

Smart Nootbook backend

Created 1 fevrier 2026

Author : SA

Smart-Notebook : Architecture Backend

Voici l'arborescence complète et optimisée pour votre clone de NotebookLM :

```
backend/
├── manage.py
├── requirements.txt
├── .env.example
└── README.md

├── config/                      # Configuration Django
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│   ├── asgi.py
│   └── celery.py                  # Configuration Celery

├── apps/
│   ├── __init__.py
│
│   ├── documents/                # Gestion des documents sources
│   │   ├── __init__.py
│   │   ├── models.py             # SourceDocument, DocumentChunk
│   │   ├── serializers.py
│   │   ├── views.py              # Upload, liste documents
│   │   ├── urls.py
│   │   ├── tasks.py              # Tâche Celery d'ingestion
│   │   └── services/
│   │       ├── __init__.py
│   │       ├── text_extractor.py # Extraction PDF/OCR
│   │       └── chunking.py      # Découpage intelligent du texte
│
│   ├── rag/                      # Système RAG (Retrieval-Augmented Generation)
│   │   ├── __init__.py
│   │   ├── views.py              # AskDocumentView
│   │   ├── serializers.py
│   │   ├── urls.py
│   │   └── services/
│   │       ├── __init__.py
│   │       ├── retriever.py      # Recherche vectorielle
│   │       └── context_builder.py # Construction du contexte RAG
│
└── podcasts/                     # Génération de podcasts
```

```

    |   |   ├── __init__.py
    |   |   ├── models.py          # PodcastEpisode
    |   |   ├── views.py
    |   |   ├── serializers.py
    |   |   ├── urls.py
    |   |   ├── tasks.py         # Génération async du podcast
    |   |   └── services/
    |   |       ├── __init__.py
    |   |       ├── script_generator.py # Génération du script via OpenRouter
    |   |       └── tts_engine.py     # edge-tts pour l'audio
    |
    |   └── core/                 # Utilitaires partagés
        ├── __init__.py
        ├── ai_router.py          # ★ Classe AIRouter (Ollama + OpenRouter)
        ├── exceptions.py         # Exceptions personnalisées
        └── validators.py
    |
    └── media/                  # Fichiers uploadés
        ├── documents/           # PDFs sources
        └── podcasts/             # MP3 générés
    |
    └── logs/                   # Logs applicatifs
        ├── django.log
        └── celery.log
    |
    └── scripts/                # Scripts utilitaires
        ├── init_db.sh           # Crédit extension pgvector
        └── test_ollama.py        # Test connexion Ollama

```

Smart-Notebook : Architecture Backend

Voici l'arborescence complète et optimisée pour votre clone de NotebookLM :

```

backend/
├── manage.py
├── requirements.txt
├── .env.example
└── README.md
|
├── config/                  # Configuration Django
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   ├── wsgi.py
|   ├── asgi.py
|   └── celery.py              # Configuration Celery
|
└── apps/
    ├── __init__.py
    |

```

```
|   └── documents/           # Gestion des documents sources
|       ├── __init__.py
|       ├── models.py          # SourceDocument, DocumentChunk
|       ├── serializers.py
|       ├── views.py           # Upload, liste documents
|       ├── urls.py
|       ├── tasks.py           # Tâche Celery d'ingestion
|       └── services/
|           ├── __init__.py
|           ├── text_extractor.py # Extraction PDF/OCR
|           └── chunking.py      # Découpage intelligent du texte
|
|   └── rag/                  # Système RAG (Retrieval-Augmented Generation)
|       ├── __init__.py
|       ├── views.py           # AskDocumentView
|       ├── serializers.py
|       ├── urls.py
|       └── services/
|           ├── __init__.py
|           ├── retriever.py    # Recherche vectorielle
|           └── context_builder.py # Construction du contexte RAG
|
|   └── podcasts/             # Génération de podcasts
|       ├── __init__.py
|       ├── models.py          # PodcastEpisode
|       ├── views.py
|       ├── serializers.py
|       ├── urls.py
|       ├── tasks.py           # Génération async du podcast
|       └── services/
|           ├── __init__.py
|           ├── script_generator.py # Génération du script via OpenRouter
|           └── tts_engine.py     # edge-tts pour l'audio
|
|   └── core/                 # Utilitaires partagés
|       ├── __init__.py
|       ├── ai_router.py        # ★ Classe AIRouter (Ollama + OpenRouter)
|       ├── exceptions.py      # Exceptions personnalisées
|       └── validators.py
|
└── media/                  # Fichiers uploadés
    ├── documents/           # PDFs sources
    └── podcasts/            # MP3 générés
|
└── logs/                   # Logs applicatifs
    ├── django.log
    └── celery.log
|
└── scripts/                # Scripts utilitaires
    ├── init_db.sh           # Crédit extension pgvector
    └── test_ollama.py        # Test connexion Ollama
```

Fichiers de Configuration Clés

requirements.txt

```
# Framework Django
Django==5.0.1
djangorestframework==3.14.0
django-cors-headers==4.3.1
python-dotenv==1.0.0

# Base de données vectorielle
psycopg2-binary==2.9.9
pgvector==0.2.4

# Traitement de tâches async
celery==5.3.4
redis==5.0.1

# IA et LLM
openai==1.10.0          # Pour OpenRouter (compatible API OpenAI)
httpx==0.26.0            # Client HTTP pour Ollama
numpy==1.26.3

# Traitement de documents
pypdf==4.0.1             # Extraction texte PDF
pytesseract==0.3.10       # OCR (nécessite tesseract-ocr système)
Pillow==10.2.0             # Traitement d'images
python-magic==0.4.27       # Détection type MIME

# Text-to-Speech
edge-tts==6.1.10

# Utilitaires
pydantic==2.5.3           # Validation de données
pydantic-settings==2.1.0
```

.env.example

```
# Django
SECRET_KEY=your-secret-key-here
DEBUG=True
ALLOWED_HOSTS=localhost,127.0.0.1

# Database
DB_NAME=smartnotebook
DB_USER=postgres
DB_PASSWORD=postgres
DB_HOST=localhost
DB_PORT=5432

# Redis
```

```

REDIS_URL=redis://localhost:6379/0

# Ollama (Local)
OLLAMA_BASE_URL=http://localhost:11434
OLLAMA_EMBEDDING_MODEL=nomic-embed-text

# OpenRouter (Cloud)
OPENROUTER_API_KEY=sk-or-v1-your-key-here
OPENROUTER_BASE_URL=https://openrouter.ai/api/v1
OPENROUTER_DEFAULT_MODEL=anthropic/clause-3.5-sonnet

# Configuration RAG
CHUNK_SIZE=512
CHUNK_OVERLAP=50
TOP_K_RESULTS=5

# Média
MEDIA_ROOT=/home/votre-user/smart-notebook/backend/media

