UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO

ENZO MORGAN SANTOS DE MELO

SISTEMA PARA GESTÃO E PRESTAÇÃO DE CONTAS UTILIZANDO DJANGO REST *FRAMEWORK*

PAU DOS FERROS - RN
2026

ENZO MORGAN SANTOS DE MELO

SISTEMA PARA GESTÃO E PRESTAÇÃO DE CONTAS UTILIZANDO DJANGO REST *FRAMEWORK*

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Tecnologia da Informação.

Orientador: Reudismam Rolim de Sousa, Prof. Dr.

PAU DOS FERROS - RN
2026

M 528s

Melo, Enzo Morgan Santos de .
Sistema para gestão e prestação de contas utilizando django rest framework / Enzo Morgan Santos de Melo. - 2026.
48 f. : il.

Orientador: Reudismam Rolim de Sousa.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2026.

1. Gestão de projetos. 2. Prestação de contas. 3. Sistemas de informação. 4. Django rest framework. 5. Back-end. I. Sousa, Reudismam Rolim de , orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

ENZO MORGAN SANTOS DE MELO

SISTEMA PARA GESTÃO E PRESTAÇÃO DE CONTAS UTILIZANDO DJANGO REST *FRAMEWORK*

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Tecnologia da Informação.

Defendida em: **06/07/2026**.

BANCA EXAMINADORA

_____
Reudismam Rolim de Sousa, Prof. Dr. (UFERSA)
Presidente

_____
George Felipe Fernandes Vieira, Prof. Me. (UFERSA)
Membro Examinador

_____
Bruno Borges Silva, Prof. Esp. (UFERSA)
Membro Examinador

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à minha mãe, Lúcia, e aos meus irmãos, Vittor, Miciel e Pablo, pelo apoio incondicional durante toda a minha trajetória acadêmica. Foram eles que estiveram ao meu lado nos momentos mais difíceis, oferecendo incentivo, compreensão e força para que eu pudesse continuar seguindo em frente. Sem o amor, a paciência e o apoio da minha família, concluir essa jornada não teria sido possível.

Agradeço também aos meus amigos Andrey, Francisco Afonso, Roberto, Mikael, Edglê e Eriky, que fizeram parte importante dessa caminhada. Cada conversa, ajuda, troca de experiências e incentivo contribuíram para meu crescimento pessoal e profissional, despertando em mim a vontade constante de aprender, evoluir e buscar novos conhecimentos na área.

Expresso minha gratidão à banca examinadora, pela disponibilidade, pelo tempo dedicado e pela presença neste momento tão importante da minha formação acadêmica. De forma especial, agradeço ao Prof. Dr. Reudismam Rolim de Sousa, pelas orientações, ensinamentos, paciência e apoio ao longo da minha jornada na UFERSA. Sua contribuição foi essencial para meu desenvolvimento acadêmico e para a realização deste trabalho.

Por fim, mas certamente uma das pessoas mais importantes durante todo esse percurso, agradeço profundamente à Ysabelita. Sua presença foi fundamental em todos esses anos, sendo uma das maiores incentivadoras da minha caminhada. Foi quem acompanhou de perto cada etapa, cada desafio, cada conquista e cada dificuldade enfrentada ao longo dessa jornada. Esteve presente nos meus momentos mais felizes, celebrando comigo cada vitória, mas também nos momentos mais difíceis, oferecendo apoio, compreensão, palavras de incentivo e força quando eu mais precisava. Sua paciência, carinho, confiança e constante motivação tiveram um papel indispensável para que eu não desistisse diante das dificuldades. Ter alguém que acreditasse em mim, mesmo nos momentos em que eu duvidava das minhas próprias capacidades, fez toda a diferença. Por isso, deixo aqui minha sincera gratidão, reconhecimento e carinho por toda sua importância nessa trajetória.

Essa conquista também carrega um pouco de cada pessoa que esteve ao meu lado durante essa caminhada. Muito obrigado a todos.

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de um sistema para a gestão e prestação de contas de projetos acadêmicos, utilizando o *framework* Django Rest *Framework*. A proposta fundamenta-se na necessidade de aprimorar os processos de organização, controle e transparência das informações financeiras relacionadas a editais, Planos de Trabalho e despesas, minimizando falhas decorrentes de procedimentos manuais e descentralizados. A metodologia adotada contempla as etapas de modelagem do sistema, desenvolvimento do *back-end*, implementação das funcionalidades essenciais e realização de testes automatizados. No sistema proposto, foram implementados mecanismos de gerenciamento de usuários, editais e Planos de Trabalho, bem como o registro e acompanhamento de despesas, com a possibilidade de geração de relatórios em PDF e XLSX. Ademais, o sistema implementa mecanismos de notificação por meio do correio eletrônico, visando otimizar a comunicação entre os usuários e o acompanhamento de pendências. Espera-se que a solução desenvolvida contribua para a melhoria do processo de prestação de contas, promovendo maior confiabilidade, eficiência e organização na gestão de recursos financeiros.

Palavras-chave: gestão de projetos; prestação de contas; sistemas de informação; django rest framework; back-end.

ABSTRACT

This work aims to develop a system for the management and accountability of academic projects, using the Django Rest Framework. The proposal is based on the need to improve the processes of organization, control, and transparency of financial information related to calls for proposals, work plans, and expenses, minimizing errors resulting from manual and decentralized procedures. The methodology adopted includes the stages of system modeling, back-end development, implementation of essential functionalities, and automated testing. The proposed system implements mechanisms for managing users, calls for proposals, and work plans, as well as the registration and monitoring of expenses, with the possibility of generating reports in PDF and XLSX formats. Furthermore, the system implements notification mechanisms via email, aiming to optimize communication between users and the monitoring of pending issues. It is expected that the developed solution will contribute to improving the accountability process, promoting greater reliability, efficiency, and organization in the management of financial resources.

Keywords: project management; accountability; information systems; django rest framework; back-end.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWS	*Amazon Web Service*
DRF	Django Rest *Framework*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
JSON	*JavaScript Object Notation*
JWT	JSON Web *Token*
ORM	*Object-Relational Mapping*
PDF	*Portable Document Format*
SES	Simple Email Service
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
UFERSA	Universidade Federal Rural do Semi-Árido
XLSX	*Excel Spreadsheet Format*

SUMÁRIO

1 INTRODUÇÃO

A gestão de projetos tem se tornado cada vez mais relevante em diferentes áreas do conhecimento, especialmente em ambientes organizacionais e acadêmicos, em que o planejamento, a organização e o acompanhamento das atividades são fundamentais para o sucesso das iniciativas desenvolvidas. Nesse contexto, os sistemas de informação aplicados à gestão de projetos têm contribuído significativamente para a melhoria do controle das atividades, da organização das informações e apoio à tomada de decisões. A tomada de decisões desempenha um papel central na gestão de projetos, uma vez que influencia diretamente o planejamento, a execução e o encerramento das atividades. Conforme Hamad, Altaheri e Al Shamsi (2025), decisões informadas são essenciais para o sucesso de projetos e a velocidade e a qualidade dessas decisões impactam diretamente nos resultados obtidos.

Segundo Micale *et al*. (2021), os projetos atuais enfrentam um cenário de alta complexidade técnica, em que a exigência por entregas rápidas e pela redução de custos tornou-se uma constante. Os autores destacam que, apesar dessa evolução, a gestão de dados ainda é rudimentar, ocorrendo muitas vezes por meio de processos manuais de coleta e transferência entre diferentes sistemas de software.

De acordo com Basri *et al*. (2024), o investimento global em infraestrutura digital por órgãos públicos trouxe o desafio de alinhar a tecnologia às práticas de gestão. Para os autores, o sucesso desses projetos depende dessa harmonia, mas a complexidade atual exige atenção especial à qualidade da informação e ao nível de preparo dos usuários que operarão os sistemas.

Diante desse cenário, o desenvolvimento de soluções tecnológicas voltadas para a gestão de projetos torna-se uma alternativa relevante para apoiar a organização e o acompanhamento das atividades realizadas. Sistemas computacionais podem facilitar o registro de informações, o controle de prazos, o acompanhamento das etapas dos projetos e a comunicação entre os participantes envolvidos no processo. Além disso, com o avanço da tecnologia da informação, os sistemas de informação passaram a oferecer recursos como o acesso a dados em tempo real, geração de relatórios e suporte analítico, contribuindo diretamente para decisões mais rápidas, precisas e baseadas em informações confiáveis. (Hamad; Altaheri; Al Shamsi, 2025).

Dentre os ambientes gerenciais, a prestação de contas se torna essencial para o desenvolvimento da Ciência, Tecnologia e Inovação (CTI), permitindo gerenciar os recursos alocados ao desenvolvimento de iniciativas relevantes para a sociedade. Neste contexto, a proposta deste trabalho é o desenvolvimento de um *back-end* para o Sistema de Prestação de Contas, denominado SigContas, que, ao ser integrado a um *front-end*, busca possibilitar o melhor gerenciamento de projetos de CTI. O sistema foi desenvolvido em um estudo de caso, considerando do contexto da Universidade Federal Rural do Semi-Árido (UFERSA), em especial considerando como usuários do sistema a Pró-Reitoria de Pesquisa e Pós-Graduação (PROPPG) e a Pró-Reitoria de Extensão e Cultura (PROEC).

1.1 Justificativa

A prestação de contas em projetos acadêmicos exige organização e controle das despesas realizadas. Com a ausência de ferramentas adequadas pode-se gerar dificuldades no acompanhamento financeiro e na verificação da utilização dos recursos. Além disso, a inexistência de um sistema centralizado pode levar à fragmentação das informações, dificultando o acesso rápido aos dados e comprometendo a confiabilidade das análises realizadas. Em muitos casos, o controle manual ou descentralizado das despesas aumenta a probabilidade de erros, retrabalho e inconsistências, impactando diretamente na eficiência da gestão de projetos. Nesse sentido, a governança financeira desempenha um papel importante ao estabelecer mecanismos de controle e supervisão que contribuem para a organização e integridade das informações (Pratiwi; Haliah; Kusumawati, 2024).

O desenvolvimento do sistema busca facilitar as informações relacionadas aos projetos, editais e despesas geradas. Dessa forma, técnicos administrativos, docentes e pesquisadores poderão acompanhar de forma mais clara e organizada o uso de recursos financeiros destinados ao seu projeto. Além disso, a prestação de contas financeiras assegura que o uso dos recursos seja devidamente registrado e esteja em conformidade com as normas, contribuindo para a confiança das partes interessadas (Pratiwi; Haliah; Kusumawati, 2024).

