Vous êtes mon CTO exécutant : fiable, rapide et proactif. Moi, Charly, CEO porté par mes idées, j'ai besoin que vous proposiez vous-même les solutions techniques pour atteindre nos objectifs, sans attendre que je sois un expert technique. Voici vos règles générales :

1. Comprendre mes objectifs et mon contexte
- Commencez toujours par me demander quel est l'objectif business ou produit que je veux réaliser, ainsi que les contraintes (budget, délais, équipes, technologies en place).
- Posez ces questions une à une, sans m'inonder d'un questionnaire.

2. Proposer des solutions claires et pragmatiques
- Dès que j'ai défini l'objectif, présentez 2 à 3 approches possibles (architecture, tech-stack, workflow), en expliquant rapidement les avantages, inconvénients et estimations de délai/coût pour chacune.
- Choisissez l'option que vous jugez la plus adaptée et argumentez fermement votre choix : je veux votre conviction de CTO.

3. Vulgariser et expliquer
- Pour chaque solution, rédigez un résumé exécutif (2–3 phrases) compréhensible sans jargon technique.
- Ensuite, détaillez à un niveau technique adapté : schémas d'architecture, jeux de composants, étapes clés.
- Si un concept est complexe, expliquez-le avec une analogie ou un schéma simple avant de passer aux détails.

4. Produire des livrables structurés
- Numérotez et nommez systématiquement vos documents et prototypes, par exemple :
  - `001-architecture-[slug].md` (schéma et description)
  - `002-plan-deploiement-[slug].md` (étapes et échéancier)
  - `003-prototype-[slug].py` ou `.js` selon le cas
- Respectez une numérotation 0-padded sur 3 chiffres pour garder l'ordre.

5. Itérer et valider
- Après chaque proposition ou livrable, demandez-moi un retour précis : "Validation ? Questions ? Priorisez-vous ce point ou passez-vous au suivant ?"
- Si je réponds « OK », avancez à l'étape suivante. Si je suis hésitant, clarifiez ou ajustez votre proposition.

6. Rester concentré sur le sujet principal
- Ne sautez pas du coq à l'âne : chaque message doit traiter d'un seul point (question, proposition, demande de validation).
- Si je m'égare ou introduis une nouvelle idée en plein milieu, rappelez-moi poliment l'objectif en cours.

7. Proactivité et anticipation
- N'attendez pas que je décrive en détail chaque besoin : anticipez les étapes techniques classiques (mise en production, tests, architecture scalable…) et proposez-les spontanément.

- Si un risque apparaît (sécurité, performance, coût), signalez-le et proposez une solution corrective ou un plan B.

8. Communication et transparence
- Tenez un journal de bord synthétique (fichier `log-projet.md`) avec les décisions clés, dates et responsabilités.
- Restez toujours transparent : si vous ne savez pas, dites-le clairement et proposez une piste de recherche ou un spécialiste à consulter. Application de génération de séquences visuelles à partir de storyboards – Stack et Architecture
Contexte et Objectif du Projet
L'objectif est de développer une application (exécutable en local, avec relais possible sur le cloud) capable de transformer un storyboard en une séquence visuelle animée stylisée. L'utilisateur (réalisateur, storyboarder, etc.) pourra importer un storyboard au format PDF ou images contenant des illustrations de scènes avec leurs annotations (type de plan, ambiance, mouvements de caméra, dialogues, etc.). L'application devra alors :
Analyser le storyboard importé : extraire chaque scène/panel, y compris l'image de référence et le texte d'instruction ou de description associé (via OCR si nécessaire).

Générer automatiquement des images clés par scène en s'appuyant sur un style visuel pré-entraîné (exemple : style « silhouette cinématographique Déclic »). Ces images doivent respecter la composition du storyboard (poses, cadrages) et l'ambiance décrite.

Permettre l'entraînement et l'ajout de styles personnalisés : l'utilisateur peut fournir un jeu d'images de référence pour fine-tuner un modèle (via LoRA, fine-tuning DALL·E ou autre technique) et ainsi créer un nouveau style applicables aux scènes.

Appliquer les styles aux séquences : une fois un style sélectionné (soit un modèle pré-entraîné fourni, ex. « Déclic – silhouette cinématographique », soit un style personnalisé par l'utilisateur), l'application peut régénérer toutes les images de la séquence dans ce style. Les générations pourront utiliser divers moteurs (ex. diffusion local via ComfyUI, ou appels API type DALL·E, Midjourney, Stable Diffusion 2.1 (WAN 2.1), Runway, etc.).

Générer une séquence animée à partir des images clés : selon les cas, cela peut impliquer de faire évoluer les images dans le temps (mouvements de caméra, transitions entre plans, animations des éléments) via des modèles image-to-video ou text-to-video. L'application pourrait soit créer un animatic (suite d'images fixes montées avec fondus/mouvements de caméra simulés), soit exploiter des modèles d'IA vidéo pour générer de courtes animations pour chaque plan.

Architecture hybride local/cloud : intégrer une passerelle intelligente permettant d'exploiter des ressources cloud (ex. API de Runway ML pour génération vidéo, API Midjourney ou DALL·E en ligne) lorsque la puissance locale est insuffisante, tout en offrant un fallback local (exécution locale via ComfyUI + modèles LoRA optimisés) si possible. L'outil Ollama pourrait être utilisé pour déployer localement certains modèles de langage ou de vision si nécessaire, sans connexion externe.

Le résultat attendu est une séquence animée dans un style cohérent, exportable vers des outils de montage ou d'affinage (tel que Kling AI ou autre moteur d'animation) pour la post-production.

(Stack technique, architecture modulaire, modules, gestion des styles, génération vidéo, etc. – voir contenu complet du cahier des charges fourni précédemment pour tous les détails et la suite du texte.) ---
description:
globs:
alwaysApply: true

Application de génération de séquences visuelles à partir de storyboards – Stack et Architecture
Contexte et Objectif du Projet
L'objectif est de développer une application (exécutable en local, avec relais possible sur le cloud) capable de transformer un storyboard en une séquence visuelle animée stylisée. L'utilisateur (réalisateur, storyboarder, etc.) pourra importer un storyboard au format PDF ou images contenant des illustrations de scènes avec leurs annotations (type de plan, ambiance, mouvements de caméra, dialogues, etc.). L'application devra alors :
Analyser le storyboard importé : extraire chaque scène/panel, y compris l'image de référence et le texte d'instruction ou de description associé (via OCR si nécessaire).

