UNIVERSIDADE DA INTEGRAÇÃO INTERNACIONAL DA LUSOFONIA AFRO-BRASILEIRA
INSTITUTO DE ENGENHARIAS E DESENVOLVIMENTO SUSTENTÁVEL
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FRANCISCO CLEITON DA SILVA NASCIMENTO

AUXILIUM MESTRE: DESENVOLVIMENTO E AVALIAÇÃO DE UMA APLICAÇÃO WEB FULL-STACK PARA GERAÇÃO DE CONTEÚDO EDUCACIONAL BASEADA EM MODELOS DE LINGUAGEM DE LARGA ESCALA (LLMs)

REDENÇÃO - CE
2025

FRANCISCO CLEITON DA SILVA NASCIMENTO

AUXILIUM MESTRE: DESENVOLVIMENTO E AVALIAÇÃO DE UMA APLICAÇÃO WEB FULL-STACK PARA GERAÇÃO DE CONTEÚDO EDUCACIONAL BASEADA EM MODELOS DE LINGUAGEM DE LARGA ESCALA (LLMs)

Trabalho de conclusão de curso apresentado ao curso de Engenharia de Computação da Universidade da Integração Internacional da Lusofonia Afro- Brasileira como requisito parcial à obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Allberson Bruno de Oliveira Dantas

REDENÇÃO - CE
2025

Universidade da Integração Internacional da Lusofonia Afro-Brasileira Sistema de Bibliotecas da UNILAB
Catalogação de Publicação na Fonte.

Nascimento, Francisco Cleiton da Silva.

N244a

Auxilium mestre: desenvolvimento e avaliação de uma aplicação Web Full-Stack para geração de conteúdo educacional baseada em Modelos de Linguagem de Larga Escala LLMs / Francisco Cleiton da Silva Nascimento. - Redenção, 2025.
67f: il.

Monografia - Curso de Engenharia De Computação, Instituto de Engenharias e Desenvolvimento Sustentável, Universidade da Integração Internacional da Lusofonia Afro-Brasileira, Redenção, 2025.

Orientador: Prof. Dr. Allberson Bruno de Oliveira Dantas.

1. inteligência artificial. 2. Geração de conteúdo. 3. Modelos de Linguagem de Larga Escala LLMs. 4. FastAPI. 5. Engenharia de Prompt. I. Título

CE/UF/BSCA CDD 006.3

FRANCISCO CLEITON DA SILVA NASCIMENTO

AUXILIUM MESTRE: DESENVOLVIMENTO E AVALIAÇÃO DE UMA APLICAÇÃO WEB FULL-STACK PARA GERAÇÃO DE CONTEÚDO EDUCACIONAL BASEADA EM MODELOS DE LINGUAGEM DE LARGA ESCALA (LLMs)

Trabalho de conclusão de curso apresentado ao curso de Engenharia de Computação da Universidade da Integração Internacional da Lusofonia Afro- Brasileira como requisito parcial à obtenção do título de Engenheiro de Computação.

Orientador: Prof. Dr. Allberson Bruno de Oliveira Dantas

Aprovado em: 19/11/2025

BANCA EXAMINADORA

Prof. Dr. Allberson Bruno de Oliveira Dantas
Universidade da Integração Internacional da Lusofonia Afro-Brasileira

---

Prof. Dr. Aurélio Wildson Teixeira de Noronha
Universidade da Integração Internacional da Lusofonia Afro-Brasileira

---

Prof. Dr. Cenez Araújo de Rezende
Universidade Federal do Ceará

Prof. Dr. Luís Otávio Rigo Júnior
Universidade da Integração Internacional da Lusofonia Afro-Brasileira

AGRADECIMENTOS

Primeiramente, agradeço a Deus, por ser a força, a luz e o sustento que me permitiram chegar até aqui. Como também agradeço pelas pessoas que sempre oram por mim para que eu mantenha firme minha fé e realize todos os meus sonhos.

Aos meus pais, Andrea e Fabiano, pilares de toda a minha vida, que sempre apoiaram minha educação com o que fosse preciso. Nenhum agradecimento seria suficiente para retribuir todo o amor, sacrifício, incentivo e apoio incondicional. Esta conquista não é apenas minha, mas o fruto da confiança, do apoio e, principalmente, da educação que vocês me deram.

Ao meu irmão mais novo Talyson, que algum dia vai entender que quem trabalha com computação também trabalha (ele pergunta quando vou trabalhar de verdade), por todo carinho e amor retribuído, por ser meu companheiro diário, por sempre está perto de mim, seja em silêncio enquanto estudo ou fazendo bagunça enquanto brincamos.

Aos meus familiares próximos Carol, Juliana, Clebson, Alisson, Júlio, Geane, Adriano, Fátima, Wenderson, Yunara, Gleiciano, meus avós Maria de Nazaré e José Borges e a outros familiares, a lista é enorme, que sempre vibraram com minhas conquistas e ofereceram o conforto e o apoio necessários. O carinho de vocês foi fundamental.

As crianças com quem tenho contato, Livinha, Mayra Sophia, Thayllan, José, Maria Alice, Antonella, entre tantas crianças, que me mostram a pureza que existe em cada uma, o que me faz querer fazer um mundo melhor para elas viverem.

Ao meu orientador, Prof. Dr. Allberson Dantas, pela confiança depositada no potencial deste projeto. Agradeço imensamente pela paciência, pelas orientações sábias, pelo acompanhamento feito desde o início da graduação em projetos de iniciação científica, extensão, nas disciplinas e, por fim, no TCC. Agradeço pela disponibilidade em me guiar. Seus conselhos técnicos foram essenciais para transformar a ideia inicial do *Auxilium Mestre*

em um projeto de engenharia coeso e funcional.

A minha professora Dr. Marcia Farias, por ter me acompanhado durante a graduação em diversas disciplinas e ter se tornado uma grande amiga que carregarei para vida toda.

Ao Padre Marco, pelas orações, pelas palavras amigas, pelo carinho e pelos conselhos que fortaleceram cada vez mais minha fé.

Ao Misael, por ter sido a primeira pessoa que falou comigo quando entrei no curso e por ter se tornado um grande irmão, me apoiando e ajudando sempre nos trabalhos, não deixando eu desistir do curso quando eu desanimava.

À Larice, minha irmã de outra mãe, por ter me mostrado o amor de irmã que eu não sabia que precisava viver, obrigado por sempre ser essa amiga que você é, por arrancar sorrisos sempre que preciso.

À Kalyne, por ter sido uma amiga especial e ter conquistado meu coração de uma forma que nem imagina e por ter sido uma pessoa essencial durante a graduação.

Aos meus demais amigos próximos, em especial, o Alex, o Thiago, o Cardoso, o Yuri e a Evillem, pelo companheirismo, pelas conversas que aliviaram a pressão e pelos momentos de descontração que foram indispensáveis durante este processo.

A Alice, que foi uma das apoiadoras para eu entrar no curso, passou quase todo o processo comigo, sempre apoiando, sendo compreensiva e companheira, seguimos rumos diferentes, mas não poderia deixar de citar a sua importância em todo o processo do curso.

Aos meus colegas de curso, que compartilharam vivências e dificuldades cotidianas do que é fazer um curso de graduação nessa área. E a todos que fizeram parte desta trajetória, seja com uma palavra amiga, com apoio, com o amor oferecido, com o companheirismo, com as vivências compartilhadas, o meu muito obrigado.

RESUMO

A transformação digital na educação, acelerada pela pandemia de COVID-19, impôs aos educadores uma sobrecarga de trabalho significativa na criação de materiais didáticos. Embora os Modelos de Linguagem de Larga Escala (LLMs) tenham surgido como ferramentas de auxílio, o processo de geração de um curso completo, incluindo ementa, aulas, apresentações e avaliações, permanece fragmentado, manual e complexo. O presente trabalho aborda este problema através do objetivo de desenvolver e avaliar a plataforma *Auxilium Mestre*, uma aplicação web full-stack para a geração automatizada e orquestrada de conteúdo educacional. A metodologia empregada foi a prototipação evolutiva que permitiu o refinamento iterativo da solução. O sistema foi implementado utilizando uma arquitetura moderna, composta por um backend em Python, com FastAPI, um banco de dados relacional, PostgreSQL, gerenciado pelo ORM SQLAlchemy, e um frontend Vanilla JS (SPA). A plataforma implementa um pipeline que orquestra a API do Google Gemini para a geração de conteúdo e, em seguida, utiliza bibliotecas como python-docx, python-pptx e ReportLab para formatar os arquivos finais. A segurança e o isolamento de dados são garantidos por um sistema de autenticação robusto, baseado em tokens JWT, OAuth 2.0 (Google) e verificação de e-mail (SMTP). A avaliação da aplicação, realizada através de estudos de caso em domínios distintos, demonstrou que a arquitetura desenvolvida é funcional, segura e flexível. Os resultados confirmam que o *Auxilium Mestre* atende aos requisitos propostos, otimizando o processo de criação de materiais didáticos e reduzindo o tempo de produção para os educadores.

Palavras-chave: Geração de Conteúdo; Inteligência Artificial; LLM; FastAPI; Engenharia de Prompt.

ABSTRACT

The digital transformation in education, accelerated by the COVID-19 pandemic, has imposed a significant workload on educators regarding the creation of didactic materials. Although Large Language Models (LLMs) have emerged as auxiliary tools, the process of generating a complete course, including syllabus, lessons, presentations, and assessments, remains fragmented, manual, and complex. This work addresses this problem through the objective of developing and evaluating the *Auxilium Mestre* platform, a full-stack web application for the automated and orchestrated generation of educational content. The methodology employed was evolutionary prototyping, which allowed for the iterative refinement of the solution. The system was implemented using a modern architecture, composed of a Python backend with FastAPI, a PostgreSQL relational database managed by the SQLAlchemy ORM, and a Vanilla JS (SPA) frontend. The platform implements a pipeline that orchestrates the Google Gemini API for content generation and subsequently uses libraries such as python-docx, python-pptx, and ReportLab to format the final files. Security and data isolation are ensured by a robust authentication system based on JWT tokens, OAuth 2.0 (Google), and e-mail verification (SMTP). The application's evaluation, conducted through case studies in distinct domains, demonstrated that the developed architecture is functional, secure, and flexible. The results confirm that *Auxilium Mestre* meets the proposed requirements, optimizing the creation process of didactic materials and reducing production time for educators.

Keywords: Content Generation; Artificial Intelligence; LLM; FastAPI; Prompt Engineering.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1. INTRODUÇÃO

1.1. Contextualização

Recentemente, observa-se que a sociedade contemporânea vive uma transformação digital, que reconfigura e afeta todas as estruturas sociais, incluindo, os ambientes de aprendizagem (BALYER; ÖZ, 2018). A partir disso, a sala de aula expandiu-se de uma sala tradicional para ambientes virtuais complexos, incluindo, Ambientes Virtuais de Aprendizagem (AVAs), plataformas de Ensino à Distância (EAD), entre outras aplicações colaborativas.

A pandemia de COVID-19 atuou como acelerador dessa transformação digital, visto que, as consequências geradas por esse contexto mundial forçaram instituições de ensino, educadores e educandos a adaptarem-se rapidamente a novos modelos institucionais e plataformas digitais (MARTIN; XIE, 2022). Esta mudança nos métodos de ensino, migrando da interação presencial para o ensino remoto emergencial, expôs não apenas às limitações de infraestrutura, mas também um novo e pesado fardo sobre os educadores (PELENKAHU, 2022, p. iii, 17).

Esta transição presenciada, definida por Hodges et al. (2020) como ensino remoto emergencial, forçou os educadores a modificarem seus planos pedagógicos de forma imediata. Algo que tornou-se um fardo, devido a necessidade de gerar um volume sem precedentes de materiais digitais coesos. Os professores tiveram que obter novas habilidades, pois precisavam atuar simultaneamente na criação de conteúdo, de multimídia, muitas vezes as ferramentas adequadas.

Paralelamente a essa nova demanda educacional, o campo da tecnologia avançou com o desenvolvimento de modelos de transdução de sequência. A introdução da arquitetura "Transformer" (VASWANI et al., 2017), baseada exclusivamente em mecanismos de atenção, demonstrou ser superior aos modelos recorrentes e convolucionais, estabelecendo um novo estado da arte em tarefas como tradução automática e trouxe uma fundação que viria a sustentar os atuais Modelos de Linguagem de Larga Escala (LLMs).

A prova de que o dimensionamento (scaling) dessa arquitetura era a chave para habilidades mais generalizáveis foi demonstrada com o modelo GPT-3 (BROWN et al., 2020). No artigo, os autores provaram que, ao aumentar massivamente o modelo para 175 bilhões de parâmetros, a IA adquiria a capacidade emergente de realizar uma vasta gama de tarefas complexas, como sumarização, resposta a perguntas e até geração de código (com pouca ou nenhuma exemplificação prévia), uma habilidade denominada “few-shot learning”, que seria uma aprendizagem de poucos exemplos.

O potencial dessa tecnologia para a educação tornou-se imediatamente aparente. A capacidade dos LLMs de processar um tópico (“prompt") e gerar rascunhos de ementas, planos de aula, questões de múltipla escolha e roteiros para apresentações posicionou esta tecnologia como uma solução direta para o gargalo da criação de conteúdo (KASNECI et al., 2023). A IA Generativa surge não como uma substituta para o educador, mas como uma ferramenta de alavancagem, auxílio ao educador, visto que, um assistente capaz de automatizar as tarefas mecânicas e demoradas da produção de material didático, permite que o professor se concentre em outras atividades.

1.2. Problema de Pesquisa

Observa-se um potencial enorme em Modelos de Linguagem de Larga Escala (LLMs) como ferramentas de auxílios em diferentes áreas, inclusive, na área da educação, porém os educadores enfrentam uma lacuna entre a tecnologia bruta e sua aplicação prática. Por exemplo, se for feito o uso de um LLM genérico, onde você coloca um prompt sobre o que deseja, como no Chatgpt, isso exige um esforço manual considerável e habilidades com engenharia de prompt para gerar cada componente de um curso que o professor deseja obter através da IA.

O educador, além de orquestrar o prompt para cada componente, ainda necessita da geração de uma ementa, depois usar a ementa para gerar os textos que ele pode usar como referência em seus cursos, em seguida, formatar a apresentação de slides e ainda criar um roteiro para seguir durante as aulas e, por fim, elaborar avaliações para os tópicos abordados em aula. Todo esse processo, embora feito com o auxílio de IA, é fragmentado, demorado e até mesmo complexo.

Portanto, o problema de pesquisa abordado não é a geração de texto em si, pois isso com o devido conhecimento e usando um bom prompt pode ser gerado com um LLM genérico. Mas o problema vem da ausência de um sistema integrado que orquestre o pipeline completo de geração de materiais didáticos. A falta de uma solução unificada, específica e de fácil acesso que transforme um único tópico de entrada em um curso completo com ementas, aulas em DOCX, apresentações em PPTX e avaliações em PDF, de forma coesa e automatizada.

1.3. Pergunta de Pesquisa

Diante do problema apresentado, a pergunta central que norteia esse trabalho é:

"Como a integração de Modelos de Linguagem de Larga Escala (LLMs) em uma arquitetura web moderna (FastAPI, PostgreSQL, JavaScript) pode otimizar e escalar o processo de criação de materiais didáticos (ementas, aulas, apresentações e avaliações) para educadores, mantendo um alto nível de qualidade e coesão?"

1.4. Objetivos

Para responder à pergunta central da pesquisa, este trabalho define um objetivo geral e objetivos específicos que detalham as etapas do desenvolvimento.

1.4.1. Objetivo Geral

Desenvolver e avaliar uma plataforma nomeada como *Auxilium Mestre*, uma aplicação web full-stack para geração automatizada de conteúdo educacional para professores.

1.4.2. Objetivos Específicos

Foram estabelecidos, com o intuito de atingir o objetivo geral, os seguintes objetivos específicos:

- Estudar a arquitetura *Transformer* e os Modelos de Linguagem de Larga Escala (LLMs), como também, as tecnologias de desenvolvimento web full-stack, incluindo o framework FastAPI, o ORM SQLAlchemy, e o padrão de autenticação JWT.
- Projetar a arquitetura da solução, definindo o modelo de dados relacional (PostgreSQL) para as entidades User, Course e GeneratedFile, e especificando os contratos da API com Pydantic.
- Implementar o módulo de autenticação e segurança do backend, incluindo o registro de usuário com verificação de e-mail (SMTP/Brevo), login local com hashing de senha (Bcrypt), login social (Google OAuth2) e gerenciamento de sessão via Token JWT.
- Desenvolver os serviços de backend para a geração de conteúdo, criando um pipeline que (1) utiliza o Gemini para gerar roteiros estruturados (ementas, aulas), (2) busca de imagens em sites que disponibilizam imagens sem problemas de direitos autorais e (3) converte o conteúdo em arquivos finais (DOCX, PPTX, PDF).
- Construir a API RESTful (FastAPI) para expor a lógica de negócios, gerenciando o ciclo de vida completo dos cursos.

- Desenvolver a interface do usuário em JavaScript como uma Single Page Application (SPA), implementando a lógica de gerenciamento de estado e a comunicação assíncrona para consumir todos os endpoints do backend.
- Avaliar a aplicação através de estudos de caso, validando a funcionalidade de todos os requisitos e analisando a qualidade, coesão e usabilidade dos materiais gerados pela plataforma.

1.5. Justificativa

Este trabalho de conclusão de curso se justifica por sua relevância em três eixos principais, sendo eles, tecnológico, acadêmico e social.

Para a relevância tecnológica, o projeto vai além da utilização superficial de Modelos de Linguagem de Larga Escala (LLMs) em interfaces de chat. A inovação surge na aplicação prática da engenharia de software para construir um pipeline de orquestração automatizado. Pois, um LLM genérico, a depender do prompt, pode gerar fragmentos de texto, porém, a aplicação *Auxilium Mestre* fornece uma solução de ponta a ponta integrada com IA generativa com múltiplos serviços de geração de documentos, isso resolve o problema da fragmentação. A plataforma serve como uma aplicação prática de como os LLMs podem ser usados em uma arquitetura web completa para resolver um problema real.

A relevância acadêmica, no contexto da Engenharia de Computação, parte do desenvolvimento de um estudo de caso completo para a construção de uma aplicação web moderna e segura. Este trabalho documenta o processo do início ao fim, integrando múltiplos conceitos aprendidos durante o curso. O TCC aborda o design de uma API RESTful (FastAPI), a modelagem de um banco de dados relacional (PostgreSQL com SQLAlchemy), a a implementação de um sistema de autenticação e segurança robusto (JWT, OAuth2, verificação de e-mail e hashing de senhas com Bcrypt) e o desenvolvimento de uma interface de usuário reativa. O projeto serve como um exemplo prático da arquitetura e das metodologias necessárias para construir um sistema full-stack completo e funcional.