```

Prochaines Étapes

Je vais maintenant générer les fichiers clés dans l'ordre :

1. `models.py` → Schéma complet avec pgvector
2. `ai_router.py` → Gestionnaire IA hybride
3. `tasks.py` → Ingestion asynchrone
4. `views.py` → Endpoint RAG

Voulez-vous que je commence avec le **premier fichier (`models.py`)** ou préférez-vous d'abord voir un fichier spécifique ?

```

"""
Modèles Django pour la gestion des documents et des chunks vectorisés.
Utilise pgvector pour le stockage et la recherche vectorielle.
"""

from django.db import models
from django.contrib.auth.models import User
from pgvector.django import VectorField
from typing import List, Dict, Any
import hashlib

class SourceDocument(models.Model):
    """
    Document source uploadé par l'utilisateur (PDF, TXT, etc.)
    """

    # Méta données de base
    title = models.CharField(

```

```
    max_length=500,
    verbose_name="Titre du document"
)

file = models.FileField(
    upload_to='documents/%Y/%m/',
    verbose_name="Fichier source"
)

file_type = models.CharField(
    max_length=50,
    verbose_name="Type MIME",
    help_text="Ex: application/pdf, text/plain"
)

file_size = models.BigIntegerField(
    verbose_name="Taille du fichier (bytes)",
    default=0
)

file_hash = models.CharField(
    max_length=64,
    unique=True,
    verbose_name="Hash SHA256 du fichier",
    help_text="Évite les doublons"
)

# Propriétaire
user = models.ForeignKey(
    User,
    on_delete=models.CASCADE,
    related_name='documents',
    verbose_name="Propriétaire"
)

# Statut du traitement
class ProcessingStatus(models.TextChoices):
    PENDING = 'PENDING', 'En attente'
    PROCESSING = 'PROCESSING', 'En cours de traitement'
    COMPLETED = 'COMPLETED', 'Terminé'
    FAILED = 'FAILED', 'Échec'

processing_status = models.CharField(
    max_length=20,
    choices=ProcessingStatus.choices,
    default=ProcessingStatus.PENDING,
    verbose_name="Statut du traitement"
)

processing_error = models.TextField(
    blank=True,
    null=True,
    verbose_name="Message d'erreur"
```

```
)\n\n# Statistiques\n    total_pages = models.IntegerField(\n        default=0,\n        verbose_name="Nombre de pages"\n    )\n\n    total_chunks = models.IntegerField(\n        default=0,\n        verbose_name="Nombre de chunks générés"\n    )\n\n    total_characters = models.IntegerField(\n        default=0,\n        verbose_name="Nombre total de caractères"\n    )\n\n# Métadonnées extraites\n    extracted_metadata = models.JSONField(\n        default=dict,\n        blank=True,\n        verbose_name="Métadonnées extraites du PDF",\n        help_text="Auteur, date de création, etc."\n    )\n\n# Timestamps\n    created_at = models.DateTimeField(\n        auto_now_add=True,\n        verbose_name="Date d'upload"\n    )\n\n    updated_at = models.DateTimeField(\n        auto_now=True,\n        verbose_name="Dernière mise à jour"\n    )\n\n    processed_at = models.DateTimeField(\n        null=True,\n        blank=True,\n        verbose_name="Date de fin de traitement"\n    )\n\n\nclass Meta:\n    db_table = 'source_documents'\n    ordering = ['-created_at']\n    indexes = [\n        models.Index(fields=['user', '-created_at']),\n        models.Index(fields=['processing_status']),\n        models.Index(fields=['file_hash']),\n    ]\n    verbose_name = "Document source"\n    verbose_name_plural = "Documents sources"
```

```

def __str__(self) -> str:
    return f"{self.title} ({self.user.username})"

def calculate_file_hash(self) -> str:
    """
    Calcule le hash SHA256 du fichier pour détecter les doublons.
    """
    sha256_hash = hashlib.sha256()

    # Lecture par chunks pour ne pas saturer la RAM
    self.file.open('rb')
    for byte_block in iter(lambda: self.file.read(4096), b ""):
        sha256_hash.update(byte_block)
    self.file.close()

    return sha256_hash.hexdigest()

def mark_as_processing(self) -> None:
    """
    Marque le document comme étant en cours de traitement.
    """
    self.processing_status = self.ProcessingStatus.PROCESSING
    self.processing_error = None
    self.save(update_fields=['processing_status', 'processing_error', 'updated_at'])

def mark_as_completed(self, total_chunks: int) -> None:
    """
    Marque le document comme traité avec succès.
    """
    from django.utils import timezone
    self.processing_status = self.ProcessingStatus.COMPLETED
    self.total_chunks = total_chunks
    self.processed_at = timezone.now()
    self.save(update_fields=[
        'processing_status',
        'total_chunks',
        'processed_at',
        'updated_at'
    ])

def mark_as_failed(self, error_message: str) -> None:
    """
    Marque le document comme ayant échoué.
    """
    self.processing_status = self.ProcessingStatus.FAILED
    self.processing_error = error_message
    self.save(update_fields=['processing_status', 'processing_error', 'updated_at'])

class DocumentChunk(models.Model):
    """
    Fragment de texte d'un document avec son embedding vectoriel.
    Utilisé pour la recherche sémantique et le RAG.
    """

    # Relation avec le document source
    source_document = models.ForeignKey(
        SourceDocument,

