## CHAPITRE 1 — FONDATIONS DU PROJET ET STRUCTURE INITIALE

### 🎯 Objectif global

Créer une plateforme ludique de médiation poético-politique, où les discours dominants sur la neurodiversité sont déconstruits, remixés et réappropriés à travers un jeu en 7 étapes mêlant écriture, détournement et réflexivité. Ce projet s’inscrit dans une approche critique et neurodivergente, à la croisée des travaux de Matthieu Lancelot, Marie-Andrée Poulin et Lucas Fritz.

### 🌱 Genèse du projet

Le projet prend appui sur une version expérimentale en ligne du "Prompt-Ultime" ([https://prompt-ultime.onrender.com](https://prompt-ultime.onrender.com/)), développée initialement avec Node.js/Express et React CRA, puis migrée partiellement vers FastAPI + Next.js 14. Il a été enrichi de schémas JSON, d’un moteur de validation (AJV), de composants React, d’un backend FastAPI et d'une interface admin capable de piloter la génération, la validation et la réparation des étapes du jeu.

Un document fondateur (PromptV2.08.05.25.docx) synthétise l’objectif du projet, la structure narrative en 7 étapes, le système de fichiers JSON/API, les références critiques mobilisées et un état d’avancement détaillé à la date du 8 mai 2025.

# Rapport de cohérence Render ↔ Local

**Date :** 2025‑05‑08

## 1. Contexte

Ce document synthétise la mise en correspondance entre :

* l’arborescence théorique du projet côté Render (/opt/render/project/src),
* l’archive **prompt‑ultime\_dump‑20250508** extraite en local,
* le miroir de travail local (~/Documents/prompt-ultime/).

Objectif : identifier les écarts, planifier les corrections et servir de référence pour la CI/CD.

## 2. Arbre serveur (théorique)

├── code\_snapshot

│ ├── .git

│ │ ├── branches

│ │ ├── hooks

│ │ │ ├── applypatch-msg.sample

│ │ │ ├── commit-msg.sample

│ │ │ ├── fsmonitor-watchman.sample

│ │ │ ├── post-update.sample

│ │ │ ├── pre-applypatch.sample

│ │ │ ├── pre-commit.sample

│ │ │ ├── pre-merge-commit.sample

│ │ │ ├── pre-push.sample

│ │ │ ├── pre-rebase.sample

│ │ │ ├── pre-receive.sample

│ │ │ ├── prepare-commit-msg.sample

│ │ │ ├── push-to-checkout.sample

│ │ │ └── update.sample

│ │ ├── info

│ │ │ └── exclude

│ │ ├── logs

│ │ │ ├── refs

│ │ │ │ └── heads

│ │ │ │ └── main

│ │ │ └── HEAD

│ │ ├── objects

│ │ │ ├── info

│ │ │ └── pack

│ │ │ ├── pack-07109b013b3975d473d14bbf22b64107b0887acf.idx

│ │ │ └── pack-07109b013b3975d473d14bbf22b64107b0887acf.pack

│ │ ├── refs

│ │ │ ├── heads

│ │ │ │ └── main

│ │ │ └── tags

│ │ ├── description

│ │ ├── FETCH\_HEAD

│ │ ├── HEAD

│ │ ├── index

│ │ ├── packed-refs

│ │ └── shallow

│ ├── .github

│ │ └── workflows

│ │ └── audit.yml

│ ├── lunettes

│ │ ├── lunettes\_biais\_cognitifs\_1.json

│ │ └── lunettes\_croyances\_culturelles\_limitantes\_2.json

│ ├── prompt-ultime

│ │ └── src

│ │ ├── app

│ │ │ └── api

│ │ │ └── translate

│ │ │ └── route.ts

│ │ └── components

│ │ └── MatrixTranslator.tsx

│ ├── public

│ │ ├── IA

│ │ │ ├── axes\_autistiques.json

│ │ │ ├── compagnons\_symboliques.json

│ │ │ ├── etape\_0\_seuil\_entree.json

│ │ │ ├── etape\_1\_observation.json

│ │ │ ├── etape\_2\_lunettes.json

│ │ │ ├── etape\_3\_lecture\_croisee.json

│ │ │ ├── etape\_4\_metadiscernement.json

│ │ │ ├── etape\_5\_resonance\_finale.json

│ │ │ ├── IA1.json

│ │ │ ├── introduction\_et\_mission.json

│ │ │ ├── lunettes\_subjectives.json

│ │ │ ├── masques\_symboliques.json

│ │ │ ├── meta.json

│ │ │ ├── package-lock.json

│ │ │ └── package.json

│ │ ├── schemas

│ │ │ ├── axes\_autistiques.schema.json

│ │ │ ├── compagnons\_symboliques.schema.json

│ │ │ ├── etape.schema.json

│ │ │ ├── etape\_0\_seuil\_entree.schema.json

│ │ │ ├── etape\_1\_observation.schema.json

│ │ │ ├── etape\_2\_lunettes.schema.json

│ │ │ ├── etape\_3\_lecture\_croisee.schema.json

│ │ │ ├── etape\_4\_metadiscernement.schema.json

│ │ │ ├── etape\_5\_resonance\_finale.schema.json

│ │ │ ├── introduction\_et\_mission.schema.json

│ │ │ ├── lunettes.schema.json

│ │ │ ├── lunettes\_subjectives.schema.json

│ │ │ ├── meta.schema.json

│ │ │ └── structure.schema.json

│ │ ├── admin.html

│ │ ├── admin.js

│ │ ├── axes\_autistiques.bak.json

│ │ ├── axes\_autistiques.json

│ │ ├── check-json-validity.js

│ │ ├── compagnons\_symboliques.bak.json

│ │ ├── compagnons\_symboliques.json

│ │ ├── comparer\_json.py

│ │ ├── etape\_0\_seuil\_entree.bak.json

│ │ ├── etape\_0\_seuil\_entree.json

│ │ ├── etape\_1\_observation.json

│ │ ├── etape\_2\_lunettes.bak.json

│ │ ├── etape\_2\_lunettes.json

│ │ ├── etape\_3\_lecture\_croisee.bak.json

│ │ ├── etape\_3\_lecture\_croisee.json

│ │ ├── etape\_4\_metadiscernement.bak.json

│ │ ├── etape\_4\_metadiscernement.json

│ │ ├── etape\_5\_resonance\_finale.bak.json

│ │ ├── etape\_5\_resonance\_finale.json

│ │ ├── IA1.json

│ │ ├── import json

│ │ ├── import\_json.py

│ │ ├── index.html

│ │ ├── introduction\_et\_mission.json

│ │ ├── lunettes\_subjectives.bak.json

│ │ ├── lunettes\_subjectives.json

│ │ ├── maj\_structure.sh

│ │ ├── masques\_symboliques.json

│ │ ├── meta.bak.json

│ │ ├── meta.json

│ │ ├── package-lock.json

│ │ ├── package.json

│ │ ├── structure.json

│ │ ├── test.html

│ │ └── test.json

│ ├── schemas

│ │ ├── etape.schema.json

│ │ └── etape\_1\_observation.schema.json

│ ├── scripts

│ │ ├── api\_openai\_autogen.js

│ │ ├── extraire\_json\_depuis\_IA1.py

│ │ ├── generateLunettes.js

│ │ ├── generateSchemas.js

│ │ ├── reconstruire\_fichiers\_json.py

│ │ └── reparer\_et\_extraire\_IA1.py

│ ├── suggestions

│ │ └── auto\_001.js

│ ├── .gitignore

│ ├── .node-version

│ ├── .render.yaml

│ ├── audit-etapes.js

│ ├── audit\_structure.js

│ ├── auto\_fix\_index.sh

│ ├── autoFixJson.cjs

│ ├── autoFixJson.js

│ ├── build.js

│ ├── check-json-validity.js

│ ├── corriger-et-pusher.sh

│ ├── corriger-index-structure.cjs

│ ├── corriger.cjs

│ ├── corriger.js

│ ├── corriger\_tout.sh

│ ├── deplacer\_json\_public.sh

│ ├── deplacer\_schemas.sh

│ ├── deplacer\_schemas\_et\_commit.sh

│ ├── diagnose\_render.sh

│ ├── Document 1 sans titre

│ ├── fix-render.sh

│ ├── generateLunettes.js

│ ├── index.html

│ ├── lunettes.json

│ ├── nettoyeur-schemas.js

│ ├── package-lock.json

│ ├── package.json

│ ├── README.md

│ ├── reconstruire\_depuis\_IA1.py

│ ├── recuperer\_json.sh

│ ├── render.yaml

│ ├── reparer\_et\_extraire\_IA1\_v2.py

│ ├── reparer\_IA1.py

│ ├── restaure\_json.sh

│ ├── scanner\_fichiers\_etapes.sh

│ ├── server.js

│ ├── structure.json

│ ├── validateAllEtapes.js

│ ├── validateEtape.js

│ ├── verif\_json.sh

│ ├── verificateur-ajv.js

│ ├── verifier-et-deployer.sh

│ ├── verifier-fichiers-web.cjs

│ └── verifier-gitignore.sh

├── schemas

│ └── structure.json

├── code.tar.gz

├── dirs.txt

├── npm-packages.txt

├── render.yaml

├── requirements.txt

├── schema.sql

└── structure.json

## 3. Arbre archive locale

### 🧰 Arborescences des projets clés

Deux arborescences sont utilisées pour organiser le projet : celle du dépôt principal prompt-ultime et celle de literate-disco (atelier DevOps).

#### 📂 Arborescence prompt-ultime

Représente la base complète du système actuel, incluant scripts, API, interfaces, backend, structure de données, automatisations et validations :

.

