# Estúdio IA de Vídeos – Documento de Requisitos do Produto (PRD)

## 1. Visão Geral do Produto

O **Estúdio IA de Vídeos** é uma plataforma web (aplicação **SaaS**) destinada a empresas que desejam criar vídeos institucionais, educacionais e de treinamento de forma automatizada a partir de apresentações em PowerPoint (arquivos **.PPTX**). Utilizando tecnologias de **Inteligência Artificial** (IA), o sistema gera narrações em voz sintetizada e integra **avatares 3D realistas** que “apresentam” o conteúdo, automatizando grande parte do processo de produção audiovisual. Em vez de gravar vídeos manualmente com câmeras e narradores humanos, os usuários podem importar seus slides e obter um vídeo profissional narrado por vozes artificiais e **apresentado por um avatar virtual** em poucos minutos[[1]](https://www.synthesia.io/powerpoint-to-video#:~:text=How%20can%20I%20make%20a,video%20presentation%20with%20PowerPoint).

O público-alvo inclui **departamentos de Recursos Humanos, equipes de Treinamento e Desenvolvimento**, produtores de **EAD (Educação a Distância)**, **agências de comunicação interna** e **times de Marketing** corporativo. Esses usuários geralmente precisam produzir vídeos de onboarding de colaboradores, cursos online, comunicados institucionais ou demos de produtos de maneira rápida e padronizada. O valor central do Estúdio IA de Vídeos está em **agilizar** e **escalar** a criação de conteúdo audiovisual de qualidade sem exigir conhecimento técnico em edição de vídeo ou estúdios de gravação. Isso permite que as empresas transformem apresentações existentes em materiais em vídeo **engajantes e acessíveis em qualquer dispositivo**, ampliando o alcance e impacto da mensagem[[2]](https://www.synthesia.io/powerpoint-to-video#:~:text=How%20do%20I%20convert%20a,PowerPoint%20to%20video). Em suma, a plataforma traz **automação, economia de tempo e redução de custos** para a produção de vídeos corporativos, ao mesmo tempo em que mantém um alto padrão de qualidade visual e narrativa através de avatares humanos virtuais hiper-realistas.

## 2. Objetivos de Negócio

Os principais objetivos de negócio do Estúdio IA de Vídeos são:

* **Automação de Treinamentos e Comunicação:** **Automatizar** a conversão de apresentações e roteiros em vídeos, eliminando tarefas manuais repetitivas na produção de conteúdo. Com auxílio de IA, o sistema **reduz o tempo necessário** para criar vídeos de alta qualidade em comparação aos métodos tradicionais[[3]](https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD#:~:text=,realistas%20criam%20v%C3%ADdeos%20envolventes%20e), permitindo que treinamentos obrigatórios e comunicados institucionais sejam produzidos e atualizados com agilidade.
* **Escalabilidade e Rapidez:** Possibilitar a **escala** na geração de vídeos. Empresas poderão criar dezenas de vídeos simultaneamente (por exemplo, em diferentes idiomas ou para diferentes filiais) de forma consistente. A plataforma suporta um grande número de usuários e projetos, garantindo desempenho mesmo com alta demanda, de modo que a produção de conteúdo audiovisual **acompanhe o crescimento do negócio** sem gargalos.
* **Redução de Custos:** **Diminuir os custos** relacionados à produção de vídeos corporativos. O sistema dispensa a necessidade de estúdios de filmagem, câmeras, microfones e profissionais de narração, resultando em economia financeira. Geradores de vídeo com IA **eliminam a necessidade de equipamentos e softwares caros** de edição[[4]](https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD#:~:text=,interativos%20que%20cativam%20os%20aprendizes), tornando viável a criação de vídeos mesmo com orçamento limitado.
* **Acessibilidade e Consistência:** Aumentar a **acessibilidade** e o alcance do conteúdo. Os vídeos gerados podem ser assistidos em qualquer dispositivo (desktop, mobile) sem depender de software de apresentação, ao contrário de slides PPT tradicionais[[2]](https://www.synthesia.io/powerpoint-to-video#:~:text=How%20do%20I%20convert%20a,PowerPoint%20to%20video). Além disso, é possível incluir **legendas automatizadas e tradução** de narração para múltiplos idiomas, ampliando a inclusão de colaboradores com diferentes necessidades (ex.: legendas para deficientes auditivos). O conteúdo em vídeo garante **consistência** na mensagem transmitida – todos os colaboradores recebem a mesma informação apresentada da mesma forma, reduzindo interpretações divergentes.
* **Melhoria do Engajamento e da Aprendizagem:** Proporcionar **vídeos mais atraentes e envolventes** que aumentem o engajamento dos funcionários ou clientes treinados. Avatares de IA realistas e narrações naturais tornam os vídeos **mais interativos e cativantes para os aprendizes**, em comparação a slides estáticos[[5]](https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD#:~:text=mesmo%20para%20iniciantes.%20,interativos%20que%20cativam%20os%20aprendizes). Espera-se, assim, melhorar a retenção do conhecimento e a eficácia dos treinamentos e comunicados internos.
* **Inovação e Competitividade:** Posicionar a empresa usuária como inovadora ao adotar **tecnologia de ponta** em seus processos de comunicação. O uso de avatares virtuais e narração por IA em escala corporativa demonstra modernidade e pode **aumentar a competitividade**, agilizando a entrega de informações e padronizando conteúdos de treinamento globalmente. Além disso, a solução pode abrir novas possibilidades de conteúdo (por exemplo, personalização de vídeos conforme o público) com o uso de IA, mantendo o negócio na vanguarda tecnológica.

## 3. Funcionalidades Principais

A plataforma Estúdio IA de Vídeos fornecerá um conjunto robusto de funcionalidades para atender aos objetivos acima. As **principais funcionalidades** previstas são:

* **Importação de PPTX com Parser Nativo:** Upload de arquivos **.ppt/.pptx** e processamento interno que **parseia** os slides, extraindo textos, imagens e formatação básica. O sistema suporta apresentações do Microsoft PowerPoint, convertendo cada slide em uma estrutura interna editável. Isso permite, por exemplo, editar o texto importado diretamente na plataforma, semelhante a editar o PPT no próprio sistema[[6]](https://www.synthesia.io/powerpoint-to-video#:~:text=match%20at%20L364%20,easy%20to%20use%20as%20PowerPoint). Todo o conteúdo da apresentação (títulos, marcadores, notas de apresentação) será capturado para uso no vídeo.
* **Geração de Narração Automática (TTS):** Conversão de texto para voz utilizando **IA de voz**. Para cada slide ou roteiro fornecido, o sistema gera automaticamente a **narração em áudio** por meio de serviços de Texto-para-Fala (TTS) avançados. As vozes geradas são **naturais e humanas**, suportando múltiplos idiomas e entonações profissionais. O usuário poderá escolher entre vozes masculinas/femininas e idiomas disponíveis. A narração é sincronizada com o conteúdo de cada slide, servindo de base para animação labial do avatar.
* **Avatares 3D Falantes Realistas:** Integração de **avatares digitais em 3D** para apresentar os vídeos. O usuário poderá selecionar um avatar humanóide (por exemplo, modelos de alta fidelidade estilo **MetaHuman** ou avatars do **ReadyPlayerMe**) que atuará como apresentador do vídeo. O sistema utiliza algoritmos de animação facial (como NVIDIA **Audio2Face** ou técnicas de **lipsync** baseadas em blendshapes) para sincronizar perfeitamente os movimentos labiais e expressões do avatar com a narração gerada[[7]](https://www.aistudios.com/pt/ai-avatars#:~:text=Os%20avatares%20de%20IA%20mais,realistas%20do%20mundo). Os avatares também apresentarão **movimentos corporais naturais** (gestos, postura) pré-definidos ou aleatórios para adicionar dinamismo[[8]](https://www.aistudios.com/pt/ai-avatars#:~:text=Movimentos%20naturais%20do%20corpo). Essa funcionalidade permite que o vídeo tenha um “rosto” humano virtual transmitindo credibilidade e engajamento.
* **Timeline de Edição de Vídeo:** Um editor de vídeo embutido com **linha do tempo (timeline)** permitirá ajustes finos após a geração automática. Nele, o usuário poderá visualizar a sequência de slides/cenas do vídeo, **ajustar a duração** de cada cena, adicionar ou remover pequenas pausas, alterar a ordem se necessário, e inserir transições entre slides. Também será possível **alterar elementos visuais** (por exemplo, posicionar o avatar em um canto ou em tela cheia sobre o slide, trocar o background ou layout) e ajustar volumes de áudio. A interface da timeline será intuitiva e semelhante a ferramentas familiares (com funcionamento tão simples quanto editar no PowerPoint[[6]](https://www.synthesia.io/powerpoint-to-video#:~:text=match%20at%20L364%20,easy%20to%20use%20as%20PowerPoint)), para que **usuários de todos os níveis** possam refinar o vídeo sem dificuldade.
* **Exportação em MP4/WebM:** O sistema fornecerá a **renderização final do vídeo** nos formatos **MP4** e **WebM**, amplamente compatíveis com navegadores modernos e dispositivos móveis. O usuário poderá baixar o vídeo gerado em qualidade HD (pelo menos 1080p) ou até 4K, conforme necessidade. Alternativamente, poderá obter um link para streaming online via o player integrado. A exportação garante que o vídeo final combine todas as mídias (narração, avatar animado, slides, trilha sonora se houver) em um único arquivo otimizado. Formatos comprimidos (MP4/H.264, WebM/VP9 ou AV1) serão utilizados para equilibrar qualidade e tamanho de arquivo.
* **Upload de Arquivos e Gerenciamento de Assets:** Além de PPTX, o usuário poderá **enviar mídias adicionais** para usar no vídeo – por exemplo, vídeos curtos, áudios ou imagens que deseja inserir em certos pontos. Haverá uma **biblioteca de assets** no sistema onde ficam organizados os arquivos enviados e os gerados (por exemplo, narrações TTS geradas, vídeos finais e thumbnails). O usuário pode gerenciar esses assets, renomeando, removendo ou substituindo conforme necessário. Todo o armazenamento será em nuvem, permitindo acesso rápido e seguro aos arquivos. Isso facilita reaproveitar conteúdos já enviados (por exemplo, reutilizar uma música de fundo ou logo em múltiplos vídeos sem precisar re-upload).
* **Sistema de Pagamento por Assinatura:** O produto operará via modelo de negócios SaaS, com **assinaturas mensais/anuais**. Haverá integração com um gateway de pagamento (ex: Stripe, PayPal) para processar pagamentos de cartão de crédito de forma segura. Diferentes planos de assinatura darão direito a certos limites (por exemplo: número de vídeos por mês, duração máxima por vídeo, acesso a vozes premium ou avatares extras, etc.). O sistema deve gerenciar o ciclo de cobrança, renovação automática e expiração de contas não pagas. Também inclui interface de **administração de assinaturas** para que o cliente acompanhe seu plano, realize upgrades/downgrades e veja histórico de faturamento. Esse módulo garante a **monetização** da plataforma e um controle de acesso baseado no status de pagamento.
* **Sistema de Autenticação com SSO Empresarial:** Controle de acesso robusto, com suporte a **autenticação de usuários corporativos via SSO (Single Sign-On)**. Além do login/senha tradicional, empresas clientes poderão integrar seu provedor de identidade (por exemplo, **Azure Active Directory, Okta, Auth0**) para que funcionários façam login usando credenciais corporativas. Isso facilita a adoção em ambiente empresarial e aumenta a segurança (pois aplica políticas de senha e 2FA já existentes na empresa). O módulo de autenticação também incluirá gerenciamento de usuários, permitindo convites para membros da equipe dentro de uma organização, definição de papéis (admin, editor, visualizador) e opções de recuperação de senha para contas locais. O **SSO via SAML/OAuth2** estará disponível para clientes Enterprise, simplificando o acesso único à plataforma no ecossistema da empresa.
* **Worker System com Fila de Renderização:** A arquitetura do sistema contará com um **subsistema de *workers*** para processar tarefas pesadas (como geração de áudio, animação de avatar e render de vídeo) em segundo plano. Ao invés de realizar operações de longa duração na requisição web (o que travaria o usuário), o backend enviará tarefas para uma **fila de processamento** baseada em Redis (usando biblioteca **BullMQ**). Um ou mais processos *worker* consumirão essa fila e executarão os jobs de renderização de forma distribuída. Essa abordagem assegura que o usuário possa continuar usando a interface enquanto o vídeo é montado, e que várias renderizações ocorram em paralelo conforme a demanda. A tecnologia BullMQ é escolhida por sua robustez e já foi comprovada em produção para casos como **transcodificação de vídeos e processamento de imagens** em larga escala[[9]](https://bullmq.io/#:~:text=BullMQ%20has%20been%20succesfully%20used,other%20types%20of%20background%20jobs). O sistema de fila inclui mecanismos de **retry** em caso de falha de job, acompanhamento de progresso e priorização de tarefas urgentes.
* **Monitoramento e Logs:** Inclusão de ferramentas de **monitoramento** da saúde do sistema e coleta de **logs** para depuração. Métricas de utilização de CPU, memória, tempo de render, tamanho dos arquivos gerados, etc., serão coletadas e exibidas em um painel administrativo. Integridades dos serviços (uptime do servidor, fila de jobs, latência de APIs externas como TTS) serão monitoradas, possivelmente usando serviços como **Prometheus/Grafana** ou serviços gerenciados (Datadog, NewRelic). Em caso de erros ou exceções (ex: falha ao interpretar um PPTX, ou erro do FFmpeg), logs detalhados serão registrados para análise pela equipe técnica. Alertas automáticos poderão ser configurados (ex: aviso se uma fila de render acumular mais de X tarefas pendentes ou se um job falhar repetidamente). Esse monitoramento garante **confiabilidade** e permite escalar/investigar problemas proativamente.
* **CDN e Distribuição de Vídeos:** Integração com uma **Content Delivery Network (CDN)** para fornecer os vídeos gerados com baixa latência aos usuários finais. Todos os arquivos de vídeo e mídias serão entregues via servidores distribuídos geograficamente (por ex: **AWS CloudFront, Azure CDN ou Cloudflare**) para otimizar o carregamento independente da localização do usuário. Isso é fundamental caso os vídeos gerados sejam incorporados em plataformas de e-learning ou sites corporativos com audiência global. A CDN também ajuda a suportar picos de tráfego, aliviando o servidor de origem. O sistema deve gerar URLs públicos/compartilháveis dos vídeos através da CDN, de modo que um colaborador possa assistir ao vídeo instantaneamente, sem sobrecarregar a aplicação central.
* **Analytics de Uso e Métricas:** Dashboard de **analytics** integrado ao produto para acompanhar o uso da plataforma e o desempenho dos vídeos. Do lado da plataforma, serão registradas métricas como número de projetos criados, tempo médio de geração de vídeo, recursos mais utilizados (vozes, avatares), engajamento dos usuários editores, etc. Essas informações ajudam a equipe de produto a aprimorar o sistema. Adicionalmente, haverá analytics básicos dos vídeos finalizados – por exemplo, contagem de visualizações (quando os vídeos são hospedados na plataforma), tempo médio assistido, retenção por parte dos espectadores, entre outros. Para vídeos de treinamento, poderá ser útil integrar a padrões como **SCORM/xAPI** para reportar conclusão de vídeo no LMS corporativo[[2]](https://www.synthesia.io/powerpoint-to-video#:~:text=How%20do%20I%20convert%20a,PowerPoint%20to%20video). As métricas darão **insights** tanto ao cliente (sobre o alcance/uso dos vídeos) quanto internamente (sobre performance do sistema e preferência de funcionalidades).