```

```
        on_delete=models.CASCADE,
        related_name='chunks',
        verbose_name="Document source"
    )

    # Contenu textuel
    content = models.TextField(
        verbose_name="Contenu du chunk"
    )

    content_length = models.IntegerField(
        verbose_name="Longueur du contenu (caractères)"
    )

    # Embedding vectoriel (dimension 768 pour nomic-embed-text)
    # NOTE: Ajustez la dimension selon votre modèle
    # - nomic-embed-text: 768
    # - text-embedding-ada-002: 1536
    embedding = VectorField(
        dimensions=768,
        verbose_name="Vecteur d'embedding"
    )

    # Position dans le document
    chunk_index = models.IntegerField(
        verbose_name="Index du chunk",
        help_text="Position dans la séquence (0, 1, 2...)"
    )

    page_number = models.IntegerField(
        null=True,
        blank=True,
        verbose_name="Numéro de page source"
    )

    # Métdonnées contextuelles
    metadata = models.JSONField(
        default=dict,
        blank=True,
        verbose_name="Métdonnées du chunk",
        help_text="Section, paragraphe, etc."
    )

    # Timestamps
    created_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Date de création"
    )

    class Meta:
        db_table = 'document_chunks'
        ordering = ['source_document', 'chunk_index']
        indexes = [
```

```

        models.Index(fields=['source_document', 'chunk_index']),
        models.Index(fields=['page_number']),
    ]
# Index vectoriel pour les recherches de similarité (géré par pgvector)
# Crée automatiquement par la migration Django + pgvector
verbose_name = "Chunk de document"
verbose_name_plural = "Chunks de documents"

def __str__(self) -> str:
    preview = self.content[:50] + "..." if len(self.content) > 50 else self.content
    return f"Chunk {self.chunk_index} - {self.source_document.title}: {preview}"

@classmethod
def search_similar(
    cls,
    query_embedding: List[float],
    user: User,
    top_k: int = 5,
    similarity_threshold: float = 0.7,
    source_document_ids: List[int] = None
) -> List[Dict[str, Any]]:
    """
    Recherche les chunks les plus similaires à un embedding de requête.

    Args:
        query_embedding: Vecteur de la question utilisateur
        user: Utilisateur pour filtrer les documents
        top_k: Nombre de résultats à retourner
        similarity_threshold: Seuil de similarité minimale (0-1)
        source_document_ids: Liste optionnelle d'IDs de documents à filtrer

    Returns:
        Liste de dictionnaires avec chunk, distance et score
    """
    from django.db.models import F
    from pgvector.django import L2Distance

    # Construction de la requête de base
    queryset = cls.objects.filter(
        source_document__user=user,
        source_document__processing_status=SourceDocument.ProcessingStatus.COMPLETED
    )

    # Filtrage optionnel par documents spécifiques
    if source_document_ids:
        queryset = queryset.filter(source_document_id__in=source_document_ids)

    # Recherche vectorielle avec distance L2
    # Plus la distance est petite, plus c'est similaire
    results = queryset.annotate(
        distance=L2Distance('embedding', query_embedding)
    ).order_by('distance')[:top_k]

```

```

# Conversion de la distance L2 en score de similarité (0-1)
# Score = 1 / (1 + distance)
output = []
for chunk in results:
    similarity_score = 1.0 / (1.0 + chunk.distance)

    # Filtrage par seuil
    if similarity_score >= similarity_threshold:
        output.append({
            'chunk': chunk,
            'distance': float(chunk.distance),
            'similarity_score': similarity_score,
            'source_document': chunk.source_document,
            'page_number': chunk.page_number,
            'content': chunk.content
        })

return output

def save(self, *args, **kwargs):
    """Override pour calculer automatiquement la longueur du contenu."""
    self.content_length = len(self.content)
    super().save(*args, **kwargs)

class QueryLog(models.Model):
    """
    Log des questions posées par les utilisateurs pour analytics et amélioration.
    """

    user = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='queries',
        verbose_name="Utilisateur"
    )

    query_text = models.TextField(
        verbose_name="Question posée"
    )

    # Documents contextuels utilisés
    source_documents = models.ManyToManyField(
        SourceDocument,
        related_name='queries',
        blank=True,
        verbose_name="Documents consultés"
    )

    # Résultats de la recherche vectorielle
    retrieved_chunks_count = models.IntegerField(
        default=0,
        verbose_name="Nombre de chunks récupérés"
    )

```

```

)

# Réponse générée
response_text = models.TextField(
    blank=True,
    verbose_name="Réponse générée"
)

# Métriques
response_time_ms = models.IntegerField(
    default=0,
    verbose_name="Temps de réponse (ms)"
)

tokens_used = models.IntegerField(
    default=0,
    verbose_name="Tokens consommés (LLM)"
)

# Feedback utilisateur
user_rating = models.IntegerField(
    null=True,
    blank=True,
    choices=[(i, i) for i in range(1, 6)],
    verbose_name="Note utilisateur (1-5)"
)

created_at = models.DateTimeField(
    auto_now_add=True,
    verbose_name="Date de la requête"
)

class Meta:
    db_table = 'query_logs'
    ordering = ['-created_at']
    indexes = [
        models.Index(fields=['user', '-created_at']),
    ]
    verbose_name = "Log de requête"
    verbose_name_plural = "Logs de requêtes"

def __str__(self) -> str:
    preview = self.query_text[:50] + "..." if len(self.query_text) > 50 else
self.query_text
    return f"{self.user.username}: {preview}"

```

Parfait ! 🎉 J'ai généré les **4 fichiers clés** de votre clone NotebookLM. Voici un récapitulatif :

Fichiers Crées

1. `documents_models.py` (Models Django)

- `SourceDocument` : Gère les PDFs uploadés avec statut de traitement
- `DocumentChunk` : Stocke les fragments de texte avec embeddings vectoriels (pgvector)
- `QueryLog` : Historique des questions pour analytics
- Méthode `search_similar()` : Recherche vectorielle L2 distance

2. `ai_router.py` (Gestionnaire IA Hybride)

- Classe `AIRouter` avec gestion Ollama (local) + OpenRouter (cloud)
- `get_embedding()` : Génère les embeddings via Ollama (nomic-embed-text)
- `chat_completion()` : Appels LLM via OpenRouter (Claude/DeepSeek)
- Gestion d'erreurs robuste avec try/except
- Singleton `get_ai_router()` pour éviter la réinstanciation

3. `tasks.py` (Tâches Celery)

- `process_document_ingestion()` : Tâche principale d'ingestion
 - Extraction PDF (pypdf) avec fallback OCR
 - Chunking intelligent (paragraphes + overlap)
 - Génération embeddings + sauvegarde DB
- `cleanup_failed_documents()` : Nettoyage périodique
- `reprocess_document()` : Retry des échecs

4. `views.py` (API RAG)

