UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Paula Fernanda Rosa do Nascimento

Desenvolvimento do Backend com Ênfase na Modelagem de Dados para o Sistema de Acompanhamento de Egressos na UFU

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Paula Fernanda Rosa do Nascimento

Desenvolvimento do Backend com Ênfase na Modelagem de Dados para o Sistema de Acompanhamento de Egressos na UFU

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientadora: Maria Adriana Vidigal de Lima

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Paula Fernanda Rosa do Nascimento

Desenvolvimento do Backend com Ênfase na Modelagem de Dados para o Sistema de Acompanhamento de Egressos na UFU

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 28 de maio de 2025:

Maria Adriana Vidigal de Lima
Orientadora

Rafael Dias Araújo

Renato de Aquino Lopes

Uberlândia, Brasil
2025

Resumo

O acompanhamento da trajetória profissional dos egressos é um processo essencial para a avaliação da qualidade dos cursos e para a adequação das instituições de ensino às demandas da sociedade. Contudo, a ausência de ferramentas tecnológicas adequadas torna esse monitoramento um desafio para muitas universidades brasileiras. Diante desse cenário, o presente trabalho tem como objetivo dar continuidade ao desenvolvimento de um sistema de acompanhamento de egressos para a Universidade Federal de Uberlândia. Ele tem como foco a reformulação da modelagem de dados, criada em trabalhos anteriores, utilizando PostgreSQL e a construção de API RESTful utilizando Java 17 e o framework Spring Boot. Como resultado, foi disponibilizada a primeira versão do *backend*, capaz de ser integrado com o *frontend* existente e possibilitando a manipulação de informações acadêmicas, profissionais, depoimentos e comunicados dos ex-alunos. O sistema desenvolvido representa um avanço na organização e gestão de dados dos egressos na UFU, fortalecendo o vínculo institucional e contribuindo as avaliações do MEC.

Palavras-chave: Modelagem de dados, egresso, *backend*, Java, Spring Boot, API.

Lista de ilustrações

Lista de abreviaturas e siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	*Application Programming Interface*
funcional	*Data Definition Language*
DML	*Data Manipulation Language*
DTO	*Data Transfer Object*
FTP	*File Transfer Protocol*
HTTP	*Hypertext Transfer Protocol*
ID	Identificador
MER	Modelo Entidade Relacionamento
REST	*Representational State Transfer*
SES	*Simple Email Service*
SG	Sistema de Gestão
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
UFU	Universidade Federal de Uberlândia
UUID	*Universally Unique Identifier*

Sumário

1 Introdução

As constantes mudanças no mercado de trabalho impactam diretamente as Instituições de Ensino Superior (IES). As IES devem preparar seus alunos não apenas com conhecimentos teóricos e práticos, mas também observar a trajetória de seus egressos após a graduação. Esse acompanhamento é necessário para avaliar a eficácia dos programas educacionais e compreender como os formandos se inserem ao mercado. Com base nessas informações, as instituições podem ajustar seus currículos de acordo com as necessidades da sociedade. A utilização de sistemas computacionais que facilitem a comunicação entre as IES e seus ex-alunos tem se tornado uma prática cada vez mais comum. Isso ocorre tanto no Brasil quanto em outros países. Essa prática é eficaz para garantir a qualidade de ensino e alinhar as universidades com as demandas no mercado de trabalho (GUIMARãES, 2013; RANTHUM; JUNIOR, 2023).

A relevância do acompanhamento de egressos para as IES se dá pelo fato de que, além de fortalecer o vínculo institucional entre elas e seus antigos alunos, esse monitoramento contribui para as avaliações realizadas pelo Ministério da Educação (MEC). O Sistema Nacional de Avaliação da Educação Superior (SINAES) possui, como critério de avaliação, a política de acompanhamento de egressos. Ademais, manter um relacionamento constante com os graduados traz retornos importantes para as universidades, como a contratação de seus alunos em cargos de estagiários e *trainees*, parcerias com empresas onde os ex-alunos trabalham e até mesmo doações de recursos. Além disso, o sucesso dos egressos no mercado de trabalho atrai novos discentes e coopera para o marketing da qualidade dos cursos oferecidos pela instituição (MICHELAN et al., 2009).

Conforme apontado por Paul (2015), em países como a França, Grã-Bretanha, Alemanha e Itália, o monitoramento dos egressos tem permitido ajustes curriculares e melhorias na empregabilidade dos graduados. Na França, o Centro de Estudos e de Pesquisas sobre as Qualificações (Céreq) foi pioneiro na implementação de pesquisas nacionais de inserção profissional. Essas pesquisas permitiram às universidades supervisionar a transição dos formandos para o mercado de trabalho e ajustar suas ofertas e cursos conforme as demandas da área de atuação. Na Itália, o consórcio *AlmaLaurea* se destacou ao criar uma base de dados abrangente de currículos de egressos, facilitando a conexão entre formandos e empregadores. Na Alemanha, o projeto *Kooperationsprojekt Absolventenstudien* (KAOB) coordenou pesquisas de ex-alunos em várias universidades, promovendo a profissionalização dos levantamentos e a criação de uma base de dados anônimos. Já na Grã-Bretanha, a *Higher Education Statistics Agency* (HESA) padronizou a coleta de dados sobre a inserção dos egressos, permitindo a criação de estudos longitudinais que auxiliam as IES a ajustarem suas políticas educacionais de acordo com as necessidades

do mercado de trabalho.

Apesar dos exemplos positivos observados internacionalmente, há instituições brasileiras que ainda enfrentam dificuldades para estruturar um acompanhamento sistemático de seus egressos. A falta de ferramentas adequadas e integradas torna o processo de monitoramento pouco eficiente e com alcance limitado. Isso evidencia a necessidade de soluções tecnológicas que apoiem essa atividade de forma mais eficaz.

Diante disso, parte-se da hipótese de que a construção de um sistema informatizado específico para o acompanhamento de egressos pode facilitar esse processo e gerar impactos positivos tanto na avaliação institucional quanto na adequação dos cursos às exigências do mercado.

Deste modo, o objetivo principal deste trabalho é dar continuidade ao desenvolvimento do sistema de acompanhamento de egressos da Universidade Federal de Uberlândia, iniciado por Mendes (2023) e Carvalho (2025). Para isso, será realizada a criação do modelo de dados utilizando PostgreSQL, bem como a implementação do *backend* com Java e Spring Framework, além da documentação das **APIs** (*Application Programming Interfaces*) com Swagger, seguindo os requisitos definidos em trabalhos anteriores.

Ao final, será disponibilizada a primeira versão funcional do *backend* do sistema, que será integrada ao *frontend* desenvolvido por Carvalho (2025), substituindo os dados fixos que estão sendo utilizados, permitindo a comunicação real com o banco de dados e tornando viável o uso efetivo do sistema.

2 Referencial Teórico

2.1 Sistemas de Gerenciamento de Egressos

Os sistemas de gerenciamento de egressos são ferramentas utilizadas pelas Instituições de Ensino Superior (IES) para acompanhar a trajetória de seus ex-alunos. Eles permitem coletar e analisar informações sobre a formação recebida e a inserção no mercado de trabalho. Segundo Silva e Bezerra (2015), esses sistemas ajudam a fortalecer a integração entre a instituição e a sociedade, promovendo uma comunicação contínua com os egressos.

O interesse por esse tipo de acompanhamento começou a crescer nos anos 1960, especialmente em países como os Estados Unidos e a França. Nesses locais, foram realizadas pesquisas para entender a transição dos formandos para o mercado de trabalho (PAUL, 1989). No Brasil, as primeiras iniciativas surgiram na década de 1980. Universidades como a Universidade Federal do Ceará (UFC) e a Universidade de São Paulo (USP) realizaram pesquisas pontuais e criaram portais para egressos (PAUL; FREIRE, 1997). Com a criação do Sistema Nacional de Avaliação da Educação Superior (SINAES) em 2004, o acompanhamento de egressos passou a ser um critério de avaliação da IES (Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (Inep), 2004).

Nos últimos anos, os sistemas de gerenciamento de egressos têm incorporado novas tecnologias. Aplicativos móveis e portais interativos são exemplos de ferramentas que facilitam a coleta e o uso de dados. Azis et al. (2018) destacam o desenvolvimento de aplicativos que conectam egressos a oportunidades de emprego e treinamentos.

No Brasil, algumas universidades têm se destacado com iniciativas nesse campo. O Portal de Egressos da Universidade Federal de Santa Catarina (UFSC) oferece serviços como atualização de dados e acesso à biblioteca universitária Silva, Nunes e Jacobsen (2011). Já o sistema do curso de Secretariado Executivo da Universidade Estadual do Oeste do Paraná (UNIOESTE) promove a interação entre ex-alunos e a instituição, além de fornecer dados para melhorias no curso (SCHMIDT; MOURA, 2015).

Esses sistemas têm contribuído para que as IES avaliem a qualidade de seus serviços e adaptem seus currículos às demandas do mercado. Além disso, ajudam a manter um vínculo ativo com os ex-alunos, criando oportunidades de colaboração e troca de informações (MICHELAN et al., 2009; SCHMIDT; MOURA, 2015).

2.2 Tecnologias Utilizadas no Desenvolvimento Backend

2.2.1 Linguagem Java

Java é uma linguagem de programação de alto nível, orientada a objetos, independente de plataforma e amplamente utilizada no desenvolvimento de aplicações. Criada em 1991 por um grupo de engenheiros da Sun Microsystems, liderado por James Gosling, a linguagem surgiu como parte do Projeto *Green*, cujo objetivo era desenvolver tecnologias modernas para dispositivos eletrônicos de consumo, como televisores e controles remotos. Inicialmente chamada de Oak, em referência a um carvalho visível do escritório de *Gosling*, a linguagem foi renomeada para Java em 1995, inspirada no café Java, amplamente consumido pela equipe durante o desenvolvimento (GLUSHACH, 2023).

A principal motivação para a criação do Java foi a necessidade de uma linguagem que fosse simples, robusta, segura e, acima de tudo, portátil, permitindo que o mesmo código fosse executado em diferentes plataformas. Essa característica, conhecida como *Write Once, Run Anywhere* (WORA), foi viabilizada pela Máquina Virtual Java (JVM), que interpreta os bytecodes gerados pela compilação do código-fonte, tornando-o independente do sistema operacional subjacente (THIAGO, 2012) .

Além de sua aplicação inicial em dispositivos eletrônicos, o Java expandiu seu alcance para diversas áreas, incluindo desenvolvimento web, sistemas empresariais, aplicativos móveis, computação em nuvem e Internet das Coisas (IoT). A linguagem também desempenha um papel fundamental no desenvolvimento de aplicativos Android, consolidando-se como uma das tecnologias mais versáteis e amplamente utilizadas no mundo. Mesmo após a aquisição da Sun Microsystems pela Oracle em 2009, o Java manteve sua relevância, com versões gratuitas como o OpenJDK e suporte contínuo para a comunidade de desenvolvedores (BEZERRA, 2022).

2.2.2 Utilização do Spring Framework

O Spring é um *framework open-source* para a plataforma Java, criado por Rod Johnson em 2003, com o objetivo de simplificar o desenvolvimento de aplicações empresariais. Desde sua criação, o Spring tem evoluído continuamente. A primeira versão estável foi lançada em 2004, trazendo funcionalidades básicas. Em 2013, o lançamento do Spring Boot revolucionou o *framework* ao simplificar a configuração e permitir a criação de aplicações prontas para produção de forma rápida. Mais recentemente, o Spring Framework 5.0 introduziu suporte à programação reativa com o Spring WebFlux, enquanto a versão 6.0, lançada em 2023, atualizou o framework para o nível Jakarta EE 9+, garantindo compatibilidade com as APIs modernas (BROADCOM, 2024; VICTOR, 2023).

