

LexOrbital Module Template Guide

Complete guide for module development

LexOrbital Core Team

23 novembre 2025

Contents

Guide LexOrbital	4
Table des matières	4
Présentation	4
Organisation de la documentation	4
Usage avec Pandoc	5
Contribuer à la documentation	6
Ressources externes	7
Licence	7
Support	7
Sommaire de la Documentation LexOrbital	8
Vue d'ensemble	8
Fiches techniques	8
Organisation du dépôt <code>docs/</code>	9
Thème visuel	9
Métriques	10
Parcours de lecture recommandés	10
Commandes rapides	10
Ressources complémentaires	11
Checklist de lecture	11
Fiche n°0 : LexOrbital – Vue d'ensemble	13
1. Objectif de la fiche	13
2. Contexte et origine	13
3. Points clés	13
4. Roadmap	14
5. À retenir / Checklist	14
6. Liens connexes	14
Fiche n°1 : Architecture orbitale (Meta-Kernel, anneaux, modules)	15
1. Objectif de la fiche	15
2. Concepts et décisions clés	15
3. Implications techniques	17
4. Checklist de mise en œuvre	17
5. Liens connexes	18
Fiche n°2 : Microservices vs modules plug'n'play	19
1. Objectif de la fiche	19
2. Concepts et décisions clés	19
3. Implications techniques	20
4. Checklist de décision	21
5. À retenir	21

6. Liens connexes	21
Fiche n°3 : Documentation & diagrammes vivants	22
1. Objectif de la fiche	22
2. Concepts et décisions clés	22
3. Implications techniques	23
4. Checklist de mise en œuvre	25
5. À retenir	26
6. Liens connexes	26
Fiche n°4 : Manifestes LexOrbital (module.json, rgpd-manifest.json)	27
1. Objectif de la fiche	27
2. Concepts et décisions clés	27
3. Implications techniques	30
4. Checklist de mise en œuvre	31
5. À retenir	32
6. Liens connexes	32
Fiche n°5 : RGPD by design – patterns techniques	33
1. Objectif de la fiche	33
2. Concepts et décisions clés	33
3. Patterns techniques	33
4. Checklist de mise en œuvre	38
5. À retenir	38
6. Liens connexes	39
Fiche n°6 : Template module (Husky, commits, SemVer)	40
1. Objectif de la fiche	40
2. Concepts et décisions clés	40
3. Implications techniques	41
4. Checklist de mise en œuvre	46
5. À retenir	46
6. Liens connexes	46
Fiche n°7 : Workflow git subtree & scripts d'amarrage	47
1. Objectif de la fiche	47
2. Concepts et décisions clés	47
3. Implications techniques	48
4. Checklist de mise en œuvre	50
5. À retenir	51
6. Liens connexes	51
Fiche n°8 : Modules types (auth, audit, CI, infra...)	52
1. Objectif de la fiche	52
2. Modules par anneau orbital	52
3. Matrice de dépendances	56
4. Checklist de mise en œuvre	57
5. À retenir	57
6. Liens connexes	58
Fiche n°9 : Console de contrôle & design diégétique	59
1. Objectif de la fiche	59
2. Concepts et décisions clés	59
3. Implications techniques	60
4. Checklist de mise en œuvre	65

5. À retenir	66
6. Liens connexes	66

Guide LexOrbital

Guide d'architecture de la station orbitale LexOrbital : Meta-Kernel, anneaux, modules, conformité RGPD et outillage.

Table des matières

0. [LexOrbital – Vue d'ensemble](#)
 1. [Architecture orbitale \(Meta-Kernel, anneaux, modules\)](#)
 2. [Microservices vs modules plug'n'play](#)
 3. [Documentation & diagrammes vivants](#)
 4. [Manifestes LexOrbital \(module.json, rgpd-manifest.json\)](#)
 5. [RGPD by design – patterns techniques](#)
 6. [Template module \(Husky, commits, SemVer\)](#)
 7. [Workflow git subtree & scripts d'amarrage](#)
 8. [Modules types \(auth, audit, CI, infra...\)](#)
 9. [Console de contrôle & design diégétique](#)
-

Présentation

LexOrbital est une station orbitale d'architecture logicielle modulaire conçue pour orchestrer des systèmes juridiques et métiers complexes. Plutôt que de multiplier les microservices autonomes, LexOrbital propose une architecture en **anneaux orbitaux** :

- **Anneau 0 (Meta-Kernel)** : Orchestration centrale, configuration, health checks
- **Anneau 1 (Core Services)** : Auth, Audit, Logs, Notifications (services transverses)
- **Anneau 2 (Business Logic)** : Dossiers, Documents, Workflow, Facturation (métier)
- **Anneau 3 (Intégrations)** : API Gateway, Webhooks, Export, Search (externe)

Chaque module s'amarre dynamiquement à la station via des manifestes déclaratifs (`lexorbital.module.json` + `rgpd-manifest.json`), permettant une conformité RGPD native et une traçabilité complète.

Organisation de la documentation

Fiches techniques (00–09)

Les **10 fiches numérotées** constituent le cœur de la documentation technique :

Fiche	Titre	Contenu
00	Vue d'ensemble	Vision, contexte, roadmap LexOrbital

Fiche	Titre	Contenu
01	Architecture orbitale	Meta-Kernel, anneaux, communication inter-modules
02	Microservices vs modules	Comparaison architecturale, choix techniques
03	Documentation vivante	Génération automatique (Pandoc, Mermaid, Swagger)
04	Manifestes LexOrbital	Format JSON des manifestes techniques et RGPD
05	RGPD by design	Patterns techniques (audit, anonymisation, DSAR)
06	Template module	Structure standard, Husky, Conventional Commits
07	Workflow subtree	Git subtree pour amarrage de modules externes
08	Modules types	Catalogue des modules par anneau
09	Console de contrôle	Design diégétique, interface 3D (Three.js)

Guides pratiques (guides/)

- **Getting Started** : Installation et premier lancement
- **Générer la documentation** : Utilisation de Pandoc et des scripts
- **Créer un module** : Step-by-step avec le template

Références légales (legal/)

- **Sources RGPD** : Articles du RGPD cités dans le projet
- **Licences** : Licences des dépendances tierces

Usage avec Pandoc

Générer un fichier DOCX

```
cd docs
pandoc -s --toc --toc-depth=2 \
    -o LexOrbital_Guide.docx \
    [0-9][0-9]_*.md
```

Générer un fichier HTML

```
cd docs
pandoc -s --toc --toc-depth=2 \
    --template="templates/lexorbital.html" \
    --css="templates/pandoc.css" \
    -o LexOrbital_Guide.html \
    README.md [0-9][0-9]_*.md
```

Générer un fichier PDF (via LaTeX)

```
cd docs
pandoc -s --toc --toc-depth=2 \
```

```
--pdf-engine=xelatex \
-V geometry:margin=1in \
-V documentclass=report \
-V colorlinks=true \
-o LexOrbital_Guide.pdf \
[0-9][0-9]_* .md
```

Note : Nécessite une installation LaTeX (TeX Live, MiKTeX, etc.)

Générer tous les formats (script automatique)

```
cd scripts
./generate-docs.sh
```

Génère automatiquement : - docs/generated/LexOrbital_Guide.html - docs/generated/LexOrbital_Guide.pdf
- docs/generated/LexOrbital_Guide.docx - Diagrammes Mermaid (SVG) - Registre RGPD (JSON + PDF)

Contribuer à la documentation

Ajouter une nouvelle fiche

1. Créer un fichier docs/NN_titre-fiche.md (NN = numéro à 2 chiffres)
2. Utiliser le template d'en-tête :

```
# Fiche n°N : Titre de la fiche {#fiche-N-titre}

> Résumé en 2-3 phrases.

## 1. Objectif de la fiche
## 2. Concepts et décisions clés
## 3. Implications techniques
## 4. Checklist de mise en œuvre
## 5. À retenir
## 6. Liens connexes

3. Ajouter une entrée dans la table des matières du README.md
4. Régénérer la doc : ./scripts/generate-docs.sh
```

Mettre à jour une fiche existante

1. Éditer le fichier docs/NN_*.md concerné
2. Respecter la structure (sections numérotées, IDs explicites)
3. Commit avec Conventional Commits : docs(fiche-N): description
4. Régénérer la doc automatiquement (CI/CD GitHub Actions)

Conventions d'écriture

- IDs explicites : {#fiche-N-titre} pour ancrages stables
 - Liens internes : [[NN_autre-fiche]] (format Obsidian)
 - Code blocks : utiliser les triple backticks avec langage ("typescript")
 - Diagrammes : utiliser Mermaid ou D2 (génération automatique)
 - Emphase : **gras** pour concepts importants, *italique* pour nuances
-

Ressources externes

- Documentation [NestJS](#) - Framework backend
 - React Three Fiber - Rendu 3D de la console
 - Pandoc Manual - Génération de documentation
 - RGPD ([texte intégral](#)) - Référence légale
 - Conventional Commits - Standard de commits
-

Licence

Ce projet est sous licence **MIT**. Voir le fichier [LICENSE](#) à la racine du projet.

Support

Pour toute question ou contribution, consultez :

- [CONTRIBUTING.md](#) - Guide de contribution
 - [CODE_OF_CONDUCT.md](#) - Code de conduite
 - [GitHub Issues](#) - Signaler un bug ou proposer une fonctionnalité
-

Version du guide : 1.0.0

Dernière mise à jour : 22 novembre 2025

Sommaire de la Documentation LexOrbital

Navigation rapide vers toutes les fiches techniques de la station orbitale.

Vue d'ensemble

Ce guide complet couvre l'architecture, les modules, la conformité RGPD et l'outillage de **LexOrbital**, la station orbitale d'architecture logicielle.

Nombre total de fiches : 10

Lignes de documentation : ~2,600

Format : Markdown → HTML/PDF/DOCX via Pandoc

Fiches techniques

Fondations (Fiches 00-02)

#	Titre	Résumé	Lignes
00	Vue d'ensemble	Vision, origine, roadmap LexOrbital	92
01	Architecture orbitale	Meta-Kernel, anneaux (0-3), modules	154
02	Microservices vs modules	Comparaison, choix architectural	129

Documentation & Conformité (Fiches 03-05)

#	Titre	Résumé	Lignes
03	Documentation vivante	Génération auto (Pandoc, Mermaid, Swagger)	239
04	Manifestes	<code>lexorbital.module.json</code> + <code>rgpd-manifest.json</code>	288
05	RGPD by design	Audit immutable, DSAR, anonymisation	329

Outillage & Workflow (Fiches 06-07)

#	Titre	Résumé	Lignes
06	Template module	Husky, Commitlint, tests, SemVer	387
07	Workflow subtree	Git subtree, amarrage de modules	254

Écosystème (Fiches 08-09)

#	Titre	Résumé	Lignes
08	Modules types	Catalogue (auth, audit, dossiers, etc.)	307
09	Console de contrôle	Design diégétique, Three.js, WebSocket	402

Organisation du dépôt docs/

```
docs/
  00_lexorbital-intro.md          # Fiche 0 : Vue d'ensemble
  01_architecture-orbitale.md    # Fiche 1 : Architecture
  02_microservices-vs-modules.md # Fiche 2 : Comparaison
  03_documentation-et-diagrammes-vivants.md # Fiche 3 : Doc vivante
  04_manifestes-lexorbital.md    # Fiche 4 : Manifestes
  05_rgpd-by-design.md          # Fiche 5 : RGPD
  06_template-module.md          # Fiche 6 : Template
  07_workflow-subtree.md         # Fiche 7 : Git subtree
  08_modules-types.md           # Fiche 8 : Catalogue
  09_console-de-controle.md     # Fiche 9 : Console
  README.md                      # Index principal
  QUICKSTART.md                 # Guide de démarrage rapide
templates/
  lexorbital.html                # Template HTML Pandoc
  pandoc.css                     # Styles CSS (thème spatial)
generated/
scripts/
  generate-docs.sh              # Script de génération
```

Thème visuel

Le thème LexOrbital utilise une palette **spatiale** :

- **Couleur primaire** : Bleu profond (#2563eb)
- **Accent** : Cyan/Turquoise (#06b6d4)
- **Fond** : Dégradé sombre (#0f172a → #1e293b)
- **Police monospace** : Fira Code, Monaco
- **Bordures** : Style holographique (cyan + transparence)

Métriques

Métrique	Valeur
Fiches techniques	10
Lignes totales	~2,600
Mots	~18,000
Temps de lecture estimé	~90 minutes
Diagrammes Mermaid	5+
Exemples de code	40+