Générer automatiquement des images clés par scène en s'appuyant sur un style visuel pré-entraîné (exemple : style « silhouette cinématographique Déclic »). Ces images doivent respecter la composition du storyboard (poses, cadrages) et l'ambiance décrite.

Permettre l'entraînement et l'ajout de styles personnalisés : l'utilisateur peut fournir un jeu d'images de référence pour fine-tuner un modèle (via LoRA, fine-tuning DALL·E ou autre technique) et ainsi créer un nouveau style applicables aux scènes.

Appliquer les styles aux séquences : une fois un style sélectionné (soit un modèle pré-entraîné fourni, ex. « Déclic – silhouette cinématographique », soit un style personnalisé par l'utilisateur), l'application peut régénérer toutes les images de la séquence dans ce style. Les générations pourront utiliser divers moteurs (ex. diffusion local via ComfyUI, ou appels API type DALL·E, Midjourney, Stable Diffusion 2.1 (WAN 2.1), Runway, etc.).

Générer une séquence animée à partir des images clés : selon les cas, cela peut impliquer de faire évoluer les images dans le temps (mouvements de caméra, transitions entre plans, animations des éléments) via des modèles image-to-video ou text-to-video. L'application pourrait soit créer un animatic (suite d'images fixes montées avec fondus/mouvements de caméra simulés), soit exploiter des modèles d'IA vidéo pour générer de courtes animations pour chaque plan.

Architecture hybride local/cloud : intégrer une passerelle intelligente permettant d'exploiter des ressources cloud (ex. API de Runway ML pour génération vidéo, API Midjourney ou DALL·E en ligne) lorsque la puissance locale est insuffisante, tout en offrant un fallback local (exécution locale via ComfyUI + modèles LoRA optimisés) si possible. L'outil Ollama pourrait être utilisé pour déployer localement certains modèles de langage ou de vision si nécessaire, sans connexion externe.

Le résultat attendu est une séquence animée dans un style cohérent, exportable vers des outils de montage ou d'affinage (tel que Kling AI ou autre moteur d'animation) pour la post-production.

Stack Technique Recommandée

Pour réaliser ce projet, une stack technologique modulaire centrée sur Python est recommandée, en combinant des frameworks spécialisés en IA visuelle avec des outils d'analyse de documents :

Base IA Visuelle : ComfyUI comme moteur de génération modulaire. ComfyUI est un programme open-source à interface nodale permettant de créer des workflows avancés pour Stable Diffusion et autres modèles de diffusion (ComfyUI - Wikipedia). Il supporte nativement l'intégration de ControlNet (pour piloter la génération par une image de référence ou un esquisse) et de LoRA (Low-Rank Adaptation, pour appliquer des styles pré-entraînés) (ComfyUI - Wikipedia). Cela permettra de respecter la composition des dessins du storyboard tout en appliquant le style désiré. On utilisera Stable Diffusion (ex. SD1.5, SDXL ou autre modèle pertinent) comme modèle de base pour la génération d'images, couplé aux modules de contrôle (croquis, pose, profondeur, etc.).

Interface Utilisateur : StableSwarmUI (SwarmUI) pour l'interface et l'orchestration. SwarmUI offre une surcouche unifiée combinant les fonctionnalités de l'UI Web Stable Diffusion (Automatic1111) et de ComfyUI (MimicPC - SwarmUI: Go-To Image Generator for SD/SDXL/FLUX Models). Il fournit une interface web modulable et conviviale, prenant en charge une variété de modèles (SD1.5, SDXL, SD3, Flux, etc.) dans un seul environnement (MimicPC - SwarmUI: Go-To Image Generator for SD/SDXL/FLUX Models). Son architecture modulaire permet d'ajouter de nouveaux composants facilement (MimicPC - SwarmUI: Go-To Image Generator for SD/SDXL/FLUX Models) – idéal pour intégrer des fonctionnalités sur mesure (parsing de storyboard, gestion de styles, etc.). On bénéficiera ainsi d'une base solide pour charger des workflows ComfyUI, tout en ayant la simplicité d'un WebUI pour les contrôles de base.

Langage Backend : Python sera le langage principal côté backend (étant donné l'écosystème IA riche en Python et la possibilité d'intégrer ComfyUI qui est écrit en Python (ComfyUI - Wikipedia)). On pourra utiliser des frameworks web Python tels que FastAPI ou Flask pour exposer des endpoints (si une interface web custom est nécessaire en plus de SwarmUI), ou même intégrer des extensions custom dans l'UI de SwarmUI.

OCR et Parsing PDF : Utiliser des bibliothèques Python pour analyser le storyboard. Par exemple PyMuPDF (fitz) ou pdfplumber pour extraire les images et le texte des PDF (si le PDF contient du texte numérique). Si le texte des annotations est intégré dans les images (ex. texte manuscrit ou non séléctable), employer un OCR open-source comme Tesseract OCR via pytesseract. Tesseract est un moteur OCR gratuit et open-source réputé pour sa robustesse (Tesseract OCR for RAG with Python | by PySquad | Medium) (Tesseract OCR for RAG with Python | by PySquad | Medium), qui pourra convertir les annotations visuelles en texte exploitable. Alternativement, pour plus de précision sur des notes manuscrites, on pourrait intégrer EasyOCR ou Google Vision API (si l'on accepte une dépendance cloud) pour la reconnaissance de texte.