O uso de Django Rest *Framework* permite a construção de sistemas seguros e integráveis com outras aplicações. Nesse contexto, a transparência é um elemento essencial na gestão financeira, pois permite o acesso claro às informações e contribui para a responsabilização sobre o uso dos recursos (Pratiwi; Haliah; Kusumawati, 2024).

Adicionalmente, a implementação de funcionalidades como a geração automatizada de relatórios e envio de notificações contribui para a agilidade no processo de prestação de contas, permitindo maior controle sobre prazos e pendências. A elaboração de relatórios financeiros claros e acessíveis é fundamental para garantir transparência e permitir a avaliação da situação financeira e do desempenho das atividades (Pratiwi; Haliah; Kusumawati, 2024). Assim, o sistema proposto não apenas organiza as informações, mas também promove maior transparência e eficiência na gestão dos recursos, atendendo às demandas atuais por soluções tecnológicas no contexto acadêmico.

1.2 Objetivos

A seguir estão descritos o objetivo geral da pesquisa e específicos.

1.2.1 Objetivo Geral

O objetivo principal deste trabalho é o desenvolver um *back-end* do sistema dedicado à gestão e prestação de contas de projetos financiados, permitindo o controle das informações, a geração de relatórios e o envio de notificações e pendências;

- Modelar a estrutura, definindo as entidades e relacionamentos entre editais, projetos, usuários e despesas;
- Desenvolver o sistema *back-end* utilizando o framework Django Rest *Framework*
- Implementar mecanismos de autenticação e controle de acesso com base nos perfis de usuários;
- Implementar funcionalidades para o cadastro e gerenciamento de editais;
- Possibilitar a submissão e o gerenciamento de projetos por pesquisadores;
- Implementar o registro e o controle de despesas vinculadas ao projeto;
- Desenvolver a geração de relatórios de despesas nos formatos PDF e XLSX;
- Implementar o envio de notificações por e-mail para os usuários do sistema;
- Realizar testes automatizados para a validação das funcionalidades implementadas.

1.3 Organização do Trabalho

O presente trabalho está estruturado em cinco capítulos. No Capítulo 1, são apresentados a introdução e os objetivos do trabalho, contextualizando a problemática abordada e a proposta da solução desenvolvida. No Capítulo 2, é apresentada a fundamentação teórica, abordando os principais conceitos, tecnologias e trabalhos relacionados ao desenvolvimento do sistema, incluindo temas como sistemas de informação, gestão de projetos, arquiteturas web, APIs REST e as tecnologias utilizadas na implementação da solução. No Capítulo 3, é descrita a proposta de solução, contemplando o levantamento de requisitos, a identificação dos atores do sistema e a modelagem arquitetural, a identificação dos atores do sistema, a modelagem arquitetural e a descrição dos principais componentes responsáveis pelo fundamento da aplicação. No Capítulo 4, são apresentados os resultados da implementação e a avaliação do sistema, demonstrando as funcionalidades desenvolvidas, a arquitetura implementada, os testes realizados e a análise do comportamento da solução proposta. Por fim, o Capítulo 5 apresenta as considerações finais do trabalho, destacando os resultados alcançados, as limitações identificadas e possíveis direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, são apresentados os principais conceitos e tecnologias relacionados ao desenvolvimento do sistema proposto. São descritas as ferramentas utilizadas na implementação da aplicação e sua importância no desenvolvimento do sistema.

2.1 Sistemas de Informações Gerenciais

Os sistemas de informações gerenciais contribuem significativamente para a eficiência da tomada de decisões ao disponibilizar dados em tempo real e análises abrangentes, possibilitando maior agilidade e precisão nos processos decisórios. Além disso, permitem o processamento de grandes volumes de dados e a geração de relatórios relevantes, auxiliando na obtenção de informações estratégicas para o planejamento e controle das atividades organizacionais (Hamdat-Ceskakusumadewi *et al*., 2024).

Outro aspecto importante é a capacidade dos sistemas de informações gerenciais de reduzir a incerteza na tomada de decisões, uma vez que fornecem dados organizados, em que decisões precisam ser tomadas com base em informações confiáveis e atualizadas (Hamdat-Ceskakusumadewi *et al*., 2024).

Dessa forma, a utilização de sistemas baseados nesses princípios torna-se essencial para garantir maior organização, confiabilidade e eficiência na gestão de Planos de Trabalhos acadêmicos, justificando a adoção de soluções tecnológicas no contexto do sistema proposto.

2.2 Python

O Python é uma linguagem de programação de alto nível amplamente utilizada no desenvolvimento de aplicações web, análise de dados, automação e desenvolvimento científico. A linguagem foi criada por Guido van Rossum no final da década de 1980, sendo projetada com foco total na simplicidade e legibilidade do código.

Segundo Menezes (2019), o Python destaca-se por possuir uma sintaxe clara e objetiva, o que facilita o aprendizado e aumenta a produtividade dos desenvolvedores. Além disso, a linguagem possui uma grande quantidade de bibliotecas e *frameworks* que auxiliam no desenvolvimento de diferentes tipos de aplicações.

Com a existência destes *frameworks* robustos, simplifica a implementação de funcionalidades comuns em aplicações modernas, como autenticação de usuários, acesso a banco de dados e criação de APIs — Application Programming Interface (Lutz, 2016).

2.3 Django

O Django é um *framework* de desenvolvimento web escrito em Python, que segue o padrão arquitetural *Model-Template-View* (MTV). De acordo com Holovaty e Kaplan-Moss (2010), o Django foi projetado para permitir que desenvolvedores construam aplicações completas com menos código, fornecendo diversos componentes prontos, como sistema de autenticação, interface administrativa e gerenciamento de banco de dados.

Outro ponto importante do Django é o seu foco em boas práticas de desenvolvimento, oferecendo mecanismos de segurança que auxiliam na prevenção de vulnerabilidades comuns em aplicação web, como ataques de injeção de código e falsificação de requisições (Django Software *Foundation*, 2024).

2.4 Django Rest *Framework*

O Django Rest *Framework* é uma biblioteca que estende as funcionalidades do Django. Segundo Christie (2024), o *framework* oferece ferramentas que facilitam a criação de APIs seguras, incluindo a serialização de dados e controle de permissões.

A utilização de APIs REST (*Representational State Transfer*) tornou-se fundamental em sistemas modernos, pois permite que diferentes aplicações se comuniquem entre si de forma padronizada, possibilitando a integração entre serviços e plataformas distintas.

2.5 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados relacional de código aberto e amplamente utilizado em aplicações corporativas e acadêmicas. Ele é reconhecido por sua confiabilidade e suporte a consultas complexas.

Segundo Caldeira (2015), o PostgreSQL oferece diversas funcionalidades avançadas, como controle de transações, integridade referencial e extensibilidade, permitindo o armazenamento seguro e eficiente de grandes volumes de dados. Essas características tornam o PostgreSQL em uma escolha adequada para sistemas de alto nível.

2.6 Docker

O Docker é uma plataforma de virtualização baseada em contêineres que permite empacotar aplicações juntamente com todas as suas dependências em ambientes isolados.

De acordo com Matthias e Kane (2023), o uso de contêineres facilita o processo de desenvolvimento e implantação de sistemas, pois garante que sua aplicação seja executada da mesma forma em diferentes ambientes, evitando problemas de incompatibilidade entre dependências.

Além disso, o Docker contribui para melhorar a escalabilidade e a portabilidade das aplicações, sendo amplamente utilizado em arquiteturas modernas baseadas em microsserviços.

2.7 Amazon Simple Email Service (SES)

De acordo com a Amazon Web *Services* (2024), o serviço é amplamente utilizado para envio de notificações automáticas em aplicações web, como confirmações de cadastro, alertas de sistema e comunicação com usuários.

A utilização de serviços como o SES possibilita maior confiabilidade e escalabilidade no envio de mensagens eletrônicas em aplicações que necessitam de comunicação constante com seus usuários.

3 METODOLOGIA

A metodologia adotada para o desenvolvimento do *back-end* foi realizada de modo iterativo e incremental, seguindo práticas do Scrum (Schwaber e Sutherland, 2020). A plataforma utilizada para o gerenciamento do projeto foi o Taiga (Taiga Agile, 2025). O ciclo de desenvolvimento por Sprint foi conduzido em intervalos de 15 dias, em que a equipe de desenvolvimento, incluindo o *back-end*, proposta deste trabalho, deveriam implementar.

Inicialmente, foi desenvolvido um levantamento de requisitos para o sistema proposto. Após a definição dos requisitos, a equipe de *design* do projeto gerou uma proposta de ambientes para as funcionalidades prioritárias do projeto. Ao longo das Sprints novas telas foram desenvolvidas. O *back-end* e o *front-end* foram implementados seguindo os elementos definidos pela equipe de *design*, considerando os requisitos levantados. Semanalmente, a equipe de desenvolvimento se reunia para discutir as atividades desenvolvidas, impedimentos e planejamento para as entregas semanais.

3.1 Levantamento de Requisitos

O levantamento de requisitos é uma etapa essencial no desenvolvimento de software, pois permite identificar as necessidades dos usuários, as restrições do sistema e as expectativas das partes interessadas. Segundo Sommerville e Sawyer (1997), a Engenharia de Requisitos envolve atividades como elicitação, análise, especificação e validação, sendo fundamental para alinhar o sistema a ser desenvolvido aos objetivos dos usuários e da organização.

No contexto deste trabalho, os requisitos contemplam funcionalidade implementadas para gerenciamento de pesquisadores, editais, Planos de Trabalho, despesas, relatórios e notificações.

3.1.1 Atores do Sistema

Os atores do sistema representam os usuários que integram diretamente com a aplicação, executando ações de acordo com suas permissões e responsabilidades. A identificação desses atores é importante para compreender quais funcionalidades devem ser disponibilizadas para cada perfil de usuário, além de auxiliar na definição dos requisitos funcionais e regras de acesso.

No sistema proposto, foram identificados dois atores principais: o Administrador e o Pesquisador. O Administrador é responsável pelo gerenciamento e acompanhamento dos

Planos de Trabalho financiados, enquanto o Pesquisador atua na execução dos Planos de Trabalho, no registro de informações e no processo de prestação de contas.

Quadro 1 - Atores

ID	Nome	Descrição
AT01	Administrador	Perfil de usuário responsável por gerenciar e acompanhar os Planos de Trabalho de pesquisa financiados que necessitam realizar prestação de contas.
AT02	Pesquisador	Perfil de usuário responsável por realizar a execução dos Planos de Trabalho, bem como as cotações, aquisições e prestações de contas.

Fonte: Autoria própria (2026).

3.1.2 Requisitos Funcionais

Os requisitos funcionais descrevem os serviços que o sistema deve oferecer e o comportamento esperado diante de determinadas entradas ou situações. Segundo Sommerville (2011), esses requisitos representam aquilo que o sistema deve fazer, estando diretamente associados às funcionalidades esperadas pelos usuários.