Por fim, a relevância social é o pilar central do projeto. O *Auxilium Mestre* é uma resposta direta para a sobrecarga de trabalho dos educadores, um problema identificado na contextualização deste trabalho. Ao automatizar a criação de materiais didáticos, uma tarefa mecânica e que consome um tempo significativo, a aplicação devolve aos professores tempo, um recurso valioso. Este tempo a mais que a plataforma fornece aos professores pode ser investido onde a presença humana é totalmente necessário e não pode ser substituída, na interação com os alunos, no aprendizado de novas técnicas pedagógicas, no acompanhamento

individual com alunos e nas suas tarefas pessoais, em vez de gastar horas desenvolvendo e formatando inúmeros documentos e slides.

1.6. Estrutura do Trabalho

Para uma exposição clara da pesquisa e do desenvolvimento da solução proposta, este Trabalho de Conclusão de Curso foi estruturado em seis capítulos. O capítulo 1, abordado até essa seção, traz uma introdução ao assunto, com contextualização, exposição do problema de pesquisa, da pergunta de pesquisa, objetivos, justificativa e estrutura do trabalho.

O Capítulo 2 apresenta a Fundamentação Teórica, a base conceitual necessária para a compreensão do projeto. Nesta seção, são revisadas as tecnologias centrais, desde os Modelos de Linguagem de Larga Escala (LLMs) e a Engenharia de Prompt, os primeiros estudos para o desenvolvimento do trabalho, até os pilares da arquitetura web moderna, podendo destacar também as bibliotecas de geração de documentos.

O Capítulo 3 detalha a Metodologia de Desenvolvimento utilizada. A seção foca na engenharia de software, definindo a abordagem de prototipação, o ambiente de desenvolvimento e os requisitos funcionais (RFs) e não-funcionais (RNFs) que guiaram a construção da plataforma *Auxilium Mestre*.

O Capítulo 4 documenta o desenvolvimento e implementação do *Auxilium Mestre.* Este capítulo é dedicado a apresentar a arquitetura da solução, a implementação detalhada do backend (API, banco de dados e segurança) e dos serviços de geração de conteúdo, bem como a construção da interface do usuário (frontend).

O Capítulo 5 foca nos Resultados e Avaliação da plataforma. Através de estudos de caso práticos, esta seção demonstra visualmente o sistema em operação, desde a entrada do usuário até a exibição dos materiais gerados, avaliando o cumprimento dos requisitos e a qualidade final dos arquivos gerados pela plataforma.

Por fim, o Capítulo 6 apresenta a Conclusão, onde os resultados são sintetizados, a pergunta de pesquisa é respondida, e são discutidas as limitações do sistema atual e as direções para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a revisão de literatura sobre os principais conceitos tecnológicos que fundamentam o desenvolvimento da plataforma *Auxilium Mestre*. A abordagem parte da definição ampla de Inteligência Artificial, depois partindo para os Modelos de Linguagem de Larga Escala (LLMs) e sua aplicação direta na geração de

conteúdo educacional.

2.1. Inteligência Artificial e Geração de Conteúdo

A Inteligência Artificial (IA) é um campo da ciência da computação focado em projetar e construir agentes inteligentes, sistemas que percebem seu ambiente e tomam ações que maximizam suas chances de atingir seus objetivos (RUSSELL; NORVIG, 2021). Este campo vasto é composto por diversos subcampos, destacando, o Aprendizado de Máquina (ML, do inglês Machine Learning), que utiliza algoritmos estatísticos para permitir que sistemas "aprendam" com dados e melhorem seu desempenho em uma tarefa específica sem serem explicitamente programados (RUSSELL; NORVIG, 2021).

O Aprendizado Profundo (DL, do inglês Deep Learning), por sua vez, é um subconjunto do ML que utiliza Redes Neurais Profundas (DNNs) com múltiplas camadas para aprender representações de dados com níveis crescentes de abstração, sendo a força motriz por trás dos avanços mais recentes (KASNECI et al., 2023). A tecnologia central do *Auxilium Mestre* reside em uma aplicação específica do Aprendizado Profundo, o Processamento de Linguagem Natural (NLP), que é a área da IA focada em capacitar computadores a processar, compreender e gerar linguagem humana.

2.1.1. Modelos de Linguagem de Larga Escala (LLMs)

O campo do Processamento de Linguagem Natural (NLP) passou por uma verdadeira revolução recentemente. A principal mudança foi a adoção de modelos pré-treinados, que aprendem a partir de volumes massivos de dados textuais (MIN et al., 2021). O marco técnico que possibilitou essa mudança foi a arquitetura Transformer, introduzida por Vaswani et al. (2017). Este novo modelo quebrou o padrão da época, deixando de lado as arquiteturas recorrentes (RNNs) e convolucionais (CNNs) e apostando tudo em um novo conceito chamado auto-atenção (self-attention). Na prática, esse mecanismo deu ao modelo a capacidade de processar frases inteiras de forma paralela e de entender como palavras distantes em um texto se relacionam, algo que os modelos anteriores faziam com muito mais dificuldade (VASWANI et al., 2017) .

O sucesso da arquitetura Transformer foi consolidado por modelos como o BERT (DEVLIN et al., 2018), que introduziu as Representações Bidirecionais do Codificador de Transformers. O BERT demonstrou o poder da pré-treinagem bidirecional profunda, habilitada pelo objetivo "Masked LM" (MLM), estabelecendo novos recordes de última geração em onze tarefas de NLP. A arquitetura Transformer também foi aplicada em

abordagens auto-regressivas, demonstrando a versatilidade do Transformer.

A prova de que o dimensionamento massivo dessa arquitetura era a chave para habilidades generalizáveis foi demonstrada com o modelo GPT-3 (BROWN et al., 2020). Brown et al. (2020) provaram que, ao aumentar o modelo para 175 bilhões de parâmetros, a IA adquiria a capacidade emergente de realizar uma vasta gama de tarefas complexas com pouca ou nenhuma exemplificação prévia. Esta habilidade, denominada “few-shot learning”, efetivamente deu início à era da IA Generativa (BROWN et al., 2020; FLORIDI; CHIRIATTI, 2020) .

O potencial dessa tecnologia para a educação tornou-se imediatamente aparente. A capacidade dos LLMs de processar um tópico e gerar conteúdo educacional, problemas práticos e questionários e personalizar experiências de aprendizagem (KASNECI et al., 2023) posicionou esta tecnologia como uma solução direta para o gargalo da criação de conteúdo. É exatamente esta capacidade de geração que a plataforma *Auxilium Mestre* busca orquestrar e automatizar.

2.1.2. Engenharia de Prompt

A simples disponibilidade de um Modelo de Linguagem de Larga Escala (LLM) não garante, por si só, resultados úteis ou corretos. Fatores como a qualidade, o formato e a precisão da saída de um LLM dependem da qualidade e clareza da entrada fornecida. A Engenharia de Prompt (do inglês, Prompt Engineering) surgiu como a disciplina de projetar, refinar e otimizar as instruções de entrada (os prompts) para guiar o modelo a produzir a saída desejada, uma técnica que se tornou central para a aplicação prática de LLMs (LIU et al., 2021).

Um prompt eficaz atua como um "contrato" para a IA, definindo o escopo, o formato e o contexto da tarefa. Para a plataforma *Auxilium Mestre*, a engenharia de prompt é a principal ferramenta de controle de qualidade e orquestração, dependendo de duas técnicas principais, sendo elas, a geração de texto formatado e a extração de saída estruturada (JSON).

Para a geração de texto formatado, como o conteúdo das aulas, a abordagem mais simples deriva da “few-shot learning” (aprendizagem de poucos exemplos). O prompt não apenas solicita o conteúdo, mas inclui uma instrução de formatação como, por exemplo, "Formate o texto usando Markdown" e, se necessário, "Use '##' para subtítulos", garantindo que a saída da IA seja facilmente processada por outros serviços, como o gerador de DOCX (BROWN et al., 2020).

A técnica mais complexa e vital para o *Auxilium Mestre* é a geração de saída

estruturada (JSON). A tarefa de gerar a ementa completa do curso com capítulos, tópicos e durações, exige que a IA retorne não um texto corrido, mas um objeto JSON válido e previsível. Para alcançar isso, uma engenharia de prompt mais sofisticada é necessária, combinando várias estratégias:

- Definição de Papel (Role-Playing): Esta é uma técnica fundamental onde o prompt instrui o modelo a assumir uma persona específica (ex: "Você é um especialista em design instrucional"). Atribuir um papel ao LLM ajuda a definir o tom, o estilo e o foco da resposta, alinhando a saída com o contexto desejado e melhorando a qualidade do conteúdo gerado (KUKA, 2024).
- Instrução de Formato: O prompt exige que a resposta seja apenas um objeto JSON, sem nenhum texto introdutório ou explicações adicionais.
- Injeção de Schema: A técnica mais robusta é injetar a própria planta (o schema) do formato de dados esperado diretamente no prompt. Esta é uma prática central da Engenharia de Prompt, pois o LLM não adivinha automaticamente o formato de saída. Conforme destacado por referências técnicas da indústria, como a documentação da Microsoft (2025), guiar o modelo fornecendo exemplos claros (few-shot learning) ou especificando explicitamente a estrutura de saída desejada (como um schema JSON) é fundamental para garantir respostas consistentes e estruturadas.

Ao definir o schema JSON desejado dentro do prompt, o *Auxilium Mestre* força a saída da IA a ser diretamente compatível com os modelos de validação Pydantic (discutidos na Seção 2.2.1) e, consequentemente, com o banco de dados.

2.2. Arquitetura de Aplicações Web Full-Stack

O *Auxilium Mestre* é uma aplicação web full-stack, ou seja, um sistema que compreende tanto a interface do usuário, executada no navegador (frontend), quanto a lógica de negócios e o gerenciamento de dados no servidor (backend). A arquitetura adotada neste projeto é a de uma aplicação web moderna onde o backend serve tanto os arquivos estáticos (HTML, CSS, JS) do frontend quanto uma API (Application Programming Interface) RESTful para a comunicação de dados.

2.2.1. Backend com FastAPI

Para o backend, a tecnologia escolhida foi o FastAPI, um framework web Python moderno e de alta performance, projetado para ser rápido de codificar e robusto para produção. Estudos que comparam frameworks modernos destacam sua performance eficiente

e arquitetura modular (CHEN, 2023).

A escolha do framework se fundamenta em três pilares tecnológicos que são cruciais para este TCC, sendo eles, o padrão ASGI, a validação de dados com Pydantic e o sistema de Injeção de Dependência.

A alta performance do FastAPI é um resultado direto da sua arquitetura, que opera sobre o padrão ASGI (Asynchronous Server Gateway Interface), contrastando com frameworks tradicionais baseados em WSGI (síncronos). A utilização do ASGI permite que o servidor gerencie operações de forma assíncrona, sendo uma arquitetura explicitamente desenhada para alta performance e concorrência em cenários IO-bound. Por exemplo, um benchmark comparando o FastAPI rodando como ASGI assíncrono com o Falcon rodando como WSGI síncrono no mesmo nó, demonstrou que o FastAPI atingiu 908.78 requisições/segundo, superando as 274.22 requisições/segundo do Falcon e reduzindo a latência média (BROOKINS, 2021). Essa eficiência é alcançada através do uso nativo de co-rotinas no Python, padronizadas e aprimoradas através da sintaxe async/await. A proposta PEP 492 introduziu a sintaxe async def para declarar uma co-rotina nativa e a expressão await para obter o resultado de uma execução de co-rotina, suspendendo a operação até que o objeto awaitable retorne o resultado, estabelecendo as co-rotinas como um recurso nativo da linguagem (SELIVANOV, 2015).

Esta é uma característica vital para o *Auxilium Mestre*, que depende intensamente de operações de Entrada/Saída (I/O) que envolve espera, como chamadas de rede para APIs externas (Google Gemini, Pexels) e a escrita de arquivos no disco. O uso de async/await permite que o servidor trate essas esperas de I/O sem bloquear o thread principal. Em vez de ficar ocioso, o event loop pode gerenciar outras requisições concorrentes, aumentando drasticamente a eficiência e a taxa de transferência (throughput) da aplicação (SELIVANOV, 2015).

No *Auxilium Mestre* é feito o uso de Pydantic para definir os modelos de dados que atuam como um “contrato” entre o frontend e o backend. O Pydantic utiliza os type hints (dicas de tipo) do Python, formalizados pela Proposta de Melhoria do Python PEP 484, para impor uma validação de dados rigorosa e criar schemas de dados claros (VAN ROSSUM; LEHTOSALO; LANGA, 2014).

Quando uma requisição (como a geração de um novo curso) chega à API, o FastAPI utiliza o modelo Pydantic para automaticamente:

- Analisar o JSON recebido.
- Validar se todos os campos obrigatórios estão presentes e com os tipos corretos, como:

num_chapters é um int.

- Converter os dados quando possível.
- Retornar automaticamente um erro JSON claro e detalhado ao frontend caso a validação falhe.

Esta validação é o que garante que os dados de entrada do usuário sejam seguros e corretos. Ela é a ferramenta que permite que os prompts (discutidos na Seção 2.1.2) forcem a saída do LLM a se conformar a um schema complexo, garantindo que o JSON gerado pela IA seja imediatamente compatível com o banco de dados.

Para a aplicação é importante citar a Injeção de Dependência (Dependency Injection). Este é um padrão de design de software amplamente conhecido por melhorar a manutenabilidade do código, permitindo que as classes da aplicação permaneçam desacopladas das dependências que elas utilizam (SUN; KIM, 2022). Dessa forma, no *Auxilium Mestre*, esse sistema é o que permite que os endpoints da API sejam protegidos (exigindo um token JWT válido através de Depends(get_current_user)) e tenham acesso ao banco de dados (Depends(get_db)) de forma eficiente, sem a necessidade de repetir código de autenticação ou conexão em cada rota.

2.2.2. Frontend com JavaScript "Vanilla"

O frontend é a camada da aplicação executada no navegador do usuário, ou seja, a parte que o usuário da aplicação tem contato, sendo o frontend responsável por apresentar a interface gráfica (UI) e gerenciar a interação. Para a aplicação foi adotada uma abordagem "Vanilla", termo que se refere ao uso do JavaScript puro, sem a dependência do uso de frameworks de frontend como React, Angular ou Vue.js.

Esta escolha se baseia na construção de uma aplicação leve, com controle total sobre o código e sem a sobrecarga de abstrações complexas, utilizando três pilares fundamentais da plataforma web moderna, a Manipulação do DOM, a Fetch API e o padrão de Single Page Application (SPA).

O Document Object Model (DOM) é a interface de programação (API) que representa um documento HTML ou XML como uma estrutura de árvore, onde cada nó corresponde a uma parte do documento (MDN WEB DOCS, 2025a). Em uma aplicação "Vanilla JS", a interatividade é alcançada através da manipulação direta do DOM.

No *Auxilium Mestre*, essa técnica é a base de toda a reatividade da interface. Em vez de recarregar a página inteira, o JavaScript "escuta" eventos do usuário e, em resposta, altera seletivamente partes da página. Isso inclui exibir e ocultar as "views" (como as telas de

geração e de **Meus Cursos**) simplesmente trocando classes de CSS, bem como preencher dinamicamente listas e exibir dados que foram recebidos do backend.

Para que o frontend possa se comunicar com o backend (FastAPI) de forma assíncrona, ele utiliza a Fetch API. Esta é uma interface moderna, nativa dos navegadores, para realizar requisições de rede (HTTP) (MDN WEB DOCS, 2025b). Ela substitui o antigo XMLHttpRequest e é baseada em Promises, o que facilita o gerenciamento de fluxos assíncronos, como esperar pela resposta de uma API.

No *Auxilium Mestre*, a Fetch API é o canal de comunicação para todas as operações, pode-se observar alguns exemplos a seguir:

- POST /api/token: Para realizar o login.
- GET /api/users/me: Para buscar os dados do usuário.
- POST /api/generate-course: Para enviar os dados do formulário do curso.
- GET /api/courses: Para listar os cursos gerados.
- GET /api/download/...: Para baixar os arquivos (tratando a resposta como um blob).
- DELETE /api/courses/...: Para excluir um curso.

A combinação da manipulação do DOM (para a interface) e da Fetch API (para os dados) permite a implementação de uma arquitetura de Single Page Application (SPA), ou Aplicação de Página Única. Em uma SPA, o usuário carrega um único arquivo HTML e o JavaScript assume o controle, renderizando dinamicamente as páginas ou views necessárias, e buscando ou enviando dados ao servidor em segundo plano (SILVA; ROSA, 2018).

No contexto de arquiteturas modernas, a camada de frontend é frequentemente implementada como uma Single Page Application (SPA), ou Aplicação de Página Única (OLIVEIRA, 2023).

Na arquitetura de referência proposta por Oliveira (2023), a SPA é um tipo de container do frontend (juntamente com aplicativos móveis). Esta camada se comunica com o backend através do protocolo HTTP, conectando-se a uma camada intermediária chamada Backend for Frontend (BFF), que por sua vez coordena as requisições para os microsserviços subjacentes (OLIVEIRA, 2023).

2.3. Persistência e Segurança de Dados

Para que uma aplicação web full-stack seja funcional, não basta apenas processar dados, é preciso armazená-los de forma eficiente, ou seja, persistente e protegê-los de acessos não autorizados. O *Auxilium Mestre*, por ser uma plataforma que armazena informações de usuários e conteúdos gerados, depende de um conjunto robusto de tecnologias para gerenciar

o ciclo de vida dos dados e a identidade do usuário.

2.3.1. Banco de Dados Relacional (PostgreSQL) e ORM (SQLAlchemy)

A persistência de dados no *Auxilium Mestre* é gerenciada por um Sistema Gerenciador de Banco de Dados Relacional (SGBDR). Foi escolhido o PostgreSQL, um SGBDR de código aberto.

Para mediar a comunicação entre a aplicação Python (FastAPI) e o banco de dados PostgreSQL, utiliza-se um Mapeador Objeto-Relacional (ORM), especificamente o SQLAlchemy. Um ORM é uma técnica que abstrai a lógica de consulta ao banco de dados, permitindo ao desenvolvedor interagir com as tabelas como se fossem classes Python e com os registros como se fossem objetos. Esta abordagem aumenta a produtividade, reduz a necessidade de escrever código SQL repetitivo e mitiga riscos de segurança, como a injeção de SQL (MIGUEL, 2022).

2.3.2. Hashing de Senhas (Passlib e Bcrypt)

Em uma aplicação segura, senhas de usuários nunca devem ser armazenadas em texto plano. Para proteger dados sensíveis, como senhas, a informação deve ser gravada de forma criptografada, evitando que seja descoberta caso haja acesso ao arquivo onde foi armazenada (CERT.br, 2012). Uma técnica criptográfica para esse fim é a função de resumo (Hash), um método criptográfico que, quando aplicado à informação, gera um resultado único e de tamanho fixo, este hash é gerado de tal forma que não é possível realizar o processamento inverso para se obter a informação original (CERT.br, 2012). A proteção das senhas é fundamental para prevenir ataques de força bruta (brute force), que consistem em adivinhar o nome de usuário e a senha por tentativa e erro (CERT.br, 2012).

Para esta finalidade, o *Auxilium Mestre* utiliza a biblioteca Passlib, que é um pacote Python abrangente para manipulação de hashes de senha (PASSLIB, 2020). A Passlib simplifica o processo e implementa o algoritmo bcrypt, que segue um padrão amplamente adotado para armazenamento seguro de senhas. Ao fazer login, ela utiliza sua função de verificação para comparar o hash da senha fornecida com o hash armazenado no banco, sem nunca reverter a criptografia (PASSLIB, 2020).