## 4. Arquitetura Técnica

A arquitetura do Estúdio IA de Vídeos é projetada para ser **modular, escalável e robusta**, combinando tecnologias web modernas com componentes de processamento assíncrono. A seguir apresentamos o *stack* tecnológico sugerido e um panorama do fluxo entre frontend, backend e serviços de apoio.

### 4.1 Stack Tecnológico Recomendado

* **Frontend:** Desenvolvido em **Next.js (React)**, em TypeScript, para aproveitar Server-Side Rendering quando necessário e oferecer uma UI responsiva. O Next.js atuará tanto como aplicação web *SPA* quanto servidor de páginas (por exemplo, a tela de editor timeline pode ser uma página dinâmica). Bibliotecas de UI modernas (como Material UI ou Tailwind CSS) podem ser usadas para acelerar o design. O frontend comunicará com o backend via **API REST** (rotas API do Next.js) ou possivelmente **GraphQL** se conveniente, para operações como upload de arquivo, iniciar geração de vídeo, etc.
* **Backend & API:** Implementado em **Node.js** (podendo utilizar o próprio Next.js API Routes ou um servidor Express separado). Linguagem TypeScript para maior segurança nos tipos. O backend conterá a lógica de negócios: gerenciamento de projetos, orquestração das chamadas aos serviços de IA, controle de fila, etc. Será responsável por validar e armazenar dados de apresentações, e interagir com o banco de dados via ORM. Também gerenciará autenticação (integração SSO/OAuth2) e autorização (permissões por usuário).
* **Banco de Dados:** Provavelmente um banco **relacional (PostgreSQL)**, acessado via **Prisma ORM**. O banco armazenará informações estruturadas: dados de usuários, empresas, planos de assinatura, meta-dados dos projetos (slides extraídos, caminhos para arquivos de mídia gerados, histórico de versões), status de tarefas, logs resumidos, etc. O Prisma facilita as migrações de schema e consultas complexas de forma segura, além de se integrar bem com Node/TypeScript.
* **Armazenamento de Arquivos (Storage):** Uso de armazenamento em nuvem como **AWS S3** (ou equivalente GCP/Azure Blob) para guardar os arquivos binários: uploads de PPTX, imagens, vídeos gerados, áudios intermediários. O Prisma pode guardar somente os paths/URLs para esses assets, enquanto o conteúdo em si fica no bucket S3. O acesso aos arquivos será restrito (ex.: URLs assinadas para downloads privados, ou bucket privado servindo via CloudFront CDN para vídeos). Esse storage é escalável e ideal para streaming de vídeo.
* **Fila e Processamento Assíncrono:** **Redis** servirá como base para a **fila de jobs**, usando a biblioteca **BullMQ** no Node.js. Quando uma tarefa pesada é solicitada (renderização), o backend insere um job na fila; do outro lado, um ou mais **processos *worker*** ficam escutando essa fila no Redis. O Redis também será usado para caching de dados frequentes e comunicação breve entre serviços (por exemplo, flags de disponibilidade, locks distribuídos se necessário). O uso de fila desacopla componentes e permite **escalar horizontalmente** facilmente (vários workers em máquinas diferentes consumindo da mesma fila)[[10]](https://bullmq.io/#:~:text=jobs%20and%20sending%20messages%20using,process%20jobs%20across%20multiple%20servers)[[9]](https://bullmq.io/#:~:text=BullMQ%20has%20been%20succesfully%20used,other%20types%20of%20background%20jobs).
* **Workers de Renderização:** Os workers serão serviços Node.js separados, invocados para processar tarefas intensivas. Cada worker, ao receber um job de render, executará etapas como: geração de áudio TTS (chamada a API externa), render de animação do avatar (via engine 3D ou microserviço dedicado), composição de vídeo via **FFmpeg**, e upload do resultado ao storage. Por serem independentes do servidor web principal, podem ser dimensionados conforme a demanda de processamento (ex.: instâncias com GPU para acelerar a parte de avatar 3D, se aplicável, ou instâncias otimizadas para CPU para o FFmpeg). **FFmpeg** será utilizado via linha de comando ou via bibliotecas wrappers para concatenar vídeos, sobrepor camadas (por exemplo, overlay do avatar sobre o slide ou vice-versa) e gerar formatos finais. Esse pipeline assíncrono garante que mesmo operações demoradas não impactem a responsividade do frontend.
* **Serviços Externos de IA:** Integração com **APIs de terceiros** para funcionalidades de IA especializadas: por exemplo, serviço de TTS como **Microsoft Azure Cognitive Services (Speech)** ou **API do ElevenLabs** para síntese de voz de alta fidelidade; possivelmente serviços de avatar como **Did** ou **Render API do MetaHuman** para gerar animações faciais a partir de áudio, caso não seja feito internamente. Essas chamadas serão feitas pelo backend ou workers conforme necessidade, gerenciando chaves de API e observando limites de rate/pricing. Também é considerado o uso de APIs de *cloud* para tarefas como transcrição (caso futuramente se extraia áudio) ou tradução.
* **CDN e Delivery:** Como mencionado, uma CDN (por ex. **CloudFront** no AWS) ficará à frente do bucket S3 para entregar vídeos e assets estáticos (JS/CSS do frontend) eficientemente. O Next.js também pode servir parte do conteúdo estático pré-construído, mas vídeos de usuário devem ser streaming via CDN.
* **Autenticação e Autorização:** Implementada possivelmente com ajuda de bibliotecas como **NextAuth.js** (que suporta integrações OAuth corporativas) ou uso de **Auth0/Okta** para gerir SSO enterprise externamente. Token JWT serão usados para proteger as APIs. Comunicação segura sempre sobre HTTPS/TLS.
* **Monitoramento e Logs:** Logging centralizado com serviços como **Winston** + envio para console/cloud, e monitoramento via serviços cloud (CloudWatch, Application Insights) ou self-hosted (Prometheus).

Em resumo, a arquitetura segue um modelo **cliente-servidor com microsserviço de processamento** assíncrono, mantendo a interface ágil e delegando processos intensivos para workers dedicados. A figura abaixo ilustra os componentes e interações (em nível conceitual):

*(Figura arquitetural hipotética: Fluxo entre usuário, frontend, backend, fila/worker e serviços externos.)*

### 4.2 Fluxo de Processamento (Frontend–Backend–Storage)

Para exemplificar a interação dos componentes, segue um **fluxo típico** desde o upload de uma apresentação até a entrega do vídeo final ao usuário:

1. **Upload e Parsing:** O usuário acessa a página web e faz *login*. Na criação de um novo projeto, ele seleciona um arquivo PPTX e realiza o **upload** através da interface. O frontend envia o arquivo para a API do backend. O backend recebe o PPTX, armazena inicialmente o arquivo bruto no storage (ex: S3) e então aciona um módulo **parser** que lê o PPTX (pode usar uma biblioteca Node especializada). O parser extrai cada slide (texto, imagens, anotações) e salva esses dados estruturados no banco (tabelas de Slides, Elementos etc.), vinculados ao projeto do usuário.
2. **Configuração do Projeto:** O frontend então exibe os slides importados em forma de lista (permitindo ao usuário conferir ou editar textos se necessário). O usuário escolhe opções para geração: por exemplo, seleciona uma voz de narração dentre as disponíveis, escolhe um avatar 3D de preferência e define parâmetros como idioma, estilo de locução ou cor de fundo do vídeo. Ao finalizar as configurações, ele clica em “Gerar Vídeo”.
3. **Criação de Tarefa de Render:** Ao solicitar a geração, o backend cria um registro de tarefa de renderização no banco (com status “PENDENTE”) e então envia um job para a **fila BullMQ** no Redis contendo todos os dados necessários (ID do projeto, referências aos slides e às escolhas de voz/avatar). Em resposta imediata, o backend retorna ao frontend uma confirmação de que o processamento foi enfileirado, podendo fornecer um ID para que o frontend consulte o status periodicamente.
4. **Processamento Assíncrono no Worker:** Um processo **worker** (rodando possivelmente em outra instância de servidor) escuta a fila e recebe o job de render. Ele então executa a sequência de etapas:
5. **Geração de Áudio TTS:** Para cada slide (ou para o script completo, conforme implementação), o worker invoca a API de TTS escolhida (por ex., ElevenLabs ou Azure Speech) passando o texto e configurações de voz. Recebe de volta um arquivo de áudio (por ex. MP3) com a narração daquela parte. Esses arquivos de áudio podem ser armazenados temporariamente.
6. **Animação do Avatar 3D:** Em seguida, para cada trecho de áudio gerado, o sistema aciona o módulo de **avatar 3D**. Se for um avatar pré-renderizado (ex: usar um vídeo existente mapeado àquele áudio), o sistema obtém as animações labiais correspondentes. Em caso de integração em tempo real (ex: usando **NVIDIA Audio2Face**), o áudio é fornecido a um serviço que devolve uma animação facial (pontos ou keyframes) aplicada ao modelo 3D do avatar. Alternativamente, se estiver integrado a uma plataforma de avatar cloud (como **D-ID** ou Avatar API), o serviço retorna já um vídeo do avatar pronunciando o texto. De qualquer forma, o resultado dessa etapa são clipes de vídeo ou sequências de imagens do avatar **falando o conteúdo de cada slide** com sincronização labial precisa.
7. **Renderização do Vídeo Final:** O worker então combina os elementos usando **FFmpeg**. Para cada slide, uma cena de vídeo é montada: pode ser a imagem do slide de fundo e o avatar em picture-in-picture, ou o avatar sobreposto no canto com o slide ocupando o restante – o layout exato pode ser fixo ou configurável. FFmpeg compõe quadro a quadro o avatar sobre o slide (caso o avatar venha com fundo verde, por exemplo, faz *chroma key* para mesclar). O áudio narrado é inserido na timeline da cena. Ao terminar todas as cenas, o worker concatena os segmentos na ordem original dos slides, insere transições (por exemplo, crossfade entre cenas se previsto) e gera o arquivo de vídeo final em formato MP4/WebM. Configurações de bitrate, resolução e codec são aplicadas conforme padrões definidos.
8. **Exportação e Armazenamento:** O vídeo final é então enviado para o armazenamento (S3). O caminho/URL é atualizado no banco (associado ao projeto) e o status da tarefa de render é marcado como “CONCLUÍDO”.

Durante esse processo, o worker atualiza o progresso da tarefa (por exemplo, % concluído, slide atual sendo processado) em um campo de progresso no Redis ou banco, que pode ser lido pelo backend. Em caso de falha em alguma etapa (por exemplo, erro na API de voz), o worker registra o erro, pode tentar nova tentativa (**retry**) conforme política definida, ou marca como “FALHOU” após exceder tentativas.