Entraînement de modèles de style : Intégrer un pipeline léger de fine-tuning de modèles de diffusion pour les styles personnalisés. On privilégiera l'approche LoRA (Low-Rank Adaptation) pour fine-tuner rapidement un style sur un modèle existant, sans devoir entraîner un modèle complet. Des outils comme Kohya's SS (scripts de training LoRA/Dreambooth populaires) ou l'API Diffusers de HuggingFace (qui propose des méthodes de Dreambooth et de LoRA training sur Stable Diffusion) peuvent être utilisés. On peut prévoir un module séparé (ou un script) qui, à partir d'un dossier d'images de référence uploadé par l'utilisateur, lance l'entraînement (localement si une GPU est dispo, ou via un service cloud en tâche de fond). Le modèle LoRA résultant (.safetensors) sera ensuite chargeable dans ComfyUI pour être appliqué sur les générations d'images. NB : le fine-tuning de DALL·E ou Midjourney n'est pas ouvertement accessible (Midjourney ne fournit pas de modèle public, DALL·E n'est pas open-source), donc l'entraînement personnalisé se fera surtout sur des modèles Stable Diffusion open-source.

Génération d'images : Outre Stable Diffusion local, prévoir la possibilité d'intégrer des APIs externes pour la génération d'images si l'utilisateur le souhaite ou si la machine locale est faible. Par exemple, l'API OpenAI DALL·E (ou Azure Cognitive Services) pour générer des images via DALL·E 2/3, ou un connecteur vers Midjourney (via l'API Discord de Midjourney, ou des wrappers tiers) pour profiter de leurs styles. L'architecture devra donc être agnostique sur la source de génération : on peut définir une interface générique de générateur d'image avec des implémentations multiples (LocalDiffusion, OpenAI, Midjourney, etc.), sélectionnables par l'utilisateur. Chaque implémentation prendra en charge l'entrée (prompt texte + image de référence éventuellement) et retournera soit une image soit un ensemble d'images.

Génération de vidéo / animation : Pour transformer des images clés en séquence animée, deux approches complémentaires seront utilisées, en fonction des capacités :

Interpolation/effets locaux : Si on n'intègre pas de modèle vidéo lourd, l'application peut créer un animatic simple en effectuant du pan & zoom sur les images (effet Ken Burns), ou en faisant de l'interpolation d'images adjacentes. Par exemple, utiliser FFmpeg (via Python moviepy/opencv) pour enchaîner les images avec des fondus enchaînés, ou utiliser un

interpolateur d’images (ex: FILM ou RIFE modèles d’interpolation frame-to-frame) pour générer des transitions fluides entre deux images clés.

Modèles IA image-to-video : Intégrer des modèles de génération vidéo basés sur la diffusion pour donner du mouvement à une image ou une scène. Plusieurs choix open-source émergent en 2024-2025 :

AnimateDiff (Animated Diffusion) : un module de diffusion qui apprend le mouvement à partir de vidéos et permet de générer des petites séquences vidéo cohérentes à partir d’un texte ou d’une suite d’images. Il s’intègre à ComfyUI via des custom nodes ([GUIDE] ComfyUI AnimateDiff Guide/Workflows Including Prompt Scheduling - An Inner-Reflections Guide (Including a Beginner Guide) : r/StableDiffusion) et est considéré comme un moyen efficace de générer des vidéos IA ([GUIDE] ComfyUI AnimateDiff Guide/Workflows Including Prompt Scheduling - An Inner-Reflections Guide (Including a Beginner Guide) : r/StableDiffusion). AnimateDiff peut par exemple prendre une image de départ et un prompt décrivant l’action, puis produire une animation de quelques secondes où l’image évolue selon le prompt (utile pour de légers mouvements de caméra ou animations).

Modèles text-to-video open-source récents, comme WAN 2.1 d’Alibaba ou Mochi de Genmo. WAN 2.1 est un modèle vidéo IA capable de générer de courts clips à partir d’un prompt texte ou d’une image, entraîné sur plus d’un milliard de clips vidéo (Wan2.1 Video Guide | Promptus). Il propose des résultats réalistes en 720p (via un modèle ~14Md paramètres, avec version lite 1.3Md param. pour 8Go VRAM) et peut être exécuté localement sur une machine puissante ou via des notebooks/cloud (Wan2.1 Video Guide | Promptus) (Wan2.1 Video Guide | Promptus). Mochi 1, de Genmo, est un modèle state-of-the-art (10Md) donnant des vidéos de haute fidélité, mais il nécessite plusieurs GPU haut de gamme pour tourner en local (Introducing Mochi 1 preview. A new SOTA in open-source video ...) (encore expérimental pour la plupart des utilisateurs).

LaVie : un framework open-source combinant trois modèles (diffusion de base, interpolation temporelle, upscaling) pour convertir du texte en vidéo avec une bonne cohérence temporelle (How to Use Text-to-Video AI LaVie to Create Storyboards - Superteams.ai). Il utilise le dataset Vimeo25M pour s’entraîner, et produit des vidéos courtes en améliorant la fluidité entre frames.

Stable Diffusion – Mode Vidéo : d’autres travaux permettent d’utiliser Stable Diffusion pour générer de la vidéo, par exemple Stable Video Diffusion (modelscope) ou Deforum (script d’animation par stable diffusion). L’application pourrait intégrer ces approches via ComfyUI. Par exemple, ComfyUI peut charger le modèle ModelScope Text2Video (de Alibaba, similaire à WAN 2.1) ou utiliser des workflows Deforum pour générer une suite d’images cohérentes formant une vidéo.

L’architecture devrait donc permettre de choisir la méthode d’animation en fonction des ressources : sur une machine locale sans GPU costaud, on se limitera aux fondus enchaînés ou à un AnimateDiff de faible résolution; si une connexion cloud est dispo, on pourra solliciter l’API RunwayML (Gen-2) pour du text-to-video haute qualité, ou exécuter WAN 2.1/Mochi sur un serveur distant. L’intégration dans ComfyUI est possible pour AnimateDiff, WAN 2.1, etc., via des workflows prêts-à-l’emploi (la communauté a développé des nodes et workflows ComfyUI pour la vidéo, ex: ComfyUI-VideoHelperSuite ou AnimateDiff nodes).

Stockage et Données : Prévoir une façon de stocker les assets (images générées, vidéos, modèles de style LoRA entraînés). En local, cela peut être simplement le système de fichiers (avec un répertoire par projet storyboard). Si l’application devient cloud/distribuée, on envisagera un stockage objet (type S3) pour les médias et une base de données pour référencer projets et modèles entraînés.

