UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Erico Gonçalves Guedes

Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin

Vitória, ES

2026

Erico Gonçalves Guedes

Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Ciência da Computação

Orientador: Prof. Vítor E. Silva Souza

Vitória, ES

2026

Erico Gonçalves Guedes
Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin/ Erico Gonçalves Guedes. – Vitória, ES, 2026-
134 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Vítor E. Silva Souza

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Colegiado do Curso de Ciência da Computação, 2026.

1. Palavra-chave1. 2. Palavra-chave2. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin

CDU 02:141:005.7

Erico Gonçalves Guedes

Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, (dia) de (mês) de (ano):

Prof. Vítor E. Silva Souza
Orientador

Profª. Patrícia Dockhorn Costa
Universidade Federal do Espírito Santo

Kaio Silva Rosa
Universidade Federal do Espírito Santo

Vitória, ES
2026

À minha família, pelo apoio incondicional, e a todos que acreditaram no meu caminho até aqui.

Agradecimentos

Agradeço primeiramente a Deus, por me conceder saúde, força e serenidade para enfrentar os desafios desta jornada.

Agradeço à minha família, pelo suporte constante, pela paciência e pelo incentivo nos momentos mais difíceis. Esse trabalho é também resultado do amor e da confiança que sempre recebi em casa.

Agradeço aos professores do Programa de Pós-Graduação e do curso de Ciência da Computação da UFES, que contribuíram para minha formação acadêmica e profissional, ampliando minha visão crítica e meu compromisso com a qualidade técnica.

Agradeço especialmente ao meu orientador, pela orientação, disponibilidade e contribuições durante o desenvolvimento do TCC, ajudando a transformar ideias em uma proposta consistente e viável.

Agradeço ao Laboratório de Engenharia de Software da UFES (LabES) e às equipes com as quais trabalhei, pela oportunidade de aprender na prática, colaborar em projetos reais e evoluir como desenvolvedor.

Agradeço também à secretaria do PPGI e aos membros da comissão de bolsas, que se disponibilizaram a discutir o processo de gestão de bolsas, esclarecer dúvidas e apresentar necessidades reais que guiaram os requisitos deste módulo.

Por fim, agradeço aos amigos e colegas que, direta ou indiretamente, contribuíram com apoio, revisões, sugestões e motivação ao longo do caminho.

"Nada é tão poderoso quanto uma ideia cujo tempo chegou."
(Victor Hugo)

Resumo

A gestão de bolsas de pós-graduação no PPGI/UFES é tradicionalmente conduzida de forma descentralizada, com apoio de planilhas e comunicação manual, o que dificulta o controle de prazos, a rastreabilidade do histórico de bolsistas e a padronização do processo seletivo e do acompanhamento de vínculos. Este trabalho apresenta o desenvolvimento de um módulo Web, integrado à plataforma Marvin, para apoiar o ciclo de vida de bolsas acadêmicas desde a publicação de editais e inscrição de candidatos até a classificação, concessão e encerramento da bolsa. A solução centraliza informações de editais, candidaturas, resultados e bolsistas, automatiza etapas repetitivas (como geração de classificação) e viabiliza a aplicação de quotas por combinação de nível e categoria. O módulo disponibiliza telas para *overview* de editais, formulário de inscrição, resultados/classificação, listagem e detalhamento de bolsistas, incluindo encerramento com registro obrigatório de motivo e data efetiva. Além disso, utiliza a infraestrutura de e-mail do Marvin, baseada em eventos e templates, para notificações como aceite de bolsa e lembretes de relatório. Para além do código entregue, o trabalho também produziu artefatos de Engenharia de Software ao longo do desenvolvimento: (i) o **documento de requisitos**, que consolida atores/perfis, requisitos funcionais e não funcionais, regras de negócio e estórias de usuário com critérios de aceitação; (ii) o **documento de projeto de arquitetura**, descrevendo a organização em camadas, responsabilidades, componentes e interdependências com o Marvin Core; e (iii) o **apêndice de macrofluxos**, que detalha o encadeamento das principais funcionalidades implementadas e serve como base para comunicação técnica. Os resultados apresentados consistem em verificação interna dos principais fluxos e na preparação de um roteiro de homologação com o cliente, visando validação final de usabilidade e aderência às necessidades operacionais do programa.

Palavras-chaves: Marvin. PPGI. Bolsas. Sistema Web. Gestão Acadêmica. Classificação com quotas. Notificações.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFES	Universidade Federal do Espírito Santo
PPGI	Programa de Pós-Graduação em Informática
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
UML	Unified Modeling Language
DAO	Data Access Object
JPA	Jakarta Persistence API
JSF	Jakarta Server Faces
EJB	Enterprise JavaBeans
CDI	Contexts and Dependency Injection
SMTP	Simple Mail Transfer Protocol
API	Application Programming Interface
CRUD	Create, Read, Update and Delete
MVC	Model-View-Controller
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
IDE	Integrated Development Environment
PDF	Portable Document Format

Sumário

1 Introdução

No contexto acadêmico da Universidade Federal do Espírito Santo (UFES) e, em particular, no Programa de Pós-Graduação em Informática (PPGI), a gestão das bolsas ofertadas aos alunos de pós-graduação representa uma atividade crítica para garantir a continuidade dos programas e a distribuição eficiente de recursos (PPGI/UFES,). Atualmente, o processo de concessão, acompanhamento e encerramento de bolsas é conduzido de forma manual, descentralizada e altamente suscetível a falhas, sendo apoiado principalmente por planilhas dispersas em ambientes compartilhados como o Google Drive. Em contextos organizacionais, a adoção de sistemas de informação visa justamente reduzir retrabalho, aumentar consistência e apoiar a tomada de decisão com dados consolidados (Laudon; Laudon, 2014).

As bolsas, ofertadas por diferentes agências de fomento — como CAPES, CNPq e FAPES —, apresentam requisitos distintos e ciclos de vida específicos, exigindo da **comissão de bolsas** e da **secretaria do programa** um controle rigoroso de prazos, obrigações contratuais e comunicações com os alunos e as próprias agências. Além disso, é necessário acompanhar continuamente os bolsistas, garantindo que eles cumpram obrigações como o envio de relatórios de atividades no meio do período de vigência da bolsa, a conclusão dentro do prazo previsto, ou ainda o correto desligamento em caso de desistência ou formatura. Tais requisitos reforçam a necessidade de atributos de qualidade como rastreabilidade, confiabilidade e manutenibilidade, comuns em normas e modelos de qualidade de software (ISO/IEC, 2011).

Esse cenário fragmentado impõe gargalos à operação da comissão e da secretaria, como: falta de avisos adequados aos alunos e orientadores sobre prazos críticos, incerteza quanto ao status dos bolsistas e a ausência de um repositório confiável e acessível de dados históricos. Por exemplo, não há controle efetivo sobre estudantes que recusaram ou cancelaram bolsas anteriormente, tampouco sobre situações em que um mesmo aluno se inscreve ou é classificado em mais de um edital simultaneamente, o que pode configurar conflito com as regras de fomento. Em sistemas que lidam com decisões administrativas e dados pessoais, mecanismos de registro, auditoria e controle de acesso são recomendados como forma de aumentar rastreabilidade e reduzir riscos operacionais (ISO/IEC, 2022a).