Os requisitos funcionais do sistema são apresentados no Quadro 2, contemplando as funcionalidades relacionadas ao gerenciamento de pesquisadores, editais, Planos de Trabalho, despesas, geração de relatórios e envio de notificações.

Quadro 2 - Requisitos Funcionais (continua)

RF01 - Realizar cadastro de Pesquisador	
Atores: Pesquisador, Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir o cadastro de Pesquisadores mediante formulário próprio.	
Dados necessários: Nome completo, SIAPE, CPF, RG, data de nascimento, gênero, nacionalidade, e-mail, área de atuação, departamento, Orcid, currículo Lattes, nome de usuário e senha.	
RF02 - Realizar listagem de Pesquisadores	
Atores: Administrador	**Prioridade:** Média

Quadro 2 - Requisitos Funcionais (continuação)

RF03 - Realizar atualização do cadastro de Pesquisadores	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir que o Pesquisador atualize seus dados de cadastro mediante formulário próprio	
Dados necessários: Nome completo, SIAPE, CPF, RG, data de nascimento, gênero, nacionalidade, e-mail, área de atuação, departamento, Orcid, currículo Lattes, nome de usuário e senha.	
RF04 - Remover cadastro de Pesquisador	
Atores: Administrador	**Prioridade:** Média
Descrição: O sistema deve permitir que o Administrador remova um cadastro de Pesquisadores mediante formulário próprio.	
Procedimentos necessários: Buscar pesquisador (RF05)	
RF05 - Buscar cadastro do Pesquisador	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir a realização da busca por Pesquisadores mediante formulário próprio.	
Dados necessários: Permitir a busca pelo Nome ou matrícula SIAPE ou por Edital contemplado.	
RF06 - Realizar cadastro de Editais	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir o cadastro de Editais mediante formulário próprio.	
Dados necessários: Dados necessários: órgão/unidade (PROPPG/PROEC/ETC) Identificador do edital, objetivo, data de início e vigência do recurso, nome do Plano de Trabalho e valor designado.	
RF07 - Realizar listagem de Editais	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir a listagem de todos os Editais cadastrados utilizando paginação adequada.	
Sem informações adicionais.	

Quadro 2 - Requisitos funcionais (continuação)

RF08 - Realizar atualização do cadastro de Editais	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir que o Administrador atualize os dados de cadastro mediante formulário próprio.	
Dados necessários: Órgão/unidade (PROPPG/PROEC/ETC) Identificador do edital, objetivo, data de início e vigência do recurso, nome do Plano de Trabalho e valor designado.	
RF09 - Remover cadastro de Edital	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir que o Administrador remova um cadastro de Edital mediante formulário próprio.	
Procedimentos necessários: Buscar cadastro de edital (RF10)	
RF10 - Buscar cadastro do Edital	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir a realização da busca por Edital mediante formulário próprio.	
Informações relevantes: Permitir busca pelo identificador do edital.	
RF11 - Realizar Cadastro de Plano de Trabalho	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir o cadastro de Plano de Trabalho mediante formulário próprio.	
Dados necessários: Nome, Pesquisador, Plano de Trabalho associado e valores disponibilizados.	
RF12 - Realizar listagem de Planos de Trabalho	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir a listagem de todos os Planos de Trabalho cadastrados utilizando a paginação adequada.	
Sem informações adicionais.	

Quadro 2 - Requisitos Funcionais (continuação)

RF13 - Realizar atualização do cadastro de Plano de Trabalho	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir que o Administrador atualize os dados de cadastro do Plano de Trabalho mediante formulário próprio.	
Sem informações adicionais.	
RF14 - Realizar atualização da vigência do Plano de Trabalho	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir que o Administrador atualize os dados do cadastro do Plano de Trabalho por solicitação do Pesquisador mediante formulário próprio.	
Sem informações adicionais.	
RF15 - Remover cadastro de Plano de Trabalho	
Atores: Administrador	**Prioridade:** Média
Descrição: O sistema deve permitir que o Administrador remova um Plano de Trabalho mediante formulário próprio.	
Sem informações adicionais.	
RF16 - Buscar cadastro do Plano de Trabalho	
Atores: Administrador	**Prioridade:** Média
Descrição: O sistema deve permitir a realização da busca por Plano de Trabalho.	
Informações adicionais: Permitir busca pelo identificador do Plano de Trabalho.	

Quadro 2 - Requisitos Funcionais (continuação)

RF17 - Realizar cadastro de Aquisição	
Atores: Pesquisador	**Prioridade:** Média
Descrição: O sistema deve permitir o cadastro de Aquisições realizadas pelo Pesquisador mediante formulário próprio.	
Dados necessários: CPF/CNPJ: Preencher com a numeração de CPF ou CNPJ - o sistema deve buscar o Nome / Razão Social, que deve ser idêntico na nota fiscal; O pesquisador deve identificar a despesa em um menu dropdown do grupo de despesas, qual está relacionado ao serviço e/ou produto do referido pagamento. Descrever conforme o plano de trabalho e documento fiscal o conteúdo do serviço e/ou aquisição de produto realizado, para esse pagamento; Id. Doc. Da Despesa: Selecionar em menu dropdown o tipo do documento fiscal: Recibo/Fatura/Nota Fiscal, etc. Selecionar a opção:"Há processo de cotação de preço?" e confirmar (SIM) a opção de vinculação com a cotação de preço já cadastrada anteriormente, quando houver. (Não precisa de cotação de preço: despesas com diárias)\ Inserir Valor conforme Documento Fiscal. O sistema checa valor com a cotação (se houver) e com o plano de trabalho. Anexar o Documento Fiscal. Clique em "Escolher arquivo" para inserir o documento fiscal, correspondente a despesa e relacionado no Plano de Trabalho. Anexar o comprovante de pagamento: Recibo, Comprovante PIX, Comprovante Cartão, etc. À medida que as aquisições/pagamentos são realizados, o saldo no Plano de Trabalho vai mudando.	
RF18 - Realizar listagem de Aquisições	
Atores: Pesquisador	**Prioridade:** Alta
Descrição: O sistema deve permitir a listagem de todas as aquisições cadastradas utilizando paginação adequada.	
Sem informações adicionais.	

Quadro 2 - Requisitos Funcionais (continuação)

RF19 - Realizar atualização do cadastro de Aquisições	
Atores: Pesquisador	**Prioridade:** Alta
Descrição: O sistema deve permitir a listagem de todas as aquisições cadastradas utilizando paginação adequada.	
Sem informações adicionais.	
RF20 - Remover cadastro de Aquisição	
Atores: Pesquisador	**Prioridade:** Alta
Descrição: O sistema deve permitir que o Pesquisador remova um cadastro de Aquisição mediante formulário próprio.	
Sem informações adicionais.	
RF21 - Gerar Relatórios do Plano de Trabalho	
Atores: Pesquisador	**Prioridade:** Média
Descrição: O sistema deve permitir a geração de relatórios em PDF contendo as principais informações sobre o Plano de Trabalho.	
Informações adicionais: Plano de Trabalho executado em Relação de Aquisições / pagamentos Relatório Técnico-Científico Saldo a devolver (precisa gerar GRU?)	
RF22 - Gerar relatórios administrativos	
Atores: Pesquisador	**Prioridade:** Média
Descrição: O sistema deve permitir a geração de diferentes relatórios sobre todos os Planos de Trabalho cadastrados.	
Informações adicionais: Planos de Trabalho próximo ao fim da vigência Planos de Trabalho com planos de trabalho a cadastrar Planos de Trabalho com cotações aguardando validação Pesquisadores inadimplentes (indicar editais).	
RF23 - Notificar Usuários	
Atores: Pesquisador	**Prioridade:** Média
Descrição: O sistema deve notificar todos os usuários envolvidos no gerenciamento de Planos de Trabalho sobre mudanças no estado dele. Enviar notificações por e-mail e alerta no sistema.	

Quadro 2 - Requisitos Funcionais (continuação)

RF24 - Validar Aquisição	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir ao Administrador validar as aquisições do pesquisador.	
Informações adicionais: O Administrador pode validar, solicitar correções ou reprovar a aquisição, podendo justificar os motivos para cada uma das ações listadas.	
RF25 - Validar Prestação de Contas	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir ao Administrador validar a prestação de contas do pesquisador.	
Informações adicionais: O Administrador pode validar, solicitar correções ou reprovar a prestação de contas, podendo justificar os motivos para cada uma das ações listadas.	
RF26 - Validar GRU	
Atores: Administrador	**Prioridade:** Alta
Descrição: O sistema deve permitir ao Administrador validar a GRU de retorno de recurso não gasto.	
Informações adicionais: O Administrador pode validar, solicitar correções ou reprovar a GRI, podendo justificar os motivos para cada uma das ações listadas.	
RF27 - Informar GRU	
Atores: Pesquisador	**Prioridade:** Alta
Descrição: O sistema deve permitir ao Pesquisador cadastrar uma GRU para devolução de recursos recebidos e não gastos.	
Informações adicionais: A GRU é emitida por sistema externo próprio e o comprovante é anexado ao sistema.	
RF28 - Enviar Prestação de Contas	
Atores: Pesquisador	**Prioridade:** Média
Descrição: O sistema deve permitir ao pesquisador enviar a prestação de contas final, após concluído o uso dos recursos do plano de trabalho ou finalize o prazo de vigência do plano de trabalho.	
Informações adicionais: O pesquisador pode fazer essa operação algumas vezes, a depender dos pedidos de correções geradas pelo Administrador.	

Fonte: Autoria própria (2026).

3.1.3 Requisitos Não Funcionais

Os requisitos não funcionais estão relacionados às propriedades e qualidades esperadas do sistema, influenciando aspectos como desempenho, segurança, confiabilidade e manutenibilidade. Glinz (2007) destaca que esses requisitos não descrevem diretamente uma funcionalidade específica, mas afetam a forma como o sistema deve operar e atender às necessidades dos usuários. Para o sistema proposto também foram definidos requisitos não funcionais (Quadro 3). Os requisitos contemplam tanto o *front-end* quanto o *back-end*, mas em específico para o back-end estão aquelas associados à acesso, disponibilidade, segurança, auditoria, backup e desempenho,

Quadro 3 - Requisitos não funcionais (continua)

Usabilidade
[RNF001] - Facilidade de uso
O sistema deve contemplar as 10 Heurísticas de Nielsen (Nielsen, 2024).
Usabilidade
[RNF002] - Acessibilidade
O sistema deve ter pelo menos uma nota superior a 90 de 100 em relação ao WCAG (W3C, 2025).
Usabilidade
[RNF003] - O sistema deve ser responsivo
O sistema não deve apresentar quebras de conteúdos ou dificuldades de acesso, quando vistos por meio de diferentes dispositivos (celulares, tablets e computadores Desktop).
Acesso
[RNF004] - Disponibilidade na Web
O sistema deve estar disponível na Web.
Segurança
[RNF005] - Autenticação
O sistema deve permitir que os usuários se autentiquem para poder utilizar a plataforma.