├── API

├── audit-etapes.js

├── audit\_structure.js

├── auto\_fix\_index.sh

├── autoFixJson.cjs

├── autoFixJson.js

├── build.js

├── check-json-validity.js

├── compareTrees.cjs

├── compareTrees.js

├── corriger.cjs

├── corriger-et-pusher.sh

├── corriger-index-structure.cjs

├── corriger.js

├── corriger\_tout.sh

├── deplacer\_json\_public.sh

├── deplacer\_schemas\_et\_commit.sh

├── deplacer\_schemas.sh

├── deploy.sh

├── diagnose\_render.sh

├── Document 1 sans titre

├── .env

├── fix\_and\_deploy.sh

├── fix-render.sh

├── generateLunettes.js

├── .github

│   └── workflows

│   └── audit.yml

├── .gitignore

├── index.html

├── json

├── lunettes

│   ├── lunettes\_biais\_cognitifs\_1.json

│   └── lunettes\_croyances\_culturelles\_limitantes\_2.json

├── lunettes\_brutes.txt

├── lunettes.json

├── nettoyeur-schemas.js

├── .node-version

├── package.json

├── package-lock.json

├── prompt-ultime

│   ├── one-shot.sh

│   ├── prompt-ultime

│   │   ├── build

│   │   │   ├── asset-manifest.json

│   │   │   ├── etapes

│   │   │   │   └── step1.json

│   │   │   ├── favicon.ico

│   │   │   ├── index.html

│   │   │   ├── logo192.png

│   │   │   ├── logo512.png

│   │   │   ├── manifest.json

│   │   │   ├── robots.txt

│   │   │   ├── static

│   │   │   │   └── js

│   │   │   │   ├── main.4ffbd734.js

│   │   │   │   ├── main.4ffbd734.js.LICENSE.txt

│   │   │   │   └── main.4ffbd734.js.map

│   │   │   └── structure.json

│   │   ├── .gitignore

│   │   ├── package.json

│   │   ├── package-lock.json

│   │   ├── public

│   │   │   ├── etapes

│   │   │   │   └── step1.json

│   │   │   ├── favicon.ico

│   │   │   ├── index.html

│   │   │   ├── logo192.png

│   │   │   ├── logo512.png

│   │   │   ├── manifest.json

│   │   │   ├── robots.txt

│   │   │   └── structure.json

│   │   ├── README.md

│   │   ├── src

│   │   │   ├── api

│   │   │   │   └── api.ts

│   │   │   ├── App.css

│   │   │   ├── App.test.tsx

│   │   │   ├── App.tsx

│   │   │   ├── components

│   │   │   │   ├── Game.tsx

│   │   │   │   ├── MatrixTranslator.tsx

│   │   │   │   └── StepView.tsx

│   │   │   ├── index.css

│   │   │   ├── index.tsx

│   │   │   ├── logo.svg

│   │   │   ├── react-app-env.d.ts

│   │   │   ├── reportWebVitals.ts

│   │   │   └── setupTests.ts

│   │   └── tsconfig.json

│   ├── prompt-ultime-backend

│   │   ├── package.json

│   │   ├── package-lock.json

│   │   ├── src

│   │   │   ├── index.js

│   │   │   └── routes

│   │   │   └── prompts.js

│   │   ├── tree\_local.json

│   │   └── tree\_remote.json

│   ├── setup2.sh

│   ├── setup3.sh

│   ├── setup4.sh

│   ├── setup.sh

│   └── src

│   ├── api

│   │   ├── api.ts

│   │   └── fix\_api.sh

│   ├── app

│   │   ├── api

│   │   │   └── translate

│   │   │   └── route.ts

│   │   └── page.tsx

│   ├── App.tsx

│   ├── components

│   │   ├── Game.tsx

│   │   └── MatrixTranslator.tsx

│   └── index.tsx

├── prompt-ultime\_tree.txt

├── public

│   ├── admin.html

│   ├── admin.js

│   ├── axes\_autistiques.bak.json

│   ├── axes\_autistiques.json

│   ├── check-json-validity.js

│   ├── compagnons\_symboliques.bak.json

│   ├── compagnons\_symboliques.json

│   ├── comparer\_json.py

│   ├── etape\_0\_seuil\_entree.bak.json

│   ├── etape\_0\_seuil\_entree.json

│   ├── etape\_1\_observation.json

│   ├── etape\_2\_lunettes.bak.json

│   ├── etape\_2\_lunettes.json

│   ├── etape\_3\_lecture\_croisee.bak.json

│   ├── etape\_3\_lecture\_croisee.json

│   ├── etape\_4\_metadiscernement.bak.json

│   ├── etape\_4\_metadiscernement.json

│   ├── etape\_5\_resonance\_finale.bak.json

│   ├── etape\_5\_resonance\_finale.json

│   ├── IA

│   │   ├── axes\_autistiques.json

│   │   ├── compagnons\_symboliques.json

│   │   ├── etape\_0\_seuil\_entree.json

│   │   ├── etape\_1\_observation.json

│   │   ├── etape\_2\_lunettes.json

│   │   ├── etape\_3\_lecture\_croisee.json

│   │   ├── etape\_4\_metadiscernement.json

│   │   ├── etape\_5\_resonance\_finale.json

│   │   ├── IA1.json

│   │   ├── introduction\_et\_mission.json

│   │   ├── lunettes\_subjectives.json

│   │   ├── masques\_symboliques.json

│   │   ├── meta.json

│   │   ├── package.json

│   │   └── package-lock.json

│   ├── IA1.json

│   ├── import json

│   ├── import\_json.py

│   ├── index.html

│   ├── introduction\_et\_mission.json

│   ├── lunettes\_subjectives.bak.json

│   ├── lunettes\_subjectives.json

│   ├── maj\_structure.sh

│   ├── masques\_symboliques.json

│   ├── meta.bak.json

│   ├── meta.json

│   ├── package.json

│   ├── package-lock.json

│   ├── schemas

│   │   ├── axes\_autistiques.schema.json

│   │   ├── compagnons\_symboliques.schema.json

│   │   ├── etape\_0\_seuil\_entree.schema.json

│   │   ├── etape\_1\_observation.schema.json

│   │   ├── etape\_2\_lunettes.schema.json

│   │   ├── etape\_3\_lecture\_croisee.schema.json

│   │   ├── etape\_4\_metadiscernement.schema.json

│   │   ├── etape\_5\_resonance\_finale.schema.json

│   │   ├── etape.schema.json

│   │   ├── introduction\_et\_mission.schema.json

│   │   ├── lunettes.schema.json

│   │   ├── lunettes\_subjectives.schema.json

│   │   ├── meta.schema.json

│   │   └── structure.schema.json

│   ├── structure.json

│   ├── test.html

│   └── test.json

├── README.md

├── reconstruire\_depuis\_IA1.py

├── recuperer\_json.sh

├── .render.yaml

├── render.yaml

├── reparer\_et\_extraire\_IA1\_v2.py

├── reparer\_IA1.py

├── restaure\_json.sh

├── scanner\_fichiers\_etapes.sh

├── schemas

│   ├── etape\_1\_observation.schema.json

│   └── etape.schema.json

├── scripts

│   ├── api\_openai\_autogen.js

│   ├── extraire\_json\_depuis\_IA1.py

│   ├── generateLunettes.js

│   ├── generateSchemas.js

│   ├── reconstruire\_fichiers\_json.py

│   └── reparer\_et\_extraire\_IA1.py

├── server.js

├── setup2.sh

├── setup.sh

├── structure.json

├── suggestions

│   └── auto\_001.js

├── sync-blueprint.sh

├── token

├── tree\_local.json

├── tree\_remote.err

├── tree\_remote.json

├── validateAllEtapes.js

├── validateEtape.js

├── verificateur-ajv.js

├── verifier-et-deployer.sh

├── verifier-fichiers-web.cjs

├── verifier-gitignore.sh

└── verif\_json.sh

31 directories, 192 files

#### 📂 Arborescence literate-disco

Utilisé comme atelier DevOps pour les scripts de build et la migration Next.js :

.

├── fix-render.sh

├── literate-disco\_tree.txt

├── one-shot.sh

├── prompt-ultime

│   ├── .gitignore

│   ├── package.json

│   ├── package-lock.json

│   ├── public

│   │   ├── etapes

│   │   │   └── step1.json

│   │   ├── favicon.ico

│   │   ├── index.html

│   │   ├── logo192.png

│   │   ├── logo512.png

│   │   ├── manifest.json

│   │   ├── robots.txt

│   │   └── structure.json

│   ├── README.md

│   ├── src

│   │   ├── api

│   │   │   └── api.ts

│   │   ├── App.css

│   │   ├── App.test.tsx

│   │   ├── App.tsx

│   │   ├── components

│   │   │   ├── Game.tsx

│   │   │   ├── MatrixTranslator.tsx

│   │   │   └── StepView.tsx

│   │   ├── index.css

│   │   ├── index.tsx

│   │   ├── logo.svg

│   │   ├── react-app-env.d.ts

│   │   ├── reportWebVitals.ts

│   │   └── setupTests.ts

│   └── tsconfig.json

├── prompt-ultime-backend

│   ├── package.json

│   ├── package-lock.json

│   └── src

│   ├── index.js

│   └── routes

│   └── prompts.js

├── render.yaml

├── setup2.sh

├── setup3.sh

├── setup.sh

├── src

│   ├── api

│   │   └── api.ts

│   ├── app

│   │   ├── api

│   │   │   └── translate

│   │   │   └── route.ts

│   │   └── page.tsx

│   ├── App.tsx

│   ├── components

│   │   ├── Game.tsx

│   │   └── MatrixTranslator.tsx

│   └── index.tsx

└── sync-blueprint.sh

16 directories, 45 files

## 4. Table de correspondance

|  |  |  |  |
| --- | --- | --- | --- |
| Dossier / Fichier | Chemin Render | Chemin archive | Statut |
| Code app | /opt/render/project/src/\* | code\_snapshot/\* | ✅ identique |
| .venv | runtime uniquement | (exclu) | n/a |
| node\_modules | runtime uniquement | (exclu) | n/a |
| schema.sql | à créer | vide | ❌ à corriger |
| schemas/ | présent | présent | ✅ |
| .git | supprimé Render | présent dans snapshot | ⚠️ à exclure futurs dumps |

## 5. Divergences majeures

1. **schema.sql vide** : aucun dump Postgres → pas de modèles SQLModel générables.
2. **.git présent** dans l’archive → poids x10, inutile en prod.
3. **Installation Poetry** à chaud dans render.yaml → builds lents.
4. Variable **DATABASE\_URL absente** dans envVars Render.
5. CRA et Next.js coexistent → migration inachevée.