Parcours de lecture recommandés

Pour les architectes

1. 00 - Vue d'ensemble
2. 01 - Architecture orbitale
3. 02 - Microservices vs modules
4. 08 - Modules types

Pour les développeurs

1. 06 - Template module
2. 04 - Manifestes
3. 07 - Workflow subtree
4. 03 - Documentation vivante

Pour les DPO / Conformité

1. 05 - RGPD by design
2. 04 - Manifestes RGPD
3. 03 - Documentation automatique

Pour les Product Owners

1. 00 - Vue d'ensemble
 2. 09 - Console de contrôle
 3. 08 - Modules types
-

Commandes rapides

Lire la doc

```
# Ouvrir dans Obsidian (recommandé)
open -a Obsidian docs/
```

```
# Lire avec VS Code
code docs/README.md
```

Générer la doc

```
# Tout générer
./scripts/generate-docs.sh

# HTML uniquement
cd docs && pandoc -s --toc \
--template=templates/lexorbital.html \
--css=templates/pandoc.css \
-o generated/guide.html [0-9][0-9]_*.*.md

# PDF uniquement (LaTeX requis)
cd docs && pandoc -s --toc \
--pdf-engine=xelatex \
-o generated/guide.pdf [0-9][0-9]_*.*.md
```

Ouvrir la doc générée

```
open docs/generated/index.html
```

Ressources complémentaires

Documentation existante

- Architecture technique
- Origine de LexOrbital
- Générer la documentation
- Sources légales

Liens externes

- NestJS Documentation
 - React Three Fiber
 - RGPD - Texte intégral
 - Pandoc Manual
-

Checklist de lecture

Cochez au fur et à mesure de votre lecture :

- 00 - Vue d'ensemble
 - 01 - Architecture orbitale
 - 02 - Microservices vs modules
 - 03 - Documentation vivante
 - 04 - Manifestes
 - 05 - RGPD by design
 - 06 - Template module
 - 07 - Workflow subtree
 - 08 - Modules types
 - 09 - Console de contrôle
-

Version : 1.0.0

Dernière mise à jour : 22 novembre 2025

Auteur : LexOrbital Core Team

Bon voyage dans la station orbitale LexOrbital !

Fiche n°0 : LexOrbital – Vue d'ensemble

LexOrbital est une station orbitale d'architecture logicielle modulaire, conçue pour orchestrer des systèmes juridiques et métiers complexes avec conformité RGPD intégrée et traçabilité complète.

1. Objectif de la fiche

Cette fiche présente la vision globale de LexOrbital : son origine, sa philosophie architecturale et sa roadmap. Elle sert de point d'entrée pour comprendre pourquoi et comment LexOrbital se positionne comme alternative aux architectures monolithiques et microservices traditionnelles.

2. Contexte et origine

LexOrbital naît d'un constat : - Les **monolithes** deviennent ingérables à grande échelle - Les **microservices** imposent une complexité opérationnelle importante (orchestration, communication réseau, observabilité) - Les exigences **RGPD** nécessitent une traçabilité fine et native, difficile à retrofitter

Vision : la station orbitale

Plutôt que de multiplier les services autonomes, LexOrbital propose une **architecture orbitale** : - Un **Meta-Kernel** central (noyau de coordination) - Des **anneaux fonctionnels** (couches métier : auth, audit, data, etc.) - Des **modules plug'n'play** qui s'amarrent dynamiquement à la station

Cette métaphore spatiale permet de visualiser l'architecture comme un système gravitationnel où chaque module occupe une orbite définie, avec des responsabilités claires et une communication orchestrée.

3. Points clés

3.1. Architecture modulaire par anneaux

- **Anneau 0 (Meta-Kernel)** : Orchestration, configuration, health checks
- **Anneau 1 (Core Services)** : Auth, RBAC, Audit, Logging
- **Anneau 2 (Business Logic)** : Modules métier (dossiers, documents, workflow)
- **Anneau 3 (Intégrations)** : APIs externes, webhooks, notifications

3.2. Manifestes déclaratifs

Chaque module expose un `lexorbital.module.json` décrivant : - Ses dépendances - Ses points d'entrée (API, événements) - Sa conformité RGPD (`rgpd-manifest.json`)

3.3. Console de contrôle diégétique

Interface inspirée des consoles de station spatiale pour : - Visualiser l'état des modules (statut, santé, orbite)
- Gérer les amarrages/désamarrages - Auditer les flux de données et événements

3.4. RGPD by design

- Traçabilité native (qui accède à quoi, quand, pourquoi)
- Durées de rétention configurables par module
- Export/suppression GDPR automatisés

4. Roadmap

Phase 1 : Fondations (Q1 2025)

- Meta-Kernel (NestJS) avec système de plugins
- Template de module standard
- Module Auth (JWT + RBAC)
- Module Audit (event sourcing)

Phase 2 : Modules métier (Q2 2025)

- Module Dossiers juridiques
- Module Documents (versionning, signatures)
- Console de contrôle (React + Three.js pour visualisation 3D)

Phase 3 : Conformité avancée (Q3 2025)

- Manifeste RGPD automatisé
- Tableaux de flux de données
- Exports conformité (PDF, JSON)

Phase 4 : Écosystème (Q4 2025)

- Marketplace de modules
- CLI d'amarrage de modules tiers
- Documentation interactive

5. À retenir / Checklist

- LexOrbital = **station orbitale** (Meta-Kernel + anneaux + modules)
- Plug'n'play** : modules s'amarrent/désamarrent sans redéploiement complet
- RGPD native** : conformité intégrée dès la conception
- Console diégétique** : UX inspirée des stations spatiales
- Manifestes déclaratifs** : `lexorbital.module.json` pour chaque module

6. Liens connexes

- [[01_architecture-orbitale]] : Détails techniques du Meta-Kernel et des anneaux
- [[04_manifestes-lexorbital]] : Spécifications des fichiers de manifeste
- [[09_console-de-controle]] : Design et UX de la console de contrôle
- [[06_template-module]] : Comment créer un nouveau module LexOrbital

Fiche n°1 : Architecture orbitale (Meta-Kernel, anneaux, modules)

L'architecture orbitale de LexOrbital organise les composants en **anneaux concentriques** autour d'un **Meta-Kernel** central, permettant une orchestration cohérente tout en garantissant l'isolation et la modularité.

1. Objectif de la fiche

Décrire en détail l'organisation en couches (anneaux) de LexOrbital, le rôle du Meta-Kernel et les principes de communication entre modules. Cette fiche est la référence technique pour comprendre comment les modules s'intègrent à la station.

2. Concepts et décisions clés

2.1. Le Meta-Kernel : cerveau de la station

Le **Meta-Kernel** est le noyau central de LexOrbital. Il assure :

Orchestration des modules

- Découverte automatique des modules (via manifestes)
- Gestion du cycle de vie : initialisation, health checks, graceful shutdown
- Injection de dépendances inter-modules

Configuration centralisée

- Variables d'environnement par anneau/module
- Secrets management (intégration Vault/K8s secrets)
- Feature flags et configuration dynamique

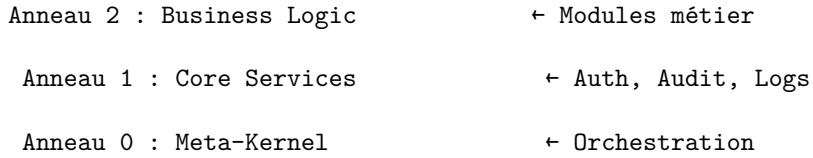
Routage et communication

- Event bus (événements inter-modules)
- API Gateway interne (routage HTTP)
- Gestion des droits d'accès (ACL par module)

2.2. Les anneaux fonctionnels

Anneau 3 : Intégrations

← APIs externes, webhooks



Anneau 0 : Meta-Kernel

- **Responsabilité** : Orchestration, configuration, health
- **Technologies** : NestJS, TypeORM, Bull (queues)
- **Modules** : core-kernel, config-manager, module-loader

Anneau 1 : Core Services

- **Responsabilité** : Services transverses critiques
- **Modules** :
 - auth-module : JWT, OAuth2, RBAC
 - audit-module : Event sourcing, logs immutables
 - logger-module : Logs structurés (Winston/Pino)
 - notification-module : Emails, push, webhooks

Anneau 2 : Business Logic

- **Responsabilité** : Logique métier spécifique
- **Modules** :
 - dossiers-module : Gestion des dossiers juridiques
 - documents-module : Versionning, signatures électroniques
 - workflow-module : Processus métier (BPMN)
 - facturation-module : Calculs, exports comptables

Anneau 3 : Intégrations

- **Responsabilité** : Communication avec l'extérieur
- **Modules** :
 - api-gateway-module : Exposition REST/GraphQL
 - webhooks-module : Intégrations tierces (Stripe, Twilio)
 - export-module : Génération de documents (PDF, Excel)

2.3. Principes de communication

Communication synchrone (HTTP/RPC)

```
// Module A appelle Module B via le Meta-Kernel
const result = await this.metaKernel.invoke('module-b', 'methodName', params);
```

Communication asynchrone (Event Bus)

```
// Module A émet un événement
this.eventBus.emit('dossier.created', { dossierId: '123' });

// Module B écoute l'événement
@OnEvent('dossier.created')
handleDossierCreated(payload: { dossierId: string }) {
  // Logique de réaction
}
```

Isolation et sandbox

- Chaque module tourne dans son propre contexte (DI isolé)
- Les modules ne se chargent **jamais** directement entre eux
- Toutes les communications passent par le Meta-Kernel

3. Implications techniques

3.1. Stack technique

Composant	Technologie principale	Alternative
Meta-Kernel	NestJS (Node.js)	Moleculer
Base de données	PostgreSQL + TypeORM	Prisma
Event Bus	Bull + Redis	RabbitMQ
Cache	Redis	Memcached
Monitoring	Prometheus + Grafana	Datadog
Logs	Winston + Loki	ELK Stack

3.2. Structure du Meta-Kernel

```
meta-kernel/
  src/
    core/
      module-loader.service.ts      # Chargement dynamique des modules
      event-bus.service.ts          # Pub/Sub inter-modules
      config.service.ts             # Configuration centralisée
      health.controller.ts          # Health checks
    modules/
      [modules amarrés dynamiquement]
    main.ts
  manifests/                      # Manifestes des modules chargés
  package.json
```

3.3. Cycle de vie d'un module

1. **Découverte** : Le Meta-Kernel scanne `manifests/` au démarrage
2. **Validation** : Vérification du manifeste (`lexorbital.module.json`)
3. **Changement** : Import dynamique du module (ESM)
4. **Injection** : Fourniture des services du kernel (eventBus, config, etc.)
5. **Initialisation** : Appel du hook `onModuleInit()`
6. **Santé** : Polling du endpoint `/health` du module
7. **Arrêt** : Appel du hook `onModuleDestroy()` lors du shutdown

4. Checklist de mise en œuvre

- Implémenter le `ModuleLoader` avec support ESM dynamique
- Configurer l'event bus (Bull + Redis) avec retry policy
- Créer le système de configuration centralisée (env + Vault)
- Implémenter les hooks de cycle de vie (`onModuleInit`, `onModuleDestroy`)
- Mettre en place le health check aggregator (tous modules)
- Configurer Prometheus pour exposer les métriques du kernel
- Documenter l'API du Meta-Kernel pour les développeurs de modules

5. Liens connexes

- [[00_lexorbital-intro]] : Vue d'ensemble de LexOrbital
- [[04_manifestes-lexorbital]] : Format des manifestes de modules
- [[06_template-module]] : Comment créer un module compatible
- [[08_modules-types]] : Catalogue des modules par anneau

Fiche n°2 : Microservices vs modules plug'n'play

LexOrbital adopte une approche **modulaire monolithique** (modules plug'n'play dans un seul runtime) plutôt qu'une architecture microservices distribuée, pour réduire la complexité opérationnelle tout en conservant la modularité.

1. Objectif de la fiche

Clarifier les différences entre l'approche microservices classique et l'architecture modulaire de LexOrbital, justifier ce choix architectural et identifier les cas où chaque approche est pertinente.