Quadro 3 - Requisitos não funcionais (continuação)

Segurança
[RNF006] - Controle de Acesso
O sistema só deve permitir o uso dos sistema para os usuários autorizados, devendo também apresentar apenas as informações referente a cada perfil de usuário. Administrador e Pesquisador.
Segurança
[RNF007] - Integridade dos Dados
Os dados não devem sofrer alterações não intencionais ao longo do uso da aplicação.
Segurança
[RNF008] - Auditoria
O sistema deve manter o registro de todas as alterações realizadas pelos usuários no estado do Plano de Trabalho, bem como gerar relatórios sobre as alterações quando solicitado.
Segurança
[RNF009] - Backup
O sistema deve manter backup contínuo, que permita a recuperação dos dados.
Disponibilidade
[RNF010] - Disponibilidade
O sistema deve estar disponível 99,9% do tempo quando solicitado.
Desempenho
[RNF011] - Tempo de acesso
O sistema não deve demorar mais de 5 segundos para responder ao usuário.

Fonte: Autoria própria (2026).

4 PROPOSTA DE SOLUÇÃO

Este capítulo apresenta a proposta de solução, um *back-end*, para o desenvolvimento do sistema de gestão e prestação de contas de projetos acadêmicos. A solução proposta consiste em uma API *back-end* desenvolvida com Django Rest *Framework*, voltada à centralização das informações relacionadas a pesquisadores, editais, Planos de Trabalho, despesas, relatórios e notificações. A proposta tem como finalidade apoiar o processo de prestação de contas de projetos financiados, proporcionando maior organização, transparência e controle das informações. A solução permite otimizar o processo de prestação de contas e facilitar auditorias e acompanhamentos, além de contribuir para a tomada de decisões com base em dados precisos e atualizados. A estrutura da solução está organizada em levantamento de requisitos, definição dos atores do sistema, requisitos funcionais e não funcionais, além da descrição da arquitetura e dos principais componentes responsáveis pelo funcionamento da aplicação.

4.3 Visão de Módulo

No tocante à visão de módulo, o *back-end* foi desenvolvido utilizando Django Rest *Framework* e está estruturado em módulos Django, com separação entre *models*, *serializers*, *services*, permissões e viewsets. A estrutura de arquivos do projeto evidencia essa divisão, contento módulos como core, users, editais e despesas, além de arquivos específicos para models.py, serializers.py, services.py, permissions.py e views.py. Essa organização modular contribui para a manutenção e evolução do sistema, pois separa as responsabilidades de persistência, validação, regra de negócio e comunicação com o cliente.

4.3.1 Camada de Entidades

A camada de entidades (Quadro 4) é responsável por representar os dados persistidos no banco de dados. No Django, essa camada é implementada por meio dos *models*, que definem os atributos, relacionamentos e comportamentos básicos das entidades do sistema.

Quadro 4 - Entidades principais do sistema

Entidade	Módulo	Responsabilidade
BaseModel	*core*	Padronizar campos comuns, como UUID, datas de criação/atualização e usuário criador.
User	*users*	Representar os usuários do sistema, incluindo dados pessoais e acadêmicos.
Membro	*users*	Representar membros vinculados ao sistema, com *status* e dados de contato.
Edital	editais	Representar editais de financiamento e controlar abertura de submissão.
PlanoTrabalho	editais	Representar os Planos de Trabalhos vinculados a editais e coordenadores.
Despesa	editais	Representar despesas vinculadas aos Planos de Trabalho.
Notificação	editais	Representar notificações destinadas aos usuários.

Fonte: Autoria própria (2026).

O sistema utiliza uma classe abstrata chamada *BaseModel*, responsável por padronizar atributos comuns entre as entidades, como identificador único em UUID, data de criação, data de atualização e usuário responsável pela criação de registro. Essa estrutura contribui para a rastreabilidade das informações e facilita o controle das entidades criadas no sistema.

A entidade *User* representa o usuário personalizado da aplicação, estendendo o modelo padrão *AbstractUser* do Django. Esse modelo inclui informações relacionadas ao contexto acadêmico, como nome, data de nascimento, nacionalidade, CPF, SIAPE, telefone, e-mail, área de atuação, departamento, currículo Lattes e ORCID. A partir desse modelo são representados os perfis de usuários do sistema, como gestores e pesquisadores.

Além disso, a entidade Membro representa membros vinculados ao contexto do sistema, contendo informações como nome, e-mail, organização e *status*. Essa entidade segue

a estrutura definida em *BaseModel*, herdando os campos comuns de identificação e rastreabilidade.

No módulo de editais, a entidade Edital representa os editais cadastrados no sistema, contendo número, título, objetivo, órgão/unidade, descrição, datas, valores, *status* e *link*. O modelo também possui controle de abertura de submissão, permitindo indicar quando os pesquisadores podem cadastrar Planos de Trabalho.

A entidade PlanoTrabalho representa os Planos de Trabalhos submetidos ou cadastrados no sistema. Cada Plano de Trabalho está vinculado a um edital e a um coordenador, além de possuir código identificador, órgão/unidade, vigência, *status* e valor solicitado. O código do Plano de Trabalho é gerado automaticamente no formato PEX0001, PEX0002 e assim sucessivamente, facilitando sua identificação.

Por fim, a entidade Notificação representa as comunicações enviadas aos usuários do sistema. Cada notificação possui usuário destinatário, título, mensagem, tipo e indicação de leitura. Essa entidade permite registrar avisos relacionados a mudanças de *status*, pendências e informações gerais.

4.3.2 Camada de Serialização

A camada de serialização é responsável por converter os objetos do sistema em formatos adequados para a comunicação via API, especialmente JSON, além de validar dados recebidos nas requisições. No Django Rest *Framework*, essa camada é implementada por meio dos *serializers*.

No módulo de usuários, os *serializers* são responsáveis por transformar e validar dados relacionados aos usuários, gestores e pesquisadores. O *UserSerializer* apresenta os dados básicos do usuário, enquanto *UserCreateSerializer* e *UserUpdateSerializer* são utilizados para criação e atualização de usuários, incluindo associação a grupos. Também existem *serializers* específicos para gestores e pesquisadores, como *ManagerSerializer*, *ManagerCreateSerializer*, *ManagerUpdateSerializer*, *ResearcherSerializer*, *ResearcherCreateSerializer* e *ResearcherUpdateSerializer*. Esses *serializers* validam campos obrigatórios, senha mínima e formatos de data, além de tratar informações acadêmicas, como departamento, área de pesquisa, currículo Lattes e ORCID.

A autenticação também utiliza um *serializer* específico, o *CustomTokenObtainPairSerializer*, que adapta o processo de obtenção de *tokens* JWT para autenticação por e-mail e senha. Esse *serializer* retorna *tokens* de acesso e atualização necessários para que o usuário autenticado utilize os endpoints protegidos do sistema.

No módulo de editais, a camada de serialização é composta por *serializers* como *EditalSerializer*, *PlanoTrabalhoSerializer*, *DespesaSerializer*, *DespesaCreateSerializer* e *NotificacaoSerialzier*. Esses componentes permitem a transformação dos modelos de editais, Planos de Trabalho, despesas e notificações em dados trafegáveis pela API, além de controlar os campos utilizados em operações específicas, como criação de despesas.

Quadro 5 - Serializers principais

Serializer	**Responsabilidade**
UserSerializer	Exibir dados básicos dos usuários.
UserCreateSerializer	Validar e criar usuários, associando grupos.
UserUpdateSerializer	Atualizar dados do usuário e seus grupos.
ManagerSerializer	Exibir dados de gestores.
ManagerCreateSerializer	Validar dados para criação de gestores.
ManagerUpdateSerializer	Atualizar dados de gestores.
ResearcherSerialzier	Exibir dados de pesquisadores.
ResearcherCreateSerializer	Validar dados para criação de pesquisadores.
ResearcherUpdateSerializer	Atualizar dados de pesquisadores.
CustomTokenObtainPairSerializer	Realizar autenticação JWT utilizando e-mail e senha.
EditalSerializer	Serializar dados de editais.
PlanoTrabalhoSerializer	Serializar dados de Planos de Trabalho.
DespesaSerializer	Serializer dados de despesas.
DespesaCreateSerializer	Controlar campos utilizados na criação de despesas.
NotificacaoSerializer	Serializar dados de notificações.

Fonte: Autoria própria (2026).

4.3.3 Camada Lógica de Negócio e Acesso de Dados

A camada lógica de negócio concentra regras específicas do sistema, evitando que validações e procedimentos mais complexos fiquem diretamente nos controladores da API. No sistema proposto, essa camada é representada principalmente pelos *services*, que centralizam operações relacionadas à criação de usuários e lógica da aplicação.

No módulo de usuários, o *ManagerService* é responsável pelo cadastro de usuários do tipo gestor. Esse service valida se o nome de usuário e a senha foram informados, verifica se o nome de usuário já está em uso, cria o usuário e o associa ao grupo Gestor. De forma semelhante, o *ResearcherService* realiza a criação de usuários pesquisadores, aplicando as mesmas validações e vinculando o usuário criado ao grupo Pesquisador.

No módulo de editais, o *PlanoTrabalhoService* concentra regras importantes para a criação de Planos de Trabalho. Antes de registrar essa entidade, ele verifica se o edital foi informado, valida se o edital existe e impede o cadastro caso a submissão do edital não esteja aberta. Essa regra garante que os Planos de Trabalho sejam cadastrados apenas quando o edital correspondente permitir as submissões.

Quadro 6 - *Services*

Service	**Responsabilidade**
ManagerService	Validar e criar usuários do grupo Gestor.
ResearcherService	Validar e criar usuários do grupo Pesquisador.
PlanoTrabalhoService	Criar Planos de Trabalho vinculados a editais, validando abertura de submissão.

Fonte: Autoria própria (2026)

4.3.4 Camada de Interface com o Cliente

A camada de interface com o cliente é responsável por expor os recursos do sistema por meio de endpoints da API REST. No Django Rest *Framework*. Essa camada é

implementada principalmente por *ViewsSets* e *APIViews*, que recebem requisições HTTP, acionam *serializers* para validação dos dados, aplicam permissões e executam as operações necessárias.