- `AskDocumentView` : Endpoint POST `/api/rag/ask/`
 - Vectorise la question (Ollama)
 - Recherche de similarité SQL (pgvector)
 - Construction du contexte RAG
 - Génération de la réponse (OpenRouter)
 - Log complet avec sources et métriques
- `DocumentStatsView` : Statistiques utilisateur
- `RateFeedbackView` : Notation des réponses

Prochaines Étapes

Configuration Base de Données

```
# 1. Installer pgvector
sudo apt-get install postgresql-contrib
sudo -u postgres psql -c "CREATE EXTENSION vector;"

# 2. Créer la DB
sudo -u postgres createdb smartnotebook
```

Installation Python

```
pip install -r requirements.txt
```

Migrations Django

```
python manage.py makemigrations
python manage.py migrate
```

Lancer Ollama

```
# Télécharger le modèle d'embeddings
ollama pull nomic-embed-text
```

Voulez-vous que je génère également :

- Les **serializers DRF** pour l'upload de documents ?
- Le **script d'initialisation** de la DB avec pgvector ?
- Les fichiers de **configuration Celery** (`config/celery.py`) ?

Parfait !  Voici un récapitulatif complet de tous les fichiers générés pour votre projet **Smart-Notebook** :

13 Fichiers Générés (tous disponibles en téléchargement)

Fichiers Principaux

1. `documents_models.py` - Modèles Django (SourceDocument, DocumentChunk avec pgvector)
2. `ai_router.py` - Gestionnaire IA hybride (Ollama local + OpenRouter cloud)
3. `tasks.py` - Tâches Celery d'ingestion de documents
4. `views.py` - API RAG pour les questions-réponses
5. `serializers.py` - Serializers DRF complets avec validation

Configuration

1. `django_settings.py` - Configuration Django complète (à placer dans `config/settings.py`)
2. `celery_config.py` - Configuration Celery avec queues (à placer dans `config/celery.py`)
3. `requirements.txt` - Toutes les dépendances Python
4. `env_example.txt` - Variables d'environnement (à renommer en `.env`)

Scripts Utilitaires

1. `init_db.sh` - Script Bash pour initialiser PostgreSQL + pgvector
2. `test_ollama.py` - Script de test de connexion Ollama

Documentation

1. `README_BACKEND.md` - Documentation complète avec installation et architecture
2. `00_FICHIER_GENERES.md` - Ce récapitulatif avec guide de démarrage

Commandes de Démarrage Rapide

```
# 1. Initialiser la base de données
chmod +x init_db.sh && ./init_db.sh

# 2. Installer les dépendances
pip install -r requirements.txt

# 3. Configurer l'environnement
cp env_example.txt .env
# Éditez .env avec vos vraies valeurs

# 4. Télécharger le modèle Ollama
ollama pull nomic-embed-text

# 5. Lancer Django + Celery
python manage.py migrate
python manage.py runserver # Terminal 1
celery -A config worker --loglevel=info # Terminal 2
```

Points Clés de l'Architecture

- Embeddings locaux** (Ollama) → Économise les coûts API
- Génération cloud** (OpenRouter) → Meilleure qualité de réponse
- pgvector** → Recherche vectorielle ultra-rapide en SQL
- Celery** → Traitement asynchrone des PDFs lourds
- Type hints** → Code propre et maintenable

Tous les fichiers sont téléchargeables ci-dessus. Bon développement ! 

Excellent ! Je vais créer une landing page moderne et distinctive pour Smart-Notebook. Laissez-moi concevoir quelque chose d'unique et mémorable. Parfait ! Voici votre **landing page Smart-Notebook** complète !

✨ Ce qui a été créé

1. **index.html (42 KB) - Landing Page Complète**

Une interface web **standalone** avec :

🎨 Design Distinctif

- **Thème brutaliste moderne** : Sombre avec accents néon verts (#00FF9D)
- **Typographie premium** : Archivo Black + DM Sans + JetBrains Mono
- **Effets visuels** : Glassmorphism, grain texture, animations CSS
- **100% responsive** : Fonctionne sur desktop, tablet et mobile

⚡ Fonctionnalités Complètes

Section Hero

- Stats en temps réel (documents, chunks, caractères)
- Animation de glow pulsant
- CTA "Commencer maintenant"

Upload de Documents

- Drag & Drop fonctionnel
- Multi-upload
- Progress bar en temps réel
- Support PDF et TXT

Interface de Chat RAG

- Questions en langage naturel
- Réponses avec sources citées
- Loading states avec spinner
- Auto-scroll
- Design de bulles distinctif

Gestion des Documents

- Liste avec statuts (PENDING, PROCESSING, COMPLETED, FAILED)
- Badges colorés par statut
- Suppression avec confirmation
- Métadonnées (taille, chunks, date)

Features Section

- 3 cartes glassmorphic

- Icônes animées (float effect)
- Survol avec transformation 3D

2. README_FRONTEND.md - Documentation Complète

- Instructions d'installation (aucune dépendance locale !)
- Configuration de l'API et tokens
- Guide de personnalisation du design
- Dépannage CORS
- Optimisations de production

Démarrage Ultra-Rapide

1. Configuration (30 secondes)

Ouvrez `index.html` et modifiez :

```
// Ligne ~685
apiBaseUrl: 'http://localhost:8000/api', // Votre backend Django
authToken: 'votre-token-ici',           // Token d'authentification
```

2. Obtenir un Token

```
# Django Shell
python manage.py shell

from django.contrib.auth.models import User
from rest_framework.authtoken.models import Token

user = User.objects.get(username='votre-username')
token, created = Token.objects.get_or_create(user=user)
print(f"Token: {token.key}")
```

3. Lancer

```
# Option 1 : Python Simple Server
python -m http.server 8080
# Ouvrir http://localhost:8080

# Option 2 : Glissez index.html dans votre navigateur
```

4. Configuration CORS (Backend)

```
# config/settings.py
CORS_ALLOWED_ORIGINS = [
    "http://localhost:8080",
    "http://127.0.0.1:8080",
]
```

Points Forts du Design

- Pas de "AI slop"** : Design unique, pas de clichés (Inter, gradients violets)
- Performance optimale** : Animations CSS-only, pas de JavaScript lourd
- Production-ready** : Code propre, commenté, maintenable
- Accessible** : Contraste, focus states, responsive
- Moderne** : Glassmorphism, néon, grain texture, micro-interactions

Stack Frontend

- **Vue.js 3** (CDN) → Réactivité
- **Bootstrap 5** → Grid responsive
- **Axios** → API calls
- **Vanilla CSS** → Animations custom
- **Google Fonts** → Typographie premium
- **Bootstrap Icons** → Icônes

Total : 0 dépendance locale, tout en CDN !