1. **Notificação e Download:** O frontend, enquanto isso, pode periodicamente perguntar ao backend o status da tarefa (polling) ou o sistema pode enviar uma notificação (ex: via WebSocket ou e-mail) quando concluído. Assim que o vídeo estiver pronto, o usuário vê na interface um link para **visualizar ou baixar** o vídeo. Ao clicar, o vídeo é carregado do storage via CDN, proporcionando reprodução fluida. O usuário pode então assistir diretamente na plataforma (um player web HTML5 será disponibilizado) ou fazer download do arquivo MP4. Caso detecte algum ajuste necessário, o usuário pode voltar ao editor (timeline) e refiná-lo, gerando uma nova versão do vídeo (o fluxo se repete, criando talvez histórico de versões).
2. **Publicação ou Integração (Opcional):** Após validar o vídeo, o usuário pode compartilhar o link com sua equipe ou **incorporar o vídeo** em um LMS ou intranet corporativa. Graças ao formato de vídeo padronizado, ele pode ser enviado para plataformas de vídeo corporativas facilmente ou utilizado em apresentações. Se a plataforma suportar exportar nos padrões SCORM, nessa etapa o usuário também poderia baixar um pacote SCORM para integração em sistemas de aprendizagem, incluindo o vídeo e os meta-dados de tracking.

Este fluxo demonstra a **orquestração entre frontend responsivo, backend organizador e workers especializados**. A **decoupling** via fila assegura que mesmo processos demorados como a renderização de vídeo não travem a experiência do usuário[[11]](https://bullmq.io/#:~:text=Why%20Message%20Queues%3F)[[12]](https://bullmq.io/#:~:text=Offload%20long%20running%20tasks). Assim, o sistema consegue atender múltiplos pedidos simultaneamente, escalando os workers conforme a demanda para manter tempos de resposta adequados.

## 5. Critérios de Aceitação

Nesta seção, listam-se os critérios que determinarão se o produto atende aos requisitos. Incluem tanto **requisitos funcionais** (comportamento esperado do sistema) quanto **requisitos não-funcionais** (qualidades e restrições do sistema, como desempenho, usabilidade, etc.). Todos os critérios aqui devem ser observáveis, mensuráveis ou testáveis durante o desenvolvimento e QA do produto.

### 5.1 Requisitos Funcionais (Funcionalidades & Comportamento)

* **Importação de Apresentações:** O sistema deve permitir o upload de arquivos .PPTX de até um tamanho limite (ex: 100 MB) e **parsear** pelo menos os elementos essenciais de cada slide: textos (caixas de texto, marcadores), imagens inseridas e notas do apresentador. O conteúdo importado deve **corresponder fielmente** ao original em termos de ordem e hierarquia (por exemplo, se um slide possuía 3 tópicos bullet, esses três textos devem ser capturados separadamente). *Critério de aceitação:* após importação, o usuário visualiza todos os slides com seu texto e imagens na plataforma, podendo editar textos antes da geração.
* **Geração de Narração por IA:** Dado um texto (de slide ou roteiro) e uma seleção de voz, o sistema deve conseguir **produzir um áudio de voz artificial** que pronuncie o texto de forma clara e natural. A pronúncia de termos técnicos ou siglas deve ser inteligível (suporte a configurações via SSML, se necessário, para ajustar pronúncias). *Critério:* para um texto de exemplo fornecido, o áudio gerado deve ter entonação adequada e ausência de erros grosseiros de pronúncia. Também, deve respeitar o idioma selecionado (ex.: se o texto está em português, não falar com sotaque inglês, etc.).
* **Avatar 3D Sincronizado:** O sistema deve gerar a animação de um **avatar digital** falando o texto em sincronismo com o áudio narrado. Os movimentos de boca (lábios) do avatar devem estar **alinhados temporalmente** ao áudio (sem atrasos perceptíveis) e representar fonemas correspondentes (movimentação labial realista). *Critério:* em um teste visual, ao reproduzir o vídeo, observa-se que o avatar “articula” as palavras de forma crível e sem desacordo evidente com a fala. Além disso, o avatar deve piscar os olhos e realizar pequenos movimentos de cabeça/corpo para não ficar totalmente estático (incrementando realismo).
* **Composição Vídeo + Slides:** Cada slide da apresentação deve aparecer no vídeo final de acordo com uma **duração adequada** à narração. O sistema deve combinar o visual do slide com o avatar falando. Caso um slide tenha muito texto, se necessário dividir em múltiplas cenas ou rolar o texto, isso deve ocorrer suavemente. *Critério:* o vídeo resultante exibe todo o conteúdo textual que estava nos slides (legível, em resolução HD), sincronizado com a fala do avatar (o slide permanece em tela pelo tempo que o avatar leva para narrá-lo). Transições entre slides não devem cortar frases.
* **Editor Timeline (Edição Fina):** Após a geração automática, o usuário deve conseguir realizar ajustes na timeline: modificar durações de slides (estender ou reduzir tempo de permanência em cena), inserir pausa entre dois slides, ou até separar um slide em duas cenas. *Critério:* na interface de edição, ao arrastar a duração de um elemento na timeline ou reordenar slides, o sistema atualiza esses valores e reflete as mudanças em uma nova renderização. Nenhum crash deve ocorrer ao manipular timeline mesmo com muitos elementos. Os ajustes feitos devem ser salvos e respeitados na geração final.
* **Re-renderização e Versões:** O sistema deve suportar iterar gerações. Ou seja, após editar algo (texto de slide ou timeline), o usuário pode gerar uma nova versão do vídeo. *Critério:* o sistema mantém histórico de versões ou pelo menos não impede refazer; a nova geração deve criar um novo arquivo de vídeo sem sobrescrever automaticamente a anterior (a não ser que desejado), permitindo comparar resultados.
* **Exportação dos Vídeos:** Deve ser possível **baixar o vídeo** final em formato .MP4 (codec H.264 ou H.265) e/ou .WebM (codec VP9/AV1) através da interface. A resolução padrão de saída deve ser 1920x1080 px (Full HD) salvo especificação contrária do usuário. *Critério:* um vídeo de teste gerado pode ser baixado e reproduzido offline em players comuns (VLC, etc.) sem erros, e terá qualidade visual e sonora satisfatória (sem cortes abruptos, artefatos graves de compressão ou áudio desincronizado).
* **Desempenho de Geração:** Para um projeto típico (por ex, apresentação de ~10 slides, 5 minutos de vídeo final), o sistema deve **gerar o vídeo em um tempo hábil**. Idealmente em poucos minutos (meta: < 5 minutos), embora para vídeos mais longos possa demorar mais. *Critério:* em testes, 90% das gerações de vídeo de até 5 min devem completar em menos de 10 minutos de processamento após o upload. O usuário deve ser informado do progresso ou ter indicação de que está em andamento.
* **Sistema de Pagamento e Acesso:** Usuários sem assinatura não devem conseguir acessar a área de criação de vídeos (ou ficam num modo de avaliação limitado). Somente contas ativas (pagamento em dia) usufruem plenamente. *Critério:* simular uma conta expirada – o sistema deve bloquear a geração de novos vídeos e exibir mensagem para renovar assinatura. Para contas novas, o processo de checkout deve ativar imediatamente o plano após confirmação do pagamento.
* **Autenticação e SSO:** O login via SSO corporativo deve permitir que um usuário da empresa acesse sem necessidade de criar senha separada. *Critério:* após configurar uma integração SAML com um provedor (e.g., Azure AD), um usuário convidado dessa empresa consegue clicar em “Login corporativo”, ser redirecionado ao IdP e retornar autenticado, vendo seus projetos. As sessões devem expirar conforme política (ex: 1h de inatividade) e permitir logout manual.
* **Colaboração Multiusuário (se previsto):** Se o sistema prever múltiplos usuários por empresa, deve permitir compartilhamento de projetos entre eles. *Critério:* um usuário “editor” convidado consegue acessar um projeto existente da sua empresa, editar e gerar vídeo, e outro usuário “admin” consegue ver e gerenciar todos projetos da empresa.
* **Upload e Asset Management:** O usuário deve conseguir enviar imagens extras ou vídeos para inserir. *Critério:* ao clicar “adicionar mídia” e selecionar um arquivo (jpeg, png, mp4 curto etc.), o upload conclui e o item aparece na biblioteca. Deve ser possível então arrastar essa mídia para um slide ou cena da timeline, e ela aparecer no resultado final no lugar configurado.
* **Analytics de Vídeo:** Caso aplicável, ao disponibilizar o vídeo via player interno, deve registrar métricas básicas. *Critério:* o sistema coleta ao menos visualizações únicas e tempo médio assistido. Essas métricas aparecem em um painel para o criador (por exemplo: “Vídeo X – 50 visualizações, média 70% assistido”).

### 5.2 Requisitos Não-Funcionais (Desempenho, Qualidade, Restrições)

* **Desempenho e Escalabilidade:** O sistema deve suportar **múltiplos usuários concorrentes** sem degradação significativa. A arquitetura de fila e workers deve escalar horizontalmente; por exemplo, adicionar mais workers para processar vídeos em paralelo. *Critério:* a aplicação web (frontend/backend) deve atender 1000 usuários simultâneos navegando/ editando projetos com tempo de resposta das páginas < 2 segundos em 95% das requisições. O sistema de render deve conseguir processar, digamos, 20 vídeos simultaneamente mantendo tempos de conclusão dentro do esperado (podendo enfileirar os demais).
* **Confiabilidade e Tolerância a Falhas:** Em caso de falha de um componente (ex: um worker travar ou API de voz ficar indisponível temporariamente), o sistema deve **recuperar** graciosamente. *Critério:* se um job de render falhar, ele deve ser re-tentado automaticamente X vezes ou marcado como erro com mensagem clara ao usuário, sem causar fila presa. O crash de um worker não deve derrubar o sistema todo; um novo worker deve assumir os jobs pendentes (cluster de workers). Backups diários do banco de dados e possibilidade de retomada de geração do ponto de falha (em futuras versões) são desejáveis.
* **Segurança:** Dados corporativos (slides enviados, vídeos gerados) devem ser armazenados com segurança. *Critério:* arquivos em repouso no S3 devem estar **criptografados** (AES-256 server-side encryption, por exemplo). O acesso aos vídeos deve ser restrito – URLs de download expiram ou são protegidas por token. A aplicação deve prevenir ataques comuns (XSS, CSRF, SQL Injection – com ORM isso é mitigado). Políticas de senha para usuários locais (mínimo 8 caracteres, etc.) ou exigência de SSO para empresas grandes. Todos acessos e mudanças relevantes devem ser auditáveis (log de auditoria de quem gerou ou baixou tal vídeo, etc.).
* **Usabilidade e UX:** A interface deve ser **intuitiva e fácil de usar**, considerando o público de negócios que pode não ter background técnico em edição de vídeo. *Critério:* testes de usabilidade com usuários-chave devem demonstrar que as pessoas conseguem importar um PPT, selecionar avatar/voz e gerar um vídeo básico **sem treinamento prévio**, em no máximo 10 minutos. Elementos da UI devem seguir padrões (drag-and-drop na timeline, ícones familiares, textos explicativos). Feedback do sistema deve ser claro (barras de progresso, notificações quando pronto, etc.).
* **Tempo de Aprendizado:** Relacionado à usabilidade, espera-se que um novo usuário consiga entender o fluxo de criação de vídeo rapidamente. Documentação e tutoriais devem estar disponíveis. *Critério:* a curva de aprendizado deve ser baixa; por exemplo, >= 90% dos usuários de teste conseguem completar uma tarefa de criação de vídeo simples sem solicitar ajuda.
* **Acessibilidade (Accessibilidade Digital):** A aplicação web deve seguir **boas práticas de acessibilidade** (WCAG 2.1 AA). *Critério:* todos os recursos devem ser utilizáveis via teclado (foco visível, atalhos), deve haver suporte a leitores de tela nos elementos principais (por exemplo, adicionar *aria-labels* nos botões de upload, play, etc.). Contraste de cores nos textos e ícones suficiente para daltônicos. Além disso, os vídeos gerados devem contemplar acessibilidade de conteúdo – p.ex., permitir ativar **legendas automáticas** embutidas no vídeo (closed caption gerado a partir do script).
* **Compatibilidade de Navegadores:** O front-end deve ser testado nos principais navegadores modernos (Chrome, Firefox, Edge, Safari) e suportar as versões mais recentes. *Critério:* funcionalidade plena (upload, editor timeline, player) no Chrome e Firefox (últimas versões) e no Safari/Edge sem bugs críticos. Em mobile browsers/tablets, ao menos reprodução de vídeos e visualização de projetos deve funcionar, embora a edição completa possa ser otimizada para desktop.
* **Escalabilidade de Armazenamento:** O design deve considerar potencial grande volume de dados (vídeos podem ser pesados). *Critério:* o sistema deve conseguir gerenciar, por exemplo, **terabytes de vídeos** sem perda - isso implica segmentação de buckets, limpeza de dados antigos, etc. O desempenho de leitura/escrita no storage via CDN deve se manter estável com milhares de vídeos armazenados.
* **Manutenibilidade e Extensibilidade:** O código e arquitetura devem facilitar futuras expansões (ex.: adicionar novos idiomas de voz, novos modelos de avatar, ou novos recursos como geração de quizzes interativos no vídeo). *Critério:* avaliação interna de código (code review) atesta modularidade (cada funcionalidade principal isolada em serviço/modulo), e documentação de API/arquitetura existe para onboard de novos desenvolvedores. Integrar um novo provedor de TTS ou trocar o mecanismo de fila (se necessário) não deve requerer reescrever todo o sistema.
* **Compliance e Privacidade:** Considerar conformidade com LGPD/GDPR dado que serão enviados possivelmente dados corporativos e vozes/rostos sintéticos. *Critério:* existência de termos de uso e consentimento do usuário para processar os dados; opção de remover definitivamente dados de usuário se requisitado. Nenhum dado sensível (ex: conteúdo confidencial dos slides) deve ser exposto a outros clientes ou utilizado fora da finalidade de geração de vídeo.