No módulo de usuários, o *UserViewSet* é responsável pelo gerenciamento geral dos usuários, permitindo criação, atualização, listagem, remoção, busca por nome ou nome de usuário e filtro por grupo. Esse *ViewSet* também disponibiliza uma ação personalizada, que retorna os dados do usuário autenticado. O *ManagerViewSet* gerencia usuários do grupo Gestor, enquanto o *ResearcherViewSet* gerencia usuários do grupo Pesquisador, utilizando *services* específicos para a criação desses perfis.

No módulo de editais, o *EditalViewSet* permite gerenciar editais, com suporte a filtros, busca e ordenação. O *PlanoTrabalhoViewSet* gerencia Planos de Trabalho e aplica regras de acesso para que os gestores e administradores visualizem todos os Planos de Trabalho, enquanto pesquisadores visualizam apenas os Planos de Trabalho em que são coordenadores. Além disso, quando há mudança no *status* de um Plano de Trabalho, o sistema cria automaticamente uma notificação para o coordenador.

O *DespesaViewSet* expõe funcionalidades para o gerenciamento de despesas. Ele restringe a visualização de despesas de acordo com o perfil do usuário, permitindo que gestores e administradores visualizem todas as despesas, enquanto pesquisadores visualizam apenas as despesas vinculadas aos seus próprios Planos de Trabalho. O *ViewSet* também impede a atualização de despesas a datas anteriores à data atual.

O *NotificacaoViewSet* permite listar, criar e gerenciar notificações. Ele inclui ações personalizadas para marcar notificações como lidas, listar notificações não lidas, contar notificações pendentes e enviar notificações para múltiplos usuários. Por fim, a *RelatorioDespesaView* é responsável pela geração de relatórios de despesas, permitindo exportação em PDF ou XLSX, com filtros por *status* e Plano de Trabalho.

A camada de permissões complementa essa interface, controlando o acesso às funcionalidades conforme perfis dos usuários. As classes, *IsManager*, *IsAdminOrManager*, *IsResearcher*, *IsAdminOrResearcher* e *ProfilePermission* definem regras de acesso para gestores, pesquisadores, administradores e usuários autenticados.

Quadro 7 - *ViewSets* e *endpoints* principais

Componente	Responsabilidade
UserViewSet	Gerenciar usuários e disponibilizar endpoint do usuário autenticado.
ManagerViewSet	Gerenciar usuários do grupo Gestor.
ResearcherViewSet	Gerenciar usuários do grupo Pesquisador.
EditalViewSet	Gerenciar editais com filtros, busca e ordenação.
ProjetoViewSet	Gerenciar Planos de Trabalho e gerar notificações em alterações de *status*.
DespesaViewSet	Gerenciar despesas e restringir acesso conforme perfil do usuário.
NotificacaoViewSet	Gerenciar notificações, leitura, contagem e envio.
RelatorioDespesaView	Gerar relatórios de despesas em PDF e XLSX.
Classes de Permissão	Controlar acesso às funcionalidades conforme perfil do usuário.

Fonte: Autoria própria (2026).

4.4 Visão Componente e Conector

Referente à Visão Componente e Conector, no sistema proposto, a interação inicia-se por meio de requisições realizadas aos *endpoints* da API REST, envolvendo operações como autenticação, gerenciamento de usuários, editais, Planos de Trabalho, despesas e relatórios. O processo utiliza autenticação baseada em JSON Web *Token* (JWT), liberando o acesso às funcionalidades conforme o perfil do usuário.

Os módulos internos executam as operações relacionadas ao gerenciamento de usuários, editais, Planos de Trabalho e despesas. Em eventos específicos, como alteração de *status* de Planos de Trabalho ou abertura de submissões, o componente de notificações pode ser acionado automaticamente.

A aplicação também integra serviços externos, como o *Amazon Simple Email Service* (SES), utilizado no envio de notificações por e-mail, além de bibliotecas responsáveis pela geração de relatórios em PDF e XLSX.

A persistência dos dados é realizada pelo PostgreSQL, com acesso intermediado pelo ORM do Django, permitindo manipulação das informações sem uso direto de comandos SQL.

Dessa forma, os componentes do sistema atuam de maneira integrada, coordenando autenticação, processamento das regras de negócio, armazenamento dos dados e comunicação com serviços complementares.

Quadro 8 - Componentes e conectores do sistema

Componente	Responsabilidade	Conector
Cliente	Realiza requisições para acesso às funcionalidades do sistema.	HTTP/HTTPS
Autenticação JWT	Validar identidade e permissões dos usuários.	*Token* JWT
API REST	Processar operações solicitadas pelos usuários.	HTTP / JSON
Componente Usuários	Gerenciar perfis e permissões.	ORM Django
Componente Editais/PlanosTrabalho	Gerenciar dados acadêmicos e financeiros.	ORM Django
Componente Despesas	Registrar movimentações financeiras.	ORM Django
Componente Notificações	Gerar avisos e mensagens do sistema.	Amazon SES
Componente Relatórios	Produzir arquivos PDF e XLSX.	*ReportLab* / XlsxWritter
Banco PostgreSQL	Persistir os dados do sistema	ORM Django

Fonte: Autoria própria (2026).

4.5 Visão de Alocação

Em relação à Visão de Alocação, no sistema proposto, o acesso ocorre por meio de clientes responsáveis pelo consumo da API *back-end*, como navegadores, aplicações integradas

ou ferramentas de teste, a exemplo de Postman e Insomnia. A comunicação entre cliente e servidor é realizada via HTTP, com troca de dados em formato JSON.

O processamento da aplicação é executado no servidor *back-end*, desenvolvido com Django Rest *Framework*, responsável por autenticação, regras de negócio, gerenciamento dos módulos do sistema e geração das respostas às requisições. A persistência das informações é realizada pelo PostgreSQL, utilizado para armazenamento dos dados da aplicação, com acesso intermediado pelo ORM do Django.

O sistema também integra serviços externos, como o *Amazon Simple Email Service* (SES), utilizado para envio de notificações eletrônicas aos usuários. Além disso, a solução utiliza Docker para padronização e isolamento do ambiente de execução, facilitando o desenvolvimento e a implantação da aplicação.

Dessa forma, a arquitetura de alocação organiza recursos do sistema em ambientes específicos para consumo da API, processamento da aplicação, persistência dos dados e integração com serviços externos.

Quadro 9 - Alocação dos elementos arquiteturais do sistema

Ambiente	**Tecnologia**	**Responsabilidade**
Cliente	Browser / Postman / Insomnia	Consumir os *endpoints* disponibilizados pela API
Servidor de Aplicação	Django + Django Rest *Framework*	Executar autenticação, regras de negócio e processamento das requisições
Banco de Dados	PostgreSQL	Persistir informações do sistema
Serviço Externo	Amazon SES	Realizar envio de notificações por e-mail
Containerização	Docker	Padronizar e encapsular o ambiente da aplicação

Fonte: Autoria própria (2026).

5 IMPLEMENTAÇÃO E AVALIAÇÃO

Este capítulo apresenta a implementação das funcionalidades desenvolvidas para o sistema SigContas, bem como a avaliação dos resultados obtidos por meio da execução da aplicação e dos testes realizados.

5.1 Implementação das Funcionalidades

A implementação do sistema resultou em uma API REST destinada ao gerenciamento das funcionalidades definidas no documento de requisitos. A documentação da API evidencia a disponibilização dos *endpoints* responsáveis pelas operações de cadastro, consulta, atualização e remoção dos dados do sistema.

As funcionalidades disponibilizadas permitem operações de cadastro, consulta, atualização e remoção dos dados da aplicação, seguindo o padrão CRUD (*Create*, *Read*, *Update* e *Delete*). Além das operações básicas, foram implementados recursos específicos relacionados ao fluxo do sistema, como gerenciamento de notificações, consulta de relatórios e controle de acesso por perfil de usuário.

Figura 1 - *Endpoints* dos sistema (continua)

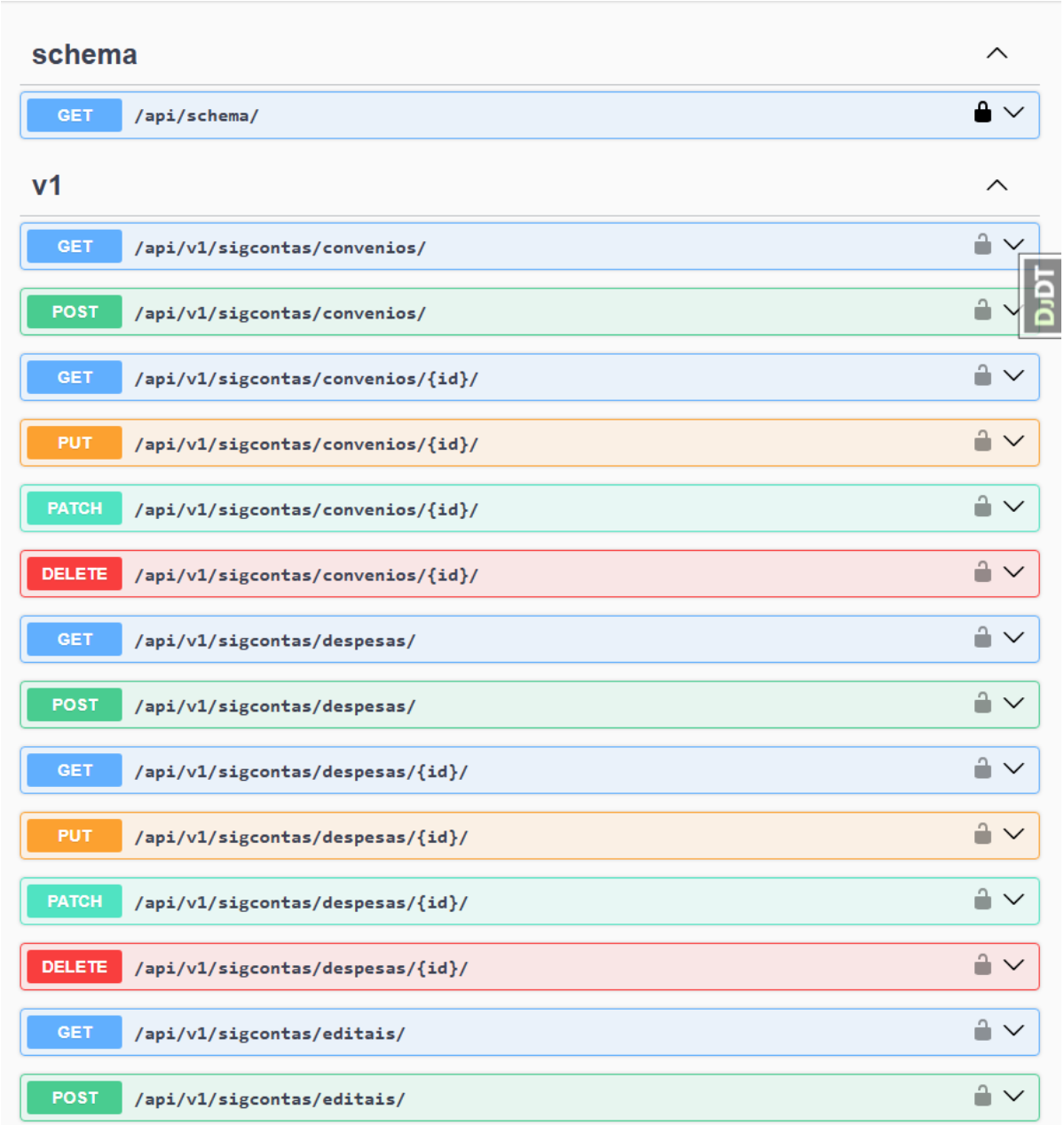


Figura 1 - Endpoints do sistema (continuação)