Diante disso, a implementação de um sistema de informação dedicado à **gestão unificada de bolsas de pós-graduação** surge como uma solução estratégica. Tal sistema deve centralizar os dados dos bolsistas, permitir a definição de categorias específicas de bolsa com suas respectivas regras, automatizar tarefas como geração de rankings e apoiar a execução dos fluxos essenciais do processo

A proposta é desenvolver este sistema como um módulo dentro da plataforma **Marvin**, um ambiente já consolidado na UFES para auxiliar na gestão de processos acadêmicos, o que garante maior integração com funcionalidades existentes e aproveitamento da infraestrutura já utilizada em outras áreas (Souza, 2022). Com isso, busca-se não apenas modernizar a gestão de bolsas, mas também conferir maior autonomia aos diferentes perfis de usuário — comissão, secretaria e coordenação —, com interfaces distintas e acesso controlado às informações e funções pertinentes a cada papel.

A construção dessa ferramenta não se restringe ao apoio operacional. Ela também se configura como um instrumento de apoio à tomada de decisão e à manutenção da regularidade institucional junto às agências de fomento, além de facilitar a transição digital e o registro histórico confiável da trajetória dos bolsistas.

1.1 Motivação e Justificativa

O *Marvin* é um sistema de informação desenvolvido pelo Laboratório de Práticas em Engenharia de Software "Ricardo de Almeida Falbo" (LabES) ao longo dos anos, com o objetivo de apoiar atividades administrativas nos cursos da Universidade Federal do Espírito Santo (UFES). Ele integra diversos projetos desenvolvidos por estudantes dos cursos de Ciência da Computação e Engenharia de Computação, sendo utilizado como base para soluções reais de gestão acadêmica (Souza, 2022; Oliveira, 2023; Chane, 2022). No contexto da pós-graduação, foi identificada a necessidade de um novo módulo que atendesse às principais demandas da comissão de bolsas.

Durante reuniões com os stakeholders e análise dos processos atuais, foram evidenciadas diversas dificuldades enfrentadas pela secretaria e pela comissão de bolsas. Entre elas, destaca-se a carga de trabalho manual, a desorganização dos prazos e responsabilidades, a fragmentação da informação entre diferentes planilhas e documentos, e a ausência de notificações automatizadas para eventos importantes, como a entrega de relatórios semestrais e o encerramento de bolsas.

Não existe atualmente um sistema que concentre em um único local dados fundamentais como: histórico de bolsistas, períodos de recebimento, status de desligamento ou formatura, obrigações cumpridas, e registros de comunicações com as agências de fomento. Essa dispersão de dados resulta em riscos operacionais relevantes, como atrasos nos processos, falhas de comunicação, perda de bolsas e duplicidade de concessão a um mesmo aluno.

A implementação de um sistema unificado e automatizado é, portanto, essencial para garantir maior eficiência, rastreabilidade e segurança à gestão de bolsas. A proposta prevê a integração do novo módulo ao sistema *Marvin*, aproveitando a estrutura já existente e promovendo maior aderência às práticas de desenvolvimento em uso pela UFES.

O escopo inicial do módulo busca centralizar os processos de seleção, acompanhamento e desligamento de bolsistas, permitindo à comissão e à secretaria monitorar e tomar decisões com base em dados consolidados. Além disso, o sistema reutiliza a infraestrutura de e-mail do Marvin para suportar notificações essenciais do ciclo do vínculo, mantendo espaço para evolução após homologação.

Embora a ideia de expansão do sistema para funcionalidades complementares, como a gestão de egressos e integração com redes sociais (ex: LinkedIn), tenha surgido durante as discussões, essas funcionalidades foram consideradas fora do escopo imediato do projeto e poderão ser desenvolvidas em trabalhos futuros. Por ora, o foco é garantir uma solução robusta e funcional para a gestão centralizada de bolsas, suprindo lacunas críticas do processo atual.

1.2 Objetivos

O objetivo geral deste trabalho é **projetar e implementar** o *Sistema de Gestão de Bolsas* como um subsistema integrado ao *Marvin*, voltado à centralização e ao suporte ao ciclo de vida das bolsas no contexto do PPGI/UFES — desde a publicação de editais e a gestão de inscrições e classificação, até a formalização do vínculo de bolsistas, acompanhamento de vigência, comunicação e registro histórico de encerramentos e egressos. Esse objetivo busca mitigar problemas do processo atual (uso intensivo de planilhas e comunicações dispersas), promovendo maior rastreabilidade, padronização e eficiência administrativa.

A seguir, são descritos os objetivos específicos que, de forma conjunta, contribuem para a consecução do objetivo geral:

- **Levantar, analisar e documentar os requisitos do módulo de bolsas do PPGI:** compreender os atores envolvidos (secretaria, comissão, coordenação, orientadores e alunos), as histórias de usuário e regras de negócio, bem como as restrições e requisitos não funcionais que delimitam o escopo do subsistema e orientam seu comportamento esperado referente à gestão de bolsas;
- **Projetar e documentar a arquitetura do módulo, de forma integrada ao *Marvin*:** definir a organização do subsistema em camadas e sua alocação em pacotes/partições, descrevendo pontos de integração com o núcleo do Marvin (por exemplo, autenticação/autorização, cadastro de usuários e infraestrutura de serviços), de modo a preservar consistência arquitetural e facilitar evolução e manutenção;
- **Implementar e testar o código-fonte do módulo:** materializar as funcionalidades priorizadas do subsistema conforme o escopo definido, com ênfase em:

- **gestão de editais e candidaturas**, incluindo parametrização e ranqueamento automático de candidatos conforme critérios do edital e da agência;
- **mecanismos de envio automático de notificações por e-mail** para bolsistas, orientadores e secretaria, suportando eventos do ciclo de vida do vínculo (por exemplo, aceite e encerramento);
- **registro do histórico completo dos bolsistas**, contemplando informações de início, término e encerramento do vínculo, garantindo rastreabilidade institucional;
- **detecção e alerta de conflitos** relacionados a editais simultâneos e a múltiplas concessões de bolsa para um mesmo aluno, conforme diretrizes institucionais e das agências de fomento consideradas no trabalho.

1.3 Método de Desenvolvimento do Trabalho

O desenvolvimento do sistema de gestão de bolsas proposto neste trabalho seguirá uma abordagem incremental e iterativa, estruturada em etapas que abrangem desde a fundamentação teórica até a implementação prática e documentação do sistema. As atividades principais estão descritas a seguir:

1. **Revisão bibliográfica:** será realizada uma pesquisa sobre boas práticas de Engenharia de Software e Engenharia de Requisitos, incluindo tópicos como modelagem de requisitos, princípios SOLID, padrões arquiteturais e metodologias de desenvolvimento de sistemas Web, além de estudos sobre bancos de dados relacionais e comunicação entre sistemas.
2. **Documentação do sistema:** serão produzidos dois documentos principais:
   - O Documento de Especificação de Requisitos, que conterá uma visão geral do sistema, definição do ambiente, levantamento dos stakeholders, estórias de usuário, regras de negócio e casos de uso;
   - O Documento de Projeto de Sistemas, no qual será descrita a arquitetura do módulo, detalhando os componentes do software, classes, atributos e comportamento esperado dos casos de uso modelados.
3. **Estudo de tecnologias:** serão analisadas e aplicadas tecnologias compatíveis com o ecossistema do sistema *Marvin*, incluindo a linguagem de programação Java, a plataforma Jakarta EE, o ambiente de desenvolvimento IntelliJ IDEA, o banco de dados PostgreSQL, a ferramenta DBeaver, o framework PrimeFaces para desenvolvimento de interfaces Web e o JUnit para testes unitários.