O Spring é utilizado para o desenvolvimento de aplicações web, sistemas distri-

buídos e microservices. Seus módulos, como Spring MVC, Spring Data e Spring Security, oferecem suporte para criação de interfaces web, integração com bancos de dados e implementação de segurança, respectivamente. Essa modularidade, combinada com a flexibilidade e a robustez do *framework*, faz do Spring uma das ferramentas mais importantes no ecossistema Java (BROADCOM, 2024; FRANçA, 2024) .

2.2.3 Conceitos e Componentes de um Sistema Cliente-Servidor

A arquitetura cliente-servidor é um modelo de *software* que organiza as funções de um sistema em dois componentes principais: o cliente e o servidor. O primeiro é responsável por enviar solicitações, enquanto o servidor as processa e retorna as respostas. Essa separação de responsabilidades permite que o processamento seja distribuído. Isso otimiza o desempenho e a escalabilidade do sistema. Segundo Zhang (2013), o modelo cliente-servidor utiliza protocolos padronizados, como HTTP, FTP e SMTP, para facilitar a comunicação entre os componentes. Esses protocolos garantem que as mensagens trocadas entre cliente e servidor sigam um formato compreensível por ambas as partes.

Além disso, a arquitetura cliente-servidor pode ser implementada em diferentes níveis, como a arquitetura de dois níveis (2-tier) e três níveis (3-tier). Na arquitetura de dois níveis, o cliente se comunica diretamente com o servidor de banco de dados. Já na arquitetura de três níveis, um servidor intermediário, conhecido como middleware, é responsável por gerenciar a lógica de negócios. Essa última abordagem é mais adequada para sistemas complexos, pois reduz a carga no cliente e melhora a segurança e a manutenção do sistema. O *middleware*, como descrito por Kambalyal (2010), é uma camada separada que executa a lógica de aplicação e facilita a comunicação entre o cliente e o servidor de banco de dados.

2.2.4 Requisições HTTP e APIs RESTful

As requisições HTTP e *APIs RESTful* desempenham um papel importante na comunicação entre cliente e servidor em sistemas modernos. O HTTP é um protocolo de comunicação que permite a troca de informações entre clientes e servidores, sendo amplamente utilizado em aplicações web. Ele segue um modelo de requisição-resposta, onde o cliente envia uma solicitação ao servidor, que processa a solicitação e retorna uma resposta. Segundo Zhang (2013), o HTTP é utilizado para transferir arquivos multimídia, como imagens e textos, entre cliente e servidor, melhorando a eficiência da comunicação.

As *APIs RESTful*, por sua vez, seguem os princípios *REST* (*Representational State Transfer*) para criar interfaces que permitem a interação entre sistemas de forma padronizada e escalável. Elas utilizam métodos HTTP, como *GET, POST, PUT* e *DELETE*, para realizar operações em recursos no servidor. Essa abordagem facilita a integração

entre diferentes sistemas e promove a reutilização de componentes. Além disso, as *APIs RESTful* são projetadas para serem leves e independentes de plataforma, o que as torna ideais para aplicações distribuídas e baseadas na web. O trabalho de Kratky e Reichenberger (2013) reforça que o uso de *APIs RESTful* melhora a interoperabilidade e a eficiência na comunicação entre cliente e servidor.

2.2.5 Swagger

O Swagger é um conjunto de ferramentas que facilita o processo de criação de *APIs RESTful*, abrangendo todas as etapas do ciclo de vida dessas interfaces, como design, documentação, testes e implantação. Seu objetivo principal é padronizar e simplificar o desenvolvimento de APIs, promovendo a reutilização de código e a integração entre sistemas (SmartBear Software, 2025a).

Entre as ferramentas disponíveis no Swagger, destacam-se o Swagger Editor, que permite a criação de definições de APIs em um ambiente acessível via navegador; o Swagger UI, que transforma essas definições em documentações interativas; e o Swagger Codegen, que gera códigos para servidores e clientes em diversas linguagens de programação. Elas automatizam tarefas como a geração de bibliotecas de clientes e a criação de testes, otimizando o trabalho das equipes de desenvolvimento (SmartBear Software, 2025a).

O uso do Swagger oferece benefícios significativos, como a aceleração do desenvolvimento de APIs e a garantia de que elas sejam bem documentadas e de fácil utilização. Além disso, a padronização promovida pela *OpenAPI Specification* favorece a colaboração entre diferentes ferramentas e equipes, resultando em maior qualidade e compatibilidade das APIs (SmartBear Software, 2025b).

2.2.6 Fundamentos de Banco de Dados Relacional

Os bancos de dados relacionais surgiram como uma solução para os desafios enfrentados pelos sistemas de gerenciamento de dados nas décadas de 1960 e 1970. Antes do modelo relacional, os dados eram armazenados em estruturas específicas e arbitrárias, o que dificultava o acesso e a manipulação eficiente das informações. O modelo relacional foi introduzido para resolver esses problemas, oferecendo uma abordagem padronizada e intuitiva para o armazenamento e a consulta de dados. Ele se baseia no uso de tabelas, onde os dados são organizados em linhas e colunas, permitindo que diferentes conjuntos de informações sejam relacionados por meio de chaves primárias e estrangeiras (CODD, 1982).

Uma das principais inovações do modelo relacional foi a separação entre a estrutura lógica e o armazenamento físico dos dados. Essa distinção, conhecida como independência de dados, permite que alterações no armazenamento físico não afetem a forma como os

dados são acessados logicamente. Além disso, o modelo relacional introduziu a linguagem SQL, que se tornou o padrão para a manipulação de dados em bancos de dados relacionais. A SQL, baseada na álgebra relacional, oferece uma maneira consistente e eficiente de realizar consultas e operações sobre os dados (CODD, 1982).

Os bancos de dados relacionais também são conhecidos por suas propriedades ACID, que garantem a integridade e a confiabilidade das transações. Essas propriedades são fundamentais para aplicações críticas, como sistemas financeiros e de comércio eletrônico, onde a precisão e a consistência dos dados são indispensáveis. Além disso, o modelo relacional é bastante utilizado devido à sua capacidade de manter a consistência de dados entre diferentes instâncias e usuários, mesmo em cenários de alta simultaneidade (CODD, 1982).

2.2.7 PostgreSQL

O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional que evoluiu a partir do projeto POSTGRES iniciado em 1986 na Universidade da Califórnia, Berkeley. Sob a liderança do professor Michael Stonebraker, o projeto foi concebido como uma evolução do sistema INGRES, com o objetivo de explorar novos conceitos de gerenciamento de dados, como suporte a tipos de dados complexos e extensibilidade. (IBM, 2024; The PostgreSQL Global Development Group, 2024).

Desde sua criação, o PostgreSQL tem se destacado por sua robustez, flexibilidade e adesão a padrões abertos. Ele é reconhecido como um dos sistemas de banco de dados mais avançados e maduros disponíveis, sendo utilizado em uma ampla gama de aplicações. Sua arquitetura extensível permite a adição de novos tipos de dados, funções e operadores, tornando-o uma escolha versátil para desenvolvedores e empresas (IBM, 2024).

O PostgreSQL também é conhecido por seu suporte a recursos avançados, como controle de concorrência multiversão (MVCC), que permite leituras e escritas simultâneas sem bloqueios, e recuperação de ponto no tempo (PITR), que facilita a restauração de dados a um estado anterior. Além disso, ele oferece suporte a várias linguagens de programação, como Python, JavaScript e C/C++, e pode ser configurado para alta disponibilidade por meio de replicação síncrona ou assíncrona(IBM, 2024).

2.3 Trabalhos Relacionados

O levantamento e a análise de requisitos para o desenvolvimento de um sistema de acompanhamento de egressos da UFU, realizado por Mendes (2023), teve como objetivo principal especificar funcionalidades essenciais e modelar processos de negócio. Para isso, foram realizadas reuniões com representantes da universidade e avaliações de sistemas similares em outras instituições. O resultado foi um documento que servirá como guia

para o desenvolvimento do sistema. A implementação que será realizada neste trabalho será amplamente baseada nos levantamentos de requisitos, casos de uso e modelagens documentadas por Mendes. Isso garante que as funcionalidades desenvolvidas estejam alinhadas às necessidades institucionais e expectativas dos usuários finais. Em paralelo, a primeira versão do *frontend*, implementada por Carvalho (2025), foi importante para o entendimento de quais *endpoints*, que são endereços de acesso às funcionalidades da API, deveriam ser criados durante a execução deste trabalho.

De forma complementar, Silva (2023) desenvolveu uma ferramenta de acompanhamento sistemático de egressos em programas de pós-graduação em saúde. O propósito era melhorar a gestão educacional e fortalecer o vínculo com os profissionais formados. O trabalho envolveu a aplicação de questionários e o desenvolvimento de um software em parceria com o Instituto Federal do Rio Grande do Sul (IFRS). A ferramenta destacou o valor de sistemas automatizados para coleta e análise de dados. A integração de funcionalidades voltadas para a coleta e análise de dados se mostra uma solução prática e eficiente para o acompanhamento de egressos.

Além disso, Ranthum e Junior (2023) propuseram um sistema computacional para o acompanhamento de egressos da pós-graduação *stricto sensu*. A aplicação inclui funcionalidades como consulta de egressos, estatísticas e integração com bases externas, como o Currículo Lattes. O desenvolvimento foi baseado em uma revisão documental e na definição de requisitos funcionais e não funcionais. Os autores demonstraram a eficácia da ferramenta na gestão de dados e na geração de indicadores. A proposta oferece contribuições valiosas sobre a integração de sistemas externos e a geração de relatórios gerenciais. Esses aspectos serão incorporados no sistema de acompanhamento de egressos da UFU.

Por fim, Silva, Mineiro e Favaretto (2022) realizaram uma revisão integrativa sobre o uso de sistemas de informação no acompanhamento de egressos. O estudo destacou a importância desses sistemas para a avaliação institucional e a melhoria contínua dos cursos. A pesquisa identificou que sistemas bem estruturados podem fortalecer o vínculo entre egressos e instituições. Ademais, fornecem dados significativos para ajustes curriculares. As boas práticas apresentadas nesse artigo reforçam a necessidade de sistemas robustos e integrados, alinhados às demandas institucionais e educacionais, o que também será um dos pilares da implementação proposta.

3 Desenvolvimento

Este capítulo apresenta o processo de desenvolvimento do sistema para acompanhamento de egressos de cursos universitários, com foco na Universidade Federal de Uberlândia (UFU). Inicialmente, são descritas as etapas de modelagem do sistema, contemplando a definição da estrutura de dados e a organização das funcionalidades previstas. Em seguida, aborda-se a inserção de dados e a implementação das APIs responsáveis pela comunicação entre os componentes do sistema.