2. Concepts et décisions clés

2.1. Architecture microservices : forces et limites

Forces

- **Isolation forte** : chaque service a son propre runtime, base de données, cycle de déploiement
- **Scalabilité indépendante** : on peut échelonner uniquement les services sous charge
- **Résilience** : la panne d'un service n'affecte pas les autres (en théorie)
- **Polyglot** : chaque service peut utiliser une stack différente

Limites (qui motivent LexOrbital)

- **Complexité opérationnelle** : orchestration (Kubernetes), service mesh (Istio), observabilité distribuée
- **Latence réseau** : chaque appel inter-service passe par le réseau (sérialisation, désérialisation)
- **Transactions distribuées** : gestion complexe de la cohérence (Saga pattern, eventual consistency)
- **Overhead infrastructure** : chaque service a besoin de son propre conteneur, monitoring, logs
- **Déploiement** : coordination des versions, rolling updates, rollback multi-services

2.2. Modules plug'n'play LexOrbital : principes

Un seul runtime, plusieurs modules

- Tous les modules tournent dans le **même process Node.js** (NestJS)
- Communication **in-memory** (appels de fonctions directes ou event bus local)
- **Pas de sérialisation réseau** pour les appels inter-modules

Isolation logique, pas physique

- Chaque module a son propre **contexte DI** (Dependency Injection)

- Les modules ne s'importent **jamais** directement : tout passe par le Meta-Kernel
- Sandboxing possible via **VM contexts** ou **Worker Threads** pour les modules non-trustés

Déploiement simplifié

- **Un seul déploiement** pour l'ensemble de la station
- Ajout/retrait de modules sans redémarrage (hot reload)
- Git subtree pour gérer les sources de modules externes

2.3. Tableau comparatif

Critère	Microservices	Modules LexOrbital
Isolation	Processus séparés	Contextes DI isolés
Communication	HTTP/gRPC (réseau)	Appels in-memory
Transactions	Saga / 2PC	ACID natif (même DB)
Latence inter-modules	~5-50ms (réseau)	<1ms (mémoire)
Scalabilité	Indépendante par service	Verticale (toute la station)
Complexité ops	Élevée (K8s, mesh, etc.)	Faible (monolithique)
Déploiement	Multi-pipelines	Pipeline unique
Debugging	Distribué (tracing)	Local (stack trace simple)
Polyglot	Oui	Non (JavaScript/TypeScript)
Résilience	Indépendance des pannes	Redémarrage global

3. Implications techniques

3.1. Quand choisir LexOrbital (modules)

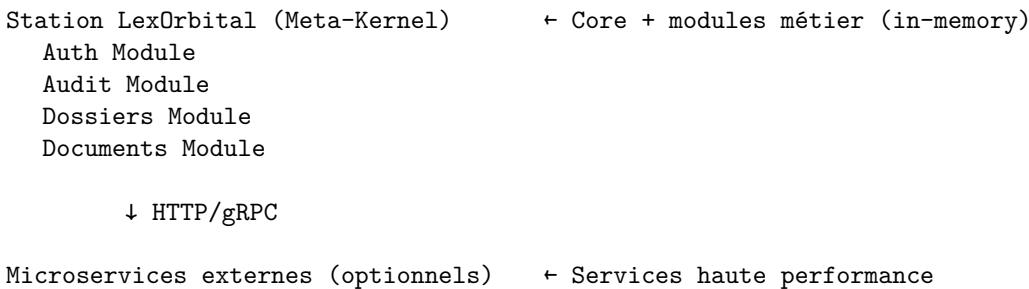
Cas d'usage adaptés : - Applications monométier ou **mono-tenant** avec forte cohésion fonctionnelle - Équipes de **petite à moyenne taille** (pas de silos organisationnels) - Besoins de **transactions complexes** entre modules (cohérence forte) - Infrastructure **limitée** (pas de K8s, pas d'équipe DevOps dédiée) - **Latence critique** (finance, temps réel) où chaque milliseconde compte

3.2. Quand privilégier les microservices

Cas d'usage adaptés : - Très grande échelle (centaines de développeurs, dizaines de services) - Besoins de **scalabilité hétérogène** (certains services x100, d'autres x1) - **Multi-tenant** avec isolation forte par client - Équipes **autonomes** avec ownership fort (chacun son service, son déploiement) - Stack **Polyglot** nécessaire (Python pour ML, Go pour perf, etc.)

3.3. Approche hybride (future évolution LexOrbital)

LexOrbital pourrait évoluer vers un **modèle hybride** :



OCR Service (Python + Tesseract)
ML Service (TensorFlow)
Search Service (Elasticsearch)

Principe : garder le cœur métier en modules (faible latence, transactions), externaliser les services spécialisés (ML, recherche, traitement lourd).

4. Checklist de décision

Avant de créer un nouveau module

- Est-ce que ce composant nécessite une **scalabilité indépendante** ? → Microservice
- Est-ce qu'il interagit **fréquemment** avec d'autres modules ? → Module in-memory
- Est-ce qu'il utilise une **stack différente** (Python, Go) ? → Microservice
- Est-ce qu'il gère des **données sensibles isolées** ? → Évaluer isolation forte
- Est-ce qu'il peut **tomber sans impacter le core** ? → Microservice optionnel

Avant d'externaliser un module en microservice

- Le gain en **scalabilité** justifie-t-il la complexité ajoutée ?
- Peut-on accepter la **latence réseau** (5-50ms par appel) ?
- A-t-on les **compétences DevOps** pour gérer le service ?
- Le service est-il **stateless** ou peut-il l'être facilement ?

5. À retenir

- LexOrbital privilégie les **modules in-memory** pour la simplicité et les performances
- Les microservices restent pertinents pour les **services spécialisés** (ML, OCR, etc.)
- L'approche modulaire réduit drastiquement la **complexité opérationnelle**
- Possible d'évoluer vers un **modèle hybride** si besoin futur de scalabilité hétérogène

6. Liens connexes

- [[01_architecture-orbitale]] : Détails techniques de l'architecture modulaire
- [[08_modules-types]] : Catalogue des modules et leur périmètre
- [[03_documentation-et-diagrammes-vivants]] : Comment documenter les dépendances entre modules

Fiche n°3 : Documentation & diagrammes vivants

La documentation de LexOrbital est **vivante et générée automatiquement** à partir du code, des manifestes et des événements réels du système. Zéro doc obsolète, zéro diagramme manuel à maintenir.

1. Objectif de la fiche

Présenter les principes et outils de documentation automatique de LexOrbital, avec focus sur les diagrammes vivants (architecture, flux de données, dépendances) et la génération de guides conformité RGPD.

2. Concepts et décisions clés

2.1. Documentation traditionnelle vs documentation vivante

Problèmes de la doc traditionnelle

- **Obsolescence rapide** : la doc est à jour le jour de sa rédaction, puis diverge du code
- **Responsabilité floue** : personne n'aime maintenir la doc
- **Recherche difficile** : PDF/Word non indexés, pas de liens hypertextes
- **Diagrammes manuels** : Visio/Draw.io désynchronisés du code

Principes de la doc vivante LexOrbital

- **Single source of truth** : le code ET les manifestes génèrent la doc
- **Diagrammes automatiques** : graphe de dépendances, flux RGPD, architecture
- **Mise à jour en CI** : chaque commit régénère la doc
- **Navigation hypertexte** : liens entre modules, événements, APIs

2.2. Sources de vérité

Manifestes de modules (`lexorbital.module.json`)

```
{  
  "name": "auth-module",  
  "version": "1.2.0",  
  "orbit": 1,  
  "dependencies": ["config-module", "logger-module"],  
  "provides": {  
    "services": ["AuthService", "TokenService"],  
    "events": ["user.login", "user.logout"]  
  },  
}
```

```

    "consumes": {
      "events": ["user.created"]
    }
}

```

→ Génère automatiquement : - Graphe de dépendances (Mermaid/D2) - Matrice de communication inter-modules - Liste des événements pub/sub

Annotations TypeScript (JSDoc/TSDoc)

```

/**
 * Authentifie un utilisateur par email/password.
 * @param email - Email de l'utilisateur
 * @param password - Mot de passe en clair (sera hashé)
 * @returns Token JWT + refresh token
 * @throws UnauthorizedException si credentials invalides
 * @gdpr Crée un log d'audit avec IP et timestamp
 */
async login(email: string, password: string): Promise<AuthTokens> {
  // ...
}

```

→ Génère automatiquement : - Documentation API (Swagger/OpenAPI) - Annotations RGPD (quel endpoint traite quelles données)

Événements runtime (Audit logs)

```

{
  "event": "document.downloaded",
  "timestamp": "2025-11-22T10:30:00Z",
  "user": "user-123",
  "module": "documents-module",
  "data": { "documentId": "doc-456" },
  "gdpr": { "legalBasis": "contract", "dataSubject": "user-123" }
}

```

→ Génère automatiquement : - Registre des activités de traitement (GDPR Art. 30) - Diagrammes de flux de données réels

2.3. Outils de génération

Type de doc	Outil principal	Format sortie
API REST	Swagger/OpenAPI	HTML interactif
Graphe de dépendances	Mermaid / D2	SVG, PNG
Architecture	C4 Model (Structurizr)	HTML + diagrammes
Registre RGPD	Custom (TypeScript)	PDF, JSON, HTML
Guides	Pandoc (Markdown → X)	PDF, DOCX, HTML
Changelog	Conventional Commits	Markdown (auto)

3. Implications techniques

3.1. Pipeline de génération (CI/CD)

```

graph LR
  A[Commit] --> B[Pre-commit hooks]

```

```

B --> C[Lint + Tests]
C --> D[Build]
D --> E[Génération doc]
E --> F[Deploy docs/]

E --> E1[Swagger API]
E --> E2[Diagrammes Mermaid]
E --> E3[Registre RGPD]
E --> E4[Pandoc HTML/PDF]

```

Script de génération (`scripts/generate-docs.sh`)

```

#!/bin/bash

# 1. Génération API REST (Swagger)
npm run swagger:generate

# 2. Graphe de dépendances (Mermaid)
node scripts/generate-dependency-graph.js > docs/generated/dependencies.mmd
mmdc -i docs/generated/dependencies.mmd -o docs/generated/dependencies.svg

# 3. Registre RGPD
node scripts/generate-gdpr-register.js > docs/generated/rgpd-register.json

# 4. Guides Pandoc
cd docs
pandoc -s --toc -o generated/guide.html [0-9][0-9]_*.md
pandoc -s --toc -o generated/guide.pdf [0-9][0-9]_*.md

echo " Documentation générée dans docs/generated/"

```

3.2. Diagrammes vivants

Graphe de dépendances (Mermaid)

Généré automatiquement depuis les `dependencies` des manifestes :

```

graph TD
    MetaKernel[Meta-Kernel]
    Auth[auth-module]
    Audit[audit-module]
    Dossiers[dossiers-module]
    Docs[documents-module]

    Auth --> MetaKernel
    Audit --> MetaKernel
    Dossiers --> Auth
    Dossiers --> Audit
    Docs --> Auth
    Docs --> Audit
    Docs --> Dossiers

```

Architecture C4 (Context, Containers, Components, Code)

Utilisation de **Structurizr** DSL pour générer les 4 niveaux :

```
// c4/workspace.dsl
workspace {
    model {
        user = person "Utilisateur" "Avocat, juriste, assistant"
        lexorbital = softwareSystem "LexOrbital" {
            metaKernel = container "Meta-Kernel" "NestJS" "Orchestration"
            authModule = container "Auth Module" "NestJS" "Authentification"
            // ...
        }
        user -> lexorbital "Utilise"
    }
    views {
        systemContext lexorbital {
            include *
        }
        container lexorbital {
            include *
        }
    }
}
```

→ Génère des diagrammes interactifs avec Structurizr Cloud ou local.

3.3. Registre RGPD automatique

À partir des manifestes rgpd-manifest.json de chaque module :

```
// scripts/generate-gdpr-register.ts
import { readdirSync, readFileSync } from 'fs';

const modules = readdirSync('modules/');
const register = [];

for (const module of modules) {
    const manifest = JSON.parse(
        readFileSync(`modules/${module}/rgpd-manifest.json`, 'utf-8')
    );
    register.push({
        module,
        dataProcessed: manifest.dataProcessed,
        legalBasis: manifest.legalBasis,
        retention: manifest.retention,
        // ...
    });
}

console.log(JSON.stringify(register, null, 2));
```

→ Sortie : fichier JSON utilisable pour audit CNIL ou export PDF.