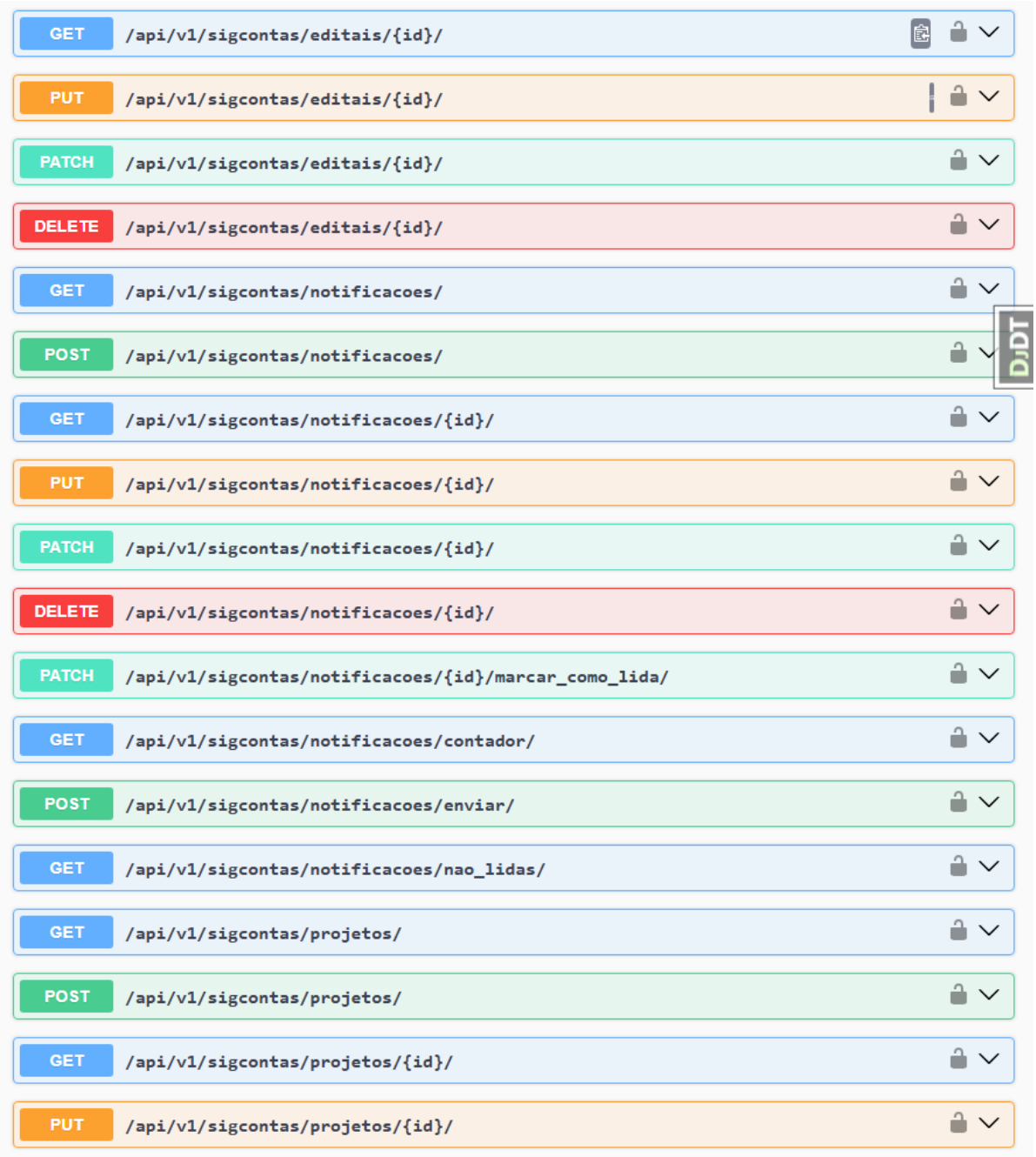


Figura 1 - Endpoints do sistema (continuação)

Fonte: Autoria Própria (2026).

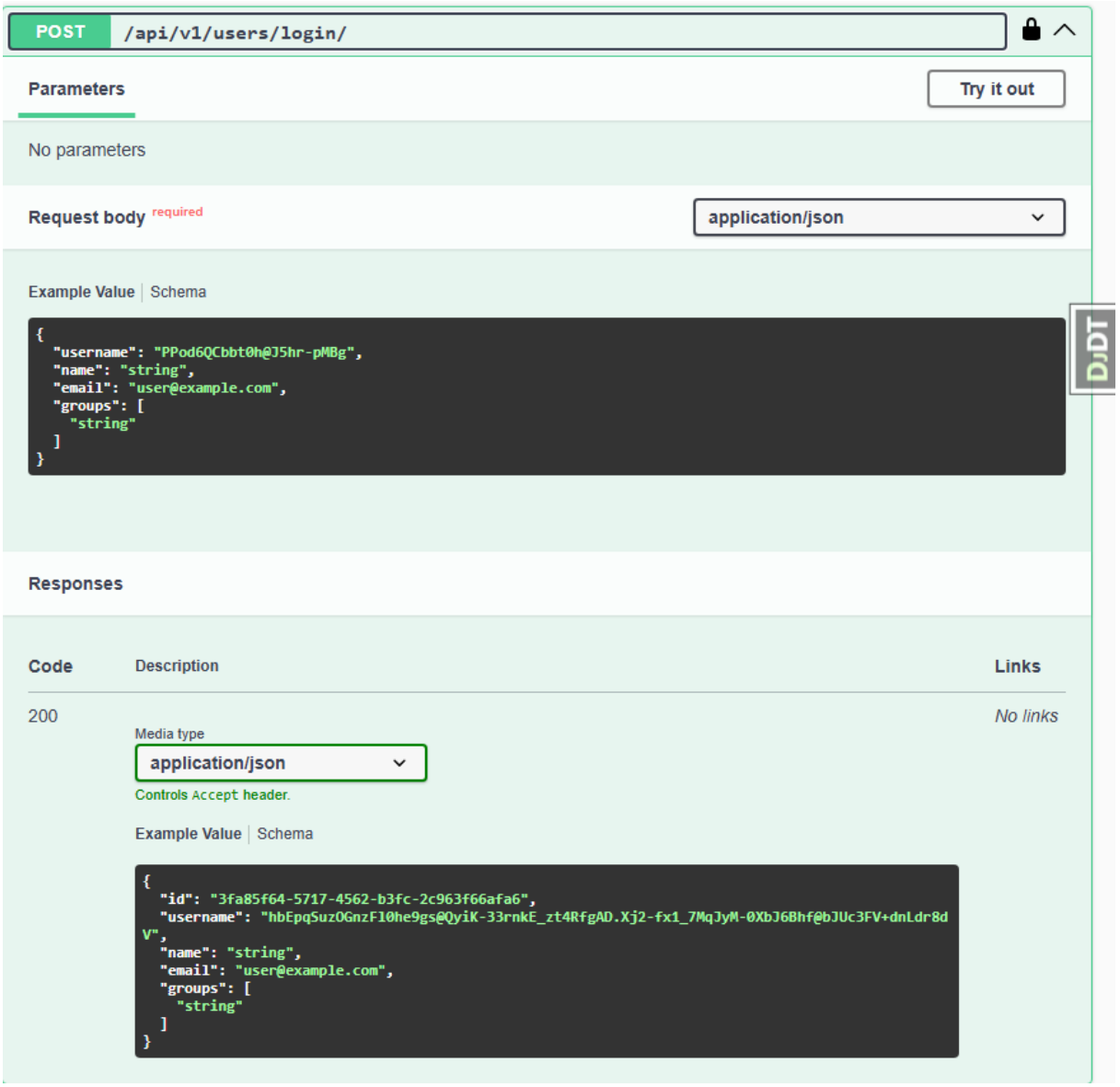
5.2 Avaliação da Implementação

A avaliação da implementação foi realizada por meio da análise dos *endpoints* disponibilizados pela API e das funcionalidades efetivamente implementadas no sistema.

A documentação demonstra a existência de mecanismos de autenticação e gerenciamento de usuários, incluindo operações de *login*, cadastro de pesquisadores, cadastro

de gestores e recuperação das informações do usuário autenticado. O *endpoint* de *login* permite a autenticação dos usuários e o acesso às funcionalidades protegidas da aplicação.