Cada critério acima deverá ser validado durante a fase de testes (incluindo testes automatizados para desempenho e segurança, testes de usabilidade com usuários reais, etc.). Somente ao atender a esses critérios o produto será considerado apto para implantação em ambiente de produção.

## 6. Módulos Detalhados

Nesta seção são descritos **12 módulos principais** do sistema (conforme listados anteriormente em funcionalidades), detalhando o escopo de cada um – o **que será implementado** – e seu **impacto no sistema** (como interage com outros módulos, requisitos técnicos especiais, efeitos na experiência do usuário ou na performance). Essa decomposição modular orienta as equipes de desenvolvimento sobre as partes constituintes do produto e suas responsabilidades específicas.

### 6.1 Importação de PPTX com Parser Nativo

**Descrição:** Módulo responsável por **carregar e interpretar arquivos .PPTX** enviados pelo usuário. Envolve a interface de upload (arrastar e soltar ou seleção de arquivo) e a lógica de parsing. Utilizará bibliotecas para leitura do formato Office Open XML (ex.: **libreoffice headless**, **Aspose.Slides**, **PptxGenJS** ou outros). Deve extrair para cada slide: textos (com hierarquia de título, subtítulo, bullets), imagens (exportando-as para arquivos PNG/JPEG temporários), formas básicas e possivelmente anotações. O resultado do parser é estruturado em objetos (ex: lista de slides contendo elementos). Deve lidar com erros de parsing de forma graciosa (ex: se o PPTX estiver corrompido, avisar o usuário). Pode incluir conversão de slides em imagens prévias (thumbnails) para exibir no editor.

**Impacto no Sistema:** Este é o primeiro passo do fluxo – se o parser falhar ou for limitado, todo o restante é afetado. Ter um parser robusto garante que o **conteúdo fonte** seja bem representado no vídeo final. O módulo impacta a carga no backend (pode consumir CPU e memória ao processar arquivos grandes). Talvez seja necessário limitar quantas conversões simultâneas ocorrem ou offload para um serviço separado se for pesado. Do ponto de vista de UX, um parser eficiente reduz o tempo de espera após upload e evita retrabalho (usuário não precisa arrumar manualmente textos faltantes). Além disso, a fidelidade do parser impacta a **qualidade do vídeo** – por exemplo, garantir que bullet points virem fala enumerada corretamente, etc. Precisaremos manter este módulo atualizado para novos formatos de PPTX ou características (por ex., se suportar vídeos/áudio embutido no slide para futura extensão). Em suma, é um módulo crítico para **qualidade de entrada de dados**, com impacto direto na satisfação do usuário ao ver seus slides corretamente interpretados.

### 6.2 Geração de Vídeo com Narração IA (TTS)

**Descrição:** Engloba a funcionalidade de **Texto-para-Fala** e coordenação da narração no vídeo. O que será feito: integração com um ou mais serviços de TTS (e.g. ElevenLabs, Azure Speech, Google Cloud TTS). Haverá uma camada para gerenciar requisições de geração de voz: enviar texto e parâmetros (idioma, voz escolhida, velocidade) e receber o áudio. É necessário também tratamento do texto: remover caracteres inválidos, possivelmente dividir textos muito longos (limites de API), e inserir marcadores de pausa se o usuário configurar. Após obter o áudio, o sistema deve **sincronizar** cada áudio com o slide correspondente (armazenar duração, etc.). Pode haver opção de pré-escuta: o usuário ouvir a voz selecionada antes de gerar todo o vídeo. Além disso, deve-se implementar *fallback* – se um serviço de TTS falhar ou estiver indisponível, tentar outro (ex: Azure fallback para Google) ou notificar erro claramente. Geração de closed captions (legendas) também entra aqui: já que temos o texto e o áudio, podemos produzir um arquivo de legenda (SRT) alinhado.

**Impacto no Sistema:** Este módulo consome **serviços externos** e, portanto, impacta custos (TTS geralmente é cobrado por caracteres ou tempo de áudio). Precisamos otimizar chamadas (não gerar vozes redundantes; talvez cache de áudio para textos iguais dentro do projeto). Em performance, a geração de áudio é relativamente rápida (segundos), mas pode se tornar gargalo se milhares de slides forem processados – outro motivo para fila e processamento distribuído. Do ponto de vista do usuário, a qualidade da narração é crucial: vozes muito robóticas ou com erros tirarão a credibilidade do vídeo. Portanto, a seleção de bons mecanismos de TTS (ex.: ElevenLabs oferece vozes muito naturais[[13]](https://elevenlabs.io/developers#:~:text=Quickly%20generate%20AI%20voices%20in,integrate%20ElevenLabs%20into%20your%20application)) impacta diretamente a **qualidade percebida** do produto. Também a latência de geração (espera pela voz) influencia o tempo total de renderização. O módulo também se relaciona com o de **Avatares 3D** – pois o áudio gerado aqui é insumo para animação labial. Qualquer alteração no texto ou voz requer re-gerar o áudio, então deve estar bem acoplado com a timeline (ex.: se ajustar texto no slide 3 e re-render, só chamar TTS para slide 3 novamente). Em resumo, este módulo afeta **custos de operação, tempo de processamento e qualidade do resultado sonoro**, sendo fundamental calibrar e testar diferentes provedores de voz.

### 6.3 Avatares 3D Falantes (Integração MetaHuman/Audio2Face/ReadyPlayerMe)

**Descrição:** Módulo dedicado à gestão e geração dos **avatares digitais** no sistema. Engloba duas partes: a seleção/gerenciamento de avatares e a animação dos mesmos conforme o áudio. O que será implementado: inicialmente teremos um conjunto de avatares disponíveis (por exemplo, um avatar masculino e um feminino padrão). Podem ser modelos pré-renderizados (vídeos) ou modelos 3D animáveis em tempo real. Considerando as tecnologias citadas, podemos ter uma integração com **ReadyPlayerMe** para permitir criação/customização de avatar pelo usuário (ex.: escolher aparência) – este seria o gerenciamento. Para animação, integração com **NVIDIA Audio2Face**: teremos uma pipeline onde dado um avatar (com rig facial compatível, ex: blendshapes padrão) e um áudio, o Audio2Face (rodando em um servidor com GPU) gera animação facial (sequência de keyframes ou mesh deformado) que aplica ao avatar. Alternativamente, usar *plugin* de lipsync (como Oculus Lipsync, disponível para modelos com blendshapes[[14]](https://forum.readyplayer.me/t/lipsync-avatar-with-recorded-voice/314#:~:text=Lipsync%20avatar%20with%20recorded%20voice,synced%20with%20the%20recorded%20audio)[[15]](https://docs.readyplayer.me/ready-player-me/api-reference/avatars/morph-targets/oculus-ovr-libsync#:~:text=Oculus%20OVR%20LipSync%20,supports%20the%20Oculus%20Lipsync%20API)) para gerar visemas a partir do áudio e animar o avatar. O módulo deve então renderizar ou produzir o vídeo do avatar falante, preferencialmente com fundo neutro (verde ou transparente) para compor sobre slides. Se for usado MetaHuman, possivelmente precisaremos usar a Unreal Engine via cloud ou pré-gravar falas, o que é complexo – talvez num MVP se use avatars menos pesados primeiro (ex: ReadyPlayerMe no Unity/Three.js via WebGL). Em resumo, esse módulo fará a **transformação de áudio em animação facial+corporal**. Inclui ajustar expressões (sorrisos, sobrancelha) para não ficar monótono. Poderá ter uma biblioteca de gestos (ex: acenar com cabeça, mover mãos) e os distribuir aleatoriamente ou sob comando (pontos específicos do texto).

**Impacto no Sistema:** Este é um dos módulos de maior impacto na **experiência do usuário final** – a credibilidade do avatar influencia diretamente o quão profissional o vídeo parece. Tecnicamente, é um módulo pesado: animação 3D e render podem consumir muita CPU/GPU. Precisaremos de infraestrutura adequada (ex: um server com GPU para Audio2Face ou Unreal if needed). Isso também impacta custo (uso de GPU na nuvem é caro). Podemos mitigar usando avatares pré-renderizados para vozes padrão (ex: uma biblioteca de vídeos para frases comuns) no início. Em termos de integração, o output deste módulo (vídeo do avatar falando) vai para o módulo de **renderização final** (com FFmpeg) onde será sobreposto ao slide – portanto, deve ser produzido em resolução consistente (por ex, 1080p), com frame rate adequado (ideal 30fps ou 60fps se buscam hiper-realismo). Um risco a gerenciar é **sincronismo**: devemos garantir que o tempo do vídeo do avatar case exatamente com o áudio (mesma duração). Também, se o avatar for renderizado a parte, precisamos sincronizar isso no tempo do FFmpeg. Impacto na UX: o usuário deve achar os avatares convincentes; se parecer um boneco sem vida, pode comprometer a adoção. Logo, possivelmente focar em avatares estilo desenho animado/cartoon inicialmente (que geram menos estranhamento) ou já tentar alta qualidade (MetaHuman) mas com custo de infra. Outro impacto: **escalabilidade** – gerar 1 avatar animado é ok, mas e 10 simultâneos? Precisaremos fila ou instâncias paralelas. Esse módulo pode se tornar gargalo se não escalado (ex: se rodar em apenas um engine). Por fim, temos considerações de formato: se usar Unity/Unreal, integrar seu output no pipeline do Node (talvez via comando ou serviço web). Em suma, o módulo de avatar é tecnicamente desafiante e crucial para o valor do produto, pois entrega o *“fator Uau”* de ter um apresentador digital no vídeo. Teremos que iterar nele para equilibrar **qualidade visual vs. recursos computacionais**.

### 6.4 Timeline de Edição de Vídeo (Editor)

**Descrição:** Este módulo corresponde à interface e lógica de **edição não-linear** do vídeo dentro do navegador, após a geração inicial. Inclui o desenvolvimento de uma **timeline interativa**, onde cada faixa representa um componente (ex: faixa de slides, faixa do avatar, faixa de áudio, faixa de legendas). O que será feito: criação de uma UI onde os slides importados são dispostos sequencialmente; o usuário pode clicar e arrastar as bordas de um item para ajustar sua duração (por exemplo, prolongar a exibição de um slide em +2 segundos após a fala). Também poderá dividir ou unir segmentos – por exemplo, inserir uma pausa (silêncio) como elemento. Implementar zoom na timeline para ajustes precisos, e pré-visualização rápida (scrubbing). O editor deve permitir ações como: *reordenar slides* (drag and drop de uma faixa do slide inteiro), *inserir transições* (talvez um menu suspenso para escolher fade, corte seco, etc.), e *adicionar trilha de música de fundo* (um upload de áudio posicionado em uma faixa de background). A lógica deve garantir que essas edições fiquem armazenadas (talvez como um JSON de *edit decision list*) para serem aplicadas na próxima renderização. Além disso, precisa refletir em tempo real mudanças básicas – por ex., se usuário corta uma frase do texto, a duração prevista muda. Haverá botão de “pré-visualizar” que toca uma versão simplificada (talvez só áudio e slides) para conferência antes do render final.

**Impacto no Sistema:** Este módulo é quase totalmente front-end, mas impacta o backend indiretamente quando as alterações precisam ser salvas e reenviadas para render. A existência do editor agrega um enorme valor de **flexibilidade** ao produto, permitindo que usuários avancados personalizem o resultado além da automatização padrão (o que é um diferencial frente a ferramentas 100% automáticas). O impacto em UX é significativo: se implementado corretamente, dá controle ao usuário sem ser complexo demais. Porém, tecnicamente, criar um editor de vídeo web tem desafios – sincronização precisa de áudio/vídeo, desempenho do navegador ao lidar com múltiplos elementos visuais (usar Canvas ou WebGL para timeline pode ser opções). Também há risco de complexidade: precisamos manter a UI simples para não assustar usuários não técnicos. Possivelmente implementar funcionalidades gradualmente (MVP da timeline com funções básicas e depois adicionar transições, etc.). Em termos de dados, o módulo gera uma representação das edições (por exemplo, um esquema JSON com timestamps ajustados) – isso precisa ser aplicado no módulo de render (workers devem interpretar para gerar o vídeo fiel ao que foi editado). Isso demanda integração forte entre front (editor) e back (render pipeline). Por exemplo, se usuário colocou música de fundo, o worker FFmpeg precisa mixar esse áudio no output. Se colocou transição crossfade, o FFmpeg precisa gerar essa transição. Então, há impacto no módulo 6.5 (exportação/renderização) que deve suportar essas instruções. Performance: o editor em si deve ser **fluido**, arrastar elementos sem travar, mesmo para vídeos de 10+ minutos (timeline longa). Pode exigir virtualização de elementos ou limitações (ex: zoom out máximo). Em resumo, o editor timeline é o **coração interativo** para o usuário refinar o vídeo, com impacto direto na usabilidade e precisando casar perfeitamente com a lógica de render do vídeo final.