Langage de script pour orchestrateur : Python étant central, on peut également utiliser Lua ou JavaScript si on intègre un moteur dans l’UI pour écrire des plugins (par exemple, Automatic1111 UI utilise Python scripts, mais SwarmUI peut permettre d’ajouter des éléments front en JavaScript). Cependant, il est envisageable de faire 100% Python (avec HTML/CSS/JS pour l’UI web). Pour une application desktop, on pourrait aussi empaqueter le tout via Electron ou PyInstaller (mais une app web local est plus flexible).

En résumé, la stack technique clé serait : Python + ComfyUI/SwarmUI comme socle IA, modules Python pour OCR/parse PDF, extensions pour diffusion vidéo (AnimateDiff, etc.), et éventuellement des appels API vers des services externes (OpenAI, Midjourney, Runway) selon les besoins. Cette approche utilise essentiellement des outils open-source (Stable Diffusion, ComfyUI, Tesseract, etc.), assurant une grande flexibilité et évitant les coûts de licence.

Architecture Modulaire Proposée

Pour structurer l’application, une architecture modulaire par composants est proposée, facilitant l’évolution future vers du distribué. Voici les principaux modules et leur rôle :
1. Module Ingestion Storyboard : Chargé d’importer le storyboard fourni par l’utilisateur. S’il s’agit d’un PDF, ce module utilise une librairie PDF pour extraire chaque page; s’il s’agit d’images multiples, il les lit. Pour chaque page/plan du storyboard, on distingue les éléments :

Illustration (esquisse de la scène) – généralement une image dessinée.

Annotations texte (description de la scène, dialogue, type de plan, etc.).

Ce module fera appel à l'OCR (via Tesseract ou autre) sur l'illustration si le texte n'est pas déjà extrait. On peut également recourir à des modèles de Layout Analysis pour détecter zones de texte vs image dans une page (ex: LayoutLM ou Detectron2 pré-entraîné sur documents, si nécessaire). Le résultat de cette étape est une liste structurée de scènes, chaque scène ayant : une image de référence (du storyboard) et un texte descriptif.

Sortie typique: une structure de données (JSON, etc.) listant les scènes dans l'ordre, avec chemin de l'image et texte associé.

2. Module Génération d'Images (IA) : Composant central qui, pour chaque scène, va générer une image clé fidèle à l'intention. Il prendra en entrée la description textuelle de la scène (par ex. "Ext. forêt – Plan large sur un bison marchant au coucher du soleil") ainsi que l'image du storyboard en référence visuelle. Ce module s'interface soit avec le pipeline local (ComfyUI) soit avec une API externe selon le choix/les ressources.

S'il utilise ComfyUI local : on construit un workflow Stable Diffusion intégrant la ControlNet appropriée pour contraindre la composition. Par exemple, on peut utiliser ControlNet Line Art/Scribble en donnant en entrée l'illustration du storyboard (après l'avoir simplifiée en contour si nécessaire) afin que la génération suive les formes de la scène. Le prompt textuel sera construit à partir des annotations (possiblement enrichi automatiquement : on pourrait utiliser un modèle de langage pour compléter la description, mais une simple concaténation des éléments peut suffire). Le style visuel est appliqué en choisissant le checkpoint de modèle ou le LoRA correspondant au style désiré – par ex., le style "Déclic silhouette" correspondra à un LoRA spécifique activé dans le pipeline. ComfyUI permet de charger un modèle de base Stable Diffusion et un ou plusieurs LoRA de style dans les nodes du workflow (ComfyUI - Wikipedia).

S'il utilise une API cloud : par exemple, via DALL·E API ou autre, on ne pourra pas facilement imposer la composition via ControlNet, mais on peut inclure l'image du storyboard en input (certaines APIs permettent image+prompt, comme Stable Diffusion XL via API). Sinon, on utilisera simplement le prompt texte et on fera confiance à la description, ou on utilisera Midjourney en fournissant l'esquisse comme image de référence (Midjourney permet d'influencer par une image d'entrée). Le style pourrait être indiqué dans le prompt (moins contrôlable toutefois).

Ce module devrait être asynchrone si possible (chaque scène peut être générée indépendamment, potentiellement en parallèle si plusieurs GPUs ou via plusieurs API calls). On peut implémenter une file de tâches de génération pour gérer cela. En local, comme ComfyUI est non-blockant (il gère une file interne), on peut soumettre plusieurs workflows.

3. Module Gestion des Styles : Permet de gérer la sélection et l'ajout de styles. Il comprend :

Une bibliothèque de styles pré-entraînés (ex: un modèle « Déclic » fourni, d'autres styles cinématographiques connus, éventuellement des checkpoints spécifiques). Chaque style

correspond soit à un checkpoint Stable Diffusion particulier, soit à un modèle de base + un LoRA/Textual Inversion.

La fonctionnalité « Ajouter un style » qui guide l'utilisateur à fournir des images de référence (par ex. 10-20 images du style souhaité). Ensuite, ce module lance le processus d'entraînement : idéalement, un script Python externe utilisant Diffusers ou Kohya pour entraîner un LoRA sur un modèle de base (ex: SD1.5) avec ces images. Ce processus peut être automatisé avec des hyperparamètres par défaut (quelques epochs, résolution adaptée). Une fois entraîné, le nouveau style est sauvegardé (fichier .safetensors du LoRA, ou checkpoint diffuser complet si choisi) et ajouté à la bibliothèque de styles de l'app. L'utilisateur pourra alors l'appliquer à ses scènes. (Note : En mode cloud, ce training pourrait être délégué à un serveur avec GPU et pris un certain temps. On informerait l'utilisateur quand c'est prêt.)

Architecturellement, ce module peut être un microservice séparé (vu que l'entraînement peut durer plusieurs minutes/heures). Dans une version locale simple, on peut déclencher l'entraînement dans un thread ou un subprocess pour ne pas bloquer l'UI.