4. **Implementação:** será realizada a codificação dos componentes do sistema, seguindo os requisitos documentados e os padrões técnicos definidos pelo projeto *Marvin*. A implementação utilizará princípios de programação orientada a objetos e práticas recomendadas, com foco em modularidade, manutenibilidade e reutilização de componentes.

5. **Integração ao *Marvin*:** a arquitetura do módulo será projetada de modo a respeitar a padronização visual, estrutural e organizacional do sistema *Marvin*, garantindo integração fluida com funcionalidades existentes e compatibilidade com os módulos já em produção.

6. **Validação e testes:** será realizada uma bateria de testes unitários e testes manuais nos fluxos críticos do sistema, com simulação de interações dos diferentes perfis de usuários (secretaria, comissão de bolsas e coordenação), a fim de garantir o correto funcionamento do sistema e a conformidade com os requisitos levantados.

1.4 Organização da Monografia

Além desta introdução, esta monografia está organizada em cinco capítulos:

- O Capítulo 2 apresenta o referencial teórico e as tecnologias utilizadas, abordando conceitos de engenharia de software e desenvolvimento Web, bem como a base técnica adotada no sistema (por exemplo, Jakarta EE, JSF/PrimeFaces, EJB, JPA, servidor de aplicação e banco de dados);

- O Capítulo 3 apresenta a análise e especificação de requisitos do módulo de gestão de bolsas, incluindo escopo, stakeholders, requisitos funcionais e não funcionais, regras de negócio, estórias de usuário com critérios de aceitação, modelos de caso de uso e o modelo conceitual;

- O Capítulo 4 descreve o projeto e a implementação da solução, detalhando a arquitetura de software, a modelagem dos principais componentes e entidades, e a apresentação das telas e fluxos implementados no protótipo;

- O Capítulo 5 apresenta a estratégia de avaliação e validação da proposta, descrevendo o procedimento adotado, os cenários analisados e os resultados obtidos a partir do feedback do cliente;

- O Capítulo 6 apresenta as considerações finais, retomando os objetivos do trabalho, sintetizando as contribuições alcançadas e discutindo limitações e possibilidades de trabalhos futuros.

2 Referencial Teórico e Tecnologias Utilizadas

Para embasar o desenvolvimento do módulo de gestão de bolsas, foi conduzido um levantamento bibliográfico abrangente em tópicos fundamentais da Engenharia de Software, desenvolvimento de sistemas Web, arquitetura de software e boas práticas de modelagem e implementação de sistemas distribuídos.

A fundamentação teórica tem como objetivo fornecer os conceitos e técnicas necessários para garantir a qualidade do sistema proposto, assegurando que ele seja funcional, escalável e manutenível. Dentre os principais tópicos estudados, destacam-se: Engenharia de Software (Seção 2.1), desenvolvimento Web (Seção 2.2), o sistema Marvin (Seção 2.3), *frameworks* (Seção 2.4), a arquitetura em três camadas (Seção 2.5) e centralização de sistemas (Seção 2.6).

2.1 Engenharia de Software

A Engenharia de Software é um campo que engloba um conjunto de métodos, técnicas e ferramentas voltados para o desenvolvimento de sistemas complexos de alta qualidade e dentro dos prazos estabelecidos (Pressman; Maxim, 2016). Segundo Mall (2020), uma definição popular do termo é: “uma coleção sistemática de boas práticas e técnicas de desenvolvimento de programas”. Essa definição destaca o papel essencial da sistematização no processo de construção de software de qualidade, ressaltando que essas boas práticas derivam tanto de inovações teóricas quanto da experiência prática acumulada por programadores ao longo dos anos.

Ainda segundo Mall (2020), outra definição alternativa pode ser: “uma abordagem de engenharia para desenvolver software”. Essa concepção reforça a ideia de que o desenvolvimento de software deve ser conduzido de forma metódica e previsível, com ênfase em planejamento, controle de qualidade e gestão de riscos — princípios típicos da engenharia tradicional.

Mall (2020) propõe uma analogia para explicar o que se entende por “abordagem de engenharia”: comparar a atuação de um pedreiro autônomo com a de um engenheiro civil. Enquanto o primeiro executa tarefas sem um projeto sistemático, o segundo adota métodos planejados e baseados em normas técnicas para garantir a segurança, funcionalidade e durabilidade da obra. Esse mesmo raciocínio aplica-se ao desenvolvimento de software: soluções improvisadas podem funcionar em curto prazo, mas carecem da robustez, escalabilidade e manutenibilidade exigidas em sistemas críticos.

O desenvolvimento de software compreende um conjunto de atividades que têm

como objetivo principal criar e entregar um produto de software ao cliente, juntamente com sua documentação. Esse processo geralmente envolve etapas como Análise e Especificação de Requisitos, Projeto, Implementação, Testes, Entrega e Implantação do Sistema (Barcellos, 2018).

Em suma, a Engenharia de Software busca transformar o desenvolvimento de programas — anteriormente informal e artesanal — em uma prática controlada, previsível e alinhada com padrões de qualidade. Essa transformação é essencial especialmente em contextos institucionais, como o desenvolvimento de sistemas acadêmicos e administrativos, nos quais a confiabilidade e a rastreabilidade são requisitos críticos.

2.1.1 Engenharia de Requisitos

Em um desenvolvimento de software, a primeira coisa a ser feita é capturar os requisitos que o sistema a ser desenvolvido tem de tratar. Um entendimento dos requisitos do software é essencial para o sucesso de um projeto de desenvolvimento de software. Os requisitos devem ser inicialmente levantados e descritos de maneira sucinta para permitir definir o escopo do sistema. Depois, os requisitos devem ser refinados em detalhes, as funções e o desempenho do software devem ser especificados e as interfaces e restrições que o software deve atender devem ser estabelecidas (Sommerville, 2007).

Uma das principais medidas do sucesso de um sistema de software é o grau no qual ele atende aos requisitos para os quais foi construído. Existem diversas definições para requisito de software na literatura, dentre elas:

- Requisitos são descrições dos serviços que devem ser providos pelo sistema e de suas restrições operacionais (Sommerville, 2007);
- Um requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos (Pfleeger, 2004);
- Um requisito é alguma coisa que o produto tem de fazer ou uma qualidade que ele precisa apresentar (Robertson; Robertson, 2006).