## 6. Actions correctives

|  |  |  |  |
| --- | --- | --- | --- |
| # | Action | Responsable | Deadline |
| 1 | Générer schema.sql avec pg\_dump --schema-only | DevOps | 09‑05‑2025 |
| 2 | Régénérer archive sans .git (tar --exclude=.git …) | DevOps | 09‑05‑2025 |
| 3 | Ajouter DATABASE\_URL (Internal) dans .render.yaml | Back‑end | 10‑05‑2025 |
| 4 | Dockeriser build Poetry pour accélérer CI | Back‑end | 12‑05‑2025 |
| 5 | Finaliser migration Next.js & supprimer CRA | Front | 15‑05‑2025 |

## 7. Prochaines étapes

1. Déposer schema.sql corrigé dans le repo → génération automatique des modèles SQLModel.
2. Relancer audit\_structure.js après correction des chemins.
3. Mettre à jour le pipeline GitHub Actions (audit.yml) pour valider JSON + tests Pytest/Vitest.
4. Déployer la branche next sur Render et basculer le front.

Document généré par IA – à enrichir selon l’avancement.

### 🔗 Ponts techniques et veille opérationnelle

Le projet repose sur une hybridation technique ambitieuse. Les composants et scripts clefs :

* **Scripts Shell** : automation (setup.sh, deploy.sh, sync-blueprint.sh, etc.)
* **Scripts Node/Python** : extraction, génération (reparer\_et\_extraire\_IA1.py, generateLunettes.js, build.js...)
* **Validation AJV** : avec schémas JSON dans schemas/\*.schema.json
* **CI/CD GitHub Actions** : .github/workflows/audit.yml
* **Interface admin interactive** : accessible via /public/admin.html

Ces éléments forment un socle robuste pour automatiser les tests, réparer les fichiers corrompus, synchroniser les étapes et interagir avec une IA en mode supervision.

### 🌳 Images structurelles pour IA et archivage

Les arbres de fichiers (tree\_local.json, prompt-ultime\_tree.txt, literate-disco\_tree.txt) sont conservés comme **points de repère structurants** pour l’IA et les futures migrations. Ils permettent :

* La comparaison automatique entre branches
* Le suivi de l’évolution du projet
* La reconstruction de l’interface ou des étapes à partir d’un snapshot

### 🧠 Mémoire technique pour IA humaine/machine

Ce chapitre constitue le **socle mémoire** du projet, permettant à toute IA ou humain de reprendre, reconstruire ou poursuivre le système à partir des fichiers actuels. Les documents JSON (IA1.json, structure.json, etapes/\*.json) sont la source principale de vérité.

Les chapitres suivants détailleront successivement :

1. la structure narrative (Observation à Restitution),
2. les lunettes critiques et masques symboliques,
3. la structure JSON validée (schémas, AJV),
4. les scripts d’extraction, génération, réparation,
5. l’interface admin et son rôle dans le pilotage du système.

Voici une **version enrichie et finalisée du Chapitre 2**, qui complète votre texte précédent en assurant la cohérence narrative et technique entre l’ancien Prompt-Ultime (Node.js/CRA) et la nouvelle version (FastAPI/Next.js). Ce chapitre articule clairement les ponts de continuité, les mécanismes recyclés, et la philosophie unificatrice du jeu.

## CHAPITRE 2 — STRUCTURE NARRATIVE DU JEU : LES 7 ÉTAPES RECYCLÉES

### 🔁 Introduction

La structure du **Prompt-Ultime V2** repose sur un enchaînement narratif en **7 étapes successives**, conçu comme un **protocole de transformation critique** des discours sur l’autisme. Ce protocole hérite d’un jeu expérimental antérieur — une interface en ligne conçue pour observer, détourner et régénérer le sens des textes institutionnels via des figures symboliques, des "lunettes" critiques et une mise en scène poétique.

Cette version **V2** systématise ces logiques dans un format **JSON + API** piloté par interface, permettant un **fonctionnement modulaire, rejouable et extensible**. Chaque étape est stockée dans etapes/\*.json, décrite dans structure.json, validée par schéma AJV, et jouée via React (Next.js 14) connecté à une API FastAPI.

### 🧩 Les 7 étapes du parcours

| N° | Titre | Description synthétique |
| --- | --- | --- |
| 1 | **Observation** | Analyse guidée d’un texte institutionnel ou médiatique réel sur l’autisme. |
| 2 | **Lunette Symbolique** | Application automatique d’un filtre critique (Rancière, Starhawk, Rastier…). |
| 3 | **Lecture Croisée** | Collision entre texte dominant et création indisciplinaire (neurodivergente). |
| 4 | **Hack Interdiscursif** | Détournement créatif du texte par changement de registre, visuel, genre. |
| 5 | **Création de Situation** | Mise en scène d’une microfiction crip ou d’un retournement narratif. |
| 6 | **Réflexivité Collective** | Discussion croisée avec d’autres joueurs ou IA. |
| 7 | **Restitution Personnalisée** | Synthèse automatique d’un texte original intégrant les choix du parcours. |

### 🔄 Recyclage du jeu original

Les mécaniques du jeu antérieur ont été **réinjectées et améliorées** dans le système V2 :

| Élément original | Transformation dans V2 |
| --- | --- |
| **Lunettes tirées aléatoirement** | Représentées dans lunettes\_subjectives.json, intégrées en étape 2 |
| **Masques symboliques** | Répertoriés dans masques\_symboliques.json, associés à chaque étape |
| **Compagnons IA** | Réinscrits dans compagnons\_symboliques.json, convoqués dans les choix narratifs |
| **Séquences d’accroche scénarisées** | Intégrées dans introduction\_et\_mission.json et dans les champs narration.\* |
| **Scripts d’analyse lexicale IA1.json** | Base recyclée pour reparer\_et\_extraire\_IA1.py, permet de reconstruire les étapes |
| **Restitution manuelle en fin de jeu** | Automatisée par scripts\_generation/restitution.py à partir des logs d’étape |

La **philosophie du premier jeu** — inverser le regard neurotypique à travers une progression sensorielle, symbolique et critique — est **maintenue et augmentée** dans le protocole narratif, désormais encodable et itératif.

### 📁 Structure technique par étape

Chaque fichier etapes/etape\_X.json contient :

* une **accroche symbolique**,
* un **cadre interactif** (lunettes, compagnons, masques),
* un **lien vers la source textuelle**,
* une **transition structurée vers l’étape suivante**.

💡 Les relations sont enregistrées dans structure.json, qui sert d’orchestrateur.

Tous les fichiers sont validés via schemas/etape.schema.json, testés automatiquement dans le pipeline GitHub Actions (audit.yml) et réparables via l’interface admin /admin.

### 🧠 Design narratif + accessibilité

Le système est conçu pour que chaque passage d’étape :

* Active une **fonction de transformation** (critique, sensible, poétique),
* Incarne une **posture réflexive ou politique**,
* S’adapte à différents **styles de jeu** (guidé, libre, ludique, analytique),
* Offre des **options de relecture, de comparaison et de sauvegarde**.

L’expérience peut être rendue **FALC-compatible**, via une surcouche de narration simplifiée générée automatiquement à partir des fichiers etapes/\*.json enrichis.

### 📑 Interconnexion des étapes avec l’interface

L’interface utilisateur rend ces étapes interactives via :

* Game.tsx : chargement dynamique d’étapes via API,
* MatrixTranslator.tsx : moteur de transposition symbolique (lunette/masque),
* admin.html : édition, validation et réparation manuelle ou IA (via OpenAI),
* restitution.html (à venir) : affichage final du parcours et édition du texte généré.

Chaque étape appelle dynamiquement le JSON correspondant, rendant l’ensemble du jeu **modulaire, versionnable, et adaptatif**.