4. Checklist de mise en œuvre

- Configurer **Swagger** pour générer l'API REST automatiquement
- Créer le script generate-dependency-graph.js (parse les manifestes)
- Installer **Mermaid CLI** (mmdc) pour générer les SVG

- Configurer **Pandoc** avec templates custom (HTML, PDF)
- Créer le script `generate-gdpr-register.ts` (agrège les manifestes RGPD)
- Configurer **Structurizr** pour les diagrammes C4 (optionnel)
- Ajouter un job **GitHub Actions** pour régénérer la doc à chaque push
- Héberger la doc sur **GitHub Pages** ou **Vercel**

5. À retenir

- La doc LexOrbital est **générée**, pas écrite manuellement
- Les **manifestes** sont la source de vérité (dépendances, RGPD, APIs)
- Les **diagrammes** se mettent à jour automatiquement (Mermaid, C4)
- Le **registre RGPD** est produit depuis les `rgpd-manifest.json`
- La **CI/CD** génère et déploie la doc à chaque commit

6. Liens connexes

- [[04_manifestes-lexorbital]] : Format des manifestes modules et RGPD
- [[05_rgpd-by-design]] : Patterns techniques pour la conformité RGPD
- [[06_template-module]] : Structure d'un module (incl. manifestes)

Fiche n°4 : Manifestes LexOrbital (module.json, rgpd-manifest.json)

Chaque module LexOrbital expose deux manifestes JSON : `lexorbital.module.json` (métadonnées techniques) et `rgpd-manifest.json` (déclaration RGPD). Ces fichiers sont la source de vérité pour l'orchestration et la conformité.

1. Objectif de la fiche

Spécifier le format exact des deux manifestes requis pour chaque module LexOrbital, leur rôle dans l'écosystème et les validations effectuées par le Meta-Kernel au chargement.

2. Concepts et décisions clés

2.1. `lexorbital.module.json` : carte d'identité technique

Ce manifeste décrit **ce que fait le module** et **comment il s'intègre** à la station.

Structure complète (avec annotations)

```
{  
  "$schema": "https://lexorbital.dev/schemas/module-manifest.v1.json",  
  
  "name": "auth-module",  
  "version": "1.2.0",  
  "description": "Authentification JWT + RBAC avec support OAuth2",  
  "author": "LexOrbital Core Team <core@lexorbital.dev>",  
  "license": "MIT",  
  "orbit": 1,  
  
  "dependencies": {  
    "required": ["config-module", "logger-module"],  
    "optional": ["notification-module"]  
  },  
  
  "provides": {  
    "services": [  
      {  
        "name": "AuthService",  
        "description": "Service principal d'authentification",  
        "methods": ["login", "logout", "verifyToken", "refreshToken"]  
      },  
    ]  
  }  
}
```

```

{
  "name": "RBACService",
  "description": "Gestion des rôles et permissions",
  "methods": ["checkPermission", "assignRole"]
}
],
"events": [
  {
    "name": "user.login",
    "payload": { "userId": "string", "timestamp": "Date", "ip": "string" }
  },
  {
    "name": "user.logout",
    "payload": { "userId": "string", "timestamp": "Date" }
  }
],
"endpoints": [
  {
    "method": "POST",
    "path": "/auth/login",
    "description": "Authentification par email/password",
    "public": true
  },
  {
    "method": "POST",
    "path": "/auth/refresh",
    "description": "Rafraîchissement du token JWT",
    "public": true
  }
]
},
"consumes": {
  "events": ["user.created", "user.updated"],
  "services": ["ConfigService", "LoggerService"]
},
"config": {
  "JWT_SECRET": { "required": true, "type": "secret" },
  "JWT_EXPIRY": { "required": false, "default": "1h" },
  "OAUTH_ENABLED": { "required": false, "default": false }
},
"health": {
  "endpoint": "/auth/health",
  "interval": 30000
},
"repository": "https://github.com/lexorbital/auth-module",
"tags": ["auth", "jwt", "rbac", "oauth2"]
}

```

Champs clés

Champ	Type	Obligatoire	Description
name	string		Nom unique du module (kebab-case)
version	semver		Version SemVer (ex: 1.2.0)
orbit	0-3		Anneau orbital (0=kernel, 1=core, 2=biz, 3=int)
dependencies	object		Modules requis/optionnels
provides.services	array		Services injectables exposés
provides.events	array		Événements émis par le module
provides.endpoints	array		Routes HTTP exposées
consumes.events	array		Événements écoutés
config	object		Variables d'environnement attendues
health.endpoint	string		Endpoint de health check

2.2. rgpd-manifest.json : déclaration RGPD

Ce manifeste décrit **quelles données** le module traite, **sur quelle base légale et combien de temps** elles sont conservées.

Structure complète

```
{
  "$schema": "https://lexorbital.dev/schemas/rgpd-manifest.v1.json",

  "module": "auth-module",
  "version": "1.2.0",
  "controller": {
    "name": "Nom de l'organisme",
    "contact": "dpo@example.com"
  },

  "dataProcessed": [
    {
      "category": "Identité",
      "fields": ["email", "nom", "prénom"],
      "purpose": "Authentification des utilisateurs",
      "legalBasis": "contract",
      "retention": {
        "duration": "5 ans après dernier login",
        "afterExpiry": "anonymisation"
      },
      "recipients": ["Meta-Kernel (logs)", "Audit Module"],
      "transfers": null,
      "rights": ["access", "rectification", "deletion", "portability"]
    },
    {
      "category": "Données de connexion",
      "fields": ["adresse IP", "user-agent", "timestamp"],
      "purpose": "Traçabilité et sécurité (Art. 6.1.f RGPD)",
      "legalBasis": "legitimate_interest",
      "retention": {
        "duration": "1 an",
        "afterExpiry": "suppression"
      },
      "recipients": ["Audit Module", "Logger Module"]
    }
  ]
}
```

```

        "transfers": null,
        "rights": ["access", "deletion"]
    },
],
]

"dataSharing": [
{
    "with": "audit-module",
    "what": ["userId", "timestamp", "action"],
    "why": "Logs d'audit immuables"
}
],
]

"securityMeasures": [
    "Hashage bcrypt des mots de passe (coût 12)",
    "Tokens JWT signés (RS256)",
    "Rate limiting sur /auth/login (10 req/min)",
    "Logs d'accès anonymisés (IP masquée après 24h)"
],
]

"dpia": {
    "required": false,
    "reason": "Pas de traitement à grande échelle de données sensibles"
},
]

"lastUpdated": "2025-11-22"
}

```

Bases légales RGPD (enum)

Valeur	Art. RGPD	Usage LexOrbital
consent	6.1.a	Inscription newsletter, cookies non-essentiels
contract	6.1.b	Données nécessaires au service
legal_obligation	6.1.c	Conservation factures (obligation légale)
vital_interest	6.1.d	Rarement utilisé (urgence vitale)
public_interest	6.1.e	Missions de service public
legitimate_interest	6.1.f	Logs de sécurité, détection fraude

3. Implications techniques

3.1. Validation au chargement (Meta-Kernel)

Lors du `onModuleInit()`, le Meta-Kernel :

1. Parse le manifeste et vérifie la présence des champs obligatoires
2. Vérifie les dépendances : tous les modules required sont-ils chargés ?
3. Valide le schéma JSON avec JSON Schema (`$schema`)
4. Enregistre les services/événements dans le registre global
5. Configure le health check (polling du endpoint)

```
// meta-kernel/src/core/module-loader.service.ts
async loadModule(modulePath: string) {
    const manifest = JSON.parse(

```

```

    readFileSync(`[${modulePath}]/lexorbital.module.json`, 'utf-8')
);

// Validation
if (!manifest.name || !manifest.version || !manifest.orbit) {
    throw new Error(`Manifeste invalide pour ${modulePath}`);
}

// Vérification dépendances
for (const dep of manifest.dependencies.required) {
    if (!this.loadedModules.has(dep)) {
        throw new Error(`Dépendance manquante : ${dep} requis par ${manifest.name}`);
    }
}

// Chargement dynamique
const module = await import(`[${modulePath}]/src/index.ts`);
this.loadedModules.set(manifest.name, { manifest, module });

// Enregistrement health check
this.healthService.register(manifest.name, manifest.health.endpoint);
}

```

3.2. Génération automatique de documentation

Les manifestes sont utilisés pour générer :

- Graphe de dépendances (Mermaid) → via dependencies
- Registre RGPD (PDF/JSON) → via rgpd-manifest.json
- API REST (Swagger) → via provides.endpoints
- Matrice Pub/Sub → via provides.events + consumes.events

Exemple de script de génération :

```

// scripts/generate-dependency-graph.ts
import { readdirSync, readFileSync } from 'fs';

const modules = readdirSync('modules/').map(dir =>
    JSON.parse(readFileSync(`modules/${dir}/lexorbital.module.json`, 'utf-8'))
);

console.log(`\`mermaid\ngraph TD`\n`);
modules.forEach(m => {
    m.dependencies?.required?.forEach(dep => {
        console.log(`  ${dep} --> ${m.name}`);
    });
});
console.log(`\``);

```

4. Checklist de mise en œuvre

Pour créer un nouveau module

- Copier le template depuis lexorbital-template-module
- Remplir lexorbital.module.json (name, version, orbit, dependencies)

- Remplir `rgpd-manifest.json` (dataProcessed, legalBasis, retention)
- Valider les manifestes avec JSON Schema (`npm run validate:manifests`)
- Implémenter le endpoint de health check (`/health`)
- Tester le chargement dans le Meta-Kernel

Pour le Meta-Kernel

- Implémenter `ModuleLoader.loadModule()` avec validation de manifeste
- Créer le registre global des services/événements
- Implémenter le health check aggregator
- Générer la doc automatique depuis les manifestes (CI)

5. À retenir

- Deux manifestes obligatoires : `lexorbital.module.json` + `rgpd-manifest.json`
- Le Meta-Kernel valide les manifestes au chargement (fail-fast)
- Les manifestes sont la source de vérité pour la doc et les diagrammes
- La conformité RGPD est déclarative (pas de code à écrire)
- Les manifestes permettent l'auto-découverte des modules (plug'n'play)

6. Liens connexes

- [[01_architecture-orbitale]] : Rôle du Meta-Kernel dans l'orchestration
- [[03_documentation-et-diagrammes-vivants]] : Génération doc depuis manifestes
- [[05_rgpd-by-design]] : Patterns techniques RGPD
- [[06_template-module]] : Template de module avec manifestes pré-remplis

Fiche n°5 : RGPD by design – patterns techniques

La conformité RGPD n'est pas une couche ajoutée a posteriori, mais **intégrée nativement** dans l'architecture LexOrbital via des patterns techniques réutilisables (audit immutable, anonymisation, retention policies).

1. Objectif de la fiche

Présenter les patterns techniques concrets pour implémenter le RGPD by design dans les modules LexOrbital : traçabilité, pseudonymisation, durées de rétention, droits des personnes (DSAR).

2. Concepts et décisions clés

2.1. Principe : Privacy by Design & by Default

Privacy by Design (Art. 25 RGPD) : intégrer la protection des données **dès la conception** du système, pas en retrofit.

Privacy by Default : les paramètres par défaut doivent minimiser le traitement de données (principe de minimisation).

2.2. Les 7 piliers RGPD de LexOrbital

Principe RGPD	Implémentation LexOrbital
Licéité	Base légale déclarée dans <code>rgpd-manifest.json</code>
Minimisation	Collecte uniquement les champs nécessaires
Exactitude	Validation + droit de rectification
Limitation de conservation	Retention policies automatiques
Intégrité/Confidentialité	Chiffrement + audit logs
Transparence	Registre RGPD généré automatiquement
Droits des personnes	API DSAR (accès, rectification, suppression)

3. Patterns techniques

3.1. Audit immutable (Event Sourcing)

Chaque action sur une donnée personnelle génère un **événement immuable** stocké dans le module **audit-module**.

Exemple : login utilisateur

```
// auth-module/src/auth.service.ts
async login(email: string, password: string, context: RequestContext) {
    // Logique d'authentification
    const user = await this.validateCredentials(email, password);

    // Émission d'un événement d'audit
    this.eventBus.emit('gdpr.data_access', {
        timestamp: new Date(),
        userId: user.id,
        action: 'login',
        dataSubject: user.id,
        legalBasis: 'contract',
        module: 'auth-module',
        ip: context.ip,
        userAgent: context.userAgent,
    });

    return this.generateTokens(user);
}
```

Stockage dans audit-module

```
// audit-module/src/audit.service.ts
@OnEvent('gdpr.data_access')
async handleDataAccess(event: GDPREvent) {
    // Insertion dans table append-only (jamais de UPDATE/DELETE)
    await this.auditRepository.insert({
        id: uuidv4(),
        timestamp: event.timestamp,
        userId: event.userId,
        action: event.action,
        module: event.module,
        ip: this.pseudonymize(event.ip), // Pseudonymisation IP
        // ...
    });
}
```

Propriétés : - Table **append-only** (pas de UPDATE/DELETE possible) - Hash de chaînage pour détecter les modifications (blockchain-like) - Rétention automatique (archive après 1 an, suppression après 5 ans)

3.2. Pseudonymisation & Anonymisation

Pseudonymisation (réversible)

Remplacer l'identifiant réel par un hash **réversible** (avec clé secrète).

```
// crypto.service.ts
import { createHmac } from 'crypto';

pseudonymize(value: string): string {
    const secret = this.configService.get('PSEUDONYMIZATION_SECRET');
    return createHmac('sha256', secret)
        .update(value)
```

```

        .digest('hex');
    }

```

Usage : logs, analytics, reporting (on peut re-identifier si besoin légal).

