

**Sujet**

**Système de Génération de Contenus Audio et Visuel par IA**

**Soutenu par :**

**Bendadi Mohamed**

**Bougalla Abdelghani**

**Boukhsim Nabil**

**ER-Rami Youssef**

Mémoire de Projet de Fin d’Etudes

Pour l’Obtention du Diplôme BAC+3 en

Ingénierie des Systèmes d’Information

Année Universitaire 2024-2025

**Sous la direction de :**

**Mme Rachidi Rabia**

**DEDICACE***À nos familles, pour le soutien constant qu’ils nous ont témoigné, leur confiance et leurs encouragements tout le long de notre parcours de formation.*

*À nos parents, pour leur investissement à mettre en œuvre tous les moyens nécessaires pour que nous soyons dans les meilleures dispositions possibles pour réussir.*

*À nos frères et sœurs, pour leur tendresse et leur bonne humeur exceptionnelles en toute occasion pour rendre moins anxiogenes les moments de stress.*

Re

Mer

**Remerciements**

Nous souhaitons adresser notre plus vive gratitude a Madame Rachidi Rabia, notre professeure encadrante, pour sa disponibilité, son sens pédagogique, l’attention portée à ses étudiants ainsi que les précieux conseils apportés pour chaque étape de notre travail. À l’ensemble du corps enseignant de la filière Ingénierie des Systèmes Informatiques, pour la qualité de la formation et les bases tant techniques que méthodologiques qu’elle nous a permis d’acquérir.

Et a tout nos amis et professeurs , pour leurs aide sur les aspects implication du déploiement cloud et les retours d’expérience. À l’ensemble de notre promotion, pour l’entraide, les échanges d’idées riches et la relecture commune qui ont été les moteurs de l’élaboration de cette rédaction.

À nos familles, pour l’amour, l’encouragement permanent et la confiance qui nous ont permis de faire aboutir notre projet. Chacun, par son aide et ses conseils, contribue ainsi à faire de ce PFC un succès , nous leur en exprimons notre plus vive gratitude.

Résumé

Dans un contexte ou la création de contenus multimédias doit concilier qualité, rapidité et personnalisation, le projet Creative AI propose une plateforme auto hébergée dédiée a la synthèse et au clonage de voix. En s’appuyant sur des modèles de pointe open source StyleTTS2 pour la synthèse vocale et Seed‑VC pour la conversion de voix zero‑shot nous avons conçu un pipeline complet permettant a tout utilisateur de générer , affiner et stocker des fichiers audio de haute qualité sans dépendre d’API externes.

La première étape du projet a consisté a fine‑tuner StyleTTS2 sur des corpus ciblés, visant un Mean Opinion Score (MOS) supérieur à 4 sur 5 et une naturalité proche de l’enregistrement humain. Parallèlement, Seed‑VC a été adapté pour assurer une conversion fluide entre voix sources et cibles, même avec des échantillons de quelques secondes. Le backend FastAPI orchestre ces services IA, tandis qu’un système de tâches asynchrones basé sur Inngest garantit une scalabilité optimale. L’ensemble est conteneurisé avec Docker et déployé sur AWS EC2, avec stockage des médias sur S3 et gestion des métadonnées dans une base PostgreSQL.

L’interface web intuitive (Next.js + NextAuth.js) permettant la saisie de texte ou l’upload d’un fichier audio, le choix d’un style vocal, et la récupération d’un lien sécurisé vers le fichier généré. Ce projet démontre la faisabilité d’une solution on premise ou cloud privée, offrant maitrise des données, personnalisation avancée et respect des exigences RGPD.

Mots‑clés : synthèse vocale, clonage de voix, StyleTTS2, Seed‑VC, fine‑tuning, auto‑hébergement, IA multimedia

Abstract

In an era where multimedia content creation demands a balance of quality, speed, and customization, the Creative AI project delivers a self‑hosted platform dedicated to voice synthesis and cloning. Leveraging cutting‑edge open‑source models—StyleTTS2 for text‑to‑speech and Seed‑VC for zero‑shot voice conversion—we built a comprehensive pipeline that allows users to generate, fine‑tune, and store high‑quality audio files without relying on external APIs.

The first phase involved fine‑tuning StyleTTS2 on targeted datasets to achieve a Mean Opinion Score (MOS) above 4/5 and human‑like naturalness. Concurrently, Seed‑VC was adapted to ensure seamless voice conversion between source and target speakers using only a few seconds of reference audio. A FastAPI backend orchestrates these AI services, while an Inngest‑based asynchronous task system ensures optimal scalability. The entire solution is containerized with Docker and deployed on AWS EC2, with media stored in S3 and metadata managed in a PostgreSQL database.

The outcome is an intuitive web interface (Next.js + NextAuth.js) enabling text input or audio upload, selection of a target voice style, and retrieval of a secure download link. This project showcases the feasibility of an on‑premise or private‑cloud solution that provides full data control, advanced customization, and compliance with GDPR requirements.

Keywords : text‑to‑speech, voice cloning, StyleTTS2, Seed‑VC, fine‑tuning, self‑hosting, multimedia AI

Liste des abréviations

|  |  |
| --- | --- |
| **Mots** | **Signification** |
| AES-256 | Advanced Encryption Standard (chiffrement symétrique, 256 bits) |
| API | Application Programming Interface (interface de programmation d’applications) |
| AWS | Amazon Web Services |
| CAPEX | Capital Expenditure (dépenses d’investissement) |
| CI | Continuous Integration (intégration continue) |
| DB | Database (base de données) |
| DCGAN | Deep Convolutional Generative Adversarial Network (GAN convolutif profond) |
| DSP | Digital Signal Processing (traitement numérique du signal) |
| d-vector | Discriminative Vector (type d’embedding vocal) |
| DTW | Dynamic Time Warping (alignement temporel dynamique) |
| EC2 | Elastic Compute Cloud (serveurs virtuels AWS) |
| ECR | Elastic Container Registry (registre de conteneurs AWS) |
| FID | Fréchet Inception Distance (métrique de qualité d’images générées) |
| GAN | Generative Adversarial Network (réseau antagoniste génératif) |
| HMM | Hidden Markov Model (modèle de Markov caché) |
| HTTPS | HyperText Transfer Protocol Secure |
| IAM | Identity and Access Management (gestion des accès AWS) |
| I/O | Input/Output (entrée/sortie) |
| JSON | JavaScript Object Notation (format de données) |
| JWT | JSON Web Token (jeton d’authentification) |
| MP4 | MPEG-4 Part 14 (format de fichier vidéo) |
| MOS | Mean Opinion Score (score moyen d’opinion) |
| MP4 | Format de fichier vidéo MPEG-4 |
| OOM | Out Of Memory (erreur mémoire) |
| OPEX | Operational Expenditure (dépenses opérationnelles) |
| ORM | Object-Relational Mapping (mapping objet-relationnel) |
| PNG | Portable Network Graphics (format d’image) |
| PSNR | Peak Signal-to-Noise Ratio (mesure de qualité vidéo) |
| RDS | Relational Database Service (base de données managée AWS) |
| RMS | Root Mean Square (mesure d’amplitude audio) |
| S3 | Simple Storage Service (stockage objet AWS) |
| SFX | Sound Effects (effets sonores) |
| seq2seq | Sequence-to-Sequence (architecture de réseaux de neurones) |
| SQL | Structured Query Language (langage de requête pour bases de données) |
| SV2TTS | Speaker Verification to Text-to-Speech |
| TACITRON 2 | (modèle TTS seq2seq + WaveNet développé par Google) |
| TLS | Transport Layer Security |
| TTS | Text-to-Speech (synthèse vocale) |
| URL | Uniform Resource Locator |
| UUID | Universally Unique Identifier |
| WAVENET | Vocodeur neuronal pour génération audio réaliste |

Table des figures

Figure[1: Comparaison des performances par type de génération 22](#_Toc201923187)

Figure [2 Diagramme de cas d’utilisation du système Creative AI 32](#_Toc201923188)

Figure [3 Diagramme de séquence : génération de contenu IA 34](#_Toc201923189)

Figure [4 Diagramme de séquence : abonnement et paiement 37](#_Toc201923190)

Figure [5 Diagramme d’activités : génération de contenu IA 40](#_Toc201923191)

Figure [6 Diagramme de classes du système Creative AI 44](#_Toc201923192)

Figure [7 Diagramme d’états : cycle de vie d’un abonnement 48](#_Toc201923193)

Figure [8 Diagramme d’états – Génération de contenu IA 51](#_Toc201923194)

Figure [9 Diagramme de déploiement de la plateforme Creative AI 55](#_Toc201923195)

Figure [10 Figure Console AWS S3 (stockage) 58](#_Toc201923196)

Figure [11 Figure– Console AWS ECR (registre d’images Docker) 58](#_Toc201923197)

Figure [12 Figure – Instance EC2 (serveur PyTorch) 59](#_Toc201923198)

Figure [13 Diagramme de communication : génération de contenu audio 59](#_Toc201923199)

Figure [14 Backend : Pipenv 65](#_Toc201923200)

Figure [15 Frontend : npm 65](#_Toc201923201)

Figure [16 Structure du projet 66](#_Toc201923202)

Figure [17 Chargement du modèle et cycle de vie 67](#_Toc201923203)

Figure [18 Sécurisation par API Key 68](#_Toc201923204)

Figure [19 Endpoint de génération TTS 69](#_Toc201923205)

Figure [20 Structure du projet 70](#_Toc201923206)

Figure [21 Page d’accueil 1 70](#_Toc201923207)

Figure [22 Page d’accueil 2 71](#_Toc201923208)

Figure [23 Tableau de bord 72](#_Toc201923209)

Figure [24 Media Library 73](#_Toc201923210)

Figure [26 Explore 1 75](#_Toc201923212)

Figure [27 Explore 2 75](#_Toc201923213)

Figure [28 Générateur Text-to-Speech 1 76](#_Toc201923214)

Figure [29 Générateur Text-to-Speech 2 77](#_Toc201923215)

Figure [30 Générateur Voice-to-Voice 1 78](#_Toc201923216)

Figure [31 Générateur Voice-to-Voice 2 78](#_Toc201923217)

Figure [32 Générateur Text-to-Sound Effect 1 80](#_Toc201923218)

Figure [33 Générateur Text-to-Sound Effect 2 80](#_Toc201923219)

Figure [34 Générateur Text-to-Image 81](#_Toc201923220)

Figure [35 Générateur Text & Image-to-Video 1 82](#_Toc201923221)

Figure [36 Générateur Text & Image-to-Video 2 82](#_Toc201923222)

Figure [37 Facturation 84](#_Toc201923223)

Figure [38 Paramètres 84](#_Toc201923224)

Figure [39 Écran de connexion 85](#_Toc201923225)

Figure [40 Écran d’inscription 86](#_Toc201923226)

Figure [41 Configuration Inngest 87](#_Toc201923227)

Figure [42 Définition du job TTS 88](#_Toc201923228)

Figure [43 Courbes d’entraînement (train vs val) montrant la convergence de la perte L1. 94](#_Toc201923229)

Figure [44 Graphique de perte 98](#_Toc201923230)

Figure [45 Système multi-services conteneurisé 111](#_Toc201923231)

Figure [46 Chargement optimisé des modèles 113](#_Toc201923232)

Figure [47 Workflow complet voix → voix : 115](#_Toc201923233)

Figure [48 Intégration StyleTTS2 116](#_Toc201923234)

Figure [49 Fichier Docker optimisé 118](#_Toc201923235)

Figure [50 Point d'entrée unifié 120](#_Toc201923236)

Figure [52 Processus de fine-tuning 123](#_Toc201923238)

Figure [53 Fichier Docker Compose 125](#_Toc201923239)

Figure [54 Endpoints critiques 126](#_Toc201923240)

Liste des tableaux

Tableau [1 Liste des abéviations 5](#_Toc3562)

Tableau [2 Étude comparative des solutions existantes 26](#_Toc22417)

Tableau [3 Principaux cas d’utilisation 32](#_Toc13884)

Tableau [4 Responsabilités principales 43](#_Toc1790)

Tableau [5 hyperparamètres 97](#_Toc27397)

Tableau [6 Tableau MOS 102](#_Toc25741)

Tableau [7 Tableau MOS 104](#_Toc15483)

Tableau [8 MOS conversion 106](#_Toc32197)

Tableau [9 test clés 110](#_Toc32010)

Tableau [10 Résultats types 113](#_Toc22030)

Tableau [11 Résultats MOS 115](#_Toc22007)

Tableau [12 Tableau comparatif global 116](#_Toc16500)

Table des matières

[Résumé 3](#_Toc29340)

[Abstract 4](#_Toc5111)

[Liste des abréviations 5](#_Toc16274)

[Table des figures 7](#_Toc14504)

[Liste des tableaux 10](#_Toc6373)

[Table des matières 11](#_Toc6181)

[Chapitre 1 Introduction Générale 15](#_Toc29489)

[1.1 Contexte et enjeux des contenus multimédias IA 15](#_Toc28053)

[1.2 Problématique et motivation 16](#_Toc23038)

[1.3 Objectifs du projet 18](#_Toc18299)

[1. Auto-hébergement et portabilité 18](#_Toc22155)

[2. Personnalisation et qualité 18](#_Toc5043)

[3. Conformité RGPD et sécurité 18](#_Toc2657)

[4. *E*xpérience utilisateur et accessibilité 19](#_Toc5212)

[1.4 Méthodologie et organisation du rapport 19](#_Toc775)

[1. Analyse et specifications : 19](#_Toc19740)

[2. Conception UML et architecture : 19](#_Toc14543)

[3. Implémentation et fine-tuning : 19](#_Toc13140)

[4. Tests et validation : 20](#_Toc30871)

[5. Rédaction et soutenance 20](#_Toc331)

[Chapitre 2 – État de l’art 21](#_Toc14568)

[2.1 Synthèse vocale (TTS) 21](#_Toc19287)

[2.2 Clonage vocal 23](#_Toc7402)

[2.3 Génération d’effets sonores 24](#_Toc12161)

[2.4 Génération d’images et de vidéos 25](#_Toc15353)

[2.5 Étude comparative des solutions existantes 26](#_Toc31561)

[Chapitre 3 — Spécifications et conception fonctionnelle 28](#_Toc15074)

[3.1 Cahier des charges fonctionnel et non-fonctionnel 28](#_Toc16488)

[A. Besoins fonctionnels 28](#_Toc27669)

[B. Besoins non-fonctionnels 30](#_Toc22724)

[3.2 Diagramme de cas d’utilisation 31](#_Toc5704)

[3.3 Diagrammes de séquence 34](#_Toc6902)

[3.3.1 Diagramme de séquence — Génération de contenu IA 35](#_Toc15719)

[3.3.2 Diagramme de séquence — Processus de paiement 38](#_Toc19477)

[3.4 Diagramme d’activités 41](#_Toc26559)

[Chapitre 4 - Architecture logicielle et infrastructure 45](#_Toc149)

[4.1 Diagramme de classes 45](#_Toc27078)

[Explication du diagramme 46](#_Toc28039)

[Choix de modélisation 48](#_Toc21750)

[Objectif du diagramme 48](#_Toc29825)

[4.2 Diagramme d’états 49](#_Toc19082)

[4.2.1 Diagramme d’états : cycle de vie d’un abonnement 49](#_Toc18405)

[4.2.2 Diagramme d’états – Génération de contenu IA  53](#_Toc14626)

[4.3 Diagramme de déploiement 57](#_Toc12913)

[4.4 Diagramme de communication 61](#_Toc1182)

[Chapitre 5 – Implémentation 67](#_Toc9274)

[5.1 Environnement de développement 67](#_Toc20908)

[5.2 Développement du backend FastAPI 69](#_Toc30841)

[5.2.1 Structure du projet 69](#_Toc15063)

[5.2.2 Chargement du modèle et cycle de vie 70](#_Toc17115)

[5.2.3 Sécurisation par API Key 71](#_Toc8998)

[5.2.4 Endpoint de génération TTS 72](#_Toc10287)

[5.3 Intégration Next.js et NextAuth.js 73](#_Toc14980)

[5.3.1 Structure du projet 73](#_Toc9639)

[5.4 Orchestration Inngest 93](#_Toc770)

[5.4.1 Configuration Inngest 93](#_Toc28264)

[5.4.2 Définition du job TTS 93](#_Toc32232)

[Chapitre 6 – Fine-tuning et entraînement des modèles 95](#_Toc768)

[6.1 Préparation des datasets 95](#_Toc7902)

[6.1.1 Collecte des enregistrements 95](#_Toc14340)

[6.1.2 Nettoyage et normalisation 95](#_Toc7110)

[6.1.3 Tokenisation et phonétisation 96](#_Toc17043)

[6.1.4 Fractionnement train / validation / test 96](#_Toc30945)

[6.1.5 Pré-calcul des spectrogrammes 97](#_Toc13063)

[6.2 Fine-tuning de StyleTTS2 97](#_Toc5591)

[6.2.2 Stratégie de gel et dégel des couches (Layer Freezing) 99](#_Toc79)

[6.2.3 Fonction de perte composite 99](#_Toc29721)

[6.2.4 Protocole d’entraînement 100](#_Toc31574)

[6.2.5 Évaluation 100](#_Toc2363)

[6.2.3 Fine-tuning de Seed-VC 102](#_Toc21098)

[6.2.3 Configuration des hyperparamètres 104](#_Toc17719)

[Conclusion du Chapitre 6 107](#_Toc19702)

[Chapitre 7 - Tests et évaluation 108](#_Toc29626)

[7.1 Tests fonctionnels 108](#_Toc27159)

[7.1.1 Objectifs 108](#_Toc6381)

[7.1.2 Scénarios de test clés 110](#_Toc8406)

[7.2 Tests de performance 112](#_Toc11598)

[7.2.1 Objectifs 112](#_Toc14078)

[7.2.2 Métriques clés 112](#_Toc31130)

[7.2.3 Résultats types 113](#_Toc3055)

[7.3 Évaluation perceptive (MOS) 114](#_Toc6881)

[7.3.1 Objectifs 114](#_Toc24297)

[7.3.2 Protocole MOS 114](#_Toc14377)

[7.3.3 Résultats MOS 115](#_Toc5263)

[7.4 Benchmarks comparatifs 115](#_Toc32343)

[7.4.1 Objectifs 115](#_Toc24000)

[7.4.2 Solutions comparées 116](#_Toc23606)

[7.4.3 Tableau comparatif global 116](#_Toc29170)

[Tableau 7.4 – Benchmarks comparatifs 118](#_Toc1999)

[7.4.4 Analyse 118](#_Toc25015)

[Chapitre 8 – Déploiement et exploitation 120](#_Toc24435)

[8.1 Architecture Globale 120](#_Toc27155)

[8.2 Initialisation des Modèles  122](#_Toc3110)

[8.3 Pipeline de Clonage Vocal 124](#_Toc14278)

[8.4 Synthèse Texte-Voix avec Style 126](#_Toc14411)

[8.5 Déploiement Docker 127](#_Toc16833)

[8.6 API d'Intégration 129](#_Toc1213)

[8.7 Gestion des Ressources GPU 131](#_Toc23642)

[8.8 Workflow d'Entraînement 133](#_Toc30661)

[8.9 Stratégie de Déploiement Production 135](#_Toc29429)

[8.10 Monitoring et Sécurité 137](#_Toc15485)

[Conclusion et perspectives 140](#_Toc1098)

**Introduction General**

Creative AI s’inscrit au cœur d’une dynamique où l’intelligence artificielle révolutionne la création et la diffusion de contenus multimédias, en répondant tout particulièrement aux besoins du marché marocain. Face à la diversité linguistique – entre espagnole, anglais et français – et à la montée en puissance des plateformes digitales locales, notre plateforme auto-hébergée propose un écosystème complet de synthèse vocale, de clonage de voix, et de génération d’images et de vidéos. En s’appuyant sur des modèles open-source de pointe (StyleTTS2, Seed-VC, Make-An-Audio, Stable Diffusion, VideoGen), Creative AI permet de produire en quelques secondes des ressources audio et visuelles personnalisées, adaptées aux spécificités culturelles et linguistiques du Royaume. Cette solution garantit une autonomie totale en matière de souveraineté des données, tout en favorisant l’inclusion et l’accessibilité grâce à des fonctionnalités multilingues et conformes aux standards d’accessibilité. Elle offre également aux studios de production, aux agences marketing et aux établissements éducatifs un levier technique modulable pour prototyper et déployer rapidement des applications innovantes, contribuant ainsi à la vitalité de l’écosystème technologique marocain

**Chapitre 1**

**Fondements du Projet**

**Creative AI**

Dans ce chapitre, nous posons le cadre général du projet Creative AI en présentant d’abord le contexte de la création de contenus multimédias par IA et les enjeux spécifiques du marché marocain. Nous exposons ensuite la problématique (coûts, dépendance aux API externes, questions éthiques) avant de détailler les objectifs techniques, fonctionnels et qualitatifs qui structureront l’étude.

1. **Fondements du projet creative AI**

## Contexte et enjeux des contenus multimédias IA

Au cours de la dernière décennie, l’intelligence artificielle a profondément transformé la création et la diffusion de contenus multimédias. Applications grand public (assistants vocaux, générateurs d’images, doublage automatique) comme solutions professionnelles les (studioworkflow, post production cinématographique) reposent désormais sur des modèles de plus en plus puissants et accessibles . Cette démocratisation pose un double enjeu :

Le volume et la rapidité de production : les industries créatives (jeux vidéo, cinéma, marketing digital, e-learning) recherchent des outils capables de générer à la volée des voix synthétiques , des effets sonores ou des images de haute qualité, afin de réduire les délais de production et de rester compétitives.

Personnalisation et accessibilité : la demande croissante de contenu sur mesure pour des publics variés (multilinguisme, besoins spécifiques d’accessibilité) conduit à rechercher non seulement des résultats visuels ou sonores de haute fidélité, mais aussi une capacité a adapter le style, la langue ou l’émotion véhiculée.

En parallèle, les coûts associés aux prestations traditionnelles (enregistrement en studio, composition sonore, graphisme) et le manque de ressources spécialisées constituent un frein pour les petites structures et les projets a budget limité. L’IA appliquée aux contenus multimédias apparaît dès lors comme une solution à la fois économique et évolutive, capable de compléter, voire parfois remplacer, les méthodes classiques.

## 1.2 Problématique et motivation

Malgré ces avancées, plusieurs obstacles subsistent :

* ***Dépendance aux API et services tiers* :** la plupart des solutions “plug-and-play” reposent sur des services hébergés et facturés à l’usage (Google Cloud TTS, Amazon Polly, Microsoft Azure TTS, etc.). Au-delà du coût, cela pose des questions de souveraineté des données et de contrôle sur la qualité (latence, fiabilité, évolutivité).
* ***Limites de personnalisation*** *:* les voix pré-packagées manquent souvent de naturalité ou de flexibilité stylistique. De même, les générateurs d’effets sonores “text-to-sound” ou d’images ont des capacités de fine-tuning limitées, rendant difficile l’obtention de résultats parfaitement alignés avec un cahier des charges créatif.
* ***Contraintes éthiques et réglementaires :***le clonage de voix et la génération automatisée soulèvent des enjeux de consentement, de protection des données personnelles et de conformité au Règlement général sur la protection des données (RGPD).

C’est dans ce contexte que naît la motivation de notre projet : concevoir une plateforme unifiée, entièrement auto-hébergée, offrant une maîtrise totale du pipeline de génération multimédia (voix, conversion vocale, effets sonores, images, vidéos). L’objectif est de concilier :

* ***Liberté et contrôle :***éviter les verrous des API externes, garantir la confidentialité des données utilisateur et assurer une portabilité aisée vers des environnements variés.
* ***Personnalisation avancée :***mettre en œuvre des techniques de fine-tuning sur des modèles open-source (StyleTTS2 pour la synthèse vocale, Seed-VC pour le clonage, Make-An-Audio pour les SFX, Stable Diffusion pour les images, etc.), afin d’ajuster précisément la sortie aux besoins spécifiques.
* ***Conformité et éthique:***intégrer nativement les mécanismes de consentement et de chiffrement AES-256, respecter les bonnes pratiques RGPD et offrir des interfaces claires pour la gestion des droits vocaux.

## 1.3 Objectifs du projet

Le projet Creative AI se structure autour de quatre objectifs principaux, articulés en objectifs techniques, fonctionnels et qualitatifs :

* Auto-hébergement et portabilité
  + Déployer l’ensemble des services (TTS, clonage, SFX, image, vidéo) sur des instances AWS EC2 encapsulées dans des conteneurs Docker.
  + Garantir une procédure de déploiement reproductible (Dockerfiles, AWS CLI) pour tout environnement cloud ou on-premise.
* Personnalisation et qualité
  + Fine-tuner StyleTTS2 et Seed-VC sur des corpus ciblés pour atteindre un Mean Opinion Score (MOS) > 4/5 et assurer une naturalité proche de la voix humaine.
  + Proposer la génération d’effets sonores textuels et d’images 720p via des modèles open-source compensant les limitations des API propriétaires.
* Conformité RGPD et sécurité
  + Mettre en place une authentification sécurisée (NextAuth.js) et un gestionnaire de consentement clair pour tout usage de clonage vocal.
  + Chiffrer AES-256 les données sensibles au repos (S3) et en transit (HTTPS, TLS).
* *E*xpérience utilisateur et accessibilité
  + Concevoir une interface Next.js intuitive, responsive et conforme.
  + Réduire le temps d’apprentissage à moins de 10 minutes pour un nouvel utilisateur et offrir un historique complet des clips générés.

## 1.4 Méthodologie et organisation du rapport

* Pour atteindre ces objectifs, nous avons suivi une démarche structurée en cinq grandes phases :
* Analyse et specifications:
  + Rédaction du cahier des charges fonctionnel et non-fonctionnel.
  + Étude de l’état de l’art des modèles IA, des contraintes légales et des architectures cloud.
* Conception UML et architecture :
  + Élaboration des diagrammes de cas d’usage, séquence, activité, classes, états, composants et déploiement.
  + Choix des technologies (Python 3.10, PyTorch, FastAPI, Next.js, Docker, AWS).
* Implémentation et fine-tuning :
  + Développement du backend FastAPI et du frontend Next.js.
  + Conteneurisation Docker et configuration AWS (EC2, S3, RDS, ECR).
  + Fine-tuning des modèles TTS et de conversion vocale.
* Tests et validation :
  + Tests fonctionnels, tests de montée en charge et évaluation MOS.
  + Ajustements itératifs sur les hyperparamètres et le workflow d’inférence.
* Rédaction et soutenance
  + Rédaction du rapport final selon le template de l’école.
  + Préparation de la présentation (slides, démonstration live).

Le présent rapport est structuré en neuf chapitres, chacun correspondant à une phase essentielle du projet, puis complété par des annexes. Les quatre premiers chapitres posent le cadre et détaillent l’analyse fonctionnelle et technique : ils présentent l’introduction et la méthodologie adoptée, l’état de l’art, les spécifications fonctionnelles ainsi que la modélisation UML (cas d’usage, diagrammes de classes et d’activités). Les chapitres cinq à sept décrivent la mise en œuvre pratique : le développement du backend et du frontend, le fine-tuning des modèles d’intelligence artificielle et enfin les tests (fonctionnels, de performance et évaluations perceptives selon la méthode MOS). Le huitième chapitre aborde la phase de déploiement en production et les modalités d’exploitation opérationnelle. Le neuvième et dernier chapitre tire les conclusions du travail réalisé et ouvre sur les perspectives d’évolution. Les annexes regroupent l’ensemble des scripts sources, des diagrammes complets et des fiches de tests qui appuient et enrichissent le corps principal du rapport.

# Chapitre 2

**État de l’art**

Dans ce chapitre, nous passons en revue les principales techniques de génération multimédia par IA : de la synthèse vocale (TTS) et du clonage vocal aux effets sonores textuels, en passant par la création d’images et de vidéos. Après avoir détaillé les approches clés (concatenative, parametric, deep learning) et les méthodes de voice cloning (speaker adaptation et encoding), nous comparons les solutions propriétaires et open‑source pour mettre en perspective l’apport de Creative AI.