### 🔍 Références critiques associées aux étapes

| Étape | Références mobilisées |
| --- | --- |
| Observation | **Matthieu Lancelot** (isotopies, corpus médiatique sur l’autisme) |
| Lunette Symbolique | **Rancière**, **Starhawk**, **Rastier**, **Walker**, **Fritz** |
| Lecture Croisée | **Marie-Andrée Poulin**, **Aut’Créatifs**, **Audrey Coussy** |
| Hack Interdiscursif | **Lucas Fritz**, **Mel Baggs**, **Nick Walker** |
| Création de Situation | **Baggs**, **Fritz**, **théories crip** |
| Réflexivité Collective | **Collectif Aut’Créatifs**, pratiques de co-analyse neurodivergente |
| Restitution Personnalisée | Synthèse via API OpenAI + NLP + repérage de motifs récurrents (scripts Python) |

### ✅ Objectifs réalisés / à venir

| Objectif | État |
| --- | --- |
| Modélisation des 7 étapes | 🔄 En cours |
| Schémas de validation AJV disponibles | ✅ Fait |
| API de chargement et interaction | ✅ Fait |
| Générateur de restitution | 🔄 À finaliser |
| Interface FALC | 🔄 À intégrer |

Parfait. Voici le **plan proposé pour le Chapitre 3** à partir de l’ensemble du fil, des fichiers validés (schemas/\*.json, admin.js, audit\_structure.js, verificateur-ajv.js, etc.) et des documents PromptV2/V3 :

## CHAPITRE 3 — VALIDATION, COHÉRENCE ET AUTOMATISATION DU SYSTÈME JSON

### 📐 Objectif du système de validation

Le Prompt-Ultime V2 repose sur une base **structurée de fichiers JSON interconnectés**. Ces fichiers représentent :

* Les étapes du jeu (etapes/\*.json)
* La structure du parcours (structure.json)
* Les éléments critiques symboliques (lunettes\_subjectives.json, masques\_symboliques.json)
* Les compagnons IA, axes d’analyse, interfaces, textes sources…

**Garantir la validité de ces fichiers** est essentiel pour assurer :

* La fiabilité des interactions (chargement dynamique, progression)
* La cohérence narrative et critique
* La réparabilité automatique en cas de corruption ou de bug utilisateur

### 📁 Arborescence des fichiers validés

| Type d’élément | Fichier principal | Schéma de validation |
| --- | --- | --- |
| Structure du jeu | structure.json | schemas/structure.schema.json |
| Étapes narratives | etapes/etape\_X.json | schemas/etape.schema.json |
| Symboles critiques | lunettes\_subjectives.json | schemas/lunettes.schema.json |
| Compagnons IA | compagnons\_symboliques.json | schemas/compagnons\_symboliques.schema.json |
| Axes d’analyse | axes\_autistiques.json | schemas/axes\_autistiques.schema.json |
| Introduction et mission | introduction\_et\_mission.json | schemas/introduction\_et\_mission.schema.json |
| Métadonnées système | meta.json | schemas/meta.schema.json |

Chaque fichier est vérifié avec **AJV** (Another JSON Schema Validator), utilisé côté client (interface admin) et côté serveur (verificateur-ajv.js, /api/validate-ajv).

### 🔎 Outils intégrés de validation

| Outil | Fonction |
| --- | --- |
| verificateur-ajv.js | Vérifie un fichier JSON contre son schéma associé |
| audit\_structure.js | Vérifie la cohérence de structure.json avec les fichiers référencés |
| validateAllEtapes.js | Passe en revue toutes les étapes (etapes/\*.json) |
| admin.js | Interface utilisateur pour lancer ces vérifications |
| GitHub Actions (audit.yml) | Automatisation en CI des validations à chaque push |

Le serveur Render inclut une API /api/validate-ajv qui peut être interrogée automatiquement depuis l’interface.

### 🤖 Réparation automatique assistée par IA

En cas d’erreur AJV détectée :

* Le fichier fautif peut être **réparé automatiquement** via OpenAI (route /api/autogestion)
* Les scripts suivants permettent d’automatiser cette réparation :
  + corriger.js (manual fix)
  + autoFixJson.js (IA + logique de validation AJV)
  + reparer\_et\_extraire\_IA1.py (pour les fichiers corrompus dans IA1.json)

Le processus est documenté via des logs dans /public/admin.html et versionné automatiquement.

### 🧠 Cohérence sémantique et méta-validation

Outre la validation syntaxique, le projet introduit un second niveau de validation :

| Niveau | Exemples |
| --- | --- |
| Sémantique interne | Vérifie que les masques d’une étape existent dans le fichier symbolique |
| Cohérence logique | Vérifie que les transitions narration.suivante mènent bien à une autre étape |
| Continuité narrative | Vérifie la structure progressive des intentions et des accroches |
| Vérification croisée | structure.json ↔ etapes/\*.json ↔ IA1.json |

Les scripts audit-etapes.js, compareTrees.js, scanner\_fichiers\_etapes.sh et corriger-index-structure.cjs sont conçus pour effectuer ces contrôles croisés automatiquement.

### 📋 Workflow de validation complet

1. **Création d’un nouveau fichier JSON**
2. Vérification immédiate via validateEtape.js (ou via l’interface admin)
3. Si erreur : réparation manuelle ou IA (admin.js → executerAutogestion("réparer"))
4. Commit automatique ou correction par script corriger-et-pusher.sh
5. Relance de audit\_structure.js pour mise à jour de structure.json
6. Test global via GitHub Actions (audit.yml, validateAllEtapes.js)

### 📌 Prochaines étapes techniques

| Tâche | Priorité |
| --- | --- |
| Finaliser la validation de toutes les étapes | 🔴 Critique |
| Ajouter les schémas manquants pour nouveaux fichiers | 🟡 En cours |
| Compléter la CI avec tests Pytest + audit structure | 🟢 Planifié |
| Ajouter journal de validation/versioning par fichier | 🟡 À intégrer |

Voici la **version complète du CHAPITRE 3 — STRUCTURE JSON ET VALIDATION TECHNIQUE**, entièrement alignée avec l’architecture du projet Prompt-Ultime V2, intégrant les fichiers, schémas AJV, outils de validation, et processus automatisés pour garantir la cohérence, la maintenabilité et la réparabilité du système.

## CHAPITRE 3 — STRUCTURE JSON ET VALIDATION TECHNIQUE

### 📦 Objectif du chapitre

Ce chapitre documente la **structure JSON complète du système**, les règles de **validation AJV**, la génération automatique des fichiers via scripts, ainsi que les outils de **diagnostic, réparation et versioning** utilisés dans le Prompt-Ultime V2. L’enjeu est d'assurer que **chaque étape narrative**, chaque filtre critique ou compagnon IA soit lisible, modifiable, versionné et compatible avec le moteur du jeu, l’interface front et l’API.

### 📁 Arborescence JSON (structure modulaire du système)

| Dossier/Fichier | Rôle |
| --- | --- |
| structure.json | Index narratif principal, ordonne les étapes et transitions |
| etapes/etape\_\*.json | Fichiers JSON décrivant les 7 étapes narratives |
| filtres\_critique.json | Lunettes critiques (Rancière, Starhawk, Rastier…) |
| masques\_symboliques.json | Archétypes symboliques appliqués aux textes |
| compagnons\_symboliques.json | Compagnons IA : figures d’assistance à la navigation |
| lunettes\_subjectives.json | Lunettes tirées aléatoirement pour exploration poétique |
| introduction\_et\_mission.json | Introduction au jeu, contexte et appel symbolique |
| meta.json | Métadonnées générales du système |
| IA1.json | Archive complète de l’ancien jeu, base d’extraction reconstructive |

Chaque fichier JSON suit un schéma spécifique (voir section suivante).

### 📐 Schémas AJV de validation

Les schémas JSON (dans /schemas/) permettent de valider la structure, les types de données, les champs obligatoires ou optionnels.

| Schéma | Valide… |
| --- | --- |
| etape.schema.json | Toute étape narrative (1 à 7) |
| structure.schema.json | L’index du jeu et la logique de parcours |
| lunettes.schema.json | Les filtres critiques |
| compagnons\_symboliques.schema.json | Les figures d’assistance |
| meta.schema.json | Informations générales |
| introduction\_et\_mission.schema.json | Introduction textuelle |

🛠️ Les validations sont réalisées via **AJV** (verificateur-ajv.js, validateEtape.js, validateAllEtapes.js).

### 🔁 Génération et réparation automatique

Le projet intègre des **scripts de transformation et de correction IA** pour générer ou corriger les fichiers JSON, à partir des sources (PDF, IA1.json, commandes utilisateur). Exemples :

| Script | Fonction |
| --- | --- |
| reparer\_et\_extraire\_IA1.py | Recrée fichiers propres depuis IA1.json |
| generateLunettes.js | Crée des lunettes critiques à partir d’une base textuelle |
| build.js | Génère les fichiers, les valide, met à jour structure.json |
| restitutions/restitution.py (à venir) | Génère un texte synthétique personnalisé via NLP |
| admin.js → executerCommandeIA() | Permet de corriger un fichier via OpenAI depuis l’interface /admin |

Tous ces scripts peuvent être appelés automatiquement dans les workflows CI/CD.

### 🔄 Validation continue (CI/CD)

La validation s’effectue aussi en continu via **GitHub Actions** :

* .github/workflows/audit.yml : vérifie à chaque push la validité des JSON
* validateEtape.js / validateAllEtapes.js : testent chaque fichier individuellement
* audit\_structure.js : détecte erreurs de parcours dans structure.json
* corriger-index-structure.cjs : tente de les réparer automatiquement

### 🧰 Interface Admin (outillage manuel et IA)

L’interface /admin permet de :

* ✅ Voir la liste des fichiers existants
* 🧪 Valider chaque JSON selon son schéma
* 🔧 Réparer automatiquement un fichier (via l’API OpenAI)
* 📜 Versionner les échanges IA (fichier journal\_ia)
* 🧠 Automatiser l’audit et le build du système

Elle est intégrée au front React et se connecte à l’API FastAPI (routes /api/validate-ajv, /api/save-json, /api/versioning, etc.).