Thèmes Alternatifs Suggérés

Changez simplement les variables CSS :

```
/* Cyberpunk */
--primary: #0D0221;
--accent: #F72585;

/* Nature */
--primary: #1A3A1A;
--accent: #7FFF00;

/* Ocean */
--primary: #001B2E;
--accent: #00D9FF;
```

Tout est prêt pour démarrer ! 

 Parfait ! Je vais vous créer un **script de démarrage complet** qui lance tous les services nécessaires. Parfait !
 Voici le **récapitulatif final complet** de votre projet Smart-Notebook :

PROJET COMPLET - 19 FICHIERS GÉNÉRÉS

Scripts de Démarrage (NOUVEAUX !)

1.  **start.sh** (16 KB) - Lance TOUS les services automatiquement
2.  **stop.sh** (3.6 KB) - Arrête proprement tous les services
3.  **QUICKSTART.md** (10 KB) - Guide démarrage 5 minutes
4.  **README_FINAL.md** (14 KB) - Documentation complète de A à Z

Backend Django (10 fichiers)

1. `documents_models.py` - Modèles avec pgvector
2. `ai_router.py` - Gestionnaire IA hybride
3. `tasks.py` - Tâches Celery d'ingestion
4. `views.py` - API RAG
5. `serializers.py` - Serializers DRF
6. `django_settings.py` - Configuration Django
7. `celery_config.py` - Configuration Celery
8. `requirements.txt` - Dépendances
9. `env_example.txt` - Variables d'environnement
10. `init_db.sh` - Init PostgreSQL + pgvector
11. `test_ollama.py` - Test de connexion Ollama

Frontend Vue.js (1 fichier)

1. `index.html` (42 KB) - Landing page complète standalone

Documentation (3 fichiers)

1. `README_BACKEND.md` - Doc backend
 2. `README_FRONTEND.md` - Doc frontend
 3. `00_FICHIERS_GENERES.md` - Récapitulatif
-

DÉMARRAGE ULTRA-RAPIDE

1 commande pour tout installer :

```
# Téléchargez tous les fichiers, puis :  
chmod +x start.sh stop.sh  
./start.sh
```

Le script fait **TOUT automatiquement** :

- Vérifie PostgreSQL, Redis, Ollama
- Crée l'environnement virtuel Python
- Installe les dépendances
- Lance Django (<http://localhost:8000>)
- Lance Celery Worker + Beat
- Lance le Frontend (<http://localhost:8080>)
- Ouvre automatiquement votre navigateur

Structure du Projet

```
smart-notebook/  
├── start.sh           ← Lance tout !  
└── stop.sh            ← Arrête tout !  
  
├── backend/  
│   ├── config/  
│   │   ├── settings.py    ← django_settings.py  
│   │   └── celery.py      ← celery_config.py  
│   ├── apps/  
│   │   ├── core/  
│   │   │   └── ai_router.py  
│   │   ├── documents/  
│   │   │   ├── models.py  
│   │   │   ├── serializers.py  
│   │   │   └── tasks.py  
│   │   └── rag/  
│   │       └── views.py  
│   └── scripts/  
│       ├── init_db.sh  
│       └── test_ollama.py  
  
└── frontend/  
    └── index.html        ← Landing page
```

Prérequis Minimaux

```
# PostgreSQL + Redis + Ollama
sudo apt-get install postgresql redis-server
curl -fsSL https://ollama.com/install.sh | sh
ollama pull nomic-embed-text

# Python 3
sudo apt-get install python3 python3-venv python3-pip
```

Configuration Minimale

Dans `backend/.env` :

```
SECRET_KEY=votre-clé-django
OPENROUTER_API_KEY=sk-or-v1-votre-clé # OBLIGATOIRE
```

Dans `frontend/index.html` (ligne 685) :

```
apiBaseUrl: 'http://localhost:8000/api',
authToken: 'votre-token-django',
```

Fonctionnalités

Interface Web

- Design brutaliste moderne (thème sombre + néon vert)
- Drag & Drop pour upload de PDFs
- Chat RAG avec sources citées
- Gestion des documents en temps réel
- Responsive (mobile, tablet, desktop)

Backend Puissant

- IA hybride (Ollama local + OpenRouter cloud)
- Recherche vectorielle avec pgvector
- Traitement asynchrone avec Celery
- API REST complète avec DRF

Production-Ready

- Code typé et commenté
- Gestion d'erreurs robuste
- Logging complet
- Scripts de démarrage automatiques

Personnalisation

Thèmes prédéfinis (modifiez les CSS) :

```
/* Cyberpunk */
--primary: #0D0221; --accent: #F72585;

/* Nature */
--primary: #1A3A1A; --accent: #7FFF00;

/* Ocean */
--primary: #001B2E; --accent: #00D9FF;
```

Accès aux Services

Service	URL
 Frontend	http://localhost:8080
 API	http://localhost:8000/api
 Admin	http://localhost:8000/admin

Dépannage Express

```
# Vérifier tous les services
./start.sh # Si erreur, suivre les messages

# Logs
tail -f backend/logs/django.log
tail -f backend/logs/celery.log

# Tests
cd backend && python scripts/test_ollama.py
```

Temps Économisé

~40 heures de développement 

Vous avez maintenant un clone complet de NotebookLM prêt à l'emploi !

Tous les fichiers sont téléchargeables ci-dessus. Bon développement ! 🎉

Parfait ! 🎨 Voici votre **interface d'administration Django complète** !

