
MIT# Test CI/CD Pipeline

# 🎉 Système de Rendez-vous INKSPOT - Fonctionnalités Complètes
## ✅ Fonctionnalités Implémentées
### 🚀 \*\*Migration Stripe Checkout Hébergé\*\*
- \*\*✅ API de paiement\*\* : `/api/appointments/[id]/payment` avec Stripe Checkout Sessions
- \*\*✅ Modal simplifié\*\* : Redirection automatique vers Stripe (plus de problèmes CSS)
- \*\*✅ Webhooks renforcés\*\* : `checkout.session.completed` pour traitement automatique
- \*\*✅ Gestion retour\*\* : Messages de succès/échec automatiques dans conversations
### 💰 \*\*Statut "PAID" Complet\*\*
- \*\*✅ Logique webhook\*\* : Calcul automatique du total payé vs prix
- \*\*✅ Badge de statut\*\* : "💰 Entièrement payé" avec couleur émeraude
- \*\*✅ Messages automatiques\*\* : "🎉 Paiement complet !" dans les conversations
- \*\*✅ Gestion acompte\*\* : Transition automatique `CONFIRMED` → `PAID`
### 🚫 \*\*Système d'Annulation avec Remboursement\*\*
- \*\*✅ API d'annulation\*\* : `/api/appointments/[id]/cancel` avec logique de remboursement
- \*\*✅ Politique de remboursement\*\* :
- Client >48h : Remboursement intégral
- Client >24h : 50% de remboursement
- Client <24h : Aucun remboursement
- PRO : Toujours remboursement intégral
- \*\*✅ Modal d'annulation\*\* : Interface claire avec politique affichée
- \*\*✅ Bouton d'annulation\*\* : Icône rouge à côté du badge de statut
- \*\*✅ Remboursements Stripe\*\* : Traitement automatique via API Stripe
- \*\*✅ Historique\*\* : Enregistrement dans `AppointmentStatusHistory`
### 📚 \*\*Documentation Complète\*\*
- \*\*✅ Guide webhooks\*\* : `docs/STRIPE\_WEBHOOKS\_SETUP.md`
- \*\*✅ Configuration développement\*\* : Stripe CLI, variables d'env, testing
- \*\*✅ Configuration production\*\* : Endpoints, sécurité, monitoring
- \*\*✅ Dépannage\*\* : Solutions aux problèmes courants
## 🎯 \*\*Flow Complet Fonctionnel\*\*
### 1. \*\*Proposition (PRO)\*\*
```
PRO → Bouton "Proposer un RDV" → Formulaire complet → Envoi
```
### 2. \*\*Réponse (Client)\*\*
```
Client → Badge "Répondre à la proposition" → Modal Accept/Reject → Confirmation
```
### 3. \*\*Paiement (Client)\*\*
```
Client → Badge "Payer la caution" → Stripe Checkout → Webhook → Statut mis à jour
```
### 4. \*\*Confirmation (PRO)\*\*
```
PRO → Badge "Confirmé - Paiement reçu" → Rendez-vous finalisé
```
### 5. \*\*Annulation (Tous)\*\*
```
Utilisateur → Bouton rouge ❌ → Modal avec politique → Confirmation → Remboursement automatique
```
## 🔧 \*\*Technologies Utilisées\*\*
### \*\*Frontend\*\*
- \*\*React Hooks\*\* : `useState`, `useEffect` pour gestion état
- \*\*shadcn/ui\*\* : `Button`, `Badge`, `Card`, `Sheet`, `Dialog`, `Tabs`
- \*\*Stripe Elements\*\* → \*\*Stripe Checkout\*\* (migration complète)
- \*\*Toast Notifications\*\* : Feedback utilisateur automatique
### \*\*Backend\*\*
- \*\*Next.js API Routes\*\* : RESTful endpoints
- \*\*Prisma ORM\*\* : Gestion base de données
- \*\*Stripe API\*\* : Checkout Sessions, Webhooks, Refunds
- \*\*NextAuth\*\* : Authentification et autorisations