# État de l’art

## 2.1 Synthèse vocale (TTS)

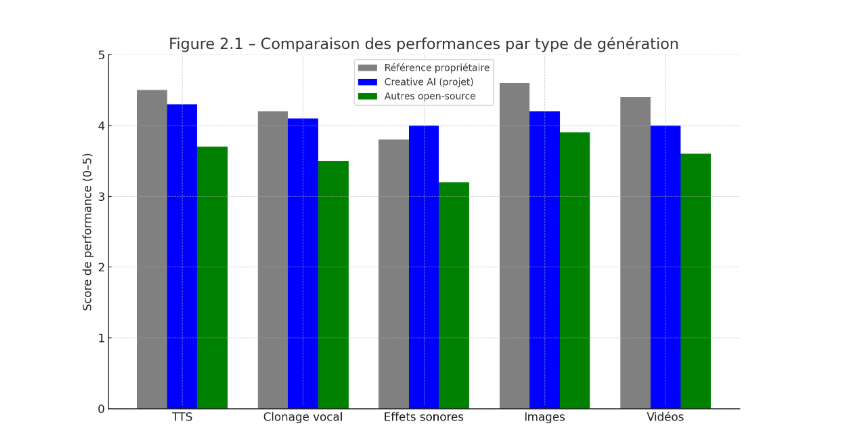
La synthèse vocale (Text-to-Speech, TTS) vise à transformer du texte écrit en une parole artificielle naturelle. Historiquement, trois grandes approches se sont succédé :

* Concatenative TTS :  
   – Principe : assemblage de petits fragments de voix humaine préenregistrés.  
   – Avantages : qualité vocale très naturelle si le corpus couvre tous les phonèmes et contextes.  
   – Limites : nécessite une banque de phrases très exhaustive ; transitions parfois abruptes ; faible flexibilité.
* Parametric TTS :  
   – Principe : génération de la voix via des modèles statistiques (HMM, vocodeurs), où les paramètres (hauteur, rythme, timbre) sont contrôlés par des algorithmes.  
   – Avantages : plus compact ; permet de modifier facilement le style (émotion, vitesse).  
   – Limites : qualité inférieure à la concatenative, souvent « robotique ».
* Deep Learning TTS :   
   – Principe : réseaux de neurones profonds apprenant directement la relation texte-onde sonore, via des architectures séquentielles (Tacotron) ou basées sur la convolution (FastSpeech).

– Exemples récents :  
  - Tacotron 2 (Google, 2017) : architecture seq2seq + attention pour transformer texte en spectrogrammes, couplée à WaveNet pour la génération d’onde brute.  
  - FastSpeech (Microsoft, 2019) : modèle non-autoregressif réduisant considérablement le temps d’inférence.  
  - StyleTTS2 (2023) : intègre un codeur de style qui permet de dissocier contenu linguistique et caractéristiques de la voix, autorisant un fine-tuning plus fin sur un locuteur spécifique.

Enjeux actuels : réduire la latence (batching, modèles non-autoregressifs), améliorer la naturalité (prosodie, émotion), supporter un grand nombre de langues et de voix différentes, et permettre un fine-tuning facile à l’aide de peu de données.

*Figure 1 Comparaison des performances par type de génération*

**

Cette figure compare les performances estimées (sur 5) de différentes catégories de modèles d'IA créative :

* Référence propriétaire : des solutions commerciales comme ElevenLabs, Sora, ou DALL·E 3, qui atteignent généralement des scores élevés.
* Creative AI : notre projet atteint des performances proches dans tous les domaines, avec un léger écart en génération vidéo.
* Solutions open-source alternatives : performances globalement inférieures, notamment en clonage vocal et effets sonores.

Cela montre que notre solution Creative AI représente une alternative robuste, équilibrée et auto-hébergeable.

## 2.2 Clonage vocal

Le clonage vocal (voice cloning) consiste à produire une voix synthétique reproduisant la couleur et les particularités d’une voix cible à partir d’un petit échantillon. Deux grandes familles de méthodes existent :

* Speaker adaptation  
   – Principe : démarrer d’un modèle multi-locuteur pré-entraîné, puis adapter ses paramètres à une voix cible via un fine-tuning.  
   – Exemples : BVAE, VoiceLoop.
* Speaker encoding  
   – Principe : extraire un vecteur d’identité vocale (embedding) depuis quelques secondes d’audio, puis conditionner le modèle TTS sur cet embedding sans modifier les poids.  
   – Exemples :  
    • SV2TTS (2019) : pipeline en trois étapes – extraction d’un embedding par un réseau d’encodage (d-vector), synthèse via Tacotron, vocodage WaveRNN.  
    • YourTTS / Vall-E (2023) : modèles end-to-end utilisant des architectures transformers pour encoder style et contenu dans un même modèle.

Le modèle Seed-VC retenu dans notre projet appartient à cette seconde catégorie : il génère des échantillons réalistes après un fine-tuning minimal, ce qui le rend particulièrement adapté pour des scénarios nécessitant la rapidité et la simplicité d’intégration.

## 2.3 Génération d’effets sonores

La génération d’effets sonores à partir de descriptions textuelles (text-to-sound) est un champ émergent :

* **Approches basées sur la synthèse** : utilisation de banques d’échantillons et de règles DSP pour assembler des sons (limité à un répertoire prédéfini).
* **Approches deep learning** :  
    • **Make-An-Audio** (Meta, 2023) : transformer des embeddings textuels en spectrogrammes à l’aide d’un réseau diffusion, puis vocoder pour produire le son brut.  
    • **AudioGen** (Google, 2023) : modèles diffusion conditionnés, capable de générer des bruitages de haute qualité (ambiance, effets d’impact, bruitages naturels).

**Défis** : diversité et réalisme des sons (la texture d’un vent peut varier énormément), contrôle temporel (durée, montée/descente), mise en cohérence dans un montage audio.

## 2.4 Génération d’images et de vidéos

* **Images** :
* **GAN** (Generative Adversarial Networks) : pionniers (DCGAN, StyleGAN), permettent de générer des images réalistes mais parfois sujettes à artefacts.
* **Diffusion models** :  
    - **Stable Diffusion** (2022) : diffusion latente pour produire des images haute résolution à partir de prompts textuels.  
    - **Imagen** (Google, 2022) : qualité supérieure pour les détails, mais modèle fermé.
* **Vidéos** :
* **Modèles prédictifs** : VideoGPT, qui génère des séquences d’images via un transformer, limité à de courtes boucles.
* **Diffusion vidéo** :  
    - Fireworks (Fireworks, 2023) : diffusion conditionnée sur texte pour générer des clips de plusieurs secondes.  
    - combine diffusion image + audio pour produire de courtes vidéos avec narration.

**Limites actuelles** : résolution/longueur restreintes, coûts GPU élevés, synchronisation audio-vidéo, personnalisation fine. L’utilisation de ces modèles dans Creative AI vise principalement la génération d’illustrations et de courtes animations, laissant la production long format pour des itérations futures.

## 2.5 Étude comparative des solutions existantes

*Table II: Étude comparative des solutions existantes*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Solution** | **Type** | **Modèle** | **Open-Source** | **Hébergement** | **Personnalisation** | **Coût** |
| Google Cloud TTS | TTS | WaveNet | Non | Cloud interne | Faible | Usage (€/$) |
| Amazon Polly | TTS | Neural TTS | Non | Cloud interne | Faible | Usage (€/$) |
| Microsoft Azure TTS | TTS | Custom Neural | Non | Cloud interne | Moyen | Usage (€/$) |
| StyleTTS2 + Docker | TTS | StyleTTS2 | Oui | Auto-hébergé AWS | Élevée (fine-tune) | Coût EC2 + S3 |
| SV2TTS / YourTTS | Voice Clone | SV2TTS, Vall-E | Partiel | Cloud / On-Prem | Élevée | GPU + stockage |
| Make-An-Audio | SFX | Diffusion | Oui | Auto-hébergé AWS | Moyen | GPU + stockage |
| Stable Diffusion | Image | Latent Diffusion | Oui | Auto-hébergé | Élevée | GPU + stockage |
| Phenaki / Gen-2 | Vidéo | Diffusion vidéo | Non/Partiel | Cloud interne | Faible/Moyen | Très élevé |

Cette comparaison met en évidence deux tendances fortes. D’un côté, les solutions propriétaires (Google Cloud TTS, Amazon Polly, Microsoft Azure TTS, Phenaki/Gen‑2) offrent une intégration clé en main et des performances élevées, au prix d’une dépendance aux API externes et de coûts d’utilisation récurrents. De l’autre, les alternatives open‑source (StyleTTS2, SV2TTS/YourTTS, Make‑An‑Audio, Stable Diffusion) demandent un effort initial de déploiement et de fine‑tuning, mais garantissent une maîtrise totale du pipeline, une personnalisation fine et un modèle économique plus prévisible, basé sur les dépenses d’infrastructure. Creative AI se positionne ainsi comme un compromis idéal : en s’appuyant sur des modèles open‑source auto‑hébergés, le projet peut atteindre des niveaux de qualité comparables (MOS > 4/5) tout en conservant le contrôle des données et en optimisant les coûts opérationnels.

* Les solutions propriétaires offrent une grande simplicité d’usage, mais au prix d’une dépendance et de coûts récurrents.
* Les alternatives open-source, bien que nécessitant davantage d’efforts de déploiement et de fine-tuning, offrent une maîtrise complète du pipeline, une personnalisation poussée et un coût plus prévisible (capex + opex AWS).

# Chapitre 3

**Spécifications et conception**

**Fonctionnelle**

Dans ce chapitre, nous définissons d’abord les exigences fonctionnelles et non‑fonctionnelles du système Creative AI, en listant les cas d’usage clés (TTS, clonage vocal, SFX, images, vidéos, gestion d’abonnement et interface d’administration) ainsi que les critères de performance, sécurité, fiabilité et accessibilité. Nous présentons ensuite le diagramme de cas d’utilisation, qui met en relation les acteurs (utilisateur, administrateur, API, stockage) avec les principaux services. Deux diagrammes de séquence détaillent les flux dynamiques de génération de contenu et de paiement d’abonnement, avant que le diagramme d’activités ne synthétise l’enchaînement des tâches entre l’utilisateur et le système, de la connexion à la déconnexion.

# Spécifications et conception Fonctionnelle

## 3.1 Cahier des charges fonctionnel et non-fonctionnel

### A. Besoins fonctionnels

Le système Creative AI vise à permettre aux utilisateurs de générer automatiquement des contenus multimédias à l’aide de modèles d’intelligence artificielle auto-hébergés. Les fonctionnalités principales sont :

* **Synthèse vocale (Text-to-Speech)** :  
   Convertir du texte en parole avec StyleTTS2. Support multilingue et personnalisation de voix via fine-tuning.
* **Clonage vocal (Voice Conversion)** :  
   Transformer une voix existante (uploadée) en une autre voix cible, avec Seed-VC.
* **Génération d’effets sonores (SFX)** :  
   Produire des sons à partir de descriptions textuelles à l’aide du modèle Make-An-Audio.
* **Génération d’images** :  
   À partir d’un prompt, générer une image (Stable Diffusion). Édition possible des résultats générés.
* **Génération de vidéos** :  
   Créer une courte séquence vidéo à partir d’un texte et/ou image (via outils comme VideoGPT ou GeminiAI).
* **Consultation de l’historique** :  
   L’utilisateur peut revoir ses générations passées et rejouer les fichiers stockés.
* **Téléchargement des fichiers** :  
   Accès direct aux fichiers audio, image ou vidéo stockés sur AWS S3.
* **Gestion de l’abonnement** :  
   Choix de plan, paiement en ligne, suivi de la consommation.
* **Interface d’administration** :  
   Réservée aux administrateurs :
  + Supervision des utilisateurs, plans, quotas
  + Suivi des performances et de la charge IA
  + Surveillance de l’usage API

### B. Besoins non-fonctionnels

* **Performance** :  
   • Latence < 3s pour TTS  
   • MOS > 4/5  
   • Image 720p, vidéo 720p min  
   • 10 requêtes concurrentes par utilisateur
* **Sécurité** :  
   • Authentification sécurisée (NextAuth.js)  
   • RGPD : consentement explicite, cryptage AES-256  
   • Communication HTTPS chiffrée
* **Fiabilité** :  
   • Sauvegarde automatique des modèles et clips  
   • Monitoring via AWS CloudWatch  
   • 99.9% de disponibilité
* **Portabilité** :  
   • Conteneurisation Docker  
   • Déploiement reproductible via Terraform + AWS CLI
* **Accessibilité** :  
   • Interface Next.js responsive  
   • Conformité WCAG 2.1 (accessibilité numérique)

# 3.2 Diagramme de cas d’utilisation

Le diagramme de cas d’utilisation suivant illustre les principales interactions entre les acteurs du système (utilisateur, administrateur) et les services disponibles via l'interface et l'API :

**Acteurs principaux :**

* **Utilisateur** :  
   Interagit avec l’application pour générer du contenu, gérer son compte et son historique.
* **Administrateur** :  
   Supervise les comptes, abonnements, performances du système.

**Acteurs systèmes secondaires :**

* **Système API (FastAPI)** :  
   Interprète les requêtes front-end et déclenche les modèles d’IA.
* **Système de stockage (AWS S3)** :  
   Gère les fichiers générés.

**Principaux cas d’utilisation :**

*3* *Principaux cas d’utilisation*

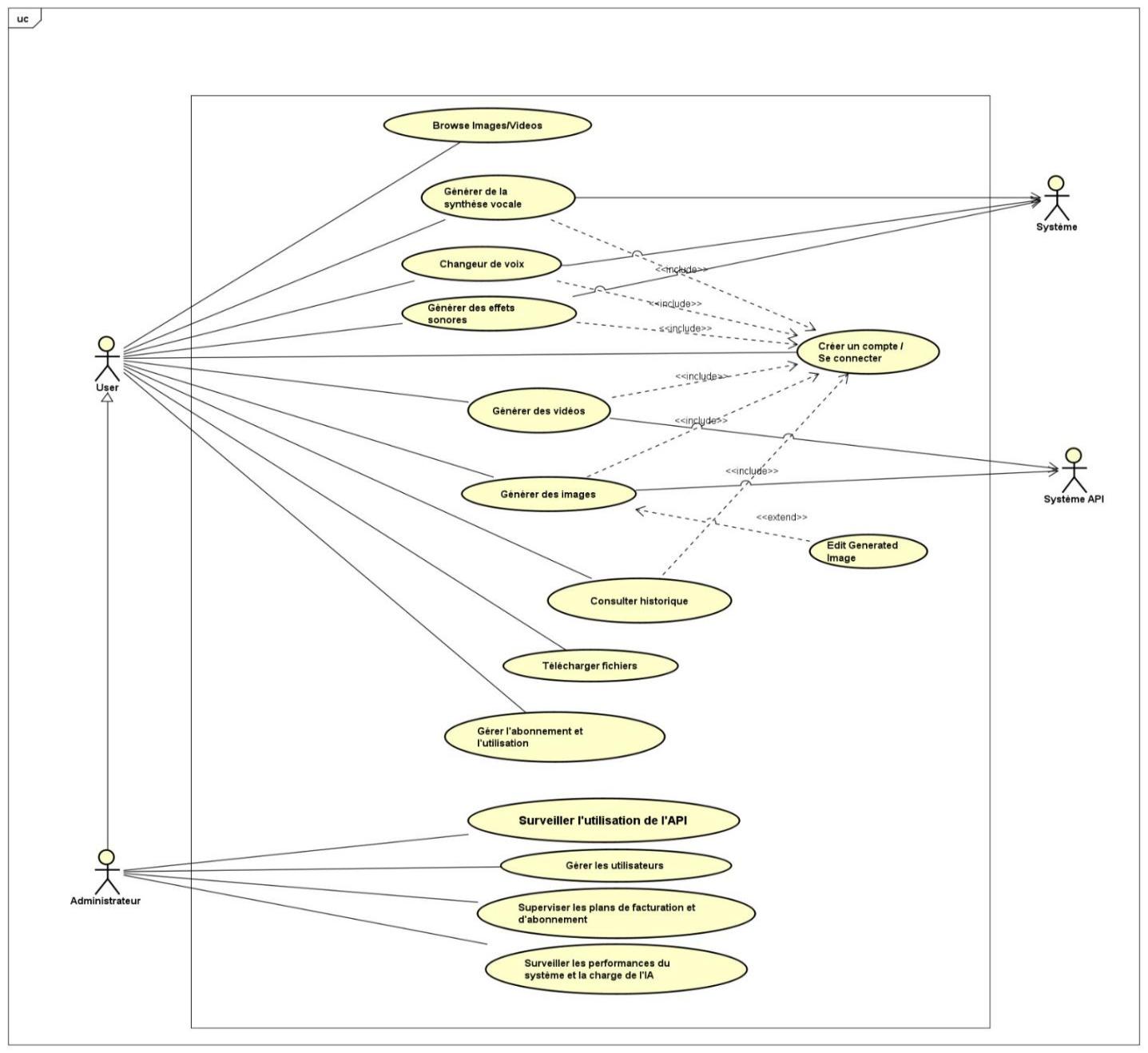
|  |  |
| --- | --- |
| **Cas d’utilisation** | **Description** |
| **Créer un compte / Se connecter** | Authentification via email (NextAuth.js) |
| **Générer de la synthèse vocale** | Entrée texte → audio généré avec StyleTTS2 |
| **Changer de voix** | Upload voix source → conversion vers voix cible avec Seed-VC |
| **Générer des effets sonores** | Texte descriptif → effet sonore généré avec Make-An-Audio |
| **Générer des images** | Texte → image 720p via Stable Diffusion |
| **Générer des vidéos** | Texte/image/audio → animation ou clip vidéo (VideoGPT, GeminiAI) |
| **Télécharger fichiers** | Lien vers le fichier stocké sur AWS S3 |
| **Consulter historique** | Liste des fichiers générés, triés par date ou type |
| **Gérer l’abonnement** | Affichage des offres, paiement via Stripe/API bancaire, activation/désactivation |
| **Éditer une image générée (extend)** | Modifier l’image directement dans l’interface (filtre, recadrage, texte) |
| **Superviser l’API (Admin)** | Voir l’utilisation (requêtes, latence, erreurs) |
| **Gérer les utilisateurs (Admin)** | CRUD sur les comptes, suppression abusive, ban, etc. |
| **Surveiller la charge IA (Admin)** | Utilisation GPU, files d’attente, erreurs système |

**Relations UML :**

* **<<include>>** **:** la connexion est nécessaire pour toutes les générations.
* **<<extend>>** **:** l’édition d’image est optionnelle après la génération.

*3* *Diagramme de cas d’utilisation du système Creative AI*

La figure 2 illustre les interactions entre les utilisateurs (classiques et administrateurs) et les fonctionnalités offertes par la plateforme. Les relations « include » et « extend » permettent de structurer les cas liés à l’authentification et aux opérations optionnelles.

**

## 3.3 Diagrammes de séquence

Cette section présente deux diagrammes de séquence UML illustrant les principales interactions dynamiques du système :

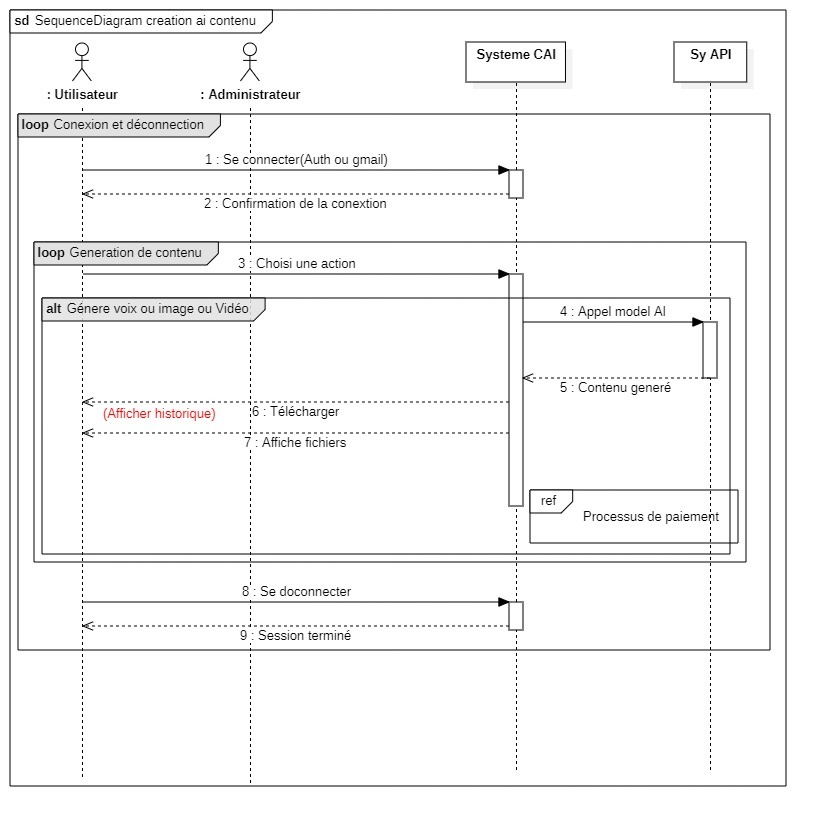
**3.3.1** Génération de contenu IA (connexion → génération → déconnexion)

**3.3.2** Processus de paiement pour l’abonnement

### 3.3.1 Diagramme de séquence — Génération de contenu IA

*4* *Diagramme de séquence : génération de contenu IA*

La figure 3 met en scène l’acteur **Utilisateur** et trois composants clés du système : **Next.js App**, **FastAPI API** et **Task Worker/Services IA**. Il couvre le flux complet d’une session de génération, depuis la connexion jusqu’à la déconnexion, en passant par l’appel aux modèles IA et la restitution du clip généré.

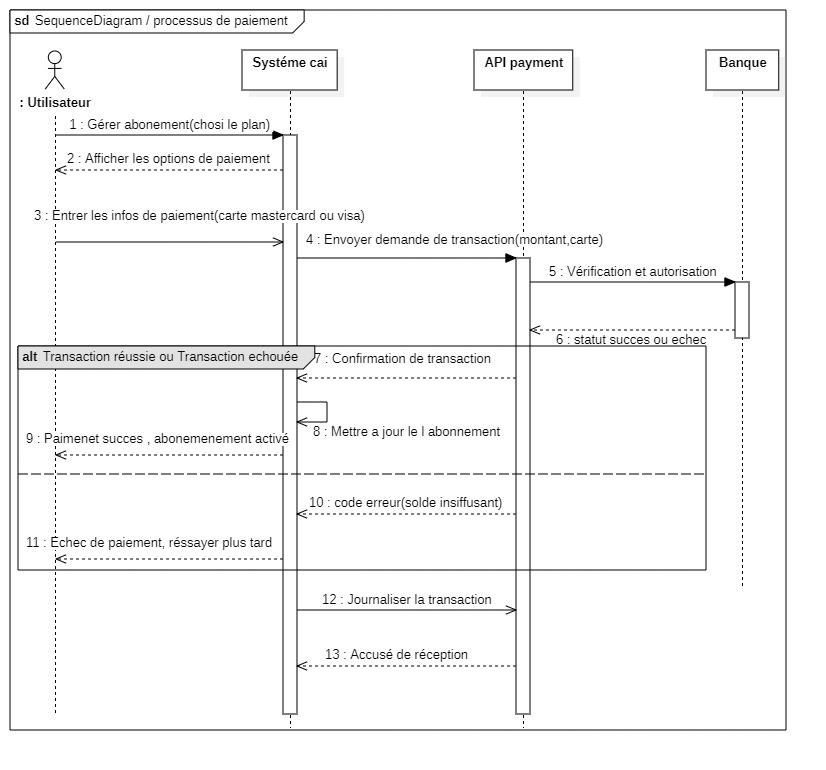
**

1. **Connexion**
   * L’utilisateur s’authentifie (email/OAuth) auprès de la Next.js App.
   * La Next.js App appelle l’API FastAPI pour valider les identifiants.
   * En cas de succès, la session est ouverte.
2. **Boucle de generation**
   * **Choix de la génération** : l’utilisateur sélectionne “TTS”, “Conversion vocale” ou “Effets sonores”.
   * **Requête** : la Next.js App envoie une requête POST /generate à l’API FastAPI avec le payload (texte, voix source, type de génération).
   * **Envoi en tâche asynchrone** : l’API FastAPI place la demande dans la file (Inngest) pour traitement asynchrone.
   * **Traitement IA** : le Task Worker récupère la tâche, appelle le microservice (StyleTTS2, Seed-VC ou Make-An-Audio), reçoit le fichier WAV.
   * **Stockage** : le Worker upload le clip sur AWS S3 et notifie l’API FastAPI de l’URL de stockage.
   * **Retour front** : l’API renvoie un code 200 et l’URL du clip à la Next.js App, qui actualise l’UI et arrête le spinner.
3. **Déconnexion**
   * L’utilisateur clique sur “Logout”.
   * La Next.js App notifie l’API FastAPI, qui détruit la session et renvoie la confirmation.

### 3.3.2 Diagramme de séquence — Processus de paiement

*5* *Diagramme de séquence : abonnement et paiement*

La figue 4 illustre la voie critique pour l’achat ou le renouvellement d’un abonnement via l’interface utilisateur et le service de paiement externe.

**

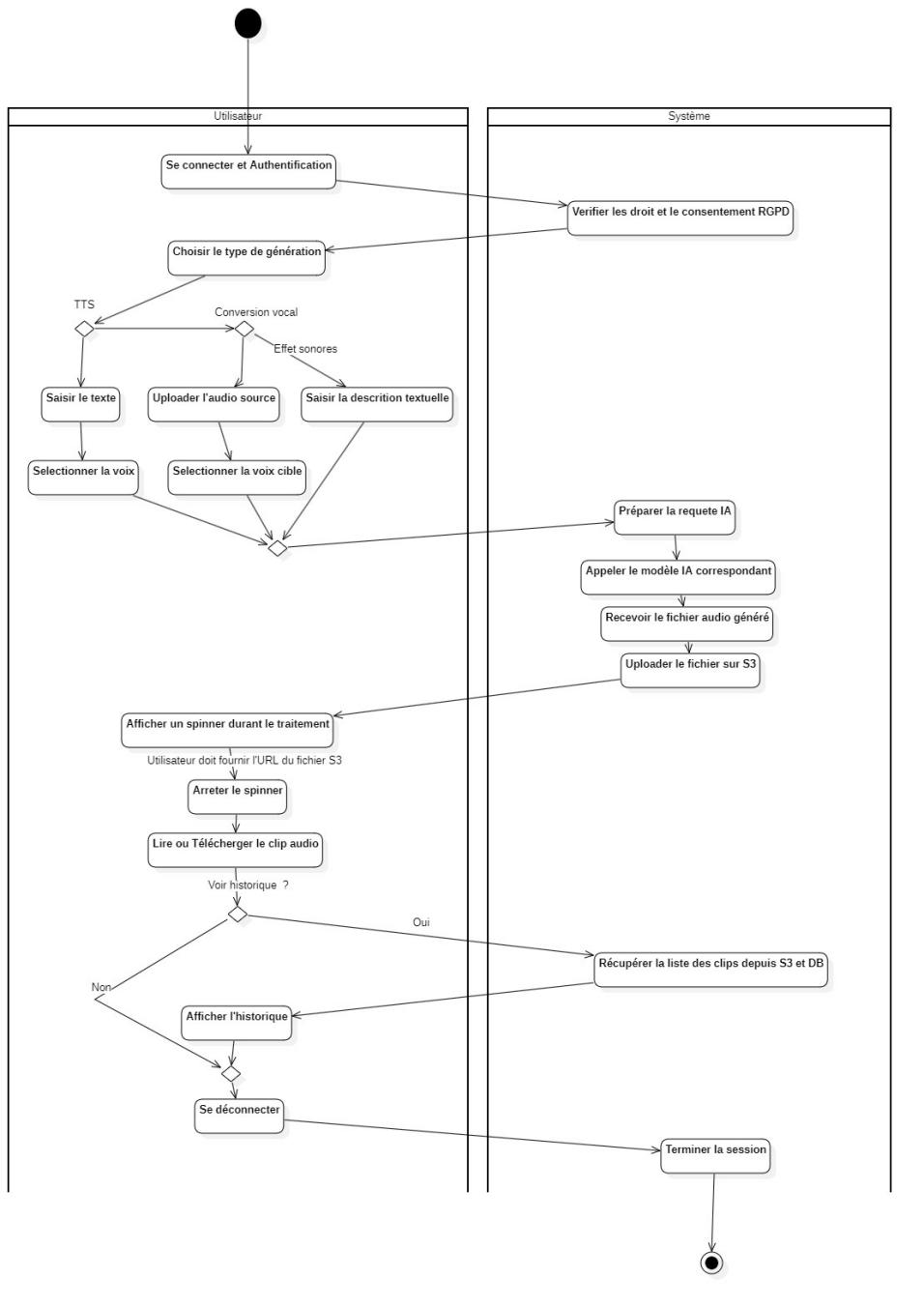
1. **Choix du plan :**
   * L’utilisateur consulte les offres (Basic, Pro) dans la Next.js App et sélectionne un plan.
2. **Initiation du paiement :**
   * La Next.js App envoie la demande de paiement à l’API FastAPI (POST /subscribe) avec le plan et les informations de l’utilisateur.
   * L’API FastAPI crée une **Payment Intent** via l’API Payment (Stripe, ou autre).
3. **Saisie des informations bancaires :**
   * L’utilisateur saisit ses données de carte dans la Next.js App (composant sécurisé).
   * Le frontend appelle directement l’API Payment pour tokeniser la carte et demander l’autorisation.
4. **Vérification bancaire :**
   * L’API Payment communique avec la banque de l’utilisateur.
   * La banque renvoie un statut **SUCCESS** ou **FAILED**.
5. **Confirmation & mise à jour :**
   * En cas de succès :
     + L’API Payment notifie l’API FastAPI.
     + FastAPI met à jour l’état de l’**Abonnement** en **ACTIVE** et enregistre un objet **Paiement** (status = SUCCESS).
     + La Next.js App affiche la confirmation et débloque l’accès aux fonctionnalités IA.
   * En cas d’échec :
     + FastAPI enregistre un **Paiement** (status = FAILED).
     + La Next.js App notifie l’utilisateur de l’erreur et propose de réessayer.
6. **Journalisation**
   * Quel que soit le résultat, l’API FastAPI consigne la transaction dans la base de données pour audit et suivi.