Anonymisation (irréversible)

Supprimer toute possibilité de ré-identification.

```

anonymize(user: User): AnonymizedUser {
    return {
        id: 'anonymized-' + uuidv4(),
        email: null,
        nom: null,
        prénom: null,
        dateCreation: user.dateCreation.toISOString().slice(0, 7), // "2025-11" (mois seulement)
        // ...
    };
}

```

Usage : après expiration de la durée de rétention.

3.3. Retention Policies automatiques

Chaque module déclare ses durées de rétention dans `rgpd-manifest.json`, et un **cron job** applique les politiques.

Déclaration dans le manifeste

```
{
  "dataProcessed": [
    {
      "category": "Dossiers clients",
      "retention": {
        "duration": "10 ans après clôture du dossier",
        "afterExpiry": "archivage"
      }
    }
  ]
}
```

Implémentation du cron

```
// retention-policy.service.ts
@Cron('0 2 * * *') // Tous les jours à 2h du matin
async applyRetentionPolicies() {
  const modules = await this.metaKernel.getLoadedModules();

  for (const module of modules) {
    const manifest = module.rgpdManifest;

    for (const data of manifest.dataProcessed) {
      const expiryDate = this.calculateExpiryDate(data.retention.duration);

      if (data.retention.afterExpiry === 'deletion') {
        await this.deleteExpiredData(module.name, data.category, expiryDate);
      }
    }
  }
}
```

```

        } else if (data.retention.afterExpiry === 'anonymisation') {
            await this.anonymizeExpiredData(module.name, data.category, expiryDate);
        } else if (data.retention.afterExpiry === 'archivage') {
            await this.archiveExpiredData(module.name, data.category, expiryDate);
        }
    }
}
}
}

```

3.4. DSAR API (Data Subject Access Rights)

API unifiée pour les droits des personnes (accès, rectification, suppression, portabilité).

Endpoint : Droit d'accès (Art. 15 RGPD)

```

// gdpr.controller.ts
@Get('/gdpr/my-data')
@UseGuards(JwtAuthGuard)
async getMyData(@CurrentUser() user: User) {
    const data = {};

    // Agrégation des données depuis tous les modules
    const modules = await this.metaKernel.getLoadedModules();

    for (const module of modules) {
        const moduleData = await module.instance.getPersonalData(user.id);
        data[module.name] = moduleData;
    }

    // Audit de l'accès
    this.eventBus.emit('gdpr.data_access_request', {
        userId: user.id,
        timestamp: new Date(),
        modules: modules.map(m => m.name),
    });
}

return {
    user: user,
    data: data,
    exportedAt: new Date(),
    format: 'JSON',
};
}

```

Endpoint : Droit à l'effacement (Art. 17 RGPD)

```

@Delete('/gdpr/delete-my-data')
@UseGuards(JwtAuthGuard)
async deleteMyData(@CurrentUser() user: User) {
    // Vérification : peut-on supprimer ? (obligation légale ?)
    const canDelete = await this.checkDeletionEligibility(user.id);
    if (!canDelete) {
        throw new ForbiddenException('Suppression impossible (obligation légale de conservation)');
    }
}

```

```

// Suppression cascadée dans tous les modules
const modules = await this.metaKernel.getLoadedModules();
for (const module of modules) {
    await module.instance.deletePersonalData(user.id);
}

// Audit de la suppression
this.eventBus.emit('gdpr.data_deletion', {
    userId: user.id,
    timestamp: new Date(),
});

return { success: true, message: 'Données supprimées avec succès' };
}

```

3.5. Chiffrement des données sensibles

Chiffrement au repos (Database)

Utiliser les fonctionnalités natives de PostgreSQL :

```

-- Activer pgcrypto
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Chiffrer une colonne
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email TEXT NOT NULL,
    password_hash TEXT NOT NULL,
    ssn TEXT, -- Numéro de sécurité sociale (sensible)
    ssn_encrypted BYTEA GENERATED ALWAYS AS (
        pgp_sym_encrypt(ssn, current_setting('app.encryption_key'))
    ) STORED
);

```

Chiffrement en transit

- **TLS 1.3** obligatoire pour toutes les connexions HTTP
- **Mutual TLS** pour les communications entre modules (si externalisation future)

Chiffrement applicatif (pour données très sensibles)

```

import { createCipheriv, createDecipheriv, randomBytes } from 'crypto';

class EncryptionService {
    private algorithm = 'aes-256-gcm';
    private key = Buffer.from(process.env.ENCRYPTION_KEY, 'hex'); // 32 bytes

    encrypt(plaintext: string): string {
        const iv = randomBytes(16);
        const cipher = createCipheriv(this.algorithm, this.key, iv);

        let encrypted = cipher.update(plaintext, 'utf8', 'hex');
        encrypted += cipher.final('hex');

        const authTag = cipher.getAuthTag();
    }
}

```

```

    return `${iv.toString('hex')}:${authTag.toString('hex')}:${encrypted}`;
}

decrypt(ciphertext: string): string {
  const [ivHex, authTagHex, encrypted] = ciphertext.split(':');
  const iv = Buffer.from(ivHex, 'hex');
  const authTag = Buffer.from(authTagHex, 'hex');

  const decipher = createDecipheriv(this.algorithm, this.key, iv);
  decipher.setAuthTag(authTag);

  let decrypted = decipher.update(encrypted, 'hex', 'utf8');
  decrypted += decipher.final('utf8');

  return decrypted;
}
}

```

4. Checklist de mise en œuvre

Pour chaque module

- Déclarer les données traitées dans `rgpd-manifest.json`
- Émettre un événement `gdpr.*` pour chaque accès/modification de données perso
- Implémenter `getPersonalData(userId)` pour le droit d'accès
- Implémenter `deletePersonalData(userId)` pour le droit à l'effacement
- Configurer les retention policies (cron ou manuel)
- Chiffrer les données sensibles (AES-256-GCM)

Pour le Meta-Kernel

- Créer le `GDPRService` centralisant les DSAR
- Implémenter le cron de retention policies
- Configurer l'audit module (event sourcing)
- Générer le registre RGPD automatique (PDF export)

Pour la conformité

- Mapper chaque traitement à une base légale (Art. 6 RGPD)
- Vérifier les durées de rétention (légales + métier)
- Tester les endpoints DSAR (accès, suppression, portabilité)
- Réaliser une DPIA si traitement à risque élevé

5. À retenir

- **Audit immutable** : tout accès à une donnée perso génère un événement
- **Pseudonymisation** : réversible (HMAC-SHA256 avec clé)
- **Anonymisation** : irréversible (après expiration rétention)
- **Retention policies** : automatiques via cron (déclarées dans manifeste)
- **DSAR API** : endpoints unifiés pour les droits des personnes
- **Chiffrement** : TLS 1.3 + AES-256-GCM pour données sensibles

6. Liens connexes

- [[04_manifestes-lexorbital]] : Format du `rgpd-manifest.json`
- [[03_documentation-et-diagrammes-vivants]] : Génération du registre RGPD
- [[08_modules-types]] : Module `audit-module` (event sourcing)

Fiche n°6 : Template module (Husky, commits, SemVer)

Le **template LexOrbital** (`lexorbital-template-module`) fournit une structure standardisée pour créer des modules : configuration Git (Husky), commits conventionnels, tests, CI/CD et manifestes RGPD pré-configurés.

1. Objectif de la fiche

Décrire la structure et l'outillage du template de module LexOrbital, permettant aux développeurs de démarer rapidement tout en respectant les standards de qualité (linting, tests, commits, versioning).

2. Concepts et décisions clés

2.1. Pourquoi un template ?

Problèmes sans template

- **Duplication de configuration** : chaque module réinvente la roue (ESLint, Prettier, Husky)
- **Commits incohérents** : pas de standard de messages (debug, fix bug, etc.)
- **Qualité variable** : certains modules ont des tests, d'autres non
- **Versioning anarchique** : 1.0.0 → 2.5.3 sans justification

Avantages du template LexOrbital

- **Standardisation** : tous les modules ont la même structure
- **Quality gates** : impossible de commit sans lint/tests
- **Versioning automatique** : SemVer calculé depuis les commits
- **Conformité RGPD** : manifeste pré-configuré à remplir

2.2. Structure du template

```
lexorbital-template-module/
  .github/
    workflows/
      ci.yml          # Tests + lint sur PR
      release.yml     # Publication automatique (npm/GitHub)
      docs.yml        # Génération doc Pandoc
  .husky/
    pre-commit       # Lint-staged (ESLint + Prettier)
    commit-msg       # Validation Conventional Commits
    pre-push         # Tests avant push
```

```

docs/
    README.md                      # Documentation du module
    00_getting-started.md
    01_api-reference.md
    templates/
        lexorbital.html
        pandoc.css
src/
    index.ts                         # Point d'entrée principal
    module.ts                        # Classe NestJS Module
    services/
    controllers/
    types/
tests/
    unit/
    integration/
    e2e/
lexorbital.module.json          # Manifeste technique
rgpd-manifest.json              # Manifeste RGPD
package.json                     # Dépendances + scripts
tsconfig.json                    # Configuration TypeScript
eslint.config.cjs               # ESLint flat config
.prettierrc                      # Prettier config
commitlint.config.ts             # Commitlint (Conventional Commits)
CHANGELOG.md                     # Changelog auto-généré
README.md                         # Documentation principale

```

3. Implications techniques

3.1. Git Hooks avec Husky

Installation

```
npm install --save-dev husky lint-staged
npx husky install
```

.husky/pre-commit : Lint + format avant commit

```
#!/bin/sh
. "$(dirname "$0")/_/husky.sh"
```

```
npx lint-staged
```

Fichier package.json :

```
{
  "lint-staged": {
    "*.{ts,tsx)": [
      "eslint --fix",
      "prettier --write"
    ],
    "*.{json,md)": [
      "prettier --write"
    ]
  }
}
```

```

}

.husky/commit-msg : Validation Conventional Commits
#!/bin/sh
. "$(dirname "$0")/_/husky.sh"

npx commitlint --edit "$1"
Fichier commitlint.config.ts :

import type { UserConfig } from '@commitlint/types';

const config: UserConfig = {
  extends: ['@commitlint/config-conventional'],
  rules: {
    'type-enum': [
      2,
      'always',
      [
        'feat',           // Nouvelle fonctionnalité
        'fix',            // Correction de bug
        'docs',           // Documentation
        'style',          // Formatage (pas de changement de code)
        'refactor',       // Refactoring (pas de feat ni fix)
        'perf',           // Amélioration de performance
        'test',           // Ajout/modification de tests
        'chore',          // Tâches de build, config, etc.
        'ci',              // Modification CI/CD
        'revert',          // Revert d'un commit
      ],
    ],
    'scope-enum': [
      2,
      'always',
      ['auth', 'audit', 'core', 'docs', 'config', 'deps'],
    ],
  },
};

export default config;
.husky/pre-push : Tests avant push
#!/bin/sh
. "$(dirname "$0")/_/husky.sh"

npm run test

```

3.2. Conventional Commits → SemVer automatique

Format des commits

<type>(<scope>): <subject>

[optional body]

[optional footer]

Exemples :

feat(auth): add OAuth2 support

Implements OAuth2 authorization code flow with PKCE.