2.3.3. Autenticação baseada em Token (JWT)

Para gerenciar a sessão de usuários logados em uma API RESTful, a abordagem moderna é a autenticação stateless (sem estado) via token. O padrão adotado é o JSON Web

Token (JWT), definido pela RFC 7519, sendo um formato compacto e autossuficiente para transmitir informações (chamadas claims) entre duas partes (SEGUINS, 2022).

Um JWT consiste em três partes:

- Header (Cabeçalho): Informa o tipo de token e o algoritmo de assinatura.
- Payload (Carga Útil): Contém os claims, como o identificador do usuário (sub) e o tempo de expiração (exp).
- Signature (Assinatura): Garante a autenticidade do token.

No fluxo do *Auxilium Mestre*, quando um usuário faz login, o servidor gera um JWT assinado com uma chave secreta e o envia ao frontend. O frontend então armazena esse token e o envia no cabeçalho Authorization de todas as requisições futuras, provando sua identidade sem que o servidor precise armazenar o estado da sessão.

2.3.4. Autenticação Social (OAuth 2.0 com Google)

Para simplificar o processo de registro e login, a plataforma implementa o **Login com o Google**. O OAuth 2.0 é um framework de autorização. Seu objetivo é permitir que uma aplicação de terceiros obtenha acesso delegado e limitado a recursos protegidos (como os repositórios do GitHub ou a lista de contatos do Google) em nome do usuário, sem que este precise expor sua senha à aplicação (NEVES, 2024).

No *Auxilium Mestre*, o fluxo de OAuth 2.0 é usado para que o Google ateste a identidade do usuário. O Google retorna ao *backend* as informações de perfil do usuário, como nome e e-mail. O *backend* então utiliza essas informações para criar uma conta local para o usuário se for o primeiro acesso e gerar seu próprio *token* JWT, integrando o usuário ao sistema de autenticação padrão da aplicação.

2.3.5. Verificação de E-mail (SMTP)

Para garantir que o e-mail fornecido em um cadastro local é válido e pertence ao usuário, o *Auxilium Mestre* implementa um fluxo de verificação de e-mail. Este processo utiliza o SMTP (Simple Mail Transfer Protocol), que é o protocolo padrão usado para fazer a comunicação com o servidor remoto para enviar o e-mail de um cliente local para o servidor de e-mail do destinatário (RAFAEL, 2025).

Quando um novo usuário se cadastra, a aplicação utiliza uma biblioteca para se conectar a um serviço de SMTP (como o Brevo) e enviar um e-mail transacional. Este e-mail contém um token único de verificação (um JWT de curta duração) com um link de ativação. A conta do usuário permanece inativa no banco de dados (is_active=False) até que ele clique

neste link, validando a posse do e-mail e ativando sua conta.

2.4. Tecnologias de Geração de Documentos

A simples geração de conteúdo textual pela IA não cumpre o requisito de entregar materiais didáticos prontos para o uso. O *Auxilium Mestre* necessita de uma camada de software capaz de converter essa saída de texto bruto em formatos de arquivo profissionais e universais: .docx, .pptx e .pdf. Esta seção descreve as bibliotecas Python de código aberto selecionadas para essa finalidade.

Para a geração de documentos Microsoft Word (.docx), a plataforma utiliza a biblioteca python-docx. Esta biblioteca permite a criação e manipulação programática de documentos Word, possibilitando a adição de parágrafos, cabeçalhos, listas (com marcadores ou numeração) e a aplicação de estilos (CANNY, 2024a). No *Auxilium Mestre*, ela é a ferramenta que traduz o texto em Markdown (gerado pela IA para as aulas) em um documento de aula formatado.

De forma análoga, para a criação de apresentações de slides, foi utilizada a biblioteca python-pptx. Esta biblioteca oferece uma API para criar e modificar arquivos Microsoft PowerPoint (.pptx). Sendo fundamental para o serviço de geração de slides, pois permite que o backend construa a apresentação slide a slide, adicionando caixas de texto, inserindo imagens (obtidas pelo ImageManagerService) e formatando o layout de acordo com o roteiro JSON gerado pela IA (CANNY, 2024b).

Por fim, para a geração das avaliações, foi selecionada a biblioteca ReportLab. Trata-se da principal solução de código aberto em Python para a criação programática de documentos no formato PDF (Portable Document Format). Diferente das bibliotecas anteriores, o ReportLab não modifica um template existente, mas "desenha" o documento do zero. Ele utiliza um sistema de flowables e folhas de estilo, permitindo um controle de layout complexo e preciso (REPORTLAB, 2025). Essa granularidade é o que possibilita ao *Auxilium Mestre* gerar os dois PDFs de quiz (versão do aluno e versão do professor) a partir da mesma fonte de dados JSON.

3. METODOLOGIA

Este capítulo detalha os procedimentos metodológicos que guiaram a concepção, implementação e avaliação da plataforma *Auxilium Mestre*. Este trabalho é um projeto de Engenharia de Computação focado no desenvolvimento de um produto de software, sendo esse produto uma aplicação web full-stack projetada para a geração automatizada de conteúdo

educacional. O núcleo desta solução utiliza Modelos de Linguagem de Larga Escala (LLMs) para orquestrar a criação de cursos completos a partir de uma única entrada do usuário.

3.1. Abordagem Metodológica

Esta abordagem de engenharia de software é uma implementação prática do método de Pesquisa-Ação (Action Research). A Pesquisa-Ação é definida como um processo cíclico que visa o aprimoramento da prática através de uma permutação sistemática entre a investigação e a prática, seguindo um ciclo de planejamento, implementação, registro dos efeitos e avaliação dos resultados (OLIVEIRA, 2023).

3.1.1. O Protótipo de Viabilidade

A primeira etapa do projeto foi focada na mitigação de riscos técnicos. Antes de investir esforço na construção de uma aplicação web completa, com autenticação e banco de dados, era fundamental validar a hipótese central do projeto: a viabilidade de usar um LLM para gerar um conjunto coeso de arquivos de curso (o core business). Logo, nessa primeira parte, os esforços foram focados na busca de um LLM que pudesse ser capaz de rodar por trás da aplicação sem grandes custos.

Esta etapa foi executada em um ambiente isolado, logo foi usado um notebook do Google Colab. Nesse momento, o objetivo não era criar um produto, mas responder a perguntas técnicas específicas:

- Um LLM poderia, através da Engenharia de Prompt, gerar saídas estruturadas e previsíveis (JSON) para uma ementa?
- Esse mesmo LLM poderia gerar conteúdo textual formatado (Markdown) para aulas?
- As bibliotecas Python conseguiriam consumir essas saídas brutas e convertê-las em arquivos finais (DOCX, PPTX, PDF) de forma confiável?

O resultado desta fase foi um protótipo funcional, mas não escalável, uma prova de conceito (PoC). Ele validou com sucesso a funcionalidade central e forneceu os insights iniciais para seguir com o planejamento da arquitetura completa.

3.1.2. Ação e Implementação

Com a prova de conceito validada, o protótipo não foi descartado, mas sim evoluído incrementalmente, sendo envolvido por camadas de arquitetura de software para transformá-lo em uma aplicação robusta, modular e segura.

Esta evolução incluiu quatro grandes frentes de ação:

A **migração do backend (FastAPI)**, onde a lógica de negócios foi migrada do script do protótipo para um backend FastAPI. Esta escolha foi planejada para implementar o padrão ASGI (como fundamentado na Seção 2.2.1), permitindo que as operações de I/O, como chamadas à API do Gemini e escrita de arquivos no disco, fossem tratadas de forma assíncrona, um requisito crucial para a performance da aplicação.

A **implementação da persistência (PostgreSQL e SQLAlchemy)** para superar a natureza efêmera do protótipo e permitir o armazenamento de dados, visto que, inicialmente foi feito o uso do Google Colab e não era feito o armazenamento em banco de dados, com isso, foi implementada a camada de persistência. O SQLAlchemy foi usado para modelar as entidades (User, Course, GeneratedFile), e o PostgreSQL foi escolhido como SGBDR para garantir a integridade relacional e a escalabilidade futura dos dados.

A **adição de uma camada de segurança completa** foi necessária para que a aplicação pudesse servir a múltiplos usuários, endereçando o requisito de **SaaS Multilocatário** (Software as a Service Multilocatário). Este conceito é definido como uma arquitetura onde uma única instância da aplicação serve múltiplos locatários (clientes) (MICROSOFT, 2025). A principal vantagem dessa abordagem é a otimização de recursos e a economia de escala, pois os custos de infraestrutura são compartilhados.

Porém, o desafio fundamental desta arquitetura é garantir o rigoroso isolamento dos dados, assegurando que um locatário não possa, em hipótese alguma, acessar os dados de outro (MICROSOFT, 2025). Para atender os requisitos de segurança e isolamento no *Auxilium Mestre*, a camada implementada incluiu o hashing de senhas (Bcrypt/Passlib), a autenticação via token (JWT) e o login social (Google OAuth2), garantindo a separação lógica e segura dos dados de cada usuário.

A **construção do frontend (SPA)** torna a ferramenta acessível ao público-alvo e foi construída a interface do usuário como uma Single Page Application (SPA). Esta camada consome a API do backend, via Fetch, e manipula o DOM para criar uma experiência de usuário fluida e reativa.

3.1.3. Avaliação

Esta etapa foi crucial para identificar desafios técnicos que não eram aparentes no protótipo inicial. Foi nesta etapa que os desafios de engenharia mais complexos foram identificados e resolvidos de forma prática. E como deve ser possível perceber, o projeto foi desenvolvido em fases, essas fases podem ser chamadas de ciclos, onde no ciclo 1 foi implementado um protótipo da aplicação no colab e, a partir disso, surgiram novos ciclos

necessários para o desenvolvimento da aplicação, podemos observar isso na Tabela 1.

Quadro 1 - Ciclos Evolutivos do Desenvolvimento da Plataforma

Ciclo	Avaliação (Problema Identificado)	Planejamento (Solução Aplicada)
1	A viabilidade de usar um LLM para gerar um conjunto coeso de arquivos educacionais.	Foi planejado e desenvolvido uma primeira versão de teste utilizando um notebook no Google Colab, onde foi possível usar o LLM para gerar arquivos educacionais, cada arquivo por vez.
2	O LLM, sob certas condições, retornava um JSON malformado para a ementa, quebrando o *backend*.	Foi planejado um refinamento da Engenharia de Prompt (Seção 2.1.2), injetando um *schema* explícito na instrução.
3	A geração do curso (chamada de IA + 5 escritas de arquivo) era uma operação longa que bloqueava o servidor.	Foi planejado refatorar o *endpoint* /generate-course para ser totalmente assíncrono (async def), liberando o servidor (conforme Seção 2.2.1).
4	Os arquivos gerados na pasta "output/" estavam publicamente acessíveis se o nome do arquivo fosse adivinhado.	Foi planejado um novo *endpoint* (/api/download/...) que exigia um *token* JWT válido, garantindo a segurança dos arquivos.

Fonte: O autor (2025)

Este processo iterativo garantiu que o produto final não fosse apenas funcional, mas também robusto, seguro e escalável.

3.2. Definição de Requisitos do Sistema

Com a abordagem metodológica definida, o próximo passo foi a elicitação e definição dos requisitos do sistema. Os requisitos são divididos em Requisitos Funcionais (RFs), que descrevem o que o sistema faz, e Requisitos Não Funcionais (RNFs), que descrevem como o sistema opera (por exemplo: performance, segurança).

3.2.1. Requisitos Funcionais

Os Requisitos Funcionais (RFs) especificam os comportamentos e serviços que o *Auxilium Mestre* deve prover ao usuário. Eles foram derivados da Pergunta de Pesquisa

(Seção 1.3) e do Problema de Pesquisa (Seção 1.2), focado em resolver o fardo do educador através da automação, fornecendo a eles uma plataforma completa para geração de cursos, como abordado anteriormente. Os RFs que guiaram o desenvolvimento são detalhados a seguir.

O **RF01** define o cadastro de usuário com verificação de e-mail, logo o sistema deve permitir que um novo usuário crie uma conta, esse processo exige um nome, um e-mail válido e uma senha.

O **RF02** define a autenticação do usuário, com isso, o sistema prover duas formas de autenticação para o usuário, sendo elas o Login Local (E-mail e Senha) e o Login Social (Google).

O **RF03** define o processo de geração de curso, onde o usuário, já autenticado, pode submeter um formulário contendo o tópico do curso, o nível e o número de capítulos, sendo este o requisito funcional central da aplicação.

O **RF04** define o dashboard, ou seja, a listagem de cursos, onde o usuário, já autenticado, pode visualizar uma lista de todos os cursos gerados por ele. Ao clicar na aba **Meus Cursos** (Figura 4), o frontend envia uma requisição GET para /api/courses. O backend consulta o banco de dados e retorna uma lista de cursos onde o user_id da tabela Course corresponde ao ID do usuário autenticado, obtido do token JWT. O frontend então renderiza dinamicamente os cards dos cursos.

O **RF05** define a exclusão segura de um curso, visto que, o usuário autenticado deve poder excluir um curso de sua propriedade, com isso, para prevenir ações acidentais, o sistema deve exigir a senha do usuário como confirmação (Figura 5).

Por fim, o **RF06** define o download seguro de materiais gerados, assim, o usuário pode baixar os arquivos gerados de um curso.

3.2.2. Requisitos Não Funcionais

Os Requisitos Não Funcionais (RNFs) não descrevem o que o sistema faz, mas como ele o faz. Eles definem os atributos de qualidade, restrições e padrões operacionais da aplicação, sendo essenciais para a usabilidade e a robustez do software.

O **RNF01** representa a segurança e isolamento de dados, sendo um atributo de qualidade crítico. A aplicação deve garantir a confidencialidade, integridade e disponibilidade dos dados do usuário. Sendo uma aplicação SaaS Multilocatário (conceito discutido na Seção 2.2.2), o sistema deve impor um isolamento de dados rigoroso, garantindo que um usuário não possa, sob nenhuma circunstância, acessar os dados de outro.

Já o **RNF02** destaca a usabilidade e a responsividade, pois o sistema deve ser intuitivo e fácil de usar para o público-alvo. A interface do usuário (UI) deve ser responsiva, adaptando-se a diferentes tamanhos de tela (desktop, tablet e mobile) para garantir uma experiência de uso consistente.

Por fim, o **RNF03** define a escalabilidade e desempenho, visto que, a arquitetura do sistema deve ser capaz de crescer para suportar um aumento no número de usuários, requisições e dados. O desempenho (tempo de resposta) e a escalabilidade (capacidade de expansão) são atributos de qualidade fundamentais.

3.3. Ferramentas e Ambiente

A implementação da arquitetura definida e o cumprimento dos requisitos funcionais e não funcionais exigiram um conjunto específico de hardware e software. O ambiente de desenvolvimento e produção utilizado para construir o *Auxilium Mestre* é detalhado a seguir:

- Requisitos de Hardware: Notebook (Intel Core i3, 4GB RAM), utilizado para codificação, execução de testes locais e gerenciamento do projeto.

- Ambiente de SO: WSL 2 (Windows Subsystem for Linux) com Ubuntu 24.04.2 LTS, para criar um ambiente de desenvolvimento Linux em paridade com o servidor de produção.
- Linguagens: Python (para o backend), JavaScript ES6 (para o frontend) e CSS3/HTML5.
- Desenvolvimento: Visual Studio Code (VS Code), utilizado como Ambiente de Desenvolvimento Integrado (IDE) principal.
- Controle de Versão: Git e GitHub, para gerenciamento do código-fonte.
- Banco de Dados: PostgreSQL, SGBDR de código aberto utilizado para armazenar os dados dos usuários, cursos e arquivos.
- APIs de Terceiros:
    - Google: Utilizado para a API do Google Gemini (geração de conteúdo) e para a API Google Identity (login social com OAuth 2.0).
    - Brevo: Utilizado como o serviço de gateway SMTP (Simple Mail Transfer Protocol) para o envio de e-mails transacionais (verificação de conta).

4. DESENVOLVIMENTO E IMPLEMENTAÇÃO DO AUXILIUM MESTRE

Este capítulo constitui o núcleo prático deste Trabalho de Conclusão de Curso, detalhando a implementação da plataforma *Auxilium Mestre*. Ele traduz os conceitos teóricos revisados no Capítulo 2 e os Requisitos Funcionais e Não Funcionais (RFs/RNFs) definidos no Capítulo 3 em uma solução de software funcional.

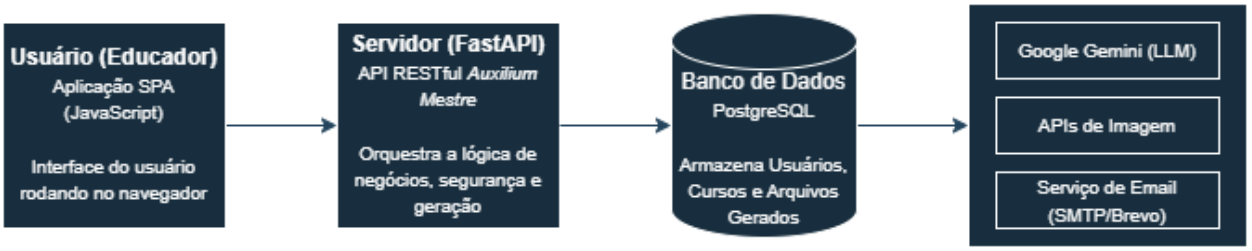
4.1. Arquitetura da Solução

A arquitetura do *Auxilium Mestre* foi projetada como uma aplicação web

full-stack moderna, seguindo um padrão de frontend desacoplado que consome uma API backend central. Este design modular permite uma separação de preocupações, onde o frontend (Cliente) gerencia exclusivamente a interface e a experiência do usuário, enquanto o backend (Servidor) gerencia toda a lógica de negócios, segurança e persistência de dados.

A Figura 1 ilustra o diagrama de arquitetura de alto nível da solução, detalhando os quatro principais componentes e os fluxos de comunicação entre eles.

Figura 1 - Diagrama da Arquitetura de Alto Nível do *Auxilium Mestre*



Fonte: O autor (2025)

Os componentes desta arquitetura são descritos a seguir:

No **frontend**, cliente, a implementação foi feita como uma Single Page Application (SPA) em JavaScript "Vanilla" (conforme Seção 2.2.2). Esta camada é executada inteiramente no navegador do usuário e é responsável por toda a interação (UI/UX). Ela gerencia as telas de login, cadastro, geração de curso e listagem de cursos, comunicando-se com o backend de forma assíncrona através de requisições Fetch API para buscar ou enviar dados.

O **backend**, servidor API, que é considerado o cérebro da aplicação, foi implementado com o FastAPI (conforme Seção 2.2.1). Esta camada expõe uma API RESTful segura e é responsável por toda a lógica de negócios. Suas funções primárias são: implementar o módulo de autenticação e segurança (RF01, RF02), orquestrar o pipeline de geração de conteúdo (RF03) e gerenciar o ciclo de vida dos dados (RF04, RF05, RF06). Ele utiliza o padrão ASGI para alta performance assíncrona.