### 6.5 Exportação em MP4/WebM (Renderização Final)

**Descrição:** Corresponde ao módulo de **render engine** que pega todos os componentes (áudios, vídeos do avatar, imagens dos slides, edições da timeline) e produz o arquivo final nos formatos requeridos. O que será implementado: scripts FFmpeg ou utilização de APIs de multimídia (por ex, **GStreamer** ou bibliotecas FFmpeg/avconv) no backend para automatizar a composição do vídeo. A exportação suportará pelo menos dois formatos de saída: MP4 (codec H.264 ou H.265) e WebM (VP9 ou AV1). Serão definidas configurações padrão: bitrate de vídeo (ex: 4Mbps para 1080p), codec de áudio (AAC para MP4, Opus para WebM), etc., com possibilidade de ajustar qualidade se o usuário escolher. O módulo recebe como entrada: os arquivos gerados (imagens dos slides ou vídeo dos slides se pré-renderizar transições, áudio narrado para cada trecho, vídeos do avatar correspondentes). Então, para cada cena/slide, o FFmpeg vai combinar vídeo do slide (pode ser estático exibido por X segundos) com o vídeo do avatar (como *overlay* menor ou full screen side-by-side dependendo do design). Inserir o áudio correspondente naquele segmento. Entre cenas, inserir transições – ex: 1s fade. Por fim, concatenar todos segmentos codificando em um só fluxo de saída. O módulo também adiciona eventualmente **legendas** (que podemos queimar no vídeo ou gerar faixa separada). Após renderizar, salva no arquivo de destino, e valida integridade (checar se vídeo gerado tem duração esperada, etc.). Se o usuário escolheu ambos MP4 e WebM, poderemos fazer dupla passagem ou converter um a partir do outro (mas provavelmente renderizar duas vezes com FFmpeg diferentes codecs).

**Impacto no Sistema:** Este módulo é o **núcleo do processamento pesado**. Impacta uso de CPU intensivo no servidor; FFmpeg em resoluções altas demanda bastante poder computacional e I/O. É aqui que a decisão de usar um sistema de fila e workers separados se justifica – para isolar essa carga. A eficiência deste módulo afeta diretamente o tempo de espera do usuário para obter o vídeo. Temos que otimizar comandos FFmpeg (e.g., usar *hardware acceleration* se disponível, encoders rápidos, etc.). Também cuidar de memory/disk: processar vídeos de longa duração consome espaço temporário e memória; possivelmente precisa escrever segmentos em disco temporário para depois juntar. Impacto de qualidade: a forma como compomos slide+avatar definirá o visual – precisamos escolher layout adequado e garantir que a resolução final mantenha textos legíveis (talvez ampliando slides se contêm muito texto). Além disso, se a timeline permitiu música de fundo, esse módulo mixa o áudio de narração com a música (ajustando volume – ducking se necessário durante fala). Qualquer descompasso aqui será percebido no resultado, então requer muitos testes. Impacto de formatos: oferecer dois formatos agrega compatibilidade, porém dobra processamento para cada vídeo (podemos tornar WebM opcional ou gerar sob demanda). Em termos de fluxo, uma vez gerado o arquivo, ele é grande – enviar pro storage e disponibilizar via CDN impacta rede; ideal ter local do worker no mesmo data center do storage para upload rápido. Devemos também limpar arquivos temporários pós uso, para não encher disco do servidor. Por fim, sucesso deste módulo significa entregar um vídeo final que atenda expectativas – se falhar ou produzir glitch, o usuário notará imediatamente. Portanto, terá atenção especial em testes de qualidade de vídeo e áudio. Em suma, é o módulo onde todas as peças se unem; seu desempenho e confiabilidade determinam em grande parte a **satisfação do usuário** (eles julgam pelo vídeo final).

### 6.6 Upload de Arquivos e Gerenciamento de Assets

**Descrição:** Módulo dedicado a **gerenciar arquivos de mídia** dentro da plataforma. Inclui o sistema de upload genérico (imagens, vídeos adicionais, áudios de música) e a interface de **biblioteca de mídia**. O que será implementado: no frontend, uma seção "Meus Arquivos" onde o usuário vê todos uploads feitos (por projeto ou globais). Possibilidade de organizar em pastas simples (ou tags) para facilitar busca se houver muitos assets. O ato de upload será tratado via backend (com possivelmente assinatura de URL S3 para envio direto do navegador, otimizando throughput). Após upload, metadados do arquivo (nome original, tipo, tamanho, data) são salvos no banco. O usuário poderá **pré-visualizar** esses assets (thumbnails para imagens, player para áudio/vídeo curto) e inserir em projetos. Por exemplo, arrastar uma imagem da biblioteca para um slide específico ou definir um vídeo de introdução. O gerenciamento inclui também **quota** de armazenamento por usuário/plano – verificar e impedir uploads que excedam (ex: plano básico tem 1GB total de assets). Implementar exclusão de asset (com confirmação), levando em conta se está em uso em algum projeto (talvez bloquear remoção se usado).

**Impacto no Sistema:** Este módulo lida com possivelmente muitos arquivos binários, implicando volume de armazenamento e tráfego. Impacta custos de infraestrutura (armazenamento em nuvem e egress de dados da CDN). Precisamos garantir segurança: assets de um usuário não acessíveis por outro (controle por pasta ou prefixo no bucket e autorização). Também otimização: possivelmente gerar versões otimizadas de imagens (thumbnails menores) para não sobrecarregar a interface. Em termos de UX, ter um gerenciador de assets robusto facilita o workflow de produção – o usuário pode reutilizar conteúdo existente facilmente, aumentando eficiência. Se mal implementado (ex: sem busca, sem organização), pode virar um caos para usuários frequentes. Impacto na integração: O módulo de render precisará acessar esses assets (por exemplo, se usuário inseriu um vídeo introdutório, o FFmpeg deve pegar esse vídeo do storage). Então, os links ou paths devem ser resolvidos e disponíveis durante o processamento (podemos pré-baixar local no worker antes de compor). O sistema de upload também impacta *scalability* – se muitos usuários subindo arquivos simultâneos, devemos usar transferência direta a S3 e não carregar tudo via servidor (para não congestionar). Isso implica mais complexidade (presign URLs, etc.), mas alivia backend. Do ponto de vista de segurança, validação de tipos (não permitir executáveis disfarçados) e antivírus em arquivos pode ser necessário em ambiente corporativo. Além disso, assets aumentam a **riqueza dos vídeos** (ex: inserir logotipos, fotos reais da empresa nos slides), o que torna o produto mais útil. Portanto, esse módulo suporta a **personalização** do conteúdo e precisa ser confiável na gestão desses arquivos.

### 6.7 Sistema de Pagamento por Assinatura

**Descrição:** Módulo encarregado do **gerenciamento de planos e cobranças** dos usuários/empresas. O que será feito: integração com um serviço de pagamento (ex.: **Stripe**) para criar planos (products) e assinatura recorrente. Na aplicação, telas para escolher um plano, inserir dados de pagamento (cartão de crédito), e efetivar a assinatura. Implementar webhooks do provedor de pagamento para ouvir eventos como pagamento aprovado, assinatura cancelada, cartão expirado, etc., e atualizar o estado do usuário no nosso banco. Também permitir que o usuário veja/edite suas informações de billing: atualizar cartão, fazer upgrade ou downgrade de plano (pró-rata se aplicável), cancelar renovação. O sistema deve oferecer pelo menos dois níveis de plano (ex: **Basic** e **Enterprise**), com diferenças como limites ou recursos extras (talvez Enterprise com SSO e mais avatares). Uma consideração é oferta de um **trial** gratuito X dias – implementável via Stripe trial period ou logic custom. No backend, funções para checar status do usuário antes de permitir ações (ex: se não pago, bloquear geração). Emails transacionais podem ser enviados (recibo, aviso de falha de pagamento).

**Impacto no Sistema:** Este módulo garante a **sustentabilidade financeira** da plataforma, então é crucial que funcione corretamente – erros aqui podem significar perda de receita ou travar usuários pagantes indevidamente. Segurança é vital: não armazenamos dados de cartão (isso fica no Stripe), mas devemos proteger endpoints de assinatura para evitar uso indevido (ex: alguém tentando alterar preço via API). A integração de webhook deve ser robusta (verificar assinaturas para garantir que é o provedor chamando) e lidar com casos como pagamento recusado (notificar usuário para ação). Impacto na UX: o fluxo de pagamento deve ser o mais fluido e confiável possível, pois é um ponto sensível – se for complicado ou falhar muito, afasta clientes. Portanto, optar por uma solução conhecida (Stripe Checkout) minimiza problemas. Esse módulo também impõe restrições no sistema: precisamos ter pontos de verificação do plano do usuário – por exemplo, ao iniciar uma geração de vídeo, verificar se seu plano permite (não estourou limite de vídeos do mês). Essas regras de negócio permeiam outros módulos (e.g., Upload: plano básico tem 5GB, etc.). Então, a lógica de limites deve ser centralizada (talvez no módulo de pagamento/planos ou numa camada de middleware) para fácil manutenção. Do ponto de vista de dados, registraremos assinaturas, pagamentos, período de validade – informação sensível e deve ser consistente (provavelmente confiar no Stripe para verdade e armazenar só ref). Em caso de usuários enterprise com faturamento offline, possivelmente não via este módulo – mas fora de escopo inicial. Resumindo, o impacto é principalmente **controle de acesso e habilitação**: sem pagar, sem acesso completo. Deve ser sólido para garantir renda e não frustrar usuários legítimos.

### 6.8 Sistema de Autenticação com SSO Empresarial

**Descrição:** Módulo que lida com **login de usuários e integrações de Single Sign-On**. O que será implementado: toda a parte de cadastro/login convencional (usuário e senha com hash seguro, recuperação de senha via email) e adicionalmente o suporte a **logins federados**. Para empresas, permitir configuração de SSO via protocolos padrão (SAML 2.0 ou OIDC/OAuth2). Por exemplo, oferecer na interface de admin da empresa um modo de registrar seu provedor (carregar metadata XML do SAML, etc.). Implementar usando bibliotecas ou serviços como **Passport.js**, **NextAuth** ou diretamente integrações (Auth0 as broker). Quando configurado, usuários daquela empresa clicam em “Login com [Empresa]” e são redirecionados ao IdP para autenticar. Precisamos mapear os atributos retornados (pelo menos email) para contas na nossa base – possivelmente auto-provisionar o usuário na primeira vez (Just-In-Time provisioning). Além disso, suportar **social logins** (Google, Microsoft) para facilitar testes ou uso individual (embora foco seja enterprise). O módulo também cobre controle de sessão (cookies JWT), refresh de sessão, logout. Deve ter **MFA** opcional se via SSO (isso seria gerido pelo IdP corporativo). E roles/permissions básicas: por ex., flag de admin no usuário para acesso ao painel de gerenciamento da empresa.

**Impacto no Sistema:** Este módulo é central para **segurança e gerenciamento de acesso**. Uma falha aqui pode comprometer todos dados (ex: se autenticação for burlada). Portanto, usar padrões testados (bibliotecas) e realizar testes de penetração são necessários. O SSO enterprise é um fator de venda importante para clientes grandes – impacta a adoção (empresas querem que seus funcionários usem credenciais existentes). Mas também traz complexidade: cada IdP pode ter nuances, certificações, etc. Precisaremos robustez no parsing SAML, relógio sincronizado (para tokens), etc. Em termos de UX, um login SSO bem feito significa o usuário nem percebe nosso login – clica e entra direto se já estiver logado no domínio. Isso aumenta muito a conveniência para usuários corporativos. Por outro lado, ter login tradicional ainda é útil para pequenos clientes ou teste – então mantemos ambos. Impacto nas demais partes: o sistema de autorização (quem pode ver o quê) depende do usuário estar corretamente associado à empresa/conta. Precisamos garantir que, por exemplo, ao entrar via SSO, associamos ao tenant correto e aplicamos limites daquele tenant. Isso implica configurações do lado da empresa – esse módulo deve talvez incluir interface para administradores gerirem usuários (convidar por email, definir se só SSO ou permitir exceções com senha). Outro impacto é escalabilidade: autenticação em si não pesa muito, mas se integrando com SSO de muitos clientes, latência de IdPs externos e meta-dados caching importa. Provavelmente armazenaremos metadata SAML e certificados em nosso DB para cada conexão enterprise. Em suma, um módulo de complexidade **média-alta** dada integrações, mas de alto impacto porque **sem autenticação não há produto** – deve ser extremamente confiável e seguro.