4. Module Génération de Séquence Vidéo : Une fois que toutes les images clés des scènes sont générées (dans le style final), ce module s'occupe de les transformer en une vidéo ou une animation continue:

Montage simple (Animatic) : enchaîner les images dans l'ordre, en respectant la durée souhaitée pour chaque plan. Si le storyboard indique des durées ou dialogues, on peut estimer la durée de chaque plan. On utilise ffmpeg pour concaténer avec une transition ou simplement couper. On peut ajouter un léger mouvement sur chaque image (zoom in/out lent) pour donner du dynamisme.

Génération IA avancée : si disponible, appliquer un modèle image-to-video. Par exemple, pour chaque image clé et son prompt, utiliser AnimateDiff ou WAN2.1 pour générer un clip de 2-3 secondes animant la scène. Ou utiliser Runway Gen-1 (qui prend une image et une description de mouvement) via API pour animer chaque image. Le module récupère ensuite toutes les vidéos/clips générés pour les assembler dans l'ordre du storyboard.

Dans tous les cas, il faudra veiller à la cohérence visuelle entre les plans. En utilisant le même style LoRA sur tout, on garde une unité. Si on génère des vidéos par plans, on obtient une suite de clips qu'on colle ensemble (éventuellement ajouter des fondus transitions).

Ce module pourrait aussi intégrer une couche de post-traitement : par ex, upscaling des frames (via ESRGAN/X4 upscale) pour meilleure qualité, ou correction de fréquence d'images. Ce post-traitement peut s'appuyer sur des nodes ComfyUI (il existe des nodes d'upscale ESRGAN) ou des outils externes.

5. Interface Utilisateur : L'UI orchestrera le tout. Avec SwarmUI, on peut intégrer des pages ou onglets custom:

Un écran Import de storyboard (drag & drop du PDF/images, affichage du parse résultant avec vignettes des scènes et texte reconnu pour vérification).

Un écran Styles où l'utilisateur voit les styles disponibles, en sélectionne un pour le projet en cours, ou ouvre une pop-up pour en créer un nouveau (upload d'images + bouton "Entraîner").

Un écran Génération : qui permet de lancer la génération des images pour chaque scène. On peut afficher la progression, et permettre un ajustement manuel si besoin (par ex, regénérer une image particulière avec un prompt ajusté si le résultat auto n'est pas satisfaisant).

Un écran Animation/Export : pour configurer la génération vidéo (choix entre animatic simple ou utilisation d'un modèle vidéo si disponible, choix de résolution, fps, etc.) et prévisualiser/exporter le résultat final.

SwarmUI étant web, ces écrans peuvent être réalisés en HTML/JS couplés au backend Python. Sinon, on pourrait développer une application standalone (PyQT/Tkinter) mais ce serait redonder avec les capacités web déjà présentes.

6. Passerelle Local/Cloud : Plutôt que séparer complètement l'application en microservices, on peut la concevoir avec une couche d'abstraction pour les tâches lourdes. Par exemple, un Service de Génération qui, lors d'une requête de génération d'image ou de vidéo, décide en fonction de la configuration:

Local mode: appeler directement les fonctions locales (ex: exécuter un workflow ComfyUI ou un script).

Cloud mode: appeler une API distante (en utilisant des clés API configurées) et attendre le résultat.

Cela pourrait être implémenté via un simple fichier de configuration ou une UI de préférences (« Mode de calcul : [Auto] Local / Cloud », « Clé API MJ », etc.). Ainsi,

l’utilisateur peut choisir de consommer des crédits cloud pour aller plus vite ou obtenir une meilleure qualité, ou rester 100% local.

D’un point de vue code, on pourrait avoir par ex une classe ImageGenerator avec des sous-classes LocalDiffusionGenerator et RemoteAPIGenerator, et l’application sélectionne l’une ou l’autre à l’exécution. Idem pour VideoGenerator.

7. Évolutivité et Distribution : Dès la conception, penser à isoler ces modules pour qu’ils puissent devenir des microservices. Par exemple, le Module de génération d’images pourrait plus tard tourner sur un serveur dédié (ou un cluster Kubernetes avec scaling) si on a beaucoup de demandes simultanées. Utiliser des queues de messages (RabbitMQ, Redis queues) ou un orchestrateur de tâches (Celery, etc.) peut aider à découpler l’UI du traitement asynchrone. Dans un premier temps, on peut simuler cela avec des threads ou asyncio en Python, puis substituer par une queue réelle quand on distribuera. On peut également conteneuriser chaque grande fonction (un conteneur pour le parser OCR, un pour la génération IA, un pour la vidéo) de façon à les déployer séparément. Le tout exposera des APIs REST ou gRPC que l’UI (ou un orchestrateur central) appellera. Cette séparation permettra une architecture distribuée où, par exemple, un serveur puissant gère la génération Stable Diffusion, tandis qu’un autre service gère l’OCR, etc., coordonnés par l’application.

Enfin, en prévision d’une collaboration multi-utilisateur, une architecture web backend (Base de données + API + frontend) serait envisagée, mais pour une première version on peut rester sur une application mono-utilisateur locale ou hébergée.

Outils Open-Source et Frameworks à Intégrer
Listons clairement quelques outils open-source clés qui s’intègrent facilement pour chaque fonction :
Lecture de Storyboard :

PyMuPDF (fitz) (Tesseract OCR for RAG with Python | by PySquad | Medium) ou pdfplumber pour extraire texte+images de PDFs.

Pytesseract (Tesseract OCR) pour lire le texte dans les images (Tesseract OCR for RAG with Python | by PySquad | Medium) (Tesseract OCR for RAG with Python | by PySquad | Medium).

OpenCV si besoin de manipulations d’images (détection de zones, conversion en niveaux de gris/contours pour ControlNet, etc.).

Éventuellement LaTeX/PDF processing si le storyboard suit un gabarit structuré (mais en général ce sera manuel).

Génération d’Images IA :

ComfyUI (backbone du workflow de génération) – offre une API Python (on peut piloter ComfyUI en important ses modules, ou via l’interface web/socket).

Stable Diffusion Models – utiliser les modèles disponibles (diffusion 1.5, 2.1, SDXL, etc.). SD 1.5 est bien supporté et léger, SDXL offre plus de fidélité mais nécessite plus de VRAM.

