UPRISING NODE

AI Wiring Manual — Connecter les écrans & les systèmes

Version: v1.0 • Date: 2026-01-25

Mission: donner à un agent IA une procédure standard pour créer l'application, connecter chaque écran à la donnée (DB/API), au temps réel (Socket.IO), et aux jobs (BullMQ/IA) — sans improvisation.

SOMMAIRE

0. Règles d'exécution (Agent IA)

Un agent IA doit suivre cet ordre. Aucun écran n'est 'terminé' tant que son contrat n'est pas respecté.

- **Installer** toutes les dépendances (Node + Python) et démarrer l'infra Docker (Postgres/Redis) avant d'écrire du code applicatif.
- **Créer** d'abord les schémas (Prisma) + migrations, ensuite l'API, ensuite l'UI.
- **Écrire** les contrats de données (DTO/Zod/Pydantic) avant de connecter l'écran.
- **Refuser** tout état 'magic': chaque statut doit exister en DB, et chaque événement temps réel doit être idempotent.
- **Tracer** chaque flow avec traceId/leadId dans les logs.

```
# Checklist boot (exemple)
docker compose up -d
pnpm install
pnpm --filter api prisma:generate
pnpm --filter api prisma:migrate
cd apps/ai && python -m venv .venv && source .venv/bin/activate && pip install -r
requirements.txt
pnpm dev
```

1. Carte des systèmes

Le Node est un flux. Chaque bloc produit un artefact persistant (DB/Storage) et un signal temps réel (Socket.IO).

- **Web (Next.js)**: écrans + actions utilisateur + abonnements temps réel.
- **API (NestJS)**: source de vérité (CRUD), RBAC, validation, émission d'événements.
- **DB (Postgres/Prisma)**: statuts, ledger, historique immuable.
- **Queue (BullMQ/Redis)**: jobs asynchrones, retries, SLA.
- **AI (FastAPI)**: scraping + analyse structurée + génération contenu (JSON/PDF).
- **Realtime (Socket.IO)**: rooms, présence, pipeline updates, chat.

```
User Action (Web)
-> REST (API) create/update DB
-> enqueue jobs (BullMQ)
-> worker/AI produces artifacts (analysis, pdf)
-> API updates DB (READY)
-> Socket.IO broadcasts events to rooms
-> Web updates UI instantly
```

2. Template 'Screen Contract' (obligatoire)

Avant de coder un écran, l'agent IA remplit ce contrat. Zéro ambiguïté.

Champ	Contenu requis (exemple)
Route Next.js	/dashboard/leads (app/dashboard/leads/page.tsx)
UI Components	LeadDropForm, DataCard, EmptyState
RBAC	Partenaire: read/create • Opérateur: update status
Data source	GET /leads, POST /leads
Realtime	subscribe room:lead:{leadId} • events: lead.job.status, lead.updated
DB entities	Lead, PipelineEvent, LeadAnalysis, AuditPdf
Jobs	scrape_site → extract_text → ai_analysis → generate_pdf
Errors	422 validation, 429 rate limit, 500 retryable
Tests	E2E: login → action → UI updated; unit: reducers/validators
Telemetry	traceId, leadId, timing, job durations

3. Standards Next.js (App Router)

- Chaque écran = un segment /app/.../page.tsx.
- Appels data: Server Components (fetch côté serveur) ou Client + fetch (si besoin d'interaction).
- Pour API interne: privilégier l'API NestJS comme backend principal (éviter double backend).
- Les formulaires déclenchent POST vers l'API, puis l'écran se synchronise via Socket.IO (source: events).
- Contrats front: Zod schemas partagés (/packages/shared).

```
Pattern recommandé (UI):
- server page fetch initial list (GET)
- client component handles create (POST) + optimistic UI optionnel
- socket subscription merges updates -> state store
```

4. Standards API NestJS (REST + RBAC)

- Chaque ressource a un module: leads.module, chat.module, payouts.module.
- Validation entrée: DTO + pipes (ou Zod) => 422 si invalide.
- RBAC: Guard par rôle + ownership (ownerId / membership workspace).
- Événements: après transaction DB, broadcast Socket.IO (idempotent).
- Aucune mutation sur LedgerEntry (append-only).

```
Convention endpoints (MVP)
POST /leads
PATCH /leads/:id (status op-only)
GET /leads/:id
POST /leads/:id/requeue
GET /channels/:id/messages
POST /channels/:id/messages
```