3.1 Modelagem do Banco de Dados

A primeira etapa do desenvolvimento consistiu na análise das tabelas necessárias para o armazenamento dos dados do Egresso. No levantamento feito por Mendes (2023), o diagrama ficou conforme demonstrado na Figura 1:

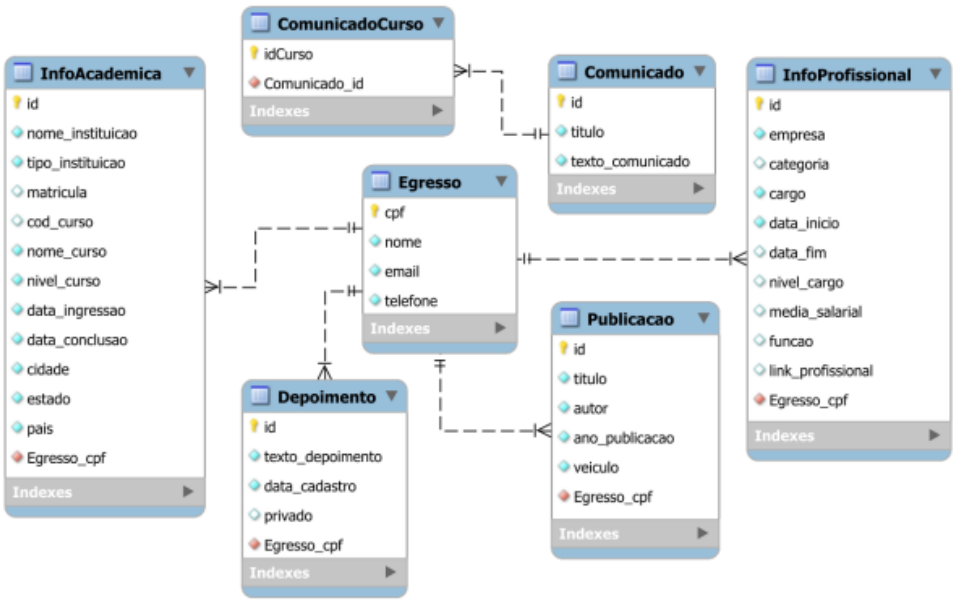

Figura 1 – MER do banco de dados desenvolvido por Mendes (2023)

Entretanto, na modelagem validada pelo CTIC (Centro de Tecnologia da Informação e Comunicação) da Universidade Federal de Uberlândia, há algumas diferenças:

1. Na tabela de egresso foram incluídos dois campos: **link_lattes** e **link_orcid**. Ambos são opcionais e armazenam o endereço do perfil do egresso nas plataformas Lattes e Orcid, respectivamente.

2. No depoimento, o campo **privado**, do tipo booleano, que determina se o depoimento é público ou não, passa a ser obrigatório. Além disso, houve mudança no seu relacionamento: deixou-se de armazenar o cpf do egresso como chave estrangeira para gravar o id da informação acadêmica.

3. A tabela Comunicado passa a guardar o tipo e a data de envio de forma obrigatória. A data de criação, porém, torna-se opcional.

4. A tabela que define a qual curso o comunicado é destinado, anteriormente chamada de ComunicadoCurso, passa a ser uma tabela intermediaria originada do relacionamento muitos-para-muitos entre o Comunicado e a Informação Acadêmica.

5. A tabela InformaçãoProfissional, que antes possuía relação de muitos-para-um com a tabela de Egresso, passa a ter relacionamento muitos-para-muitos com a tabela de Informação Acadêmica, gerando uma nova tabela intermediária.

6. O relacionamento entre Publicação e Egresso foi desfeito. Agora, a publicação faz referência à informação acadêmica.

O diagrama completo é mostrado na Figura 2 a seguir:

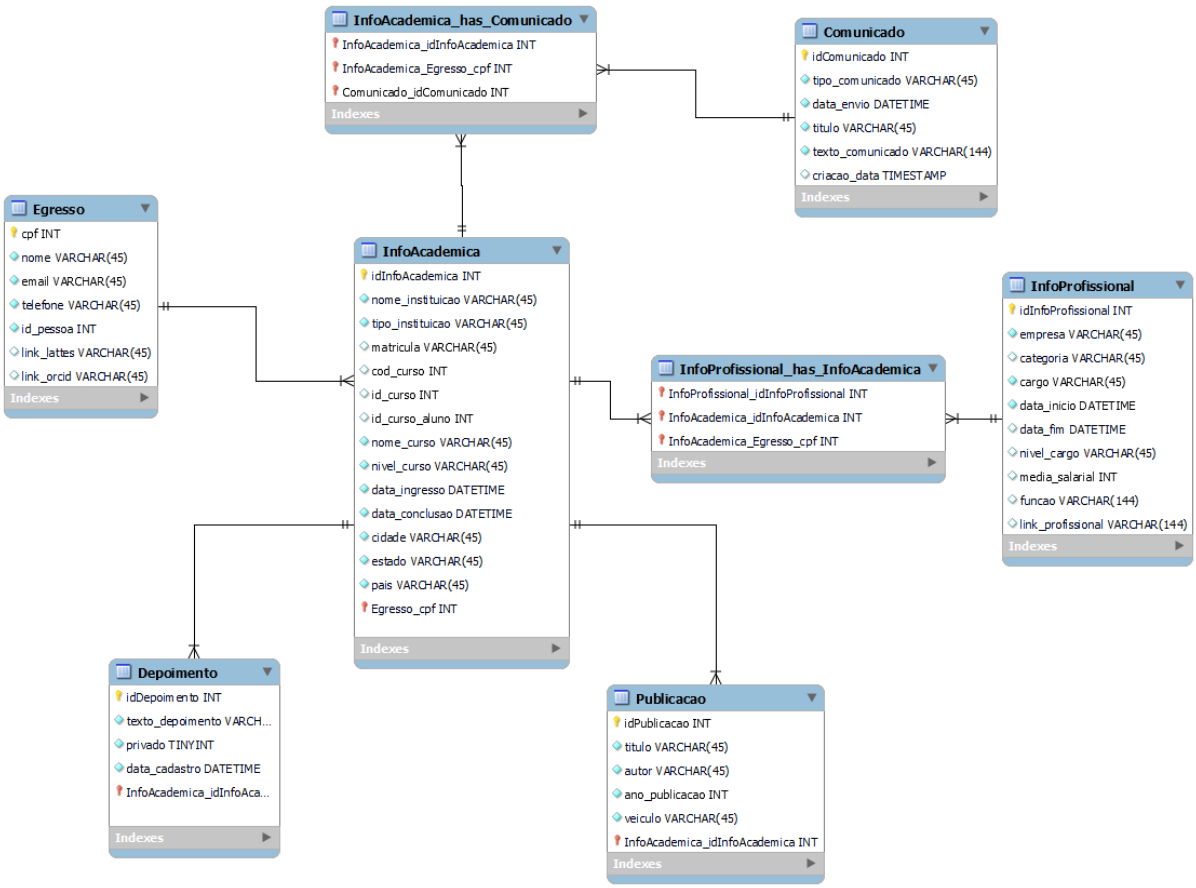

Figura 2 – MER do banco de dados validado pelo CTIC

Com essas mudanças, é possível perceber que o egresso deixa de ser a entidade principal, sendo a informação acadêmica quem assume esse papel. Essa abordagem foi considerada mais adequada pelos seguintes motivos:

- Para uma pessoa ser considerada egresso, ela deve possuir um vínculo acadêmico com a UFU.
- Tanto o depoimento quanto a publicação se referem a uma informação acadêmica.
- O comunicado pode ser destinado a egressos de determinados cursos, e o registro de qual curso cada egresso fez está armazenada na tabela de Informação Acadêmica.
- Uma informação acadêmica pode ter várias experiências profissionais associadas, mas cada experiência está ligada a apenas uma formação.

Desta forma, a modelagem feita pelo Coordenador da Divisão de Sistemas do CTIC foi considerada como base. Ela foi ajustada para ficar de acordo com o que é esperado no *frontend* implementado por Carvalho (2025). As alterações foram as seguintes:

- De forma geral, o tipo das chaves primárias foi alterado de int para UUID. Isso foi feito com o objetivo de melhorar a organização e a segurança dos dados. O uso de UUIDs permite gerar identificadores únicos automaticamente, sem depender de sequências numéricas. Isso evita conflitos de IDs, facilita a integração com outros sistemas e aumenta a segurança, dificultando que usuários adivinhem IDs de outros registros.
- Em relação a entidade egresso, a chave primária **cpf** foi alterada de int para varchar(14). Apesar de conter apenas dígitos, o CPF pode conter zeros à esquerda, o que seria perdido caso fosse armazenado como número inteiro. O varchar(14) também permite o armazenamento do CPF formatado, facilitando sua leitura. Além disso, foram adicionados os seguintes campos opcionais: **nome_social**, **email_secundario** e **telefone_secundário**. Eles visam respeitar a identidade de pessoas que utilizam um nome diferente do registrado oficialmente e oferecer meios alternativos de contato. Além disso, foi adicionado um campo para armazenar a url do perfil do egresso no LinkedIn.
- Na tabela depoimento, o campo **privado** foi renomeado para privacidade e seu tipo foi alterado de booleano para varchar(7). Essa mudança se deve ao fato de quando o usuário cria um depoimento no *frontend* ele deve selecionar entre as seguintes opções: Público, Privado, Anônimo.

- Na informação profissional houve a inclusão do campo **tipo**, que armazena o tipo de contrato do emprego, como por exemplo: tempo integral, meio período e *freelancer*. Também foi adicionado o campo localidade, que se refere à modalidade do trabalho, podendo ser presencial, remoto ou híbrido. O relacionamento com a tabela de informação acadêmica foi modificado para muitos-para-um. Essa alteração foi uma decisão pragmática para adaptar o *backend* à implementação do *frontend* existente, que, em seu formulário de cadastro, permitia ao egresso selecionar apenas uma formação acadêmica para associar a uma nova experiência profissional. Desse modo não é mais necessária a tabela intermediária. Contudo, reconhece-se que essa abordagem é uma simplificação. O modelo conceitual mais adequado seria o de muitos-para-muitos, como o validado pelo CTIC, pois uma mesma atuação profissional pode ser fruto de competências adquiridas em múltiplas formações, como graduação, mestrado e doutorado. A reavaliação deste relacionamento é, portanto, recomendada para futuras versões do sistema. Por fim, houve a remoção do campo **link_profissional**, pois ele não foi contemplado no *frontend*.

- Quanto à entidade comunicado, foram removidos os campos **tipo_comunicado** e **criacao_data**, porque eles não estavam sendo utilizados. Além disso, foi incluído o campo booleano **para_todos**, que determina se o comunicado deve ser enviado para todos os egressos ou não. Caso o comunicado seja específico para um curso e/ou nível (como graduação, mestrado ou doutorado) isso é informado nos campos **curso_destino** e **nivel_curso_destino**. Sendo assim, a tabela intermediária entre comunicado e informação acadêmica só será populada, caso o campo **curso_destino** não seja nulo.

- Na tabela de informação acadêmica, o campo **data_conclusao** passa a ser opcional, considerando que o egresso pode inserir manualmente uma formação acadêmica de outra instituição de ensino que não seja a UFU e não necessariamente ela foi concluída na data de cadastro. Ademais, foram adicionados os campos **campus**, **ano_evasao**, **semestre_evasao** e **ativo**. Esses campos são opcionais, pois serão utilizados apenas no cadastro automático com os dados retornados do sistema SG da UFU. Eles foram adicionados pois permite a filtragem e a visualização dos dados exibidos no *dashboar*d conforme implementado por Carvalho (2025).

O MER que contempla essas mudanças foi criado na ferramenta *MySQL Workbench* e é mostrado na Figura 3.