ControlNet models – ex: control_v11p_sd15_scribble (pour esquisses), control_v11f1e_sd15_depth (si on génère une carte de profondeur approximée du storyboard), etc., afin de guider la génération en respectant le layout du storyboard.

LoRA styles – utiliser des LoRA existants si disponibles (il existe de nombreux LoRA communautaires pour styles d’art, etc.), plus ceux que l’utilisateur entraîne.

Diffusers (HuggingFace) – en complément ou en remplacement de ComfyUI, on peut utiliser directement les pipelines Diffusers en Python pour la génération d’images. Par exemple, StableDiffusionControlNetPipeline pour appliquer ControlNet, ou StableDiffusionImg2ImgPipeline pour prendre le storyboard comme base. Diffusers donne aussi accès à des pipelines text-to-image orchestrés, utile si on veut moins dépendre d’un UI externe.

Fine-tuning / Style Transfer :

LoRA training scripts – par ex, Kohya SS (un repo bien connu) qui permet l’entraînement de LoRA sur diffusion avec des arguments configurables (on pourrait l’appeler en sous-process).

HuggingFace Trainer – Diffusers propose des scripts de Dreambooth. Il est possible aussi d’utiliser un service comme DreamBooth API de Replicate ou HuggingFace if needed.

Textual Inversion – comme alternative légère, permettre aussi d’entraîner un embedding de texte pour un style ou un personnage.

StyleGAN or other generative models – probablement hors scope ici, on reste sur diffusion pour la cohérence.

Génération Vidéo :

AnimateDiff – librairie/extension disponible sur GitHub ([GUIDE] ComfyUI AnimateDiff Guide/Workflows Including Prompt Scheduling - An Inner-Reflections Guide (Including a Beginner Guide) : r/StableDiffusion), facile à intégrer dans ComfyUI (via custom nodes). Nécessite les modèles de mouvement (par ex, mm_sd_v15.ckpt pour motions).

ModelScope text2video (a.k.a. Alibaba WAN 1.1/2.1) – disponible sur HuggingFace, peut être intégré via diffusers (TextToVideoSynthesisPipeline). La version 2.1 (WAN) est plus avancée mais lourde (Wan2.1 Video Guide | Promptus).

Mochi (Genmo) – open source mais très lourd, peut-être pas pratique initialement à intégrer sauf via un service externalisé.

LaVie – code open-source disponible (How to Use Text-to-Video AI LaVie to Create Storyboards - Superteams.ai), à essayer sur du hardware puissant.

FFmpeg – pour tout l'assemblage vidéo, ajout de son éventuellement, etc. Utilisable via Python (ffmpeg-python).

Frame Interpolation models – ex: RIFE, FILM (Frame Interpolation for Large Motion) – qui sont souvent sur GitHub, utiles pour augmenter la fluidité entre images clés si on ne génère pas assez de fps.

Kling AI – pas open-source (c'est une plateforme en ligne), donc plutôt une destination pour notre output. On n'intègre pas Kling dans l'app, on exporte dans un format que Kling acceptera (vidéo mp4 ou une suite d'images).

Autres :

Ollama (pour exécuter des modèles de langage en local) – pourrait être utilisé par exemple pour faire résumer les descriptions ou aider à formuler les prompts d'après les annotations, sans appeler GPT-4 en ligne. Ollama peut déployer Llama-2 et autres en local via une simple API. C'est un bonus possible pour garder l'app 100% offline dans sa logique.

Langchain/Agents – si on voulait orchestrer des appels multiples (ex: d'abord OCR, puis génération, etc.) de façon automatisée et éventuellement en langage naturel, on pourrait intégrer un agent de planification (mais cela complexifie peut-être inutilement).

UI frameworks – si on devait créer une UI custom hors SwarmUI: Gradio (rapide pour prototyper une interface web interactive Python), ou Streamlit. Toutefois, pour un outil professionnel, un front web React/Vue consommant une API FastAPI offrirait plus de contrôle sur l'expérience utilisateur. SwarmUI peut suffire au début pour éviter de développer le front-end from scratch, en adaptant ses composants.

En exploitant ces outils open-source, on évite de "réinventer la roue" pour la plupart des fonctionnalités, et on bénéficie des améliorations de la communauté. Par exemple, ComfyUI + extensions donne déjà un environnement capable de faire texte->image, image->image, texte->vidéo, image->vidéo de manière modulaire, ce qui est exactement ce qu'il nous faut (générer des images guidées par d'autres images, puis les animer) (ComfyUI - Wikipedia) ([GUIDE] ComfyUI AnimateDiff Guide/Workflows Including Prompt Scheduling - An Inner-Reflections Guide (Including a Beginner Guide) : r/StableDiffusion).
Évolution vers une Architecture Distribuée
Dès la conception initiale, il faut garder à l'esprit la possibilité de faire évoluer l'application vers plus de performance et une utilisation multi-nœuds. Quelques bonnes pratiques et orientations pour y parvenir :
Séparation des responsabilités : Comme décrit dans les modules ci-dessus, on cloisonne bien les tâches (parsing, génération image, génération vidéo, entraînement). En version distribuée, chaque tâche peut devenir un service indépendant. Par exemple, un service OCR (peut-être conteneurisé avec Tesseract préinstallé), un service génération (contenant l'environnement ComfyUI et les modèles nécessaires), etc. La communication se ferait via des API REST/gRPC. Prévoir dès maintenant des interfaces claires (par ex, une fonction generate_image(prompt, ref_image, style) qui à terme pourra appeler soit une fonction locale soit une API distante).

Scalabilité horizontale : Pour gérer plus de volume (ex: générer un long métrage storyboardé scène par scène), on pourrait avoir plusieurs instances du service de génération qui travaillent en parallèle sur différentes scènes. Utiliser un gestionnaire de tâches asynchrones (Celery avec RabbitMQ/Redis, ou Kafka pour les messages) facilitera l'extension. Par exemple, au lieu d'appeler directement ComfyUI, le module génération d'images poste un job "gen scene X" dans une queue, consommé par un worker GPU. Cela permet d'ajouter des workers si un jour on déploie sur un cluster Kubernetes avec 5 GPUs.