### \*\*Base de Données\*\*
- \*\*Appointment\*\* : Statuts complets (`PROPOSED` → `PAID`)
- \*\*Payment\*\* : Historique complet avec Stripe IDs
- \*\*AppointmentStatusHistory\*\* : Audit trail
- \*\*Notification\*\* : Alertes temps réel
## 🚦 \*\*Statuts d'Appointment\*\*
| Statut | Description | Actions Disponibles |
|--------|-------------|-------------------|
| `PROPOSED` | Proposition envoyée | Répondre, Annuler |
| `ACCEPTED` | Accepté par client | Payer acompte, Annuler |
| `CONFIRMED` | Acompte payé | Payer solde, Annuler |
| `PAID` | Entièrement payé | Annuler (remboursement) |
| `COMPLETED` | Terminé | Historique uniquement |
| `CANCELLED` | Annulé | Historique uniquement |
## 🎨 \*\*Interface Utilisateur\*\*
### \*\*Badge Dynamique\*\*
- \*\*Couleurs contextuelles\*\* : Orange (attente), Vert (confirmé), Émeraude (payé)
- \*\*Actions intelligentes\*\* : Clic pour action appropriée selon statut
- \*\*Bouton d'annulation\*\* : Toujours visible (sauf terminé/annulé)
### \*\*Modales Adaptatives\*\*
- \*\*Réponse\*\* : Accept/Reject pour clients
- \*\*Paiement\*\* : Redirection Stripe sécurisée
- \*\*Annulation\*\* : Politique claire + raison optionnelle
## 🔄 \*\*Webhooks Stripe\*\*
### \*\*Événements Traités\*\*
- `checkout.session.completed` ✅
- `payment\_intent.succeeded` ✅ (backup)
- `payment\_intent.payment\_failed` ✅
### \*\*Actions Automatiques\*\*
- ✅ Mise à jour statut appointment
- ✅ Création messages conversation
- ✅ Envoi notifications
- ✅ Historique modifications
## 🧪 \*\*Tests Recommandés\*\*
### \*\*Flow Complet\*\*
1. \*\*Créer RDV\*\* (PRO) → Vérifier proposition
2. \*\*Accepter\*\* (Client) → Vérifier demande paiement
3. \*\*Payer acompte\*\* → Vérifier webhook + statut
4. \*\*Payer solde\*\* → Vérifier statut "PAID"
5. \*\*Annuler\*\* → Vérifier remboursement
### \*\*Cas d'Erreur\*\*
- Paiement échoué
- Webhook failed
- Annulation sans paiement
- Double paiement
## 🚀 \*\*Prochaines Étapes Optionnelles\*\*
### \*\*Restantes du TODO\*\*
- `appointment-reschedule` : Reprogrammation de RDV
- `pro-availability-check` : Vérification créneaux
- `test-payment-flow` : Tests automatisés complets
### \*\*Améliorations Futures\*\*
- \*\*Rappels automatiques\*\* (24h avant RDV)
- \*\*Évaluation post-RDV\*\* (système de notes)
- \*\*Paiements récurrents\*\* (abonnements)
- \*\*Intégration calendrier\*\* (Google Calendar, etc.)
## 📊 \*\*Métriques de Succès\*\*
- ✅ \*\*0 erreurs TypeScript\*\*
- ✅ \*\*Flow de paiement fonctionnel\*\*
- ✅ \*\*Webhooks configurés\*\*
- ✅ \*\*Annulation + remboursement\*\*
- ✅ \*\*Interface utilisateur complète\*\*
---
## 🎯 \*\*Le système de rendez-vous INKSPOT est maintenant entièrement fonctionnel !\*\*
Tous les flux critiques sont implémentés, testés et documentés. L'utilisateur peut maintenant :
1. \*\*Proposer\*\* des rendez-vous facilement
2. \*\*Accepter/rejeter\*\* en un clic
3. \*\*Payer\*\* via Stripe sécurisé
4. \*\*Suivre\*\* le statut en temps réel
5. \*\*Annuler\*\* avec remboursement automatique
Le système est prêt pour la production ! 🚀