5. Standard Realtime (Socket.IO)

- Rooms: workspace, project, lead, channel.
- Auth socket: token session/JWT au handshake.
- Events doivent être idempotents: inclure version/updatedAt.
- Presence/typing = éphémère (pas DB). Messages = persistés (DB).

```
Rooms:
room:workspace:{workspaceId}
room:project:{projectId}
room:lead:{leadId}
room:channel:{channelId}

Core events:
lead.updated
lead.job.status
pipeline.moved
chat.message.new
presence.update
typing.start/stop
```

6. Standard Jobs (BullMQ) — orchestration Lead Drop → War Room

Objectif: découper en jobs courts, observables, retryables. Les jobs écrivent en DB, l'API diffuse en temps réel.

- Producer: API (POST /leads) enfile job:scrape_site (leadId).
- Worker: exécute scraping, écrit LeadEnrichment, push job:ai_analysis.
- AI service: renvoie JSON structuré; worker écrit LeadAnalysis.
- Worker: génère PDF (template) => storageKey => AuditPdf version 1.
- API marque le lead READY et émet lead.updated + lead.job.status.

```
Job graph (minimal):
scrape_site -> extract_text -> ai_analysis -> generate_pdf -> notify
```

7. Connexion écran par écran (MVP)

Chaque sous-section ci-dessous = un Screen Contract exécutable.

7.1 Lead Drop (app/dashboard/lead-drop)

- UI: champ URL/Nom + bouton 'Démarrer analyse'.
- POST /leads crée Lead(status=NEW) + enqueue scrape_site.
- Socket: écouter lead.job.status pour afficher progression.
- Rediriger vers /dashboard/war-room/[leadId] dès que READY.

```
Flow:
Submit -> POST /leads -> receive {leadId}
Join room:lead:{leadId}
Render progress from lead.job.status
On lead.updated(status=READY) -> navigate to war room
```

7.2 Pipeline Kanban (app/dashboard/pipeline)

- GET /leads (filtré workspace) pour initial state.
- Drag&drop; (opérateur): PATCH /leads/:id {status}.
- API écrit PipelineEvent + update Lead.status, puis emits pipeline.moved.
- UI: applique update reçu via socket (source de vérité).

7.3 War Room (app/dashboard/war-room/[leadId])

- GET /leads/:id + analyses + pdf versions.
- Afficher: douleurs, angles, scripts, timeline (PipelineEvent).
- Bouton 'Requeue' (opérateur): POST /leads/:id/requeue.
- Bouton 'Générer PDF': POST /leads/:id/pdf -> nouvelle version.

7.4 Gains (app/dashboard/gains)

- GET /ledger (agrégé côté API) + GET /payouts.
- Socket: payout.status pour état temps réel.
- Cash-out: POST /payouts (montant) => status Requested.

7.5 Collaboration & Chat (app/dashboard/chat/[channelId])

- GET /channels/:id/messages (pagination).
- POST /channels/:id/messages (persist).
- Socket: join room:channel:{channelId}; event chat.message.new.
- Presence: join room + presence.update; typing.start/stop.

8. Checklists livraison

Un écran est livré quand la checklist est verte.

8.1 Checklist écran

- Screen Contract rempli et commit dans /docs/contracts.
- UI conforme Lindy (noir/blanc, 1px, espaces).
- Toutes les écritures passent par l'API (pas d'accès DB direct côté web).
- Abonnement Socket.IO en place + cleanup (disconnect) sur unmount.
- Gestion erreurs (422/429/500) + états vides + loading.
- Logs: traceId sur requêtes; events auditables.

8.2 Checklist flow Lead Drop

- Lead créé (DB) -> jobs en queue -> analysis JSON -> pdf -> READY.
- Retry contrôlé; erreurs visibles War Room.
- P95 sous 10 minutes en staging.

8.3 Checklist sécurité

- RBAC: partenaire ne voit que ses leads (ownership).
- Rate limiting sur OTP + lead drop + chat.
- API keys hashées; secrets en .env uniquement.