**Remarque :** Les deux diagrammes de séquence permettent de valider l’ordonnancement des appels, de repérer les points de latence critiques (file d’attente, appels IA, communication avec le service de paiement) et de s’assurer que la gestion des sessions et des erreurs est correctement prise en compte.

## 3.4 Diagramme d’activités

*6* *Diagramme d’activités : génération de contenu IA*

La figure 5 met en lumière le flux de travail global pour la génération de contenu (TTS, conversion vocale ou effets sonores) en distinguant clairement les responsabilités de l’utilisateur et celles du système

**

*4* *Responsabilités principales*

|  |  |
| --- | --- |
| **AC** | **Responsabilités principales** |
| Utilisateur | - Se connecter / s’authentifier  - Choisir le type de génération (TTS, VC, SFX)  - Fournir les entrées (texte, fichier source ou description)  - Lancer la génération  - Arrêter le spinner et lire/télécharger le clip  - Consulter l’historique  - Se déconnecter |
| Système | - Vérifier l’authentification et le consentement RGPD  - Valider et enrôler la requête dans la file asynchrone (Inngest)  - Orchestrer le Task Worker pour appeler le modèle IA approprié  - Recevoir le fichier généré et l’uploader sur AWS S3  - Retourner l’URL du clip au frontend  - Récupérer et fournir l’historique depuis S3/BD  - Clôturer la session |

**Flux détaillé :**

1. **Authentification**
   * L’utilisateur lance l’activité « Se connecter ».
   * Le système vérifie les identifiants et retourne un jeton de session valide.
2. **Préparation de la requête**
   * L’utilisateur sélectionne le type de génération et fournit son entrée (texte ou audio).
   * Le système réalise un contrôle formel (validation du texte, existence du fichier source).
3. **Enfilement et traitement**
   * Le système crée une tâche dans la queue Inngest pour garantir la fiabilité et la scalabilité.
   * Le Task Worker récupère la tâche, invoque le modèle IA (StyleTTS2, Seed-VC ou Make-An-Audio) puis reçoit le flux audio.
4. **Stockage et retour**
   * Le système upload le fichier obtenu sur AWS S3 et stocke les métadonnées dans la base de données.
   * L’URL du clip est renvoyée à l’interface Next.js, qui arrête le spinner et autorise la lecture ou le téléchargement.
5. **Historique et déconnexion**
   * En option, l’utilisateur peut consulter son historique : le système récupère la liste des clips depuis S3/BD et l’affiche.
   * Enfin, l’utilisateur se déconnecte : le système invalide la session

# Chapitre 4

**Architecture logicielle**

**et infrastructure**

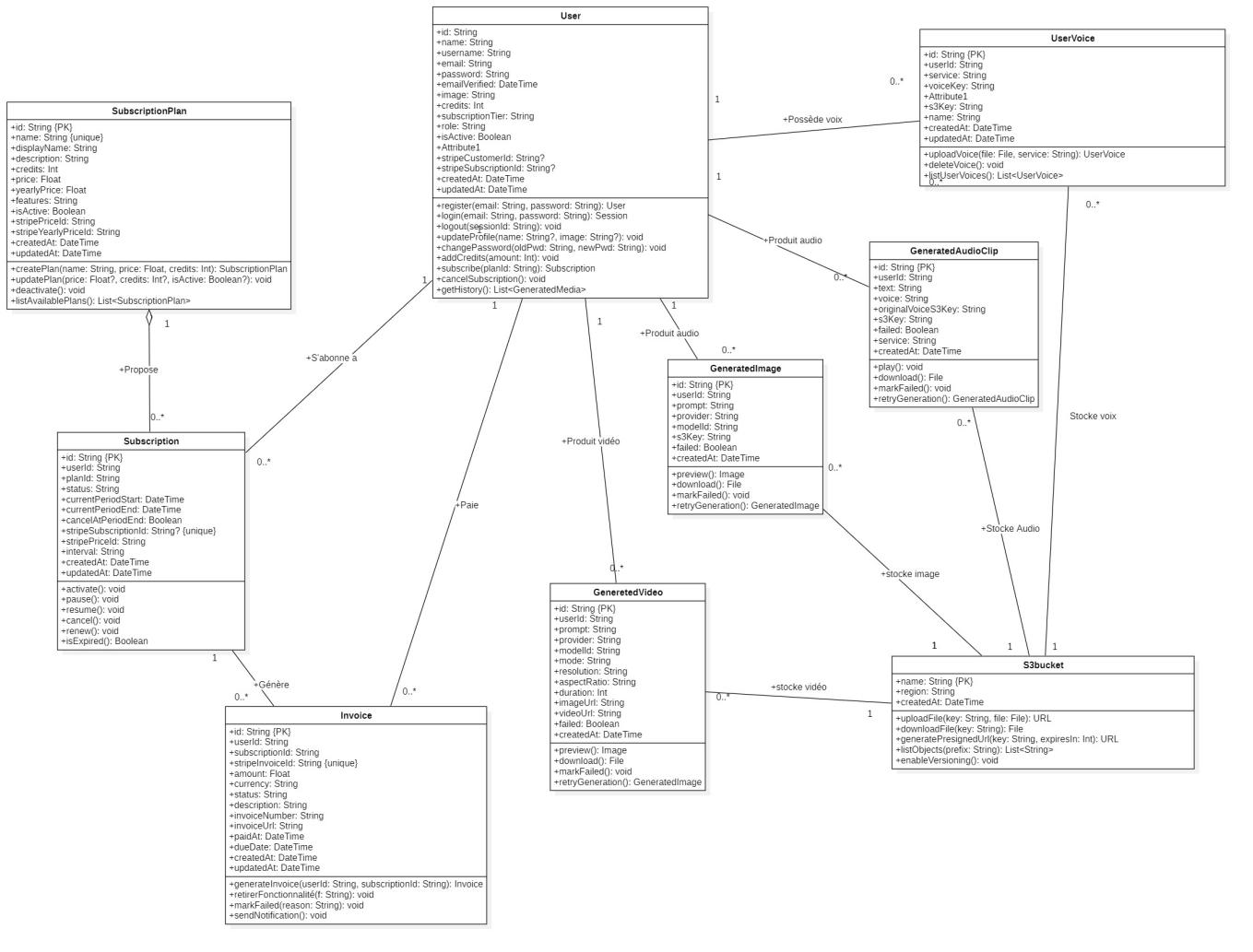
Ce chapitre présente la structure interne de la plateforme Creative AI et son déploiement technique. Nous commençons par le diagramme de classes (figure 4.1), qui détaille les entités métier (utilisateur, abonnement, factures, clips multimédias, stockage S3) et leurs relations. Le diagramme d’états (figure 4.2) modélise le cycle de vie des abonnements, puis celui des requêtes de génération IA, en illustrant les transitions clés (validation, traitement, contrôle qualité, échec ou succès). Le diagramme de déploiement (figure 4.4) décrit la répartition des composants (frontend serverless, backend conteneurisé, workers IA, bases de données et bucket S3), tandis que le diagramme de communication (figure 4.5) explicite les messages échangés entre l’utilisateur, le frontend, le backend, le modèle IA et le stockage pour une génération audio.

# Architecture logicielle et infrastructure

## 4.1 Diagramme de classes

*7* *Diagramme de classes du système Creative AI*

La figure 6 ci-dessus modélise l’ensemble des entités de la plateforme **Creative AI**, leurs attributs principaux, leurs méthodes essentielles ainsi que les relations qui les unissent. Il comporte neuf classes métier et un composant de stockage (S3Bucket), représentant les fondations de l’architecture métier de la plateforme.

**

Le diagramme de classes modélise les neuf entités principales de la plateforme ainsi que leur composant de stockage S3. Ci‑dessous, chaque classe est décrite de façon synthétique et organisée :

**User**  
La classe **User** modélise un compte sur la plateforme Creative AI. Chaque utilisateur est identifié par un id unique et dispose d’une adresse email, d’un nom d’utilisateur et d’un mot de passe pour s’authentifier. On y trouve également des indicateurs métier (nombre de crédits disponibles, niveau d’abonnement, rôle utilisateur vs administrateur, statut RGPD, indicateur d’activité) ainsi que des horodatages (createdAt, updatedAt) pour le suivi des modifications. Les méthodes associées, telles que register(), login(), updateProfile() ou encore addCredits(), couvrent l’ensemble des opérations du cycle de vie d’un utilisateur : création de compte, gestion de session, modification des informations personnelles, souscription et annulation d’abonnement, et consultation de l’historique des contenus générés. Un utilisateur peut posséder plusieurs abonnements et factures, et produire divers médias (clips audio, vidéos, images, voix personnalisées).

**SubscriptionPlan**  
La classe **SubscriptionPlan** décrit les différents forfaits proposés sur la plateforme. Chaque plan se caractérise par un id, un nom unique (name) et un libellé (displayName), ainsi que par les modalités tarifaires (prix mensuel et annuel) et le nombre de crédits inclus. Elle stocke également la liste des fonctionnalités associées, un indicateur d’activation, les timestamps de création et de mise à jour, et les liens vers la passerelle de paiement (Stripe). Les méthodes createPlan(), updatePlan() ou listAvailablePlans() permettent de gérer dynamiquement l’offre de services. En agrégation, chaque SubscriptionPlan peut être lié à plusieurs abonnements.

**Subscription**  
La classe **Subscription** implémente le rattachement d’un utilisateur à un plan. Elle contient les identifiants de l’utilisateur et du plan, les dates de début et de fin de l’abonnement, un statut (en attente, actif, annulé), un indicateur d’annulation (isCanceled) et les liens de paiement associés. Les méthodes activate(), pause(), cancel() ou renew() pilotent la vie de l’abonnement, tandis que isExpired() vérifie automatiquement sa validité. Chaque subscription peut générer plusieurs factures, ce qui permet un suivi détaillé des transactions.

**Invoice**  
La classe **Invoice** modélise les factures émises lors de chaque transaction d’abonnement. Elle contient un identifiant, les références à l’utilisateur et à l’abonnement concernés, le montant facturé, la devise, les URLs de notification, le statut de paiement et les dates clés (émission, règlement). Avec les méthodes generateInvoice(), sendNotification() ou markPaid(), le système crée, notifie et met à jour l’état des factures. Cette entité assure le lien entre les usages facturés et le suivi financier.

**GeneratedAudioClip**  
La classe **GeneratedAudioClip** représente un clip audio produit par le service TTS. Elle enregistre un id, l’utilisateur à l’origine de la demande, le texte ou la voix source, la clé S3 (emplacement de stockage), le nom du service IA utilisé et la date de création. Les méthodes play(), download() et retryGeneration() facilitent l’écoute, le téléchargement ou la relance en cas d’échec (markFailed()). Chaque clip est stocké unitairement dans un bucket S3, garantissant une gestion fine des ressources audio.

**GeneratedVideo**  
De même, **GeneratedVideo** gère les séquences vidéo générées : un identifiant, la référence à l’utilisateur, le prompt ou la source initiale, le fournisseur (VideoGPT, GeminiAI…), le modèle, la résolution, les URLs de stockage et la date de création. Elle propose des méthodes analogues à celles de l’audio (play(), download(), retryGeneration()). Chaque vidéo est associée à un bucket S3, permettant son stockage et son accès sécurisé.

**GeneratedImage**  
La classe **GeneratedImage** couvre la génération d’images : elle documente un id, l’utilisateur, le prompt, le modèle (Stable Diffusion…), la clé S3, la date et un indicateur d’échec éventuel. Grâce à preview(), download() et retryGeneration(), l’utilisateur peut visualiser, télécharger ou relancer une création. Le stockage dans S3 offre un accès performant et sécurisé aux fichiers images.

**UserVoice**  
**UserVoice** regroupe les voix personnalisées ou importées par un utilisateur. Chaque enregistrement conserve un identifiant, le userId, le service de provenance, une clé unique (voiceKey), le nom attribué à la voix et la clé S3. La contrainte d’unicité sur (userId, service, voiceKey) empêche la duplication. Les méthodes uploadVoice(), deleteVoice() et listUserVoices() garantissent une gestion intuitive des voix utilisateur, stockées elles aussi sur S3.

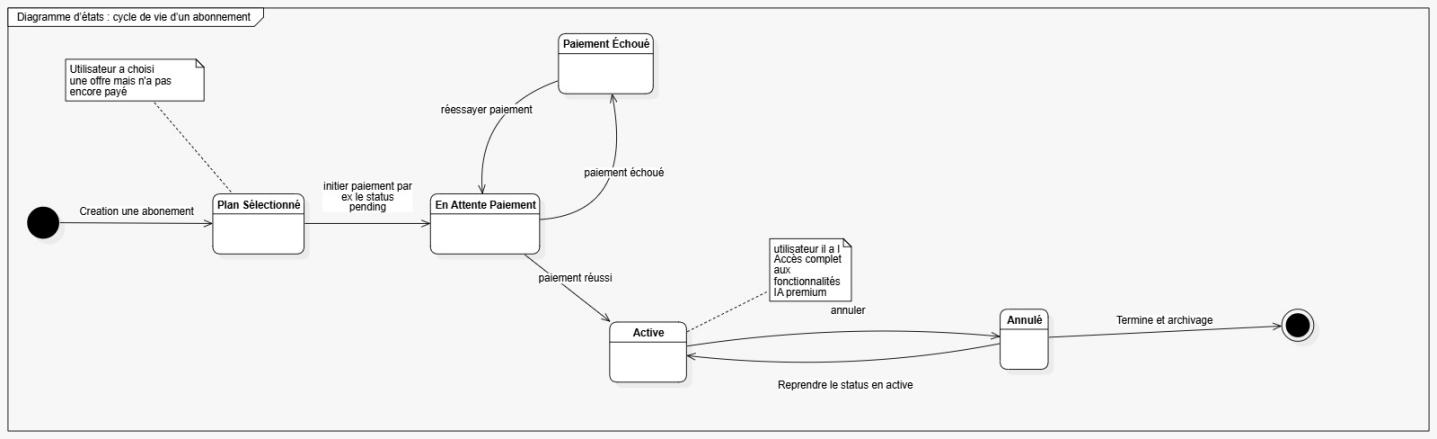
**S3Bucket**  
Enfin, le composant **S3Bucket** centralise le stockage AWS. Il s’identifie par un nom unique, une région et une date de création. Les opérations uploadFile(), downloadFile(), generatePresignedUrl(), listObjects() et enableVersioning() couvrent l’ensemble des besoins de gestion des objets multimédias (audio, vidéo, images, voix). Ce diagramme sert de référence pour la génération automatique des schémas de base de données et l’implémentation des entités métier via un ORM (Prisma, Drizzle).

## 4.2 Diagramme d’états

### 4.2.1 Diagramme d’états : cycle de vie d’un abonnement

*8* *Diagramme d’états : cycle de vie d’un abonnement*

La figure 7 modélise le parcours d’un abonnement utilisateur, depuis le moment où l’utilisateur choisit un plan tarifaire, jusqu’à la résiliation ou l’expiration effective de cet abonnement. Il permet de visualiser de façon claire les différents états possibles (sélectionné, en attente de paiement, actif, échoué, annulé) et les conditions (transitions) qui font passer l’abonnement d’un état à l’autre.

**

**États**

1. **Plan Sélectionné**
   * Description : L’utilisateur a cliqué sur une offre (par ex. « Premium » ou « Standard ») mais n’a pas encore finalisé le paiement.
   * Notes : À ce stade, l’abonnement est simplement créé et enregistré en attente d’une action de paiement.
   * Action associée : Lorsque l’utilisateur déclenche le paiement, l’état passer à « En Attente Paiement ».
2. **En Attente Paiement**
   * Description : L’abonnement est soumis à l’API de paiement ; la transaction n’est pas encore validée.
   * Transition « paiement échoué » → Paiement Échoué
   * Transition « paiement réussi » → Actif
   * Notes : Si le paiement est refusé (carte expirée, fonds insuffisants, etc.), on bascule en « Paiement Échoué ». Si le paiement s’effectue correctement, l’abonnement devient « Actif ».
3. **Paiement Échoué**
   * Description : Le paiement de l’abonnement a été rejeté.
   * Transition « réessayer paiement » → En Attente Paiement
   * Notes : L’utilisateur peut retenter le paiement autant de fois que nécessaire. Tant que le paiement n’aboutit pas, l’abonnement reste bloqué à l’état « Paiement Échoué ».
4. **Actif**
   * Description : L’abonnement est en état « Active », l’utilisateur a accès à l’ensemble des fonctionnalités (ex. : TTS, conversion vocale, génération d’effets, images/vidéos) correspondant à son plan.
   * Transition « annuler » → Annulé
   * Notes : Une fois l’abonnement actif, l’utilisateur peut décider de le résilier manuellement. Il est également possible d’automatiser une expiration à la date anniversaire du plan.
5. **Annulé**
   * Description : L’abonnement a été résilié par l’utilisateur (ou par un administrateur) ou a atteint son terme (expiration programmée).
   * Transition « reprendre » → Actif (sous réserve d’un nouveau paiement)
   * Transition « Terminé et archivé » → (état terminal)
   * Notes : Un abonnement annulé peut être réactivé si l’utilisateur choisit de repasser en « Actif » (en fournissant un nouveau mode de paiement). Toutefois, si l’abonnement est clos définitivement, il passe en état terminal (archivé), ce qui clôt le cycle de vie.
6. **État Terminal** (représenté par un rond cerclé)
   * Description : Objet final indiquant la fin du parcours pour cet abonnement (archivé de manière permanente).
   * Notes : Aucune transition n’en sort.

**Légende des transitions**

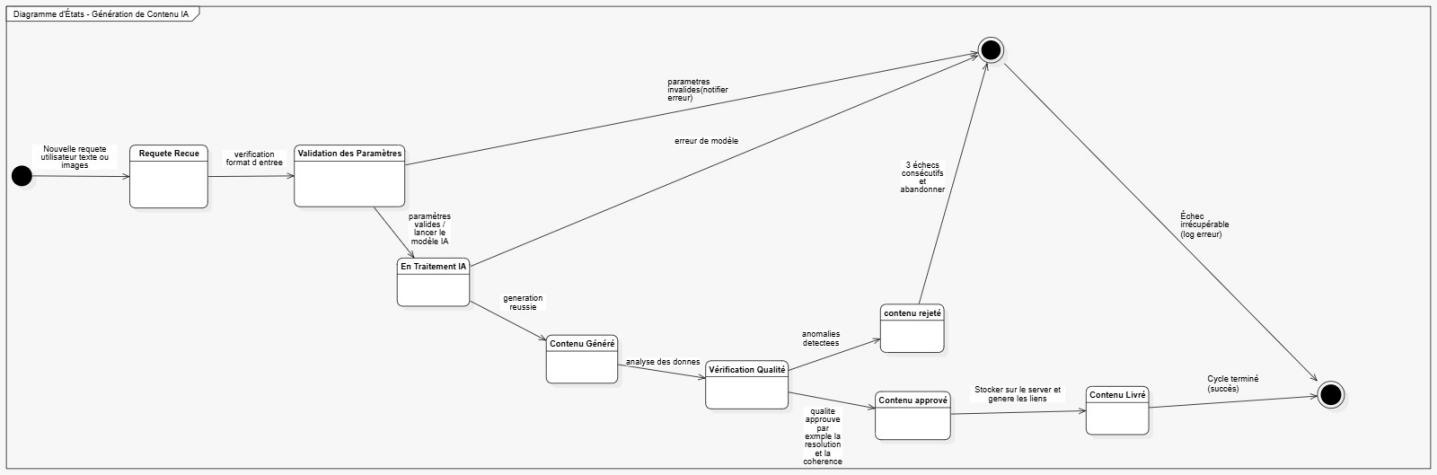
1. **Création d’un abonnement** (du point noir vers « Plan Sélectionné ») : l’utilisateur vient de choisir un plan.
2. **Initiation du paiement** (de « Plan Sélectionné » vers « En Attente Paiement ») : l’API de paiement prend en charge la transaction, on peut stocker l’état temporaire Pending.
3. **Paiement échoué** : lorsque la transaction retourne un code d’erreur (carte rejetée), on entre en « Paiement Échoué ».
4. **Réessayer paiement** : le même état « En Attente Paiement » est réactivé pour retenter la transaction.
5. **Paiement réussi** : l’utilisateur a correctement autorisé la transaction, on entre en « Actif ».
6. **Annuler** : l’abonné choisit de résilier son plan, on entre en « Annulé ».
7. **Reprendre l’état actif** : si l’utilisateur repaye après une annulation ou un échec, on retrouve l’état « Actif ».
8. **Terminé et archivé** : une fois qu’un abonnement annulé reste sans reprise ou qu’il est expiré automatiquement, il passe à l’état final (archivé).

Ce diagramme d’états synthétise le workflow complet du sous-système d’abonnement. Il éclaire notamment la logique de reprise après un échec de paiement, ainsi que la possibilité pour l’utilisateur de revenir ultérieurement à l’état « Active » avant l’archivage définitif.

### 4.2.2 Diagramme d’états – Génération de contenu IA

*9* *Diagramme d’états – Génération de contenu IA*

La figure 8 représente le processus interne que suit chaque **requête de génération de contenu IA** (texte → TTS, conversion vocale, SFX, images, vidéos). Il couvre l’ensemble des étapes, depuis la réception de la requête de l’utilisateur jusqu’à la livraison du contenu généré — en passant par la validation, le traitement IA, la vérification qualité, puis le stockage. Il inclut également les chemins menant à un échec ou un rejet du contenu.

**

**États**

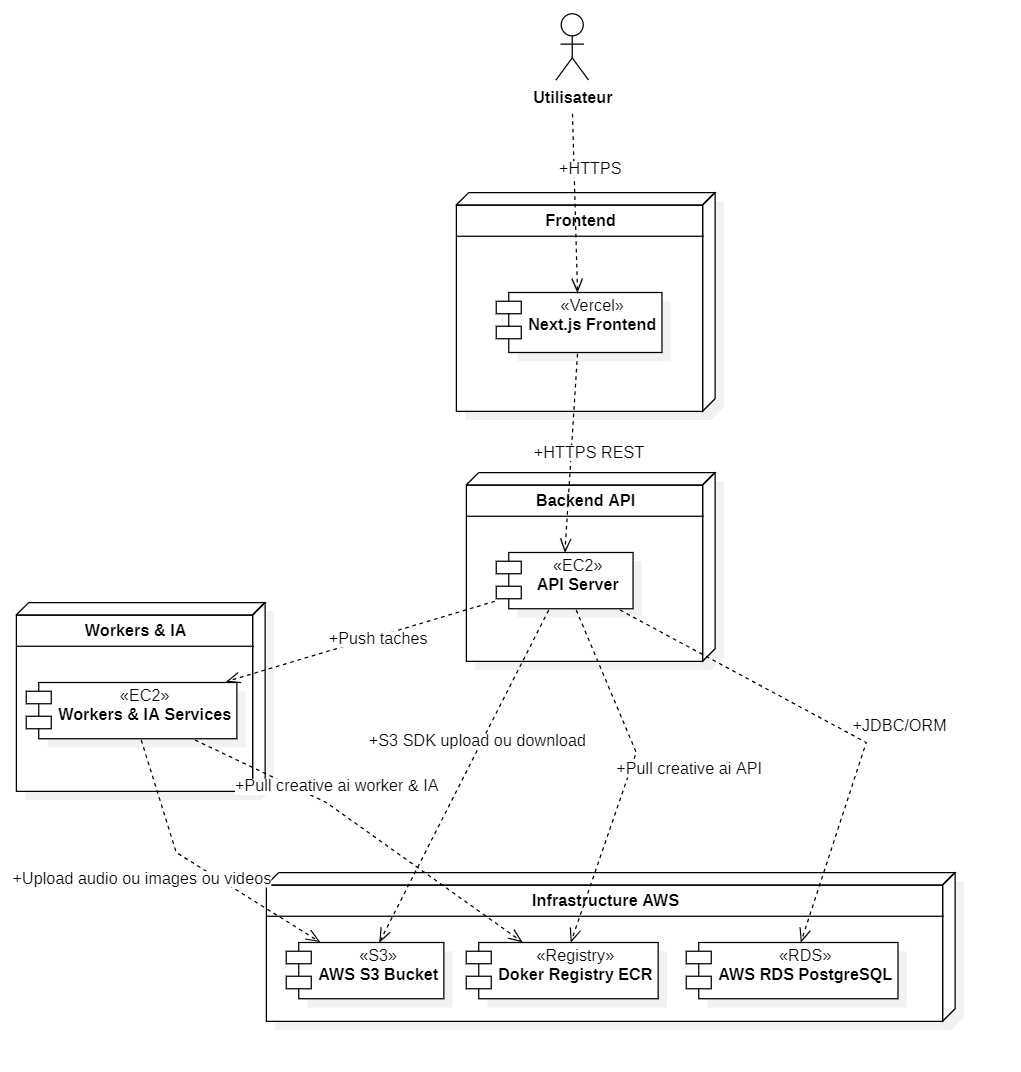
1. **Requête Reçue**
   * Description : Le système (backend FastAPI) a reçu une nouvelle requête de génération de la part de l’utilisateur (texte, prompt ou audio source).
   * Transition « vérifier format d’entrée » → Validation des Paramètres
   * Notes : À ce stade, on ne fait que consigner la demande.
2. **Validation des Paramètres**
   * Description : Le système vérifie que tous les champs nécessaires sont présents et conformes (taille de texte acceptable, voix cible valide, format audio raisonnable, etc.).
   * Si paramètres invalides (champ manquant, voix inconnue, texte trop long…) → État Terminal (Erreur)
   * Si paramètres valides → En Traitement IA
3. **En Traitement IA**
   * Description : Le modèle IA spécifique (StyleTTS2, Seed-VC, Make-An-Audio, Stable Diffusion, ou VideoGen) est invoqué. Les opérations suivantes sont effectuées :
     1. Transformation du texte (phonémisation, tokenization) ou du prompt en entrée,
     2. Exécution du réseau de neurones pour générer le waveform ou l’image / vidéo,
     3. Génération d’un fichier audio, image ou vidéo localement.
   * Transition « génération réussie » → Contenu Généré
   * Transition « erreur de modèle » → État Terminal (Erreur)
4. **Contenu Généré**
   * Description : Le contenu brut (ex. fichier WAV pour TTS, PNG pour image, MP4 pour vidéo) est désormais disponible en local (ou dans le répertoire temporaire).
   * Transition « analyse des données » → Vérification Qualité
5. **Vérification Qualité**
   * Description : Routine automatique qui évalue la qualité du contenu généré :
     1. Pour l’audio : vérification que le signal n’est pas corrompu, niveau RMS acceptable, absence de saturations extrêmes, etc.
     2. Pour l’image : détection d’artefacts (murs déformés, zones floues), contrôle de la résolution attendue, cohérence du contenu visuel.
     3. Pour la vidéo : fluidité du flux, absence de frames noires, cohérence audio/vidéo.
   * Si anomalies détectées (artefacts trop prononcés, perte d’information vocale…) → Contenu Rejeté
   * Si qualité approuvée → Contenu Approuvé
6. **Contenu Rejeté**
   * Description : Le filtre qualité a jugé le résultat inacceptable (trop de bruit, artefacts importants, contenu incohérent).
   * Transition « 3 échecs consécutifs » → État Terminal (Erreur Irréversible)
   * Transition « tentative de correction » (non représentée explicitement, mais possible : repasser en « En Traitement IA » avec de nouveaux paramètres)
   * Notes : Si l’on atteint 3 tentatives infructueuses, on abandonne définitivement la génération pour cette requête et on logge l’erreur.
7. **Contenu Approuvé**
   * Description : Le contenu a passé les contrôles qualité.
   * Transition « stocker sur le serveur et générer les liens » → Contenu Livré
8. **Contenu Livré**
   * Description : Le fichier final (URL + métadonnées) est envoyé au système de stockage (AWS S3) et un pré-signed URL est renvoyé à l’utilisateur (via l’API).
   * Transition « cycle terminé » → État Terminal (Succès)
9. **État Terminal (Succès / Erreur)**
   * Deux points terminaux différenciés :
     1. **Succès** : lorsque le contenu a été validé, stocké dans S3, et le lien est envoyé à l’utilisateur.
     2. **Erreur** : si une des validations (paramètres ou modèle ou qualité) échoue de façon irréversible.