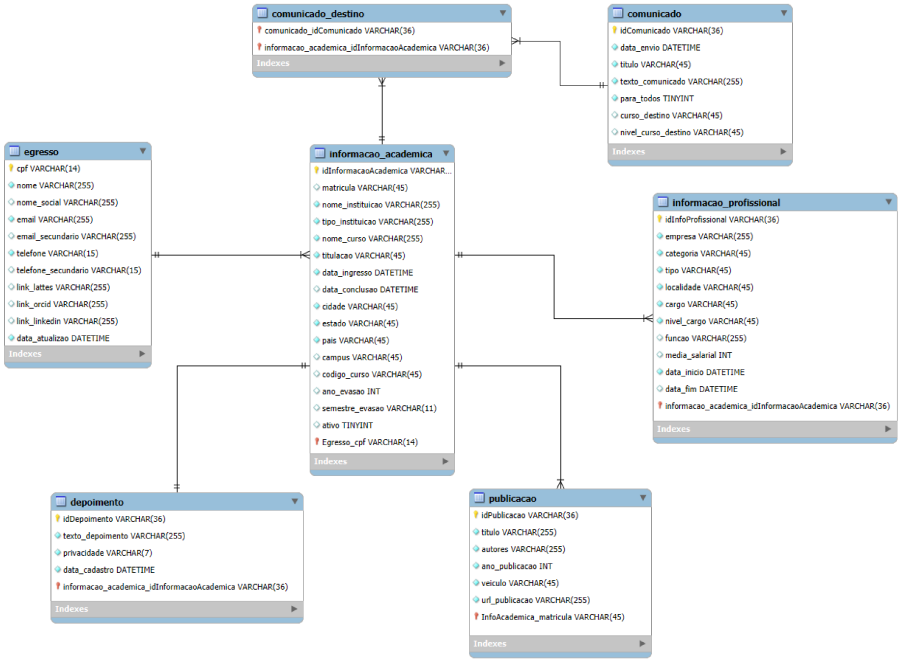

Figura 3 – MER do banco de dados utilizado no sistema de egressos

3.1.1 Criação e Inserção de Dados

Com a modelagem concluída, foi criado o DDL (Data Definition Language), conjunto de comandos para criar e alterar a estrutura das tabelas, para a geração das entidades utilizando PostgreSQL. Na Figura 4 é mostrado o DDL da tabela egresso como exemplo.

```
CREATE TABLE egresso (
    cpf VARCHAR(14) PRIMARY KEY NOT NULL,
    nome VARCHAR(255) NOT NULL,
    nome_social VARCHAR(255),
    email VARCHAR(255) NOT NULL,
    email_secundario VARCHAR(255),
    telefone VARCHAR(15) NOT NULL,
    telefone_secundario VARCHAR(15),
    link_lattes VARCHAR(255),
    link_orcid VARCHAR(255),
    link_linkedin VARCHAR(255),
    data_atualizacao DATE
);
```

Figura 4 – DDL da tabela de egressos

Em seguida, foi criado o DML (Data Manipulation Language), utilizado para inserir e gerenciar os registros nas tabelas, com a operação de inserção de egressos e informações acadêmicas, com o objetivo de simular o cadastro automático com os dados do SG (Sistema de Gestão) da UFU, onde são armazenados os dados acadêmicos oficiais da universidade, visto que este trabalho com contempla a integração com esse sistema. Os dados dos egressos foram gerados aleatoriamente e os dados acadêmicos foram adaptados de um relatório anonimizado do SG. Cada registro acadêmico foi vinculado a um egresso fictício. Nas Figuras 5 e 6 são mostrados exemplos de DML na tabela de egresso e da informação acadêmica:

```
INSERT INTO egresso (cpf, nome, nome_social, email, email_secundario, telefone, telefone_secundario, link_lattes, link_orcid, link_linkedin, data_atualizacao) VALUES(
 '123.456.789-01',
 'Ana Souza',
 NULL,
 'ana.souza@example.com',
 'ana.sec@example.com',
 '(34) 99999-0001',
 NULL,
 'http://lattes.cnpq.br/12345678901',
 'https://orcid.org/0000-0001-2345-6789',
 'https://www.linkedin.com/in/anasouza', NOW());
```

Figura 5 – DML da tabela de egressos

```
INSERT INTO informacao_academica (id, matricula, institution_name, institution_type, course_name, course_level, campus, registration_number, start_date, end_date, end_year, end_semester, city, state, country, egresso_cpf) VALUES
uuid_generate_v4(),
'31111BSI111',
'Universidade Federal de Uberlandia',
'Instituição Pública',
'Graduação em Sistemas de Informação: Bacharelado - Integral - Monte Carmelo',
'Bacharelado',
'Monte Carmelo',
'11377178I',
'2015-03-19',
'2024-04-25',
'2023',
'2º Semestre',
'Uberlândia',
'Minas Gerais',
'Brasil',
'123.456.789-01');
```

Figura 6 – DML da tabela de informação acadêmica

3.2 Implementação das APIs

O *backend* foi implementado em Java 17 com o *framework* Spring Boot na versão 3.3.4. A utilização do ecossistema Java e Spring atende a um requisito não funcional definido no levantamento de Mendes (2023), que visa à padronização com as tecnologias já adotadas pelo CTIC. Já a escolha por estas versões específicas deu-se pela familiaridade da equipe com as ferramentas, não havendo um requisito que impedisse o uso de outras alternativas.

A arquitetura da aplicação segue o padrão de camadas para organizar as responsabilidades do sistema, conforme ilustra o diagrama de blocos na Figura 7.

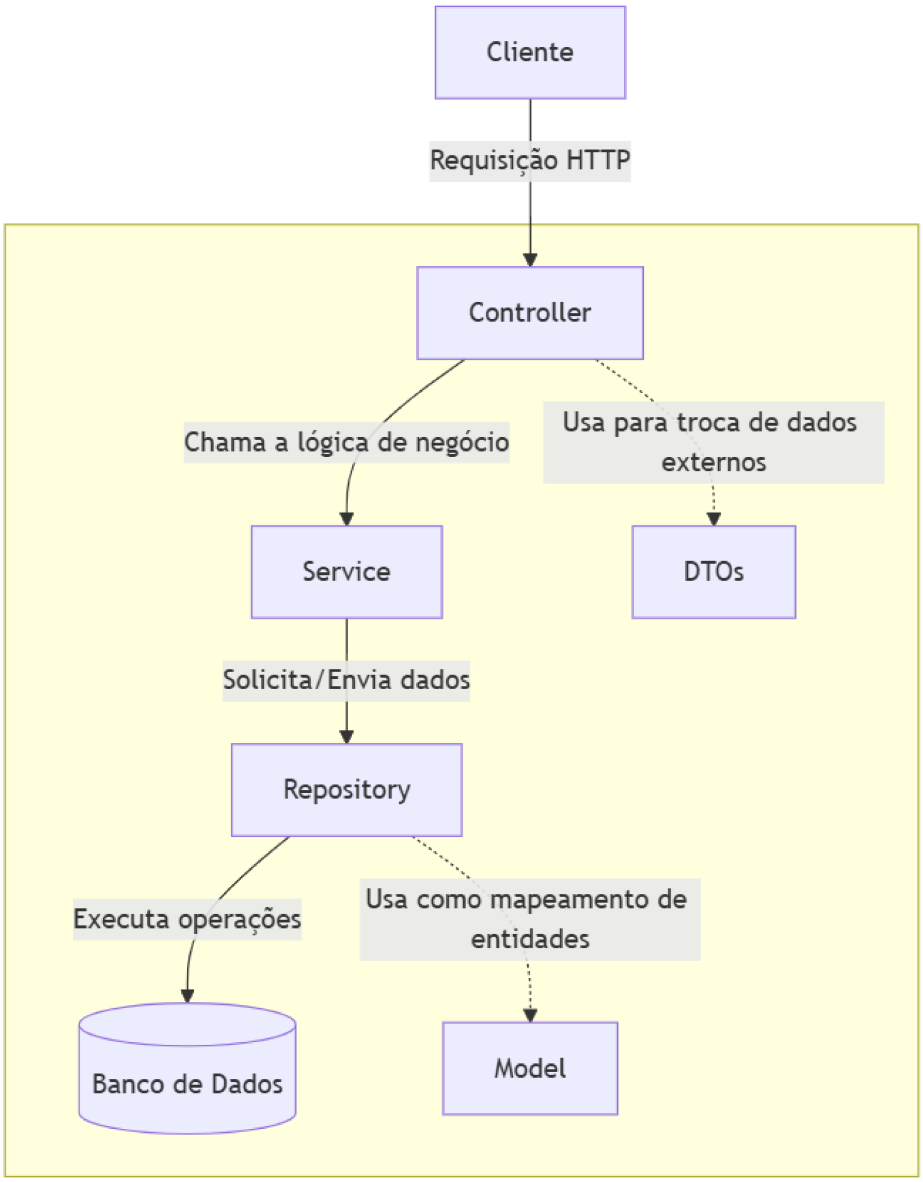

Figura 7 – Diagrama da arquitetura em camadas

- **Model**: representa as entidades do domínio, ou seja, os objetos principais do sistema que refletem tabelas do banco de dados. Exemplo do model da tabela publicação é apresentado na Figura 8.

```java
@Entity
@Table(name = "publicacao")
@JsonInclude(JsonInclude.Include.NON_NULL)
public class PublicacaoModel {

    2 usages
    @Id
    @GeneratedValue(generator = "UUID")
    @Column(nullable = false)
    private UUID id;

    2 usages
    @Column(nullable = false)
    private String titulo;

    2 usages
    @Column(nullable = false)
    private String autores;

    2 usages
    @Column(nullable = false)
    private Integer ano_publicacao;

    2 usages
    @Column(nullable = false, length = 45)
    private String veiculo;

    2 usages
    @Column(nullable = false, length = 255)
    private String url_publicacao;

    2 usages
    @ManyToOne
    @JoinColumn(name = "id_informacao_academica", referencedColumnName = "id", nullable = false)
    private InformacaoAcademicaModel informacao_academica;
```

Figura 8 – Model da entidade publicação

- **Repository**: é responsável por acessar o banco de dados. Ela contém os métodos para buscar, salvar, atualizar e excluir dados. Exemplo do repository do depoimento é demonstrado na Figura 9.

```java
5 usages  Paula Nascimento +1
public interface DepoimentoRepository extends JpaRepository<DepoimentoModel, UUID> {
    Paula Nascimento
    @Query("SELECT d FROM DepoimentoModel d WHERE d.informacao_academica.egresso.cpf = :cpf")
    List<DepoimentoModel> buscarPorEgresso(String cpf);
}
```

Figura 9 – Repository da entidade depoimento

- **Service**: contém as regras de negócio do sistema. Essa camada realiza as chamadas para os repositórios, faz validações e implementa a lógica principal da aplicação, mantendo o Controller mais limpo. Um exemplo service do egresso está presente na Figura 10.

```java
@Service
public class EgressoService {
    8 usages
    private final EgressoRepository egressoRepository;

    Paula Nascimento
    @Autowired
    public EgressoService(EgressoRepository egressoRepository) { this.egressoRepository = egressoRepository; }

    3 usages  Paula Nascimento
    public List<EgressoModel> buscarTodos() { return egressoRepository.findAll(); }

    3 usages  Paula Nascimento
    public Optional<EgressoModel> buscarPeloCPF(String cpf) { return egressoRepository.findById(cpf); }

    Paula Nascimento
    public EgressoModel criar(EgressoCriarDTO egressoDTO) {
        EgressoModel egresso = new EgressoModel();
        BeanUtils.copyProperties(egressoDTO, egresso);
        egresso.setData_atualizacao(LocalDate.now());
        return egressoRepository.save(egresso);
    }

    Paula Nascimento
    public EgressoModel atualizar(String cpf, EgressoAtualizarDTO egressoDTO) {
        Optional<EgressoModel> optionalEgresso = egressoRepository.findById(cpf);
        if (optionalEgresso.isPresent()) {
            EgressoModel egresso = optionalEgresso.get();
            atualizarCamposEgresso(egresso, egressoDTO);
            egresso.setData_atualizacao(LocalDate.now());
            return egressoRepository.save(egresso);
        }
        return null;
    }
```

Figura 10 – Service da entidade egresso

- **Controller**: recebe requisições HTTP, processa dados, chama serviços e retorna respostas. Na Figura 11 é mostrado o controller da publicação.

```java
@RestController
@RequestMapping("/api/publicacoes")
public class PublicacaoController {

    5 usages
    private final PublicacaoService publicacaoService;

    Paula Nascimento
    @Autowired
    public PublicacaoController(PublicacaoService publicacaoService) { this.publicacaoService = publicacaoService; }

    Paula Nascimento
    @GetMapping("/egresso/{cpf}")
    public List<PublicacaoModel> buscarPublicacoesPorEgresso(@PathVariable String cpf) {
        return publicacaoService.buscarPorEgresso(cpf);
    }

    Paula Nascimento
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public PublicacaoModel criarPublicacao(@RequestBody PublicacaoDTO publicacaoDTO) {
        return publicacaoService.criar(publicacaoDTO);
    }

    Paula Nascimento
    @PutMapping("/{id}")
    public PublicacaoModel atualizarPublicacao(@PathVariable UUID id, @RequestBody PublicacaoDTO publicacaoDTO) {
        return publicacaoService.atualizar(id, publicacaoDTO);
    }

    Paula Nascimento
    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void excluirPublicacao(@PathVariable UUID id) { publicacaoService.excluir(id); }
}
```

Figura 11 – Controller da entidade publicação

- **DTO**: transporta dados entre camadas sem expor o model diretamente. O DTO de atualização doo egresso é apresentado na Figura 12.