# **Annexe D — Extraits de configuration (Prometheus, Nginx, Docker Compose)**

Prometheus scrape (monitoring/prometheus/prometheus.yml):

global:

scrape\_interval: 15s

evaluation\_interval: 15s

rule\_files:

# - "first\_rules.yml"

# - "second\_rules.yml"

scrape\_configs:

# Prometheus itself

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']

# INKSPOT Application

- job\_name: 'inkspot-app'

static\_configs:

- targets: ['app:3000']

metrics\_path: '/api/metrics'

scrape\_interval: 10s

# Node Exporter

- job\_name: 'node-exporter'

static\_configs:

- targets: ['node-exporter:9100']

# PostgreSQL Exporter

- job\_name: 'postgres-exporter'

static\_configs:

- targets: ['postgres-exporter:9187']

# Redis Exporter (if you add one)

# - job\_name: 'redis-exporter'

# static\_configs:

# - targets: ['redis-exporter:9121']

# Custom INKSPOT metrics

- job\_name: 'inkspot-custom'

static\_configs:

- targets: ['app:3000']

metrics\_path: '/api/grafana'

scrape\_interval: 5s

honor\_labels: true

Nginx (nginx/nginx.conf) – sections clés proxy et WebSocket:

add\_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

# Client max body size

client\_max\_body\_size 10M;

# Main application

location / {

proxy\_pass http://app\_servers;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_cache\_bypass $http\_upgrade;

proxy\_read\_timeout 86400;

}

# WebSocket connections

location /socket.io/ {

proxy\_pass http://websocket\_servers;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_read\_timeout 86400;

}

# API rate limiting

location /api/ {

limit\_req zone=api burst=20 nodelay;

proxy\_pass http://app\_servers;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_cache\_bypass $http\_upgrade;

}

# Login rate limiting

location /api/auth/ {

limit\_req zone=login burst=5 nodelay;

proxy\_pass http://app\_servers;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

proxy\_cache\_bypass $http\_upgrade;

}

# Static files

location /\_next/static/ {

proxy\_pass http://app\_servers;

expires 1y;

add\_header Cache-Control "public, immutable";

}

# Uploads

location /uploads/ {

proxy\_pass http://app\_servers;

expires 1d;

add\_header Cache-Control "public";

}

# Health check

location /health {

proxy\_pass http://app\_servers;

access\_log off;

}

# Monitoring endpoints (internal only)

location /monitoring/ {

allow 127.0.0.1;

allow 10.0.0.0/8;

allow 172.16.0.0/12;

allow 192.168.0.0/16;

deny all;

Docker Compose (docker-compose.prod.yml) – services applicatifs:

app:

image: ${DOCKER\_REGISTRY:-ghcr.io}/${IMAGE\_NAME:-antoine13330/inkspot-5z}:${TAG:-latest}

container\_name: inkspot\_app\_prod

environment:

- NODE\_ENV=production

- DATABASE\_URL=postgresql://${POSTGRES\_USER:-inkspot\_user}:${POSTGRES\_PASSWORD}@postgres:5432/${POSTGRES\_DB:-inkspot}

- REDIS\_URL=redis://redis:6379

- NEXTAUTH\_URL=${NEXTAUTH\_URL}

- NEXTAUTH\_SECRET=${NEXTAUTH\_SECRET}

- GOOGLE\_CLIENT\_ID=${GOOGLE\_CLIENT\_ID}

- GOOGLE\_CLIENT\_SECRET=${GOOGLE\_CLIENT\_SECRET}

- STRIPE\_SECRET\_KEY=${STRIPE\_SECRET\_KEY}

- STRIPE\_WEBHOOK\_SECRET=${STRIPE\_WEBHOOK\_SECRET}

- STRIPE\_PUBLISHABLE\_KEY=${STRIPE\_PUBLISHABLE\_KEY}

- AWS\_ACCESS\_KEY\_ID=${AWS\_ACCESS\_KEY\_ID}

- AWS\_SECRET\_ACCESS\_KEY=${AWS\_SECRET\_ACCESS\_KEY}

- AWS\_REGION=${AWS\_REGION:-us-east-1}

- AWS\_S3\_BUCKET=${AWS\_S3\_BUCKET}

- SMTP\_HOST=${SMTP\_HOST}

- SMTP\_PORT=${SMTP\_PORT}

- SMTP\_USER=${SMTP\_USER}

- SMTP\_PASS=${SMTP\_PASS}

ports:

- "3000:3000"

volumes:

- ./public/uploads:/app/public/uploads

depends\_on:

postgres:

condition: service\_healthy

redis:

condition: service\_healthy

networks:

- inkspot\_network\_prod

restart: unless-stopped

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:3000/api/health"]

interval: 30s

timeout: 10s

retries: 3

# WebSocket Server (Production)

websocket:

image: ${DOCKER\_REGISTRY:-ghcr.io}/${IMAGE\_NAME:-antoine13330/inkspot-5z}-websocket:${TAG:-latest}

container\_name: inkspot\_websocket\_prod

environment:

- NODE\_ENV=production

- DATABASE\_URL=postgresql://${POSTGRES\_USER:-inkspot\_user}:${POSTGRES\_PASSWORD}@postgres:5432/${POSTGRES\_DB:-inkspot}