Ao tratar de persistência, o banco de dados utiliza o PostgreSQL, um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) robusto (conforme Seção 2.3.1). Esta camada é responsável por armazenar permanentemente os dados da aplicação, incluindo os registros dos usuários, os metadados dos cursos gerados e o manifesto dos arquivos (GeneratedFile), todos gerenciados pelo ORM SQLAlchemy.

Por fim, na arquitetura da plataforma foi incluído serviços necessários para o funcionamento da aplicação, os serviços de terceiros (APIs Externas), que representam as

dependências externas que o backend orquestra para cumprir suas funções. Para o *Auxilium Mestre*, eles são:

- Google (LLM e OAuth): API do Google Gemini para a geração de conteúdo (Seção 2.1.1) e a API Google Identity para o login social (Seção 2.3.4).
- APIs de Imagem (Pexels, Pixabay): Utilizadas pelo ImageManagerService para enriquecer as apresentações.
- Brevo (SMTP): Utilizado como o gateway de e-mail para a verificação de contas (Seção 2.3.5).

As seções seguintes deste capítulo (4.2, 4.3 e 4.4) detalharão a implementação de cada um desses componentes.

4.2. Arquitetura da Solução

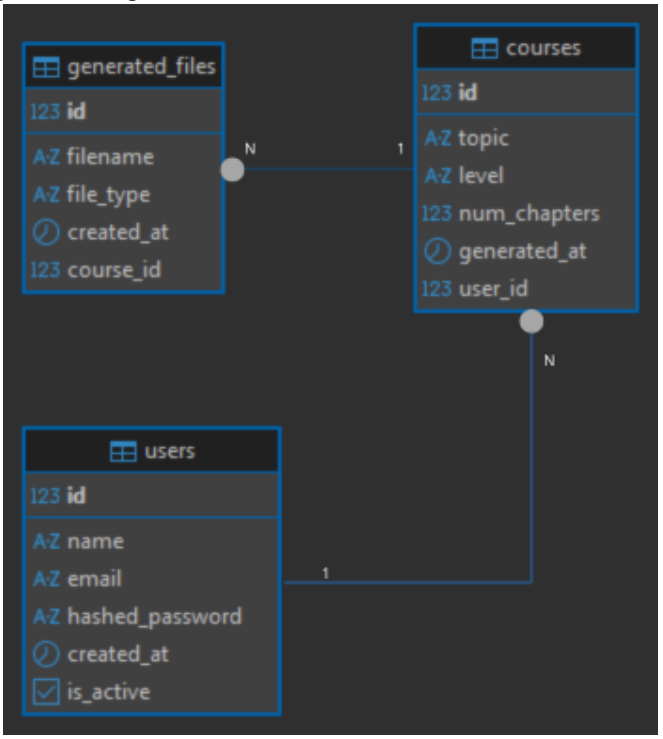
O backend é o componente central da arquitetura (conforme Figura 1), responsável por executar toda a lógica de negócios, aplicar as regras de segurança e gerenciar a persistência dos dados. A implementação foi realizada utilizando o framework FastAPI, com base nos conceitos de ASGI, Pydantic e Injeção de Dependência, conforme fundamentado nas seções 2.2.1 e 2.3. Vamos abordar inicialmente nesta seção modelos de dados e banco de dados.

4.2.1. Modelo de Dados e Banco de Dados

A fundação de qualquer aplicação robusta é o seu modelo de dados. Para atender aos Requisitos Funcionais (RFs) e Não Funcionais (RNFs) definidos, foi projetado um banco de dados relacional, implementado no SGBDR PostgreSQL e gerenciado através do ORM SQLAlchemy. O modelo é composto por três tabelas principais: User, Course e GeneratedFile, cujos relacionamentos garantem a integridade dos dados e o isolamento (multilocação) exigido pelo RNF01.

A Figura 2 apresenta o Diagrama Entidade-Relacionamento (DER) da solução, ilustrando as tabelas, seus atributos e os relacionamentos de chave estrangeira que conectam os dados.

Figura 2 - Diagrama Entidade-Relacionamento do *Auxilium Mestre*



Fonte: O autor (2025)

A Tabela **User (Modelo db_models.User)** é a base do sistema de autenticação e armazena as informações de perfil de cada usuário (locatário) e tem o objetivo de atender aos requisitos RF01 (Cadastro) e RF02 (Autenticação).

Campos principais (conforme Figura 2):

- id: Chave primária (PK) do tipo Inteiro, autoincrementada.
- name: (String) Nome do usuário, coletado no cadastro.
- email: (String) E-mail único, usado como nome de usuário para o login local.
- hashed_password: (String) Armazena o hash seguro (Bcrypt) da senha do usuário, conforme RNF01. A senha original nunca é armazenada.
- created_at: (Timestamp) Registra a data e hora em que a conta foi criada.
- is_active: (Boolean) Campo de controle para o fluxo de verificação de e-mail (RF01). O valor é definido como False no cadastro e True após o usuário validar o e-mail.

Já a tabela courses armazena os metadados de cada curso gerado pela plataforma, atuando como a entidade principal do core business. Atender ao RF03 (Geração de Curso) e RF04 (Listagem).

Campos Principais (conforme Figura 2):

- id: Chave primária (PK).
- topic: (String) O tópico do curso inserido pelo usuário (ex: "Python para Iniciantes").
- level: (String) O nível de dificuldade (ex: "Iniciante").

- num_chapters: (Integer) O número de capítulos solicitado.
- generated_at: (Timestamp) Registra a data e hora em que o curso foi gerado.
- user_id: (Integer) Chave Estrangeira (FK) que referencia users.id.

A tabela **generated_files** atua como um manifesto ou registro de cada arquivo gerado e salvo no servidor. O objetivo da tabela é atender ao RF06 (Download de Materiais), conectando os arquivos aos seus cursos correspondentes.

Campos Principais (conforme Figura 2):

- id: Chave primária (PK).
- filename: (String) O nome real do arquivo no disco (ex: curso_python_cap_1_aula.docx).
- file_type: (String) Um identificador do tipo de material (ex: ementa_docx, slides_pptx, quiz_aluno_pdf).
- created_at: (Timestamp) Registra a data e hora em que o arquivo foi gerado.
- course_id: (Integer) Chave Estrangeira (FK) que referencia courses.id.

O pilar central da arquitetura de dados e do requisito de segurança RNF01 reside nos relacionamentos de Chave Estrangeira:

O relacionamento das tabelas User e Course é 1-N, ou seja, um-para-muitos, sendo definido pela chave estrangeira courses.user_id apontando para users.id. Isso garante que um usuário pode ter muitos cursos, mas um curso pertence a um usuário. É esta restrição que permite ao backend filtrar e isolar os dados, garantindo que um usuário autenticado só possa listar ou acessar os cursos que ele mesmo criou.

O relacionamento das tabelas Course e GeneratedFile é de 1-N, ou seja, um-para-muitos, sendo definido pela chave estrangeira generated_files.course_id apontando para courses.id. Isso garante que um curso pode ter muitos arquivos associados (DOCX, PPTX, etc.), mas um arquivo pertence a um, e somente um, curso. É este relacionamento que permite ao backend recuperar todos os materiais de um curso (RF06) e excluí-los de forma segura (RF05).

4.2.2. Módulo de Autenticação e Segurança

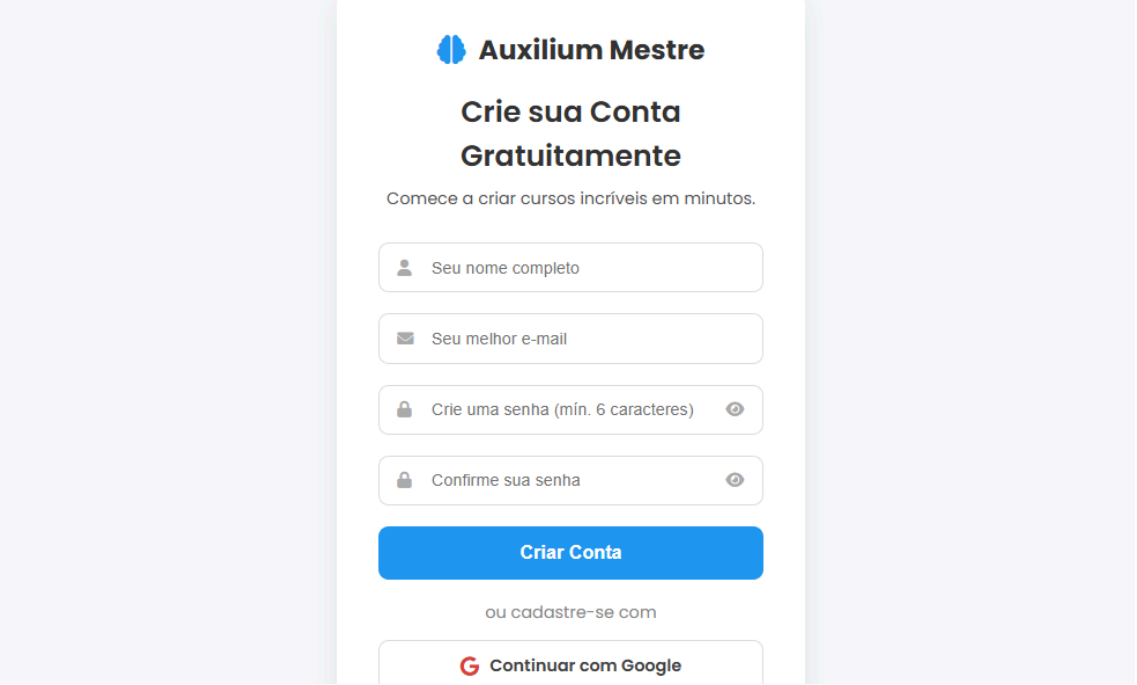
O Módulo de Autenticação e Segurança é o guardião (gatekeeper) da aplicação. Ele é responsável por implementar os requisitos funcionais RF01 e RF02, requisitos funcionais de cadastro e autenticação, respectivamente, bem como o requisito não funcional RNF01, de segurança e isolamento de dados.

Este módulo foi implementado no backend através de um conjunto de endpoints

dedicados no arquivo routes.py e funções de serviço no auth.py. O fluxo de registro (**RF01**) inicia-se na interface de cadastro (signup.html), ilustrada na Figura 3.

O frontend coleta os dados (nome, e-mail, senha) e os envia via POST para o endpoint /api/users. No backend, o FastAPI utiliza o modelo Pydantic UserCreate para validar os dados e a função security.get_password_hash() (Bcrypt/Passlib) para gerar um hash seguro da senha. O novo usuário é salvo no banco de dados com o campo is_active definido como False. Imediatamente, o email_service (Seção 2.3.5) é chamado para disparar um e-mail transacional, via SMTP/Brevo, contendo um link de ativação. Este link, que aponta para o endpoint GET /api/auth/verify-email, contém um JWT de curta duração. A conta só é ativada (is_active=True) quando o usuário clica neste link.

Figura 3 - Interface de Cadastro



Fonte: O autor (2025)

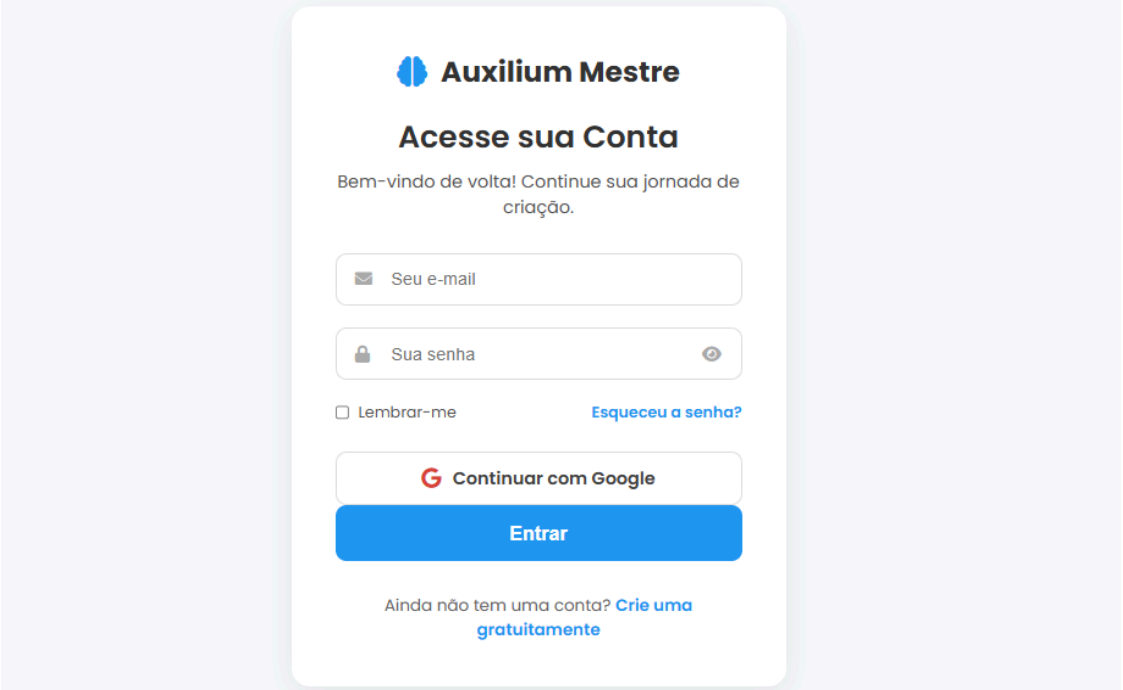
O requisito **RF02**, que define a autenticação de usuário (seção 3.2.1), detalha como um usuário existente acessa a plataforma. A Figura 4 ilustra a interface de autenticação, que oferece dois fluxos distintos.

O primeiro é a autenticação local, que utiliza o endpoint POST /api/token. A interface coleta as credenciais, conforme esperado pela dependência OAuth2PasswordRequestForm do FastAPI. O backend então chama a função auth.authenticate_user, que valida o hash da senha e, crucialmente, verifica se a conta está ativa (user.is_active é True).

O segundo fluxo é a autenticação social, o **LOGIN COM GOOGLE**, que utiliza GET /api/login/google. O frontend direciona o usuário para este endpoint do backend, que o

redireciona para a página de consentimento do Google (OAuth 2.0). Após o consentimento, o Google retorna o usuário ao callback da aplicação por /api/auth/google/callback, onde o backend valida o usuário e o cria no banco de dados local, se for novo. Em ambos os fluxos bem-sucedidos, o servidor gera e retorna um JSON Web Token (JWT) (Seção 2.3.3) para o frontend gerenciar a sessão.

Figura 4 - Tela de Login



Fonte: O autor (2025)

A implementação do RNF01, segurança e isolamento de dados, é o componente de segurança mais importante do backend e é possibilitada pelo JWT gerado no login. Este requisito não é um endpoint, mas sim uma Dependência do FastAPI (Seção 2.2.1), definida no arquivo auth.py como get_current_user. Todas as rotas que manipulam dados sensíveis, como /api/generate-course, /api/courses, etc, incluem esta dependência em sua assinatura (um exemplo disso pode ser visto no **Código-Fonte 1**).

Código-Fonte 1 - Dependência do FastAPI

```
@router.get("/courses", response_model=models.CoursesListResponse,
tags=["Courses"])
def get_user_courses(
    db: Session = Depends(get_db),
    current_user: db_models.User = Depends(auth.get_current_user)
):
```

Fonte: O autor (2025)

Quando o frontend faz uma chamada a uma rota protegida, o FastAPI primeiro executa a função get_current_user. Esta função extrai o token JWT do cabeçalho

Authorization, utiliza a biblioteca *jose* para decodificar e validar a assinatura e o tempo de expiração do token e, em seguida, busca o usuário correspondente no banco de dados. O objeto User do SQLAlchemy é então "injetado" na variável current_user da rota. Se qualquer uma dessas etapas falhar significa que o token está ausente, é inválido ou expirou, logo, a dependência get_current_user levanta uma exceção HTTPException 401 (Não Autorizado) imediatamente, bloqueando a requisição antes que qualquer lógica de negócios seja executada e garantindo a segurança de todos os endpoints protegidos.

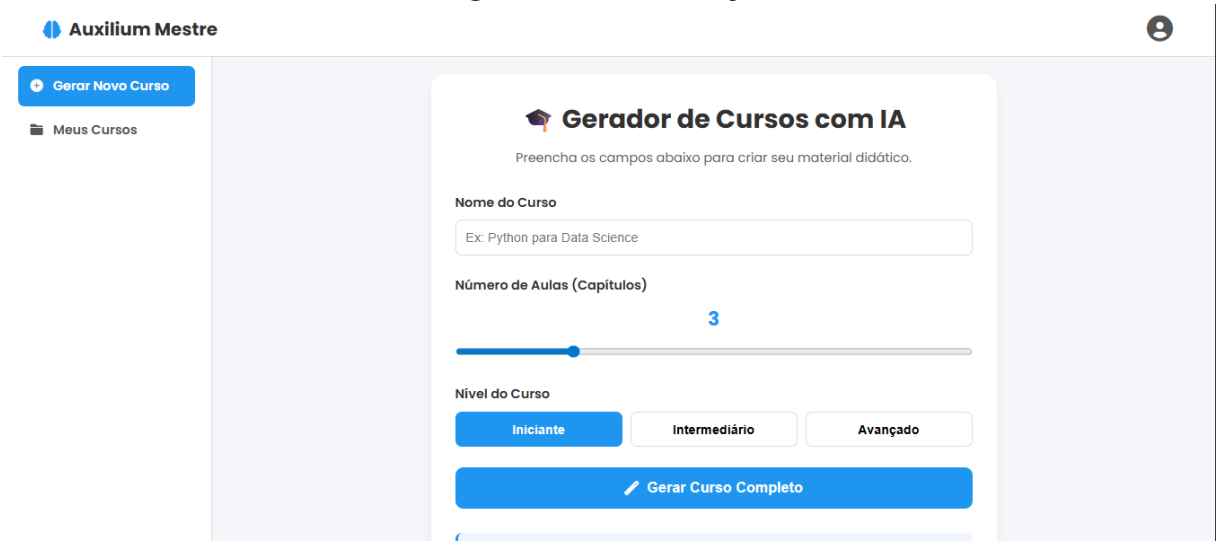
4.2.3. Módulo de Gerenciamento de Cursos

O Módulo de Gerenciamento de Cursos é o componente central da lógica de negócios do *Auxilium Mestre*. Implementado no arquivo routes.py, este módulo é inteiramente protegido pela dependência get_current_user (descrita na Seção 4.2.2) e é responsável por executar o ciclo de vida completo dos materiais educacionais, desde a sua criação até a exclusão. Ele implementa diretamente os requisitos RF03, RF04, RF05 e RF06.