BREAKING CHANGE: AuthService.login() signature changed

fix(audit): correct retention policy calculation

Fixes #123

Génération automatique de version (semantic-release)

npm install --save-dev semantic-release @semantic-release/changelog @semantic-release/git

Fichier .releaserc.json :

```
{
  "branches": ["main"],
  "plugins": [
    "@semantic-release/commit-analyzer",
    "@semantic-release/release-notes-generator",
    "@semantic-release/changelog",
    "@semantic-release/npm",
    "@semantic-release/git",
    "@semantic-release/github"
  ]
}
```

Règles de versioning : - fix: → PATCH (1.0.0 → 1.0.1) - feat: → MINOR (1.0.0 → 1.1.0) - BREAKING CHANGE: → MAJOR (1.0.0 → 2.0.0)

Workflow GitHub Actions (.github/workflows/release.yml)

```
name: Release

on:
  push:
    branches:
      - main

jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - uses: actions/setup-node@v4
        with:
          node-version: 20
      - run: npm ci
      - run: npm test
      - run: npx semantic-release
```

```

env:
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
  NPM_TOKEN: ${{ secrets.NPM_TOKEN }}

```

3.3. Tests automatisés

Structure des tests

```

tests/
  unit/
    services/
      auth.service.spec.ts
    controllers/
      auth.controller.spec.ts
  integration/
    auth-flow.spec.ts
  e2e/
    auth.e2e-spec.ts

```

Configuration Vitest (vitest.config.ts)

```

import { defineConfig } from 'vitest/config';

export default defineConfig({
  test: {
    globals: true,
    environment: 'node',
    coverage: {
      provider: 'v8',
      reporter: ['text', 'json', 'html'],
      exclude: ['**/node_modules/**', '**/tests/**'],
      thresholds: {
        lines: 80,
        functions: 80,
        branches: 75,
        statements: 80,
      },
    },
  },
});

```

Exemple de test unitaire

```

// tests/unit/services/auth.service.spec.ts
import { describe, it, expect, beforeEach } from 'vitest';
import { AuthService } from '../../../../../src/services/auth.service';

describe('AuthService', () => {
  let authService: AuthService;

  beforeEach(() => {
    authService = new AuthService();
  });

  it('should generate a valid JWT token', async () => {

```

```

const token = await authService.generateToken({ userId: '123' });
expect(token).toBeDefined();
expect(typeof token).toBe('string');
});

it('should throw UnauthorizedException for invalid credentials', async () => {
  await expect(
    authService.login('invalid@example.com', 'wrong')
  ).rejects.toThrow('Invalid credentials');
});
});

```

3.4. CI/CD avec GitHub Actions

.github/workflows/ci.yml : Tests sur chaque PR

```

name: CI

on:
  pull_request:
    branches: [main, develop]
  push:
    branches: [main, develop]

jobs:
  test:
    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [18, 20]

    steps:
      - uses: actions/checkout@v4
      - name: Setup Node.js ${{ matrix.node-version }}
        uses: actions/setup-node@v4
        with:
          node-version: ${matrix.node-version}

      - name: Install dependencies
        run: npm ci

      - name: Lint
        run: npm run lint

      - name: Type check
        run: npm run type-check

      - name: Run tests
        run: npm run test:ci

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          files: ./coverage/coverage-final.json

```

4. Checklist de mise en œuvre

Pour créer un nouveau module

- Cloner le template : `git clone https://github.com/lexorbital/lexorbital-template-module my-module`
- Initialiser Husky : `npm install && npx husky install`
- Remplir `lexorbital.module.json` (name, version, orbit)
- Remplir `rgpd-manifest.json` (dataProcessed, legalBasis)
- Adapter les scopes Commitlint dans `commitlint.config.ts`
- Configurer GitHub Actions secrets (NPM_TOKEN si publication npm)
- Écrire les premiers tests (au moins un test unitaire)
- Premier commit : `git commit -m "feat(core): initial module setup"`

Pour contribuer à un module existant

- Installer les dépendances : `npm install`
- Vérifier que Husky est actif : les hooks doivent se déclencher
- Créer une branche feature : `git checkout -b feat/my-feature`
- Écrire les tests avant le code (TDD)
- Commit avec Conventional Commits : `git commit -m "feat(auth): add 2FA support"`
- Pousser et ouvrir une PR : les tests CI doivent passer

5. À retenir

- **Husky** : hooks Git automatiques (lint, tests, commits)
- **Conventional Commits** : format standardisé (`feat:`, `fix:`, etc.)
- **Semantic Release** : versioning SemVer automatique
- **Tests obligatoires** : coverage minimum 80% (configuré dans Vitest)
- **CI/CD** : GitHub Actions pour tests + release automatique

6. Liens connexes

- [[04_manifestes-lexorbital]] : Format des manifestes à remplir
- [[07_workflow-subtree]] : Comment amarrer un module via git subtree
- [[08_modules-types]] : Catalogue des modules existants
- [[03_documentation-et-diagrammes-vivants]] : Génération doc avec Pandoc

Fiche n°7 : Workflow git subtree & scripts d'amarrage

LexOrbital utilise `git subtree` pour amarrer des modules externes (dépôts Git indépendants) dans le monorepo `lexorbital-core`, tout en conservant l'historique Git et la possibilité de contribuer upstream.

1. Objectif de la fiche

Expliquer le workflow de gestion des modules via git subtree, les avantages par rapport à git submodule, et fournir les scripts d'amarrage/mise à jour/contribution pour les développeurs.

2. Concepts et décisions clés

2.1. Git subtree vs Git submodule

Problèmes des submodules

- **Complexité** : `git submodule init`, `git submodule update --recursive`, etc.
- **État détaché** : les sous-modules sont souvent en HEAD détaché (detached HEAD)
- **Friction pour les contributeurs** : oubli fréquent de `--rebase-submodules`
- **Commits fantômes** : références de commits qui peuvent disparaître

Avantages des subtrees

- **Transparence** : le code du module est **physiquement présent** dans le repo
- **Clone simple** : `git clone` suffit, pas de commande supplémentaire
- **Historique préservé** : l'historique du module externe est fusionné
- **Contribution upstream facile** : `git subtree push` renvoie les commits vers le module

2.2. Architecture du monorepo avec subtrees

```
lexorbital-core/
  backend/
    src/
      meta-kernel/
      modules/
        auth-module/      ← git subtree de lexorbital-auth-module
        audit-module/    ← git subtree de lexorbital-audit-module
        dossiers-module/ ← git subtree de lexorbital-dossiers-module
        ...
  frontend/
```

```
modules/                                ← Aussi des subtrees (modules standalone)
Chaque module sous modules/ est un subtree d'un dépôt Git externe (ex: https://github.com/lexorbital/lexorbital-auth-module.git)
```

3. Implications techniques

3.1. Ajouter un module (amarrage initial)

Commande manuelle

```
# 1. Ajouter le remote du module
git remote add auth-module https://github.com/lexorbital/lexorbital-auth-module.git

# 2. Ajouter le subtree
git subtree add --prefix=modules/auth-module auth-module main --squash

# 3. (Optionnel) Supprimer le remote si on ne veut pas polluer
git remote remove auth-module
```

Explication des flags : - `--prefix=modules/auth-module` : où placer le module dans le monorepo - `auth-module` : nom du remote (peut être l'URL directement) - `main` : branche du module à ajouter - `--squash` : fusionner l'historique en un seul commit (recommandé pour éviter pollution)

Script d'amarrage (`scripts/add-module.sh`)

```
#!/bin/bash
set -e

MODULE_NAME=$1
MODULE_REPO=$2
MODULE_BRANCH=${3:-main}
TARGET_DIR="modules/${MODULE_NAME}"

if [ -z "$MODULE_NAME" ] || [ -z "$MODULE_REPO" ]; then
    echo "Usage: ./scripts/add-module.sh <module-name> <repo-url> [branch]"
    echo "Exemple: ./scripts/add-module.sh auth-module https://github.com/lexorbital/lexorbital-auth-module.git"
    exit 1
fi

echo "  Amarrage du module ${MODULE_NAME}..."

# Ajouter le remote temporaire
git remote add -f "${MODULE_NAME}-remote" "$MODULE_REPO"

# Ajouter le subtree
git subtree add --prefix="$TARGET_DIR" "${MODULE_NAME}-remote" "$MODULE_BRANCH" --squash

# Nettoyer le remote
git remote remove "${MODULE_NAME}-remote"

echo "  Module ${MODULE_NAME} amarré avec succès dans ${TARGET_DIR}"
echo "  N'oubliez pas de mettre à jour lexorbital.module.json si nécessaire."
Usage :
chmod +x scripts/add-module.sh
```

```
./scripts/add-module.sh auth-module https://github.com/lexorbital/lexorbital-auth-module.git
```

3.2. Mettre à jour un module (pull upstream)

Commande manuelle

```
# 1. Ajouter le remote si pas déjà fait  
git remote add auth-module https://github.com/lexorbital/lexorbital-auth-module.git  
  
# 2. Pull les changements upstream  
git subtree pull --prefix=modules/auth-module auth-module main --squash  
  
# 3. Résoudre les conflits si nécessaire
```

Script de mise à jour (scripts/update-module.sh)

```
#!/bin/bash  
set -e  
  
MODULE_NAME=$1  
MODULE_REPO=$2  
MODULE_BRANCH=${3:-main}  
TARGET_DIR="modules/${MODULE_NAME}"  
  
if [ -z "$MODULE_NAME" ] || [ -z "$MODULE_REPO" ]; then  
    echo "Usage: ./scripts/update-module.sh <module-name> <repo-url> [branch]"  
    exit 1  
fi  
  
echo " Mise à jour du module ${MODULE_NAME}..."  
  
# Ajouter le remote temporaire  
git remote add -f "${MODULE_NAME}-remote" "$MODULE_REPO" 2>/dev/null || true  
  
# Pull les changements  
git subtree pull --prefix="$TARGET_DIR" "${MODULE_NAME}-remote" "$MODULE_BRANCH" --squash  
  
# Nettoyer le remote  
git remote remove "${MODULE_NAME}-remote" 2>/dev/null || true  
  
echo " Module ${MODULE_NAME} mis à jour avec succès"  
Usage :  
./scripts/update-module.sh auth-module https://github.com/lexorbital/lexorbital-auth-module.git
```

3.3. Contribuer à un module (push upstream)

Si vous modifiez un module dans le monorepo et voulez renvoyer les changements vers le dépôt du module :

Commande manuelle

```
# 1. Ajouter le remote du module  
git remote add auth-module https://github.com/lexorbital/lexorbital-auth-module.git  
  
# 2. Push uniquement les commits concernant ce module
```

```
git subtree push --prefix=modules/auth-module auth-module main
```

3. Créer une PR sur le dépôt du module

Note : git subtree push peut être lent (il doit filtrer l'historique). Alternative : travailler directement dans le dépôt du module, puis pull dans le monorepo.

Workflow recommandé pour contributions importantes

1. Cloner le module indépendamment :

```
git clone https://github.com/lexorbital/lexorbital-auth-module.git
cd lexorbital-auth-module
```

2. Développer dans le module (avec tests, lint, etc.)

3. Push et créer une PR sur le dépôt du module

4. Une fois mergé, mettre à jour le subtree dans le monorepo :

```
./scripts/update-module.sh auth-module https://github.com/lexorbital/lexorbital-auth-module.git
```

3.4. Désamarrer un module (suppression)

```
#!/bin/bash
# scripts/remove-module.sh
set -e

MODULE_NAME=$1
TARGET_DIR="modules/${MODULE_NAME}"

if [ -z "$MODULE_NAME" ]; then
    echo "Usage: ./scripts/remove-module.sh <module-name>"
    exit 1
fi

echo "  Désamarrage du module ${MODULE_NAME}..."

# Supprimer le répertoire
git rm -r "$TARGET_DIR"

# Commit
git commit -m "chore(modules): remove ${MODULE_NAME}"

echo "  Module ${MODULE_NAME} désamarré avec succès"
```

4. Checklist de mise en œuvre

Pour ajouter un module externe

- Vérifier que le module a un `lexorbital.module.json` valide
- Exécuter `./scripts/add-module.sh <nom> <url-repo>`
- Vérifier que le Meta-Kernel charge bien le module (démarrer l'app)
- Commit et push

Pour mettre à jour un module

- Exécuter `./scripts/update-module.sh <nom> <url-repo>`
- Vérifier les breaking changes (lire CHANGELOG du module)
- Exécuter les tests du monorepo : `npm test`
- Commit et push

Pour contribuer à un module

- Option 1 (petits changements)** : modifier dans `modules/`, puis `git subtree push`
- Option 2 (gros changements)** : cloner le module séparément, développer, PR, puis update subtree

Pour désamarrer un module

- Vérifier qu'aucun autre module ne dépend de lui (check `dependencies` des manifestes)
- Exécuter `./scripts/remove-module.sh <nom>`
- Supprimer les références au module dans la config du Meta-Kernel

5. À retenir

- **Git subtree** : modules externes intégrés dans le monorepo (transparence totale)
- **Amarrage** : `git subtree add --prefix=modules/X <repo> main --squash`
- **Mise à jour** : `git subtree pull --prefix=modules/X <repo> main --squash`
- **Contribution** : `git subtree push` (lent) OU développer dans le repo du module
- **Scripts fournis** : `add-module.sh`, `update-module.sh`, `remove-module.sh`

6. Liens connexes

- [[06_template-module]] : Structure standard des modules à amarrer
- [[01_architecture-orbitale]] : Comment le Meta-Kernel charge les modules
- [[04_manifestes-lexorbital]] : Manifestes requis pour l'amarrage
- [[08_modules-types]] : Catalogue des modules disponibles

Fiche n°8 : Modules types (auth, audit, CI, infra...)