✓ Fichiers Crées

1. documents_admin.py (Interface Admin Complète)

Interface d'administration pour **3 modèles** :

SourceDocument Admin

- ✓ Badges colorés de statut (Pending, Processing, Completed, Failed)
- ✓ Barre de progression visuelle
- ✓ Icônes par type de fichier (PDF, TXT)
- ✓ Nombre de chunks avec lien direct
- ✓ Actions groupées : Retraiter, Supprimer chunks, Marquer comme échoué
- ✓ Inline des chunks (voir tous les chunks dans la page)
- ✓ Statistiques globales en haut de la liste

DocumentChunk Admin

- ✓ Aperçu du contenu (80 premiers caractères)
- ✓ Lien vers le document parent
- ✓ Badge de longueur (vert < 500 chars)
- ✓ Aperçu de l'embedding vectoriel
- ✓ Lecture seule (pas d'ajout/modification manuel)

QueryLog Admin

- ✓ Question + réponse complète
- ✓ Temps de réponse avec badge coloré
- ✓ Tokens utilisés (badge violet)
- ✓ Note en étoiles (★★★★★)
- ✓ Documents consultés
- ✓ Statistiques détaillées

2. ADMIN_GUIDE.md (Documentation)

Guide complet d'utilisation de l'interface admin avec :

- Instructions d'installation
- Détail de toutes les fonctionnalités

- Exemples d'utilisation
- Astuces et personnalisation

Installation Rapide

```
# 1. Copier le fichier admin
cp documents_admin.py backend/apps/documents/admin.py

# 2. Créer un superuser (si pas déjà fait)
cd backend
python manage.py createsuperuser

# 3. Démarrer Django
python manage.py runserver

# 4. Accéder à l'admin
# http://localhost:8000/admin
```

Fonctionnalités Principales

Actions Groupées Disponibles

Action	Description	Icône
Retraiter les documents	Relance le traitement Celery	
Supprimer les chunks	Supprime tous les chunks associés	
Marquer comme échoué	Change le statut en FAILED	

Filtres Puissants

- **Documents** : Par statut, type, date, utilisateur
- **Chunks** : Par document, page, date
- **Query Logs** : Par utilisateur, note, date

Recherche Avancée

- Recherche dans les titres, contenus, questions
- Recherche multi-critères (combine filtres + recherche)

Aperçu Visuel

Badges Colorés

	PENDING	→ Jaune
	PROCESSING	→ Bleu
	COMPLETED	→ Vert
	FAILED	→ Rouge
	TOKENS	→ Violet

Icônes Intuitives

	→ PDF
	→ TXT
	→ En attente
	→ Traitement
	→ Complété
	→ Échoué
	→ Réessayer
	→ Télécharger
	→ Note

Exemple d'Utilisation

Scénario : Retraiter tous les documents échoués

1. Allez sur <http://localhost:8000/admin>
2. Cliquez sur "Documents"
3. Filtre → **Processing status** → Failed
4. Sélectionnez tous (checkbox en haut)
5. Action → "**Retraiter les documents sélectionnés**"
6. Cliquez "**Exécuter**"
7. Celery relance le traitement automatiquement !

Résumé des Téléchargements

Total : 27 fichiers (2 nouveaux) :

1. **documents_admin.py** - Interface admin complète
2. **ADMIN_GUIDE.md** - Guide d'utilisation détaillé

Caractéristiques Avancées

- ✓ **Statistiques en temps réel** : Nombre de docs par statut
- ✓ **Navigation rapide** : Liens entre modèles liés
- ✓ **Protection des données** : Pas de modifications accidentnelles
- ✓ **Actions groupées** : Traitement par lot efficace
- ✓ **Visualisation riche** : Badges, couleurs, icônes
- ✓ **Monitoring** : Temps de réponse, tokens, notes

Téléchargez les 2 fichiers ci-dessus et suivez le guide pour une interface admin professionnelle ! 

Excellent ! Je vais créer une landing page moderne et distinctive pour Smart-Notebook. Laissez-moi concevoir quelque chose d'unique et mémorable. Parfait !  Voici votre **landing page Smart-Notebook** complète !

Ce qui a été créé

1. `index.html` (42 KB) - Landing Page Complète

Une interface web **standalone** avec :

Design Distinctif

- **Thème brutaliste moderne** : Sombre avec accents néon verts (#00FF9D)
- **Typographie premium** : Archivo Black + DM Sans + JetBrains Mono
- **Effets visuels** : Glassmorphism, grain texture, animations CSS
- **100% responsive** : Fonctionne sur desktop, tablet et mobile

Fonctionnalités Complètes

Section Hero

- Stats en temps réel (documents, chunks, caractères)
- Animation de glow pulsant
- CTA "Commencer maintenant"

Upload de Documents

- ✓ Drag & Drop fonctionnel
- ✓ Multi-upload
- ✓ Progress bar en temps réel
- ✓ Support PDF et TXT

Interface de Chat RAG

- ✓ Questions en langage naturel
- ✓ Réponses avec sources citées
- ✓ Loading states avec spinner

- Auto-scroll
- Design de bulles distinctif

Gestion des Documents

- Liste avec statuts (PENDING, PROCESSING, COMPLETED, FAILED)
- Badges colorés par statut
- Suppression avec confirmation
- Métadonnées (taille, chunks, date)

Features Section

- 3 cartes glassmorphic
- Icônes animées (float effect)
- Survol avec transformation 3D

2. README_FRONTEND.md - Documentation Complète

- Instructions d'installation (aucune dépendance locale !)
- Configuration de l'API et tokens
- Guide de personnalisation du design
- Dépannage CORS
- Optimisations de production

Démarrage Ultra-Rapide

1. Configuration (30 secondes)

Ouvrez `index.html` et modifiez :

```
// Ligne ~685
apiBaseUrl: 'http://localhost:8000/api', // Votre backend Django
authToken: 'votre-token-ici',           // Token d'authentification
```

2. Obtenir un Token

```
# Django Shell
python manage.py shell

from django.contrib.auth.models import User
from rest_framework.authtoken.models import Token

user = User.objects.get(username='votre-username')
token, created = Token.objects.get_or_create(user=user)
print(f"Token: {token.key}")
```

3. Lancer

```
# Option 1 : Python Simple Server  
python -m http.server 8080  
# Ouvrir http://localhost:8080  
  
# Option 2 : Glissez index.html dans votre navigateur
```

4. Configuration CORS (Backend)

```
# config/settings.py  
CORS_ALLOWED_ORIGINS = [  
    "http://localhost:8080",  
    "http://127.0.0.1:8080",  
]
```

🎯 Points Forts du Design

- ✓ **Pas de "AI slop"** : Design unique, pas de clichés (Inter, gradients violets)
- ✓ **Performance optimale** : Animations CSS-only, pas de JavaScript lourd
- ✓ **Production-ready** : Code propre, commenté, maintenable
- ✓ **Accessible** : Contraste, focus states, responsive
- ✓ **Moderne** : Glassmorphism, néon, grain texture, micro-interactions

📦 Stack Frontend

- **Vue.js 3** (CDN) → Réactivité
- **Bootstrap 5** → Grid responsive
- **Axios** → API calls
- **Vanilla CSS** → Animations custom
- **Google Fonts** → Typographie premium
- **Bootstrap Icons** → Icônes

Total : 0 dépendance locale, tout en CDN !