O endpoint POST /api/generate-course é o orquestrador principal e a implementação direta do RF03, que define a Geração de Curso. Esta rota, definida como assíncrona (async def) para atender ao RNF03, recebe do frontend (Figura 5) um objeto JSON contendo o tópico, nível e número de capítulos, validado pelo schema Pydantic CourseRequest. A rota então executa o pipeline de geração:

- (1) chama o CourseGeneratorService para interagir com a API do Google Gemini e gerar a ementa e os textos das aulas;
- (2) salva a entidade Course no banco de dados, vinculando-a ao current_user.id;
- (3) itera sobre o conteúdo gerado, chamando os serviços DocxGenerator, PptxGeneratorService e PdfGeneratorService para criar os arquivos físicos (DOCX, PPTX, PDF);
- (4) salva os registros de cada arquivo na tabela GeneratedFile, vinculando-os ao course_id recém-criado.

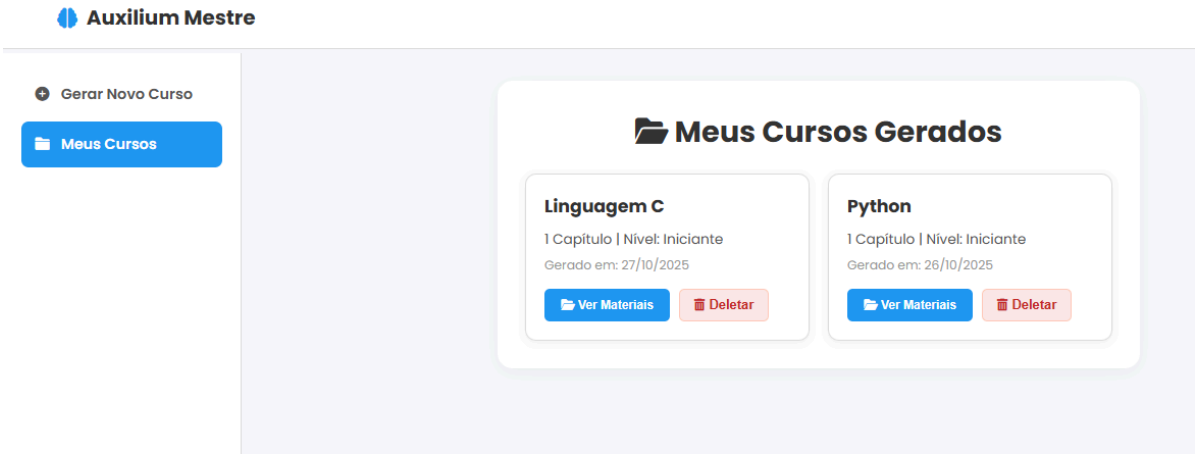
Figura 5 - Interface Principal



Fonte: O autor (2025)

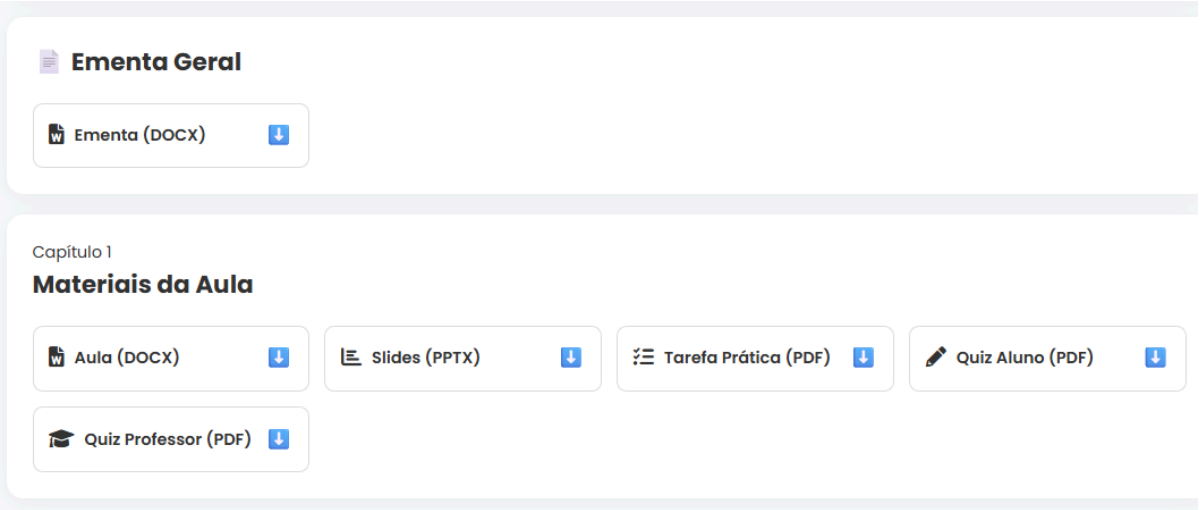
Uma vez que os cursos são gerados, o frontend os exibe utilizando dois endpoints de listagem. O primeiro, GET /api/courses, implementa o RF04. Ele consulta a tabela Course e retorna uma lista de todos os cursos onde o user_id corresponde ao current_user.id, garantindo o isolamento de dados (RNF01) (Figura 6). O segundo endpoint, GET /api/courses/{id}/files, implementa parte do RF06. Quando o usuário seleciona um curso, este endpoint é chamado para consultar a tabela GeneratedFile e retornar a lista de todos os arquivos associados àquele course_id (Figura 7).

Figura 6 - Dashboard



Fonte: O autor (2025)

Figura 7 - Download de Arquivos

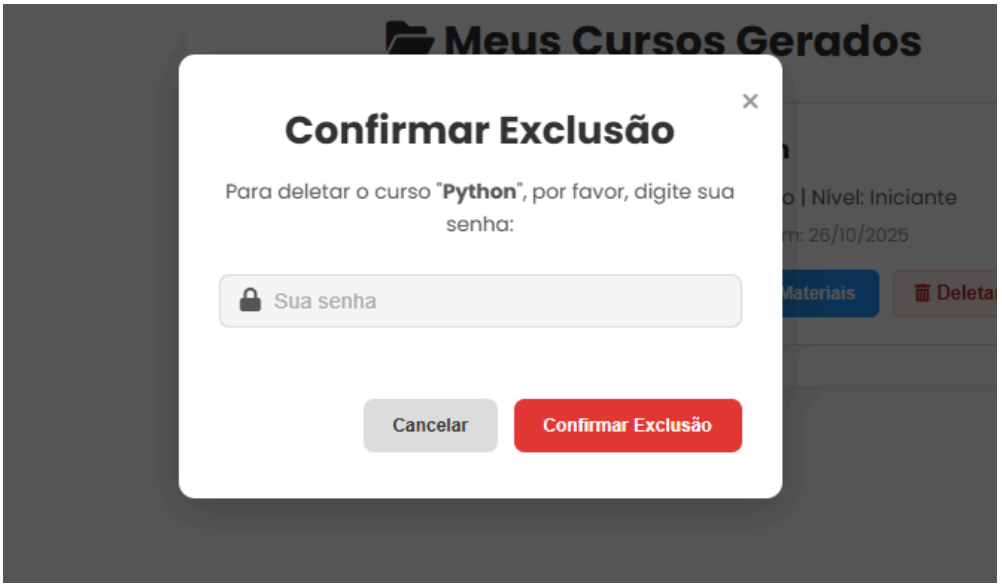


Fonte: O autor (2025)

Para implementar o RF06, de forma segura, foi criado o endpoint GET /api/download/{filename}. Este endpoint é protegido pela dependência get_current_user. Quando uma solicitação de download é recebida, o backend executa uma consulta que junta as tabelas GeneratedFile e Course. Ele então verifica se o filename solicitado existe e se o user_id associado ao curso daquele arquivo é o mesmo do current_user.id. Somente se ambas as condições forem verdadeiras, o backend retorna o arquivo físico do diretório output/ usando uma FileResponse, impedindo o acesso indevido a arquivos de outros usuários.

Por fim, o endpoint DELETE /api/courses/{id} implementa o RF05, que define a exclusão segura de curso (seção 3.2.1). Esta rota de alta sensibilidade exige uma confirmação de identidade. O frontend exibe um modal (ilustrado na Figura 8) que solicita a senha atual do usuário.

Figura 8 - Exclusão de Curso



Fonte: O autor (2025)

A requisição DELETE envia a senha em um corpo JSON, que é validado pelo modelo Pydantic PasswordValidationRequest. O backend primeiro verifica se a senha fornecida corresponde ao hashed_password do current_user (usando security.verify_password). Se a senha estiver correta, ele então prossegue para deletar os arquivos físicos associados do disco (usando os.remove) e, em seguida, remove os registros das tabelas GeneratedFile e Course do banco de dados, concluindo o ciclo de vida do curso.

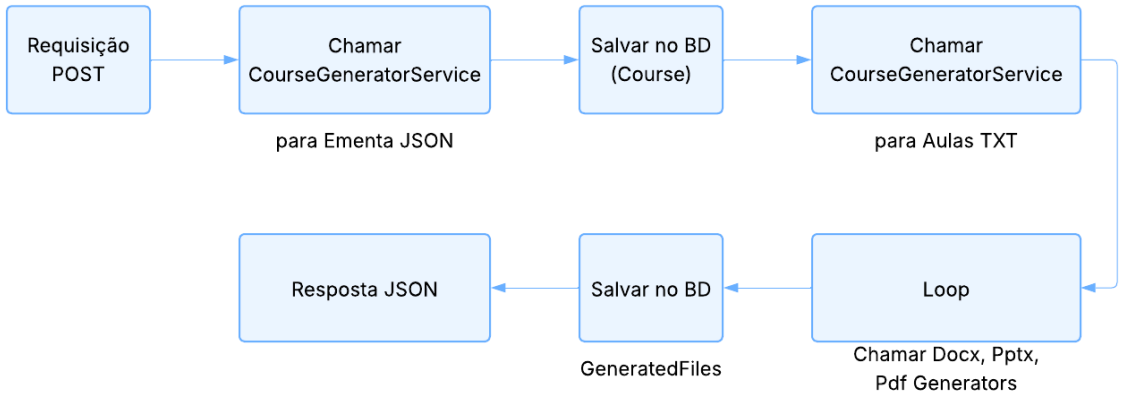
4.3. Módulo de Gerenciamento de Cursos

Os serviços de geração são os componentes responsáveis por executar o pipeline de criação de conteúdo (RF03). Esta lógica não é monolítica, mas sim orquestrada por uma única rota de API, conforme detalhado a seguir.

4.3.1. Orquestração da Geração (POST /api/generate-course)

O endpoint POST /api/generate-course, definido no arquivo routes.py, é o responsável por todo o processo de geração. Esta rota, protegida pela dependência get_current_user (conforme Seção 4.2.2) e implementada de forma assíncrona (async def), é a implementação direta do RF03. Ela recebe do frontend um objeto JSON, validado pelo Pydantic CourseRequest, contendo o tópico, nível e número de capítulos. A partir daí, o endpoint executa um pipeline de orquestração em uma sequência estrita, conforme ilustrado na Figura 9.

Figura 9 - Fluxograma de Orquestração da Geração de Curso



Fonte: O autor (2025)

O processo de orquestração se inicia com a primeira chamada de I/O de rede, onde o endpoint aciona o CourseGeneratorService para interagir com a API do Google Gemini, utilizando a Engenharia de Prompt (Seção 2.1.2) para gerar a ementa estruturada em formato JSON. Após receber a ementa, o endpoint realiza a primeira operação de banco de

dados, criando, assim, o registro principal na tabela courses, vinculando-o ao user_id do usuário autenticado. Esta etapa é crucial para obter o course_id (Chave Primária) que será usado como Chave Estrangeira para todos os arquivos subsequentes.

Com o curso registrado, o pipeline prossegue para a geração do conteúdo principal. O CourseGeneratorService é chamado novamente, desta vez em um processo iterativo para gerar o conteúdo textual, que ficam inicialmente em arquivos .txt de cada capítulo individualmente. O endpoint então entra em um loop, processando cada arquivo .txt gerado. Dentro deste loop, os serviços de formatação (fundamentados na Seção 2.4) são acionados em sequência, dessa forma, o DocxGenerator é chamado para criar o .docx da aula; o PptxGeneratorService é chamado para criar os slides .pptx (que por sua vez aciona o ImageManagerService e o LLM para sumarização); e o PdfGeneratorService é chamado para gerar os quizzes e tarefas em .pdf.

Cada arquivo gerado com sucesso tem seu nome (filename) e tipo (file_type) registrados na tabela generated_files do banco de dados, vinculados ao course_id do curso pai. Todo este pipeline de múltiplas etapas é encapsulado em um bloco try...except...db.rollback(). Esta estrutura garante a atomicidade da transação, pois, se qualquer etapa do processo falhar, seja uma falha na API do Gemini, um erro de escrita no disco ou uma falha no gerador de PPTX, o db.rollback() é chamado, desfazendo o registro do curso no banco de dados. Isso impede a criação de cursos incompletos, de cursos com erros ou registros de dados inconsistentes na aplicação.

4.3.2. Geração de Documentos

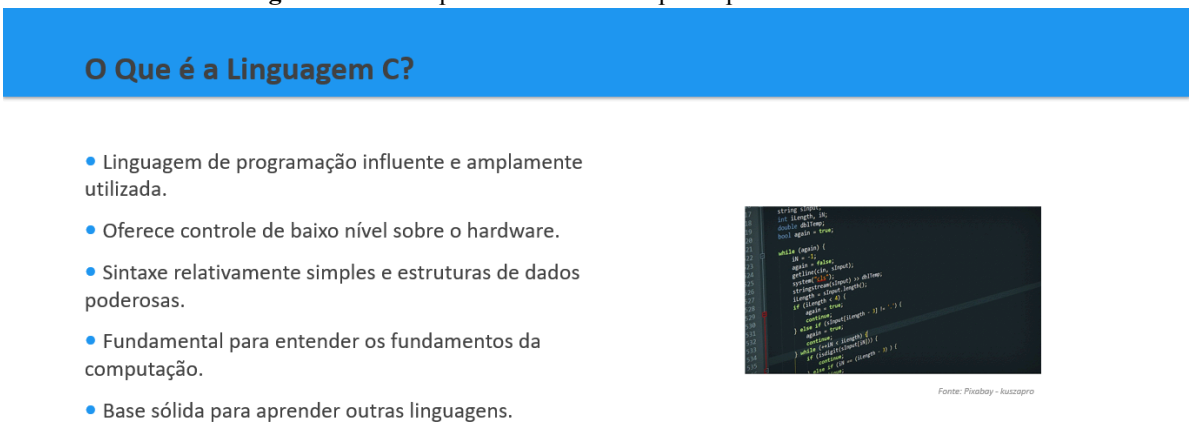
Uma vez que o pipeline de orquestração (Seção 4.3.1) obtém o conteúdo bruto da IA (arquivos JSON e .txt), ele aciona os serviços de geração de documentos. Cada serviço é uma classe Python especializada (conforme fundamentado na Seção 2.4) que resolve um desafio de implementação específico para traduzir texto em um formato de arquivo complexo.

O desafio na geração de documentos .docx, tratado pelo DocxGenerator, foi a interpretação do formato Markdown gerado pela IA. O CourseGeneratorService foi instruído via prompt a usar ## para subtítulos, * para listas e ** para negrito. Para converter isso, a função convert_lesson_txt_to_docx foi implementada como um parser simples. Ela lê o arquivo .txt linha por linha, utiliza expressões regulares e métodos de string para detectar essa sintaxe. Esta implementação traduz o conteúdo textual simples da IA em um documento de aula formatado e estruturado.

A geração de arquivos .pptx apresentou o desafio mais complexo, exigindo uma

orquestração de múltiplos serviços. O PptxGeneratorService primeiro chama novamente o LLM para sumarizar o texto completo da aula em um roteiro JSON de bullet points. Em seguida, ele chama o ImageManagerService para buscar uma imagem relevante. O principal desafio de implementação foi o posicionamento e dimensionamento da imagem no slide. Para evitar distorção, a função _add_modern_content_slide não insere a imagem diretamente, ela utiliza a biblioteca PIL (Pillow) para ler as dimensões originais em pixels da imagem baixada. Com esses dados, ela calcula a proporção da imagem e a compara com a proporção do container de destino no slide. Somente após esse cálculo, ela define as dimensões finais em EMUs (a unidade interna do python-pptx), garantindo que a imagem seja redimensionada mantendo sua proporção original. Por fim, a imagem é centralizada programaticamente dentro do seu espaço de layout, resultando em um slide profissional e dinâmico, conforme ilustrado na Figura 10.

Figura 10 - Exemplo de Slide Gerado pelo PptxGeneratorService



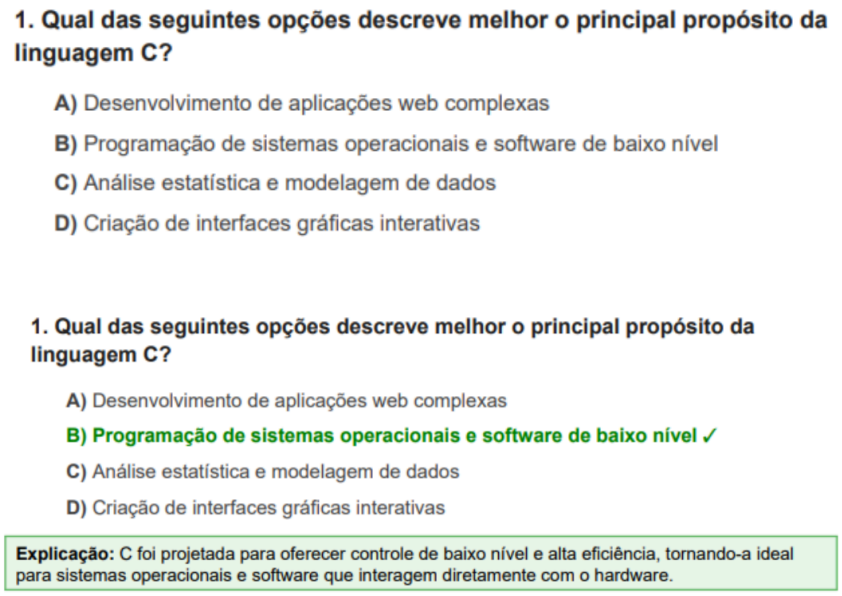
Fonte: O autor (2025)

Para a geração dos arquivos PDF, o PdfGeneratorService implementa um padrão de geração pareada. O desafio era criar materiais de avaliação, ou seja, um quiz para o aluno e um gabarito para o professor, a partir de uma única fonte de dados. A função create_assessment_pdfs primeiro chama o LLM para gerar um único objeto JSON contendo todas as perguntas, alternativas, a resposta_correta e a explicação.

Em seguida, este JSON é passado para duas funções de construção distintas do ReportLab, sendo elas, (1) _create_quiz_student_pdf, que itera pelo JSON e omite as respostas e explicações; e (2) _create_quiz_teacher_pdf, que itera pelos mesmos dados, mas

usa lógica condicional para destacar a resposta_correta e incluir o campo explicação. Esta implementação demonstra um pipeline eficiente de um-para-muitos, onde uma única chamada de IA produz dois artefatos distintos e complementares, como visto na Figura 11.