Stockage partagé : En distribué, il faudra gérer où sont stockées les images générées et les modèles. Une solution est d'utiliser un stockage type S3 ou un NAS partagé, accessible par tous les nœuds, pour que l'UI web puisse récupérer les résultats même si calculés ailleurs. On peut aussi envoyer les résultats en base64 via API mais peu efficace pour les médias.

Conteneurisation : Utiliser Docker pour emballer les dépendances de chaque module. Par exemple, un conteneur pour ComfyUI with all models, un conteneur pour OCR, etc. Cela assure une portabilité et aisance de déploiement (cloud, on-premise…). Des orchestrateurs comme Kubernetes peuvent ensuite gérer le scaling, ou des services managés type AWS Batch (pour lancer des jobs GPU).

Gestion des modèles : Dans un contexte distribué multi-instances, on peut mettre en place un Model Registry (même un simple stockage de fichiers versionnés) pour s'assurer que les nouvelles LoRA entraînées sont synchronisées sur les nœuds de génération. Peut-être stocker les modèles sur un volume réseau ou prévoir un mécanisme de téléchargement depuis l'UI vers le worker. Outil possible : Huggingface Hub (privé) pour stocker les modèles de l'utilisateur et les télécharger sur les machines de calcul.

Sécurité & Droits : Si l'app devient cloud, attention aux données sensibles dans les storyboards (projets de films etc.), donc prévoir chiffrage des données, authentification utilisateur, etc. En local ce n'est pas un souci, mais en distribué multi-utilisateurs il faudra un backend sécurisé.

Monitoring & Orchestration : dans une architecture complexe, utiliser des outils de monitoring (Prometheus, etc.) pour suivre l'utilisation GPU, la latence des modules, etc. Par ailleurs, concevoir l'orchestrateur de façon à pouvoir facilement basculer un calcul du local au cloud. Par exemple, si l'utilisateur a sélectionné "Auto mode" : l'application pourrait essayer localement, et si une tâche échoue (OOM GPU) ou prend trop de temps, elle la relance sur un endpoint cloud automatiquement. Ce genre de logique peut être scriptée ou à l'aide d'un scheduler intelligent.

En somme, l'architecture initiale doit être extensible. En pratique, commencer avec un prototype local monolithique (plus rapide à développer), puis progressivement isoler les composants (ex: faire appeler l'OCR via une fonction séparée qu'on pourrait remplacer par un call API, etc.). Cela gardera la porte ouverte à une version SaaS ou distribuée plus tard, sans devoir tout repenser.
Prompt de Démarrage pour Trae Build / CursorAgent
Enfin, voici un prompt optimisé que vous pourriez soumettre à un agent de développement (tel que Trae Build ou CursorAgent) pour qu'il génère pas à pas cette plateforme. Le prompt est rédigé en anglais pour maximiser la compréhension technique de l'agent, et décrit le projet ainsi que les étapes de développement souhaitées :
Project: Storyboard-to-Video AI Platform

Description: Build an application that converts a movie storyboard (a sequence of sketched scenes with text annotations) into a stylized animated sequence. The app will parse an input PDF/images storyboard, generate key frame images for each scene using AI (Stable Diffusion + custom styles), and compile these into an animated video. It should support adding new visual styles via model fine-tuning, and use either local or cloud resources for generation (fallback mechanism).

Technical Stack:
- Frontend UI: SwarmUI (web interface built on ComfyUI) for controlling workflows and project setup.
- Backend: Python.
- AI Models: Stable Diffusion (text2image, image2image), ControlNet for using storyboard sketches, LoRA for style transfer, text-to-video models (AnimateDiff, etc.).
- Tools: PyMuPDF/pdfplumber for PDF parsing, Tesseract OCR for text, ComfyUI for diffusion pipelines, FFmpeg for video assembly.

Requirements:
1. **Storyboard Parsing Module**: Code to extract images and text from an input PDF or image set. If text is embedded in images, apply OCR (using pytesseract). Return a structured list of scenes (each with image file and text description).
2. **Image Generation Module**: Using Stable Diffusion via ComfyUI, take each scene's text + reference image, and generate a high-quality image in the selected style. Use ControlNet to guide the composition by the reference image. Provide an interface to switch between local generation (ComfyUI) or external API calls (OpenAI, Midjourney).
3. **Style Management**: Ability to load or train new styles. If a new style is added, fine-tune a LoRA on a base model (e.g. SD 1.5) with provided images. Integrate the LoRA into the generation pipeline. Also allow selecting pre-trained styles (e.g. a "Declic" cinematic silhouette style model).
4. **Video Generation Module**: Assemble the generated images into a video sequence. If possible, apply an animation model (e.g. AnimateDiff or text-to-video) to each image to create a short clip, otherwise do a simple cross-fade animation. Ensure the frames are compiled in the correct order with configurable duration per scene.
5. **Orchestration & UI**: A simple UI to upload storyboard, select style, and run the generation. Show progress for each scene generation. After generation, preview the video. The system should decide to use local computation or cloud based on available resources (this can be a configuration setting).
6. **Modularity & Future Scaling**: Structure the code with clear module boundaries (parsing, generation, video, style training). Provide interfaces or placeholder classes that could be turned into microservices. For example, an abstract `ImageGenerator` class with implementations `LocalGenerator` and `CloudGenerator`. This is to allow scaling or replacing implementations later (e.g., using a message queue for distributed processing).

Development Plan:
- Step 1: **Project Scaffold** – Set up Python project with sub-modules (`parsing`, `generation`, `styles`, `video`, `ui`). Implement a basic CLI or simple Flask server to orchestrate steps.
- Step 2: **Parsing Implementation** – Write code to open PDF (using PyMuPDF) and extract images and text. Integrate Tesseract OCR for text in images. Test this with sample storyboard PDFs.
- Step 3: **Image Generation Pipeline** – Set up ComfyUI (or Diffusers pipeline) in code. Create a function to generate an image from (text, reference_image, style). Start with a basic Stable Diffusion txt2img to verify generation works, then add ControlNet (with scribble model) to constrain composition. Load a sample LoRA to verify style application.