É importante ressaltar que os requisitos de software podem assumir diferentes níveis de abstração como destacado por Sommerville (2007), dependendo da etapa do processo de desenvolvimento em que são tratados. Durante as fases iniciais, os requisitos tendem a ser mais gerais e voltados à compreensão do problema. À medida que o projeto avança, esses requisitos são refinados e transformados em especificações técnicas mais detalhadas, que servirão de base para o projeto e implementação do sistema.

Um resultado comum dessa análise de requisitos é a listagem dos requisitos funcionais e não funcionais do sistema. Barcellos (2018) descreve esses conceitos da seguinte

forma:

- **Requisitos funcionais:** são declarações de serviços que o sistema deve fornecer, descrevendo o que o sistema deve fazer. Os requisitos funcionais descrevem as interações entre o sistema e seu ambiente, podendo especificar como o sistema deve reagir a entradas específicas, como deve se comportar em situações específicas e o que não deve fazer;
- **Requisitos não funcionais:** descrevem restrições sobre os serviços ou funções oferecidos pelo sistema, limitando as opções para criar uma solução para o problema. Esses requisitos não funcionais são de grande importância para a fase de projeto, servindo como base para tomada de decisões nessa etapa.

2.1.2 Projeto de Software

O projeto de software ocupa um papel central no processo de desenvolvimento, como proposto por Pressman & Maxim (2016), sendo uma das etapas mais críticas para o sucesso de um sistema. Independentemente do modelo de ciclo de vida adotado ou do paradigma utilizado, o projeto inicia-se a partir do momento em que os requisitos do sistema estão, ao menos parcialmente, definidos e documentados. Trata-se da primeira das três atividades principais do domínio da solução computacional: projeto, implementação e testes.

Enquanto a análise se concentra na compreensão do problema de forma abstrata e desvinculada da solução tecnológica, a etapa de projeto é responsável por transformar essa compreensão em uma proposta concreta de implementação. Pressman & Maxim (2016) propõem que essa transformação é feita por meio da modelagem do sistema, levando em consideração os requisitos funcionais e não funcionais, e aplicando restrições tecnológicas e decisões arquiteturais.

De acordo com Barcellos (2018), na fase de projeto, modelos de projeto são gerados a partir dos modelos de análise, com o objetivo de representar o que deverá ser codificado na fase de implementação.

Independentemente do paradigma adotado, o projeto de software deve contemplar os seguintes aspectos, segundo Barcellos (2018):

- **Projeto da Arquitetura do Software:** responsável por definir os principais componentes estruturais do sistema e suas interações;
- **Projeto de Dados:** foca na definição da estrutura de armazenamento de dados necessária para sustentar as funcionalidades do software;

- **Projeto de Interfaces:** define como os componentes do sistema interagem entre si (interfaces internas), com outros sistemas (interfaces externas) e com os usuários (interface homem-máquina);
- **Projeto Detalhado:** tem como propósito refinar os componentes definidos na arquitetura, especificando comportamentos, atributos e operações de cada elemento do sistema.

Por fim, é essencial considerar desde o início as tecnologias disponíveis e os ambientes de desenvolvimento e produção em que o sistema será implantado. Segundo Falbo (2018), as decisões de projeto devem ser tomadas progressivamente, partindo de níveis mais abstratos — que se aproximam das definições da análise — até alcançar níveis mais concretos, próximos da implementação real do sistema.

2.1.3 Testes de Software

Uma vez projetado o sistema, é necessário escrever os programas que implementem esse projeto e testá-los. Os testes devem ser feitos em diversos níveis, começando pelos módulos isolados (teste de unidade), passando a integrá-los (teste de integração), até atingir os testes do sistema como um todo (teste de sistema). Diversas técnicas podem ser empregadas para este fim (Barcellos, 2018).

Os testes de software constituem uma etapa fundamental no processo de desenvolvimento, tendo como principal objetivo verificar se o sistema implementado atende aos requisitos previamente definidos. Além disso, como dizem Delamaro, Maldonado & Jino (2007), eles são essenciais para identificar falhas e garantir a qualidade do produto final antes que este seja disponibilizado ao usuário final.

Existem diferentes níveis de teste (Pressman; Maxim, 2016), cada um com foco em aspectos específicos da aplicação. Os testes unitários verificam o comportamento de pequenos componentes do sistema, como métodos ou funções. Já os testes de integração avaliam como os diferentes módulos interagem entre si. Os testes de sistema analisam o comportamento do software como um todo, enquanto os testes de aceitação buscam confirmar se o sistema atende às necessidades dos usuários e *stakeholders*.

Além disso, os testes podem ser manuais ou automatizados. Os testes automatizados são especialmente úteis em ambientes de desenvolvimento ágil, pois permitem a reexecução frequente de testes regressivos de forma rápida e confiável, promovendo maior eficiência e redução de custos a longo prazo (Crispin; Gregory, 2009).

Portanto, os testes são indispensáveis para garantir a entrega de um software robusto, funcional e em conformidade com os padrões de qualidade esperados. Sua aplicação cuidadosa é um dos pilares para o sucesso de qualquer projeto de software.

Testar um software não se resume apenas à execução de códigos para verificar se não há erros. Trata-se de um processo sistemático, como definido por Myers, Sandler & Badgett (2011), que envolve planejamento, definição de casos de teste, análise de cobertura, execução controlada e verificação de resultados. O objetivo central é aumentar a confiabilidade do sistema e reduzir os riscos associados ao seu uso.

Em relação às técnicas de teste funcional de maneira geral, vale ressaltar que, como os critérios funcionais se baseiam apenas na especificação, não podem assegurar que partes críticas e essenciais do código tenham sido cobertas. Assim, é importante que outras técnicas sejam aplicadas em conjunto, para que o software seja explorado por diferentes pontos de vista (Delamaro; Maldonado; Jino, 2007). Os testes estruturais são uma boa opção para este fim.

2.2 Desenvolvimento Web

A Internet é uma infraestrutura de informação generalizada, considerada o protótipo inicial da Infraestrutura Internacional de Informação. Sua história é complexa, abrangendo diversos aspectos tecnológicos, organizacionais e comunitários. Seu impacto se estende para além das áreas técnicas das comunicações de computador, alcançando toda a sociedade à medida que avançamos rumo ao aumento do uso de ferramentas online para realizar atividades como comércio eletrônico, busca de informações e operações comunitárias (Badalotti, 2018).

Em vista disso, o desenvolvimento Web, como definido por Badalotti (2018) é uma área da computação voltada para a criação, manutenção e evolução de aplicações e sites que funcionam através da Internet. Com o crescimento exponencial do acesso à Web nas últimas décadas, a construção de sistemas acessíveis por navegadores se tornou uma competência fundamental no mercado de tecnologia da informação.

Essa área envolve uma combinação de tecnologias, linguagens de programação e boas práticas que permitem a criação de aplicações robustas, interativas e escaláveis. De forma geral, o desenvolvimento Web pode ser dividido em três camadas principais: Interface com o Usuário, responsável pela interação e comunicação entre o sistema e as pessoas que o utilizam; Lógica de Negócio, que trata das regras de negócio e manipulação de dados; e os bancos de dados, que armazenam de forma persistente as informações manipuladas pelo sistema (Pressman; Lowe, 2008). Além do conhecimento técnico, o desenvolvimento Web exige atenção à usabilidade, acessibilidade, segurança e desempenho das aplicações. Um bom desenvolvedor Web precisa garantir que o sistema seja funcional e intuitivo, mesmo em diferentes navegadores, dispositivos e condições de rede.