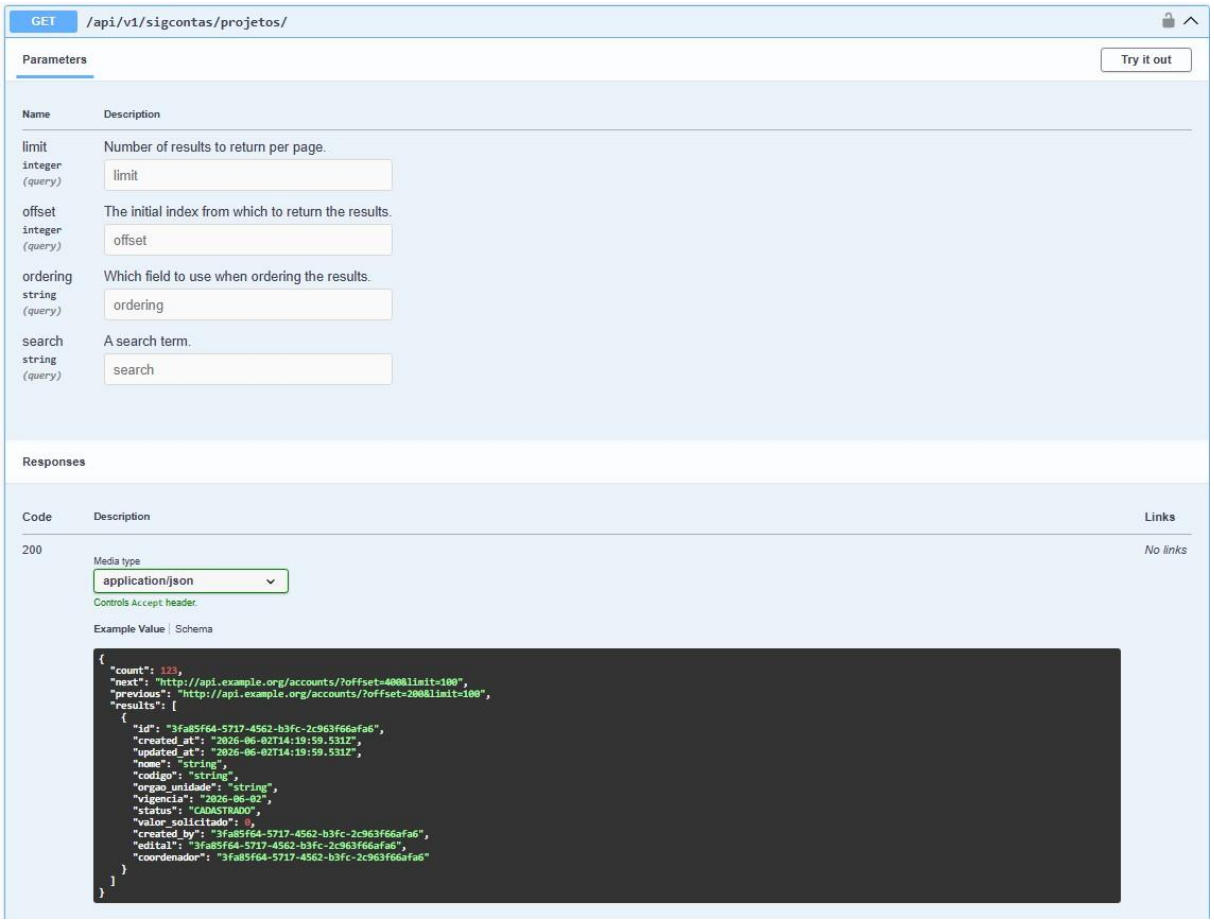
Figura 2 - Endpoints de autenticação de usuários.



Fonte: Autoria Própria (2026).

Além do controle de acesso, a API disponibiliza *endpoints* destinados ao gerenciamento de Planos de Trabalho acadêmicos. A documentação evidencia funcionalidades de listagem, paginação, busca e ordenação de resultados, permitindo o gerenciamento das informações relacionadas aos Planos de Trabalho cadastrados.

Figura 3 - Endpoint de gerenciamento de Planos de Trabalho.



Fonte: Autoria Própria (2026).

Observa-se ainda a implementação de funcionalidades complementares relacionadas a notificações e relatórios, reforçando o suporte do sistema às atividades de acompanhamento e prestação de contas dos Planos de Trabalho.

5.3 Testes de unidade

Conforme destacam Pressman e Maxim (2019), os testes de software desempenham um papel fundamental na verificação da qualidade e da confiabilidade das funcionalidades implementadas. Nesse contexto, foram realizados testes automatizados utilizando *framework* Pytest para validar os módulos desenvolvidos no sistema, contemplando regras de negócio, validações e operações disponibilizadas pela API.

A execução da suíte de testes permitiu validar funcionalidades relacionadas ao cadastro e gerenciamento de usuários, associação de perfis, gerenciamento de editais e

comportamento dos *endpoints* implementados. Os testes também contribuíram para verificar a integridade das regras de acesso e das funcionalidades disponibilizadas pelo sistema.

Figura 4 - Resultado da execução dos testes. (continua)

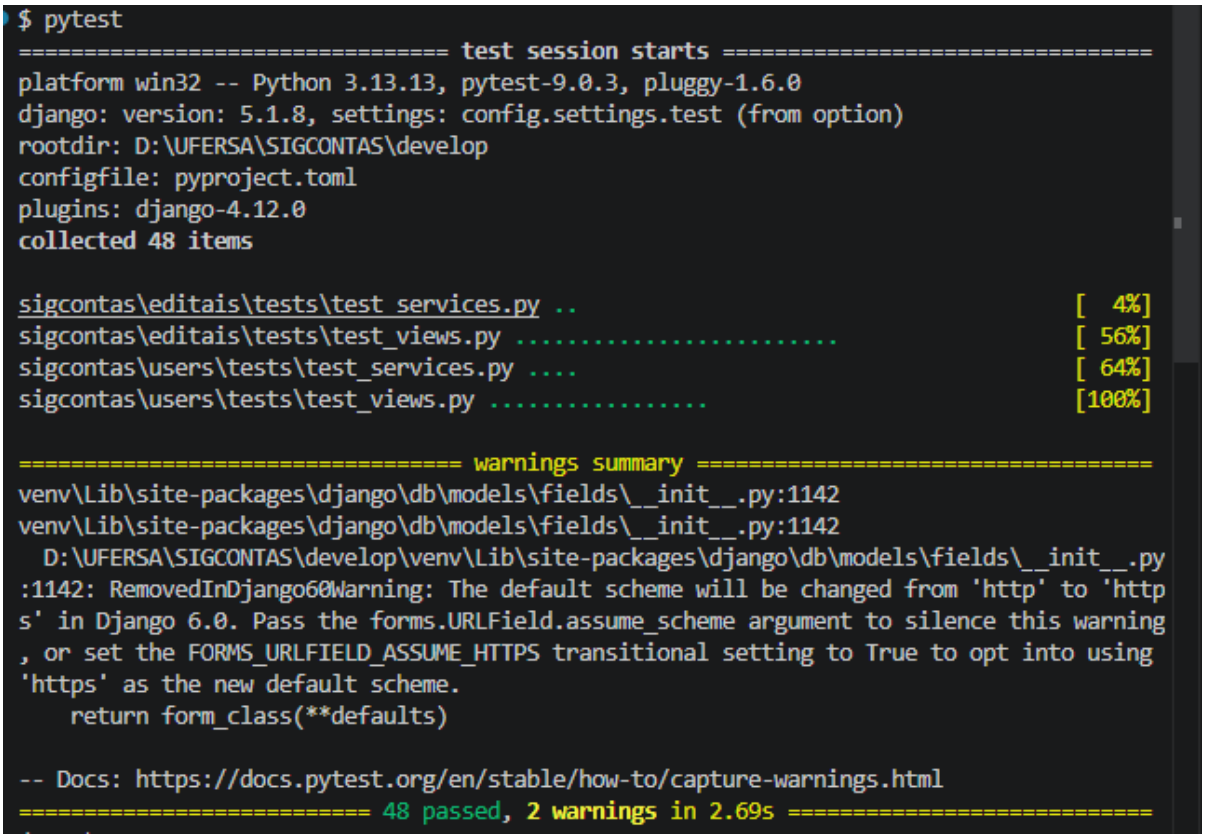
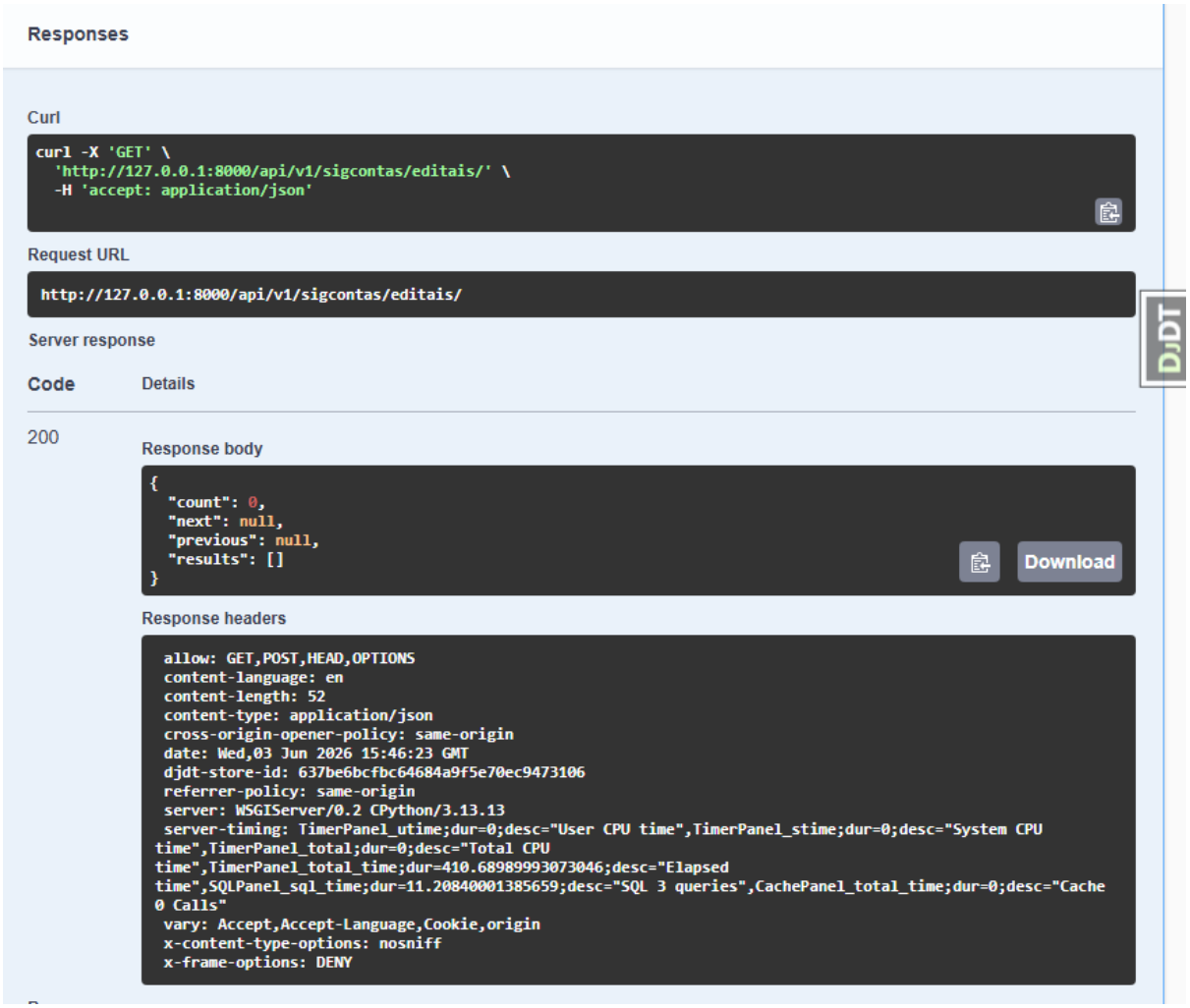


Figura 4 - Resultado da execução dos testes. (continuação)



Fonte: Autoria Própria (2026).