Figura 11 - Comparativo do Quiz do Aluno (acima) e Quiz do Professor (abaixo)



Fonte: O autor (2025)

4.4. Implementação do Frontend

A implementação do frontend, a interface, conforme fundamentado na Seção 2.2.2, foi desenvolvido utilizando a abordagem Vanilla JS (JavaScript puro), HTML5 e CSS3, sem depender de frameworks externos. Esta camada consome a API RESTful do backend (FastAPI) de forma assíncrona, através da Fetch API, para realizar todas as operações.

4.4.1. Implementação do Frontend

A estrutura de navegação da aplicação é dividida em dois segmentos principais, sendo um deles o site público, para marketing e autenticação, e a aplicação privada, o dashboard de geração.

O primeiro segmento é composto por páginas HTML estáticas. O arquivo index.html atua como a landing page do projeto, conforme ilustrado na Figura 12. Sua única função é apresentar a proposta de valor da ferramenta e direcionar usuários não autenticados para as rotas de login ou cadastro, através dos links para /login e /signup.

Figura 12 - Interface da Landing Page

Fonte: O autor (2025)

As páginas login.html e signup.html servem como os portais de autenticação, implementando as interfaces para os requisitos RF01 e RF02, conforme já ilustrado nas Figuras 3 e 4 (Seção 4.2.2). Cada uma dessas páginas importa seu próprio arquivo JavaScript (login.js ou signup.js) para validar os formulários e executar as chamadas de API (fetch) para os endpoints de autenticação.

O segundo segmento é o núcleo da aplicação, o arquivo generator.html. Este arquivo é a implementação da arquitetura Single Page Application (SPA). Após o login bem-sucedido, o usuário é redirecionado para esta única página, que nunca mais é recarregada. O generator.html atua como um contêiner, conforme conceito de Oliveira (2023), para as diferentes telas da aplicação, que são, na verdade, elementos com a classe .view.

A navegação dentro da SPA é inteiramente controlada pelo JavaScript (principalmente generator.js), conforme ilustrado na Figura 6. O generator.html define a estrutura visual principal, o cabeçalho (main-header), a barra de navegação lateral (app-sidebar) e a área de conteúdo principal (app-main). O generator.js anexa event listeners aos botões da barra lateral, como o **Gerar Novo Curso** e **Meus Cursos**. Quando um usuário clica em um desses botões, o JavaScript não recarrega a página; ele simplesmente altera as classes de CSS, escondendo a div.view atual e exibindo a div.view de destino. Todo o conteúdo dinâmico é gerado pelo JavaScript, que chama a Fetch API para buscar os dados do backend e, em seguida, manipula o DOM para construir e injetar o HTML necessário na view ativa.

4.4.2. Gerenciamento de Estado e Comunicação

A implementação do frontend como uma Single Page Application (SPA), descrita na seção anterior, exige um mecanismo robusto para gerenciar o estado de login do usuário e para se comunicar de forma segura com a API backend. No *Auxilium Mestre*, esta responsabilidade é dividida entre os arquivos profile.js e generator.js, e o pilar central de todo o processo é o JSON Web Token (JWT), fundamentado na Seção 2.3.3.

O gerenciamento de estado começa no momento do login (RF02). Após o usuário se autenticar, o backend retorna um access_token (JWT). O frontend, através de login.js ou auth_callback.js, tem a responsabilidade única de salvar este token no localStorage do navegador. O localStorage funciona como o armazenamento persistente da sessão do usuário no cliente.

A partir desse ponto, o arquivo profile.js assume o controle de autenticação da SPA. Este script é carregado em todas as páginas privadas, especificamente, generator.html. Sua primeira ação é executar um guarda de rota (route guard) do lado do cliente, implementando o RNF01. Ele verifica imediatamente se o token de acesso existe no localStorage. Se o token não for encontrado, o script interrompe a renderização da página e redireciona o usuário de volta para /login, protegendo efetivamente a aplicação contra acesso não autenticado.

O profile.js também é responsável por centralizar a lógica de comunicação com o backend. Ele expõe uma função utilitária global (window.getAuthHeader) que lê o token do localStorage e o formata corretamente no cabeçalho HTTP Authorization. Este mecanismo é a base para todas as comunicações seguras subsequentes:

Código-Fonte 2 - Função Utilitária para Cabeçalho de Autenticação

```
// Obtém o token JWT do localStorage.
window.getToken = function() {
    return localStorage.getItem('accessToken');
}

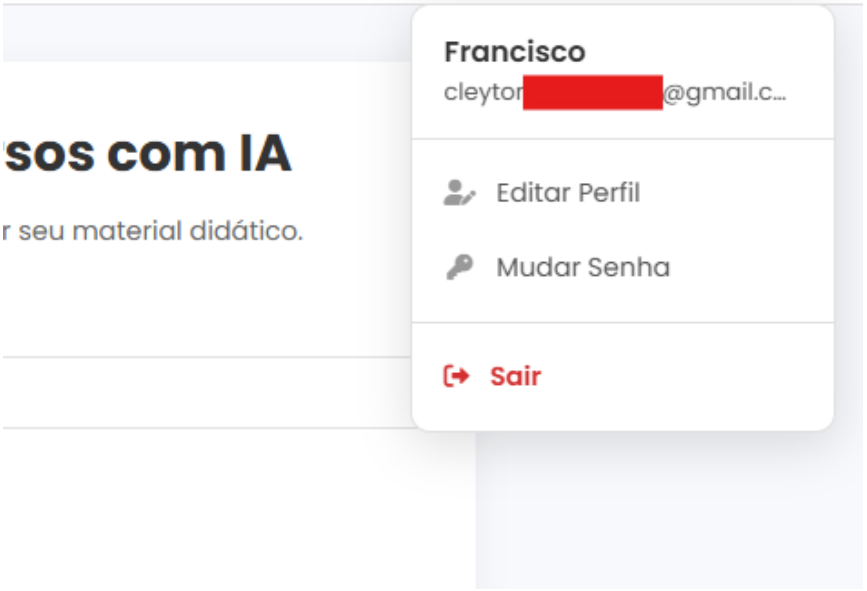
// Cria o objeto de cabeçalho Authorization para requisições
fetch.
window.getAuthHeader = function() {
    const token = window.getToken();
    return token ? { 'Authorization': `Bearer ${token}` } : {};
}
```

Fonte: O autor (2025)

Com o estado de login armazenado e as ferramentas de comunicação prontas, o

profile.js executa a função loadUserProfile(), que faz uma chamada fetch assíncrona ao endpoint protegido GET /api/users/me, utilizando o cabeçalho de autorização, conforme Código-Fonte 2. O backend (FastAPI) utiliza a dependência get_current_user (Seção 4.2.2) para validar o token e retornar os dados do usuário. O JavaScript, ao receber o JSON de resposta, manipula o DOM para preencher dinamicamente o menu de perfil com o nome e o e-mail do usuário logado, como ilustrado na Figura 13.

Figura 13 - Menu de Perfil Preenchido com Dados do Usuário



Fonte: O autor (2025)

O arquivo generator.js consome essas funcionalidades para executar a lógica de negócios. Ele não gerencia o estado de login diretamente, em vez disso, ele utiliza as funções globais fornecidas pelo profile.js. Todas as chamadas de API feitas pelo generator.js utilizam a função window.getAuthHeader() para se autenticar. O gerenciamento de estado se completa com a função window.logout(), que simplesmente remove o accessToken do localStorage e redireciona o usuário para a página inicial, encerrando a sessão.

4.4.3. Componentes Interativos

Além da estrutura de navegação, a implementação do frontend depende de componentes interativos em JavaScript para gerenciar o estado da interface (UI) e exibir dados dinamicamente. Esta lógica está centralizada nos arquivos profile.js e generator.js.

O **menu de perfil** (visto na Figura 13 da Seção 4.4.2) é um exemplo de componente de UI reativo. Sua implementação, contida no profile.js, não depende de recarregamento de página. Um event listener é anexado ao ícone do avatar #profile-avatar-btn. Quando o usuário clica neste ícone, o JavaScript simplesmente alterna entre adiciona ou

remove a classe .active do elemento de dropdown #profile-dropdown-menu.

De forma similar, o **modal de exclusão** (visto na Figura 8 da Seção 4.2.3) implementa o requisito de segurança RF05. No entanto, ele é mais complexo que o menu de perfil, pois precisa gerenciar dados. A função openDeleteModal(courseId, courseName) no generator.js não apenas alterna a classe .show do modal para exibi-lo, ela também passa dados para o modal. O JavaScript manipula o DOM para injetar dinamicamente o nome do curso no elemento #modal-course-name, garantindo que o usuário saiba exatamente o que está excluindo. A função confirmDeletion então lê o valor do campo de senha e o ID do curso para construir a requisição DELETE para a API.

O componente mais complexo é a **exibição dinâmica dos resultados**, que implementa o núcleo da arquitetura SPA. As views **Meus Cursos** e **Resultados** contêm contêineres vazios no HTML, que quando o usuário navega para **Meus Cursos**, a função loadMyCourses dispara, executando uma fetch para a API (GET /api/courses). Ao receber o array JSON de cursos, o JavaScript não recarrega a página; ele itera sobre esse array.

Para cada item do array, o script cria novos elementos HTML em tempo real usando document.createElement('div') e define suas propriedades. O conteúdo do card é então construído e injetado usando a propriedade .innerHTML. Finalmente, container.appendChild(courseCard) é chamado para adicionar o card recém-criado ao DOM, fazendo-o aparecer na tela. Este processo é o que torna a aplicação reativa, construindo a interface dinamicamente a partir dos dados da API.

5. RESULTADOS E AVALIAÇÃO

Este capítulo apresenta os resultados práticos obtidos com a implementação da plataforma *Auxilium Mestre*. O objetivo é validar o sistema, demonstrar o cumprimento dos Requisitos Funcionais (RFs) e Requisitos Não Funcionais (RNFs) definidos na Seção 3.2, e avaliar a qualidade dos artefatos gerados. Para isso, foi conduzido um estudo de caso prático.

5.1. Estudo de Caso 1: Geração de Curso Técnico (Python para Iniciantes)

Para validar a funcionalidade central do *Auxilium Mestre*, foi realizado um estudo de caso simulando o fluxo completo de um educador. O objetivo deste primeiro teste foi gerar um curso de domínio técnico, **Python para Iniciantes**, configurado para o nível **Iniciante** e com uma estrutura de **3 capítulos**.

O primeiro passo do estudo de caso foi a interação com a interface de geração para executar o RF03, que define a geração de curso. O usuário autenticado navegou até a

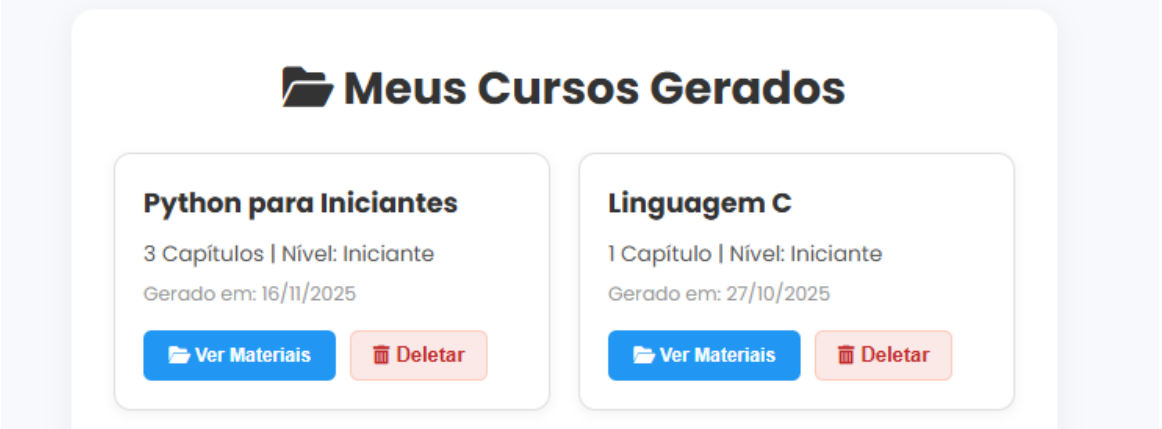
view **Gerar Novo Curso** (generator.html) e preencheu o formulário de entrada com os parâmetros definidos, conforme ilustrado na Figura 14.

Figura 14 - Preenchimento do Formulário para Geração do Estudo de Caso

Fonte: O autor (2025)

Após a submissão, a aplicação exibiu a view de processamento (conforme Seção 4.4.1) enquanto o backend executava o pipeline de orquestração (Seção 4.3.1). Ao término da geração, o frontend foi atualizado e o usuário navegou para a aba **Meus Cursos**, implementando o RF04, listagem de cursos. A Figura 15 demonstra o resultado, o curso **Python para Iniciantes** foi criado com sucesso, persistido no banco de dados e agora é listado no dashboard do usuário, validando o cumprimento do RF04 e do RNF01, isolamento de dados.

Figura 15 - Dashboard Meus Cursos Exibindo o Curso Gerado (RF04)

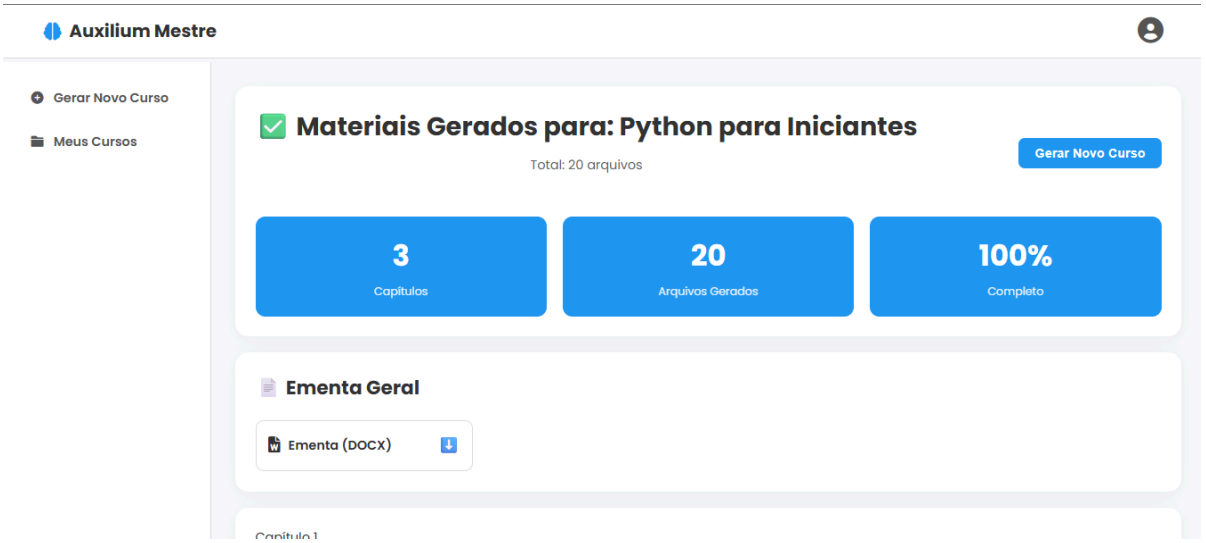


Fonte: O autor (2025)

O segundo passo do estudo de caso foi a verificação dos artefatos gerados, implementando o RF06, download de materiais. Ao clicar no botão **Ver Materiais** (Figura 6),

o frontend acionou o endpoint GET /api/courses/{id}/files e renderizou a view de resultados, detalhada na Figura 16. Esta tela comprova que o pipeline de backend executou com sucesso, gerando e registrando um conjunto completo de 20 arquivos para os 3 capítulos, sendo eles, 1 Ementa DOCX, 3 Aulas TXT, 3 Aulas DOCX, 3 Slides PPTX, 3 Quizzes Aluno PDF, 3 Quizzes Professor PDF, 3 Tarefas PDF e 1 Ementa JSON.

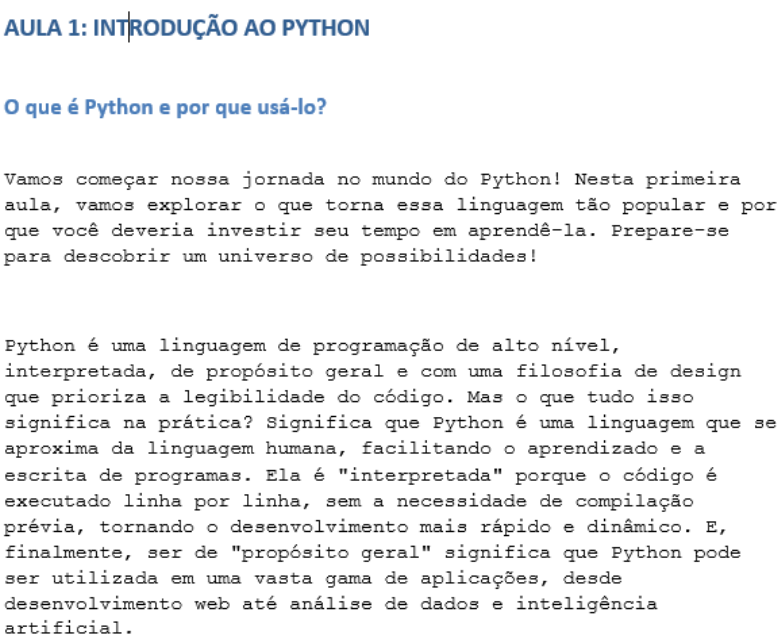
Figura 16 - Preenchimento do Formulário para Geração do Estudo de Caso



Fonte: O autor (2025)

Para avaliar a qualidade e a coesão do conteúdo gerado, os principais arquivos foram abertos e inspecionados. A Figura 17 apresenta um trecho do documento de aula do Capítulo 1. Esta imagem valida a implementação do DocxGenerator (Seção 4.3.2), demonstrando a correta conversão da sintaxe Markdown, usada pela IA, em um documento .docx formatado, com títulos (##), listas (*) e negrito (**) devidamente aplicados

Figura 17 - Trecho do Documento de Aula (.docx) Gerado



AULA 1: INTRODUÇÃO AO PYTHON

O que é Python e por que usá-lo?