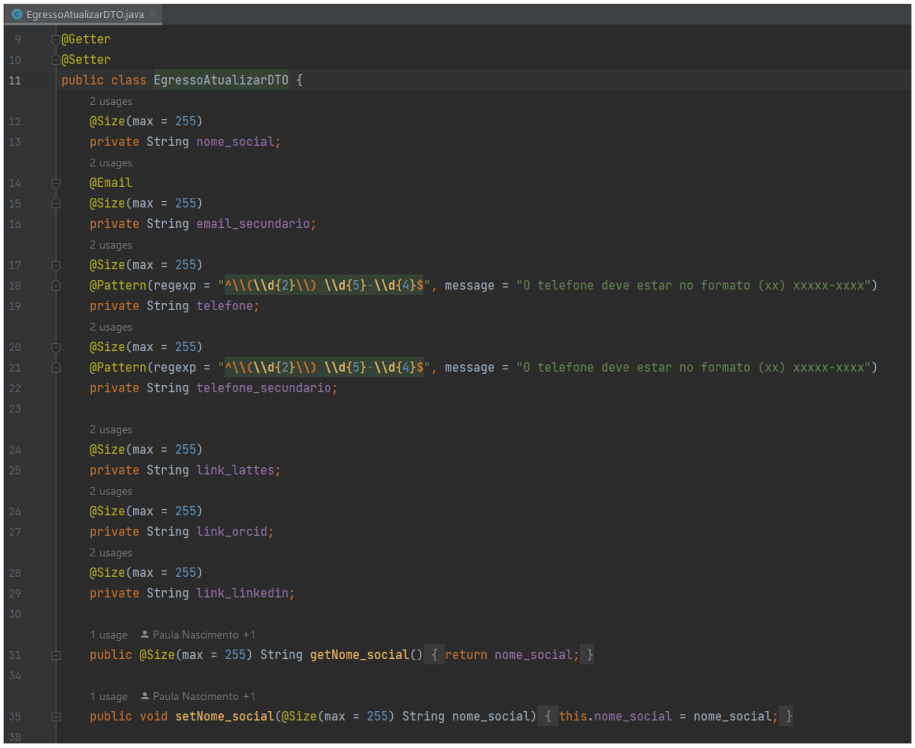

Figura 12 – DTO da entidade egresso

Foram implementadas diversas *APIs RESTful*, responsáveis por realizar operações sobre os dados do sistema. Cada API está relacionada a uma entidade e conta com *endpoints* para consulta, criação, atualização e remoção de informações. A seguir, são detalhados os *endpoints* implementados.

3.2.1 API de Informação Acadêmica

Esta API é responsável pelo gerenciamento das informações acadêmicas dos egressos e está disponível no caminho /api/informacoes/academicas.

- *GET* - /egresso/{cpf}: Retorna todas as informações acadêmicas vinculadas ao egresso identificado pelo CPF fornecido, conforme demonstrado na Figura 13.
- *GET* - /editar/{id}: Recupera os dados de uma informação acadêmica específica. Esse *endpoint* é utilizado no processo de edição, preenchendo o formulário com os dados previamente cadastrados. Um exemplo de utilização é mostrado na Figura 14.
- *POST*: Cria uma nova informação acadêmica no banco de dados, conforme mostrado na Figura 15.

- *PUT* - /{id}: Atualiza uma informação acadêmica existente, identificada pelo seu UUID. Na Figura 16 é apresentada a atualização da cidade e do estado da informação acadêmica cadastrada na Figura 15.

- *DELETE* - /{id}: Exclui uma informação acadêmica. A chamada deste *endpoit* é mostrado nas Figuras 17, 18 e 19, para os casos de ID válido, inválido e com dados relacionados, respectivamente.

- *GET* - /sem-depoimento/{cpf}: Retorna todas as informações acadêmicas do egresso que ainda não possuem um depoimento associado, vide Figura 20. Esse *endpoint* é utilizado na etapa de cadastro de depoimentos, garantindo que cada informação acadêmica só possa ter um único depoimento vinculado.