### 6.9 Worker System com Fila de Render (Backend Assíncrono)

**Descrição:** Módulo que implementa a infraestrutura de **processamento em segundo plano** usando filas e workers. Já abordamos conceitualmente em arquitetura, mas detalhando aqui: O que será feito é configurar um **servidor Redis** e usar a biblioteca **BullMQ** no Node para criar uma ou mais filas (ex.: fila “render-video”). Desenvolver no backend funções produtoras (que adicionam jobs na fila com os dados necessários) e um serviço worker consumidor que executa os jobs. Cada job de render contém: referência ao projeto ou toda informação autônoma (slides, caminhos de arquivos, etc.). O worker, ao pegar o job, coordena os sub-módulos (como TTS, avatar, ffmpeg, detalhados acima) para realizar a tarefa. Precisaremos também de uma fila possivelmente para **pré-processamento** (ex: parser PPTX poderia ser enfileirado se for pesado) ou para tarefas como geração de thumbnails, etc., mas inicialmente foco no render principal. O módulo incluirá mecanismos de **retries** (config do BullMQ: número de tentativas e delays se falhar) e talvez divisão de tarefas (BullMQ suporta jobs filhos/fluxos[[16]](https://bullmq.io/#:~:text=Divide%20tasks%20into%20smaller%20ones) – podemos dividir por slide, e um job pai agregar, mas pode ser complexity extra). Também implementar notificação de eventos: BullMQ emite eventos de progresso, conclusão; o backend pode escutar e então atualizar status no DB ou notificar via WebSocket. Monitoramento da fila: talvez usar **Bull Board** (dashboard) para ver jobs, ou criar API para listagem do status dos jobs para admin.

**Impacto no Sistema:** Este módulo fornece a **coluna vertebral** para tarefas pesadas – sua estabilidade afeta diretamente a capacidade de gerar vídeos sem travamentos. Se a fila falhar (ex: Redis fora do ar), as gerações param. Então, alta disponibilidade de Redis e backup (RDB or AOF persist) é importante para não perder jobs. O módulo impacta também a **escalabilidade**: com ele, podemos adicionar mais processos de worker em outras máquinas e todos consumirem da fila, aumentando throughput – isso será fundamental se a demanda crescer. Em termos de desempenho, BullMQ alega suportar >100k jobs/segundo em Redis[[17]](https://bullmq.io/#:~:text=,Scale%20horizontally), então não será gargalo facilmente; o gargalo são os jobs em si (CPU/GPU). O impacto na complexidade de código: temos que lidar com jobs assíncronos, garantir idempotência (um job não rodar duplicado inadvertidamente), e cuidar do tamanho dos dados no job (não colocar dados enormes na queue – preferível passar IDs e pegar do DB/FS). O sistema de fila também traz latência mínima (pequeno overhead para puxar job), mas isso é compensado pela robustez. O impacto UX: usando fila, conseguimos dar feedback "Processando..." sem travar interface, e potencial para permitir múltiplos jobs por usuário (ex: gerar 2 vídeos diferentes em paralelo se plano permitir). Precisamos, porém, comunicar claramente estados – o usuário deve sempre saber se está em andamento, quantos % ou tempo estimado. Isso depende do módulo de fila prover info de progresso, que teremos que calcular (por exemplo, quantos slides concluídos de total, atualizar progresso a cada slide terminado). Em resumo, o worker system é um módulo essencial para cumprir a promessa de **automação escalável** – seu bom funcionamento garante que possamos **processar 24/7** em background e entregar resultados confiáveis, mesmo sob cargas variáveis, isolando a lógica pesada do resto do app.

### 6.10 Monitoramento, Logs e Alertas

**Descrição:** Módulo dedicado a **monitorar a saúde do sistema** e registrar eventos importantes. O que será feito: integrar ferramentas de monitoramento de aplicação e infraestrutura. Por exemplo, incorporar um agente como **Prometheus** scraping métricas custom no Node (via endpoint /metrics) para dados como número de jobs na fila, duração média de render, uso de CPU nos workers, etc. E visualização com **Grafana** ou outro. Configurar **logs estruturados** em todos os serviços: backend e workers enviando logs (JSON) para um storage central ou serviço (como CloudWatch Logs, Elastic Stack). Definir níveis de log (info, warn, error) e logar eventos-chave: upload realizado, início/fim de geração de vídeo tal, integrações chamadas, exceções stacktrace. Implementar também **alertas**: por exemplo, se fila acumular > X jobs ou tempo médio de job > Y, enviar alerta (email/Slack) à equipe; se taxa de erro de API externa passar limite, notificar. Pode usar integr. com PagerDuty ou similar para incidentes. Além disso, coletar métricas de negócio no monitoramento (número de vídeos gerados hoje, etc.) para fins de acompanhamento. Esse módulo também abriga possivelmente a parte de **analytics interno** (mas analytics de uso foi tratado separadamente; aqui foco em sistema). Outras ações: monitorar uso de disco, memória nos servidores, conexões DB, etc., com alertas para evitar downtime.

**Impacto no Sistema:** Embora invisível para o usuário final, esse módulo impacta fortemente a **manutenibilidade e confiabilidade** do produto. Um bom monitoramento permite detectar problemas antes que afetem usuários (ex: se um worker travar em loop, a fila crescerá e um alerta dispara antes dos clientes reclamarem de demora). O logging detalhado facilita depuração de bugs – ex: se um vídeo saiu com falha, olhar logs do job para ver qual passo deu problema. Isso reduz o tempo de resolução de problemas e aumenta confiança na plataforma. Em termos de performance, adicionar monitoramento adiciona leve overhead (expor métricas, enviar logs), mas nada comparado ao resto; devemos apenas garantir que logging excessivo não loteie IO. O impacto na equipe de desenvolvimento/operações é grande: precisam manter as painéis de monit. e reagir aos alertas (processo devops). Do lado do usuário, eles não interagem com isso, mas se bem feito resulta em menos downtime e melhor qualidade de serviço (SLA). Também podemos expor algumas métricas aos clientes enterprise (ex: status das integrações, tempo médio de geração) se fizer sentido, mas não prioritário. No design do monitoramento, importante cobrir todos módulos críticos: fila, API, autenticação, uso CPU/GPU, latência de dependências. Isso alimenta decisões de scaling (ex: se CPU dos workers vive em 90%, hora de aumentar instâncias). Em resumo, é o módulo que **garante operação confiável em produção**, e sem ele seria arriscado escalar usuários. É quase obrigatório para um sistema complexo como este.

### 6.11 CDN e Distribuição de Conteúdo

**Descrição:** Módulo que lida com a **entrega dos vídeos e assets de forma otimizada** via Content Delivery Network. O que será implementado: configurar um **CDN (Content Delivery Network)** para cachear e servir os arquivos estáticos pesados, principalmente os vídeos MP4/WebM gerados, mas também possivelmente thumbnails e assets de mídia. Se usarmos AWS, por exemplo, criar uma distribuição **CloudFront** apontando para o bucket S3 onde vídeos são armazenados. Definir políticas de cache (vídeos podem ter cache longo, pois não mudam, a não ser que se regenere e mude URL). Implementar também invalidações caso um vídeo seja deletado ou atualizado (ou usar versionamento nos nomes para evitar isso). Na aplicação, garantir que as URLs fornecidas para download ou embed sejam as do CDN, não diretamente do storage. Adicionalmente, habilitar entrega adaptativa se for possível – por ex, entregar via HTTP/2, TLS, e talvez usar **HLS/DASH** no futuro para streaming adaptativo (embora inicialmente só arquivos progressivos). Também cuidar para restringir acesso: talvez configurar CDN + S3 para só permitir acesso com token (se quisermos que vídeo só seja visto logado) – ou fornecer apenas para públicos aberto se não confidencial. Possivelmente integrar com a aplicação algum player que use a CDN (ex: se fornecermos embed code).

**Impacto no Sistema:** Usar uma CDN impacta **positivamente a performance** para usuários finais: diminui latência e acelera carregamento, especialmente se a empresa tiver pessoas em várias regiões. Isso melhora a experiência de quem assiste aos vídeos (o que é crucial se os vídeos são o produto final consumido por funcionários ou público). Impacta também custos: CDNs têm custo extra, mas servem conteúdo a menor custo por GB que puxar tudo do servidor ou storage de origem, além de poupar carga de origem. Precisamos monitorar custos e talvez impor limites (ex: se um vídeo viraliza externamente, garantir que está dentro do esperado, ou colocar proteção). Em termos de segurança, se alguns vídeos forem internos apenas, talvez não queira expô-los em um CDN público – mas CDNs como CloudFront permitem restrição por assinatura de URL; implementaremos se necessário para que apenas quem tem permissão acesse. Impacto no backend: praticamente nenhum direto, além de gerar URLs apropriadas. Em integração, ao salvar o arquivo final, já podemos compor a URL CDN (que pode ser simplesmente substituir domínio do S3 pelo do CloudFront). Assim, o frontend já consome do CDN. Este módulo também possibilita escalabilidade – mesmo se muitos usuários baixarem vídeos simultaneamente, a CDN distribui a carga; sem ela, o bucket ou servidor poderia virar gargalo. Em suma, um módulo essencial para garantir **distribuição rápida e escalável do conteúdo** gerado, afetando diretamente a percepção de qualidade (vídeos carregando sem engasgos, etc.). Ele trabalha em conjunto com Monitoramento, pois métricas de CDN (hit rate, usage) devem ser acompanhadas.

### 6.12 Analytics e Métricas Integradas

**Descrição:** Módulo que implementa **coleta e apresentação de métricas de uso** da plataforma e dos vídeos. O que será feito: no âmbito do produto, instrumentar ações do usuário para analytics – ex: usar uma ferramenta como **Amplitude ou Google Analytics** para eventos dentro da app (projetos criados, clique em gerar vídeo, etc.) a fim de entender engajamento. Mas principalmente, desenvolver um painel para o usuário (criador) ver métricas dos vídeos produzidos. Isso envolve coletar dados de visualização: se o vídeo é assistido na plataforma (player custom), implementaremos tracking de play, pause, completo. Poderemos armazenar no banco contadores de visualização por vídeo e possivelmente logs por usuário (no caso de integração com login, mas se público talvez só totais). Também podemos integrar com LMS via SCORM/xAPI: neste caso, ao ser visualizado em LMS externo, as estatísticas ficam lá – poderíamos ler se API permitir. Mas num escopo inicial, focamos em métricas simples e exibimos na interface do produto. O painel de analytics de vídeo pode mostrar: número de vezes que cada vídeo foi visto (total e único), tempo médio assistido (% do vídeo), e distribuição geográfica ou por dispositivo se coletarmos (via GA embed, por ex.). Também, se multi-idioma for usado, quantos em cada língua etc. No lado admin interno, analytics agregada: total de vídeos gerados na plataforma, uso por cliente, etc., para insights de negócio e para limitar misuse. Implementar possivelmente exportação dos dados (CSV) se clientes quiserem integrar a relatórios internos.

**Impacto no Sistema:** Embora não afete a geração em si, este módulo agrega valor para medir eficácia dos vídeos – importante para empresas que querem saber se seus funcionários assistiram aos treinamentos. Impacta a base de dados – teremos tabelas de eventos de visualização que podem crescer muito se vídeos forem muito assistidos. Precisaremos de estratégias de agregação ou limpeza (ou usar uma ferramenta analítica externa). A coleta de analytics no player deve ser cuidadosa para não introduzir muita carga (talvez ping progress a cada X segundos – mas isso gera muitos eventos). Podemos simplificar: contar uma visualização quando atingido 80% do vídeo, etc., para reduzir granularidade. Em termos de privacidade, se for uso interno somente, não há problema; mas se usuários externos assistem, seguir LGPD – talvez avisar que “este vídeo monitora seu progresso” dependendo do caso. Impacto de UX: para o criador, ter essas métricas na plataforma poupa trabalho de usar outra ferramenta (ex: se fosse no YouTube, já teria analytics do YouTube; aqui fornecemos nosso). Isso torna o produto mais completo e **demonstrar ROI** (ex: "95% dos funcionários viram o vídeo até o final"). Internamente, analytics de uso (quantos projetos, etc.) ajuda produto a priorizar melhorias. Na implementação, deve ser leve para não afetar playback (evitar que tracking cause travamento do vídeo se falhar). Poderíamos usar *beacon* async ou background. Em resumo, esse módulo impacta **visibilidade e avaliação de sucesso** dos vídeos e da plataforma, tanto para clientes quanto para nós, e deve ser feito de forma escalável para possivelmente milhões de eventos se a base de usuários for grande.

## 7. Roadmap de Entregas

A implementação do Estúdio IA de Vídeos será organizada em **sprints** seguindo metodologias ágeis (Scrum), priorizando primeiro os componentes centrais (**Engine de Renderização** e **Avatares 3D**) e depois agregando funcionalidades de suporte e melhorias. Abaixo está um roadmap sugerido, dividindo o desenvolvimento em fases lógicas:

### 7.1 Sprint 1: Fundação – Renderização Básica e Avatar 3D Inicial

**Objetivo:** Desenvolver o núcleo do sistema de geração de vídeos com um avatar simples. Nesta etapa, focamos em conseguir importar um PPTX, gerar áudio TTS e **produzir um vídeo MP4** com um avatar (mesmo que rudimentar) apresentando um slide. - Implementar o **Worker de render** inicial com fila: configurar BullMQ + Redis e uma rotina simples que pega um texto fixo e gera um vídeo de teste (por exemplo, tela preta com áudio narrado) para validar o pipeline. - Integrar um serviço de **TTS** (ex: Azure Speech) e testar geração de áudio via código. - Integrar um avatar 3D básico: pode ser um vídeo dummy ou usar uma solução pronta (ex: API do **D-ID** ou um avatar do Audio2Face com modelo demo) para obter um pequeno vídeo do avatar falando o texto de teste. - Combinar áudio + visual avatar + texto do slide em vídeo via FFmpeg (começar com slide como imagem de fundo fixa e avatar picture-in-picture). - Entregar um **protótipo funcional**: dado um slide com uma frase, retornar um MP4 com avatar falando essa frase. - **Impacto/Resultado:** Ao fim do Sprint 1, teremos validado a viabilidade técnica principal (gerar vídeo com fala automatizada e avatar). Pode ser tudo hard-coded ou muito simples, mas reduz riscos das tecnologias núcleo (FFmpeg, TTS, avatar pipeline). Isso servirá de base para os próximos sprints.