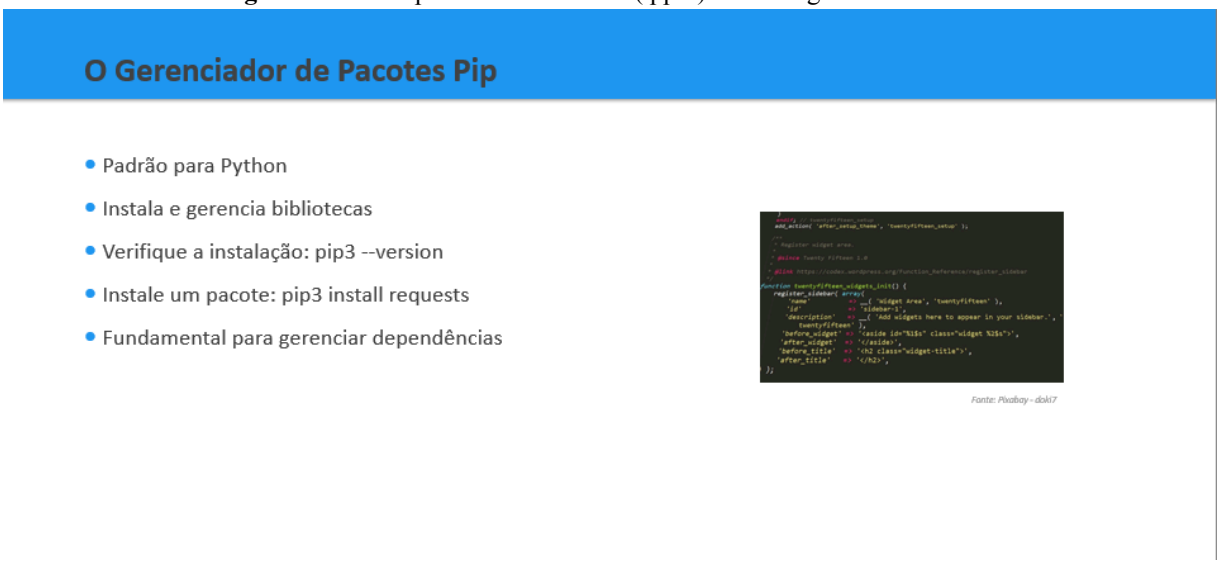
Vamos começar nossa jornada no mundo do Python! Nesta primeira aula, vamos explorar o que torna essa linguagem tão popular e por que você deveria investir seu tempo em aprendê-la. Prepare-se para descobrir um universo de possibilidades!

Python é uma linguagem de programação de alto nível, interpretada, de propósito geral e com uma filosofia de design que prioriza a legibilidade do código. Mas o que tudo isso significa na prática? Significa que Python é uma linguagem que se aproxima da linguagem humana, facilitando o aprendizado e a escrita de programas. Ela é "interpretada" porque o código é executado linha por linha, sem a necessidade de compilação prévia, tornando o desenvolvimento mais rápido e dinâmico. E, finalmente, ser de "propósito geral" significa que Python pode ser utilizada em uma vasta gama de aplicações, desde desenvolvimento web até análise de dados e inteligência artificial.

Fonte: O autor (2025)

A Figura 18 exibe um slide da apresentação (.pptx) gerada para o Capítulo 1. Esta imagem é uma evidência crucial do sucesso da orquestração de múltiplos serviços, o PptxGeneratorService (Seção 4.3.2) não apenas utilizou o LLM para sumarizar o conteúdo da aula em tópicos, mas também acionou o ImageManagerService para buscar uma imagem de contexto. A imagem foi corretamente dimensionada e posicionada, resolvendo o desafio de engenharia de layout dinâmico.

Figura 18 - Exemplo de Slide Gerado (.pptx) com Imagem de Contexto

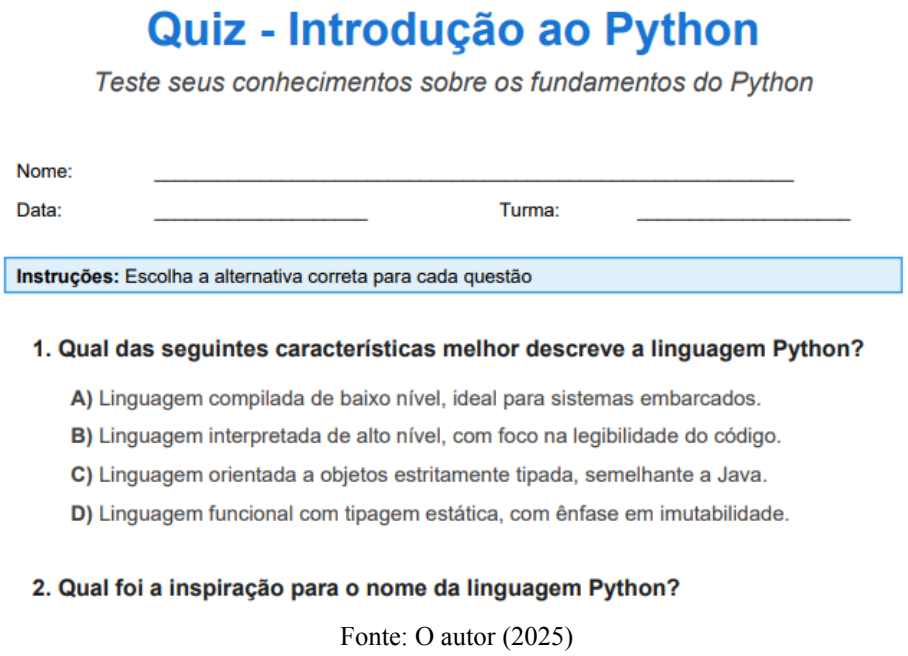


Fonte: O autor (2025)

Por fim, a Figura 19 demonstra o resultado do PdfGeneratorService (Seção 4.3.2), exibindo uma questão do **Quiz Aluno** (.pdf) gerado para o Capítulo 1. A análise do arquivo

valida a capacidade da IA de criar avaliações pertinentes ao tópico (sintaxe de Python) e a capacidade da biblioteca ReportLab de formatar essas questões em um documento PDF pronto para aplicação.

Figura 19 - Trecho do Quiz de Aluno (.pdf) Gerado



Quiz - Introdução ao Python

Teste seus conhecimentos sobre os fundamentos do Python

Nome: _____

Data: _____ Turma: _____

Instruções: Escolha a alternativa correta para cada questão

1. Qual das seguintes características melhor descreve a linguagem Python?

A) Linguagem compilada de baixo nível, ideal para sistemas embarcados.

B) Linguagem interpretada de alto nível, com foco na legibilidade do código.

C) Linguagem orientada a objetos estritamente tipada, semelhante a Java.

D) Linguagem funcional com tipagem estática, com ênfase em imutabilidade.

2. Qual foi a inspiração para o nome da linguagem Python?

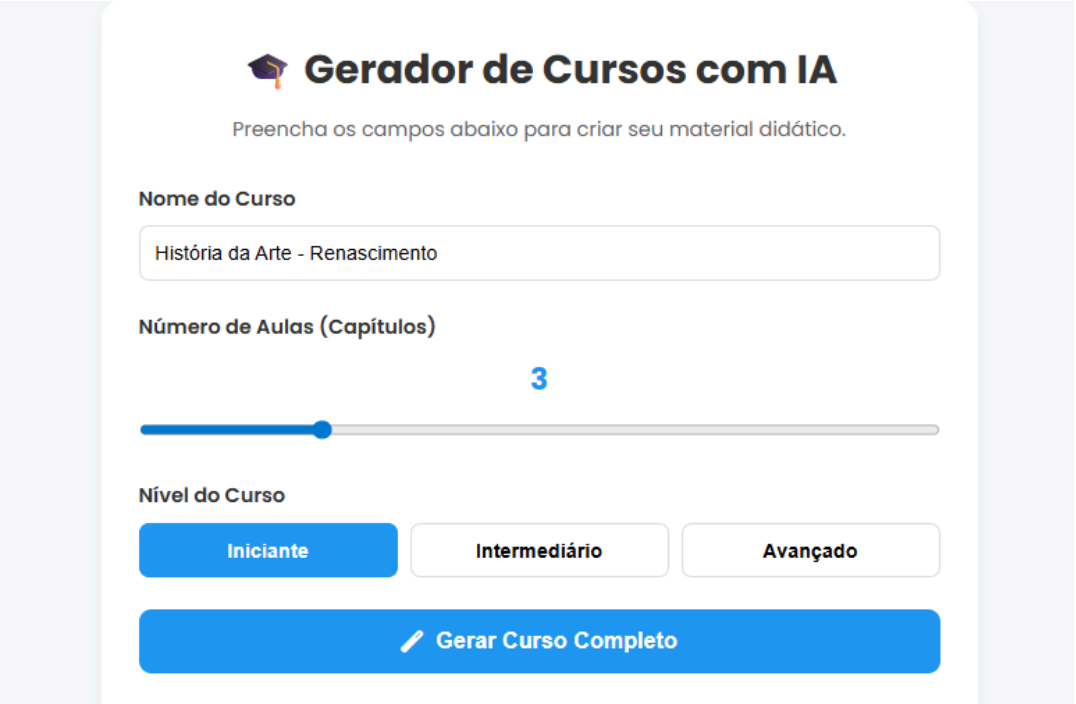
Fonte: O autor (2025)

5.2. Estudo de Caso 2: Geração de Curso Humanas (História da Arte)

Para avaliar a flexibilidade da arquitetura (RNF02) e demonstrar que o *Auxilium Mestre* não está limitado a domínios técnicos, foi conduzido um segundo estudo de caso. O objetivo foi testar o pipeline de geração com um tópico do campo das humanidades: História da Arte - Renascimento.

O sistema foi configurado com os mesmos parâmetros do estudo de caso anterior (Nível Iniciante, 3 Capítulos), para permitir uma comparação direta da qualidade da saída. Assim como no primeiro estudo, o usuário autenticado preencheu o formulário da generator-view, conforme Figura 20. O backend recebeu a requisição POST /api/generate-course e orquestrou o pipeline de geração.

Figura 20 - Parâmetros de Entrada para o Estudo de Caso 2

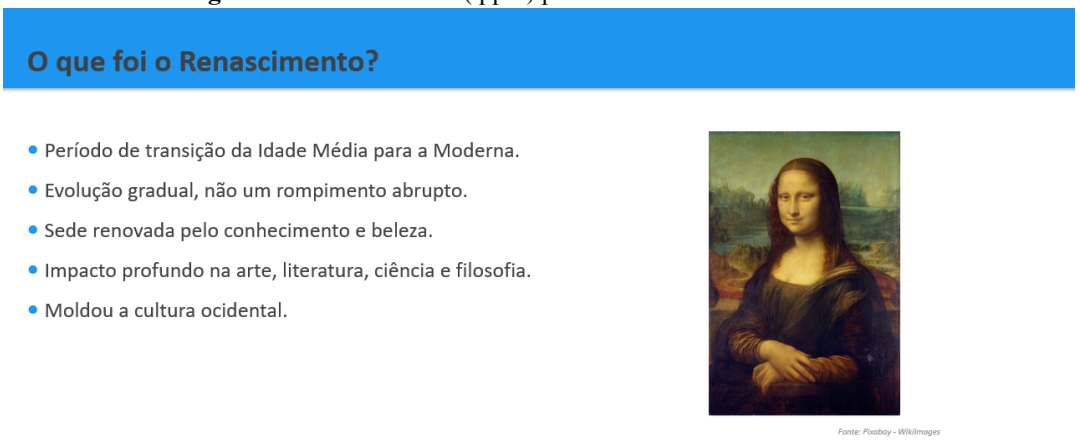


Fonte: O autor (2025)

O resultado deste estudo de caso é a evidência mais importante da flexibilidade do sistema, pois o mesmo código de backend produziu resultados drasticamente diferentes, adaptados ao novo domínio. A Figura 21 é a prova principal.

No Estudo de Caso 1, o PptxGeneratorService (Seção 4.3.2) identificou a necessidade de imagens técnicas e o ImageManagerService buscou trechos de código. Neste estudo de caso, para o tópico Principais Artistas do Renascimento, o pipeline comportou-se de forma diferente, pois o LLM (Gemini) sumarizou o conteúdo e instruiu que uma imagem de Mona Lisa era necessária e, assim, o ImageManagerService interpretou este prompt e buscou com sucesso a obra de arte correspondente nos bancos de imagem.

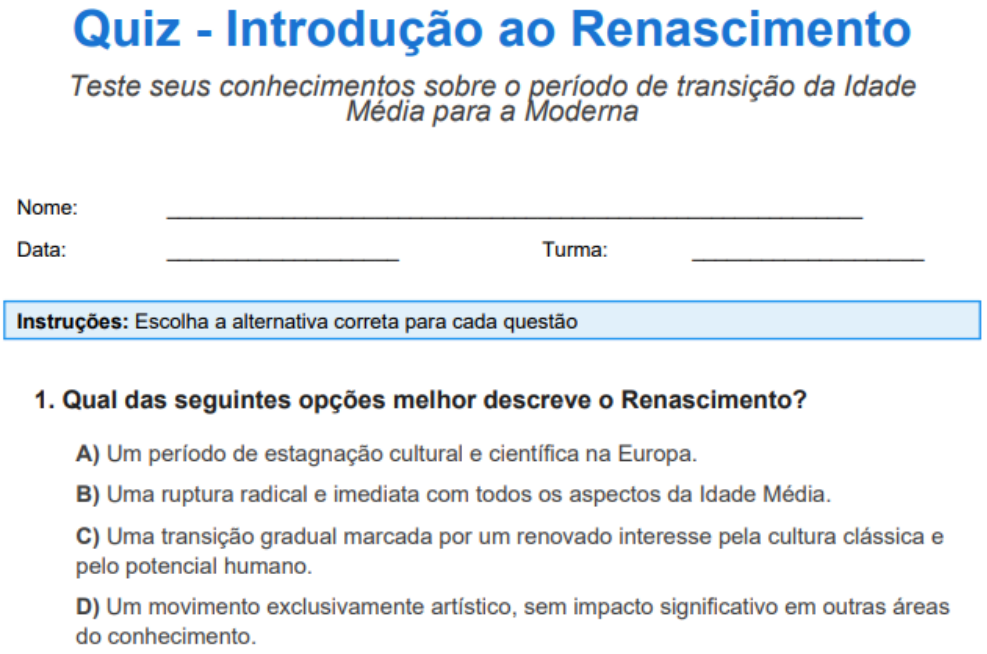
Figura 21 - Slide Gerado (.pptx) para o curso de Humanidades



Fonte: O autor (2025)

Da mesma forma, a flexibilidade foi validada nos materiais de avaliação. O PdfGeneratorService (Seção 4.3.2), que no caso anterior gerou questões sobre sintaxe de Python, agora gerou questões pertinentes ao domínio das humanidades. A Figura 22 ilustra uma questão de múltipla escolha extraída do Quiz Aluno (.pdf) gerado, onde o LLM criou uma pergunta conceitual sobre o período artístico.

Figura 22 - Slide Gerado (.pptx) para o curso de Humanidades



Fonte: O autor (2025)

Este estudo de caso comprova que a arquitetura do *Auxilium Mestre*, ao delegar a geração de conteúdo a um LLM generalista e a formatação a bibliotecas de software, funciona como uma verdadeira plataforma de automação, capaz de lidar com domínios técnicos (Engenharia de Computação), de humanidades (História da Arte), ou qualquer outra área, com a mesma eficiência.

5.3. Avaliação da Aplicação

A execução dos estudos de caso (Seções 5.1 e 5.2) permitiu a validação prática do *Auxilium Mestre*. Esta seção apresenta uma avaliação crítica dos resultados, dividida em três eixos:

- A conformidade funcional com os requisitos;
- A qualidade da usabilidade e dos materiais gerados;
- As limitações inerentes ao sistema.

5.3.1. Análise Funcional

A análise funcional tem como objetivo confirmar, de forma objetiva, se todos os Requisitos Funcionais (RFs) definidos na Seção 3.2.1 foram atendidos com sucesso. Esta atividade de avaliação, formalmente conhecida como Teste Funcional, é um processo de teste de software onde que verifica se as funcionalidades de uma aplicação funcionam conforme o esperado, com base nos requisitos especificados (POWELL; SMALLEY, 2025).

Os requisitos RF01 (Cadastro) e RF02 (Autenticação) foram validados como um pré-requisito para ambos os estudos de caso. Foi necessário criar uma conta, verificá-la por e-mail e realizar o login para acessar o dashboard. A capacidade de listar cursos (RF04) e acessar arquivos (RF06) restritos apenas ao usuário logado confirma que a implementação da dependência get_current_user e o user_id no banco de dados (Seção 4.2.1) atendem com sucesso ao RNF01 (Isolamento de Dados).

Mais importante, a plataforma demonstrou flexibilidade de domínio. O pipeline de orquestração se mostrou agnóstico ao tópico, onde no Estudo de Caso 1, com a geração do curso Python para Iniciantes, ele gerou slides técnicos (Figura 18), identificando a necessidade de uma imagem de código. No Estudo de Caso 2, História da Arte, o mesmo pipeline gerou slides de humanidades (Figura 21), identificando a necessidade de uma obra de arte.

Isso valida a premissa de que a arquitetura (FastAPI + LLM + Geradores) funciona como uma ferramenta de alavancagem, capaz de criar conteúdo coeso e relevante, alinhando-se ao potencial dos LLMs na educação (KASNECI et al., 2023).

O requisito RF03 (Geração de Curso) foi o foco principal de ambos os estudos de caso. As Figuras 14 e 20 demonstram que a interface de entrada comunicou-se com sucesso com o endpoint /api/generate-course. O backend foi capaz de orquestrar o pipeline completo e gerar os artefatos para ambos os domínios (técnico e humanidades), validando o RF03.

O requisito RF04 (Listagem de Cursos) foi validado pela Figura 15, que demonstrou que o curso gerado no RF03 foi corretamente persistido no banco de dados e exibido na lista do usuário.

O requisito RF06 (Download de Materiais) foi validado pelas Figuras 16 (Lista de Arquivos), 17 (.docx), 18 (.pptx) e 19 (quiz). O frontend foi capaz de buscar (via GET /api/courses/{id}/files) e exibir a lista de arquivos. O backend foi capaz de servir os arquivos de forma segura.

Por fim, o requisito RF05 (Exclusão Segura) foi validado em um teste funcional. A interface (Figura 8) coletou a senha do usuário, que foi enviada ao endpoint DELETE

/api/courses/{id}. O backend validou a senha (RNF01) e removeu com sucesso todos os registros do banco de dados e arquivos físicos do servidor.

Conclui-se, portanto, que todos os Requisitos Funcionais definidos para o *Auxilium Mestre* foram implementados e validados com sucesso.

5.3.2. Limitações do Sistema

A primeira e mais significativa limitação é a qualidade do conteúdo, onde pode-se ter uma preocupação. Visto que, o sistema não entende o conteúdo; ele é um orquestrador, a qualidade dos materiais gerados é inteiramente dependente da eficácia da Engenharia de Prompt (Seção 2.1.2). Os prompts que geram a ementa, as aulas e os quizzes estão hard-coded (fixos) nos serviços do backend. Uma mudança na API do Google Gemini ou um tópico muito complexo pode resultar em saídas de baixa qualidade, JSON malformado ou falhas de formatação que o sistema não sabe corrigir.

A segunda limitação é o tempo de geração. A implementação do RF03, embora assíncrona no nível do servidor (ASGI), é uma operação síncrona do ponto de vista do usuário. O processo (POST /api/generate-course) envolve múltiplas chamadas de rede sequenciais para a API do Gemini, seguidas de múltiplas operações de escrita em disco. Em nossos testes, a geração de um curso de 3 capítulos levou cerca de 5 minutos. Durante esse tempo, o usuário fica em uma tela de processamento, o que pode ser uma experiência de usuário (UX) ruim para cursos maiores. Uma arquitetura mais robusta implementaria background tasks com WebSockets ou polling para notificar o usuário quando o curso estivesse pronto.

A terceira limitação é a dependência de APIs externas e custos. O *Auxilium Mestre* não é autônomo. Seu funcionamento depende criticamente de serviços de terceiros (Google Gemini, Brevo, Pexels). Se qualquer uma dessas APIs estiver fora do ar, apresentar lentidão, ou se as chaves de API atingirem seus limites de cota, a funcionalidade central da aplicação falhará, isso será um grande problema para a aplicação. Além disso, o uso em larga escala implica custos financeiros diretos associados ao uso dessas APIs.