Catalogue des modules LexOrbital par anneau orbital, avec description, responsabilités et dépendances. Cette fiche sert de référence pour comprendre l'écosystème modulaire de la station.

1. Objectif de la fiche

Lister et décrire les modules standards de LexOrbital, leur positionnement dans les anneaux orbitaux, leurs responsabilités et leurs interactions. Cette fiche est la carte stellaire de l'architecture modulaire.

2. Modules par anneau orbital

2.1. Anneau 0 : Meta-Kernel (noyau)

`meta-kernel` (Core)

- **Responsabilité** : Orchestration centrale de tous les modules
- **Technologies** : NestJS, Bull (queues), TypeORM
- **Fonctionnalités** :
 - Chargement dynamique des modules (ESM)
 - Event bus inter-modules (pub/sub)
 - Configuration centralisée (env, Vault)
 - Health check aggregator
 - Injection de dépendances globale
- **Dépendances** : Aucune (c'est le noyau)
- **Endpoint** : `/health` (statut de tous les modules)

`config-module`

- **Responsabilité** : Gestion de la configuration applicative
- **Technologies** : NestJS Config, Vault (secrets)
- **Fonctionnalités** :
 - Variables d'environnement par module
 - Secrets management (intégration Vault/K8s)
 - Feature flags dynamiques
 - Hot reload de config (sans redémarrage)
- **Dépendances** : `meta-kernel`
- **Endpoint** : `/config` (lecture config par module)

2.2. Anneau 1 : Core Services (services transverses)

auth-module

- **Responsabilité** : Authentification et autorisation
- **Technologies** : Passport.js, JWT, OAuth2
- **Fonctionnalités** :
 - Login/logout (email/password)
 - JWT + refresh tokens
 - OAuth2 (Google, Microsoft)
 - RBAC (Role-Based Access Control)
 - 2FA (TOTP via Authenticator)
- **Dépendances** : config-module, logger-module
- **Événements émis** : user.login, user.logout, token.refreshed
- **Événements consommés** : user.created, user.deleted
- **Endpoints** :
 - POST /auth/login
 - POST /auth/refresh
 - POST /auth/logout
 - GET /auth/me

audit-module

- **Responsabilité** : Event sourcing et logs d'audit RGPD
- **Technologies** : EventStoreDB, PostgreSQL (append-only)
- **Fonctionnalités** :
 - Stockage immuable de tous les événements gdpr.*
 - Génération du registre RGPD (Art. 30)
 - Recherche/export d'audit trails
 - Hash de chaînage (blockchain-like)
- **Dépendances** : config-module, logger-module
- **Événements consommés** : gdpr.* (tous les événements RGPD)
- **Endpoints** :
 - GET /audit/events?userId=X
 - GET /audit/export?format=pdf
 - GET /audit/gdpr-register

logger-module

- **Responsabilité** : Logs structurés applicatifs
- **Technologies** : Winston, Loki (ou ELK Stack)
- **Fonctionnalités** :
 - Logs structurés JSON
 - Niveaux : debug, info, warn, error
 - Corrélation par request (trace ID)
 - Agrégation centralisée (Loki/Grafana)
- **Dépendances** : config-module
- **Endpoints** : Aucun (service interne uniquement)

notification-module

- **Responsabilité** : Notifications multi-canaux
- **Technologies** : Nodemailer (email), Twilio (SMS), OneSignal (push)
- **Fonctionnalités** :
 - Envoi d'emails transactionnels (templates)
 - SMS (codes 2FA, alertes)

- Push notifications (web + mobile)
- Queue d'envoi avec retry
- **Dépendances :** config-module, logger-module, audit-module
- **Événements consommés :** user.login, dossier.created, document.signed
- **Endpoints :**
 - POST /notifications/send
 - GET /notifications/templates

2.3. Anneau 2 : Business Logic (métier)

dossiers-module

- **Responsabilité :** Gestion des dossiers juridiques
- **Technologies :** NestJS, TypeORM, PostgreSQL
- **Fonctionnalités :**
 - CRUD dossiers (création, consultation, clôture)
 - Association clients/dossiers
 - Workflow de statuts (brouillon, actif, archivé, clôturé)
 - Recherche full-text (ElasticSearch)
- **Dépendances :** auth-module, audit-module, documents-module
- **Événements émis :** dossier.created, dossier.updated, dossier.closed
- **Événements consommés :** document.added, user.deleted
- **Endpoints :**
 - POST /dossiers
 - GET /dossiers/:id
 - PATCH /dossiers/:id
 - DELETE /dossiers/:id
 - POST /dossiers/:id/close

documents-module

- **Responsabilité :** Gestion documentaire (versionning, signatures)
- **Technologies :** NestJS, S3 (stockage), PDFKit, DocuSign
- **Fonctionnalités :**
 - Upload/download de documents
 - Versionning automatique (Git-like)
 - Signature électronique (intégration DocuSign)
 - OCR (extraction de texte via Tesseract)
 - Génération PDF (templates)
- **Dépendances :** auth-module, audit-module, dossiers-module
- **Événements émis :** document.uploaded, document.signed, document.downloaded
- **Événements consommés :** dossier.created, dossier.closed
- **Endpoints :**
 - POST /documents/upload
 - GET /documents/:id/download
 - POST /documents/:id/sign
 - GET /documents/:id/versions

workflow-module

- **Responsabilité :** Orchestration de processus métier (BPMN)
- **Technologies :** Camunda, NestJS
- **Fonctionnalités :**
 - Définition de workflows (BPMN 2.0)
 - Exécution de processus (états, transitions)

- Tâches humaines (assignation, validation)
- Timers et escalades automatiques
- **Dépendances** : auth-module, audit-module, notification-module
- **Événements émis** : workflow.started, task.assigned, workflow.completed
- **Événements consommés** : dossier.created, document.signed
- **Endpoints** :
 - POST /workflows/start
 - GET /workflows/:id/tasks
 - POST /workflows/tasks/:id/complete

facturation-module

- **Responsabilité** : Facturation et comptabilité
- **Technologies** : NestJS, Stripe, PDF generation
- **Fonctionnalités** :
 - Génération de factures (templates)
 - Paiements en ligne (Stripe)
 - Export comptable (CSV, FEC)
 - Gestion de la TVA
- **Dépendances** : auth-module, audit-module, dossiers-module
- **Événements émis** : invoice.created, payment.received
- **Événements consommés** : dossier.closed
- **Endpoints** :
 - POST /invoices
 - GET /invoices/:id/pdf
 - POST /invoices/:id/pay

2.4. Anneau 3 : Intégrations (APIs externes)

api-gateway-module

- **Responsabilité** : Exposition REST/GraphQL unifiée
- **Technologies** : NestJS, Apollo (GraphQL)
- **Fonctionnalités** :
 - Routage vers les modules métier
 - Rate limiting (par utilisateur/IP)
 - API versioning (/v1, /v2)
 - Swagger/OpenAPI automatique
- **Dépendances** : auth-module, logger-module
- **Endpoints** : Tous les endpoints publics passent par ici

webhooks-module

- **Responsabilité** : Intégrations tierces (webhooks sortants/entrants)
- **Technologies** : NestJS, Bull (queues)
- **Fonctionnalités** :
 - Webhooks sortants (notifier des services externes)
 - Webhooks entrants (Stripe, DocuSign, Twilio)
 - Retry automatique avec backoff exponentiel
 - Validation de signatures (HMAC)
- **Dépendances** : auth-module, audit-module
- **Événements consommés** : tous (peut relayer vers externe)
- **Endpoints** :
 - POST /webhooks/stripe
 - POST /webhooks/docusign

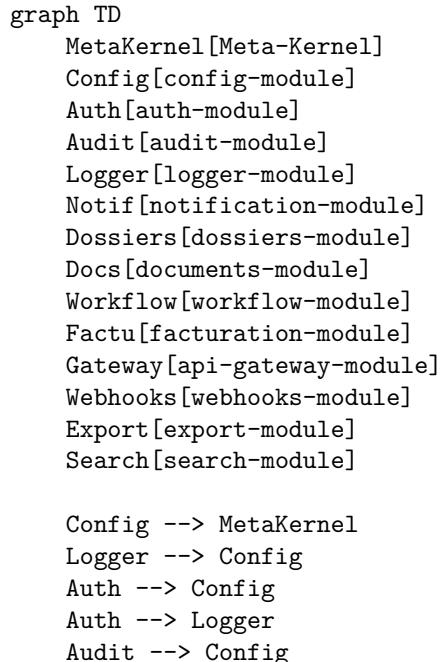
export-module

- **Responsabilité** : Génération de rapports et exports
- **Technologies** : Puppeteer (PDF), ExcelJS, Pandoc
- **Fonctionnalités** :
 - Export PDF (dossiers, factures, audits)
 - Export Excel (données tabulaires)
 - Export Word (templates de contrats)
 - Génération asynchrone (queue)
- **Dépendances** : dossiers-module, documents-module, audit-module
- **Événements consommés** : export.requested
- **Endpoints** :
 - POST /exports/pdf
 - POST /exports/excel
 - GET /exports/:id/download

search-module

- **Responsabilité** : Recherche full-text avancée
- **Technologies** : ElasticSearch, NestJS
- **Fonctionnalités** :
 - Indexation de dossiers, documents, clients
 - Recherche avec filtres et facettes
 - Autocomplétion
 - Recherche phonétique (noms propres)
- **Dépendances** : dossiers-module, documents-module
- **Événements consommés** : dossier.created, document.uploaded (réindexation)
- **Endpoints** :
 - GET /search?q=contrat&type=dossier
 - GET /search/suggest?q=dupon

3. Matrice de dépendances



```
Audit --> Logger  
Notif --> Config  
Notif --> Logger  
Notif --> Audit
```

```
Dossiers --> Auth  
Dossiers --> Audit  
Docs --> Auth  
Docs --> Audit  
Docs --> Dossiers  
Workflow --> Auth  
Workflow --> Audit  
Workflow --> Notif  
Factu --> Auth  
Factu --> Audit  
Factu --> Dossiers
```

```
Gateway --> Auth  
Gateway --> Logger  
Webhooks --> Auth  
Webhooks --> Audit  
Export --> Dossiers  
Export --> Docs  
Export --> Audit  
Search --> Dossiers  
Search --> Docs
```

4. Checklist de mise en œuvre

Pour ajouter un nouveau module à la station

- Déterminer l'anneau orbital (0, 1, 2, ou 3)
- Cloner le template : `lexorbital-template-module`
- Remplir `lexorbital.module.json` (name, orbit, dependencies)
- Remplir `rgpd-manifest.json` (si traite des données perso)
- Implémenter les services et controllers
- Définir les événements émis/consommés
- Écrire les tests (unit + integration)
- Amarrer via git subtree : `./scripts/add-module.sh`
- Mettre à jour cette fiche (08_modules-types.md)

Pour retirer un module

- Vérifier qu'aucun module ne dépend de lui (matrice de dépendances)
- Désamarrer via `./scripts/remove-module.sh`
- Mettre à jour la doc (supprimer de cette fiche)

5. À retenir

- **4 anneaux** : Kernel (0), Core (1), Business (2), Intégrations (3)
- **Modules Core** : auth, audit, logger, notification (transverses)
- **Modules Business** : dossiers, documents, workflow, facturation (métier)
- **Modules Intégrations** : api-gateway, webhooks, export, search (externe)

- **Communication** : event bus (pub/sub) + appels directs via Meta-Kernel

6. Liens connexes

- [[01_architecture-orbitale]] : Détails des anneaux orbitaux
- [[04_manifestes-lexorbital]] : Format des manifestes de modules
- [[06_template-module]] : Template pour créer un nouveau module
- [[07_workflow-subtree]] : Comment amarrer un module via git subtree

Fiche n°9 : Console de contrôle & design diégétique

La **Console de Contrôle LexOrbital** est une interface web immersive inspirée des stations spatiales, permettant de visualiser l'état des modules, gérer les amarrages/désamarrages et auditer les flux de données en temps réel.

1. Objectif de la fiche

Présenter le design, l'UX et les fonctionnalités de la Console de Contrôle LexOrbital, avec focus sur le design diégétique (interface narrative, cohérente avec la métaphore spatiale) et les outils de monitoring/debug.