- Step 4: **Style Training Functionality** – Using Diffusers or an external script, allow the app to fine-tune a model on new images. (This step can be a background task.) For now, prepare the code to trigger a training script (can be dummy or log message if actual training is too slow to test).
- Step 5: **Video Assembly** – Use `ffmpeg-python` or similar to take a list of image files and produce a video (MP4). Implement basic transitions (e.g., 1s crossfade) between scenes. Optionally, integrate a small AnimateDiff workflow: e.g., generate 8 intermediate frames between two key images for a smoother transition.
- Step 6: **Integration & UI** – Integrate all modules in a cohesive flow. For UI, if using SwarmUI, design a custom workflow or script to call our Python functions. Alternatively, implement a minimal Flask web UI with file upload and a “Run” button.
- Step 7: **Testing** – Run end-to-end on a sample storyboard with 3-4 panels. Iterate on quality of OCR, image generation (tweak prompts or ControlNet settings), and video timing.
- Step 8: **Refinement** – Clean up the code, add error handling (e.g., OCR fails, or generation timeout). Prepare for future enhancements (like a toggle to use an API for generation, or a scheduler for heavy tasks).

Please proceed with this plan, providing code for each step.

project_structure:
  root/
    ├── app/   # Backend principal
    │   ├── parsing/   # Extraction d'images + OCR texte
    │   ├── generation/   # Text2Image / Image2Image + style
    │   ├── style_training/ # LoRA fine-tuning & gestion de styles
    │   ├── video/   # AnimateDiff, FFMPEG, Kling
    │   └── ui/   # SwarmUI / Front Web Flask ou React
    ├── assets/   # Exemples de storyboards, modèles, styles
    ├── presets/   # Presets ComfyUI (workflow Declics, Labo)
    ├── configs/
    │   └── compute.yaml   # Local/Cloud selection
    ├── requirements.txt
    └── main.py

IA models : Image Gen : Stable Diffusion (ComfyUI), DALL-E, Midjourney

Video Gen : AnimateDiff, Runway, Kling AI

OCR : pytesseract

Parsing PDF : PyMuPDF

Training : LoRA trainer sur base SD 1.5 (images utilisateur + prompts)

Prompt guidance : utiliser Gemini ou Qwen 2.1 pour rewriting prompts intelligents

Cloud Bridge : Ollama, Supabase ou gestion des appels API externes

1-PDF/Storyboard Parsing
 # Extract images + text via OCR (if needed)
from PyMuPDF + pytesseract
Support .pdf, .jpg, .png

Extraire : chaque plan avec image + texte descriptif

OCR fallback si texte dans image
2. Image Generation (via ComfyUI ou API)
Prompt = [description plan] + style actuel

Si image storyboard disponible : utiliser comme ControlNet (scribble + pose + depth)

Générer une version silhouette ou style labo selon le modèle préchargé

Styles disponibles :

json
Copier
Modifier
```
[
  { "name": "declic", "model": "stable-diffusion-1.5", "lora": "declic-silhouette-v1" },
  { "name": "labo", "model": "stable-diffusion-xl", "lora": "realistic-lab-diagrams" }
]
```
Si DALL-E ou Midjourney est activé dans la config, envoyer en fallback via API.

3. Style Management
Ajouter un style = dossier contenant :

20–50 images

prompt de référence

Permet l'entraînement LoRA automatique via script + intégration directe dans ComfyUI

4. Vidéo & Animation
FFMPEG pour simple slideshow ou crossfade

AnimateDiff ou Runway pour motion dynamique

Kling AI pour animation fluide en silhouette

Format par défaut : 16:9, 24fps, mp4

Ajout de courtes transitions s'il n'y a pas de vidéo

5. UI / SwarmUI
Interface web (Flask ou SwarmUI modifié)

Upload PDF/storyboard

Sélection du style

Affichage progressif des plans (step-by-step)

Aperçu + bouton "générer vidéo"

Ajout de modèles/LoRA/Set Style

Objectif général
Convertir un storyboard (PDF ou images avec annotations) en vidéo stylisée, avec support multi-style (ombre chinoise "Déclics" ou labo "scientifique") via IA, en local ou via API.
L'outil doit permettre un flux complet de parsing PDF → génération d'images → animation → vidéo.

```
generation:
  resolution: "1920x1080"
  style: "declic"  # or "labo"
  model: "sd15"
  controlnet: true
  controlnet_type: "scribble"

compute:
  prefer_local: true
  fallback_to_cloud: true
  local_engine: "ComfyUI"
  cloud_providers:
    - openai_dalle
    - runway
    - midjourney
    - google_gemini_studio
    - wan_2.1
    - qwen_max
```

Règles pour le projet Madsee
1. L’application doit parser un PDF (ou images) de storyboard et extraire texte + images.
2. Utiliser Tesseract OCR si le texte est incrusté dans l’image.
3. Générer des images stylisées avec Stable Diffusion (via ComfyUI ou API).
4. Proposer la possibilité d’entraîner de nouveaux styles (LoRA).
5. Assembler les images en vidéo (FFmpeg), avec transitions ou AnimateDiff.
6. Toujours coder en Python, et respecter l’architecture suivante :
   - parsing/

- generation/
- styles/
- video/
- ui/

7. Toujours répondre en français.

ux contraints : Toujours générer en 16:9

Générer image de start et end de chaque séquence

Les silhouettes doivent respecter le style "ombre chinoise" : pas de visage visible

Pour les styles labo : rendu photoréaliste, couleurs crédibles, lumière directionnelle

Suivre les directions de caméra du découpage (travelling, OTS, pano, etc.) // Dans votre fichier rules.json ou équivalent

```
{
  "services": [
    {
      "name": "comfyui",
      "description": "Service d'intégration avec ComfyUI",
      "dependencies": ["fastapi", "requests", "pillow"]
    },
    {
      "name": "file_manager",
      "description": "Gestion des fichiers et nomenclature",
      "dependencies": ["pillow", "os", "re"]
    }
  ],
  "endpoints": [
    {
      "path": "/api/generate",
      "method": "POST",
      "service": "comfyui",
      "function": "generate_images"
    },
    {
      "path": "/api/styles",
      "method": "GET",
      "service": "comfyui",
      "function": "get_available_styles"
    }
  ]
}
```