Como apontam Conte, Mendes & Travassos (2005), processos de desenvolvimento para aplicações de software Web devem produzir representações para o projeto de aspectos

de aplicações tradicionais, como estrutura e funcionalidades, e também para aspectos orientados à Web, como navegação e apresentação.

No contexto de sistemas acadêmicos e administrativos, como os utilizados em universidades, o desenvolvimento Web permite a criação de plataformas acessíveis por alunos, professores e servidores, promovendo a digitalização de processos antes realizados manualmente. Isso resulta em maior eficiência, rastreabilidade e transparência no gerenciamento de informações institucionais.

Portanto, o desenvolvimento Web é uma área dinâmica, essencial e em constante evolução, que exige atualização contínua, domínio técnico e capacidade de adaptação para lidar com novos desafios e tecnologias emergentes.

2.3 O sistema Marvin

O *Marvin* é um sistema Web desenvolvido e mantido por discentes e docentes da UFES, no contexto de um projeto de extensão, com o propósito de oferecer funcionalidades de apoio às atividades de ensino, pesquisa e tarefas administrativas que não são cobertas pelos sistemas institucionais existentes (Souza, 2022).

Ele é construído de forma **modular**: diferentes módulos são desenvolvidos ao longo do tempo (frequentemente como projetos de graduação), e posteriormente integrados para compor um sistema único e com interface unificada (Souza, 2022). .

Essa característica torna o Marvin adequado para a evolução incremental de funcionalidades, permitindo que novos subsistemas sejam incorporados reaproveitando infraestrutura já consolidada (Souza, 2022).

No Marvin, existe um **módulo núcleo** (*Marvin: núcleo*) que concentra funcionalidades fundamentais, como autenticação e cadastros, sobre as quais os demais módulos são construídos e integrados (Souza, 2022).

Do ponto de vista arquitetural, o Marvin adota uma combinação dos estilos **Camadas** e **Partições**: os subsistemas (incluindo o núcleo) formam partições, e cada partição é subdividida em camadas de Interface com o Usuário (CIU), Lógica de Negócio (CLN) e Gerência de Dados (CGD) (Souza, 2022).

Na CIU, a aplicação Web é organizada segundo o padrão MVC, separando elementos de `view` e `controller`; na CLN, o padrão *Service Layer* organiza responsabilidades em `domain` e `application`; e a CGD concentra a persistência no pacote `persistence`, seguindo o padrão DAO (Souza, 2022).

Assim, ao desenvolver um novo módulo (como o módulo de Gestão de Bolsas), a solução pode reutilizar componentes do núcleo (por exemplo, autenticação/autorização e serviços de infraestrutura), mantendo consistência arquitetural e reduzindo duplicação de

mecanismos transversais já disponíveis na plataforma (Souza, 2022).

2.4 Frameworks

A utilização de *frameworks* no desenvolvimento de sistemas é essencial em muitos contextos, especialmente quando se deseja entregar soluções complexas dentro de prazos apertados e com alto nível de confiabilidade. Um *framework* pode ser entendido como um conjunto de ferramentas, bibliotecas, componentes e regras que fornecem uma estrutura padrão para o desenvolvimento de software. Como definido em (Fayad; Schmidt; Johnson, 1999), ele atua como uma base reutilizável que permite aos desenvolvedores construir aplicações de forma mais rápida, padronizada e com menor propensão a erros.

No contexto da Web, essas ferramentas de desenvolvimento de software, também conhecidas como *frameworks* para aplicações Web ou Web *frameworks*, permitem que os desenvolvedores criem aplicações Web de forma mais rápida e simples, fornecendo um modelo básico, bem como um conjunto de APIs, bibliotecas e extensões relevantes (Swacha; Kulpa, 2023).

No desenvolvimento do sistema de gestão de bolsas proposto, que visa unificar e automatizar processos atualmente conduzidos de forma manual por meio de planilhas e comunicações descentralizadas, é essencial adotar um conjunto de tecnologias robusto e consolidado para aplicações corporativas. Nesse sentido, foi utilizado o ecossistema Jakarta Enterprise Edition (Jakarta EE), por oferecer especificações e bibliotecas amplamente empregadas no desenvolvimento de sistemas empresariais, como injeção de dependência, persistência de dados, transações e componentes de apresentação. Além disso, o módulo foi desenvolvido e integrado à plataforma *Marvin*, um sistema modular já adotado e consolidado na Universidade Federal do Espírito Santo para gestão acadêmica e administrativa. A escolha por Jakarta EE já se encontra alinhada ao contexto tecnológico do próprio Marvin, o que favorece a reutilização de padrões arquiteturais e de infraestrutura existentes, bem como a integração coesa das funcionalidades do módulo e sua evolução contínua.

Conforme destacado por Sommerville (2007), a adoção de *frameworks* consolidados não apenas acelera o desenvolvimento, mas também promove a consistência arquitetural, reduz riscos e melhora a manutenibilidade do sistema ao longo do tempo, especialmente em projetos de larga escala com múltiplos requisitos e integrações.

A **Jakarta EE**[1] fornece um conjunto abrangente de especificações e APIs que padronizam o desenvolvimento de aplicações Java distribuídas, o que é ideal para um sistema que exigirá funcionalidades como notificações automatizadas, controle de prazos, geração de documentos e gerenciamento de usuários com diferentes papéis (secretaria, comissão de bolsas e coordenação).

[1] <https://jakarta.ee>

Entre as principais especificações utilizadas da plataforma, destacam-se:

- **Jakarta Servlets**: componentes Java executados no servidor que permitem gerar conteúdo dinâmico (como HTML ou JSON) de forma eficiente;
- **Jakarta Faces (JSF)**: *framework* voltado ao desenvolvimento de interfaces reutilizáveis e componentizadas para aplicações Web;
- **Jakarta Persistence (JPA)**: API para mapeamento objeto-relacional (ORM), essencial para persistência estruturada de dados como informações de bolsas, relatórios e usuários;
- **Jakarta Enterprise Beans (EJB)**: facilitam o desenvolvimento de componentes corporativos, seguros e escaláveis;
- **Contexts and Dependency Injection (CDI)**: provê um modelo flexível para injeção de dependências e controle de ciclo de vida de componentes da aplicação.

A escolha dessas tecnologias alinha-se com a proposta do projeto de desenvolver uma ferramenta confiável e escalável, com potencial de expansão futura para controle de egressos, relatórios automatizados e integração com sistemas existentes como o Marvin e, eventualmente, com outros sistemas utilizados pela universidade. Além disso, o uso da Jakarta EE oferece maturidade e estabilidade, importantes para sistemas que lidam com dados sensíveis e com múltiplas regras de negócio simultâneas.