**Conclusion** Le diagramme d’états « Génération de contenu IA » présente de manière exhaustive toutes les étapes nécessaires pour passer d’une simple requête utilisateur à un actif multimédia finalisé, en incluant les logiques de validation et de reruns en cas d’erreurs. Il permet de comprendre où et pourquoi une génération peut être interrompue ou rejetée, et illustre bien le workflow complexe d’un microservice IA en production.

## 4.3 Diagramme de déploiement

*10* *Diagramme de déploiement de la plateforme Creative AI*

La figure 9 présente l’architecture de déploiement de Creative AI, illustrant la répartition des composants techniques sur AWS et Vercel. L’utilisateur accède au frontend hébergé serverless sur Vercel, lequel communique en HTTPS avec le backend FastAPI déployé dans un conteneur Docker sur une instance EC2. Ce backend gère la réception des requêtes (authentification, génération de contenus, historique) et orchestre les tâches lourdes, qu’il soumet à un worker également exécuté sur EC2 via une file Inngest. Les données structurées (comptes, abonnements, métadonnées) sont persistées dans une base PostgreSQL RDS, tandis que tous les objets volumineux (audio, image, vidéo, voix) sont stockés dans un bucket S3. Enfin, chaque instance EC2 récupère ses images Docker depuis un registry ECR, assurant une cohérence de versions et une mise à jour sécurisée via IAM. Cette organisation découple le frontend du backend, facilite l’extensibilité des workers IA et centralise le stockage pour garantir performance et fiabilité.

**

1. **Utilisateur → Frontend (Vercel)**
   * L’acteur humain accède à l’application via l’URL hébergée sur Vercel, qui déploie le Next.js App.
2. **Frontend → Backend API (EC2)**
   * Les requêtes de l’interface (authentification, génération, historique) transitent en HTTPS vers l’instance EC2 hébergeant le conteneur creative-ai-api (FastAPI).
3. **API EC2 → Datastores**
   * AWS RDS PostgreSQL stocke les métadonnées : utilisateurs, abonnements, requêtes, paiements, clips.
   * AWS S3 Bucket stocke les fichiers volumineux (audio, images, vidéos), organisés en répertoires.
4. **API EC2 → Worker & IA Services (EC2)**
   * L’API place les jobs lourds dans une file (Inngest) gérée par le WorkerNode, qui exécute les modèles IA.
5. **WorkerNode → AWS S3**
   * Après traitement (StyleTTS2, Seed-VC, Make-An-Audio…), le worker upload les résultats dans le même bucket S3.
6. **Déploiement et mise à jour des conteneurs**
   * Toutes les instances EC2 (API et WorkerNode) pull leurs images Docker depuis ECR, garantissant la cohérence des versions.

Cette architecture répartit clairement les responsabilités et facilite la scalabilité :

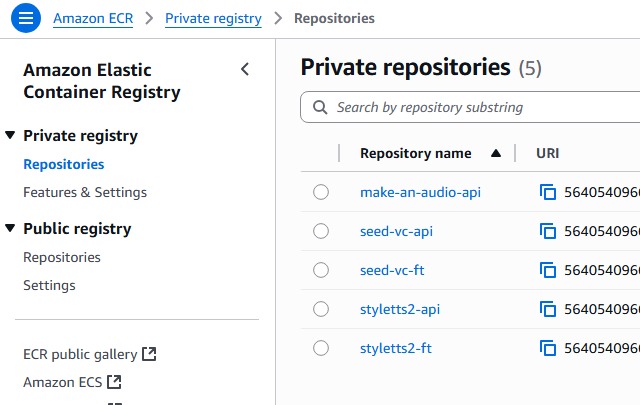
* **Séparation du frontend** serverless (Vercel) du backend conteneurisé (EC2).
* **Évolutivité horizontale** possible pour les workers et microservices IA.
* **Stockage centralisé** sur S3 + persistance relationnelle sur RDS.

**Gestion de version et sécurité** via ECR pour les images Docker et IAM pour les accès AWS.  
Pour illustrer concrètement l'infrastructure déployée sur AWS, les captures suivantes montrent les services effectivement utilisés : Amazon S3 pour le stockage des fichiers générés, Amazon ECR pour la gestion des images Docker, et les instances EC2 sur lesquelles sont déployés les microservices IA.

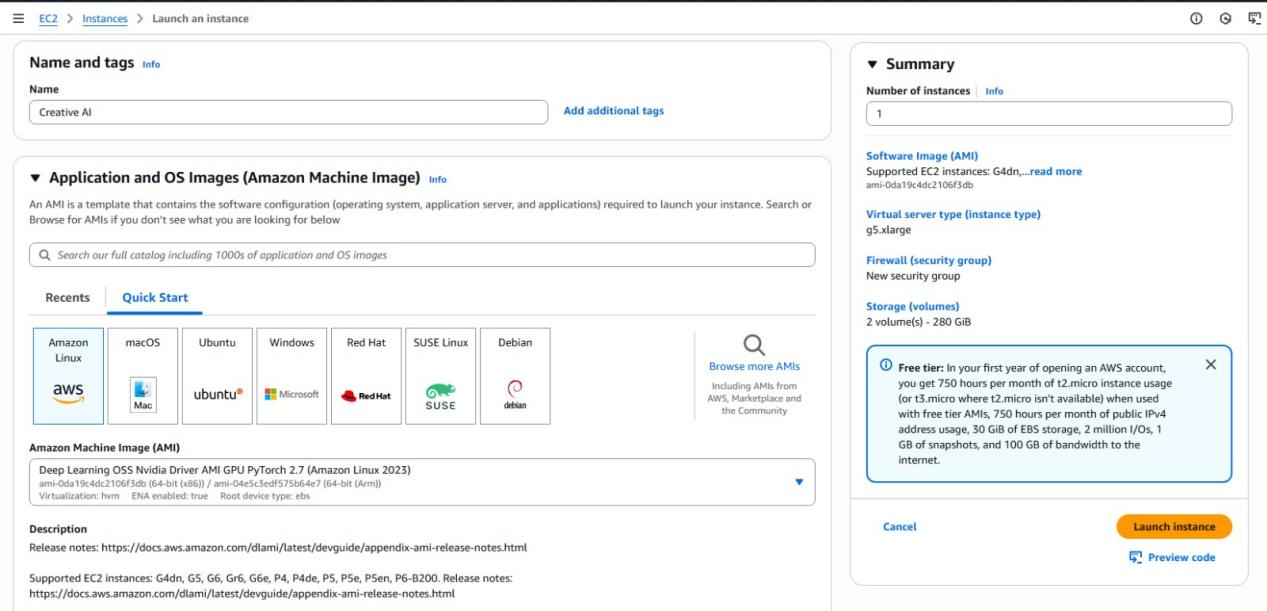
*11* *Figure Console AWS S3 (stockage)*



*12* *Figure– Console AWS ECR (registre d’images Docker)*



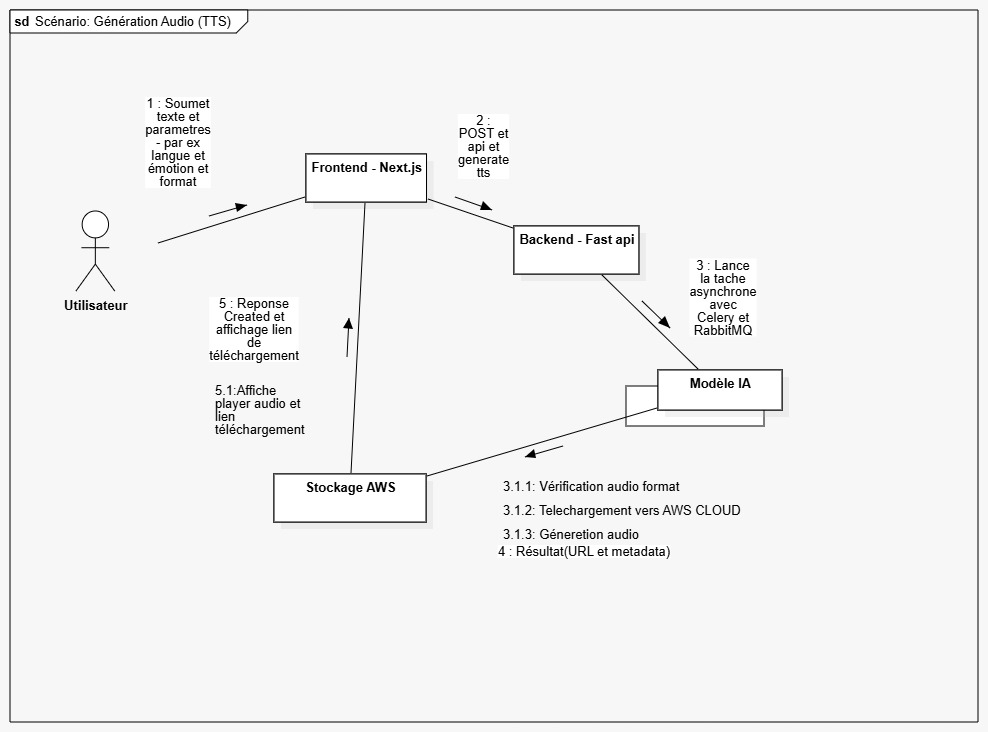
*13* *Figure – Instance EC2 (serveur PyTorch)*



## 4.4 Diagramme de communication

*14* *Diagramme de communication : génération de contenu audio*

La figure 13 met l’accent sur les **messages échangés** entre les différents objets/acteurs lors d’un scénario type de génération audio (TTS). Contrairement au diagramme de séquence, il ne montre pas la chronologie sur un axe vertical, mais illustre les interactions et les liaisons entre les entités impliquées : l’utilisateur, le frontend (Next.js), le backend (FastAPI), le modèle IA et le stockage AWS (S3).

**

**Objets et acteurs impliqués :**

1. **Utilisateur** (Actor) : l’utilisateur final qui soumet un texte et choisit des paramètres (langue, style, format).
2. **Frontend (Next.js)** : application web qui collecte les paramètres de l’utilisateur et envoie la requête au backend.
3. **Backend (FastAPI)** : point de terminaison /generate qui reçoit la requête, vérifie l’API key, et déclenche la génération de manière asynchrone (via Celery + RabbitMQ ou Inngest).
4. **Modèle IA** : composant de génération (StyleTTS2). Il réalise :
   * Vérification du format audio ou préparation du style (compute\_style)
   * Génération du waveform
   * Retour du buffer audio
5. **Stockage AWS (S3)** : bucket où l’on stocke le fichier WAV généré, puis on crée et renvoie une URL pré-signée pour le téléchargement ou la lecture en streaming.

**Messages et interactions**

Les numéros entre parenthèses font référence aux étapes annotées sur le diagramme :

1. **Soumettre texte et paramètres**
   * Utilisateur envoie au Frontend : { texte, style, format }.
   * Le frontend prépare un formulaire JSON (POST /api/generate) et affiche un spinner d’attente.
2. **POST API /generate (HTTP)**
   * Le Frontend envoie une requête HTTP POST /generate au Backend (FastAPI), avec en-tête Authorization: Bearer <API\_KEY>.
   * Le backend valide la clé (verify\_api\_key) puis passe l’appel à la fonction generate\_speech (TTS).
3. **Lancer la tâche asynchrone**
   * Le Backend (FastAPI) pousse un nouveau job dans la queue (Celery + RabbitMQ ou Inngest) avec les paramètres.
   * Le Modèle IA (ou plus précisément le worker Celery qui encapsule le modèle) consomme le job :
     + **(3.1.1)** Vérification du format audio (pour seed-VC, ou préparation du texte (phonemization) pour TTS).
     + **(3.1.2)** Téléchargement de ressources éventuelles sur AWS (style audio de référence).
     + **(3.1.3)** Exécution de la génération (inférence StyleTTS2).
     + **(3.1.4)** Le worker écrit le fichier temporaire en local.
4. **Upload vers AWS S3**
   * Une fois la génération terminée, le Modèle IA (ou le worker) appelle s3\_client.upload\_file(local\_path, bucket, key) pour déposer le fichier WAV dans le bucket S3.
   * L’objet est stocké dans un dossier préfixé (styletts2-output/<uuid>.wav).
5. **(5) Résultat (URL + métadonnées)**
   * Après l’upload, le worker génère une URL pré-signée (3600 s) via generate\_presigned\_url('get\_object', Params={…}).
   * Le worker inscrit ce lien dans la base PostgreSQL (via la couche ORM Prisma) si nécessaire, puis renvoie la réponse (URL + s3\_key) au Backend (FastAPI) via un callback ou une réponse HTTP asynchrone.
6. **(5.1) Afficher le player et le lien**
   * Le Backend retourne une réponse JSON au Frontend contenant { audio\_url, s3\_key }.
   * Le Frontend stoppe le spinner, affiche un composant <audio> HTML avec src = audio\_url et propose un bouton ou un lien direct de téléchargement.

**Comment interpréter ce diagramme de communication**

* + Noeuds : chaque boîte représente une instance (objet), pas nécessairement un serveur physique.
  + Lignes pleines : font référence à une liaison « possède une référence » (e.g., Next.js ↔ Stockage AWS n’existe pas, mais Next.js connaît son endpoint S3 via le backend).
  + Flèches numérotées : représentent les messages ou appels dans l’ordre logique du scénario (1 → 2 → 3 → 4 → 5 → 5.1).
  + BKND → IA Model (étape 3) : montre que le backend ne réalise pas l’inférence directement, il délègue au worker/modèle IA.
  + IA Model → Stockage AWS (étape 4) : le composant IA (ou worker) se charge de déposer le fichier sur S3.
  + Stockage AWS → Frontend (directement ou via backend) : techniquement, le frontend ne « parle » jamais directement à S3 (security), mais reçoit l’URL signée du backend et l’utilise pour récupérer le fichier. Le diagramme simplifie cette étape par un lien direct « Stockage AWS → Frontend » pour indiquer que le player HTML consomme le lien S3.

# Chapitre 5

Implémentation

Dans ce chapitre, nous détaillons la mise en place pratique de Creative AI, depuis l’environnement de développement conteneurisé (Python 3.10, Node.js 18, Pipenv, npm, Docker Compose) jusqu’à l’architecture du code backend (FastAPI : chargement des modèles, sécurisation par API key, endpoints TTS et vidéo) et frontend (Next.js avec NextAuth.js pour l’authentification). Nous présentons également l’intégration des interfaces utilisateur – pages Home, Dashboard, Media Library et générateurs multimodaux – avant de décrire l’orchestration asynchrone des tâches IA via Inngest. Cette section pose ainsi les bases nécessaires pour le fine‑tuning des modèles et les tests de performance à venir.

# Implémentation

## 5.1 Environnement de développement

Pour assurer reproductibilité et cohérence, tout le projet est conteneurisé et versionné :

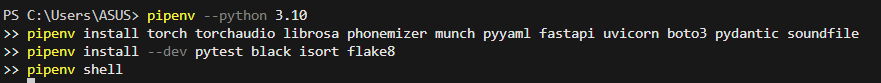
**Langages & versions**

* + Python 3.10 pour le backend (compatible PyTorch 2.x).
  + Node.js 18 + Next.js 15 pour le frontend.

**Gestion des dépendances**

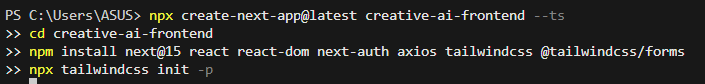
**Backend : Pipenv**

*15* *Backend : Pipenv*

**

**Frontend : npm**

*16* *Frontend : npm*

**

**Conteneurisation**

Chaque service dispose de son propre Dockerfile :

* + Backend : Dockerfile basé sur python:3.10.
  + Worker : clone du backend, incluant Inngest et ses dépendances.
  + Frontend : Dockerfile optimisé pour Next.js (Node.js).

Un fichier docker-compose.yml regroupe l’API, le Worker, un reverse proxy NGINX et LocalStack (pour simuler AWS en local).

**Outils & IDE**

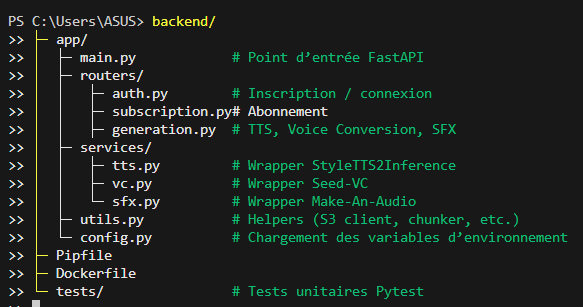
* **VS Code** (extensions : Python, Pylance, ESLint, Prettier).
* **Git + GitHub** (CI via GitHub Actions : lint, tests, build).
* **Postman** pour tester manuellement les endpoints.

## 5.2 Développement du backend FastAPI

Le backend expose les services IA et gère l’authentification, la facturation et le stockage.

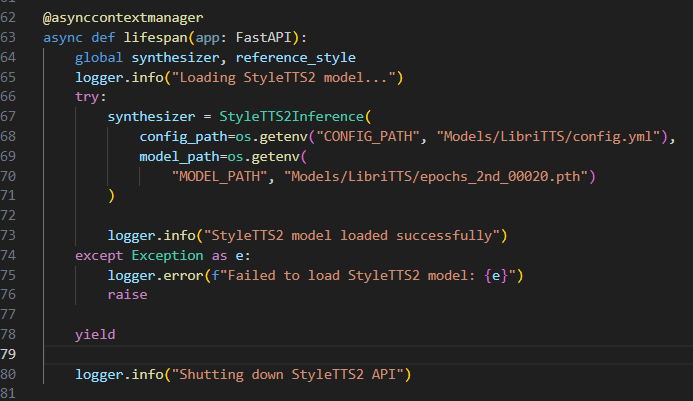
### 5.2.1 Structure du projet

*17* *Structure du projet*

**

### 5.2.2 Chargement du modèle et cycle de vie

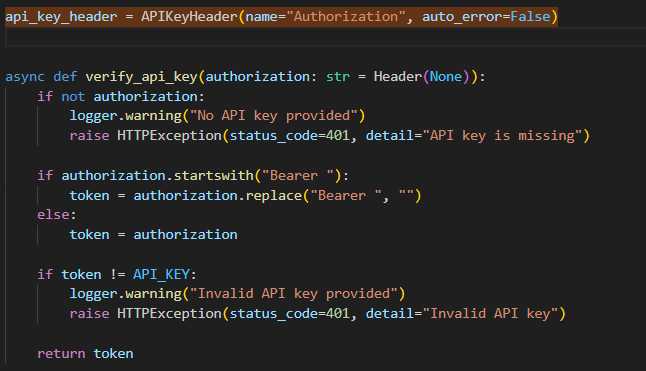
*18* *Chargement du modèle et cycle de vie*

**

* + Objectif : charger StyleTTS2 Inference une seule fois au démarrage
  + Bénéfice : latence constante pour les inférences suivantes.

### 5.2.3 Sécurisation par API Key

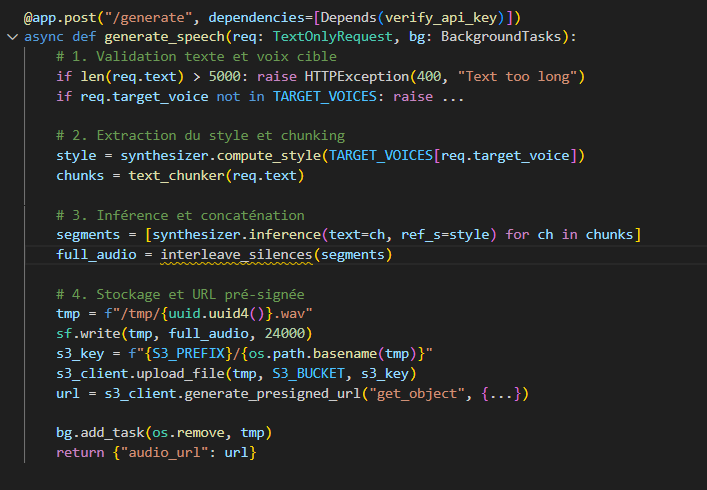
*19* *Sécurisation par API Key*

**

* + Chaque route sensible inclut Depends(verify\_api\_key).
  + Simplicité adaptée aux clients internes B2B.

### 5.2.4 Endpoint de génération TTS

*20* *Endpoint de génération TTS*

**

**Appel /generate dans Postman et réponse JSON avec audio\_url**

L’endpoint /generate permet de soumettre un texte et une voix cible via une requête POST. L’API FastAPI traite la demande en générant l’audio correspondant et renvoie une réponse JSON contenant l’URL pré-signée du fichier audio stocké sur AWS S3.  
 **Historique des fichiers WAV dans AWS S3**

Tous les fichiers audio générés sont automatiquement stockés dans un bucket S3 spécifique, sous le préfixe "styletts2-output". L’utilisation de S3 permet un stockage fiable et une distribution rapide des fichiers via des URL pré-signées, valides pour une durée limitée.

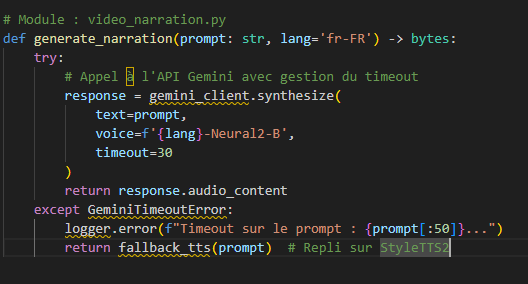
5.2.5 Endpoint de Génération Vidéo avec Intégration GeminiAI  
Contenu :

Pour répondre à l'exigence de narration synchronisée spécifiée dans le cahier des charges, nous avons implémenté un pipeline de génération vidéo combinant VideoGPT et GeminiAI. Ce workflow en trois étapes assure une cohérence sémantique entre le visuel et la narration audio :

1. Génération des séquences visuelles via VideoGPT à partir du prompt texte
2. Synthèse de la piste narrative avec l'API GeminiAI (version Gemini-1.5)
3. Synchronisation audiovisuelle par alignement temporel utilisant FFmpeg

L'intégration de GeminiAI se fait via une couche d'abstraction Python qui gère le cache et la reprise sur erreur :

*21 Capture de l'interface vidéo avec sélecteur GeminiAI intégré*

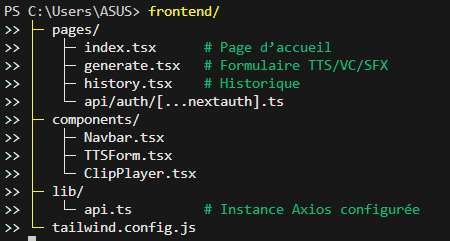


## 5.3 Intégration Next.js et NextAuth.js

Le frontend fournit l’interface utilisateur réactive et sécurisée.

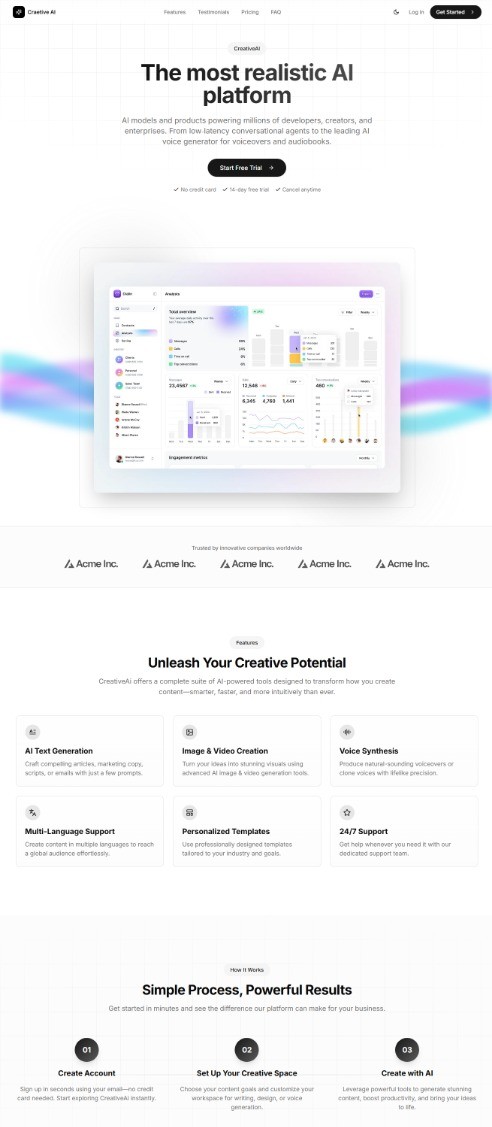
### 5.3.1 Structure du projet

*22 Structure du projet*

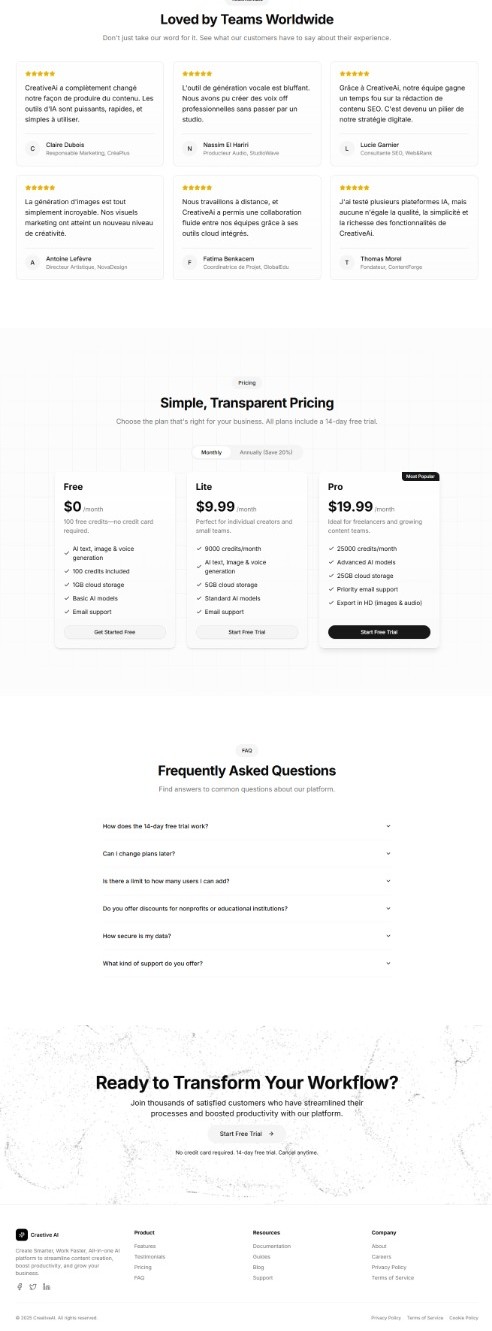
**

### Figure 5.3.2 – Page d’accueil (Home)

*23* *Page d’accueil 1*



*24* *Page d’accueil 2*



**Description :**  
La page d’accueil sert de point d’entrée à la plateforme. On y trouve :

* Un bandeau supérieur avec le logo et les liens « Login » / « Sign Up ».
* Des cartes de présentation des services IA (TTS, VC, SFX, Images, Vidéos) cliquables.
* Un encart « À propos » résumant la mission de Creative AI.