🎨 Thèmes Alternatifs Suggérés

Changez simplement les variables CSS :

```
/* Cyberpunk */
--primary: #0D0221;
--accent: #F72585;

/* Nature */
--primary: #1A3A1A;
--accent: #7FFF00;

/* Ocean */
--primary: #001B2E;
--accent: #00D9FF;
```

Tout est prêt pour démarrer ! 

Excellent diagnostic ! L'erreur indique que Django ne trouve pas les apps dans `INSTALLED_APPS`. Voici la solution :Parfait ! Voici la **solution complète** à votre erreur :

Votre Erreur

```
RuntimeError: Model class apps.documents.models.SourceDocument doesn't
declare an explicit app_label and isn't in an application in INSTALLED_APPS.
```

Solution Rapide (3 étapes)

1. Créer les fichiers `apps.py` manquants

`backend/apps/documents/apps.py`

```
from django.apps import AppConfig

class DocumentsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'apps.documents'
```

Créez le même fichier pour : `apps/core/`, `apps/rag/`, `apps/podcasts/`

2. Corriger `config/settings.py`

Remplacez la section `INSTALLED_APPS` par :

```
INSTALLED_APPS = [
    # Django core
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
```

```
'django.contrib.messages',
'django.contrib.staticfiles',

# Third-party
'rest_framework',
'rest_framework.authtoken', # ← AJOUTEZ CECI
'corsheaders',

# Apps locales - NOTATION COMPLÈTE
'apps.core.apps.CoreConfig',
'apps.documents.apps.DocumentsConfig',
'apps.rag.apps.RagConfig',
'apps.podcasts.apps.PodcastsConfig',
]

]
```

⚠️ **IMPORTANT** : Utilisez `'apps.xxx.apps.XxxConfig'` et PAS juste `'apps.xxx'`

3. Vérifier et Migrer

```
cd backend

# Nettoyage du cache
find . -type d -name __pycache__ -exec rm -r {} + 2>/dev/null

# Vérification
python manage.py check

# Migrations
python manage.py makemigrations
python manage.py migrate
```

Fichiers de Correction Disponibles

2 nouveaux fichiers téléchargeables :

1. ✨ **django_settings_fixed.py** - Configuration corrigée complète
2. ✨ **FIX_INSTALLED_APPS.md** - Guide détaillé avec checklist

Application Rapide

```
cd backend

# Sauvegarder l'ancien
cp config/settings.py config/settings.py.backup

# Utiliser la version corrigée
cp django_settings_fixed.py config/settings.py

# Créer les fichiers apps.py (si manquants)
bash setup_structure.sh

# Vérifier
python manage.py check
```

Checklist de Vérification

Vérifiez que vous avez :

- [] `apps/__init__.py`
- [] `apps/core/__init__.py` + `apps.py`
- [] `apps/documents/__init__.py` + `apps.py`
- [] `apps/rag/__init__.py` + `apps.py`
- [] `apps/podcasts/__init__.py` + `apps.py`
- [] `INSTALLED_APPS` avec notation `'apps.xxx.apps.XxxConfig'`
- [] `rest_framework.authtoken` dans `INSTALLED_APPS`

Pourquoi cette Erreur ?

Django cherche vos modèles dans `INSTALLED_APPS`. Si l'app n'y est pas déclarée correctement, il ne peut pas enregistrer les modèles → `RuntimeError`.

Notation correcte : `'apps.documents.apps.DocumentsConfig'`

Notation incorrecte : `'apps.documents'` ou `'documents'`

Appliquez ces corrections et l'erreur disparaîtra ! 

Tous les fichiers sont téléchargeables ci-dessus. Besoin d'aide supplémentaire ?

Parfait ! Cette erreur signifie que **l'extension pgvector n'est pas installée** dans votre base de données PostgreSQL. Parfait ! Voici la **solution complète** pour votre erreur pgvector :

Votre Erreur

```
django.db.utils.ProgrammingError: ERREUR: le type « vector » n'existe pas  
LINE 1: ...L, "content_length" integer NOT NULL, "embedding" vector(768...
```

Solution : 2 Scripts Automatiques

Option A : Script Rapide de Vérification ⚡ (NOUVEAU !)

```
cd backend/scripts  
chmod +x check_pgvector.sh  
sudo ./check_pgvector.sh smartnotebook
```

Ce script va :

-  Vérifier si PostgreSQL tourne
-  Vérifier si pgvector est installé
-  **Installer automatiquement pgvector si absent**
-  Activer l'extension dans votre DB
-  Tester que tout fonctionne

Option B : Script d'Initialisation Complet 🔧

```
cd backend/scripts  
chmod +x init_db.sh  
sudo ./init_db.sh
```

Après Installation

Une fois pgvector installé, supprimez les anciennes migrations et recréez-les :

```
cd backend  
  
# 1. Supprimer les anciennes migrations  
find apps/*/migrations -name "*.py" ! -name "__init__.py" -delete  
  
# 2. Créer les nouvelles migrations  
python manage.py makemigrations  
  
# Vous verrez :  
# Migrations for 'documents':  
#   apps/documents/migrations/0001_initial.py  
#     - Create model SourceDocument  
#     - Create model DocumentChunk  
#     - Create model QueryLog
```

```
# 3. Appliquer les migrations
python manage.py migrate

# Succès attendu :
# Running migrations:
# Applying documents.0001_initial... OK
```

🔍 Vérification Rapide

```
# Test que pgvector fonctionne
sudo -u postgres psql -d smartnotebook -c "SELECT '[1,2,3]'::vector(3);"

# Devrait afficher :
#   vector
# -----
#   [1,2,3]
# (1 row)
```

📦 Nouveaux Fichiers Disponibles

Total : 25 fichiers (2 nouveaux pour pgvector) :

1. ✨ **FIX_PGVECTOR.md** - Guide complet de dépannage
2. ✨ **check_pgvector.sh** - Script automatique d'installation