2. Concepts et décisions clés

2.1. Qu'est-ce que le design diégétique ?

Définition : Le design diégétique (du grec *diégèsis*, “narration”) est une approche d’interface où les éléments UI sont intégrés dans l’univers narratif du produit, plutôt que d’être de simples widgets génériques.

Exemples célèbres : - *Dead Space* : l’UI (vie, munitions) est affichée sur le personnage lui-même (pas de HUD externe) - *Iron Man* : l’interface holographique de Jarvis fait partie de l’univers technologique - *Star Citizen* : les consoles de vaisseaux sont des objets 3D interactifs

2.2. Pourquoi pour LexOrbital ?

Problèmes des dashboards génériques

- **Ennuyeux** : tableaux de bord Grafana/Kibana = grilles de graphiques sans âme
- **Déconnectés** : pas de lien narratif avec l’architecture (pourquoi des “modules” ?)
- **Surchargés** : trop d’infos, difficile de voir l’essentiel d’un coup d’œil

Avantages du design diégétique

- **Engagement** : l’interface raconte une histoire (vous pilotez une station orbitale)
- **Clarté** : la métaphore spatiale rend l’architecture compréhensible visuellement
- **Mémorabilité** : expérience marquante, différenciation forte

2.3. Métaphore : Console de station orbitale

La Console LexOrbital s’inspire des **centres de contrôle NASA** et des **interfaces de science-fiction** (*Interstellar*, *The Expanse*).

Éléments narratifs

- **Vue 3D de la station** : visualisation des modules amarrés aux anneaux
- **Panneaux holographiques** : widgets flottants avec infos temps réel
- **Alarmes et notifications** : alertes visuelles/sonores (comme dans une station)
- **Logs défilants** : console de commande style terminal spatial
- **Orbites animées** : les modules gravitent autour du Meta-Kernel

3. Implications techniques

3.1. Stack technique

Composant	Technologie	Justification
Framework	React + TypeScript	Standard frontend, typage fort
3D Engine	Three.js + React Three Fiber	Visualisation 3D des modules/orbites
UI Components	Tailwind CSS + shadcn/ui	Composants modernes, customisables
État global	Zustand	Plus simple que Redux, performant
Temps réel	WebSockets (Socket.io)	Mise à jour live des statuts modules
Graphiques	Recharts + D3.js	Visualisations de données élégantes
Animations	Framer Motion	Animations fluides (transitions, micro-interactions)

3.2. Architecture de la Console

```
frontend/
src/
  components/
    Station3D/
      Station3D.tsx          # Rendu 3D principal (Three.js)
      Module3D.tsx           # Représentation 3D d'un module
      OrbitRing.tsx          # Anneaux orbitaux
      Camera.tsx             # Contrôles caméra
    Panels/
      ModuleStatusPanel.tsx   # État d'un module (health, logs)
      EventLogPanel.tsx       # Flux d'événements temps réel
      GDPRPanel.tsx          # Tableau de bord RGPD
      PerformancePanel.tsx   # Métriques (CPU, RAM, latence)
    Controls/
      ModuleSelector.tsx     # Sélection d'un module
      DockingControl.tsx     # Amarrage/désamarrage
      CommandLine.tsx        # Console de commandes
    UI/
      HologramPanel.tsx       # Conteneur "holographique"
      Alert.tsx              # Alertes spatiales
      LoadingSpinner.tsx      # Animation chargement
  stores/
    stationStore.ts          # État global de la station
    socketStore.ts           # Connexion WebSocket
  hooks/
    useModules.ts            # Récupération modules
    useEvents.ts             # Flux événements temps réel
    useHealthChecks.ts       # Polling health checks
```

```

App.tsx
public/
  sounds/
    dock.mp3          # Son d'amarrage
    alert.mp3         # Alerte
    ambient.mp3       # Ambiance station
  models/
    station-module.glb # Modèle 3D (optionnel)

```

3.3. Vue 3D de la station (Three.js)

Composant principal (Station3D.tsx)

```

import { Canvas } from '@react-three/fiber';
import { OrbitControls, Stars } from '@react-three/drei';
import { useModules } from '@hooks/useModules';
import Module3D from './Module3D';
import OrbitRing from './OrbitRing';

export default function Station3D() {
  const { modules } = useModules();

  return (
    <Canvas camera={{ position: [0, 0, 50], fov: 60 }}>
      {/* Fond étoilé */}
      <Stars radius={300} depth={60} count={5000} factor={7} />

      {/* Lumières */}
      <ambientLight intensity={0.3} />
      <pointLight position={[0, 0, 0]} intensity={1.5} color="#4a9eff" />

      {/* Meta-Kernel (sphère centrale) */}
      <mesh position={[0, 0, 0]}>
        <sphereGeometry args={[2, 32, 32]} />
        <meshStandardMaterial
          color="#2563eb"
          emissive="#1d4ed8"
          emissiveIntensity={0.5}
        />
      </mesh>

      {/* Anneaux orbitaux */}
      <OrbitRing radius={8} orbit={0} />
      <OrbitRing radius={15} orbit={1} />
      <OrbitRing radius={25} orbit={2} />
      <OrbitRing radius={35} orbit={3} />

      {/* Modules (positionnés sur les anneaux) */}
      {modules.map((module, index) => (
        <Module3D
          key={module.name}
          module={module}
          angle={(index / modules.length) * Math.PI * 2}
        />
      ))}
    
```

```

    /* Contrôles caméra (rotation, zoom) */
    <OrbitControls
      enableDamping
      dampingFactor={0.05}
      minDistance={20}
      maxDistance={80}
    />
  </Canvas>
);
}

```

Composant Module3D

```

import { useRef, useState } from 'react';
import { useFrame } from '@react-three/fiber';
import { Module } from '@/types';

interface Module3DProps {
  module: Module;
  angle: number;
}

export default function Module3D({ module, angle }: Module3DProps) {
  const meshRef = useRef<THREE.Mesh>(null);
  const [hovered, setHovered] = useState(false);

  // Animation : rotation autour du Meta-Kernel
  useFrame((state) => {
    if (meshRef.current) {
      const radius = module.orbit * 10;
      const speed = 0.0001 * (4 - module.orbit); // Plus proche = plus rapide
      const time = state.clock.elapsedTime;

      meshRef.current.position.x = Math.cos(angle + time * speed) * radius;
      meshRef.current.position.z = Math.sin(angle + time * speed) * radius;

      // Rotation du module sur lui-même
      meshRef.current.rotation.y += 0.01;
    }
  });
}

// Couleur selon le statut (healthy, warning, error)
const color = module.health === 'healthy' ? '#10b981' :
  module.health === 'warning' ? '#f59e0b' : '#ef4444';

return (
  <mesh
    ref={meshRef}
    onPointerOver={() => setHovered(true)}
    onPointerOut={() => setHovered(false)}
    scale={hovered ? 1.2 : 1}
  >
    <boxGeometry args={[1.5, 1.5, 1.5]} />

```

```

        <meshStandardMaterial
            color={color}
            emissive={color}
            emissiveIntensity={hovered ? 0.8 : 0.3}
        />
    </mesh>
);
}

```

3.4. Panneaux holographiques (HUD)

Composant HologramPanel

```

import { motion } from 'framer-motion';
import { ReactNode } from 'react';

interface HologramPanelProps {
    title: string;
    children: ReactNode;
    position?: 'top-left' | 'top-right' | 'bottom-left' | 'bottom-right';
}

export default function HologramPanel({ title, children, position = 'top-left' }: HologramPanelProps) {
    const positionClasses = {
        'top-left': 'top-4 left-4',
        'top-right': 'top-4 right-4',
        'bottom-left': 'bottom-4 left-4',
        'bottom-right': 'bottom-4 right-4',
    };

    return (
        <motion.div
            className={`absolute ${positionClasses[position]} w-96 bg-slate-900/80 backdrop-blur-md border-2 border-cyan-500 rounded-full`}
            initial={{ opacity: 0, y: -20 }}
            animate={{ opacity: 1, y: 0 }}
            transition={{ duration: 0.5 }}
        >
            <div className="flex items-center gap-2 mb-3 pb-2 border-b border-cyan-500/30">
                <div className="w-2 h-2 bg-cyan-400 rounded-full animate-pulse" />
                <h3 className="text-cyan-300 font-mono text-sm tracking-wider uppercase">
                    {title}
                </h3>
            </div>
            <div className="text-slate-200 text-sm font-mono">
                {children}
            </div>
        </motion.div>
    );
}

```

Utilisation

```

<HologramPanel title="Module Status" position="top-right">
    <div className="space-y-2">
        <div className="flex justify-between">

```

```

<span>Health:</span>
<span className="text-green-400">OPERATIONAL</span>
</div>
<div className="flex justify-between">
  <span>Uptime:</span>
  <span>42h 17m</span>
</div>
<div className="flex justify-between">
  <span>CPU:</span>
  <span>23%</span>
</div>
</div>
</HologramPanel>

```

3.5. Flux d'événements temps réel (WebSocket)

Hook useEvents

```

import { useEffect, useState } from 'react';
import { io } from 'socket.io-client';

interface Event {
  id: string;
  timestamp: string;
  module: string;
  type: string;
  message: string;
}

export function useEvents() {
  const [events, setEvents] = useState<Event[]>([]);

  useEffect(() => {
    const socket = io('ws://localhost:3000');

    socket.on('event', (event: Event) => {
      setEvents(prev => [event, ...prev].slice(0, 100)); // Garder 100 derniers
    });

    return () => {
      socket.disconnect();
    };
  }, []);

  return { events };
}

```

Composant EventLogPanel

```

import { useEvents } from '@/hooks/useEvents';
import HologramPanel from '@/components/UI/HologramPanel';

export default function EventLogPanel() {
  const { events } = useEvents();

```

```

    return (
      <HologramPanel title="Event Log" position="bottom-left">
        <div className="h-64 overflow-y-auto space-y-1 text-xs">
          {events.map(event => (
            <div key={event.id} className="flex gap-2 hover:bg-cyan-500/10 p-1 rounded">
              <span className="text-slate-500">
                {new Date(event.timestamp).toLocaleTimeString()}
              </span>
              <span className="text-cyan-400">[{event.module}]</span>
              <span className="text-slate-300">{event.message}</span>
            </div>
          )))
        </div>
      </HologramPanel>
    );
}

```

3.6. Effets sonores (immersion)

```

// hooks/useSoundEffects.ts
import { useRef } from 'react';

export function useSoundEffects() {
  const dockSound = useRef(new Audio('/sounds/dock.mp3'));
  const alertSound = useRef(new Audio('/sounds/alert.mp3'));

  const playDock = () => dockSound.current.play();
  const playAlert = () => alertSound.current.play();

  return { playDock, playAlert };
}

```

Usage lors d'un amarrage :

```

const { playDock } = useSoundEffects();

async function dockModule(moduleName: string) {
  await api.post('/modules/dock', { name: moduleName });
  playDock(); // Son d'amarrage
}

```

4. Checklist de mise en œuvre

Phase 1 : Fondations

- Setup React + TypeScript + Vite
- Installer Three.js + React Three Fiber
- Créer la scène 3D de base (Meta-Kernel + anneaux)
- Implémenter les composants HologramPanel et Module3D

Phase 2 : Fonctionnalités

- Connexion WebSocket pour événements temps réel
- Panel de statut des modules (health, uptime, CPU)
- Panel de logs défilants (EventLogPanel)

- Panel RGPD (registre, audits)
- Contrôles d'amarrage/désamarrage

Phase 3 : Polish & UX

- Ajouter les effets sonores (amarrage, alertes)
- Animations Framer Motion (transitions fluides)
- Responsive design (fonctionnel sur tablette)
- Mode sombre/clair (optionnel, thème spatial par défaut)
- Documentation utilisateur (tooltips, tour guidé)

5. À retenir

- **Design diégétique** : l'UI raconte l'histoire de la station orbitale
- **Vue 3D** : Three.js pour visualiser les modules en orbite
- **Panneaux holographiques** : style futuriste avec bordures cyan + backdrop blur
- **Temps réel** : WebSocket pour flux d'événements live
- **Effets sonores** : amarrages, alertes (immersion)
- **Stack** : React + Three.js + Framer Motion + Tailwind

6. Liens connexes

- [[01_architecture-orbitale]] : Comprendre les anneaux et modules à visualiser
- [[08_modules-types]] : Catalogue des modules à afficher dans la console
- [[03_documentation-et-diagrammes-vivants]] : Diagrammes complémentaires