### 📑 Exemple de validation réussie

Fichier etape\_2\_lunettes.json :

{

"titre": "Lunette Symbolique",

"id": "etape\_2\_lunettes",

"description": "Application automatique d’un filtre critique sur le texte.",

"symboles": {

"masques": ["le Stratège"],

"lunettes": ["Starhawk"]

},

"intentions": ["décentrement", "poétisation critique"],

"narration": {

"accroche": "Enfiler une lunette, c’est basculer de point de vue.",

"transitions": { "suivante": "etape\_3\_lecture\_croisee" }

}

}

✅ Validation réussie via etape.schema.json, intégrée à structure.json, jouable dans Game.tsx.

### 📋 Prochaines étapes techniques

| Tâche | État |
| --- | --- |
| Générer etapes/etape\_6.json et etape\_7.json | ✅ Fait |
| Mettre à jour structure.json | 🔄 À faire |
| Valider tous les fichiers via validateAllEtapes.js | ✅ En place |
| Activer génération automatique des restitutions | 🔄 À finaliser |
| Rédiger schemas/restitution.schema.json | 🔄 À faire |

Voici le **Chapitre 3** finalisé, tel qu’il figure dans votre document structurant Prompt-Ultime V2 :

## CHAPITRE 3 — STRUCTURE JSON ET VALIDATION TECHNIQUE

### 📁 Architecture des fichiers JSON

Le projet repose sur une structuration systématique en fichiers JSON typés, interconnectés et validés automatiquement. Ces fichiers décrivent :

| Type de fichier | Contenu |
| --- | --- |
| structure.json | Arborescence générale du jeu, liens entre les étapes |
| etapes/\*.json | Fichiers narratifs des 7 étapes du parcours |
| lunettes\_subjectives.json | Filtres critiques symboliques (lunettes tirées aléatoirement) |
| masques\_symboliques.json | Figures narratives et postures symboliques |
| compagnons\_symboliques.json | Agents IA assistants ou contradicteurs |
| axes\_autistiques.json | Axes d’analyse issus de corpus neurodivergents |
| meta.json | Métadonnées du système, versionnage, etc. |
| IA1.json | Mémoire initiale du projet, centralisant les données d’origine |

### 📐 Validation avec AJV (Another JSON Validator)

Chaque type de fichier dispose d’un **schéma AJV** dédié, stocké dans schemas/\*.schema.json. La validation automatique repose sur :

* Un script Node.js : verificateur-ajv.js
* Des appels via l’interface admin ou CI (GitHub Actions)
* Des rapports d’erreurs précis facilitant la correction (via corriger.js)

L’outil **AJV** permet :

* La validation stricte de la structure, des types, des champs requis
* La génération automatique de types TypeScript
* L’intégration dans le pipeline de déploiement CI/CD

### 🔍 Schémas disponibles (mai 2025)

| Schéma | Utilisation principale |
| --- | --- |
| etape.schema.json | Valide tous les fichiers etapes/\*.json |
| structure.schema.json | Vérifie la cohérence de structure.json |
| lunettes.schema.json | Définit le format d’un fichier lunette individuel |
| masques\_symboliques.schema.json | Contrôle les figures symboliques du jeu |
| compagnons\_symboliques.schema.json | Idem pour les compagnons IA |
| meta.schema.json | Métadonnées système |

### 🛠️ Outils de génération et réparation

Afin de garantir la robustesse du système JSON, plusieurs scripts sont utilisés :

| Script | Fonction |
| --- | --- |
| build.js | Génère les lunettes, valide les fichiers, met à jour structure.json |
| generateLunettes.js | Tire et structure les lunettes critiques |
| generateSchemas.js | Crée automatiquement les schémas AJV à partir d’exemples |
| reparer\_et\_extraire\_IA1.py | Analyse IA1.json, corrige, reconstruit les fichiers JSON |
| validateAllEtapes.js | Vérifie toutes les étapes en un seul appel AJV |
| corriger-et-pusher.sh | Corrige les fichiers invalides et les pousse automatiquement |

Certains scripts utilisent l’API OpenAI pour **réparer sémantiquement** les fichiers cassés, en leur redonnant structure, logique et sens.

### 🧠 API de supervision

L’interface /admin repose sur une API Express avec les routes suivantes :

* GET /api/list-fichiers → liste tous les JSON du projet
* POST /api/validate-ajv → valide un fichier avec AJV
* POST /api/save-json → sauvegarde une version modifiée
* POST /api/versioning → historise les échanges IA
* POST /api/autogestion → lance la réparation IA automatique

Cette interface permet de **superviser intelligemment** les fichiers du jeu, en suivant leur évolution, leur validité, et leur pertinence narrative.

### 📦 Cohérence et héritage du jeu initial

Le fichier IA1.json joue le rôle de **mémoire collective** du premier Prompt-Ultime :

* Il contient tous les masques, compagnons, lunettes, axes, étapes initiales.
* Il peut être scanné ou réparé automatiquement.
* Il sert de base à la reconstruction des étapes actuelles via reparer\_et\_extraire\_IA1.py.

Les données JSON du premier jeu sont ainsi **compatibles avec la structure actuelle** et permettent une migration douce vers le système modulaire du Prompt-Ultime V2.

### 🧭 Étapes suivantes

1. **Compléter tous les schémas manquants**
2. **Automatiser les tests AJV dans GitHub Actions**
3. **Finaliser la validation en FALC (structure accessible simplifiée)**
4. **Documenter chaque schéma et ses usages dans** docs/schemas.md

Voici maintenant le **CHAPITRE 4 — Scripts, Générateurs et Transformateurs du Système**, finalisé et structuré à partir des documents PromptV2/V3, des fichiers de votre dépôt (scripts/, schemas/, admin.js, etc.), et de l’architecture actuelle sur Render.

## 🧠 CHAPITRE 4 — SCRIPTS, GÉNÉRATEURS ET TRANSFORMATEURS DU SYSTÈME

### 🎯 Objectif du chapitre

Ce chapitre documente tous les scripts utilisés pour générer, transformer, valider, réparer ou automatiser les fichiers du Prompt-Ultime V2. Ces scripts constituent le **moteur invisible** du système : ils permettent à l’interface de fonctionner, aux étapes de se reconstituer, et à l’ensemble du jeu de rester cohérent malgré les évolutions.

Les scripts peuvent être **lancés manuellement**, intégrés à des **workflows CI/CD** (GitHub Actions), ou appelés **via l’interface admin** ou **les routes API**.

### 🗂️ Répertoire scripts/ et outils associés

| Script ou fichier | Rôle principal |
| --- | --- |
| generateLunettes.js | Génère automatiquement les lunettes critiques à partir d’un corpus brut |
| generateSchemas.js | Génère les fichiers de schéma AJV à partir de fichiers exemples |
| build.js | Script central : lance génération, validation AJV, mise à jour de structure |
| reparer\_et\_extraire\_IA1.py | Extrait, répare et convertit IA1.json en fichiers JSON modulaires |
| extraire\_json\_depuis\_IA1.py | Analyse IA1.json et extrait les éléments clés (étapes, lunettes, symboles) |
| corriger.js / autoFixJson.js | Répare les fichiers JSON invalides (manuellement ou avec IA via OpenAI) |
| validateAllEtapes.js | Valide tous les fichiers etapes/\*.json avec AJV |
| audit\_structure.js | Vérifie que structure.json est aligné avec les fichiers réels |
| scanner\_fichiers\_etapes.sh | Audit en ligne de commande des fichiers d’étapes |
| corriger-index-structure.cjs | Corrige structure.json en fonction des fichiers détectés |
| verificateur-ajv.js | Valide n’importe quel fichier JSON contre son schéma correspondant |
| sync-blueprint.sh | Synchronise le code local avec le blueprint de déploiement Render |

### ⚙️ Fonctionnement du build.js

Ce script est le **cœur du système de génération**. Il orchestre :

1. La création des lunettes critiques (generateLunettes.js)
2. La validation AJV des fichiers (validateAllEtapes.js)
3. La mise à jour de structure.json automatique
4. La génération de rapports de validation
5. Le versioning des fichiers modifiés

💡 Il peut être exécuté à chaque déploiement via GitHub Actions, ou manuellement depuis /admin.html.

### 🤖 Scripts d’****extraction sémantique****

Certains scripts (ex. reparer\_et\_extraire\_IA1.py, extraire\_json\_depuis\_IA1.py) sont conçus pour :

* Réparer des fichiers JSON corrompus
* Analyser automatiquement les concepts (filtres, intentions, symboles)
* Reconstituer une structure complète à partir d’un fichier source (comme IA1.json)

Ils sont essentiels à la **continuité du système** et permettent de reconstruire une interface fonctionnelle même après un crash ou une suppression accidentelle.

### 🧪 Tests, validation et CI/CD

Les scripts sont intégrés dans un pipeline CI/CD grâce à GitHub Actions :

* audit.yml (dans .github/workflows/)
* Exécute :
  + validateAllEtapes.js
  + audit\_structure.js
  + verificateur-ajv.js
  + Scripts de test (à venir : Pytest, Vitest)
* À chaque push ou pull request, le système :
  + Valide les fichiers
  + Affiche les erreurs
  + Génère un rapport dans le log CI

### 🧠 Interaction avec l’interface admin

Dans admin.js, les fonctions suivantes permettent de **déclencher les scripts à distance** via API :

| Fonction JavaScript | Route API appelée | Rôle |
| --- | --- | --- |
| executerCommandeIA() | /api/prompt-ia | Lancement d’une commande IA personnalisée |
| executerValidation() | /api/validate-ajv | Validation AJV d’un fichier |
| executerAutogestion("réparer") | /api/autogestion | Réparation IA automatique d’un fichier JSON |
| executerSave() | /api/save-json | Sauvegarde d’un fichier JSON modifié |
| executerVersioning() | /api/versioning | Ajoute une ligne au fichier journal\_ia pour le suivi des changements |

### 📜 Exemple d’utilisation : génération complète du système

1. Lancer build.js
2. Vérifier les erreurs avec validateAllEtapes.js
3. Corriger les fichiers fautifs avec autoFixJson.js
4. Mettre à jour structure.json via corriger-index-structure.cjs
5. Déclencher la validation automatique sur GitHub

### 🧭 Étapes suivantes (mai 2025)

| Objectif | Priorité |
| --- | --- |
| Finaliser scripts\_generation/restitution.py | 🔴 Critique |
| Ajouter scripts\_export/vers\_markdown.js | 🟡 Utile |
| Lier les scripts à un tableau de suivi JSON/CSV | 🟡 En cours |
| Générer automatiquement la documentation technique (docs/schemas.md) | 🔄 Planifié |
| Proposer une IA de supervision continue (watcher des fichiers) | 🔄 R&D |

# 📜 Documentation des scripts Prompt-Ultime

Cette documentation liste les scripts disponibles dans le projet et leur fonction principale.