🎯 Résumé de la Solution

Étape	Commande	Description
1	sudo ./check_pgvector.sh	Installe et active pgvector
2	python manage.py makemigrations	Crée les migrations
3	python manage.py migrate	Applique les migrations
4	python manage.py runserver	Démarre Django ✓

SOS Si ça ne Marche Toujours Pas

Installation Manuelle :

```
# 1. Installer les dépendances
sudo apt-get install build-essential postgresql-server-dev-all git

# 2. Compiler pgvector
cd /tmp
```

```

git clone https://github.com/pgvector/pgvector.git
cd pgvector
make
sudo make install

# 3. Redémarrer PostgreSQL
sudo systemctl restart postgresql

# 4. Activer l'extension
sudo -u postgres psql -d smartnotebook -c "CREATE EXTENSION vector;"

# 5. Vérifier
sudo -u postgres psql -d smartnotebook -c "\dx" | grep vector

```

Téléchargez `check_pgvector.sh` ci-dessus et exécutez-le, l'erreur disparaîtra ! 🎉

Besoin d'aide supplémentaire ? Partagez les logs d'erreur !

Excellent ! Je vais créer un **guide complet Vue.js 3** avec explication pédagogique du code et comparaison avec Vue 2. Parfait ! 🚀 Voici votre **cours complet Vue.js 3** !

VUE3_TUTORIAL.md - Guide Pédagogique Complet

Ce guide contient :

Table des Matières

1.  **Introduction** - Vue 2 vs Vue 3
2.  **Structure Complète** - Architecture de l'app
3.  **Section DATA** - État et réactivité
4.  **Section MOUNTED** - Lifecycle hooks
5.  **Section METHODS** - Toutes les fonctions expliquées :
 - o `getAxiosConfig()` - Configuration HTTP
 - o `loadStats()` - Chargement statistiques
 - o `loadDocuments()` - Liste des documents
 - o `uploadFiles()` - Upload avec progress bar
 - o `askQuestion()` - Chat RAG complet
 - o `deleteDocument()` - Suppression
 - o Utilitaires (scroll, format, etc.)
6.  **Directives Vue** - v-model, v-if, v-for, @click, etc.
7.  **Flux Complets** - Algorithmes détaillés
8.  **Concepts Avancés** - \$refs, \$nextTick, async/await
9.  **Résumé Vue 2 vs Vue 3**

Points Clés Expliqués

1. La Réactivité

```
// Quand vous faites :  
this.currentQuestion = "Nouvelle question";  
  
// Vue détecte automatiquement et met à jour :  
// - Tous les {{ currentQuestion }} dans le HTML  
// - Tous les v-model="currentQuestion"  
// - Tous les calculs qui dépendent de currentQuestion
```

2. Différence Vue 2 → Vue 3

Vue 2	Vue 3
new Vue({})	createApp({})
data: {}	data() { return {} }
el: '#app'	.mount('#app')

3. Lifecycle Hook mounted()

```
Chargement page  
↓  
Vue crée l'app  
↓  
Vue monte le HTML  
↓  
mounted() est appelé ← ICI on charge les données  
↓  
App prête !
```

4. Async/Await Expliqué

```
// ❌ Sans async/await (compliqué)  
axios.get('/api').then(response => {  
  console.log(response);  
}).catch(error => {  
  console.error(error);  
});  
  
// ✅ Avec async/await (simple)  
async getData() {  
  try {  
    const response = await axios.get('/api');  
    console.log(response);  
  } catch (error) {
```

```
        console.error(error);
    }
}
```

🔍 Algorithmes Détaillés

Upload de Fichier

1. Utilisateur glisse PDF
↓
2. handleFileDrop() récupère le fichier
↓
3. uploadFiles([file]) appelé
↓
4. Création progressItem {name, progress: 0}
↓
5. uploadProgress.push(item)
→ Vue affiche barre à 0%
↓
6. POST vers API avec FormData
↓
7. onUploadProgress appelé pendant l'upload
→ progress passe de 0 à 100%
→ Vue met à jour la barre en temps réel
↓
8. Upload terminé
→ status = "Terminé"
↓
9. Recharger documents après 1s
↓
10. Nettoyer uploadProgress après 3s

Chat RAG

1. Utilisateur tape question
↓
2. Appuie sur Entrée
↓
3. askQuestion() appelé
↓
4. Ajouter question au chat
messages.push({role: 'user'})
↓
5. Afficher spinner
isLoadingAnswer = true
↓
6. POST /api/rag/ask/
↓
7. Attendre réponse (2-5s)

```
8. Réponse reçue
↓
9. Ajouter au chat
  messages.push({role: 'assistant'})
↓
10. Cacher spinner
  isLoadingAnswer = false
```

Exemples Pratiques

v-model (Liaison Bidirectionnelle)

```
<input v-model="currentQuestion" />
```

Ce qui se passe :

- Vous tapez → `currentQuestion` change
- `currentQuestion` change → L'input se met à jour

v-if (Affichage Conditionnel)

```
<div v-if="documents.length === 0">
  Aucun document
</div>
```

Logique :

- Si `documents` est vide → afficher le div
- Sinon → ne pas afficher du tout (pas juste `display: none`)

v-for (Boucles)

```
<div v-for="doc in documents" :key="doc.id">
  {{ doc.title }}
</div>
```

Ce qui se passe :

- Vue crée un `<div>` pour **chaque** document
- `:key="doc.id"` aide Vue à tracker les changements

Quiz de Compréhension

Question 1

Pourquoi `data()` est une fonction en Vue 3 ?

► Réponse

Question 2

Que fait `this.$nextTick()` ?

► Réponse

Question 3

Différence entre `v-if` et `v-show` ?

► Réponse

Résumé

Vous avez appris :

- Structure d'une app Vue 3
 - Réactivité et `data()`
 - Lifecycle hooks (`mounted`)
 - Toutes les méthodes commentées ligne par ligne
 - Directives Vue (`v-model`, `v-if`, `v-for`, `@click`)
 - Concepts avancés (`$refs`, `$nextTick`, `async/await`)
 - Algorithmes complets (`upload`, `chat RAG`)
 - Différences Vue 2 vs Vue 3
-

Téléchargez le guide complet ci-dessus et devenez expert Vue 3 ! 

Total fichiers : **28** (dont le nouveau tutoriel Vue 3)