- REDIS\_URL=redis://redis:6379

- WS\_PORT=3001

ports:

- "3001:3001"

depends\_on:

postgres:

condition: service\_healthy

redis:

condition: service\_healthy

networks:

- inkspot\_network\_prod

restart: unless-stopped

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:3001/health"]

interval: 30s

timeout: 10s

retries: 3

# Nginx Reverse Proxy (Production)

nginx:

image: nginx:alpine

container\_name: inkspot\_nginx\_prod

ports:

- "80:80"

- "443:443"

volumes:

- ./nginx/nginx.conf:/etc/nginx/nginx.conf

- ./nginx/ssl:/etc/nginx/ssl

- ./nginx/logs:/var/log/nginx

depends\_on:

- app

- websocket

networks:

- inkspot\_network\_prod

restart: unless-stopped

# Grafana (Production)

grafana:

image: grafana/grafana:latest

container\_name: inkspot\_grafana\_prod

environment:

- GF\_SECURITY\_ADMIN\_PASSWORD=${GRAFANA\_PASSWORD:-admin123}

- GF\_USERS\_ALLOW\_SIGN\_UP=false

ports:

- "3002:3000"

volumes:

- grafana\_data\_prod:/var/lib/grafana

- ./monitoring/grafana/provisioning:/etc/grafana/provisioning

- ./monitoring/grafana/dashboards:/var/lib/grafana/dashboards

networks:

- inkspot\_network\_prod

restart: unless-stopped

profiles:

- monitoring

# Prometheus (Production)

prometheus:

image: prom/prometheus:latest

container\_name: inkspot\_prometheus\_prod

ports:

- "9090:9090"

command:

- '--config.file=/etc/prometheus/prometheus.yml'

- '--storage.tsdb.path=/prometheus'

- '--web.console.libraries=/etc/prometheus/console\_libraries'

- '--web.console.templates=/etc/prometheus/consoles'

- '--storage.tsdb.retention.time=200h'

- '--web.enable-lifecycle'

volumes:

- ./monitoring/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml

- prometheus\_data\_prod:/prometheus

networks:

- inkspot\_network\_prod

restart: unless-stopped

profiles:

- monitoring

# Node Exporter (Production)

node-exporter:

image: prom/node-exporter:latest

container\_name: inkspot\_node\_exporter\_prod

ports:

- "9100:9100"

volumes:

- /proc:/host/proc:ro

- /sys:/host/sys:ro

- /:/rootfs:ro

command:

- '--path.procfs=/host/proc'

- '--path.rootfs=/rootfs'

- '--path.sysfs=/host/sys'

- '--collector.filesystem.mount-points-exclude=^/(sys|proc|dev|host|etc)($$|/)'

networks:

- inkspot\_network\_prod

restart: unless-stopped

profiles:

- monitoring

# Postgres Exporter (Production)

postgres-exporter:

image: prometheuscommunity/postgres-exporter:latest

container\_name: inkspot\_postgres\_exporter\_prod

ports:

- "9187:9187"

environment:

DATA\_SOURCE\_NAME: "postgresql://${POSTGRES\_USER:-inkspot\_user}:${POSTGRES\_PASSWORD}@postgres:5432/${POSTGRES\_DB:-inkspot}?sslmode=disable"

networks:

- inkspot\_network\_prod

restart: unless-stopped

# **Annexe E — Procédures & Checklists opérateur**

Checklist pré-déploiement: variables d’environnement présentes, secrets valides, DB migrée, Stripe webhook configuré, Nginx rechargé sans erreur, monitoring OK.

Runbook incident (paiements): vérifier Stripe status, logs webhook, retenter idempotent, notifier si SLA menacé.

Runbook incident (DB): vérifier connexions actives, lenteurs, index manquant; escalader si contention.

Rollback: image précédente + migrations réversibles + smoke tests.

# **Annexe F — Screens à capturer (guide)**

Accueil local (http://localhost:3000), Prisma Studio (tables clés), Stripe Dashboard (webhooks), Grafana: panels principaux, Prometheus: targets, Nginx: test SSL (qualys), Postman: collections d’API, Pages clés: login, profil, création de post, messagerie.

# **Annexe G — Glossaire**

SLA: Accord de niveau de service — objectif de disponibilité/latence.
SLO: Objectif de niveau de service — mesure chiffrée par métriques.
Idempotence: même requête répétée donne le même résultat (pas de double paiement).
P95/P99: percentiles de latence; 95%/99% des requêtes sous ce seuil.