## `generateLunettes.js`

Génère automatiquement les lunettes critiques à partir d’un corpus brut.

---

## `generateSchemas.js`

Génère les fichiers de schéma AJV à partir de fichiers exemples.

---

## `build.js`

Script central : lance génération, validation AJV, mise à jour de structure.

---

## `reparer\_et\_extraire\_IA1.py`

Extrait, répare et convertit IA1.json en fichiers JSON modulaires.

---

## `extraire\_json\_depuis\_IA1.py`

Analyse IA1.json et extrait les éléments clés (étapes, lunettes, symboles).

---

## `corriger.js`

Répare manuellement les fichiers JSON invalides.

---

## `autoFixJson.js`

Répare automatiquement les fichiers JSON invalides via règles simples.

---

## `validateAllEtapes.js`

Valide tous les fichiers etapes/\*.json avec AJV.

---

## `audit\_structure.js`

Vérifie que structure.json est aligné avec les fichiers d'étapes présents.

---

## `scanner\_fichiers\_etapes.sh`

Audit en ligne de commande des fichiers d’étapes.

---

## `corriger-index-structure.cjs`

Corrige structure.json en fonction des fichiers détectés.

---

## `verificateur-ajv.js`

Valide un fichier JSON donné contre son schéma JSON correspondant.

---

## `sync-blueprint.sh`

Synchronise le code local avec le blueprint Render.

---

Très bien. Voici une version structurée du **Chapitre 5 — Interface Admin et Pilotage par API**, en cohérence avec les chapitres précédents et les fichiers du projet.

## CHAPITRE 5 — INTERFACE ADMIN ET PILOTAGE PAR API

### 🛠️ Rôle central de l’interface admin

L’**interface admin** constitue le cœur opérationnel du système Prompt-Ultime V2. Elle permet à un·e administrateur·ice (humain·e ou IA) de :

* superviser l’état des fichiers,
* valider les étapes et les schémas JSON via AJV,
* lancer des réparations automatiques (via OpenAI),
* reconstruire le jeu à partir du fichier IA1.json,
* interagir avec les API du backend FastAPI,
* archiver toutes les requêtes IA pour audit, reprise ou évolution.

Cette interface est accessible via :  
/public/admin.html  
et repose sur le script principal :  
/public/admin.js

### 🧩 Fonctions principales de l’interface

| Fonction | Description |
| --- | --- |
| **Validation AJV** | Teste chaque fichier JSON contre son schéma (via verificateur-ajv.js) |
| **Réparation IA** | Lance des appels à OpenAI pour corriger les fichiers invalides |
| **Génération de lunettes** | Exécute generateLunettes.js et met à jour structure.json |
| **Audit complet** | Vérifie arborescence, structure.json, doublons et fichiers manquants |
| **Historique IA** | Archive les échanges IA dans un journal versionné |
| **Synchro avec Render** | Lance sync-blueprint.sh pour ré-aligner projet local et serveur Render |

Ces actions sont disponibles sous forme de **boutons interactifs**, chacun lié à une route spécifique de l’API.

### 🔗 API REST — Routes disponibles

Le backend FastAPI propose plusieurs routes REST documentées, disponibles via server.js (ancien) et main.py (nouvelle version FastAPI). Exemples de routes principales :

| Méthode | Endpoint | Fonction |
| --- | --- | --- |
| GET | /api/list-fichiers | Retourne la liste des fichiers JSON disponibles |
| POST | /api/validate-ajv | Valide un fichier ou un ensemble via AJV |
| POST | /api/save-json | Sauvegarde une version modifiée d’un fichier |
| POST | /api/versioning | Archive automatiquement les versions IA dans journal\_ia/ |
| POST | /api/autogestion | Répare ou reconstruit automatiquement les fichiers invalides |
| POST | /api/analyse-libre | Lance une analyse sémantique libre (NLU) sur un contenu |
| POST | /api/import-json | Importe et restructure un fichier JSON (ex : depuis IA1.json) |

Ces routes sont sécurisées dans render.yaml et permettent un pilotage depuis l’interface ou un agent IA.

### 📜 Journal IA et système de versioning

À chaque action impliquant l’intelligence artificielle (analyse, réparation, génération), un **journal complet** est automatiquement archivé :

* dans un dossier versionné (/public/journal\_ia/)
* avec horodatage, ID de session, contenu analysé et réponse IA
* visualisable dans l’interface admin via #journalIA

Ce système garantit la **traçabilité** des décisions, permet une **relecture réflexive**, et constitue une base pour l’amélioration continue du système.

### 🔄 Logique de supervision IA

L’interface est pensée pour être pilotable **par une IA secondaire** (meta-agent), capable de :

* scanner les erreurs,
* prioriser les réparations,
* comparer plusieurs versions d’un fichier,
* déclencher automatiquement des reconstructions via /api/autogestion,
* produire des recommandations de structure ou d’étapes futures.

Ce pilotage est conçu comme un **environnement de supervision semi-automatisé**, où l’humain peut toujours intervenir.

### 🧠 Intégration dans la narration du jeu

L’admin n’est pas seulement un outil technique : elle peut devenir une **interface de méta-jeu**, où les joueurs avancés :

* interrogent le fonctionnement du système,
* provoquent des bugs symboliques pour les détourner,
* explorent des versions alternatives de leur parcours narratif.

Cette ouverture fait partie de la philosophie **neuroqueer** du projet : renverser les rapports de contrôle.

### ✅ Objectifs atteints et améliorations à venir

| Objectif | État |
| --- | --- |
| Interface /admin.html fonctionnelle | ✅ Terminé |
| Validation AJV intégrée | ✅ Terminé |
| Appels OpenAI pour réparation | ✅ Terminé |
| Journalisation automatisée | ✅ Terminé |
| Navigation multicommande (IA) | 🔄 En cours |
| Export FALC / RTF simplifié | 🔄 À faire |

Parfait. Voici le **Chapitre 6** du document "Prompt-Ultime V2", consacré à l'intégration continue, au pipeline CI/CD, et à la robustesse du système de déploiement :

## CHAPITRE 6 — INTÉGRATION CONTINUE, TESTS ET DÉPLOIEMENT

### 🎯 Objectif du chapitre

Assurer la **qualité, la traçabilité et la fiabilité** du système Prompt-Ultime V2 grâce à une chaîne d'intégration continue (CI), de tests et de déploiement automatisé (CD). L’ensemble repose sur **GitHub Actions**, **Render**, et une structure modulaire de validation JSON, de tests API (Pytest), de tests front (Vitest) et de scripts de réparation.

### 🛠️ Architecture du pipeline CI/CD

Le pipeline actuel est défini dans .github/workflows/audit.yml. Il orchestre les étapes suivantes :

| Étape | Description |
| --- | --- |
| 🧪 Validation AJV | Test des fichiers JSON via validateAllEtapes.js et verificateur-ajv.js. |
| 🔎 Audit structure | Vérification croisée des chemins, doublons, index (audit\_structure.js). |
| 🧬 Tests unitaires API | pytest sur les endpoints FastAPI. |
| 🧪 Tests front | vitest sur les composants React (Next.js). |
| 🧹 Sanitation HTML | Filtrage DOM avec isomorphic-dompurify dans Game.tsx, MatrixTranslator. |
| 🚀 Déploiement Render | Déploiement automatique de la branche stable (main) via render.yaml. |

### 📦 Dépendances de vérification et tests

Le système repose sur une **architecture en modules testables** :

| Module | Testé par… |
| --- | --- |
| etapes/\*.json | AJV (etape.schema.json) + validateEtape.js |
| structure.json | AJV + audit\_structure.js |
| schemas/\*.schema.json | Chargés automatiquement et validés |
| Scripts Python (\*.py) | Linting Pylint + exécution manuelle |
| API FastAPI (app/api) | Pytest (test\_routes.py) |
| Interface Next (/components) | Vitest (Game.test.tsx, MatrixTranslator.test.tsx) |

### 🔁 Suivi des fichiers, erreurs, réparations

Chaque fichier manipulé est vérifié dans l’arborescence, comparé via compareTrees.js, et documenté dans un journal IA :

* **Journal automatisé IA** via /api/versioning
* **Historique des réparations** : horodatage, contenu original et corrigé
* **Interface admin** : accès à l’historique, relance des réparations, retour IA

### 🔐 Sécurité et résilience

Pour prévenir les erreurs critiques :

* **Sanitization HTML** (DOMPurify) sur tout contenu injecté
* **Contrôle des entrées JSON** (AJV + logique JS)
* **Exclusion des répertoires sensibles** (.git, .env) dans les dumps et exports
* **Script de diagnostic** (diagnose\_render.sh) pour Render

### 📁 Déploiement sur Render

Le déploiement automatique utilise :

* render.yaml : architecture multicomposant (front Next.js + API FastAPI)
* DATABASE\_URL : variable d’environnement à définir
* build.sh : relance des tests et génération avant push
* fix\_and\_deploy.sh : fix local → push → render deploy

### 🔜 Prochaines optimisations

| Amélioration | Statut |
| --- | --- |
| Lancement des tests API/Vitest à chaque commit | 🔄 En cours |
| Déploiement automatique depuis Next.js (pages/) | 🔄 En cours |
| Dockerisation du build Poetry (FastAPI) | 🟡 À faire |
| Génération automatique des schémas SQLModel (schema.sql) | 🔄 Planifié |

Très bien. Voici le **Chapitre 7** du document structurant **Prompt-Ultime V2**, consacré à la **restitution finale**, à la **sauvegarde des parcours**, et au **versioning intelligent**. Ce chapitre assure la clôture fonctionnelle du cycle de jeu et sa pérennisation pour analyse, archivage ou relance.