### 7.2 Sprint 2: Importação de PPTX e Narrativa Multislides

**Objetivo:** Permitir que usuários importem suas apresentações e gerar vídeos multi-slide sequenciais com narração contínua. - Desenvolver o módulo de **Importação de PPTX**: implementar upload de arquivo, parse dos slides (usar biblioteca ou serviço). Focar em extrair textos dos slides. - Criar estrutura de **Projeto** no backend para armazenar slides importados e seu conteúdo. - Aprimorar o **TTS**: em vez de texto fixo, integrar com os textos extraídos de cada slide. Gerar áudio para cada slide. - Extender o **worker de render** para iterar pelos slides: compor cada slide com seu áudio e concatenar no vídeo final. Implementar transição simples (corte ou fade). - Melhorar a apresentação do avatar: configurar para cada trecho de áudio gerar o avatar correspondente (pode ser o mesmo avatar para todos slides, repetindo o processo). - **Interface mínima**: criar um front-end básico onde o usuário pode subir um PPT e clicar “Gerar vídeo”, depois fazer download. Não precisa editor ainda, mas mostrar lista de slides talvez. - **Impacto/Resultado:** Ao final deste sprint, um usuário poderá fazer o fluxo completo rudimentar: fornecer um PPTX de múltiplos slides e receber um vídeo narrado passando por todos os slides com um avatar apresentando. Isso valida a integração de importação e multi-cena. Provavelmente neste ponto, a qualidade do avatar ainda básica e não há ajustes finos, mas funcionalidade end-to-end existe.

### 7.3 Sprint 3: Timeline de Edição e Refinamento de Avatar

**Objetivo:** Introduzir a capacidade de edição e melhorar qualidade visual dos avatares. - Desenvolver o **Editor Timeline** (versão inicial): implementar no front-end uma representação dos slides na timeline. Permitir reordenar slides e ajustar duração de forma simples (por exemplo, input manual de segundos ou slider). - Conectar as alterações da timeline ao backend: se usuário muda ordem ou duração, refletir esses dados antes de re-renderizar. - Integrar **avatar 3D mais avançado**: por exemplo, incorporar ReadyPlayerMe avatars via Three.js no browser para pré-visualização, ou melhorar o pipeline do Audio2Face para gerar expressões mais naturais. Possivelmente permitir escolha entre 2-3 avatares. - Adicionar **gestos ou movimentos** no avatar: talvez selecionar se avatar aparece do lado ou full-screen em algum slide (configurável por slide). - Melhorar composição visual no vídeo: definir layout final (posição do avatar e slide), adicionar background se necessário. - **Teste de UX:** Fazer teste interno de usar timeline para ajustar tempos e ver se reflete corretamente no output. - **Impacto/Resultado:** Nesse sprint, o usuário passa a ter controle sobre o vídeo gerado. Pode arrumar tempos se alguma narração for muito rápida/lenta para o conteúdo, e garantir sincronia desejada. O avatar também estará com aparência/expressões melhores, aumentando profissionalismo do vídeo. Essa iteração foca em polir a experiência de edição e visual.

### 7.4 Sprint 4: Infraestrutura de Suporte – Autenticação, Assets e Pagamentos

**Objetivo:** Implementar os módulos de suporte ao produto para prepará-lo para uso real: autenticação robusta, gerenciamento de arquivos e sistema de pagamento. - **Autenticação & Contas:** Integrar **NextAuth** (ou similar) para login de usuários. Criar páginas de login, registro (para pequenos clientes) e inserir autenticação nas rotas protegidas. Iniciar configuração de **SSO** (talvez implementar OIDC com Google para teste, e esboçar SAML). - **Gerenciamento de Assets:** Desenvolver a biblioteca de mídia no front: permitir upload de imagem (ex.: logo da empresa) e inserção em um slide (UI para o usuário colocar a imagem em um slide importado). Salvar esses assets no S3, criar referência no projeto. - **Sistema de Pagamentos:** Integrar **Stripe** em modo de teste. Criar uma página de preços com planos (dummy), permitir checkout do plano básico. Restringir certas funções (talvez limite de duração) para free vs pago para testar. - **Controle de Limites:** Implementar backend checks para coisas como: usuários não autenticados não geram vídeo; se free trial, limitar 1 vídeo de X min. Preparar estrutura para mais limites por plano. - **Impacto/Resultado:** Ao fim do Sprint 4, o produto estará mais próximo de um SaaS completo: usuários podem registrar/entrar de forma segura, a mídia adicional pode ser gerida (embora simples), e teremos a base para cobrar pelo serviço. Isso é importante para começar pilotos controlados. Embora nem todos limites ou SSO enterprise estejam finalizados, o arcabouço estará pronto. Esse sprint aborda muitas áreas transversais, pode se dividir em sub-sprints se necessário.

### 7.5 Sprint 5: Aprimoramentos de Performance e Monitoramento

**Objetivo:** Antes de lançar beta, otimizar performance, escalabilidade e introduzir monitoramento para estabilidade. - **Otimização do Render:** Analisar pontos lentos – ex: paralelizar geração de avatar por slide se possível, usar cache para não reprocessar slides sem mudança. Ajustar parâmetros do FFmpeg para melhor velocidade (ex: presets ultrafast para teste). - **Escalabilidade do Worker:** Testar colocando 2+ workers processando fila, verificar condições de concorrência (ex: vários jobs simultâneos, Redis lock). - **Monitoramento:** Configurar log central (pode usar um serviço gerenciado tipo Sentry para erros front/back). Adicionar métricas Prometheus e montar alguns gráficos (mesmo simples) para fila de jobs. - **CDN:** Configurar CDN para servir vídeos gerados. Testar entrega de um vídeo via CloudFront e checar latência. Integrar o link CDN no app. - **Analytics Básico:** Inserir Google Analytics no front para page views e eventos principais (ex: criação de projeto). - **Testes de Carga:** Simular geração de, por ex, 10 vídeos simultâneos e observar se sistema aguenta ou se precisa tunar (aumentar máquinas, etc.). - **Impacto/Resultado:** Após este sprint, o sistema deve estar **estável e mais rápido**. Com monitoramento, teremos visibilidade para manutenção. A entrega de vídeos via CDN garante boa experiência a quem for assistir. Otimizações reduzirão tempo de espera por vídeo e permitirão mais usuários simultâneos. Em suma, o produto estará tecnicamente preparado para um lançamento beta controlado, com ferramentas para observar e reagir a problemas.

### 7.6 Sprint 6: Recursos Avançados e Feedback Beta

**Objetivo:** Implementar funcionalidades adicionais de valor agregado e incorporar feedback inicial dos beta testers. - **Analytics de Vídeo:** Desenvolver o painel de analytics para o usuário – mostrando quantas vezes cada vídeo foi visto. Isso envolve tracking no player: implementar eventos onPlay, onComplete e salvar contagem. - **Legendas Automáticas:** Utilizar o texto dos slides para gerar legendas no vídeo (SRT ou gravadas). Permitir ao usuário baixar legendas ou habilitar no player. - **Melhorias de Avatar:** Se feedback indicar necessidade, adicionar mais avatares (diversidade étnica, trajes formais vs casuais) e permitir customizações simples (ex: escolher cor da roupa ou fundo). - **SSO Enterprise:** Finalizar suporte SAML/OpenID for Enterprise logins, documentar como clientes podem configurar (talvez preparar guia). - **Polir UI/UX:** Refinar elementos visuais, adicionar tutoriais in-app (ex: dicas orientando a usar timeline, importar PPT correto). Garantir responsividade do layout para diferentes resoluções. - **Feedback Loop:** Reunir feedback de usuários beta (dificuldades, bugs) e corrigir os principais. Ex: se parser PPT falha em certos arquivos, melhorar. Se geração de voz tem latência alta, talvez adicionar um loading indicativo por slide. - **Impacto/Resultado:** Este sprint consolida o produto com recursos complementares (ex: analytics demonstrando eficácia) e garante que a experiência esteja refinada conforme retorno de usuários reais. Ao final, o **Estúdio IA de Vídeos** estará pronto para um lançamento geral (v1.0) com confiança, cobrindo tanto as funções essenciais quanto detalhes de apresentação e usabilidade.

O roadmap acima pode ser ajustado conforme prioridades de negócio ou descobertas técnicas. No entanto, a ordem proposta – primeiro garantir o core (render/video/avatar), depois adicionar editabilidade, seguido por infraestrutura (auth/pagamento) e otimização/monitoramento – busca mitigar riscos cedo e entregar valor incremental. Iniciar pelo Engine de Renderização e Avatar 3D (Sprint 1 e 2) é fundamental, pois são os elementos diferenciadores do produto e com maior desafio técnico. Subsequentemente, as camadas de suporte e melhoria vão transformando o protótipo em um produto completo pronto para o mercado.

## 8. Anexos Técnicos

Nesta seção complementar, fornecemos detalhes técnicos, recomendações e checklists para assegurar a qualidade em aspectos específicos do sistema, tais como realismo dos avatares, sincronização labial e integrações externas de IA e infraestrutura.

### 8.1 Checklist de Hiper-Realismo de Avatares

Para garantir que os avatares 3D atinjam um nível **hiper-realista** convincente, especialmente ao apresentar conteúdo corporativo, seguem critérios a serem observados: - **Modelagem e Texturas:** Utilizar modelos de alta definição (ex: **MetaHuman** da Epic) que possuem malha detalhada do rosto e corpo, além de texturas 4K realistas para pele, cabelo e olhos. Verificar que os materiais reagem à luz de forma natural (sombras, reflexos nos olhos). - **Expressões Faciais Dinâmicas:** Implementar um conjunto amplo de **blendshapes/visemas** cobrindo fonemas essenciais (AO, AI, E, etc.) e também expressões emocionais leves (sobrancelhas movendo, sutil sorriso) para evitar aspecto inerte. O avatar deve transicionar suavemente entre expressões conforme o contexto da fala. - **Movimentação Corporal:** Incluir animações corporais e gestos predefinidos durante a apresentação: por exemplo, leve movimento de mãos ao enfatizar um ponto, inclinar a cabeça, mudar peso de um pé para outro para simular postura viva[[8]](https://www.aistudios.com/pt/ai-avatars#:~:text=Movimentos%20naturais%20do%20corpo). Garantir que os movimentos sejam moderados (evitar exagero que distraia) e condizentes com o tom da narração. - **Sincronização Labial Perfeita:** A sincronização lábio-fala deve ser precisa quadro a quadro[[7]](https://www.aistudios.com/pt/ai-avatars#:~:text=Os%20avatares%20de%20IA%20mais,realistas%20do%20mundo). Realizar testes em câmera lenta para ajustar qualquer offset entre áudio e movimento dos lábios. A boca do avatar deve abrir nas vogais e tocar lábios/dentes nas consoantes adequadamente. Se houver desvio de mais de ~100ms, o olho humano percebe – manter abaixo disso. - **Olhar e Contato Visual:** Programar o avatar para **manter contato visual** com a “câmera” (público) a maior parte do tempo. Piscar os olhos em intervalos naturais (a cada 5-10 seg) e ocasionalmente desviar o olhar brevemente para simular cognição. Isso evita o olhar fixo e vazio. - **Iluminação e Cenário:** Renderizar o avatar com iluminação profissional: três-pontos (frontal, preenchimento e contraluz) ou usar HDRI environment para iluminação global realista. O fundo pode ser neutro (ex: escritório desfocado) para contexto, mas assegure contraste suficiente para destacar o avatar. - **Taxa de Quadros Alta:** Gerar animações do avatar em pelo menos **30 FPS**, idealmente 60 FPS, para capturar detalhes sutis de movimento e evitar *choppiness*. Avatares fluidos parecem mais reais. - **Verificação de Uncanny Valley:** Realizar avaliações subjetivas internas – se o avatar parecer inquietante ou artificial, identificar o porquê (olhos sem vida? movimentos rígidos?) e iterar modelagem/anim. Incluir pessoas de fora do dev para opinar se o avatar convence como apresentador. - **Variação de Avatares:** Fornecer opções de avatars diversos (gênero, idade, traços étnicos) todos com qualidade realista semelhante, para que empresas escolham representantes que se adequem à sua imagem. Todos devem atender aos critérios acima.

Esse checklist deve orientar tanto a escolha de tecnologia (por ex: optar por MetaHuman se possível, que já atende muitos itens) quanto a configuração dos parâmetros de animação. O resultado final esperado é um avatar que, em um rápido olhar, passe por um humano verdadeiro em vídeo para a maioria dos espectadores – aumentando assim a aceitação e impacto dos vídeos gerados.