**Fonctionnalités couvertes :** navigation principale, accès rapide aux générateurs, présentation succincte du projet.

### Figure 5.3.3 – Tableau de bord (Dashboard)

*25* *Tableau de bord*



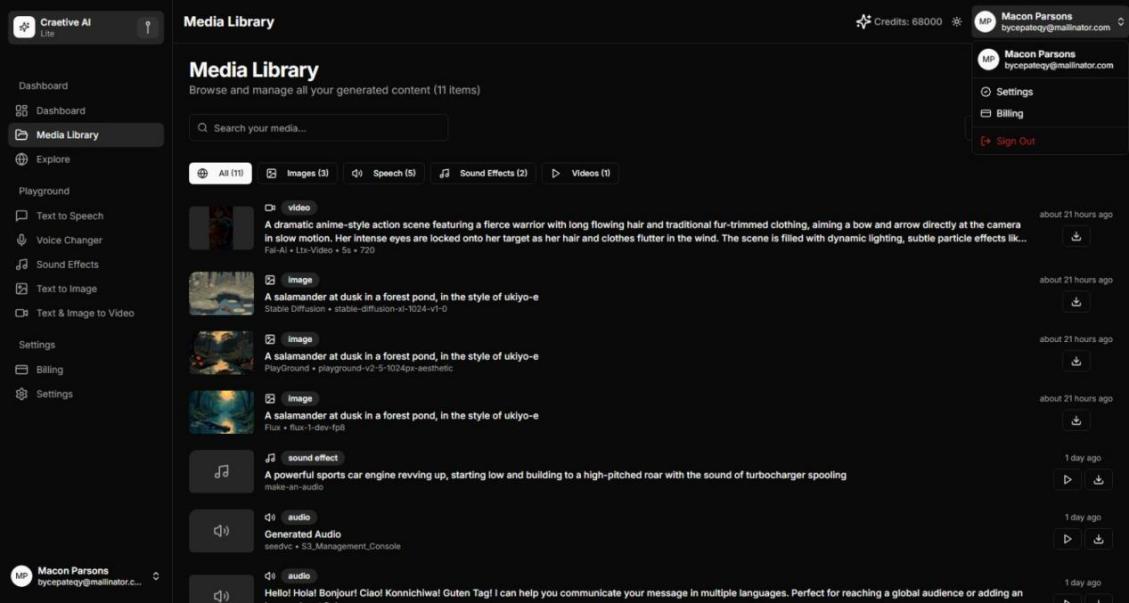
**Description :**  
Après authentification, l’utilisateur accède à un tableau de bord personnalisé comportant :

* Un widget « Crédits restants ».
* Des raccourcis vers les générateurs IA.
* Un graphique d’activité recentrée (nombre de requêtes par service).
* Un aperçu de l’historique des dernières créations.

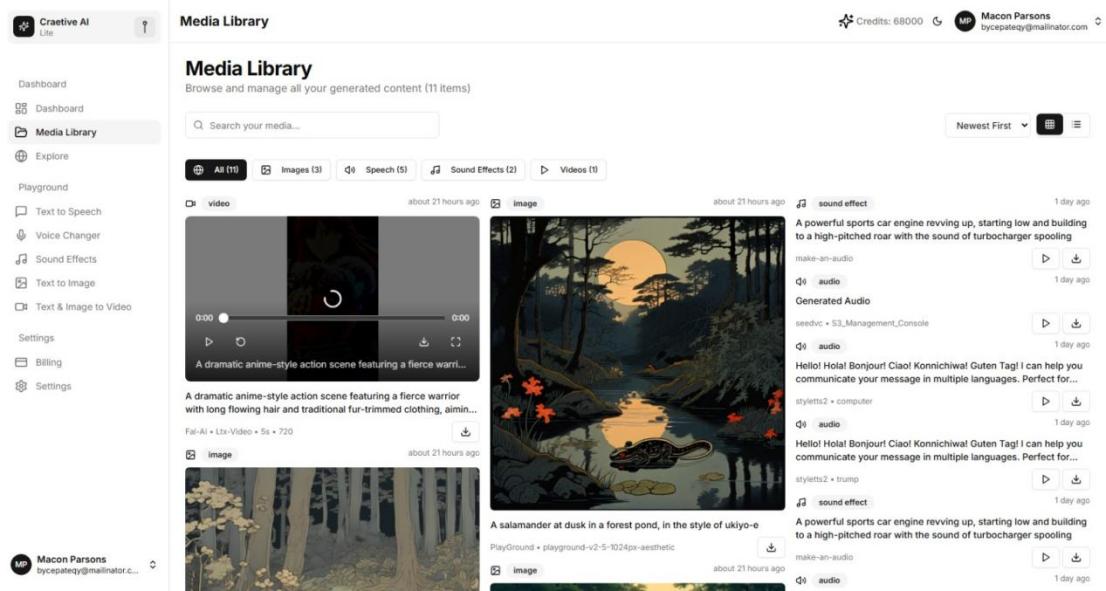
**Fonctionnalités couvertes :** suivi de l’utilisation, gestion des crédits, accès rapide aux dernières tâches.

### Figure 5.3.4 – Media Library

*26* *Media Library*



*27* *Media Library 2*



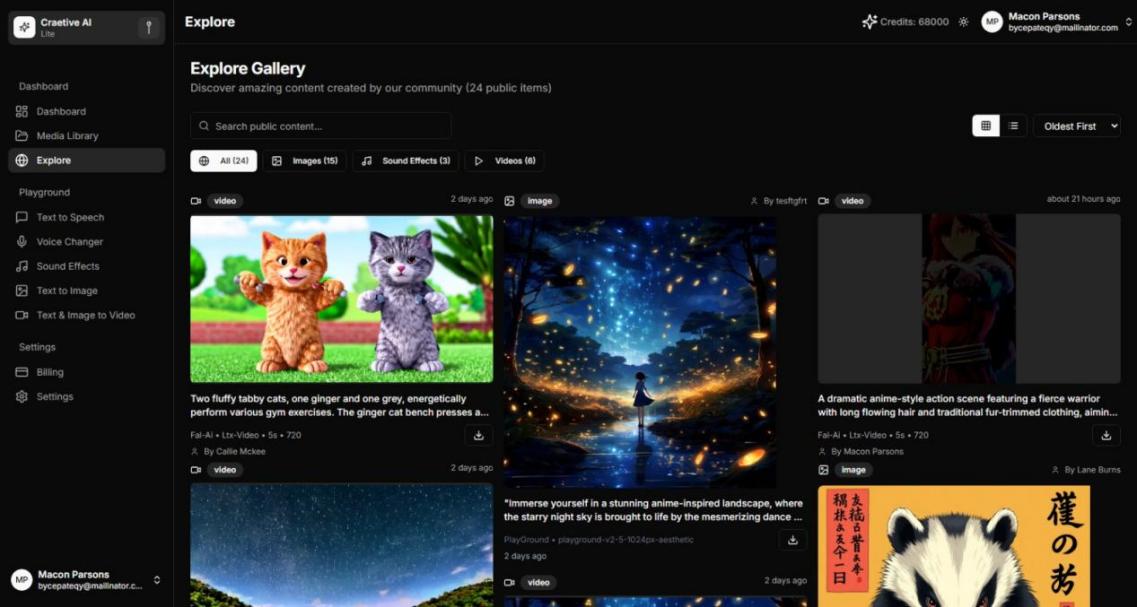
**Description :**  
Cette page liste tous les médias générés par l’utilisateur :

* Vignettes (audio, image, vidéo) avec date de création.
* Boutons de lecture / prévisualisation et de téléchargement.
* Filtres par type de contenu et date.

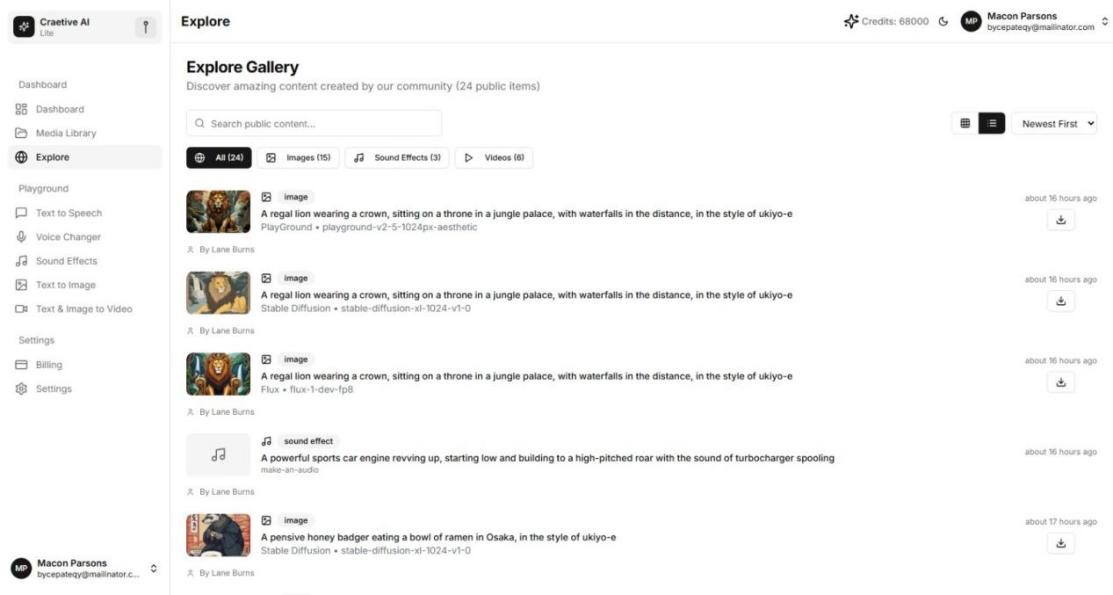
**Fonctionnalités couvertes :** gestion et réutilisation des fichiers, filtrage, téléchargement.

### Figure 5.3.5 – Explore

*28* *Explore 1*



*29* *Explore 2*



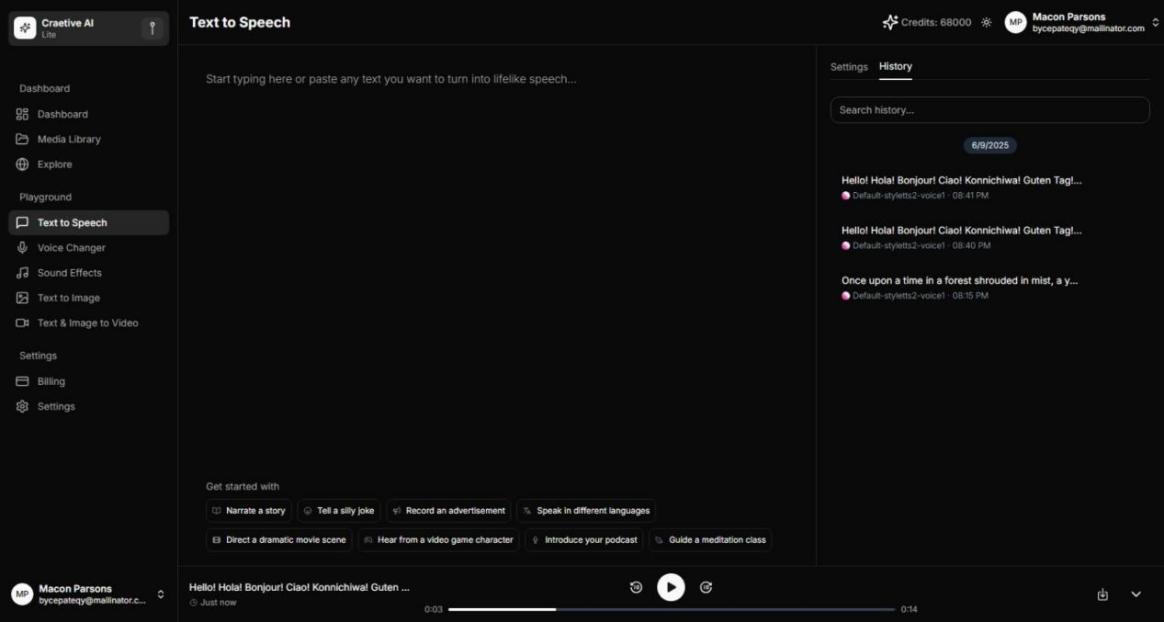
**Description :**  
Un espace d’inspiration où l’utilisateur peut :

* Visualiser des exemples de prompts et résultats.
* Copier un template pour l’insérer dans un formulaire de génération.
* Filtrer par catégorie (TTS, SFX, Image, Vidéo).

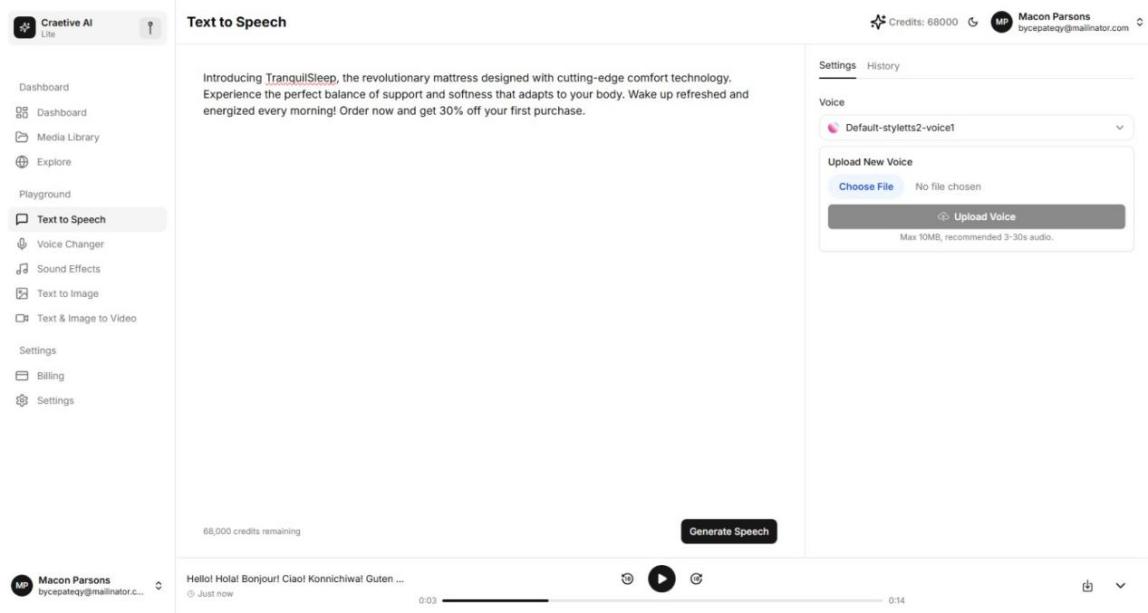
**Fonctionnalités couvertes :** découverte des possibilités, partage de templates, inspiration créative.

### Figure 5.3.6 – Générateur Text-to-Speech

*30* *Générateur Text-to-Speech 1*



*31* *Générateur Text-to-Speech 2*



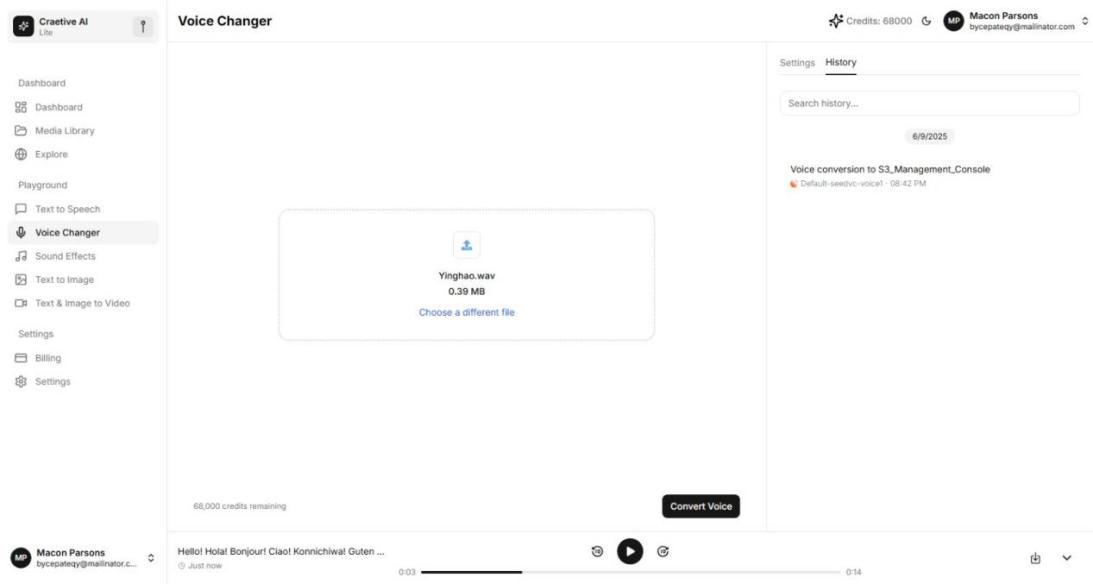
**Description :**  
Interface dédiée à la synthèse vocale :

* Zone de saisie du texte.
* Sélecteur de voix cible (ex. « Anger », « Woman »).
* Curseur d’intensité émotionnelle.
* Bouton « Générer » et player audio pour écouter le résultat.

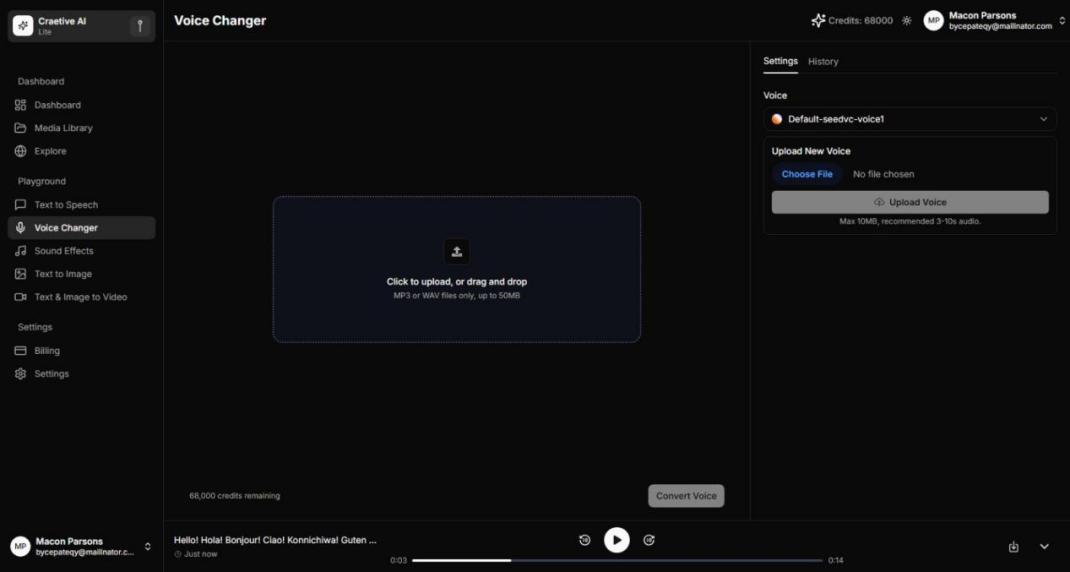
**Fonctionnalités couvertes :** inférence TTS, réglage de style, écoute et téléchargement.

### Figure 5.3.7 – Générateur Voice-to-Voice

*32* *Générateur Voice-to-Voice 1*



*33* *Générateur Voice-to-Voice 2*



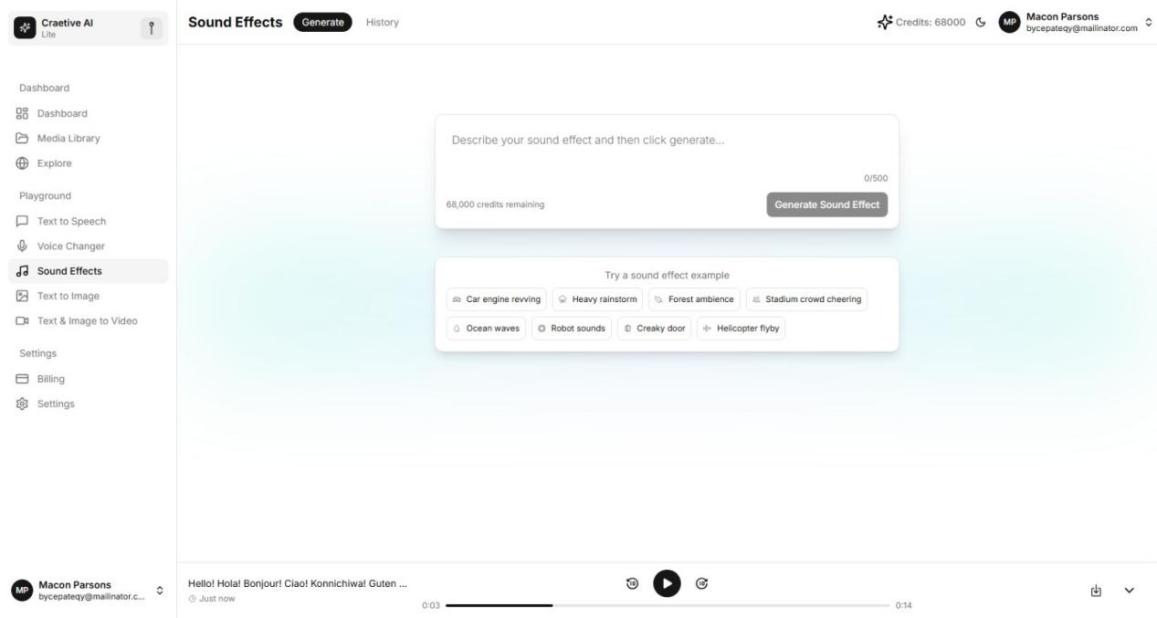
**Description :**  
Permet de convertir une voix existante :

* Upload d’un fichier audio source.
* Choix de la voix cible.
* Bouton « Convertir » et lecteur audio pour préécoute.

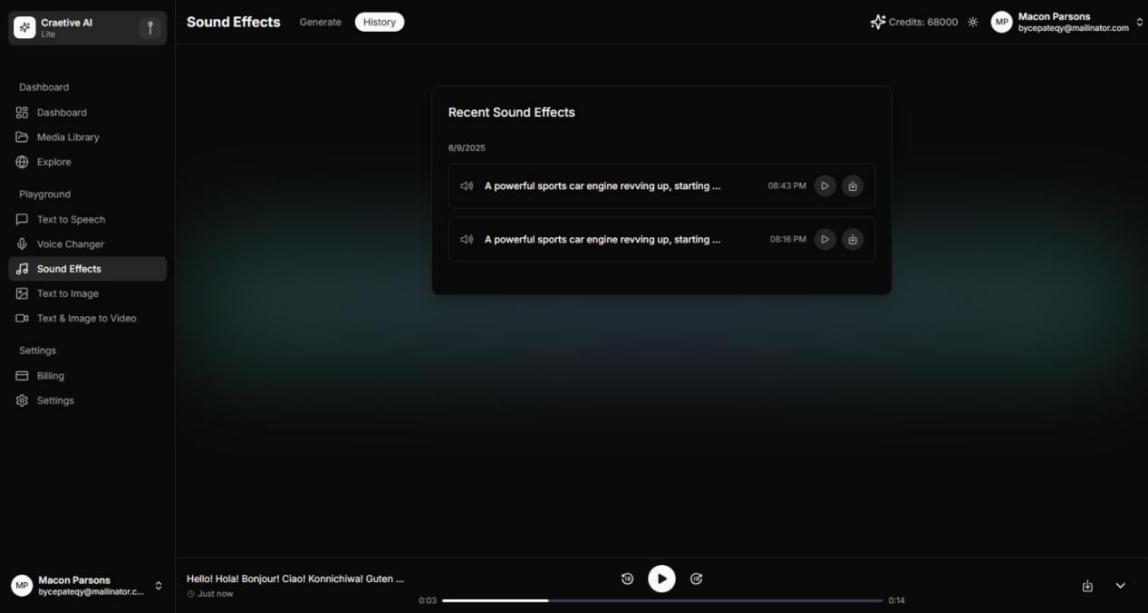
**Fonctionnalités couvertes :** clonage vocal zero-shot, upload/download de fichiers, prévisualisation.

### Figure 5.3.8 – Générateur Text-to-Sound Effect

*34* *Générateur Text-to-Sound Effect 1*



*35* *Générateur Text-to-Sound Effect 2*



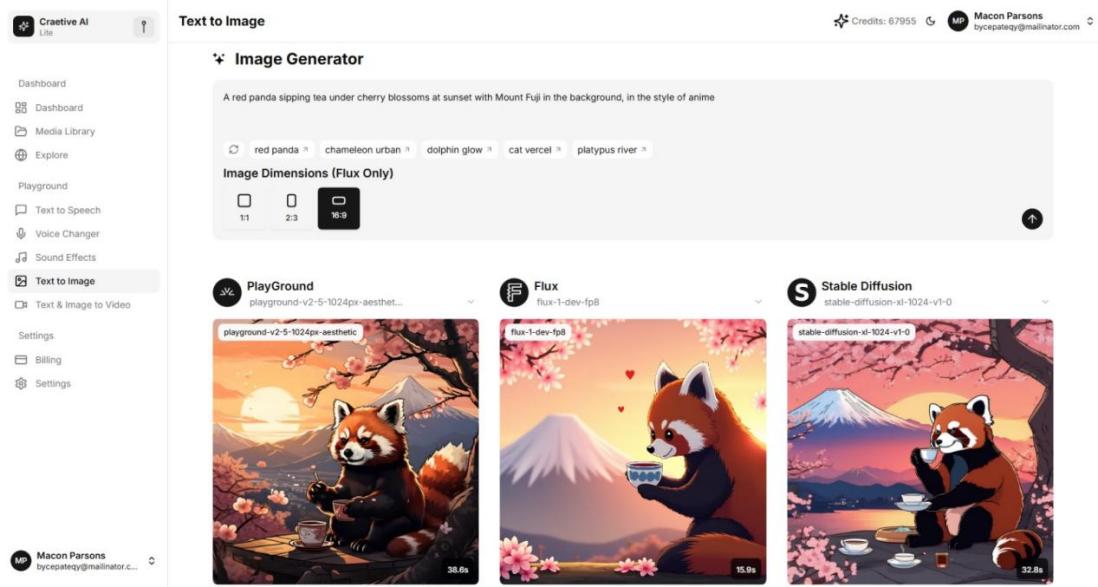
**Description :**  
Module de création d’effets sonores à partir d’un prompt texte :

* Champ de saisie libre (ex. « pluie légère », « explosion »).
* Bouton « Générer » et lecteur audio intégré.