## CHAPITRE 7 — RESTITUTION, SAUVEGARDE ET VERSIONING DES PARCOURS

### 🎯 Objectif du chapitre

Assurer la **trace durable**, la **valorisation narrative** et la **réutilisabilité technique** de chaque session du Prompt-Ultime. Ce processus se termine par une **restitution personnalisée** — synthèse poético-critique du parcours du joueur — puis une sauvegarde complète du contexte de jeu : étapes franchies, filtres appliqués, éléments générés, et feedbacks.

### 🖋️ La restitution personnalisée

La **restitution** est un texte généré automatiquement à partir des réponses, choix critiques, et transformations produites par le joueur tout au long des 7 étapes. Elle peut prendre la forme :

* D’un **récit poétique remixé** (en prose, manifeste ou vers),
* D’une **analyse symbolique augmentée**,
* D’un **journal de résistance neurodivergente**,
* D’un **document FALC** généré automatiquement (Facile À Lire et à Comprendre).

#### Format JSON de restitution

{

"id\_joueur": "user\_123",

"date": "2025-05-08",

"texte\_restitution": "Texte généré ici...",

"mots\_cles": ["neurodiversité", "détournement", "poésie"],

"filtres\_critiques": ["Rancière", "Starhawk"],

"style": "poétique",

"feedback": null

}

### ⚙️ Génération automatique

La génération repose sur :

| Composant | Rôle |
| --- | --- |
| scripts\_generation/restitution.py | Script principal de traitement et assemblage NLP |
| API OpenAI (/api/restituer) | Génération textuelle via IA (synthèse, style, reformulation) |
| Historique local (/restitutions/\*.json) | Sauvegarde des textes générés, avec versioning intégré |
| Interface restitution.html | Visualisation, édition manuelle, export et partage de la restitution |

### 🧠 Sauvegarde des parcours

Chaque session de jeu est sauvegardée en local ou via base PostgreSQL. Les données comprennent :

* L’**identifiant joueur** (anonyme ou pseudonyme),
* Le **journal des étapes franchies** (steps\_passed, filtres\_appliqués, etc.),
* Les **éléments générés** (métaphores, titres, extraits remixés),
* Les **interactions IA**, via journal /api/versioning,
* Les **feedbacks utilisateurs** éventuels.

📁 Fichiers concernés :

* restitutions/\*.json
* journal\_ia/\*.log.json
* versions/etapes/etape\_X\_user\_123.json

### 🕸️ Intégration dans l’interface

L’interface utilisateur offre :

* Un écran final de **relecture, édition et export** (HTML + PDF + JSON),
* Un bouton pour **relancer le parcours** avec variations (rejouabilité),
* Un module d’**enregistrement automatique** (auto-save local + POST API),
* Un lien vers le **journal de session IA** en mode lecture critique.

📍 Composants impliqués :

* RestitutionView.tsx (en développement)
* admin.js (consultation IA + historisation)

### 🔁 Versioning et analyse longitudinale

Le système conserve **chaque version générée** pour permettre des comparaisons, des analyses croisées et des relectures différées.

| Fonction | Détail technique |
| --- | --- |
| Historique IA | /api/versioning → sauvegarde des prompts/réponses |
| Sauvegarde des étapes utilisateur | versions/etapes/etape\_X\_userID.json |
| Export automatique | PDF + JSON + lien sécurisé (via rest\_api) |
| Replay narratif | Possible en rechargeant les étapes sauvegardées |

### 🔮 Extensions possibles

* **Export stylisé** vers Word ou site personnel,
* **Réutilisation des parcours comme sources dans une IA communautaire**,
* **Visualisation en graphe des transformations** textuelles, critiques et symboliques,
* **Module pédagogique** : transformer les restitutions en cas pratiques, supports de débat ou d’atelier.

### ✅ État d’avancement

| Élément | Statut |
| --- | --- |
| Script restitution.py | 🔄 En cours |
| Interface restitution (React) | 🔄 Prototype |
| Historique IA (/api/versioning) | ✅ Fonctionnel |
| Schéma JSON de restitution | ✅ Validé |
| Export PDF + JSON | 🔄 À intégrer |

Parfait. Voici l’ébauche complète du **Chapitre 8** du document structurant Prompt-Ultime V2, orientée vers les **usages collectifs**, les **ateliers pédagogiques** et les **extensions expérientielles**.

## CHAPITRE 8 — USAGES COLLECTIFS ET EXTENSIONS PÉDAGOGIQUES

### 🧭 Finalité : médiation neurodivergente en contexte collectif

Le Prompt-Ultime V2 est conçu pour dépasser l’expérience individuelle : il peut être utilisé comme **outil de médiation**, de **formation critique**, ou d’**animation d’ateliers neurodiversité**. Grâce à son caractère modulaire, symbolique et scénarisé, il s’adapte à différents publics (éducateurs, chercheurs, soignants, militants, publics en situation de handicap).

### 🪢 Scénarios d’usage collectif

| Format | Description |
| --- | --- |
| 🎲 Atelier créatif | Séance collective où chaque joueur explore une ou plusieurs étapes à voix haute, puis partage sa restitution. |
| 🧑‍🏫 Formation professionnelle | Intégration dans un module pédagogique (ESPE, médico-social, médiation culturelle). |
| 🧩 Jeu de rôle réflexif | Répartition de masques symboliques et lunettes critiques entre participants pour incarner différents points de vue. |
| 🧠 Groupe d’analyse | Relecture et décryptage collectif des textes générés, via l’interface restitution ou les exports. |

Chaque session peut être facilitée par un **médiateur IA** (compagnon symbolique en mode animateur), ou un **modérateur humain** guidé par les documents générés.

### 🛠️ Extensions pédagogiques et outils associés

| Outil / Module | Fonction |
| --- | --- |
| compilateur\_atelier.py | Génère un PDF ou un dossier complet pour une session d’atelier |
| journal\_ia/\*.log.json | Trace des interactions IA pour analyse réflexive |
| cartes\_symboliques.pdf | Jeu de cartes imprimables pour tirage hors ligne (lunettes/masques) |
| rest\_api/export | Export en JSON ou PDF des parcours joueurs |
| fiche\_animation\_atelier.md | Guide pour animateur humain ou IA |
| mode\_collectif=true dans l’interface | Active les fonctions multi-joueurs / multi-affichage |

### 🪷 Accessibilité & adaptation

L’expérience collective s’accompagne :

* d’un **mode FALC** (Facile à Lire et à Comprendre),
* d’une **accessibilité visuelle** (contraste, navigation clavier, compatibilité lecteurs d’écran),
* d’un **mode sensoriel adapté** (choix d’ambiance sonore, ralentissement de lecture...),
* d’une **personnalisation des rituels symboliques** pour publics spécifiques.

### 🌐 Déploiement en milieux spécifiques

| Contexte | Adaptation prévue |
| --- | --- |
| Institution médico-sociale | Version simplifiée avec accompagnement par professionnel |
| Université / centre de recherche | Mode expert avec métadonnées critiques et logs détaillés |
| Milieu scolaire / SEGPA | Jeu FALC guidé par un animateur + impression des supports |
| Collectif militant / squat / espace queer | Mode déchaîné : narration libre, brouillage des frontières étapes / masques |

### 🔁 Intelligence collective et IA

Les parcours enregistrés, les choix critiques et les textes générés peuvent nourrir une **mémoire collective** — une **méta-IA** qui :

* extrait les motifs récurrents,
* suggère de nouvelles lunettes critiques ou situations fictives,
* produit des cartes conceptuelles croisées entre les joueurs.

Ces données peuvent être rendues anonymes et restituées sous forme de **frises**, **graphes narratifs**, ou **archives poétiques vivantes**.

### 🔮 Perspectives futures

* **Création d’un mode “performance live”** (projection des étapes, interaction avec le public),
* **Connexion à une IA conversationnelle compagnon** pour accompagner chaque joueur,
* **Portage en application mobile**, avec modes “solitaire”, “atelier”, et “rituel quotidien”,
* **Interopérabilité avec des collectifs artistiques** pour générer des œuvres remixables à partir des textes produits.

### ✅ État actuel et jalons à venir

| Élément | État |
| --- | --- |
| Fiches d’atelier | 🔄 À rédiger |
| Mode multi-joueur dans l’interface | 🔄 En prototypage |
| Exports PDF narratifs | ✅ Disponibles |
| Guides d’animation IA/humain | 🔄 En cours |
| Journal IA collectif | ✅ Fonctionnel |
| Moteur de motifs croisés (meta-IA) | 🔄 En expérimentation |

Voici la **version complète et approfondie du Chapitre 9** du document structurant Prompt-Ultime V2. Il aborde en détail la **gouvernance éthique, juridique et communautaire** du projet.