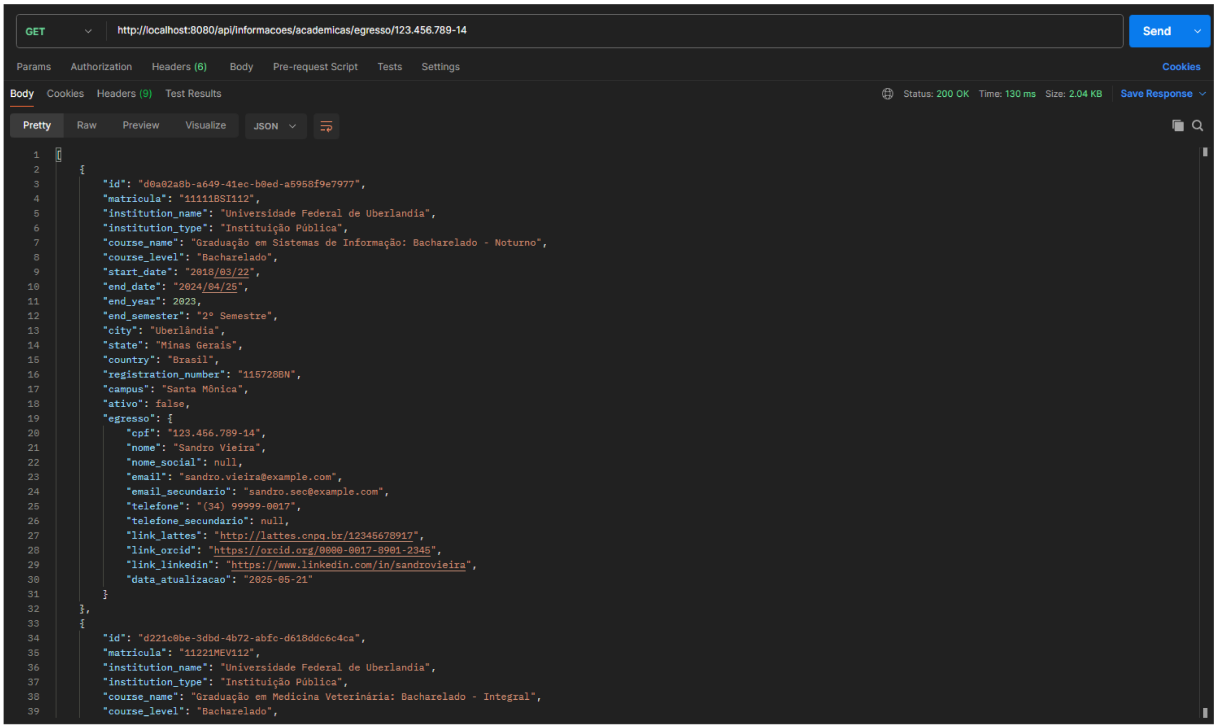

Figura 13 – Chamada do *endpoint* GET /egresso/{cpf} com um CPF válido

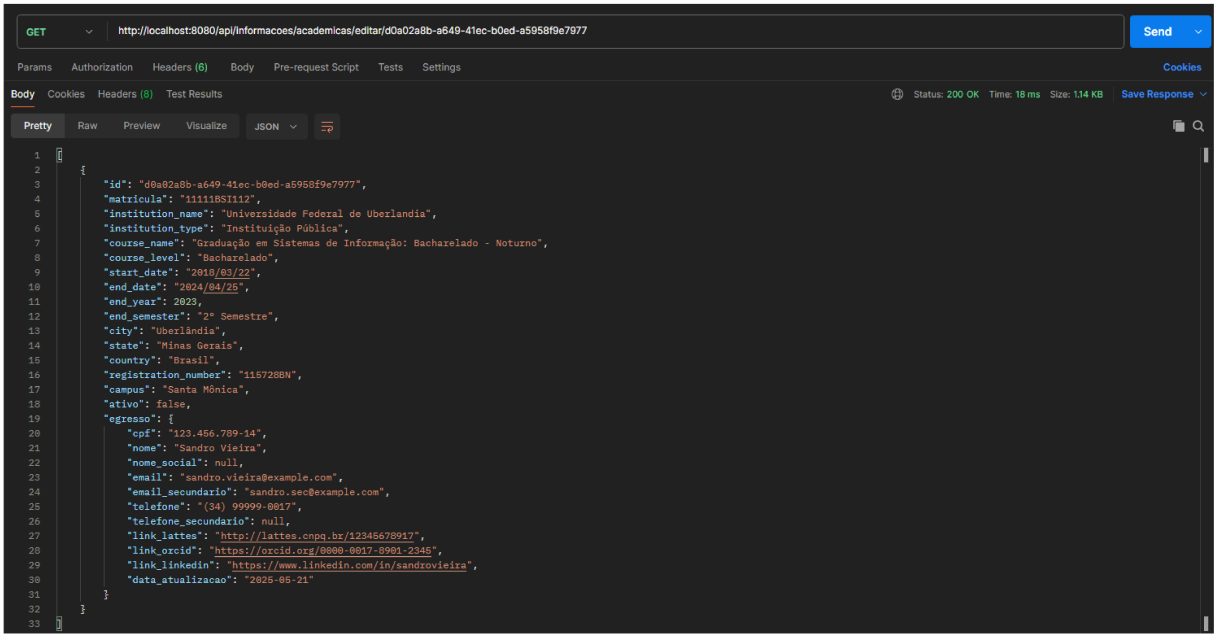

Figura 14 – Chamada do *endpoint* GET /editar/{id} com um ID válido

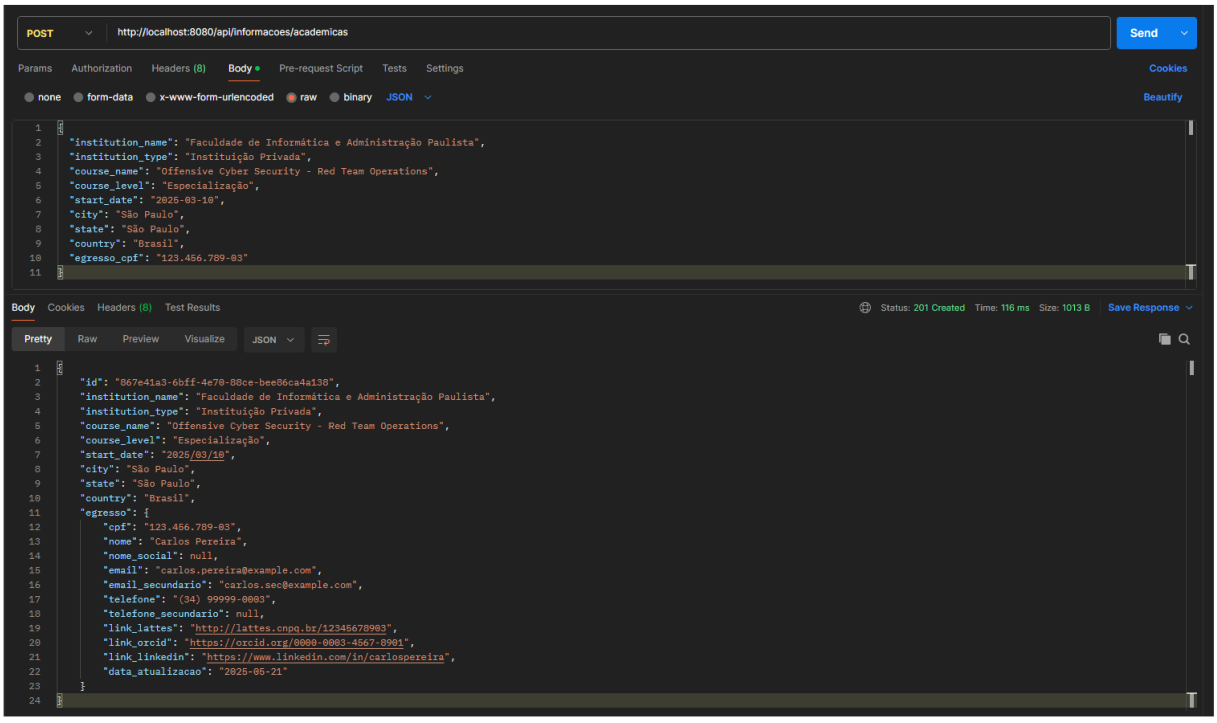

Figura 15 – Chamada do método POST

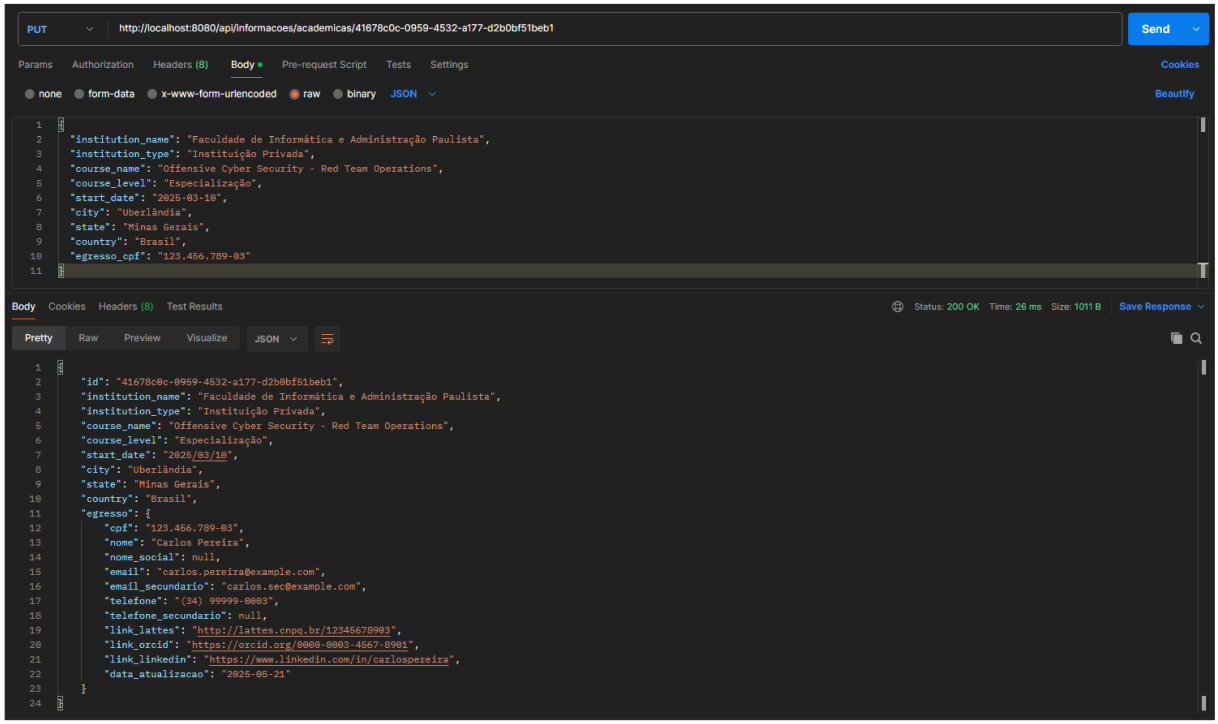

Figura 16 – Chamada do *endpoint* PUT /{id} com ID válido

Figura 17 – Chamada do *endpoint* DELETE /{id} com um ID válido

Figura 18 – Chamada do *endpoint* DELETE /{id} com um ID inválido

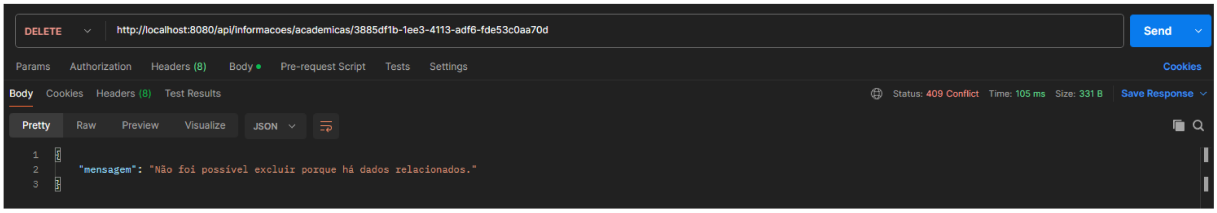

Figura 19 – Chamada do *endpoint* DELETE /{id} com um ID de uma informação acadêmica que possui dados relacionados

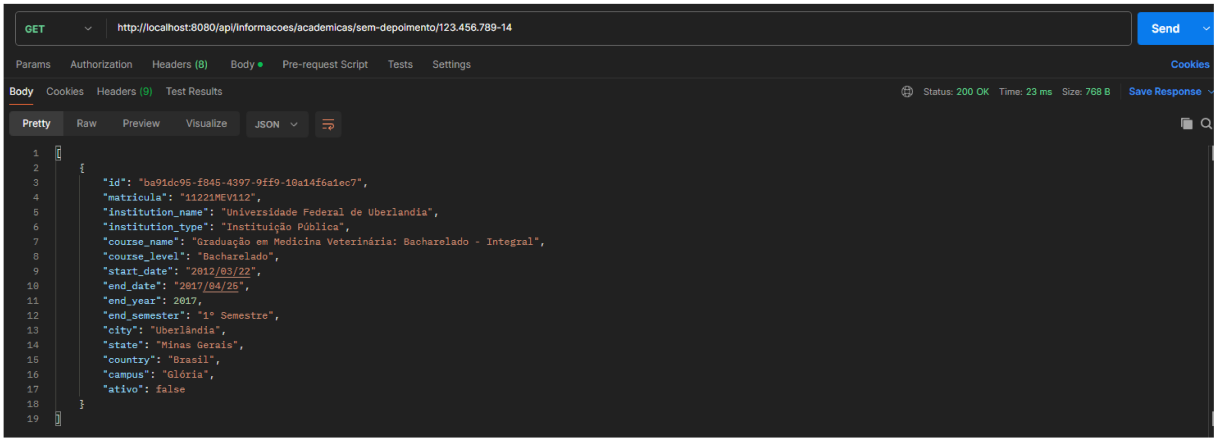

Figura 20 – Chamada do *endpoint* GET /sem-depoimento/{cpf} com CPF válido

3.2.2 API de Egressos

Por meio do caminho /api/egressos, é possível acessar a API responsável pela gestão dos dados dos egressos.

- *GET*: Lista todos os egressos cadastrados, como mostrado na Figura 21.
- *GET* - /{cpf}: Retorna os dados de um egresso específico, com base no CPF.
- *POST*: Cadastra um novo egresso.
- *PUT* - /{cpf}: Atualiza os dados de um egresso.
- *DELETE* - /{id}: Exclui um egresso.
- *GET* - /filtro: Permite realizar buscas com base em filtros como curso, nome, data de ingresso e data de conclusão. Esse recurso é exclusivo para usuários com perfil de coordenador. Na Figura 22 é apresentado um exemplo de busca por egressos onde o campus é "Santa".