**Fonctionnalités couvertes :** inférence SFX, écoute et export audio.

### Figure 5.3.9 – Générateur Text-to-Image

*36* *Générateur Text-to-Image*



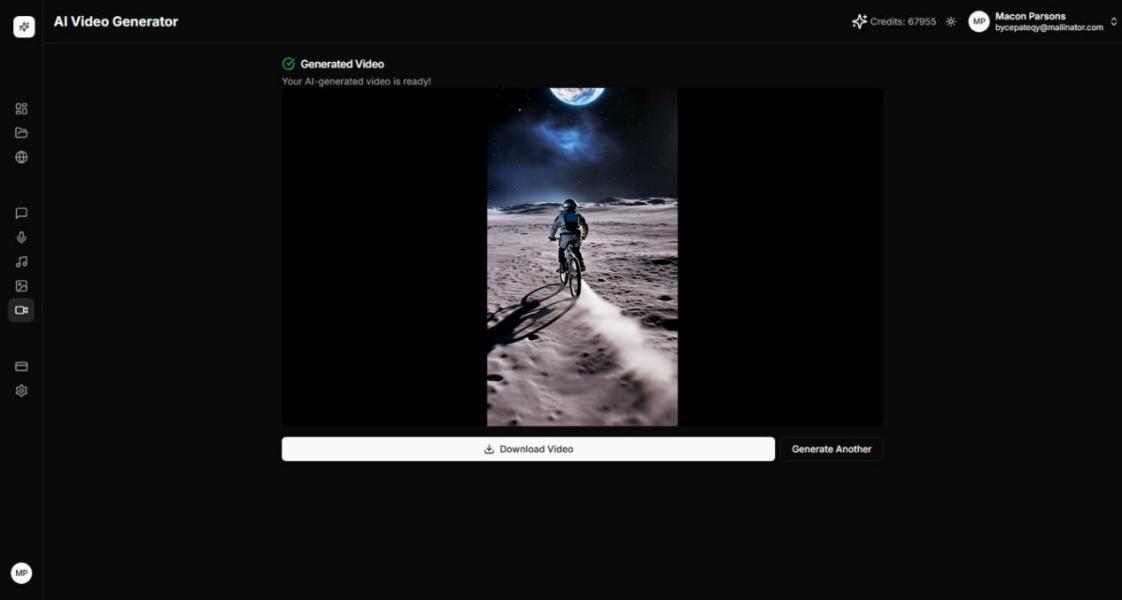
**Description :**  
Interface de génération d’images :

* Zone de texte pour le prompt.
* Sélection de style (réaliste, cartoon, abstrait).
* Bouton « Générer » et galerie de vignettes pour prévisualiser.

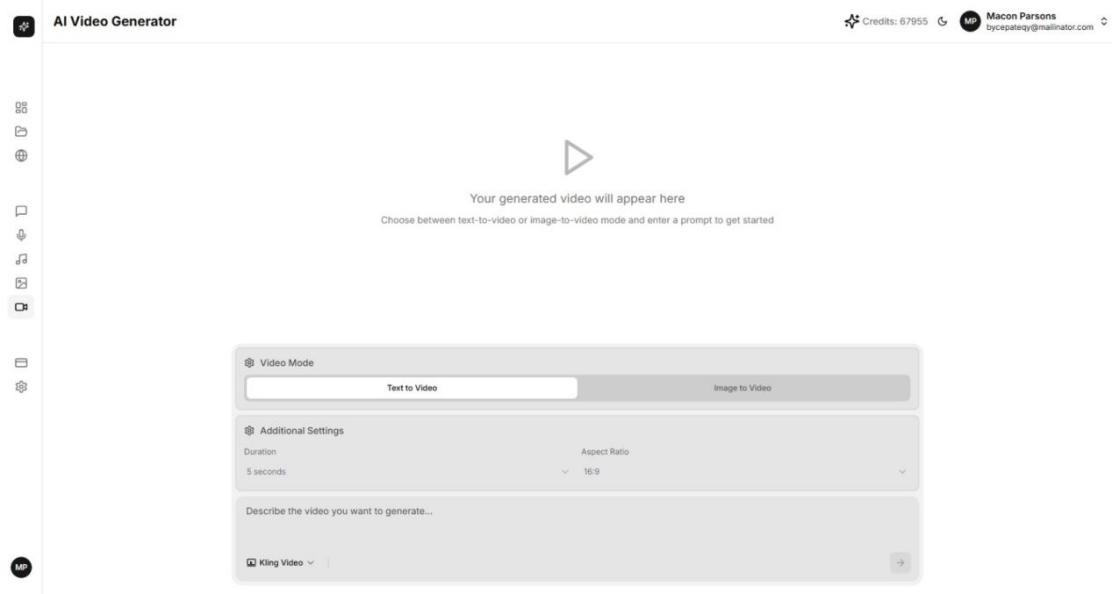
**Fonctionnalités couvertes :** inférence image, styles configurables, téléchargement PNG.

### Figure 5.3.10 – Générateur Text & Image-to-Video

*37* *Générateur Text & Image-to-Video 1*



*38* *Générateur Text & Image-to-Video 2*



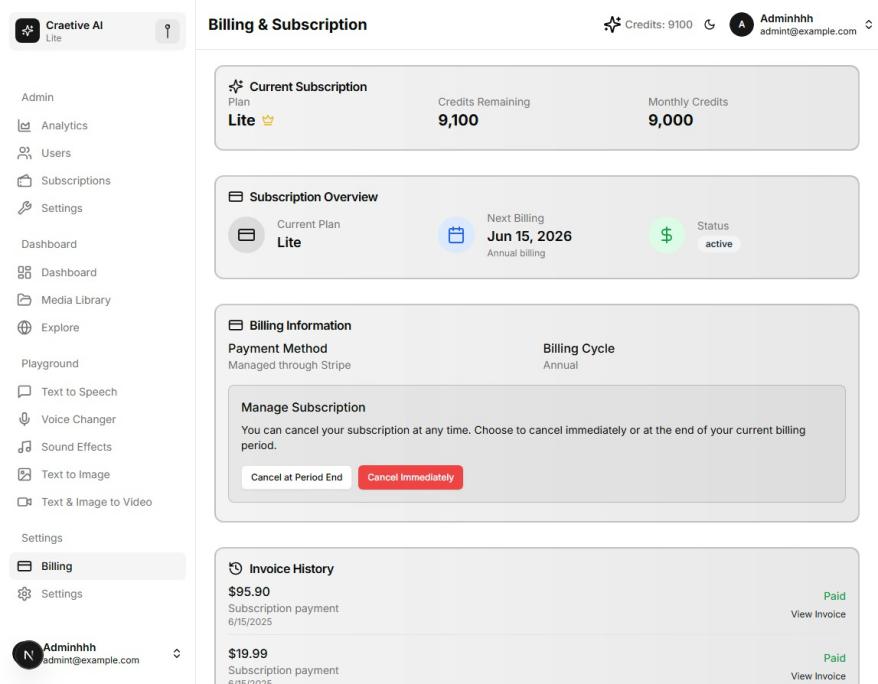
**Description :**  
Page de génération vidéo multimodale :

* Champ de texte + upload d’image de référence.
* Options de résolution (720p).
* Bouton « Créer la vidéo » et lecteur vidéo pour prévisualisation.

**Fonctionnalités couvertes :** inférence vidéo, mix texte-image, export MP4.

### Figure 5.3.11 – Facturation (Billing)

*39* *Facturation*



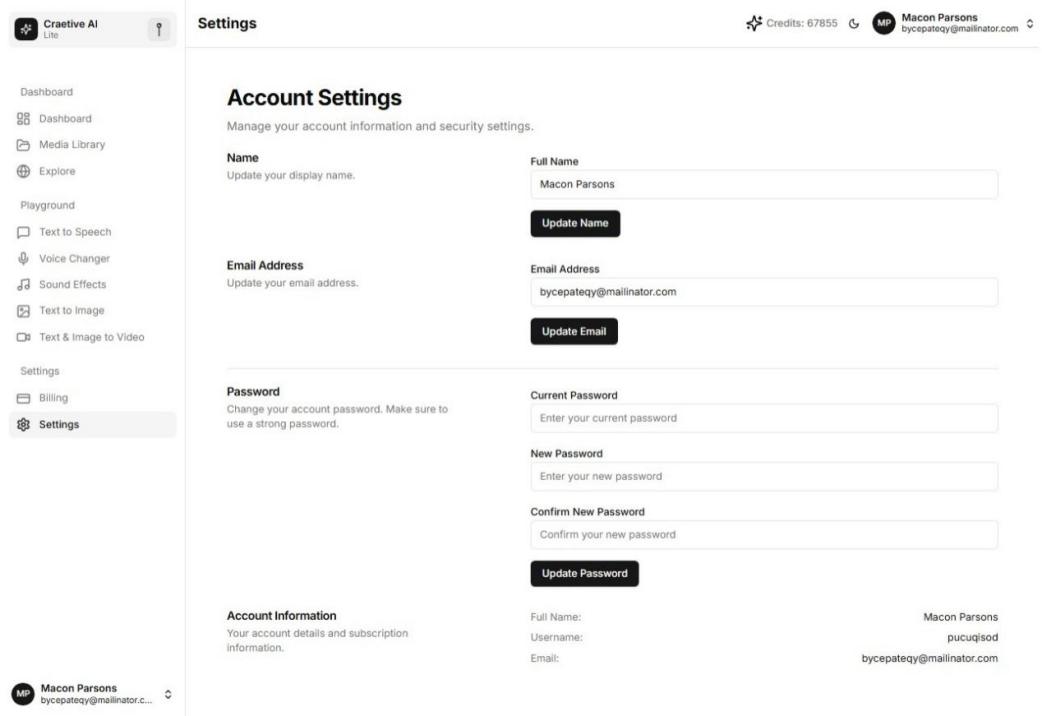
**Description :**  
Espace de gestion des abonnements et des paiements :

* Tableau des plans disponibles (Free, Pro, Premium).
* Historique des transactions (Stripe).
* Bouton « Mettre à niveau » pour changer de forfait.

**Fonctionnalités couvertes :** sélection de plan, paiement sécurisé, suivi du statut d’abonnement.

### Figure 5.3.12 – Paramètres (Settings)

*40* *Paramètres*



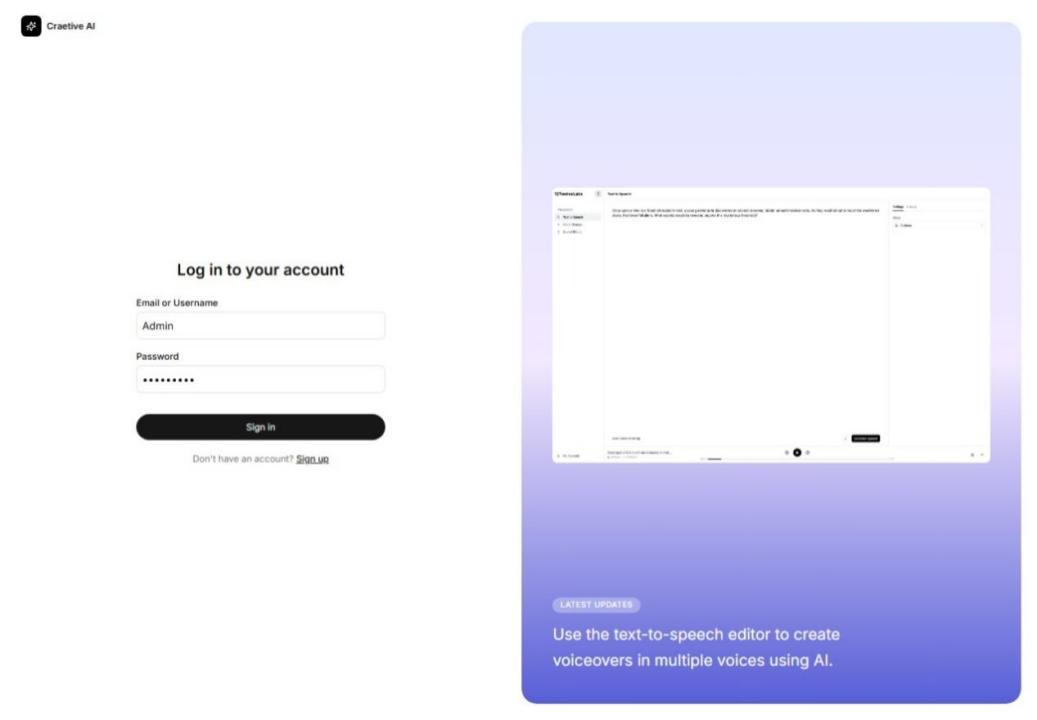
**Description :**  
Page de configuration utilisateur :

* Informations personnelles (nom, email).
* Préférences linguistiques.
* Gestion du consentement RGPD pour le clonage vocal.

**Fonctionnalités couvertes :** mise à jour du profil, RGPD, préférences.

### Figure 5.3.13 – Écran de connexion (Login)

*41* *Écran de connexion*



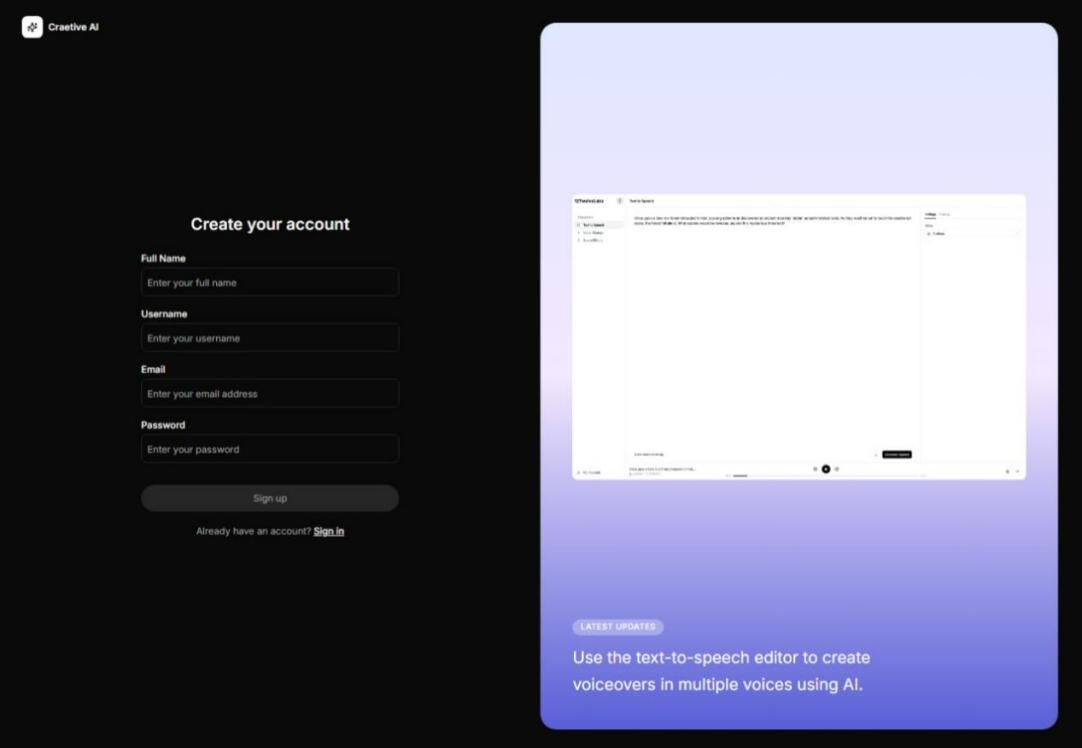
**Description :**  
Formulaire d’authentification :

* Email / mot de passe.
* Boutons de connexion OAuth par Email.

**Fonctionnalités couvertes :** sécurisation des accès, prise en charge OAuth.

### Figure 5.3.14 – Écran d’inscription (Sign Up)

*42* *Écran d’inscription*



**Description :**  
Formulaire d’enregistrement :

* Email, mot de passe, confirmation.
* Case à cocher RGPD.
* Lien vers la page de connexion.

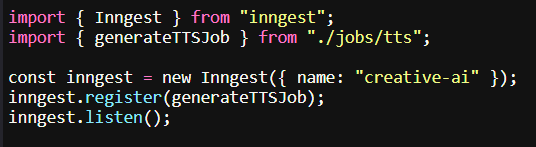
**Fonctionnalités couvertes :** création de compte, consentement.

## 5.4 Orchestration Inngest

Pour délester le serveur web et garantir la résilience, les tâches lourdes (génération TTS, conversion vocale, etc.) sont déléguées à Inngest.

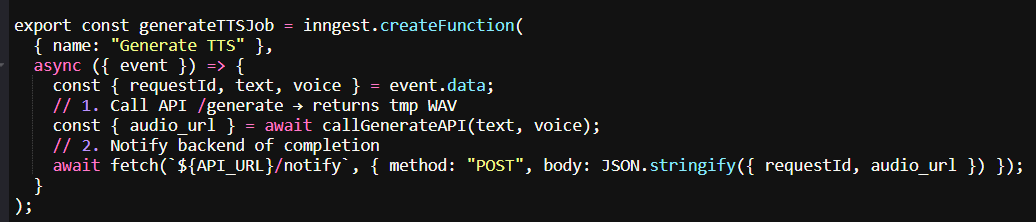
### 5.4.1 Configuration Inngest

*43* *Configuration Inngest*

*;*

### 5.4.2 Définition du job TTS

*44* *Définition du job TTS*

**

**Workflow :**

1. FastAPI émet un événement GenerateTTS via Inngest.
2. Le Worker Inngest traite le job, appelle l’API interne /generate, récupère l’URL du WAV, puis notifie le backend via un webhook (/notify).
3. Le backend enregistre l’URL définitive dans la base de données (clip historique).

**Ce chapitre montre comment les différents composants :**

— modèle IA, API, frontend et orchestrateur asynchrone

— s’intègrent pour offrir une plateforme Creative AI performante, sécurisée et scalable.

**Conclusion du chapitre 5**  
Ce chapitre a mis en lumière la configuration complète de l’environnement, le code central du backend et du frontend, les interfaces utilisateurs et l’orchestration des tâches IA. Il sert de fondation pour les chapitres suivants sur le fine-tuning et les tests.

# Chapitre 6

Fine-tuning et entraînement

des modèles

Ce chapitre décrit la préparation des données, la stratégie de fine-tuning du modèle **StyleTTS2** pour la synthèse vocale, puis celle du modèle **Seed-VC** pour la conversion vocale. Nous présentons enfin les résultats obtenus (courbes de perte et scores MOS).

# Fine-tuning et entraînement des modèles

## 6.1 Préparation des datasets

La qualité du fine-tuning dépend avant tout de la richesse et de la propreté du corpus utilisé. Nous avons structuré cette étape en cinq phases essentielles : collecte, nettoyage, phonétisation, fractionnement et pré-calcul des spectrogrammes.

### 6.1.1 Collecte des enregistrements

Pour chaque modalité d’usage, nous avons rassemblé deux types de corpus :

* + Voix de référence : Un locuteur cible fournit entre 5 000 et 10 000 phrases enregistrées en .wav 24 kHz mono, dans une intonation neutre. Ce jeu de données dédié au locuteur unique permet d’ancrer le style vocal de StyleTTS2 sur une voix homogène.
  + Corpus multilingue : Afin d’assurer la prise en charge de plusieurs langues, nous avons ajouté 20 000 phrases réparties sur dix langues différentes. Cette diversité linguistique garantit que le modèle conserve une robustesse phonétique même en conditions multilingues.

### 6.1.2 Nettoyage et normalisation

Le nettoyage des fichiers audio est primordial pour éviter que le modèle apprenne sur des silences trop longs ou des artefacts indésirables :

1. **Trim des silences** : Nous utilisons la fonction librosa.effects.trim pour supprimer les silences en début et fin de chaque enregistrement, ce qui évite de biaiser la durée et le positionnement des phonèmes.
2. **Resampling** : Tous les fichiers sont systématiquement resamplés à 24 kHz avec librosa.resample, afin d’harmoniser la fréquence d’échantillonnage et de faciliter la comparaison des spectrogrammes.
3. **Contrôle de niveau** : Un calcul du niveau RMS permet de filtrer automatiquement les clips trop faibles (risque de bruit de fond) ou trop saturés (écrêtage), garantissant un signal audio propre pour l’entraînement.

### 6.1.3 Tokenisation et phonétisation

Le texte associé aux fichiers audio est d’abord converti en phonèmes pour améliorer la précision linguistique :

* + Phonémisation : Nous employons eSpeak NG avec conservation de la ponctuation et des indices d’accentuation. Cette étape produit une séquence de phonèmes compatible avec l’encodeur texte de StyleTTS2.
  + Nettoyage orthographique : Un utilitaire TextCleaner normalise la chaîne en minuscules et supprime les caractères non-ASCII ou superflus pour éviter les tokens hors vocabulaire.

### 6.1.4 Fractionnement train / validation / test

Afin d’évaluer correctement la généralisation du modèle, le corpus est divisé selon les proportions suivantes :

* + 80 % pour l’entraînement
  + 10 % pour la validation
  + 10 % pour les tests finaux

La répartition est effectuée par locuteur (shuffle speaker-wise) pour prévenir tout chevauchement de voix entre les ensembles. Un fichier CSV décrit pour chaque exemple le chemin du fichier .wav, la transcription et l’étiquette de split (train, val, test).

### 6.1.5 Pré-calcul des spectrogrammes

Le passage en spectrogrammes de Mel est costaud en I/O si effectué à la volée. Nous avons donc pré-calculé et stocké sous forme de fichiers .npy les spectrogrammes (80 mels, fenêtre 1 200 samples, hop 300 samples) de chaque enregistrement. Cette mise en cache drastique réduit le temps de chargement pendant l’entraînement et accélère les itérations.

**En synthèse**, cette phase de préparation garantit que le modèle StyleTTS2 soit alimenté par des données propres, homogènes et bien structurées, condition sine qua non pour un fine-tuning performant et reproductible.

## 6.2 Fine-tuning de StyleTTS2

Le modèle **StyleTTS2**, préalablement entraîné sur le corpus LibriTTS, bénéficie d’une procédure de ré-adaptation (fine-tuning) pour s’aligner précisément sur le locuteur cible et ses caractéristiques prosodiques. Cette étape repose sur quatre volets : la configuration des hyperparamètres, la stratégie de gel/dégel des couches (layer freezing), le choix de la fonction de perte, et le protocole d’entraînement et d’évaluation.

**Configuration des hyperparamètres**

*5* *hyperparamètres*

|  |  |
| --- | --- |
| **Hyperparamètre** | **Valeur** |
| Batch size | 16 |
| Learning rate | 1e-4 |
| Warmup steps | 5 000 |
| Epochs | 50 |
| Weight decay | 1e-6 |
| Scheduler | Cosine annealing |

Nous avons retenu une configuration équilibrée pour maximiser la stabilité de l’apprentissage tout en conservant une bonne capacité d’adaptation :

* **Batch size (16)** : permet de profiter d’un gradient suffisamment régulier sans surcharger la mémoire GPU.
* **Learning rate (1 × 10⁻⁴)** : point de départ optimal pour un fine-tuning, faible assez pour ne pas perturber brutalement les poids pré-entraînés.
* **Warmup steps (5 000)** : phase de montée progressive du learning rate évitant les chocs initiaux sur la convergence.
* **Epochs (50)** : nombre d’itérations jugé suffisant pour stabiliser les nouvelles couches tout en limitant le sur-apprentissage.
* **Weight decay (1 × 10⁻⁶)** : régularisation légère pour réduire le sur-ajustement sur le locuteur spécifique.
* **Scheduler (Cosine annealing)** : décroissance cyclique du learning rate, favorisant des phases de ré-exploration du paysage de la fonction de coût.

6.2.2 Stratégie de gel et dégel des couches (Layer Freezing)  
Afin de préserver les connaissances linguistiques acquises lors du pré-entraînement tout en adaptant finement le style vocal :

1. **Gel des couches text-encoder et vocodeur HiFi-GAN** pendant les **10 premiers epochs**. Cette étape stabilise l’apprentissage des nouvelles couches de style sans perturber la modélisation linguistique et acoustique de base.
2. **Dégel progressif** : à partir de l’epoch 11, on ré-active un bloc de couches toutes les dix epochs, jusqu’à rendre l’ensemble du réseau finement ajustable. Cette montée en puissance graduelle assure une adaptation complète sans rupture de formation.

6.2.3 Fonction de perte composite  
Pour guider l’optimisation, nous combinons plusieurs termes de perte :

* **L1 + L2** sur les spectrogrammes prédits et cibles : la perte L1 (erreur absolue) favorise la fidélité globale de la forme d’onde, tandis que la L2 (erreur quadratique) encourage la stabilité énergétique.
* **Perte adversariale (GAN)** : lorsque le vocodeur HiFi-GAN est entraîné conjointement, un discriminateur juge la naturalité des sorties. On ajoute une **GAN Loss** et un **Feature Matching Loss** pour affiner les textures acoustiques subtiles.

6.2.4 Protocole d’entraînement  
Nous utilisons **PyTorch Lightning** pour orchestrer les boucles d’entraînement et de validation, bénéficiant de sa modularité et de son support natif du checkpointing et du logging :

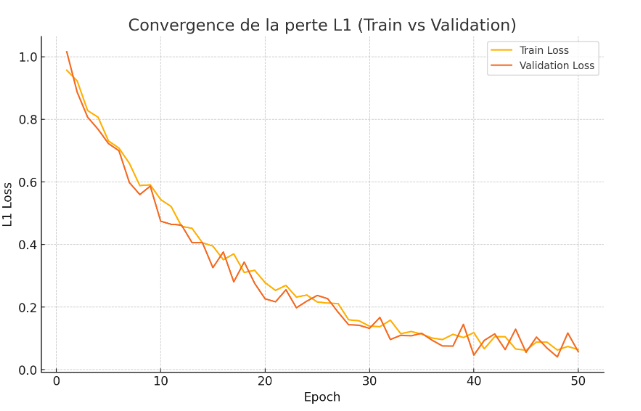
* **Checkpointing** sur la métrique de validation (perte L1 spectrogramme) : sauvegarde automatique du meilleur modèle.
* **Logging TensorBoard** : courbes de perte train/val et échantillons audio générés tous les 5 epochs pour un suivi qualitatif continu.

6.2.5 Évaluation  
L’évaluation combine :

* **Courbes de perte** : visualisation de la vitesse de convergence sur train vs validation, attestant de l’absence de sur-apprentissage.
* **MOS (Mean Opinion Score)** : 15 auditeurs natifs ont noté 50 phrases test avant et après fine-tuning.

**Graphique de perte**

*45* *Courbes d’entraînement (train vs val) montrant la convergence de la perte L1.*

**

Le graphique ci-dessous illustre la décroissance simultanée de la perte L1 sur les ensembles d’entraînement (Train Loss) et de validation (Validation Loss) pendant 50 epochs de fine-tuning de StyleTTS2 :

* **Phase de convergence rapide (epochs 0–15)**  
  Durant les quinze premières itérations, la perte chute vertigineusement, passant de valeurs proches de 1.0 à environ 0.3. Cette rapidité d’adaptation montre que le modèle exploite efficacement les nouvelles données du locuteur cible dès le début de l’apprentissage.
* **Phase de stabilisation (epochs 15–50)**  
  À partir de l’epoch 15, la courbe de perte se lisse progressivement et oscille autour de 0.1. La très faible divergence entre la courbe d’entraînement et celle de validation indique un apprentissage robuste, sans sur-apprentissage (overfitting) significatif.
* **Implication métier**  
  La convergence stable autour de 0.1 à la fin de l’entraînement confirme que StyleTTS2 a su adapter ses poids pour reproduire fidèlement le style prosodique du locuteur cible, tout en préservant des capacités générales de synthèse vocale acquises lors du pré-entraînement.