Uma última limitação identificada é a robustez da formatação. Os serviços de geração (Seção 4.3.2) são parsers simples. O DocxGenerator, por exemplo, espera um Markdown simples. Se a IA do Gemini decidir gerar uma tabela em Markdown ou uma sintaxe de bloco de código mais complexa, o parser atual falhará ou formatará o documento incorretamente.

6. CONCLUSÃO

Este Trabalho de Conclusão de Curso partiu da contextualização de uma demanda educacional crescente, a sobrecarga de trabalho dos educadores na criação de materiais didáticos digitais, que foi intensificado pela transformação digital. O Problema de Pesquisa identificou que, embora os Modelos de Linguagem de Larga Escala (LLMs) existam como ferramentas de auxílio, o processo de usá-los para criar um curso completo com ementa, aulas, slides e avaliações permanecia fragmentado, manual e tecnicamente complexo.

Diante disso, o Objetivo Geral do trabalho foi o desenvolvimento e a avaliação da plataforma *Auxilium Mestre*, uma aplicação web full-stack para geração automatizada de conteúdo educacional. Para atingir este objetivo, inicialmente foi desenvolvido um protótipo no Google Colab e, após isso, o desenvolvimento da aplicação.

O resultado prático foi a implementação de uma arquitetura full-stack completa, composta por um frontend reativo (SPA) em JavaScript "Vanilla", um backend de API robusto (FastAPI), protegido por um sistema de autenticação e verificação de e-mail; uma camada de persistência relacional e um pipeline de serviços de geração que orquestra o LLM (Google Gemini) com geradores de documentos. A validação funcional desta arquitetura foi demonstrada nos Estudos de Caso (Seções 5.1 e 5.2), que comprovaram a capacidade do sistema em gerar cursos coesos tanto para domínios técnicos quanto assuntos gerais como na área de humanidades.

6.1. Resposta à Pergunta de Pesquisa

A Pergunta de Pesquisa (Seção 1.3) que norteou este trabalho foi: "Como a integração de Modelos de Linguagem de Larga Escala (LLMs) em uma arquitetura web moderna (FastAPI, PostgreSQL, JavaScript) pode otimizar e escalar o processo de criação de materiais didáticos (ementas, aulas, apresentações e avaliações) para educadores, mantendo um alto nível de qualidade e coesão?"

Este trabalho responde a esta pergunta demonstrando que a integração é não apenas possível, mas altamente eficaz, através de três eixos de implementação, sendo eles, otimização, escalabilidade e qualidade.

A otimização foi alcançada ao mover o processo manual e fragmentado para o backend. A implementação do endpoint POST /api/generate-course (Seção 4.3.1) atua como um orquestrador que automatiza todo o pipeline. O usuário, que antes precisava executar dezenas de passos manuais, agora apenas fornece um tópico. A arquitetura de serviços assume a responsabilidade de chamar o LLM, gerar a ementa, gerar cada aula, gerar cada slide e gerar

cada avaliação, resolvendo o problema central da fragmentação.

A escalabilidade da solução é garantida pela arquitetura web moderna. O uso do FastAPI, baseado em ASGI (Seção 2.2.1), permite que o servidor lide com múltiplas requisições de geração concorrentes sem bloqueio. A arquitetura SaaS Multilocatário (Seção 3.1) e o banco de dados PostgreSQL (Seção 2.3.1) garantem que o sistema possa crescer em número de usuários, mantendo os dados de cada educador rigorosamente isolados (RNF01), conforme validado na implementação do módulo de segurança (Seção 4.2.2).

A coesão foi mantida ao utilizar a saída da primeira etapa, a ementa JSON serve como fonte para todas as etapas subsequentes. A aula é gerada a partir do texto da ementa; os slides são um resumo da aula e os quizzes são baseados no conteúdo da aula. Os Estudos de Caso (5.1 e 5.2) provaram que este pipeline é flexível, gerando conteúdo de qualidade e coeso tanto para Python quanto para História da Arte, validando a arquitetura como uma plataforma de propósito geral.

6.2. Trabalhos Futuros

Embora a plataforma tenha atingido com sucesso seus objetivos primários, as limitações abrem caminhos claros para trabalhos futuros. Entre eles:

- Implementar a geração de cursos (RF03) como uma tarefa de fundo (background task), desacoplando-a da requisição HTTP. Isso permitiria que o usuário fechasse o navegador e fosse notificado quando o curso estivesse pronto, melhorando drasticamente a experiência do usuário para cursos longos.
- Implementar as funcionalidades de gerenciamento de conta que foram deixadas de fora do escopo atual, como os endpoints e interfaces para Editar Perfil e Mudar Senha.
- Desenvolver uma interface de Configurações Avançadas onde o educador possa editar ou fornecer seus próprios prompts para os serviços de geração, em vez de depender dos prompts fixos (hard-coded) no backend.
- Permitir que a ementa JSON gerada pela IA seja editada pelo usuário antes da geração dos arquivos finais.
- Melhorar os parsers de geração (Seção 4.3.2) para suportar saídas de IA mais complexas, como tabelas em Markdown.
- Integrar o *Auxilium Mestre* com outras plataformas educacionais ou de produtividade, como o Microsoft Outlook/Office (para envio direto dos materiais) ou o Google Classroom.

7. REFERÊNCIAS

BALYER, A.; ÖZ, Ö. Academicians' views on digital transformation in education. **International Online Journal of Education and Teaching (IOJET)**, [S. l.], v. 5, n. 4, p. 809-830, 2018. Disponível em: https://eric.ed.gov/?id=EJ1250526. Acesso em: 03 nov. 2025.

BROOKINS, Andrew. Rewriting an API to Use FastAPI: Benchmarks and Lessons Learned. andrewbrookins.com, [S. l.], 23 maio 2021. Disponível em: https://andrewbrookins.com/python/is-fastapi-a-fad/. Acesso em: 12 nov. 2025.

BROWN, Tom B. et al. Language Models are Few-Shot Learners. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NEURIPS), 33., 2020, Vancouver. Anais... Vancouver: Curran Associates, 2020. p. 1877-1901. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html?utm_source=transaction&utm_medium=email&utm_campaign=linkedin_newsletter.4 Acesso em: 04 nov. 2025.

CANNY, Steve. python-docx Documentation. [S. l.], 2013a. Disponível em: https://python-docx.readthedocs.io/en/latest/. Acesso em: 14 nov. 2025.

CANNY, Steve. python-pptx Documentation. [S. l.], 2013b. Disponível em: https://python-pptx.readthedocs.io/en/latest/. Acesso em: 14 nov. 2025.

CHEN, J. Model Algorithm Research based on Python Fast API. Frontiers in Science and Engineering, [S. l.], v. 3, n. 9, p. 7-10, 2023. Disponível em: https://doi.org/10.54691/fse.v3i9.5591. Acesso em: 11 nov. 2025.

DEVLIN, Jacob et al. BERT: Pre-training of deep bidirectional transformers for language understanding. ArXiv, [S. l.], 2018. Disponível em: https://arxiv.org/abs/1810.04805. Acesso em: 09 nov. 2025.

FLORIDI, Luciano; CHIRIATTI, Massimo. GPT-3: Its nature, scope, limits, and consequences. Minds and Machines, [S. l.], v. 30, n. 4, p. 681-694, 2020. Disponível em: https://doi.org/10.1007/s11023-020-09548-1. Acesso em: 09 nov. 2025.

HODGES, Charles et al. The Difference Between Emergency Remote Teaching and Online Learning. EDUCAUSE Review, [S. l.], 27 mar. 2020. Disponível em: https://er.educause.edu/articles/2020/3/the-difference-between-emergency-remote-teaching-and-online-learning. Acesso em: 03 nov. 2025.

KASNECI, Enkelejda et al. ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, [S. l.], v. 103, p. 102274, 2023. Disponível em: https://doi.org/10.1016/j.lindif.2023.102274. Acesso em: 05 nov. 2025.

KUKA, Valeriia. Role Prompting: Guide LLMs with Persona-Based Tasks. Learn Prompting. [S. l.], 2024. Disponível em: https://learnprompting.org/docs/advanced/zero_shot/role_prompting. Acesso em: 10 nov. 2025.

LIU, Pengfei et al. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, [S. l.], v. 55, n. 9, p. 1-35, 2021.

Disponível em: https://arxiv.org/abs/2107.13586. Acesso em: 09 nov. 2025.

MARTIN, F.; XIE, K. Digital transformation in higher education: 7 areas for enhancing digital learning. EDUCAUSE Review, [S. l.], 2022. Disponível em: https://www.proquest.com/scholarly-journals/digital-transformation-higher-education-7-areas/docview/3225446859/se-2. Acesso em: 03 nov. 2025.

MDN WEB DOCS. Introdução ao DOM. [S. l.]: Mozilla, 2025a. Disponível em: https://developer.mozilla.org/pt-BR/docs/conflicting/Web/API/Document_Object_Model. Acesso em: 14 nov. 2025.

MICROSOFT. Arquitetar soluções multilocatários no Azure. [S. l.]: Microsoft, 2025. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/guide/multitenant/overview. Acesso em: 14 nov. 2025.

MICROSOFT. Prompt engineering techniques. [S. l.]: Microsoft, 2025. Disponível em: https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/prompt-engineering. Acesso em: 17 nov. 2025.

MIGUEL, Roberson. O que é ORM - Mapeamento objeto-relacional. Dev Community, [S. l.], 21 ago. 2022. Disponível em: https://dev.to/biosbug/o-que-e-orm-mapeamento-objeto-relacional-2m8n. Acesso em: 14 nov. 2025.

MIN, Bonan et al. Recent advances in natural language processing via large pre-trained language models: A survey. ArXiv, [S. l.], 2021. Disponível em: https://arxiv.org/abs/2111.01243. Acesso em: 06 nov. 2025.

NEVES, Vinicios. Autenticação moderna com OAuth 2.0 e OpenID Connect: dicas para desenvolvedores frontend. Alura, [S. l.], 20 maio 2024. Disponível em: https://www.alura.com.br/artigos/oauth-2-e-openid-connect. Acesso em: 14 nov. 2025.

OLIVEIRA, Manoel Marisergio Alves de. Integração de LPS com Microsserviços para o Desenvolvimento de SaaS Multilocatário: proposta de diretrizes para o projeto arquitetural com variabilidades implementadas por meio de microsserviços. 2023. 126 f. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Ciência da Computação, Universidade Federal Rural do Semi-Árido/Universidade Estadual do Rio Grande do Norte, Mossoró, RN, 2023. Disponível em: https://ppgcc.ufersa.edu.br/wp-content/uploads/sites/42/2024/03/Dissertacao_Mestrado_Marisergio.pdf. Acesso em: 14 nov. 2025.

PASSLIB. Passlib 1.7.4 Documentation. [S. l.]: Assurance Technologies, LLC, 2020. Disponível em: https://passlib.readthedocs.io/en/stable/. Acesso em: 14 nov. 2025.

PELENKAHU, D. Scaling Up Digital Transformation in Education: An Exploration of Teachers' Capabilities in Implementing Distance Learning in Southeast Sulawesi, Indonesia. 2022. Dissertação (Mestrado) – Lund University, Lund, 2022. Disponível em: https://lup.lub.lu.se/student-papers/record/9098945. Acesso em: 03 nov. 2025.

POWELL, Phill; SMALLEY, Ian. What is functional testing?. IBM Think, [S. l.], [2025?]. Disponível em: https://www.ibm.com/think/topics/functional-testing. Acesso em: 16 nov.

2025.

RAFAEL. H. OPortas POP3, SMTP e IMAP – protocolos de email explicados. [S. l.]: Hostinger, 09 abr. 2025. Disponível em: https://www.hostinger.com/br/tutoriais/portas-pop3-smtp-e-imap#:~:text=Passo%203%20%E2%80%93%20SMTP%20para%20comunica%C3%A7%C3%A3o,de%20e%2Dmail%20do%20destinat%C3%A1rio.. Acesso em: 14 nov. 2025.

REPORTLAB. ReportLab Documentation. [S. l.]: ReportLab Inc., [2025?]. Disponível em: https://docs.reportlab.com/reportlab/userguide/ch1_intro/. Acesso em: 14 nov. 2025.

RUSSELL, Stuart J.; NORVIG, Peter. Artificial intelligence: a modern approach. 4. ed. Hoboken: Pearson, 2021.

SEGUINS, Neilton. O que é JSON Web Tokens? Alura, [S. l.], 15 maio 2022. https://www.alura.com.br/artigos/o-que-e-json-web-tokens?srsltid=AfmBOoqryqVVe9_YV13QSXyxHEO1MUqWJe5-hP9nAsO5ZRnPlVINMRl-. Acesso em: 14 nov. 2025.

SELIVANOV, Yury. PEP 492 – Coroutines with async and await syntax. [S. l.]: Python.org, 2015. Disponível em: https://peps.python.org/pep-0492/. Acesso em: 12 nov. 2025.

SUN, Peter; KIM, Dae-Kyoo. Analyzing Impact of Dependency Injection on Software Maintainability. ArXiv, [S. l.], 2022. Disponível em: https://arxiv.org/abs/2205.06381. Acesso em: 13 nov. 2025.

VASWANI, Ashish et al. Attention Is All You Need. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NEURIPS), 30., 2017, Long Beach. Anais... Long Beach: Curran Associates, 2017. p. 5998-6008. Disponível em: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html. Acesso em: 04 nov. 2025.

APÊNDICE A – PRINCIPAIS PROMPTS DE ENGENHARIA DA IA

1. Prompt para Geração da Ementa (JSON) (Arquivo: services/course_generator.py | Função: generate_course_outline)

```
Você é um especialista em design instrucional. Crie uma ementa detalhada para um curso sobre '{topic}' para um público de nível '{nivel}'. O curso deve ter exatamente {num_chapters} capítulos.
Sua resposta DEVE ser um objeto JSON que segue estritamente o seguinte schema:
```json
{Curso.schema_json(indent=2)}
```
O campo 'id_curso' deve ser um slug em letras minúsculas do tema.
Retorne APENAS o objeto JSON, sem nenhum texto introdutório, explicação ou formatação de markdown.
"""
```

2. Prompt para Geração do Conteúdo da Aula (Markdown) (Arquivo: services/course_generator.py | Função: generate_chapter_lessons.

```
Você é um especialista em design instrucional.
CONTEXTO:
- Curso: "{curso.tema}"
- Aula: Aula {i} - "{capitulo.titulo}"

TAREFA:
 Escreva o conteúdo detalhado APENAS para o seguinte tópico: "{topico}".
O conteúdo deve ter introdução, desenvolvimento e conclusão.
Todo o conteúdo deve ser em Português do Brasil.

REGRAS DE FORMATAÇÃO:
- NÃO inclua um título para o tópico no seu texto. Gere APENAS o conteúdo.
- Para subtópicos, use o formato: ## Subtópico
- Para texto em negrito, use: **texto em negrito**
- Para listas, use: * Item da lista
- Para código, use: ```linguagem ... ```
"""
```

3. Prompt para Geração do Roteiro de Slides (JSON) (Arquivo: services/pptx_generator.py | Função: _generate_slide_instructions)

Você é um especialista em design instrucional. Sua tarefa é transformar o conteúdo de uma aula em texto em uma estrutura JSON para uma apresentação de slides completa e detalhada.

CONTEÚDO COMPLETO DA AULA:
{chapter_text}

TAREFA:
Analise o **texto completo fornecido** e crie uma apresentação abrangente. Não é um resumo, mas sim uma adaptação do conteúdo para o formato de slides.

REGRAS:
1. **Conteúdo em Português:** Todo o texto gerado (títulos, subtítulos, tópicos) deve estar em Português do Brasil.
2. **Estrutura:** Crie um título e subtítulo geral. Gere um sumário com os pontos principais.
3. **Abrangência:** Identifique as seções e conceitos mais importantes do texto e crie um slide dedicado para cada um. O objetivo é cobrir todos os pontos principais da aula.
4. **Tópicos Curtos:** Cada slide deve ter no máximo 5 tópicos (bullet points) curtos, diretos e objetivos.
5. **Imagens:** Se um slide abordar um conceito visual ou técnico (ex: trecho de código, diagrama, evento histórico), defina "imagem_necessaria": true e forneça uma descrição **em inglês**, curta e específica para a busca da imagem (ex: "C language code example", "Bell Labs 1970"). Caso contrário, defina como false.

FORMATO DE SAÍDA (responda APENAS com o JSON):

```
{{
      "titulo": "Título Principal da Apresentação em Português",
      "subtitulo": "Subtítulo da Apresentação em Português",
      "sumario": ["Tópico Principal 1", "Tópico 2", "Tópico 3"],
      "slides": [
       {{
               "titulo": "Título do Slide 1 (ex: O Que é a Linguagem C?)",
                  "topicos": ["Ponto chave 1", "Ponto chave 2", "Ponto chave
3"],
               "imagem_necessaria": true,
               "imagem_descricao": "vintage computer terminal"
           }},
           {{
```

```
              "titulo": "Título do Slide 2 (ex: A História da Linguagem C)",
                "topicos": ["Origem na linguagem B", "Desenvolvida por Dennis
Ritchie", "Padrão K&R em 1978"],
              "imagem_necessaria": false,
              "imagem_descricao": null
            }}
        ]
}}
"""
```

4. Prompt para Geração do Quiz (JSON) (Arquivo: services/pdf_generator.py | Função: _generate_quiz_data)

```
Você é um especialista em educação e avaliação pedagógica.

CONTEÚDO DO CAPÍTULO:
{chapter_text[:8000]}

TAREFA:
Crie um quiz de 10 questões de múltipla escolha baseado no conteúdo acima.

REQUISITOS:
1. 10 questões progressivas (do básico ao avançado)
2. Cada questão com 4 alternativas (A, B, C, D)
3. Apenas UMA alternativa correta
4. Questões que testem compreensão real, não memorização
5. Misture tipos: conceituais, aplicação prática, análise
6. Evite questões muito óbvias ou pegadinhas injustas

FORMATO JSON:
{{
"titulo": "Quiz - [Nome do Tópico]",
"subtitulo": "Teste seus conhecimentos sobre [tema]",
"instrucoes": "Escolha a alternativa correta para cada questão",
"questoes": [
       {{
              "numero": 1,
              "enunciado": "Texto da questão aqui?",
              "alternativas": {{
                "A": "Primeira alternativa",
                "B": "Segunda alternativa",
                "C": "Terceira alternativa",
```

```
                "D": "Quarta alternativa"
              }},
              "resposta_correta": "B",
              "explicacao": "Breve explicação da resposta correta"
            }}
          ]
        }}

IMPORTANTE:
- Questões claras e objetivas
- Alternativas plausíveis (sem respostas absurdas)
- Explicações educativas
- Respeite a complexidade do conteúdo original
- Alterne as respostas corretas entre A, B, C, D, evitar que sempre a resposta seja a mesma alternativa ao decorrer das questões

RESPONDA APENAS COM JSON VÁLIDO:
"""
```