### 8.2 Especificações para Sincronização Labial (Lipsync)

A sincronização labial entre áudio e movimentos do avatar é tecnicamente complexa. Estabelecemos algumas especificações para garantir qualidade: - **Mapeamento de Visemas:** Utilizar um mapa de pelo menos **15 visemas** padrão (AI, O, U, EE, etc.) alinhados aos fonemas do áudio. Ferramentas como Oculus Lipsync fornecem esse mapeamento pronto para modelos ReadyPlayerMe[[14]](https://forum.readyplayer.me/t/lipsync-avatar-with-recorded-voice/314#:~:text=Lipsync%20avatar%20with%20recorded%20voice,synced%20with%20the%20recorded%20audio). Confirmar que o rig facial do avatar possui blendshapes correspondentes para cada visema requerido. - **Taxa de Amostragem do Áudio:** Processar o áudio a 22 kHz ou superior e executar análise de fonemas em segmentos de ~10ms. A latência de detecção de fonema deve ser mínima para refletir imediatamente no movimento. - **Antecipação de Movimentos:** Iniciar a forma labial levemente **antes** do som correspondente (por poucos milissegundos) para compensar a leve diferença de percepção visual vs auditiva. Isso ajuda a dar sensação de sincronia perfeita. - **Ferramenta de Lipsync Automática:** Avaliar uso do **NVIDIA Audio2Face** ou similares que geram animação labial via IA. O Audio2Face pode produzir peso de blendshapes em tempo real a partir do áudio. Caso usado, calibrar sua saída com ajustes manuais se necessário – às vezes IA comete erros em fonemas curtos, então aplicar pós-processamento suave (ex: filtro para não abrir boca em sons silenciosos). - **Verificação Quadro a Quadro:** Como QA, para cada nova voz integrada, rodar um trecho de fala e revisar quadro-a-quadro se a forma da boca condiz com o som. Especialmente checar **oclusivas** (P/B/M – lábios fecham completamente; T/D – língua toca dentes, difícil ver mas lábios meio abertos; F/V – lábio superior toca inferior). Ajustar keyframes se alguma está errada. - **Coarticulação Suave:** Implementar interpolações suaves entre visemas para evitar movimentos bruscos. A boca não deve “saltar” de forma; transição linear ou cúbica entre visemas adjacentes no timeline do áudio. - **Sincronização da Linha de Voz:** Garantir que o avatar recebe exatamente o mesmo áudio final que será ouvido no vídeo, sem edições divergentes. Se o FFmpeg faz junção de áudio posteriormente, evitar descompasso. Uma forma: gerar animação lipsync por slide e depois juntar, assim cada segmento trata seu áudio local. - **Teste Multilíngue:** Testar lipsync em pelo menos os idiomas alvo principais (português, inglês, espanhol). Diferentes línguas têm cadências e fonemas distintos (ex: vogais nasais no português) – garantir que o motor de lipsync trata corretamente (pode precisar treinar modelos específicos ou ajustar visemas). - **Fallback Visual:** Caso pequeno desvio ocorra, mitigar com câmera não muito próxima do rosto – distância ou resolução ligeiramente menor esconde pequenos defeitos de lipsync. Contudo, isso é contorno; alvo é não precisar disso.

Cumprindo essas especificações, espera-se atingir um nível onde a fala do avatar **parece natural**. Em avaliações internas, pessoas não devem notar que é sincronização artificial. A fala deve “casar” com o movimento ao ponto de quem assiste focar no conteúdo e esquecer que é um avatar digital.

### 8.3 Recomendações de Integração com Plataformas Externas

O Estúdio IA de Vídeos se apoia em diversas plataformas e serviços externos para oferecer funcionalidade avançada. Abaixo, recomendações para integrar alguns dos principais:

* **Integração com ElevenLabs (Voz IA):** A ElevenLabs oferece API de TTS com vozes extremamente naturais. Recomenda-se utilizar a **API de voz generativa** deles para narradores principais. Obter uma **API Key** e usar endpoints de *Text-to-Speech* fornecendo o texto e voz ID (a ElevenLabs tem vozes pré-treinadas e possibilita vozes custom). Atentar para limites de rate (padrão ~1000 caracteres/mês no trial) – necessário plano pago robusto para uso contínuo. Implementar caching local dos resultados: a mesma frase não precisa ser gerada duas vezes. A latência da ElevenLabs é geralmente baixa (1-2 seg por frase curta). Monitorar a resposta e em caso de falha (500s) re-tentar ou cair para um TTS alternativo. **Nota:** A ElevenLabs cobra por uso, então mapear custos por caractere e talvez limitar tamanho de texto narrado ou avisar usuário de consumo.
* **Integração com Microsoft Azure Speech:** Azure Cognitive Services (Speech) oferece TTS neural em mais de 100 idiomas e permite customização por SSML. Configurar no Azure Portal um recurso Speech e obter **Key e Endpoint**. A integração via SDK ou REST requires passar texto e um voice name (ex: pt-BR-AntonioNeural). O Azure é útil para vozes em idiomas que ElevenLabs não suporte e tem forte infraestrutura (alta disponibilidade). A latência pode ser maior que ElevenLabs dependendo da região (pode colocar recurso na região mais próxima do servidor). Suporta long audio via *Long Audio API* caso necessário. Recomenda-se usar Azure para línguas diversas e para clientes enterprise Microsoft-heavy (facilita compliance). Gerenciar as credentials de forma segura (Key Vault). Também considerar o uso do **Custom Neural Voice** no Azure se cliente quiser voz exclusiva (processo que requer amostras e aprovação Microsoft).
* **Integração com AWS S3 (Storage):** Usaremos S3 para armazenar apresentações, assets e vídeos. Criar um bucket dedicado, com políticas de acesso restritas. No backend, utilizar SDK AWS (ou biblioteca como **Minio** se quiser abstracao) para upload/download. Habilitar **versão** nos objetos ou nomear outputs com UUID para evitar conflito em caso de reprocessamento. Para desempenho, usar upload multi-part para arquivos grandes (vídeos) e talvez URLs pré-assinadas para o front enviar diretamente (reduzendo uso do servidor backend). Configurar ciclo de vida do bucket – por exemplo, mover vídeos muito antigos para storage mais barato ou deletar se não mais necessários (conforme política do cliente). Ativar **CloudFront CDN** na frente do bucket para distribuição global, e usar URLs CloudFront no app. Para segurança, usar buckets privados + CloudFront com origin access identity (ou gerar signed URLs quando necessário). Testar throughput e considerar limitar banda por usuário se preciso (S3 suporta escalabilidade alta por padrão). Também, planejar backup ou replicação do bucket se for conteúdo crítico.
* **Integração com Services de Avatar (Opcional):** Se optar por serviços como **D-ID (Creative Reality)** para gerar vídeo com avatar a partir de texto, é possível usar sua API: enviar texto e escolher um apresentador, ela retorna um vídeo pronto. Isso simplifica nossa pipeline de avatar mas terceiriza custo e controle. Caso testado, verificar tempo de resposta (um vídeo de 1 min pode levar alguns min no D-ID). Custos podem ser por minuto gerado. Alternativamente, integração com **Epic MetaHuman Animator**: requer rodar Unreal Engine, possivelmente não via API mas manual. Uma via é usar **MetaHuman via Unreal Cloud** (se disponível no futuro). Por ora, se integrarmos ReadyPlayerMe, utilizar seu avatar generator (REST para criar avatar a partir de foto ou escolha) e depois animar no nosso sistema. Em suma, selecionar um serviço de avatar completo pode acelerar o go-to-market mas considerar lock-in e preço.
* **Integração com LMS (SCORM/xAPI):** Para empresas que queiram rastrear treinamento em seus LMS, podemos oferecer exportação SCORM. Isso envolve embutir o vídeo num pacotes SCORM com um HTML player + runtime SCORM que notifica conclusão quando vídeo chega ao fim. Usar biblioteca como **SCORM Wrapper**. Alternativamente, adotar **xAPI (Tin Can)**: emitir um statement "Usuario X assistiu Video Y 100%". Isso requer configuração do LRS endpoint do cliente. Recomendamos inicialmente SCORM 1.2 básico (marcar completado) por simplicidade, se demandado.
* **Integração com Identity Providers (SSO):** Para SSO, muitos clientes usarão Azure AD, Okta, Auth0, ADFS. Podemos simplificar via **Auth0** Enterprise SSO connections, porém isso implica custo. Implementação própria: usar biblioteca **passport-saml** ou **SAMLify** para SAML, e **openid-client** para OIDC. Testar com Azure AD (SAML) pois é comum: registrar nosso app there, obter metadata, configurar ACS (Assertion Consumer) URL em nosso app (ex: /auth/saml/callback). Precisaremos fornecer metadata XML do nosso lado para cliente. Documentar passo a passo para o cliente setar trust. Similar para Okta (they provide metadata). Recomenda-se robustez em handling: relógio NTP sync (SAML é time-sensitive), e logs detalhados se falha autenticação SSO (para suporte).

Em geral, cada integração externa deve ser modular, para poder ser substituída ou complementada conforme necessidade (por exemplo, se ElevenLabs estiver indisponível, usar Azure, etc.). Manter chaves e endpoints em configurações seguras e não hard-coded. Monitorar uso de cada serviço para evitar surpresas de custo ou limites atingidos.

**Observação Final:** Como se trata de um produto de ponta envolvendo IA e processamento multimídia, a arquitetura e requisitos aqui descritos devem evoluir conforme novas tecnologias surjam (por exemplo, avanços em geração de vídeo por IA, novos players no mercado de avatares virtuais, etc.). O PRD, entretanto, estabelece uma base sólida e realista para o desenvolvimento do **Estúdio IA de Vídeos**, almejando uma ferramenta inovadora que combine automação e criatividade para revolucionar a criação de vídeos corporativos.

[[1]](https://www.synthesia.io/powerpoint-to-video#:~:text=How%20can%20I%20make%20a,video%20presentation%20with%20PowerPoint) [[2]](https://www.synthesia.io/powerpoint-to-video#:~:text=How%20do%20I%20convert%20a,PowerPoint%20to%20video) [[6]](https://www.synthesia.io/powerpoint-to-video#:~:text=match%20at%20L364%20,easy%20to%20use%20as%20PowerPoint) PPT to Video Converter | Convert PowerPoint to Video

<https://www.synthesia.io/powerpoint-to-video>

[[3]](https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD#:~:text=,realistas%20criam%20v%C3%ADdeos%20envolventes%20e) [[4]](https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD#:~:text=,interativos%20que%20cativam%20os%20aprendizes) [[5]](https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD#:~:text=mesmo%20para%20iniciantes.%20,interativos%20que%20cativam%20os%20aprendizes) Como Criar Vídeos de Treinamento com IA em Minutos | Speechify

<https://speechify.com/pt-br/blog/how-to-create-ai-training-videos-in-minutes/?srsltid=AfmBOoqYx3AbJdWOcCVfJ1f9v7ACznHo5TZx15Sim4nVRMtNfkMmjHJD>

[[7]](https://www.aistudios.com/pt/ai-avatars#:~:text=Os%20avatares%20de%20IA%20mais,realistas%20do%20mundo) [[8]](https://www.aistudios.com/pt/ai-avatars#:~:text=Movimentos%20naturais%20do%20corpo) Melhores avatares de IA | Crie vídeos de IA com avatares realistas

<https://www.aistudios.com/pt/ai-avatars>

[[9]](https://bullmq.io/#:~:text=BullMQ%20has%20been%20succesfully%20used,other%20types%20of%20background%20jobs) [[10]](https://bullmq.io/#:~:text=jobs%20and%20sending%20messages%20using,process%20jobs%20across%20multiple%20servers) [[11]](https://bullmq.io/#:~:text=Why%20Message%20Queues%3F) [[12]](https://bullmq.io/#:~:text=Offload%20long%20running%20tasks) [[16]](https://bullmq.io/#:~:text=Divide%20tasks%20into%20smaller%20ones) [[17]](https://bullmq.io/#:~:text=,Scale%20horizontally) BullMQ - Background Jobs processing and message queue for NodeJS | BullMQ

<https://bullmq.io/>

[[13]](https://elevenlabs.io/developers#:~:text=Quickly%20generate%20AI%20voices%20in,integrate%20ElevenLabs%20into%20your%20application) The most powerful AI audio API and detailed documentation

<https://elevenlabs.io/developers>

[[14]](https://forum.readyplayer.me/t/lipsync-avatar-with-recorded-voice/314#:~:text=Lipsync%20avatar%20with%20recorded%20voice,synced%20with%20the%20recorded%20audio) Lipsync avatar with recorded voice - Unity - Ready Player Me Forums

<https://forum.readyplayer.me/t/lipsync-avatar-with-recorded-voice/314>

[[15]](https://docs.readyplayer.me/ready-player-me/api-reference/avatars/morph-targets/oculus-ovr-libsync#:~:text=Oculus%20OVR%20LipSync%20,supports%20the%20Oculus%20Lipsync%20API) Oculus OVR LipSync - Ready Player Me

<https://docs.readyplayer.me/ready-player-me/api-reference/avatars/morph-targets/oculus-ovr-libsync>