De forma geral, aplicações Web modernas costumam ser estruturadas de modo a separar responsabilidades entre apresentação, regras de negócio e persistência. Essa separação aparece com frequência na forma de uma arquitetura em camadas (por exemplo, a arquitetura em três camadas), pois se ajusta bem ao modelo cliente–servidor típico da Web e favorece manutenção e evolução do sistema (Pressman; Lowe, 2008; Shklar; Rosen, 2009). Além disso, decisões arquiteturais como separação em camadas e uso de padrões como MVC são recorrentes em sistemas Web e corporativos, pois contribuem para modularização, testabilidade e evolução do código, especialmente em aplicações com múltiplos perfis de acesso e regras de negócio. A seguir, discutimos os principais aspectos da arquitetura em três camadas.

2.5 Arquitetura em Três Camadas

A arquitetura em três camadas, também conhecida como *three-tier architecture*, é um modelo de organização de sistemas de software que promove a separação de responsabilidades e a modularização do código. Essa arquitetura divide a aplicação em três partes distintas: a camada de apresentação, a camada de lógica de negócios e a camada de dados.

Cada uma dessas camadas tem um papel bem definido, o que facilita a manutenção, a escalabilidade e a reutilização do sistema (Shklar; Rosen, 2009).

A Arquitetura em Três Camadas é um padrão arquitetural amplamente utilizado. Segundo Valente (2020), essas camadas são divididas da seguinte forma, ilustrado na Figura 1:

- **Interface com o Usuário ou Camada de Apresentação**: lida tanto com a exibição de informações quanto com a coleta e processamento de entradas e eventos de interface, como cliques em botões, marcação de texto, etc.;
- **Lógica de Negócio ou Camada de Aplicação**: implementa as regras de negócio do sistema;
- **Banco de Dados**: armazena os dados manipulados pelo sistema.

Figura 1 – Arquitetura em três camadas.

Fonte: Araújo (2023).

No contexto do sistema de gestão de bolsas, a separação em camadas é essencial para atender às necessidades complexas identificadas durante as reuniões com os *stakeholders*. A camada de apresentação será responsável pela interface com a comissão de bolsas, secretarias e bolsistas. Já a camada de aplicação irá concentrar as regras do sistema, como os critérios de elegibilidade, o gerenciamento de prazos de relatórios, o controle de desligamentos e notificações automáticas. Por fim, a camada de dados manterá o histórico completo de concessões de bolsas, eventos associados, e registros de interações com agências de fomento.

Normalmente, uma arquitetura em três camadas é distribuída (Shklar; Rosen, 2009). Isso significa que a camada de interface é executada nas máquinas dos clientes, a camada

de negócio é executada em um servidor, frequentemente chamado de servidor de aplicação, e temos o banco de dados.

Um padrão de projeto intimamente relacionado a essa arquitetura, segundo Valente (2020), é o *MVC (Model-View-Controller)*. O padrão MVC também busca separar preocupações, organizando a aplicação em três componentes principais: o **Model** (modelo), que representa os dados e a lógica de negócios; a **View** (visão), que trata da apresentação da interface ao usuário; e o **Controller** (controlador), que atua como intermediário entre o modelo e a visão, lidando com as entradas do usuário e coordenando a resposta apropriada.

Embora MVC e a arquitetura em três camadas não sejam a mesma coisa, ambos compartilham o objetivo de modularização e organização clara do sistema. O MVC costuma ser adotado principalmente na camada de apresentação, ajudando a organizar a interface de forma mais limpa e reativa, enquanto a arquitetura em três camadas se refere à estrutura geral da aplicação. A combinação dos dois modelos resulta em sistemas mais robustos, organizados e de fácil manutenção (Valente, 2020).

Ao separar essas responsabilidades, a arquitetura em três camadas favorece o desenvolvimento em equipe, pois permite que diferentes grupos trabalhem simultaneamente em camadas distintas. Além disso, torna o sistema mais flexível a mudanças, uma vez que modificações em uma camada (como a troca do banco de dados ou a reformulação da interface) não necessariamente impactam as demais (Fowler, 2002).

Esse modelo é amplamente utilizado em aplicações corporativas e sistemas distribuídos por sua robustez, clareza estrutural e facilidade de evolução ao longo do tempo.

2.6 Centralização de Sistemas

A centralização de sistemas é uma estratégia amplamente adotada no contexto de engenharia de software e gestão de processos organizacionais, pois favorece a integração e o controle unificado das operações. De acordo com Sommerville (2007), a integração de processos e informações em um único sistema reduz a redundância e melhora a qualidade dos dados, tornando as operações mais consistentes e confiáveis.

Trata-se da consolidação de múltiplas operações, fluxos de trabalho, dados e ferramentas em uma única plataforma integrada, o que, segundo Pressman & Maxim (2016), proporciona maior eficiência e coerência arquitetural, além de permitir que a equipe de desenvolvimento trabalhe com uma base comum e bem definida.

Entre os principais benefícios da centralização, destaca-se a redução da redundância de dados. Quando os processos são distribuídos entre planilhas, e-mails e ferramentas isoladas, é comum que a mesma informação seja registrada múltiplas vezes, de maneira inconsistente. Conforme Turban & Volonino (2015), a existência de uma única fonte

de verdade elimina duplicidades, reduz erros e melhora a confiabilidade da informação, garantindo que todos os setores da organização utilizem dados consistentes.

Outro benefício essencial é a facilidade de manutenção e evolução do sistema. Com todos os módulos integrados, a aplicação de atualizações, correções e melhorias torna-se mais eficiente, reduzindo o custo operacional e o tempo de resposta às demandas dos usuários (Turban; Volonino, 2015). Além disso, a centralização promove maior visibilidade gerencial, permitindo que os responsáveis tenham uma visão abrangente e em tempo real dos processos, o que facilita a tomada de decisões estratégicas.

Do ponto de vista da segurança, sistemas centralizados tendem a ser mais robustos, pois permitem o controle de acesso e auditoria unificada. Isso significa que é possível rastrear todas as ações realizadas por usuários, definir níveis de permissão específicos e proteger informações sensíveis com mais eficácia. Esse tipo de visão é apoiado por norma internacional ISO/IEC (2022b).

No caso da UFES, especificamente no contexto do gerenciamento de bolsas de programas de pós-graduação, os desafios decorrentes da descentralização são evidentes. Atualmente, o controle das bolsas é feito com o apoio de planilhas manuais, arquivos dispersos em drives e comunicação fragmentada entre secretaria, comissão de bolsas e agências de fomento.

Essa estrutura tem se mostrado frágil, propensa a erros e ineficiente, especialmente quando se trata de acompanhar o histórico de bolsistas, prazos de envio de relatórios, desligamentos e renovações. Além disso, não há uma rastreabilidade clara sobre quais alunos receberam bolsas, por quanto tempo, e se as exigências de cada agência de fomento foram devidamente cumpridas.

A proposta de centralização desses processos no sistema Marvin visa justamente resolver essas limitações. Por meio da criação de um módulo específico de gestão de bolsas, será possível concentrar todas as etapas — desde a inscrição em editais até o encerramento da bolsa — em uma única plataforma. Isso inclui funcionalidades como envio automático de notificações, geração de relatórios personalizados, registro de histórico de bolsas, acompanhamento de prazos e integração com os dados acadêmicos dos alunos.