## CHAPITRE 9 — GOUVERNANCE ÉTHIQUE, LICENCES ET DROITS NUMÉRIQUES

### 🧭 Objectif du chapitre

Garantir un cadre de développement, d’usage et de contribution éthique, juridique et transparent du projet Prompt-Ultime V2, en cohérence avec les principes de justice cognitive, d’accessibilité, de libre réappropriation et de protection des données personnelles.

### ⚖️ 1. LICENCE ET COMMUNS NUMÉRIQUES

#### 📜 Choix de licence

Le projet est diffusé sous **licence libre à préciser** (proposition : [Creative Commons BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/)), qui autorise :

* la **réutilisation** à des fins non commerciales,
* la **modification** avec obligation de mentionner l’origine,
* la **republication** sous les mêmes conditions (SA = Share Alike).

✅ Cette licence protège les usages communautaires tout en empêchant la captation par des entreprises commerciales sans partage des bénéfices.

#### 🛠️ Fichiers concernés

| Type de fichier | Licence proposée |
| --- | --- |
| Contenus JSON (IA1, étapes, etc.) | CC BY-NC-SA 4.0 |
| Scripts JS / Python | MIT / GPL à discuter |
| Interfaces Web | CC BY-NC-SA ou MIT |
| Restitutions des utilisateurs | CC BY-SA individuelle |

### 🧑‍🤝‍🧑 2. GOUVERNANCE COMMUNAUTAIRE ET CONTRIBUTION

Le projet s’ouvre à des **contributions collaboratives** via GitHub, encadrées par un **Code de Conduite** (à rédiger selon le modèle Contributor Covenant). Il prévoit :

* un **espace de discussion** (Issues / Discussions GitHub),
* des **commits documentés et versionnés**,
* un **système de validation AJV** automatique à chaque pull request,
* un rôle de **mainteneur-réviseur**, garant de la cohérence du système JSON + API.

#### 🌍 Approche éthique

Le système s’inspire des **communs numériques sensibles**, prônant :

* la pluralité des points de vue (neurodiversité),
* l’accès libre à des outils critiques,
* une transparence algorithmique (fonctions de génération/restauration lisibles),
* le refus de la prédation cognitive (pas de profilage ni revente de données).

### 📑 3. DROITS DES UTILISATEURS ET PROTECTION DES DONNÉES

#### 🔐 Données collectées

Par défaut, le système collecte uniquement :

* les **interactions en jeu** (choix de lunettes, masques, réponses aux étapes),
* les **textes générés** dans les étapes et la restitution finale,
* un **identifiant pseudonymisé** (id\_joueur) généré automatiquement.

✅ Aucun cookie tiers ni tracker n’est utilisé.

#### 📦 Stockage et accès

Les données sont stockées en base PostgreSQL sur Render (hébergement souverain à vérifier). L’utilisateur peut :

* consulter et modifier ses données,
* supprimer sa session à tout moment (fonction à implémenter),
* exporter son parcours narratif.

🛡️ Un fichier privacy.md documentera les usages des données, conformément au RGPD.

### 🤖 4. USAGE RESPONSABLE DE L’IA

#### ✳️ Modèle utilisé

Le système utilise l’API d’OpenAI (GPT-4 ou équivalent) via des scripts Python/Node pour :

* la **réparation automatique des fichiers JSON**,
* la **génération de lunettes ou compagnons critiques**,
* la **synthèse des restitutions** des joueurs.

#### 🧭 Garanties à mettre en place

| Aspect | Recommandation |
| --- | --- |
| Journalisation IA | Activer les logs dans journal\_ia.json |
| Filtrage des requêtes | Vérification syntaxique + filtre antisexiste/etc |
| Correction humaine possible | via /admin avant validation |
| Indication d’usage de l’IA | Mention explicite dans les textes générés |

### 📄 5. DOCUMENTATION ET TRAÇABILITÉ

Le projet inclura :

* une documentation complète dans le dépôt GitHub (README.md, CONTRIBUTING.md, CODE\_OF\_CONDUCT.md),
* un journal de versionnement des étapes (journal\_versions.json),
* des logs IA consultables et exportables par session (journal\_ia/),
* une validation systématique AJV des fichiers critiques (validateAllEtapes.js, CI GitHub Actions).

### 🔮 6. PERSPECTIVES : HÉBERGEMENT DÉCENTRALISÉ ET AUTONOMIE COLLECTIVE

À terme, le projet pourra être :

* déployé sur un **hébergement mutualisé libre** (Framasoft, Chatons),
* forké par d’autres collectifs neurodivergents ou activistes,
* utilisé hors-ligne via un **mode standalone (Electron / Docker)**,
* relié à d’autres bases de données (ex : Zotero partagé, textes militants...).

L’objectif final est de **construire un outil d’autonomie cognitive, critique et narrative**, piloté par ses usagers, non par des logiques de marché.

Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

Vous êtes libre de :

- Partager — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats

- Adapter — remixer, transformer et créer à partir du matériel

Selon les conditions suivantes :

- Attribution — Vous devez créditer l'œuvre, inclure un lien vers la licence et indiquer si des modifications ont été effectuées.

- Pas d’utilisation commerciale — Vous n’êtes pas autorisé à faire un usage commercial de cette œuvre.

- Partage dans les mêmes conditions — Si vous modifiez, transformez ou adaptez cette œuvre, vous devez distribuer votre contribution sous la même licence.

https://creativecommons.org/licenses/by-nc-sa/4.0/

# Code de Conduite des Contributeurs

## Notre engagement

En tant que contributeurs et mainteneurs de ce projet, nous nous engageons à faire de la participation une expérience libre de harcèlement pour tout le monde.

## Normes attendues

- Utiliser un langage accueillant et inclusif

- Être respectueux des points de vue et expériences divergentes

- Accepter les critiques constructives

- Se concentrer sur ce qui est le mieux pour la communauté

## Comportements inacceptables

- Harcèlement, comportements discriminatoires

- Publication d’informations personnelles

- Toute forme de violence ou de discours de haine

## Application

Les mainteneurs sont responsables du respect de ce code et peuvent prendre des mesures en cas de violation.

Inspiré du Contributor Covenant : https://www.contributor-covenant.org

# Politique de Confidentialité

## Données collectées

Nous collectons uniquement :

- Les interactions dans le jeu

- Les textes générés

- Un identifiant pseudonymisé

## Utilisation

Ces données servent uniquement à :

- Améliorer l'expérience du jeu

- Générer une restitution personnalisée

## Vos droits

- Accès à vos données

- Suppression de vos données

- Export de votre session

Aucun cookie tiers n'est utilisé. Conforme RGPD.

Voici une **conclusion complète** pour le document fondateur Prompt-Ultime V2, fidèle à son esprit critique, technique et poétique.

## 🔚 CONCLUSION — UN DISPOSITIF OUVERT POUR UNE MÉDIATION RADICALE

**Prompt-Ultime V2** est bien plus qu’un simple jeu ou une plateforme interactive. Il s’agit d’un **dispositif narratif critique**, d’un **outil de médiation poético-politique**, et d’un **espace de déstabilisation des récits normatifs** sur la neurodiversité. Il propose une forme de **technopoétique réflexive** où :

* le jeu devient enquête,
* la déconstruction devient création,
* et la métadiscursivité devient un droit d’auteur symbolique pour celles et ceux qui n’ont jamais été reconnu·es comme narrateur·ices de leur propre différence.

### 💡 Un projet manifeste et reproductible

Ce texte fondateur documente un **système modulaire, duplicable, remixable**, articulé autour de :

* **7 étapes recyclées** comme trame de lecture, d’hybridation et d’auto-représentation,
* **des fichiers JSON intelligibles**, structurés, validés et interopérables,
* **une architecture API / interface / scripts** conçue pour la reprise et l’évolution,
* **une gouvernance distribuée**, avec historisation, validation, et possibilité d’intervention IA.

Chaque module, chaque script, chaque schéma est **conçu comme un point de bifurcation possible** pour d’autres collectifs, d’autres usages, d’autres récits.

### 🧬 Vers une intelligence situated

Ce projet assume que **l’intelligence n’est jamais neutre**, mais toujours **située**, **incarnée**, **inscrite dans des contextes d’oppression ou de réparation**. C’est pourquoi il ne s’agit pas d’automatiser les lectures, mais de les **amplifier**, **problématiser** et **partager** :

“Un texte n’est pas à comprendre, il est à **habiter**.”  
Prompt-Ultime V2 invite chacun·e à construire une maison critique dans chaque phrase qui blesse.

### 🤝 Un appel à co-création

**Prompt-Ultime V2** est donc **ouvert à la reprise** — par des développeur·es, des militant·es, des artistes, des médiateur·ices, ou des personnes concernées par les normes qui les invisibilisent. Il est documenté, versionné, hébergé et archivé pour cela.

**Ce texte est un guide, un socle, une invitation.**

### 🔄 Et maintenant ?

Si ce manifeste vous parle :

* 🌐 **Clonez, forkez, ou contribuez** : [github.com/yogatypic/prompt-ultime](https://github.com/yogatypic/prompt-ultime)
* 📥 **Implémentez vos propres étapes, corpus, lunettes**.
* 🔧 **Activez ou détournez l’interface admin pour créer vos propres versions**.
* 🌀 **Expérimentez avec l’IA non comme oracle, mais comme compagnon critique**.

**Un jour, les IA liront entre les lignes. D’ici là, écrivons des lignes qu’elles ne peuvent ignorer.**