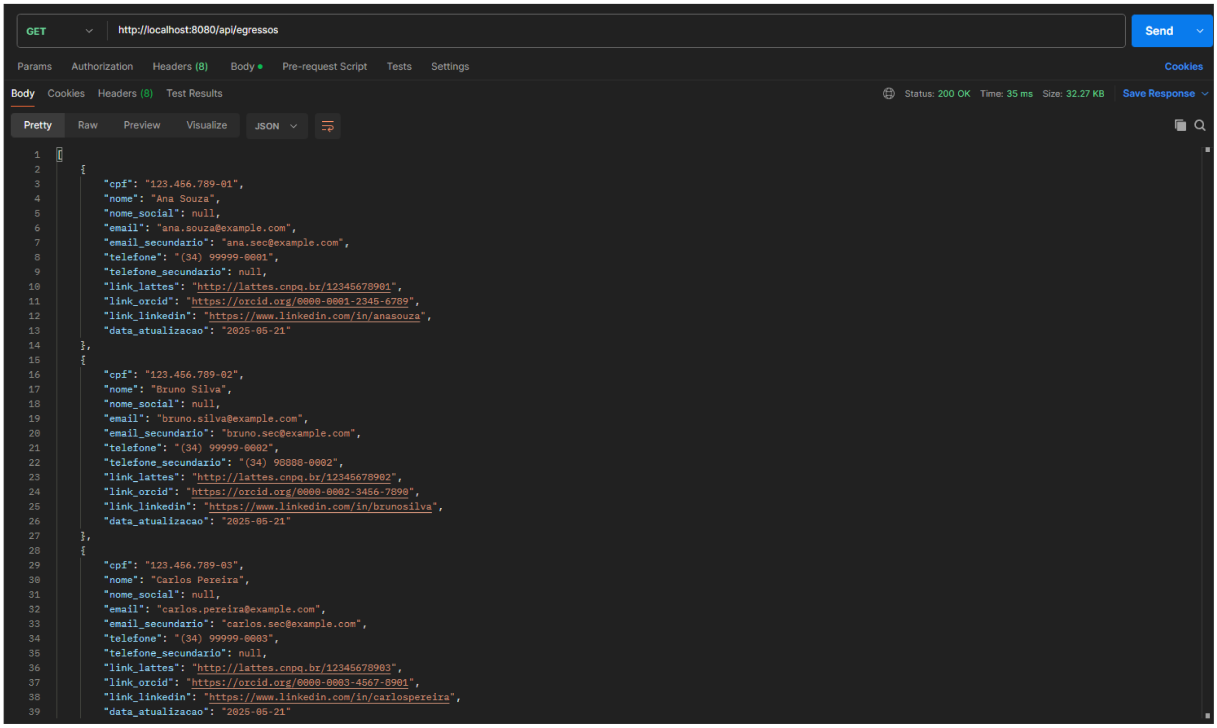

Figura 21 – Chamada do método GET

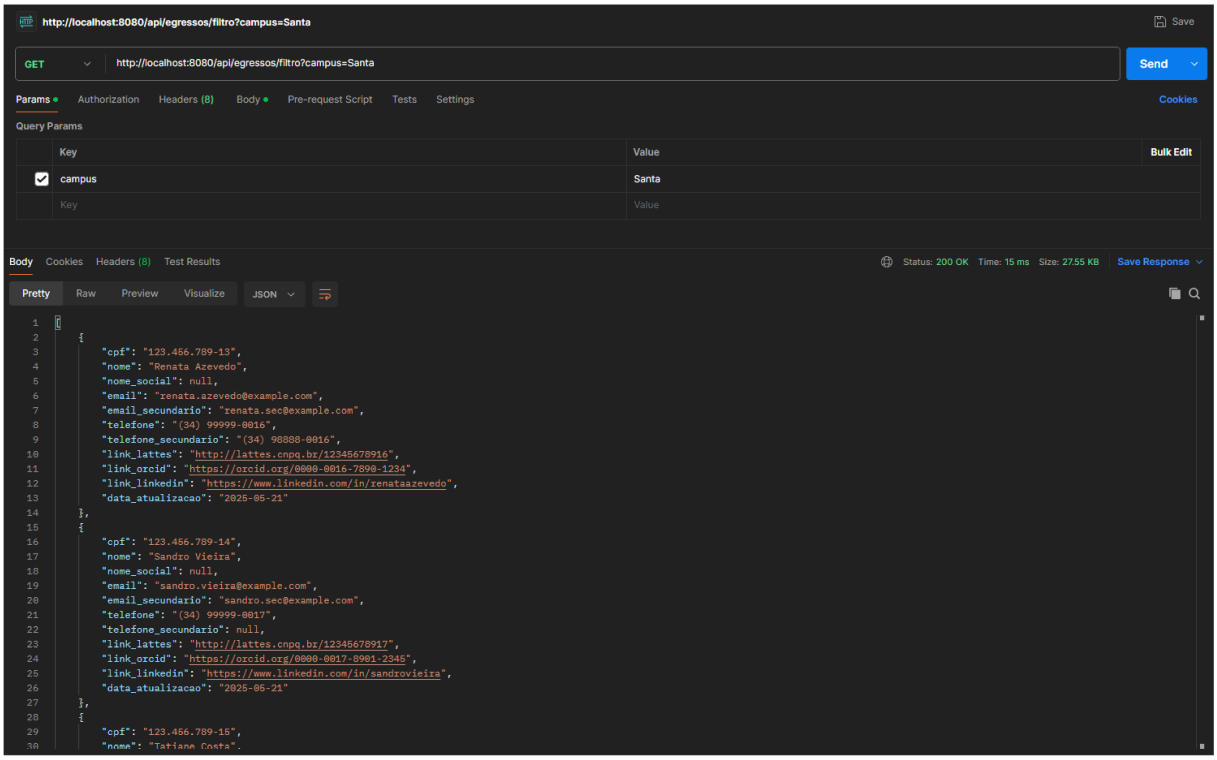

Figura 22 – Chamada do *endpoint* GET /filtro

3.2.3 API de Depoimentos

É acessada por meio do caminho /api/depoimentos e lida com os depoimentos cadastrados pelos egressos.

- *GET*: Lista todos os depoimentos cadastrados no sistema. Permite aplicar filtros como campus, curso, semestre letivo e titulação. Esse *endpoint* é utilizado no dashboard.
- *GET* - /{id}: Retorna um depoimento específico com base no seu identificador.
- *POST*: Cadastra um novo depoimento.
- *PUT* - /{id}: Atualiza um depoimento.
- *DELETE* - /{id}: Remove um depoimento.
- *GET* - /egresso/{cpf}: Retorna todos os depoimentos de um egresso específico.

3.2.4 API de Informação Profissional

A API de informações profissionais, gerencia os dados sobre as experiências profissionais dos egressos e está acessível por meio do caminho /api/informacoes/profissionais.

- *GET*: Lista todas as informações profissionais registradas.
- *GET* - /editar/{id}: Retorna os dados de uma informação profissional específica, geralmente utilizada no processo de edição.
- *GET* - /egresso/{cpf}: Retorna todas as informações profissionais de um egresso específico.
- *POST*: Cadastra uma nova informação profissional.
- *PUT* - /{cpf}: Atualiza uma informação profissional cadastrada anteriormente.
- *DELETE* - /{id}: Exclui uma informação profissional do banco de dados.

3.2.5 API de Publicações

A API de publicações, localizada em /api/publicacoes, permite que egressos registrem seus artigos, livros, resumos e demais produções acadêmicas e científicas.

- *GET* - /egresso/{cpf}: Retorna as publicações associadas a um egresso.
- *POST*: Cadastra uma nova publicação.

- *PUT* - /{id}: Atualiza uma publicação.
- *DELETE* - /{id}: Remove uma publicação.

3.2.6 API de Comunicados

Por meio do endereço /api/comunicados, é possível utilizar a API de comunicados, voltada para a criação e consulta de mensagens direcionadas aos egressos.

- *GET*: Lista todos os comunicados registrados.
- *POST*: Cria um novo comunicado.
- *GET* - /filtro: Permite buscar comunicados com base em filtros como curso, nível do curso e se a mensagem é direcionada a todos os egressos ou apenas a um grupo específico. A chamada deste *endpoit* é mostrado nas Figuras 23 e 24, para os casos do comunicado ser para todos e limitado por curso, respectivamente.

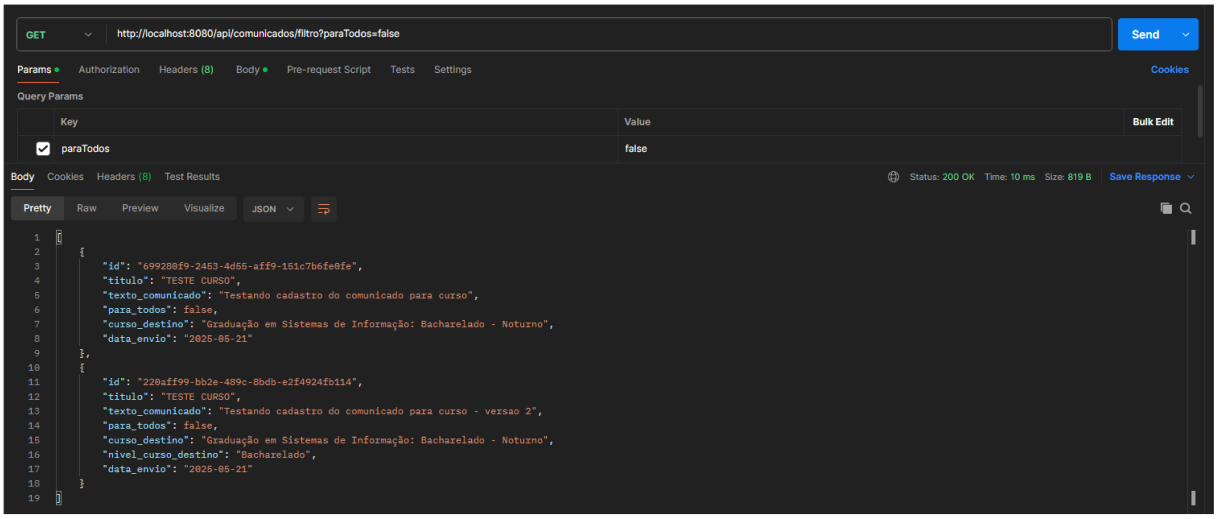

Figura 23 – Chamada do *endpoint* GET /filtro com o filtro para todos

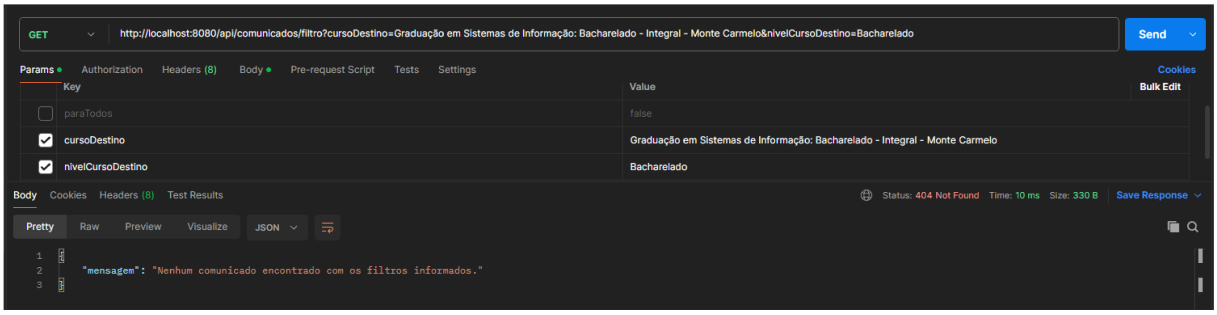

Figura 24 – Chamada do *endpoint* GET /filtro limitando pelo nome e nível do curso

3.3 Testes e Documentação

Por fim, para garantir o comportamento esperado das funcionalidades implementadas, foram feitos testes unitários para as camadas de controle e serviço. Para a execução, foram utilizadas as bibliotecas JUnit[1], para a estruturação dos testes, e Mockito[2], para a simulação de dependências.

A abordagem de teste para cada camada foi a seguinte:

- **Testes da Camada de Controle**: Utilizando a anotação *@WebMvcTest*, os testes focaram em validar o comportamento dos *endpoints* da *API* de forma isolada. Com o uso de *@MockBean*, a camada de serviço foi simulada, permitindo verificar se os *controllers* respondiam corretamente aos diferentes tipos de requisições HTTP e se delegavam as chamadas adequadamente, sem a necessidade de processar a lógica de negócio real.
- **Testes da Camada de Serviço**: O objetivo foi validar a lógica de negócio contida nas classes de serviço. Através das anotações *@Mock* e *@InjectMocks*, os repositórios de dados foram simulados. Isso permitiu testar as regras de negócio em completo isolamento, confirmando que o serviço interagia corretamente com a camada de dados sem depender de uma conexão real com o banco de dados.

Todos os testes unitários implementados foram executados e passaram com sucesso, validando o comportamento esperado para cada unidade de código testada.

Além disso, foi configurada a documentação das APIs utilizando o Swagger, conforme Figura 25. Por possuir uma interface interativa, ele facilita a visualização, teste manual e o entendimento dos *endpoints* disponíveis. Isso será útil para os futuros mantenedores do sistema.

[1] Disponível em: <https://junit.org>
[2] Disponível em: <https://site.mockito.org/>

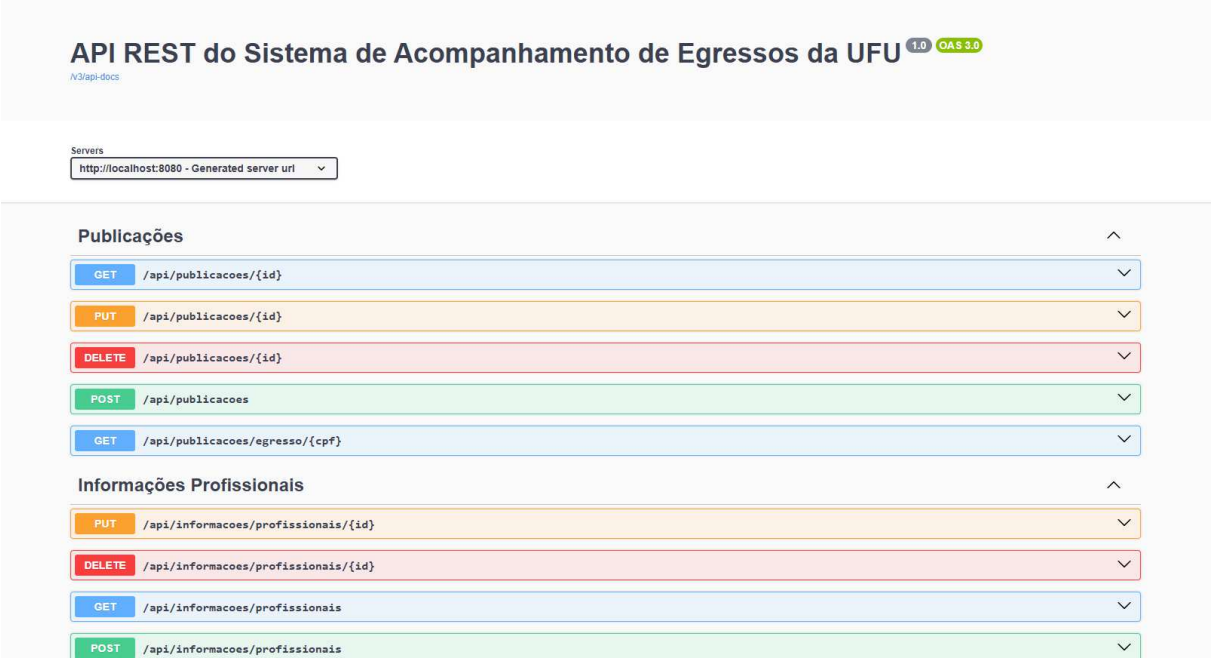

Figura 25 – Documentação das APIs no Swagger

4 Conclusão

Este trabalho teve como objetivo a modelagem de dados e a implementação do *backend* para o sistema de acompanhamento de egressos da Universidade Federal de Uberlândia. O resultado foi uma solução funcional que permite manipular e persistir os dados dos ex-alunos. Isso inclui informações acadêmicas, profissionais, depoimentos e comunicados. Além disso, a implementação conta com testes unitários, tratamento de erros e documentação das *APIs*[1].