Com isso, o sistema oferece não apenas uma solução tecnológica para problemas operacionais, mas também um instrumento de governança institucional, promovendo transparência, rastreabilidade e eficiência na gestão das bolsas. A centralização no Marvin, portanto, representa um passo estratégico para modernizar e profissionalizar os processos acadêmico-administrativos do programa.

3 Análise e Especificação de Requisitos

Este capítulo apresenta a análise e a especificação de requisitos do *Sistema de Gestão de Bolsas do PPGI* integrado ao Marvin. São descritos o escopo da solução, os principais stakeholders e perfis de usuário, os requisitos funcionais e não funcionais, as regras de negócio mais relevantes, histórias de usuário com critérios de aceitação, bem como o modelo de casos de uso, a identificação de subsistemas e interdependências e o modelo conceitual em alto nível.

3.1 Descrição do escopo

O módulo de Gestão de Bolsas tem por objetivo apoiar o ciclo de vida completo de bolsas acadêmicas do Programa de Pós-Graduação em Informática (PPGI) da UFES, desde a criação e manutenção de editais até a inscrição, classificação, concessão e acompanhamento de bolsistas.

Escopo incluído

Estão incluídas no escopo deste trabalho as seguintes capacidades:

- **Gestão de editais de bolsa**: abertura e encerramento de editais (*ScholarshipNotice*) com informações como título, período de inscrição, agência de fomento e status;
- **Inscrição de candidatos**: formulário web de inscrição (*ScholarshipApplication*) para alunos do PPGI, com seleção de edital aberto, preenchimento de dados pessoais e acadêmicos, nível de formação, faixa de renda, categoria de cota, coeficiente de rendimento e justificativa;
- **Classificação de candidatos**: processo de classificação parametrizável por edital, com cálculo de pontuação, ordenação de candidatos, aplicação de cotas por nível e categoria e geração de lista de *classificados*, *lista de espera* e *indeferidos*;
- **Concessão e gerenciamento de bolsas**: criação de registros de bolsistas (*ScholarshipStudent*) a partir das classificações, associação a orientadores, definição de datas de início e término previsto, acompanhamento de status da bolsa (ativa, encerrada, etc.);
- **Encerramento de bolsas**: registro da data de término efetivo e do motivo de encerramento, com atualização de status e rastreabilidade;
- **Listagens e consultas**: visualização consolidada de editais, inscrições e bolsistas,

com contagem de inscrições e bolsistas por edital e filtros básicos por status, nível e outros atributos;

- **Notificações**: envio de e-mail de aceite de bolsa para alunos selecionados, utilizando a infraestrutura de notificação já presente no Marvin.

Integração com sistemas UFES

A solução foi projetada para integrar-se ao Marvin e, indiretamente, aos serviços de autenticação da universidade. Integrações adicionais com sistemas institucionais (por exemplo, importação automática de dados de matrícula ou coeficiente de rendimento) são consideradas trabalhos futuros, mas não foram implementadas neste TCC.

3.2 Stakeholders e perfis de usuário

Os principais stakeholders do sistema e seus perfis de uso são:

- **Secretaria do PPGI**: responsável por cadastrar e atualizar editais, registrar resultados, gerar relatórios para colegiado e pró-reitoria e lançar informações complementares sobre bolsistas;
- **Comissão de bolsas**: grupo de docentes que define critérios específicos de classificação, valida resultados, interpreta rankings e acompanha a alocação de vagas e cotas;
- **Coordenação do PPGI**: toma decisões sobre abertura e encerramento de editais, homologação de resultados, resolução de conflitos entre candidatos e comunicação oficial com a pró-reitoria;
- **Bolsista**: aluno com bolsa ativa, que acompanha sua situação no sistema, confirma dados cadastrais e cumpre prazos de entrega de relatórios (quando integrados ao módulo);
- **Equipe de TI / Administração do Marvin**: responsável por manter o sistema em produção, realizar configurações técnicas (banco de dados, servidor de aplicação), monitorar logs e apoiar evoluções futuras do módulo.

3.3 Requisitos funcionais

Os requisitos funcionais a seguir foram identificados a partir da análise do contexto do PPGI, das reuniões com a coordenação e da documentação de processos internos de bolsas. Para efeito de rastreabilidade, são identificados pelo prefixo **RF**.

Para apoiar a organização da especificação e as etapas de projeto e implementação, os requisitos foram rotulados por tipo: requisitos funcionais são identificados por **RF**, requisitos não funcionais por **RNF** e regras de negócio por **RN**. Ao longo do documento, esses identificadores permitem referenciar diretamente a origem de uma funcionalidade (RF), um atributo de qualidade (RNF) ou uma restrição do domínio (RN) associada ao comportamento esperado do módulo.

RF01 O sistema deve permitir o cadastro, edição e exclusão manual, por meio do código, de editais de bolsa, com campos como título, descrição, agência de fomento, período de inscrição e status.

RF02 O sistema deve permitir abrir e encerrar editais, alterando o status entre `ABERTO` e `ENCERRADO`, respeitando regras de data quando configuradas.

RF03 O sistema deve listar todos os editais cadastrados em uma visão consolidada, exibindo título, período, status, número de inscrições e número de bolsistas associados.

RF04 O sistema deve disponibilizar um formulário de inscrição em edital de bolsa, permitindo ao aluno selecionar um edital aberto e informar seus dados pessoais, acadêmicos, nível, faixa de renda, categoria de cota, coeficiente e justificativa.

RF05 O sistema deve validar, no momento da inscrição, se o edital selecionado está aberto e dentro do período de inscrição.

RF06 O sistema deve registrar a inscrição com status inicial `INSCRITO` e data de submissão.

RF07 O sistema deve impedir que um mesmo aluno realize múltiplas inscrições ativas para o mesmo edital utilizando o mesmo número de matrícula.

RF08 O sistema deve permitir configurar, para cada edital, o número de vagas para cada combinação de nível de formação e categoria de cota (por exemplo, Graduação / Ampla Concorrência, Mestrado / Cotas).

RF09 O sistema deve executar o processo de classificação de inscritos em um edital, calculando pontuação e posição de cada candidato, com base em regras específicas por agência ou modalidade.

RF10 O sistema deve aplicar o limite de vagas por combinação *nível–categoria*, marcando candidatos dentro do limite como `CLASSIFICADO` e excedentes como `LISTA_ESPERA`.

RF11 O sistema deve permitir marcar, em lote, todos os candidatos `CLASSIFICADO` e `LISTA_ESPERA` de um edital como `INDEFERIDO`, quando a coordenação necessitar invalidar um resultado anterior.

RF12 O sistema deve permitir criar registros de bolsistas a partir de candidaturas classificadas, copiando dados essenciais da inscrição e associando o vínculo ao edital de origem.

RF13 O sistema deve selecionar, de forma automática, um orientador disponível para o bolsista criado, a partir da lista de orientadores cadastrados.

RF14 O sistema deve permitir consultar e filtrar bolsistas por edital, status, orientador, nome ou matrícula.

RF15 O sistema deve permitir editar informações de vínculo de bolsa, incluindo status, orientador, data prevista de término e data de término efetivo.