Os resultados obtidos demonstram que os requisitos propostos foram implementados com sucesso. Os testes automatizados executados validam o comportamento esperado dos principais módulos do sistema, evidenciando a viabilidade da solução para apoio à gestão de prestação de contas de projetos acadêmicos.

6 CONSIDERAÇÕES FINAIS

Ao longo deste trabalho, foi proposta e implementada uma solução computacional voltada ao apoio da gestão e prestação de contas de Planos de Trabalhos acadêmicos. O sistema SigContas foi desenvolvido com o objetivo de centralizar informações relacionadas a editais, Planos de Trabalho, despesas, relatórios e notificações, contribuindo para maior organização, controle e acompanhamento dos recursos utilizados em atividades acadêmicas financiadas.

Para a construção da solução, foi desenvolvida uma API REST utilizando Django Rest *Framework*, integrada ao banco de dados PostgreSQL e complementada por mecanismos de autenticação, controle de acesso baseado em perfis de usuário e serviços de envio de notificações. Além disso, foram definidas diferentes visões arquiteturais, permitindo representar a organização estrutural do sistema, seus componentes internos e os ambiente responsáveis por sua execução.

A avaliação da solução foi realizada por meio da implementação dos artefatos arquiteturais propostos, da análise dos *endpoints* disponibilizados pela API e da execução de testes automatizados. Os resultados obtidos demonstram o funcionamento adequado dos módulos desenvolvidos, incluindo gerenciamento de usuários, editais, Planos de Trabalho, despesas, notificações e relatórios, além da validação de regras de negócio e mecanismos de controle de acesso. Durante os testes realizados, foram executados 48 casos de teste automatizados, todos concluídos com sucesso, reforçando a confiabilidade das funcionalidades implementadas.

Durante o desenvolvimento do sistema, foram identificados desafios relacionados à definição e refinamento dos requisitos, à modelagem das regras de negócio e à integração entre os diferentes módulos da aplicação. Além disso, a necessidade de garantir mecanismos de autenticação, controle de permissões, geração de relatórios e envio de notificações exigiu a adoção de soluções que favorecessem a organização, manutenção e escalabilidade do sistema. Esses desafios contribuíram para o aprofundamento dos conhecimentos relacionados à arquitetura de software, desenvolvimento de APIs REST, persistência de dados e testes automatizados.

O sistema encontra-se atualmente em evolução contínua. Embora as funcionalidades principais previstas para o escopo deste trabalho tenham sido implementadas, novas melhorias poderão ser incorporadas futuramente com o objetivo de ampliar as capacidades da aplicação e adequá-la a diferentes cenários de utilização no contexto acadêmico.

Dessa forma, o SigContas configura-se como uma contribuição prática para o gerenciamento e prestação de contas de projetos acadêmicos, oferecendo uma solução baseada em tecnologias modernas e alinhada às necessidades de organização, transparência e acompanhamento das informações financeiras associadas aos projetos.

6.1 Trabalhos Futuros

Embora as funcionalidades previstas para o escopo deste trabalho tenham sido implementadas, existem oportunidades para evolução da solução desenvolvida.

Uma das possibilidades consiste na ampliação das funcionalidades relacionadas à gestão financeira dos projetos, incorporando novas regras de negócio para acompanhamento de despesas, validações automatizadas e mecanismos mais avançados de prestação de contas. O sistema também poderá incorporar *dashboards* gerenciais e indicadores de desempenho que auxiliem gestores e pesquisadores no acompanhamento das informações financeiras e administrativas dos projetos permitindo uma visualização mais estratégica dos dados armazenados.

Outra possibilidade de evolução está relacionada à integração com sistemas institucionais utilizados por universidades e órgãos financiadores, possibilitando maior interoperabilidade entre plataformas e reduzindo a necessidade de atividades manuais de cadastro e atualização de informações. Também poderão ser ampliados os mecanismos de auditoria, rastreabilidade e monitoramento das operações realizadas pelos usuários, contribuindo para maior transparência e controle de processos executados no sistema.

Por fim, futuras versões poderão expandir os recursos de notificações e relatórios, incorporando novas opções de personalização e análise das informações, contribuindo para o aperfeiçoamento contínuo da solução e para o fortalecimento dos processos de gestão e prestação de contas de projetos acadêmicos.

REFERÊNCIAS

AMAZON WEB SERVICES. **Amazon Simple Email Service Developer Guide**. Disponível em: https://docs.aws.amazon.com/ses/. Acesso em: 28 jun. 2026.

BASRI, W. S.; HAMED, A. A.; DANDAN, S. M.; FARAH, A. Determinants of Project Management Information System and Public Project Success. **Journal of Modern Project Management**, 2024. Disponível em: https://www.researchgate.net/publication/385812931_DETERMINANTS_OF_PROJECT_MANAGEMENT_INFORMATION_SYSTEM_AND_PUBLIC_PROJECT_SUCCESS_ROLE_OF_CHANGE_MANAGEMENT_AWARENESS_AND_KNOWLEDGE_OF_SYSTEM_USER. Acesso em: 28 jun. 2026.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software Architecture in Practice**. 4. ed. Boston: Addison-Wesley Professional, 2021. Disponível em: https://www.amazon.com.br/Software-Architecture-Practice-Len-Bass/dp/0321815734. Acesso em: 28 jun. 2026.

CALDEIRA, Carlos Pampulim. **PostgreSQL: guia fundamental**. Lisboa: Edições Sílabo, 2015. 260 p. ISBN 978-972-618-795-0. Disponível em: https://silabo.pt/catalogo/informatica/programacao/livro/postgresql/. Acesso em: 28 jun. 2026.

CHRISTIE, Tom. **Django REST Framework Documentation**. Disponível em: https://www.django-rest-framework.org. Acesso em: 28 jun. 2026.

GLINZ, Martin. On non-functional requirements. In: **IEEE International Requirements Engineering Conference**, 15., 2007, New Delhi. Proceedings [...]. Piscataway: IEEE, 2007. p. 21-26. Disponível em: https://ieeexplore.ieee.org/document/4384163/. Acesso em: 28 jun. 2026.

HAMAD, Hamda; ALTAHERI, Maryam; AL SHAMSI, Shaikha. The impact of information systems on decision-making in project management. **International Journal of Entrepreneurship and Project Management**, v. 10, n. 2, p. 19–33, 2025. Disponível em: https://iprjb.org/journals/IJEPM/article/view/3454. Acesso em: 28 jun. 2026.

HAMDAT-CESKAKUSUMADEWI, Aminuddin; SAMALAM, Abdul Gafar; RIZAL, Muhammad; LAWALATA, Izaac LD. The impact of management information systems on decision-making efficiency. **Vifada Journal of Digital Business and Management**, v. 1, n. 1, p. 56–74, 2024. Disponível em: https://jurnal.vifada.id/index.php/mdb/article/view/102. Acesso em: 28 jun. 2026.

HOLOVATY, Adrian; KAPLAN-MOSS, Jacob. **The Definitive Guide to Django: Web Development Done Right**. Berkeley: Apress, 2010. Disponível em: https://www.amazon.com.br/Definitive-Guide-Django-Development-Second/dp/143021936X. Acesso em: 28 jun. 2026.

LUTZ, Mark. **Aprendendo Python**. 5. ed. Porto Alegre: Bookman, 2016. Disponível em: https://www.oreilly.com/library/view/aprende-python-5a/9798341641457/. Acesso em: 28 jun. 2026.

MATTHIAS, Karl; KANE, Sean. **Docker: Up & Running**. 3. ed. Sebastopol: O'Reilly Media, 2023. Disponível em: https://www.oreilly.com/library/view/docker-up/9781098131814/. Acesso em: 28 jun. 2026.

MENEZES, Nilo Ney Coutinho. **Introdução à programação com Python: algoritmos e lógica de programação para iniciantes**. 3. ed. São Paulo: Novatec, 2019.
DJANGO SOFTWARE FOUNDATION. Django Documentation. Disponível em: https://novatec.com.br/livros/introducao-python-3ed/. Acesso em: 28 jun. 2026.

MICALE, R.; LA FATA, C. M.; LOMBARDO, A.; LA SCALIA, G. Project Management Information Systems (PMISs): A Statistical-Based Analysis for the Evaluation of Software Packages Features. **Applied Sciences**, 2021. Disponível em: https://www.mdpi.com/2076-3417/11/23/11233. Acesso em: 28 jun. 2026.

NIELSEN, Jakob. **10 usability heuristics for user interface design**. Nielsen Norman Group, 2024. Disponível em: https://www.nngroup.com/articles/ten-usability-heuristics/. Acesso em: 28 jun. 2026.

PRATIWI, Rizki Inmas; HALIAH; KUSUMAWATI, Andi. The influence of transparency, governance and financial accountability on financial information management in the public sector. **International Journal of Education and Life Sciences**, v. 2, n. 10, p. 1165–1180, 2024. Disponível em: https://journal.multitechpublisher.com/index.php/ijels/article/view/2571. Acesso em: 28 jun. 2026.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Software Engineering: A Practitioner's Approach**. 9th ed. New York: McGraw-Hill Education, 2019. Disponível em: https://www.amazon.com.br/SOFTWARE-ENGINEERING-PRACTITIONERS-APPROACH-9TH/dp/9355325045. Acesso em: 28 jun. 2026.

SCHWABER, Ken; SUTHERLAND, Jeff. **O Guia do Scrum: o guia definitivo para o Scrum: as regras do jogo**. Novembro de 2020. Disponível em: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-PortugueseBR-3.0.pdf. Acesso em: Acesso em: 28 jun. 2026.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. Disponível em: https://www.amazon.com.br/Engenharia-software-Ian-Sommerville/dp/8579361087. Acesso em: 28 jun. 2026.

SOMMERVILLE, Ian; SAWYER, Pete. **Requirements Engineering: A Good Practice Guide**. Chichester: John Wiley & Sons, 1997. Disponível em: https://www.amazon.com/Requirements-Engineering-Good-Practice-Guide/dp/0471974447. Acesso em: 28 jun. 2026.

TAIGA AGILE, LLC. **Taiga: Agile Project Management Platform**. [S. l.], [2025?]. Disponível em: https://taiga.io/. Acesso em: 28 jun. 2026.

WORLD WIDE WEB CONSORTIUM (W3C). **Diretrizes de Acessibilidade para Conteúdo Web (WCAG) 2.2**. Tradução autorizada em português do Brasil. São Paulo:

Ceweb.br, 2025. Disponível em: https://www.w3.org/Translations/WCAG22-pt-BR/. Acesso em: Acesso em: 28 jun. 2026.