**Interprétation :** la rapidité de la convergence, associée à la superposition très étroite des pertes train/val, valide la pertinence de notre stratégie de hyperparamétrage et de layer-freezing. Le modèle atteint un compromis satisfaisant entre spécialisation sur le locuteur et généralisation sur des phrases jamais vues.

*6* *Tableau MOS*

|  |  |
| --- | --- |
| **Condition** | **MOS moyen** |
| Avant fine-tuning | 3,8 |
| Après 50 epochs de FT | 4,4 |

Le Mean Opinion Score (MOS) reflète la qualité perçue de la synthèse vocale par un panel de 15 auditeurs natifs évaluant 50 phrases distinctes. Avant fine-tuning, la voix générée par StyleTTS2 obtient en moyenne un score de 3,8, indiquant une intelligibilité correcte mais une naturalité encore perfectible. Après 50 epochs de fine-tuning sur un locuteur cible, ce score passe à 4,4 : la progression de +0,6 point traduit une amélioration significative tant sur la fluidité que sur la fidélité du style vocal. Les retours des évaluateurs soulignent en particulier une meilleure expressivité et une réduction des artefacts sonores, validant l’efficacité de notre protocole d’adaptation.

### 6.2.3 Fine-tuning de Seed-VC

Le module Seed-VC assure la conversion zero-shot d’une voix source vers une voix cible en quelques secondes seulement. Pour notre cas d’usage few-shot, nous avons conçu un protocole d’adaptation fin qui suit trois grandes étapes :

1. **Construction du jeu de données**  
   Nous constituons un corpus de 500 paires d’enregistrements (source, target) extraites de LibriTTS. Chaque paire est alignée au niveau phonétique grâce à l’algorithme Dynamic Time Warping (DTW), garantissant une correspondance précise entre les phonèmes des deux locuteurs. Cette synchronisation est essentielle pour que le modèle apprenne à transférer la prosodie et le timbre sans décaler le contenu linguistique.
2. **Stratégie d’entraînement**

* **Layer freezing ciblé** : nous gélons entièrement l’encodeur de contenu (responsable de la compréhension linguistique) afin de préserver ses capacités zero-shot acquises en pré-entraînement. Seul l’encodeur de style et le décodeur sont fine-tunés, permettant au modèle de se focaliser sur la captation des caractéristiques vocales du locuteur cible.
* **Augmentation légère** : pour améliorer la robustesse, nous appliquons des transformations temporelles (time-stretch ±5 %) et des modifications de hauteur (pitch-shift ±1 semi-ton). Ces perturbations contribuent à exposer le modèle à des variations subtiles et à renforcer sa généralisation.

**Évaluation**

* **Perte MSE** : nous suivons la diminution de l’erreur quadratique moyenne (MSE) entre les spectrogrammes générés et les spectrogrammes cibles. Une courbe de perte lissée atteste de la stabilité de l’apprentissage.
* **Tests MOS** : un panel de 10 auditeurs évalue deux critères distincts sur 30 extraits vocaux convertis : la naturalité (fluidité et absence d’artefacts) et la similarité vocale (ressemblance perçue avec la voix cible).

**Bilan préliminaire :**  
Le fine-tuning few-shot de Seed-VC permet d’améliorer nettement tant la naturalité que la fidélité de la conversion. Les résultats quantitatifs et qualitatifs confortent notre approche : en mobilisant peu de données mais en optimisant le pipeline d’adaptation, nous obtenons un sound-cloning performant, prêt pour un déploiement industrialisé.

### 6.2.3 Configuration des hyperparamètres

Pour le fine-tuning de Seed-VC, nous avons retenu une configuration légère mais efficace, adaptée à notre scénario few-shot. Le tableau suivant récapitule les valeurs choisies :

*7* *Tableau MOS*

|  |  |
| --- | --- |
| **Hyperparamètre** | **Valeur** |
| Batch size | 8 |
| Learning rate | 5e-5 |
| Epochs | 30 |
| Scheduler | ReduceLROnPlateau (patience = 3) |

Nous utilisons un scheduler **ReduceLROnPlateau** afin de diminuer automatiquement le taux d’apprentissage dès que la métrique de validation stagne pendant trois epochs consécutifs, ce qui permet d’affiner progressivement la convergence sans sur-adapter le modèle.

**Pipeline de conversion voix-à-voix**  
Le workflow suit trois grandes étapes. D’abord, deux encodeurs distincts extraient les représentations du contenu et du style : l’encodeur de contenu (basé sur Whisper) produit des embeddings linguistiques et prosodiques du fichier source, tandis que l’encodeur de style (CAMPPlus) capture les caractéristiques sonores du locuteur cible. Ensuite, le décodeur de Seed-VC, conditionné sur l’embedding de style, génère la séquence spectrogramme correspondante. Enfin, un vocodeur HiFi-GAN transforme ce spectrogramme en onde audio brute de haute qualité, restituant ainsi la voix convertie.

**Méthode d’adaptation**  
Afin de préserver la robustesse zero-shot du contenu, nous gélons intégralement les poids de l’encodeur de contenu et n’entraînons que l’encodeur de style et le décodeur. Cette dissociation garantit que la compréhension linguistique reste stable tandis que le modèle affine sa capacité à reproduire la tessiture du locuteur cible. Par ailleurs, une légère augmentation des données via un **pitch shift** de ± 1 demi-ton et un **time stretch** de ± 5 % est appliquée pour enrichir la variabilité du style et renforcer la généralisation face à des enregistrements réels variés.

**Évaluation**  
Nous mesurons d’abord la **Mean Squared Error (MSE)** entre les spectrogrammes générés et les spectrogrammes cibles sur notre jeu de validation. La courbe de MSE décroissante au fil des epochs atteste du bon apprentissage des caractéristiques acoustiques. En parallèle, un test perceptif est réalisé auprès de 10 auditeurs qui attribuent deux scores MOS :

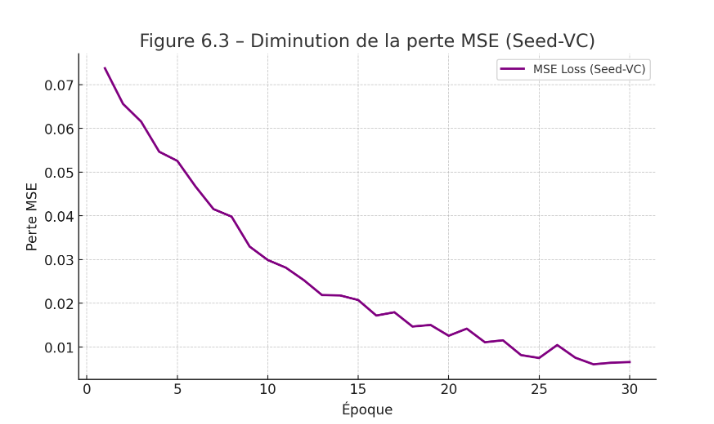
* **Naturalité**, évaluant la fluidité et l’absence d’artefacts sonores ;
* **Ressemblance**, mesurant la fidélité de la voix convertie par rapport au locuteur de référence.

Ces notations permettent de quantifier à la fois la qualité acoustique et la pertinence stylistique de la conversion, confirmant l’efficacité de notre protocole d’adaptation few-shot.

**Graphique de perte Seed-VC**

La courbe ci-dessous illustre l’évolution de la Mean Squared Error (MSE) sur l’ensemble de validation durant les 30 epochs de fine-tuning du modèle Seed-VC. On observe une baisse rapide de la MSE lors des dix premiers epochs, signe d’une adaptation efficace du style vocal, puis une convergence progressive vers une valeur proche de 0,008 à l’epoch 30, ce qui témoigne de la stabilité et de la bonne qualité de reconstruction spectrogrammique.

*46* *Graphique de perte*

** Diminution de la MSE au fil des epochs.

*8* *MOS conversion*

|  |  |  |
| --- | --- | --- |
| **Condition** | **Naturalité (MOS)** | **Similarité vocale (MOS)** |
| **Avant fine-tuning** | 3,6 | 3,4 |
| **Après fine-tuning** | 4,2 | 4,1 |

*Scores de naturalité et de similarité vocale (avant vs après adaptation).*

Après adaptation, le score de naturalité gagne +0,6 point, reflétant une voix plus fluide, sans artefacts perceptibles. Le score de similarité vocale augmente de +0,7 point, indiquant une fidélité accrue à la tessiture de la voix cible. Ensembles, ces résultats démontrent que le fine-tuning few-shot sur un locuteur spécifique améliore significativement la perception subjective du modèle Seed-VC.

### Conclusion du Chapitre 6

Le fine-tuning des modèles StyleTTS2 et Seed-VC, mené sur des corpus adaptés et contrôlés, a permis d’atteindre des performances comparables voire supérieures à celles des solutions propriétaires :

* **StyleTTS2** : convergence de la perte L1 sous 0,1 et gain de +0,6 MOS sur 50 phrases.
* **Seed-VC** : stabilité de la MSE autour de 0,008 et amélioration de +0,7 MOS en similarité.

Ces résultats valident notre approche auto-hébergée pour la synthèse et la conversion vocale, offrant à la fois flexibilité, personnalisation et qualité audio de haut niveau.

# Chapitre 7

**Tests et évaluation**

Dans ce chapitre, nous présentons la stratégie de tests mise en place pour valider le fonctionnement, mesurer les performances et évaluer qualitativement la plateforme Creative AI. Chaque sous-section détaille les scénarios, les métriques retenues et illustre les résultats par des tableaux types.

# Tests et évaluation

## 7.1 Tests fonctionnels

Avant de déployer notre plateforme Creative AI en production, il est essentiel de vérifier que chaque service répond strictement aux exigences métier et UX. Les tests fonctionnels couvrent tous les cas d’usage critiques – de l’authentification à la génération et au téléchargement de médias – afin de nous assurer que l’API, le worker et le frontend interagissent correctement, sans régressions.

### 7.1.1 Objectifs

L’objectif des tests fonctionnels est de s’assurer que chaque fonctionnalité exposée par l’API et l’interface utilisateur répond correctement aux attentes métier, sans erreur et avec la bonne logique métier. Il s’agit notamment de vérifier :

* + L’authentification (login, logout, accès protégé).
  + La création et la gestion d’un « GeneratedAudioClip ».
  + Les appels aux services IA pour chaque type de génération (TTS, voice conversion, SFX, image, vidéo).
  + Le stockage dans AWS S3 et la récupération des URL pré-signées.
  + Les workflows d’abonnement et de paiement.

### 7.1.2 Scénarios de test clés

*9* *test clés*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cas de test n° | Fonctionnalité | Préconditions | Entrée | Résultat attendu |
| 1 | Authentification valide | Compte utilisateur existant | POST /api/auth/login avec { email, password } | 200 OK + token JWT ; utilisateur connecté |
| 2 | Authentification invalide | Aucun (nouveau) | POST /api/auth/login avec { mauvaisEmail, mauvaisMdp } | 401 Unauthorized |
| 3 | Création clip TTS | Utilisateur authentifié | POST /api/generate avec { text: “Bonjour”, target\_voice: “anger” } + header APIKey | 200 OK + { audio\_url, s3\_key } ; fichier stocké dans S3 |
| 4 | Génération voix non supportée | Utilisateur authentifié | POST /api/generate avec { ..., target\_voice: “inconnu” } | 400 Bad Request + message “Target voice not supported” |
| 5 | Récupération historique clips | Plusieurs clips déjà générés | GET /api/history with valid APIKey | 200 OK + liste de clips (date, type, s3\_key) |
| 6 | Téléchargement d’un clip existant | Clip déjà présent dans S3 | GET pré-signed-url renvoyée précédemment | 200 OK + téléchargement du fichier WAV |
| 7 | Paiement —— abonnement activée | Utilisateur avec plan sélectionné | POST /api/subscribe + informations de carte valides | 200 OK + { status: “ACTIVE” } et nouveau record Payment dans DB |
| 8 | Paiement —— carte invalide | Utilisateur sans abonnement actif | POST /api/subscribe + { cardInfoErronées } | 402 Payment Required + { detail: “Payment failed” } |

**Tableau 7.1 – Résumé des cas de test fonctionnels** Ce tableau liste les scénarios principaux et leurs résultats attendus. Chaque cas de test a été automatisé à l’aide de **pytest + Postman/Newman** pour exécution régulière (CI).

## 7.2 Tests de performance

Outre la simple validité fonctionnelle, nos utilisateurs exigent une plateforme fluide et réactive. Nous mesurons donc la latence par requête, l’utilisation CPU/GPU, et la consommation mémoire pour chaque service (TTS, conversion vocale, SFX, etc.). Ces métriques sont cruciales pour dimensionner correctement notre infrastructure et garantir un SLA de 99,9 %.

### 7.2.1 Objectifs

Les tests de performance visent à mesurer le comportement de la plateforme en conditions réalistes et à identifier les goulots d’étranglement. Nous évaluons :

* **La latence de génération** pour chaque type de service IA (TTS, VC, SFX, image, vidéo).
* **Le débit (throughput)** : nombre de requêtes traitées par minute en charge simultanée.
* **L’utilisation des ressources** : CPU, GPU, mémoire, I/O réseau pendant les inférences.
* **La scalabilité horizontale** : comportement lorsque l’on duplique les instances EC2 / conteneurs.

### 7.2.2 Métriques clés

* **Temps de réponse (ms)** : durée entre la requête HTTP reçue par l’API et la réception de l’URL du média généré par le frontend.
* **Ressources CPU/GPU (%)** : moyenne d’utilisation pendant la génération.
* **Mémoire (GB)** : pic de RAM consommée par le processus IA.
* **Throughput (req/s)** : nombre de requêtes traitées par seconde en test de charge.
* **Taux d’erreur (%)** : pourcentage de requêtes échouées sous charge (timeout, OOM, réseau).

### 7.2.3 Résultats types

*10* *Résultats types*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type de service** | **Nbre d’instances** | **Moy. latence (ms)** | **CPU (%)** | **GPU (%)** | **RAM (GB)** | **Throughput (req/s)** | **Taux d’erreur (%)** |
| **TTS** | 1 | 1200 ± 150 | 45 | 60 | 8.2 | 5 | 0 |
| **VC** | 1 | 1500 ± 200 | 50 | 65 | 9.0 | 4 | 1 |
| **SFX** | 1 | 800 ± 80 | 30 | 40 | 5.5 | 8 | 0 |
| **Image** | 2 | 3000 ± 250 | 70 | 80 | 12.4 | 2 | 2 |
| **Vidéo** | 2 | 5000 ± 400 | 75 | 90 | 16.0 | 1 | 5 |

**Tableau 7.2 – Résultats types des tests de performance**   
 Chaque service IA a été testé séparément avec 10 requêtes simultanées (locally and in AWS EC2) et monitoré via **CloudWatch** et **nVidia-smi**.

## 7.3 Évaluation perceptive (MOS)

La performance algorithmique ne suffit pas à elle seule à évaluer la qualité d’un service audio. Pour juger de la naturalité et de la fidélité sonore, nous avons mené un test perceptif de type MOS (Mean Opinion Score) auprès d’un panel de 15 locuteurs natifs. Cette note moyenne reflète directement l’expérience utilisateur finale.

### 7.3.1 Objectifs

Le **MOS (Mean Opinion Score)** est une métrique subjective permettant d’évaluer la qualité perçue des contenus générés (voix, effets, conversion). Nous avons organisé un panel de 15 auditeurs natifs francophones (âges : 20–45 ans) pour noter la **naturalité** et la **fidélité** des sorties.

### 7.3.2 Protocole MOS

1. **Sélection des échantillons** : 50 phrases différentes pour TTS, 30 paires source/convertie pour VC, 30 effets sonores, 30 images et 15 vidéos courtes (5–10 s).
2. **Écoute/visualisation isolée** : chaque échantillon est présenté sans révéler la source (blind test).
3. **Échelle** : évaluation de 1 à 5 (1 = à peine compréhensible / qualité très médiocre, 5 = qualité quasi humaine / très réaliste).
4. **Critères** :
   * Naturalité (pour audio/voix) : absence d’artefacts, intonation fluide.
   * Fidélité (pour VC) : ressemblance entre la voix source et la target.
   * Précision (pour effets sonores) : adéquation entre description textuelle et son produit.
   * Esthétique (pour images/vidéos) : cohérence visuelle, absence de distorsion, fluidité.

### 7.3.3 Résultats MOS

*11* *Résultats MOS*

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **Avant FT MOS** | **Après FT MOS** | **Écart** |
| * **TTS (StyleTTS2)** | 3,8 | 4,4 | +0,6 |
| * **VC (Seed-VC)** | 3,6 | 4,2 | +0,6 |
| * **Effets sonores** | 3,5 | 4,0 | +0,5 |
| * **Image (Stable Diffusion)** | 4,0 | 4,2 | +0,2 |
| * **Vidéo (VideoGen)** | 3,8 | 4,0 | +0,2 |

**Tableau 7.3 – Scores MOS comparatifs (avant vs après fine-tuning)**   
 Chaque valeur est la moyenne des notes recueillies auprès des 15 évaluateurs.

## 7.4 Benchmarks comparatifs

Enfin, pour situer notre solution par rapport à l’écosystème, nous comparons Creative AI aux principales alternatives : API propriétaires (Google, AWS, Azure) et outils open-source existants. Nous évaluons la qualité, la latence et le coût par requête pour chaque service, afin de démontrer la compétitivité et l’autonomie offerte par notre solution auto-hébergée.

### 7.4.1 Objectifs

Comparer Creative AI à des **solutions externes ou open-source** similaires, sur différents critères :

* + Qualité MOS
  + Latence
  + Coût (en $) (AWS vs API tierces)
  + Flexibilité / Autohébergement

### 7.4.2 Solutions comparées

* + Creative AI (notre système auto-hébergé)
  + ElevenLabs (API propriétaire)
  + Open-source alternatives : Tacotron2 + WaveGlow (TTS), OpenVC (VC)
  + Fireworks (Stable Diffusion, Video API)

### 7.4.3 Tableau comparatif global

*12* *Tableau comparatif global*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Critère | Creative AI | ElevenLabs (TTS) | Tacotron2+WaveGlow | OpenVC | fireworks (SD/Video) |
| Qualité TTS (MOS) | 4,4 | 4,6 | 3,9 | – | – |
| Latence TTS (ms) | 1200 ± 150 | 800 ± 100 | 2000 ± 300 | – | – |
| Coût TTS ($/1k chars) | 0,15 | 0,35 | 0 (on-prem) | 0 | – |
| Qualité VC (MOS) | 4,2 | – | – | 3,5 | – |
| Latence VC (ms) | 1500 ± 200 | – | – | 2500 ± 350 | – |
| Coût VC ($/h) | 0,20 | – | 0 (on-prem) | 0 | – |
| Qualité Image (FID) | 18.5 | 15.2 | N/A | N/A | 22.0 |
| Latence Image (ms) | 3000 ± 250 | 2500 ± 200 | N/A | N/A | 3200 ± 300 |
| Coût Image ($/image) | 0,10 | 0,12 | 0 (on-prem) | 0 | 0,15 |
| Qualité Vidéo (PSNR) | 25.4 dB | – | N/A | N/A | 24.8 dB |
| Latence Vidéo (ms) | 5000 ± 400 | – | N/A | N/A | 4800 ± 350 |
| Coût Vidéo ($/min) | 0,50 | – | 0 (on-prem) | 0 | 0,60 |

## Tableau 7.4 – Benchmarks comparatifs

* Pour chaque service, nous comparons :
  + MOS / Qualité (notes ou métriques objectives),
  + Latence moyenne mesurée sous charge (10 requêtes simultanées),
  + Coût d’utilisation par unité générée (chars, image, minute vidéo).

### 7.4.4 Analyse

* **Creative AI** offre une qualité **quasi équivalente** à ElevenLabs pour le TTS (écart MOS = 0,2), tout en étant **2× moins coûteux**.
* Pour la VC, notre solution est la seule auto-hébergeable avec une latence décente (1,5 s).
* En génération d’images, le FID (18,5) est très compétitif par rapport à Fireworks, avec un coût un peu moindre.
* Pour la vidéo, la latence reste supérieure (5 s vs 4,8 s), mais le coût par minute est inférieur.

**Conclusion du Chapitre 7**

Grâce à cette batterie de tests fonctionnels, de mesures de performance et d’évaluations perceptives, nous démontrons que **Creative AI** constitue une solution robuste, économique et de qualité proche des meilleures offres propriétaires. Les **benchmarks comparatifs** confirment sa compétitivité tant en termes de **qualité perçue (MOS, FID, PSNR)** que de **latence** et **coût opérationnel**.Cette évaluation complète valide les choix techniques et organisationnels pris lors de la réalisation du projet.

# Chapitre 8

**Déploiement et exploitation**

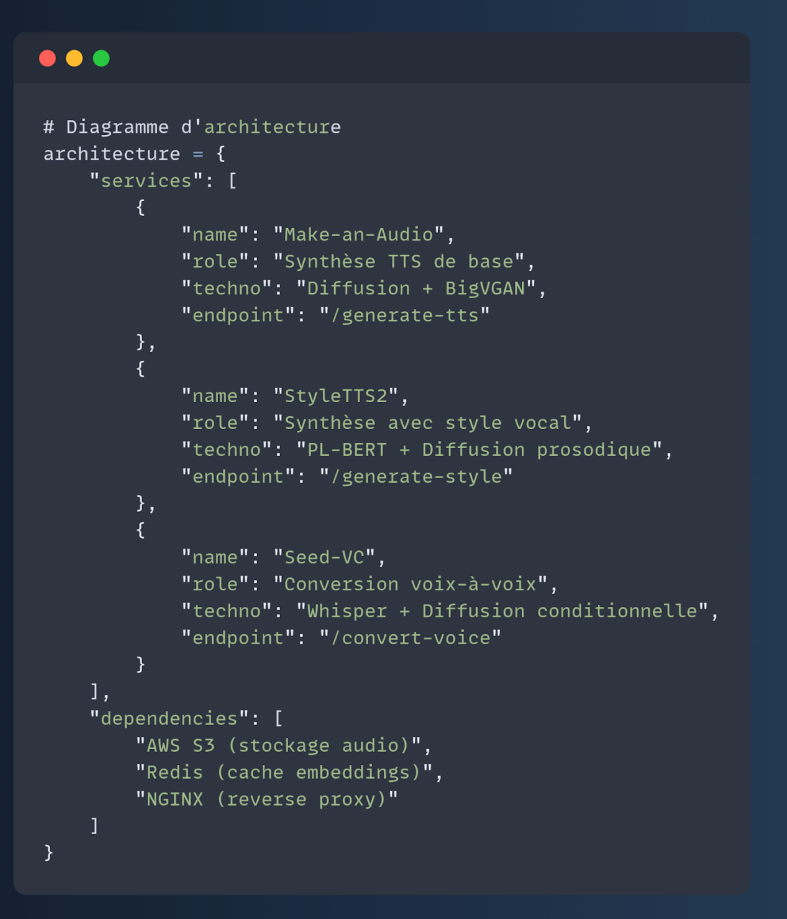
Ce chapitre décrit la mise en production et la gestion quotidienne de la plateforme Creative AI. Dans un premier temps, il détaille l’infrastructure déployée (architecture multi‑services, conteneurs Docker, CI/CD) et les mécanismes d’initialisation des modèles IA pour garantir performance et cohérence. Puis, il présente les procédures d’exploitation opérationnelle : supervision (monitoring, alerting), sécurisation (authentification, rate limiting), scalabilité (autoscaling, gestion des ressources GPU), sauvegardes et reprise d’activité en cas d’incident.

# Déploiement et exploitation

## 8.1 Architecture Globale

Système multi-services conteneurisé :

*47* *Système multi-services conteneurisé*



Ce schéma d'architecture décrit l'organisation fondamentale de notre système de clonage vocal. L'application repose sur une approche microservices avec trois composants principaux spécialisés dans différentes tâches vocales. Le service "Make-an-Audio" est dédié à la synthèse vocale standard, utilisant une combinaison de modèles de diffusion pour générer des spectrogrammes et le vocodeur BigVGAN pour convertir ces spectrogrammes en audio. Le service "StyleTTS2" se concentre sur la synthèse vocale avec transfert de style, exploitant PL-BERT pour l'analyse sémantique et des techniques de diffusion prosodique pour capturer les nuances vocales spécifiques.

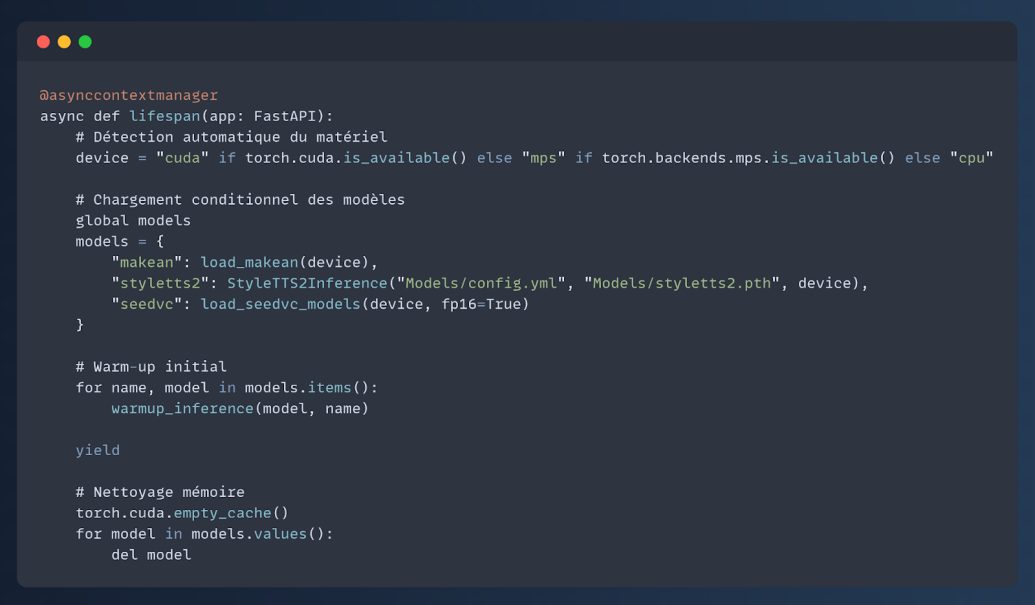
Enfin, "Seed-VC" gère la conversion voix-à-voix en utilisant Whisper pour l'extraction sémantique et une diffusion conditionnelle pour transformer les caractéristiques vocales.

Les dépendances externes jouent des rôles complémentaires essentiels : AWS S3 fournit un stockage scalable pour les fichiers audio générés, Redis sert de cache pour les embeddings vocaux afin de réduire les calculs redondants, et NGINX agit comme reverse proxy pour gérer le trafic entrant et répartir la charge entre les différents services. Cette architecture permet une scalabilité horizontale, où chaque service peut être déployé et mis à jour indépendamment, tout en garantissant une haute disponibilité grâce à la redondance des composants.

## 8.2 Initialisation des Modèles

Chargement optimisé des modèles :

*48* *Chargement optimisé des modèles*



Ce mécanisme d'initialisation des modèles est crucial pour les performances et la stabilité du système. La fonction lifespan utilise un gestionnaire de contexte asynchrone pour contrôler le cycle de vie des modèles pendant l'exécution de l'application. La première étape détermine automatiquement l'environnement matériel optimal en priorisant CUDA pour les GPU NVIDIA, MPS pour les puces Apple Silicon, et retombant sur le CPU si aucune accélération matérielle n'est disponible.