A implementação, realizada em conjunto, representou uma valiosa oportunidade de aprendizado em trabalho em equipe. O auxílio mútuo foi essencial para superar dúvidas e dificuldades técnicas. Nesse processo, os conhecimentos da disciplina de Banco de Dados foram fundamentais. A base teórica sobre o modelo relacional, tabelas e relacionamentos foi aplicada diretamente na construção do modelo de dados.

A principal dificuldade durante o desenvolvimento foi a necessidade de refazer o modelo entidade relacionamento mais de uma vez. Durante a implementação, foram percebidas inconsistências que não faziam sentido na prática. Isso exigiu pausas para reformular a estrutura. Esse desafio reforçou um aprendizado da disciplina de Interação Humano-Computador sobre a importância do planejamento. A experiência provou que é crucial dedicar mais tempo à análise e validação da solução antes de começar a codificar. Esse cuidado inicial evita retrabalho e a resistência a mudanças em fases futuras do desenvolvimento.

Como trabalhos futuros, podem ser considerados:

- Integração com o sistema de gestão da UFU, permitindo a verificação automática do vínculo institucional e o cadastro de usuários.
- Limitação da visão dos coordenadores, restringindo o acesso aos egressos de seus respectivos cursos.
- Definição do recebimento dos comunicados pelos ex-alunos, visto que eles estão apenas sendo registrados no banco de dados para histórico, mas não são efetivamente entregues. Algumas soluções possíveis incluem uma área de comunicados visível para o egresso, ums aba de notificações internas ou envio de e-mails usando serviços como Amazon SES.
- Implementação de um mecanismo de importação automática de dados profissionais, reduzindo o esforço manual.

[1] Repositório do sistema desenvolvido: <https://github.com/VitoriaCardoso/sistemaegressos>

- Adição de validação dos campos de localização por meio de APIs externas confiáveis, evitando a inserção de dados incorretos.
- Revisão do relacionamento entre informação profissional e acadêmica para um modelo muitos para muitos, a fim de refletir com maior precisão a realidade dos egressos.

O sistema desenvolvido representa um avanço importante na organização e acompanhamento dos egressos da universidade. Além de fortalecer o vínculo com ex-alunos, a solução atende a critérios exigidos em processos de avaliação externa, como os conduzidos pelo MEC. Contudo, melhorias podem ser aplicadas a fim de garantir maior confiabilidade e usabilidade para os usuários.

Referências

AZIS, A.; SUGIARTI, Y.; KUMALADEWI, N.; HUDA, M. Q. Designing and building an information system of career development and alumni based on android (case study: Information systems department, syarif hidayatullah state islamic university jakarta). In: **The 6th International Conference on Cyber and IT Service Management (CITSM 2018)**. [s.n.], 2018. Disponível em: <https://repository.uinjkt.ac.id/dspace/handle/123456789/58370>. Citado na página 9.

BEZERRA, G. Uma breve história do java. **Medium**, 2022. Disponível em: <https://giulianabezerra.medium.com/uma-breve-historia-do-java-f58d1761154>. Citado na página 10.

BROADCOM. **Spring Framework Overview**. [S.l.], 2024. Acesso em: 2024. Disponível em: <https://docs.spring.io/spring-framework/reference/overview.html>. Citado 2 vezes nas páginas 10 e 11.

CARVALHO, L. A. da S. **Arquitetura Frontend e Interface de um Sistema de Acompanhamento de Egressos para a UFU**. 56 p. Monografia (Graduação em Sistemas de Informação) — Universidade Federal de Uberlândia, Uberlândia, 2025. Disponível em: <https://repositorio.ufu.br/handle/123456789/45437>. Citado 4 vezes nas páginas 8, 14, 17 e 18.

CODD, E. F. Relational database: A practical foundation for productivity. **IBM San Jose Research Laboratory**, 1982. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/1283920.1283937>. Citado 2 vezes nas páginas 12 e 13.

FRANçA, E. Spring: O framework que revolucionou o desenvolvimento java. **DIO**, 2024. Disponível em: <https://www.dio.me/articles/spring-o-framework-que-revolucionou-o-desenvolvimento-java>. Citado na página 11.

GLUSHACH, R. The evolution of java: A historical perspective. **Medium**, August 11 2023. Disponível em: <https://romanglushach.medium.com/the-evolution-of-java-a-historical-perspective-e15c3d7e5f85>. Citado na página 10.

GUIMARãES, M. A. M. **O acompanhamento de egressos como ferramenta de inserção no mercado de trabalho do ponto de vista do setor de estágio e emprego do sistema CEFET/RJ**. Dissertação (Mestrado) — Universidade Federal Fluminense, 2013. Acesso em: 29 set. 2024. Disponível em: <https://app.uff.br/riuff/handle/1/20507>. Citado na página 7.

IBM. **O que é PostgreSQL?** 2024. Disponível em: <https://www.ibm.com/br-pt/topics/postgresql#:~:text=Originalmente%20desenvolvido%20em%201986%20como,ciÃłncia%20da%20computaÃğÃčo%20em%20Berkeley>. Citado na página 13.

Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (Inep). **Roteiro de Autoavaliação Institucional: Orientações Gerais**. 2004. Disponível em: <https://www.gov.br/inep/pt-br/centrais-de-conteudo/acervo-linha-editorial/publicacoes-institucionais/avaliacoes-e-exames-da-educacao-superior/roteiro-de-auto-avaliacao-institucional-2013-orientacoes-gerais>. Citado na página 9.

KAMBALYAL, C. **3-tier architecture**. 2010. Retrieved On, 2. Disponível em: <https://channukambalyal.tripod.com/NTierArchitecture.pdf>. Citado na página 11.

KRATKY, S.; REICHENBERGER, C. Client/server development based on the apple event object model. In: **Atlanta**. [s.n.], 2013. Disponível em: <https://preserve.mactech.com/articles/mactech/Vol.14/14.11/Client-ServerDevelopment/index.html>. Citado na página 12.

MENDES, F. C. **Sistema de acompanhamento de egressos UFU: levantamento e análise de requisitos**. 77 p. Monografia (Graduação em Ciência da Computação) — Universidade Federal de Uberlândia, Uberlândia, 2023. Disponível em: <https://repositorio.ufu.br/handle/123456789/39896>. Citado 4 vezes nas páginas 8, 13, 15 e 20.

MICHELAN, L. S.; HARGER, C. A.; EHRHARDT, G.; MORé, R. P. O. Gestão de egressos em instituições de ensino superior: possibilidades e potencialidades. **INPEAU**, 2009. Acesso em: 6 out. 2024. Disponível em: <http://repositorio.ufsc.br/xmlui/handle/123456789/36720>. Citado 2 vezes nas páginas 7 e 9.

PAUL, J.-J. **Algumas reflexões sobre as relações entre o ensino superior e o mercado de trabalho no Brasil**. [S.l.], 1989. Disponível em: <https://ppgee.poli.usp.br/wp-content/uploads/sites/762/2020/12/dt8908.pdf>. Citado na página 9.

_____. Acompanhamento de egressos do ensino superior: experiência brasileira e internacional. **Caderno CRH**, v. 28, n. 74, 2015. Acesso em: 8 out. 2024. Disponível em: <https://doi.org/10.9771/ccrh.v28i74.19899>. Citado na página 7.

PAUL, J.-J.; FREIRE, Z. D. R. **O Mercado de Trabalho para os Egressos do Ensino Superior de Fortaleza**. [S.l.], 1997. 60 p. Disponível em: <https://sites.usp.br/nupps/wp-content/uploads/sites/762/2020/12/dt9701.pdf>. Citado na página 9.

RANTHUM, G.; JUNIOR, G. d. S. Desenvolvimento de uma ferramenta computacional para o acompanhamento de egressos da pós-graduação stricto sensu. **Educere - Revista da Educação da UNIPAR**, 2023. Acesso em: 29 set. 2024. Disponível em: <https://revistas.unipar.br/index.php/educere/article/view/10380>. Citado 2 vezes nas páginas 7 e 14.

SCHMIDT, C. M.; MOURA, J. E. de. Gestão de egressos no ensino superior: Construção teórica e o caso do curso de graduação em secretariado executivo da unioeste - pr. **Revista Expectativa**, 2015. Disponível em: <https://e-revista.unioeste.br/index.php/expectativa/article/view/13263/9474>. Citado na página 9.

SILVA, E. C. d.; MINEIRO, A. A. d. C.; FAVARETTO, F. Graduate monitoring systems in higher education institutions: an integrative review. **Research, Society and Development Journal**, 2022. Acesso em: 10 out. 2024. Disponível em: <https://rsdjournal.org/index.php/rsd/article/view/26281>. Citado na página 14.

SILVA, J. F. d. **Desenvolvimento de uma ferramenta de acompanhamento sistemático de egressos em programa de pós-graduação em saúde**. Monografia (Monografia de Pós-Graduação) — Universidade Federal do Rio Grande do Sul, 2023. Acesso em: 10 out. 2024. Disponível em: <https://lume.ufrgs.br/handle/10183/259891>. Citado na página 14.

SILVA, J. M. da; BEZERRA, R. O. Sistema de acompanhamento dos egressos aplicado na universidade federal de santa catarina. **Revista GUAL**, Florianópolis, 2015. Disponível em: <https://periodicos.ufsc.br/index.php/gual/article/view/41923>. Citado na página 9.

SILVA, J. M. da; NUNES, R. da S.; JACOBSEN, A. de L. O programa de acompanhamento dos egressos da universidade federal de santa catarina: A definição perfil dos estudantes no período 1970-2011. **Repositório Institucional da UFSC**, INPEAU, 2011. Disponível em: <https://repositorio.ufsc.br/xmlui/handle/123456789/25981?show=full>. Citado na página 9.

SmartBear Software. **About Swagger**. 2025. Disponível em: <https://swagger.io/about/>. Citado na página 12.

_____. **What Is OpenAPI?** 2025. Disponível em: <https://swagger.io/docs/specification/v3_0/about/>. Citado na página 12.

The PostgreSQL Global Development Group. **Documentação do PostgreSQL 14.5**. Tradução para o português do brasil. The PostgreSQL Global Development Group, 2024. Edição: 31/10/2024, Copyright © 1996–2024 The PostgreSQL Global Development Group. Disponível em: <https://halleyoliv.gitlab.io/pgdocptbr/history.html>. Citado na página 13.

THIAGO. Java: história e principais conceitos. **DevMedia**, 2012. Disponível em: <https://www.devmedia.com.br/java-historia-e-principais-conceitos/25178#:~:text=A%20ideia%20principal%20do%20Java,despertador%2C%20sendo%20algo%20do%20gÃłnero.> Citado na página 10.

VICTOR, J. Spring framework, history, and its structure. **Dev**, 2023. Disponível em: <https://dev.to/jeanv0/spring-framework-history-and-its-structure-361>. Citado na página 10.

ZHANG, H. Architecture of network and client-server model. **arXiv preprint**, arXiv:1307.6665, 2013. Disponível em: <https://arxiv.org/abs/1307.6665>. Citado na página 11.