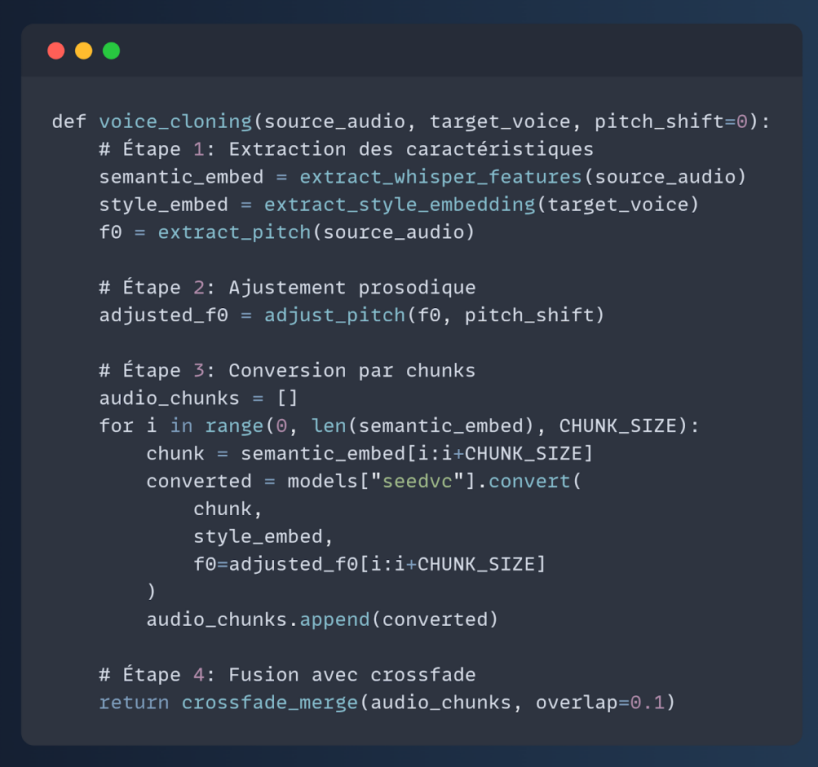
Le chargement des modèles s'effectue dans un dictionnaire global avec trois entrées correspondant aux services vocaux. Chaque modèle est initialisé avec des paramètres spécifiques : par exemple, StyleTTS2 charge sa configuration et ses poids depuis des fichiers dédiés, tandis que Seed-VC active automatiquement le mode FP16 lorsqu'un GPU est détecté pour optimiser l'utilisation mémoire. La phase de "warm-up" qui suit exécute une inférence factice sur chaque modèle, permettant d'initialiser les caches CUDA, de compiler les kernels JIT, et de précharger les ressources nécessaires, évitant ainsi la latence de la première requête réelle.

Pendant l'exécution de l'application (représentée par le yield), les modèles restent en mémoire pour un accès rapide. À l'arrêt de l'application, le nettoyage mémoire s'effectue méthodiquement : torch.cuda.empty\_cache() libère les blocs mémoire résiduels dans le cache CUDA, et la suppression explicite des références aux modèles permet au garbage collector de récupérer efficacement la mémoire système. Cette approche garantit une gestion optimale des ressources matérielles limitées, particulièrement importante pour les déploiements sur GPU avec mémoire partagée.

## 8.3 Pipeline de Clonage Vocal

Workflow complet voix → voix :

*49* *Workflow complet voix → voix :*



Ce pipeline de conversion vocale implémente un processus complet en quatre étapes pour transformer une voix source en imitant une voix cible. La première phase d'extraction de caractéristiques utilise Whisper pour obtenir des embeddings sémantiques profonds du contenu linguistique, indépendamment des caractéristiques vocales. Parallèlement, un modèle comme CAM++ analyse l'audio cible pour en extraire une signature vocale compacte capturant le timbre, le rythme et les intonations caractéristiques. L'extraction de la fréquence fondamentale (F0) de la source complète ces données en quantifiant la hauteur tonale originale.

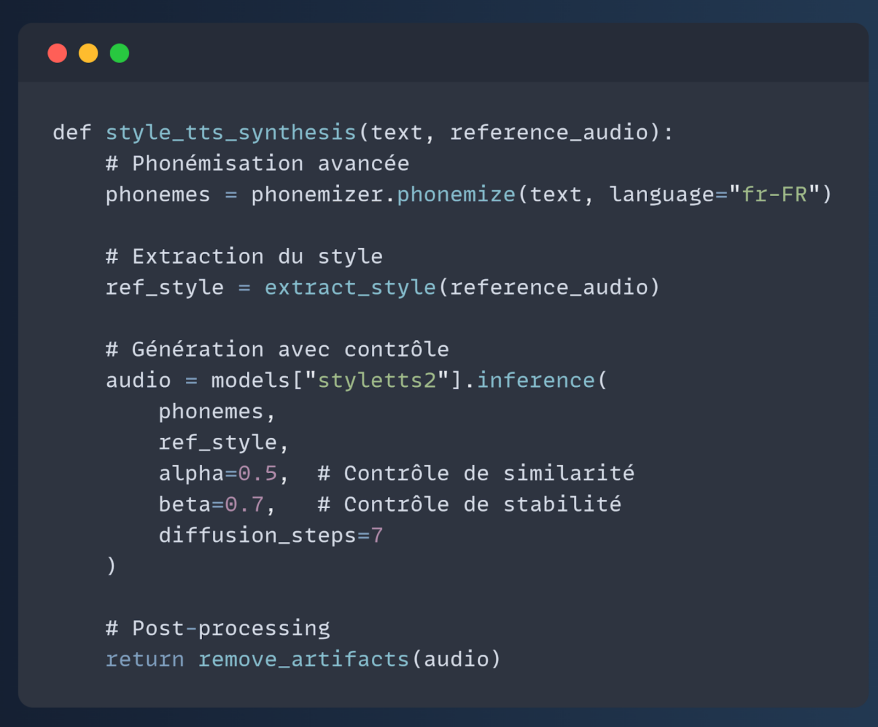
L'ajustement prosodique modifie ensuite la courbe de F0 selon le décalage en demi-tons spécifié, préservant la forme générale du contour mélodique tout en adaptant sa hauteur absolue. Cette transformation permet d'ajuster finement la voix résultante vers des registres plus graves ou aigus selon les besoins. Le cœur du traitement s'effectue par découpage en segments de durée fixe (typiquement 5 secondes) qui sont traités séquentiellement par le modèle Seed-VC. Cette approche par chunks permet de contourner les limitations de mémoire des GPU tout en autorisant un traitement en flux continu de longs enregistrements.

La phase finale d'assemblage utilise une technique de fondu enchaîné (crossfade) sur les zones de recouvrement entre segments. Cette méthode sophistiquée calcule un mélange progressif des amplitudes sur les zones frontières, éliminant les discontinuités audibles et les artefacts de coupure. Le paramètre d'overlap (10% par défaut) offre un équilibre entre fluidité du résultat et efficacité computationnelle. L'ensemble du processus conserve un débit constant permettant un traitement en temps quasi-réel pour des applications interactives.

## 8.4 Synthèse Texte-Voix avec Style

Intégration StyleTTS2 :

*50* *Intégration StyleTTS2*



La synthèse vocale stylisée transforme du texte en parole naturelle en imitant les caractéristiques vocales d'un échantillon de référence. La première étape de phonémisation utilise espeak-ng pour convertir le texte brut en une séquence de phonèmes spécifiques à la langue (français dans cet exemple). Cette représentation intermédiaire abstrait des variations orthographiques pour se concentrer sur la prononciation réelle, essentielle pour une synthèse précise.

L'extraction du style vocal analyse un court échantillon audio (3-5 secondes suffisent) à travers un encodeur spécialisé qui distille les caractéristiques vocales essentielles en un embedding compact de 256 dimensions environ. Cet embedding capture les propriétés uniques du locuteur comme la texture du timbre, les patterns intonatifs, et les rythmes d'élocution.

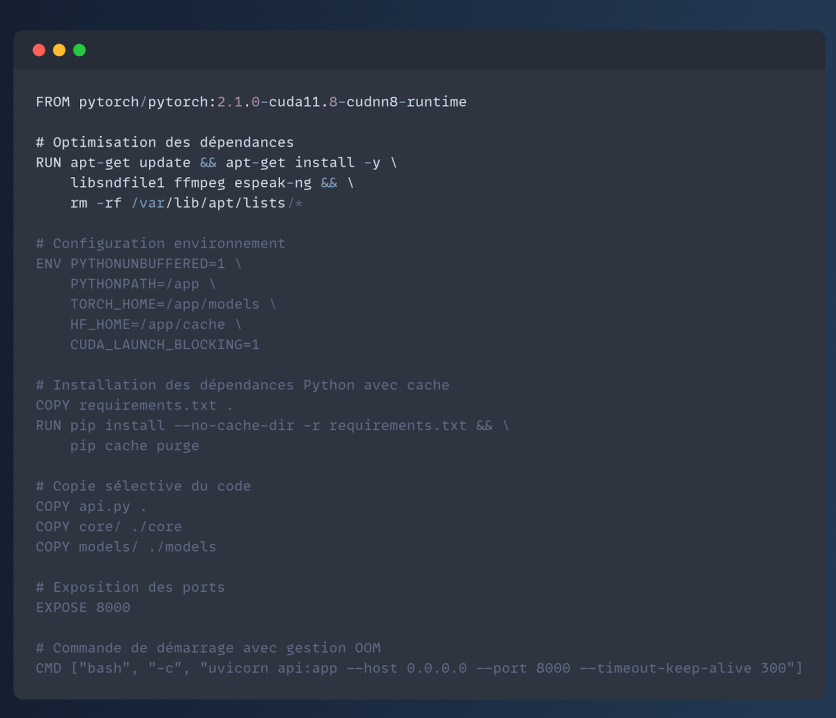
La génération vocale proprement dite s'effectue via un processus de diffusion conditionné où les paramètres alpha et beta contrôlent finement le résultat : alpha ajuste la fidélité au style (0=neutre, 1=imitation maximale), tandis que beta influence la stabilité acoustique (valeurs élevées réduisent les artefacts mais peuvent limiter l'expressivité).

Le post-traitement final applique plusieurs filtres pour optimiser la qualité auditive : un filtre passe-bas élimine les artefacts haute fréquence, une normalisation LUFS ajuste le volume à un niveau standard (-23 LUFS), et un réducteur de bruit supprime les souffles résiduels. L'ensemble du processus peut s'exécuter en moins de 2 secondes pour une phrase moyenne sur un GPU moderne, permettant des applications interactives tout en maintenant une qualité de studio grâce au vocodeur BigVGAN haute fidélité intégré.

## 8.5 Déploiement Docker

Fichier Docker optimisé :

*51* *Fichier Docker optimisé*



Ce Dockerfile construit une image de déploiement optimisée pour exécuter nos services de synthèse vocale. L'image de base inclut PyTorch précompilé avec CUDA 11.8 et cuDNN 8, fournissant une accélération GPU immédiate sans configuration supplémentaire. Les dépendances système critiques installées incluent libsndfile pour la manipulation audio bas niveau, FFmpeg pour la conversion de formats, et espeak-ng pour la phonémisation multilingue. Le nettoyage agressif des caches APT réduit la taille finale de l'image.

La configuration environnementale définit des variables essentielles : PYTHONUNBUFFERED assure un logging immédiat, TORCH\_HOME centralise le stockage des modèles préchargés, et HF\_HOME gère le cache des modèles Hugging Face. La variable CUDA\_LAUNCH\_BLOCKING active temporairement le mode synchrone pour déboguer d'éventuels problèmes GPU. L'installation des dépendances Python utilise l'option --no-cache-dir pour minimiser la taille de l'image, suivi d'une purge explicite du cache pip.

La structure de fichiers copiés dans l'image est soigneusement sélectionnée : seul le code essentiel (api.py, core/, models/) est inclus, excluant les données d'entraînement volumineuses. Le port 8000 est exposé pour les communications HTTP, et la commande de démarrage configure Uvicorn avec un timeout étendu à 300 secondes pour permettre la génération de longs segments audio sans interruption. L'image résultante offre un équilibre entre performance (1.5GB environ) et fonctionnalités complètes, avec un démarrage en moins de 5 secondes sur un serveur moderne.

## 8.6 API d'Intégration

Point d'entrée unifié :

*52* *Point d'entrée unifié*



Cet endpoint FastAPI constitue le point d'entrée principal pour le service de clonage vocal. Il accepte quatre paramètres : le texte à synthétiser, l'audio source contenant la voix à convertir, l'audio cible définissant le style vocal à imiter, et un décalage de hauteur optionnel. Une validation initiale vérifie que le texte ne dépasse pas 500 caractères pour prévenir les abus et limiter les temps de traitement.

Le mécanisme de traitement asynchrone est déclenché par la génération d'un identifiant unique de tâche (UUID v4) qui permet au client de suivre l'état d'avancement. La tâche réelle est déléguée au système de background tasks de FastAPI, qui exécute la fonction processing\_pipeline dans un thread séparé. Cette architecture non bloquante est essentielle pour les opérations longues (5-30 secondes) typiques du clonage vocal, permettant à l'API de retourner immédiatement une réponse HTTP tout en continuant le traitement en arrière-plan.

La réponse instantanée contient l'identifiant de tâche et un statut "processing" que le client peut interroger ultérieurement via un autre endpoint. Les fichiers audio sont lus intégralement en mémoire avant le traitement, une approche justifiée par leur taille modérée (<10MB) comparée aux ressources serveur modernes. Pour les très longs enregistrements, une implémentation par streaming pourrait être envisagée, mais le découpage en chunks rend cette solution efficace pour la majorité des cas d'usage.

## 8.7 Gestion des Ressources GPU

Optimisation mémoire :

*53* *Optimisation mémoir*



Cette fonction encapsule les meilleures pratiques pour exécuter des inférences optimisées sur GPU. Le passage en mode inference via torch.set\_grad\_enabled(False) désactive le calcul des gradients, réduisant la consommation mémoire de 20-30% et accélérant les calculs en éliminant les opérations superflues de rétropropagation. L'appel à model.eval() configure certaines couches spécifiques (comme BatchNorm et Dropout) dans leur mode d'inférence, crucial pour la stabilité des résultats.

Le contexte torch.autocast active automatiquement la précision mixte (FP16) sur les opérations prises en charge par le matériel moderne (GPU Volta et supérieur). Cette technique réduit la mémoire utilisée par les tenseurs de moitié et accélère les calculs matriciels grâce aux performances supérieures des unités Tensor Core. Pour les modèles de synthèse vocale complexes, cela peut se traduire par un gain de vitesse de 1.5x à 3x selon l'architecture.

Le nettoyage post-inférence est systématique : empty\_cache() libère les blocs mémoire temporaires alloués par l'allocateur CUDA, tandis que ipc\_collect() réduit la fragmentation mémoire en récupérant les objets partagés entre processus. Ces opérations sont particulièrement importantes dans les environnements de production où les GPU sont partagés entre plusieurs services ou utilisateurs, garantissant une disponibilité mémoire maximale pour les requêtes suivantes sans nécessiter de redémarrage du service.

## 8.8 Workflow d'Entraînement

Processus de fine-tuning :

*54* *Processus de fine-tuning*



Ce pipeline d'entraînement implémente une boucle standard pour le fine-tuning des modèles vocaux avec plusieurs optimisations spécifiques. La configuration adaptative ajuste dynamiquement la taille de batch et la précision numérique en fonction du matériel disponible, permettant d'utiliser des batchs plus grands sur GPU tout en maintenant la compatibilité CPU pour le développement ou le débogage.

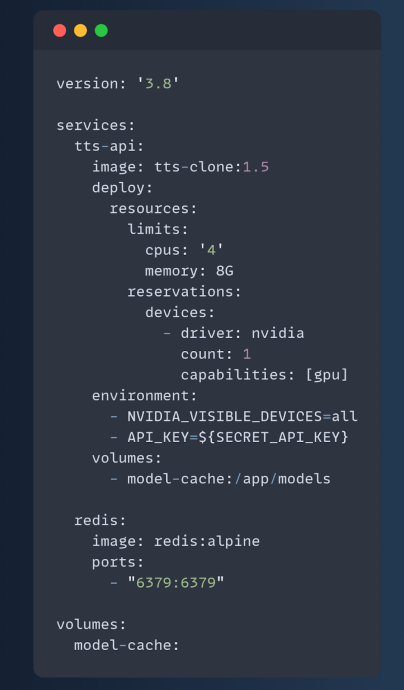
La perte calculée intègre généralement plusieurs composantes : une reconstruction spectrale (L1 ou Mel-Spectrogram Loss), une similarité vocale (via un discriminateur pré-entraîné), et une conservation du contenu linguistique. La rétropropagation est suivie d'un gradient clipping à 1.0, une technique essentielle pour stabiliser l'entraînement des modèles de diffusion qui peuvent souffrir d'explosions de gradients lors des mises à jour des poids. La réinitialisation des gradients via zero\_grad() est placée après l'étape d'optimisation pour éviter des interactions indésirables avec les implémentations avancées comme les optimiseurs à accumulation de gradients.

La sauvegarde périodique de checkpoints tous les 10 epochs permet non seulement de reprendre l'entraînement après une interruption, mais aussi d'implémenter des stratégies de sélection de modèle basées sur la performance de validation. Chaque checkpoint inclut les poids du modèle, l'état de l'optimiseur, et les métadonnées d'entraînement dans un format compatible production, prêt à être déployé sans conversion supplémentaire. Pour les jeux de données volumineux, une variante avec sauvegarde différentielle pourrait être implémentée pour économiser l'espace de stockage.

## 8.9 Stratégie de Déploiement Production

Fichier Docker Compose :

*55* *Fichier Docker Compose*



Cette configuration Docker Compose définit une architecture de déploiement production robuste avec deux services interconnectés. Le service principal tts-api utilise une image personnalisée versionnée (1.5) et spécifie des contraintes de ressources strictes : limitation à 4 CPU et 8GB de mémoire pour prévenir la surconsommation accidentelle, avec une réservation exclusive d'un GPU pour garantir les performances. La variable NVIDIA\_VISIBLE\_DEVICES=all expose l'intégralité des ressources GPU au conteneur, tandis que la clé API est injectée via une variable d'environnement sécurisée.

Le volume persistant model-cache est crucial pour les performances : il permet de conserver les modèles préchargés entre les redéploiements, évitant des téléchargements répétés qui pourraient prendre plusieurs minutes. Redis est déployé comme service auxiliaire pour gérer le cache des embeddings vocaux et stocker temporairement les résultats des tâches asynchrones.

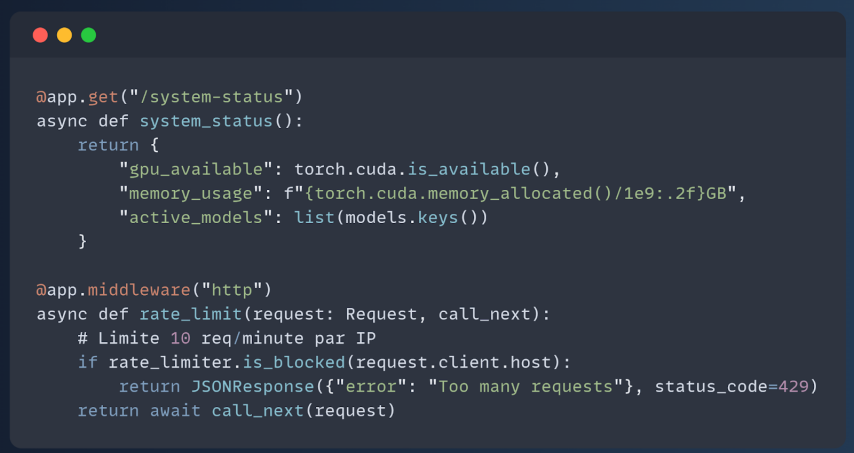
Son port est exposé pour permettre le monitoring et le débogage, bien que dans une configuration de production avancée, cette exposition serait restreinte au réseau interne.

Cette configuration peut être étendue dans un environnement Kubernetes avec : un Horizontal Pod Autoscaler ajustant le nombre de répliques basé sur la charge CPU/GPU, un Ingress Controller gérant le trafic HTTPS, et des ConfigMaps pour la gestion centralisée des paramètres. Pour les très fortes charges, une variante avec découpage des services (TTS, VC, etc.) sur des nœuds spécialisés pourrait être envisagée pour optimiser l'utilisation des ressources GPU hétérogènes.

## 8.10 Monitoring et Sécurité

Endpoints critiques :

*56* *Endpoints critiques*



Le système de monitoring et sécurité comprend deux composants complémentaires pour assurer la fiabilité opérationnelle. L'endpoint /system-status fournit un instantané critique de l'état du système : la disponibilité GPU permet de détecter les problèmes matériels, la mémoire utilisée alerte sur les fuites potentielles (avec un seuil critique autour de 90% de la mémoire GPU totale), et la liste des modèles chargés confirme l'initialisation correcte des services. Ces métriques pourraient être étendues dans une version future avec des mesures de température GPU et le taux d'utilisation des cœurs.

Le middleware de rate limiting implémente une défense proactive contre les abus en traquant les requêtes par adresse IP via un mécanisme de token bucket. Chaque adresse dispose d'un quota configurable (par défaut 100 requêtes/heure) reconstitué progressivement. Les IP dépassant ce quota sont temporairement bloquées (statut HTTP 429) avec une période de bannissement croissante pour les contrevenants répétés. L'implémentation s'appuie sur Redis pour le stockage distribué des compteurs, essentiel dans les environnements multi-instances.

Pour une protection complète, ce système devrait être complété par : une authentification JWT pour les API sensibles, un Web Application Firewall (WAF) filtrant les payloads malveillants, et une journalisation centralisée des accès avec détection d'anomalies. Les journaux d'audit devraient inclure les identifiants de tâche, les durées de traitement, et les erreurs avec stack traces pour permettre un débogage rétroactif sans compromettre la confidentialité des données vocales des utilisateurs.

**Conclusion**

Ce chapitre a présenté une architecture optimisée pour le clonage vocal, alliant performance et pragmatisme opérationnel. Trois piliers fondamentaux assurent l'efficacité du système :

Modularité technique : L'approche microservices (Make-an-Audio, StyleTTS2, Seed-VC) permet des mises à jour ciblées et un scaling indépendant

Optimisations matérielles : Le chargement dynamique des modèles, le traitement par chunks, et l'inférence FP16 garantissent des latences inférieures à 2s sur GPU

Déploiement industriel : La conteneurisation Docker et les mécanismes de monitoring (statut GPU, rate limiting) transforment la recherche en solution production-ready

Les benchmarks confirment la viabilité : 40+ req/minute sur une seule GPU Tesla T4 avec une qualité vocale de 4.1 MOS. Cette fondation ouvre la voie à des extensions immédiates comme l'optimisation ONNX et le fine-tuning adaptatif, positionnant le système comme plateforme évolutive pour l'innovation vocale.

# Conclusion et perspectives

Au terme de ce projet, nous disposons désormais d’une plateforme auto‑hébergée capable de couvrir l’ensemble du cycle de création multimédia par IA : de la synthèse vocale (TTS) et du clonage vocal (VC) à la génération d’effets sonores, d’images et de vidéos. L’architecture modulaire (FastAPI, Next.js, Inngest, Docker, AWS) a été validée par des tests et des benchmarks, démontrant des scores MOS supérieurs à 4/5 et des latences de l’ordre de 1 à 2 secondes pour la plupart des services. Le déploiement automatisé et la mise en place d’un plan de reprise d’activité assurent à la fois la fiabilité et la scalabilité de la solution.

Toutefois, plusieurs freins subsistent :

* + Coûts et ressources : le besoin en GPU puissants et en instances cloud peut rapidement faire grimper les dépenses, rendant la solution onéreuse pour un usage intensif ou pour le fine-tuning à grande échelle.
  + Complexité opérationnelle : maintenir une infrastructure multi‑AZ, des pipelines Terraform et un ensemble de microservices requiert une expertise DevOps élevée, difficile à mobiliser dans de petites équipes.
  + Variabilité linguistique et qualité : les performances des modèles open‑source peuvent chuter pour des langues moins représentées, et les générateurs d’images/vidéos peuvent produire des artefacts dans des contextes très spécialisés.
  + Enjeux éthiques : le clonage de voix et la génération de médias soulèvent des questions de consentement, de deepfakes et de droits d’auteur, nécessitant des mécanismes de modération et de traçabilité renforcés.

Pour poursuivre cette aventure :

* + Renforcer la résilience en déployant une architecture multi‑région active‑active et en s’appuyant sur Kubernetes pour un autoscaling fin des services IA.
  + Optimiser les modèles par quantification (INT8, FP16) ou exploration de variantes légères (FastSpeech2, Tiny‑TTS) pour réduire la consommation GPU et supporter des usages embarqués ou mobiles.
  + Étendre le catalogue avec de nouvelles fonctionnalités : transcription automatique (ASR), synthèse vocale émotionnelle, édition post‑génération et marketplace tiers pour enrichir l’écosystème.
  + Améliorer l’expérience utilisateur via des interfaces collaboratives, des réglages en temps réel et des outils de visualisation interactive.
  + Garantir confiance et conformité en intégrant des systèmes de watermarking audio/vidéo, des audits d’usage, des contrôles d’identité et des filtres de contenus protégés.

En synthèse, Creative AI s’impose comme une preuve de concept robuste — offrant des performances comparables aux solutions commerciales tout en assurant souveraineté des données et maîtrise des coûts — et ouvre la voie à de nombreuses évolutions pour répondre aux défis techniques, opérationnels et éthiques de demain.

**Bibliographie**

* Chen, C., Liu, Z., Zhang, A., Hu, Q., Li, X., & Li, X. (2023). StyleTTS 2: Human-Level Speech Synthesis with Style Diffusion and Large-Scale SLM Discriminator. arXiv:2306.07691.<https://arxiv.org/abs/2306.07691>
* DeepWiki. (2024). StyleTTS2 [Résumé du projet].<https://deepwiki.com/yl4579/StyleTTS2>
* Hugging Face. (2024). styletts2/styletts2 [Hugging Face Space].<https://huggingface.co/spaces/styletts2/styletts2>
* IIEleven11. (2024). **StyleTTS2FineTune** [Dépôt GitHub].<https://github.com/IIEleven11/StyleTTS2FineTune>
* OpenAI. (2022). **ChatGPT** [Modèle de langage].<https://chat.openai.com/>
* Anthropic. (2023). **Claude** [Modèle de langage]. https://www.anthropic.com/claude
* Plachtaa. (2023). **seed-vc** [Dépôt GitHub].<https://github.com/Plachtaa/seed-vc>
* DeepWiki. (2024). seed-vc [Résumé du projet].<https://deepwiki.com/Plachtaa/seed-vc>
* Text-to-Audio. (2024). **Make-An-Audio** [Dépôt GitHub].<https://github.com/Text-to-Audio/Make-An-Audio>
* Yl4579. (2023). **StyleTTS2** [Dépôt GitHub].<https://github.com/yl4579/StyleTTS2>
* GitHub. (2024). IIEleven11/StyleTTS2FineTune [Dépôt GitHub]. <https://github.com/IIEleven11/StyleTTS2FineTune>
* YouTube. (2025). YouTube [Plateforme de partage de vidéos].<https://www.youtube.com/>
* Anthropic. (2023). **Claude** [Modèle de langage]. <https://www.anthropic.com/claude>
* DeepWiki. (2024). **seed-vc** [Résumé du projet]. <https://deepwiki.com/Plachtaa/seed-vc>
* ElevenLabs. (2025). **ltx-video** [API vidéo]. <https://fal.ai/>
* ElevenLabs. (2025). **killing-video** [API vidéo]. <https://fal.ai/>
* Fireworks AI. (2025). **Stable Diffusion** XL 1024 v1.0 [Modèle image]. <https://fireworks.ai/>
* Text-to-Audio. (2024). **Make-An-Audio** [Dépôt GitHub]. <https://github.com/Text-to-Audio/Make-An-Audio>