RF16 O sistema deve permitir encerrar uma bolsa exigindo o registro do motivo de encerramento e da data de término efetivo, atualizando o status para `ENCERRADA`.

RF17 O sistema deve enviar uma notificação por e-mail ao aluno quando uma bolsa for concedida.

RF18 O sistema deve permitir o cadastro, edição e exclusão manual, por meio do código, e notificações de bolsas, incluindo tipo, assunto e mensagem, vinculadas a um edital.

RF19 O sistema deve permitir o cadastro, edição e exclusão manual, por meio do código, de orientadores, incluindo nome, e-mail e informações institucionais necessárias para associação com bolsistas.

RF20 O sistema deve permitir o cadastro, edição e exclusão manual, por meio do código, de classificadores específicos por edital (por exemplo, regras/métodos de cálculo e ordenação), de modo que diferentes editais possam adotar estratégias distintas de classificação.

RF21 O sistema deve permitir que alunos criem uma conta no Marvin e sejam habilitados com o perfil necessário para realizar inscrições em editais do módulo de bolsas.

RF22 O sistema deve inserir automaticamente, durante a inicialização da aplicação, os dados-base definidos manualmente (por exemplo, editais, orientadores, notificações e classificadores), quando não existirem na base, de modo a tornar o ambiente inicial utilizável e reprodutível.

3.4 Requisitos não funcionais

Os requisitos não funcionais são agrupados por categorias, identificados pelo prefixo **RNF**.

Segurança

RNF01 O módulo de bolsas deve reutilizar o mecanismo de autenticação e autorização do Marvin, garantindo que somente usuários autenticados acessem as funcionalidades.

RNF02 O sistema deve restringir ações administrativas e de configuração do módulo (por exemplo, manutenção de editais, definição de regras específicas de classificação por edital, manutenção de orientadores e notificações, geração de classificação, concessão e encerramento de bolsas) a perfis autorizados, como secretaria, comissão de bolsas e coordenação.

RNF03 Dados sensíveis de alunos (e-mail, matrícula, faixa de renda) devem ser armazenados e transmitidos de forma segura, em conformidade com as práticas institucionais vigentes.

Auditoria e rastreabilidade

RNF04 O sistema deve manter histórico de classificações, incluindo pontuação e posição atribuídas a cada candidato, bem como o edital de referência.

RNF05 Alterações de status de inscrição e de bolsa (por exemplo, de `CLASSIFICADO` para `INDEFERIDO`) devem ser rastreáveis para fins de auditoria interna.

Usabilidade

RNF06 As telas destinadas à secretaria e à coordenação devem apresentar visão consolidada dos editais e bolsistas, com indicadores e ações principais facilmente acessíveis.

RNF07 Formulários de inscrição devem ser claros e concisos, com mensagens de validação em português e instruções sobre campos obrigatórios.

Manutenibilidade e integração

RNF08 O módulo deve seguir a arquitetura em camadas do Marvin, separando claramente responsabilidades de apresentação, aplicação e persistência.

RNF09 O código deve obedecer às convenções de nomenclatura e organização de pacotes existentes, facilitando a manutenção futura por outros membros da equipe de desenvolvimento.

RNF10 O módulo deve ser implantado no mesmo servidor de aplicações (WildFly) e utilizar o mesmo banco de dados PostgreSQL do Marvin, reduzindo o esforço de operação e suporte.

RNF11 A inicialização automática de dados-base do módulo deve ser idempotente,evitando duplicação de registros em reimplantações e garantindo reprodutibilidade do ambiente de testes e demonstração.

3.5 Regras de negócio

Segundo Falbo (2018), regras de negócio são declarações que definem ou restringem aspectos do negócio (políticas, normas e regulações), normalmente externas ao sistema, mas que devem ser cumpridas por ele. Neste trabalho, as regras de negócio do módulo de Gestão de Bolsas são apresentadas com o prefixo **RN**.

RN01 É vedado manter mais de uma inscrição *ativa* de um mesmo aluno, para o mesmo edital, associada ao mesmo número de matrícula.

RN02 Inscrições somente podem ser registradas para editais abertos e durante o respectivo período de inscrição.

RN03 Um aluno não pode possuir duas bolsas vigentes simultaneamente no mesmo programa, salvo exceções explicitamente autorizadas pelo colegiado (não contempladas neste trabalho).

RN04 A classificação de candidatos deve respeitar a configuração de vagas por combinação *nível–categoria*. Candidatos que excedam o limite de vagas de sua combinação devem compor lista de espera.

RN05 Para fins de concessão, todo bolsista deve estar associado a um orientador docente cadastrado.

RN06 O encerramento de uma bolsa requer o registro da data de término efetivo e do motivo de encerramento, para garantir rastreabilidade institucional.

3.6 Histórias de usuário e critérios de aceitação

Para apoiar a comunicação com a equipe de coordenação e secretaria, alguns requisitos foram capturados na forma de histórias de usuário. Por uma questão de tempo, uma limitação do trabalho é a ausência de rastreabilidade entre as histórias de usuário e os requisitos elencados anteriormente. As tabelas 1, 2 e 3 ilustram exemplos de histórias e seus critérios de aceitação.

Tabela 1 – Histórias de usuário – Secretaria / Coordenação / Comissão.

ID	História e critérios de aceitação
US-02	**História:** Como coordenação, quero configurar o número de vagas por nível e categoria antes de gerar a classificação para que o sistema respeite as cotas definidas. **Critérios de aceitação:** • O sistema deve listar todas as combinações nível–categoria relevantes para o edital; • O usuário deve informar, para cada combinação, um número inteiro maior ou igual a zero; • A classificação só é concluída se houver uma configuração válida para todas as combinações.
US-03	**História:** Como comissão de bolsas, quero indeferir em lote todos os candidatos classificados ou em lista de espera em um edital para publicar um novo resultado. **Critérios de aceitação:** • A ação deve alterar o status de todas as inscrições `CLASSIFICADO` e `LISTA_ESPERA` para `INDEFERIDO`; • O sistema deve exibir mensagem de confirmação antes da operação; • Após a operação, nenhuma inscrição do edital deve manter status `CLASSIFICADO` ou `LISTA_ESPERA`.
US-06	**História:** Como coordenação, quero publicar o resultado final da classificação de um edital para que o processo seletivo possa avançar com segurança. **Critérios de aceitação:** • O sistema só deve permitir marcar o resultado como *final* se houver configuração de vagas válida para todas as combinações *nível–categoria* do edital; • Caso exista alguma combinação sem configuração, o sistema deve bloquear a finalização e informar quais combinações estão pendentes; • Uma vez marcado como *final*, o resultado deve ficar visível para consulta pela secretaria e coordenação.
US-07	**História:** Como coordenação, quero invalidar (ou substituir) um resultado de classificação já gerado para que seja possível publicar uma versão corrigida. **Critérios de aceitação:** • O sistema deve solicitar confirmação antes de invalidar/substituir o resultado; • Ao invalidar/substituir, o sistema deve atualizar, em lote, o status das inscrições afetadas conforme a operação (por exemplo, marcando como `INDEFERIDO` as inscrições em `CLASSIFICADO` e `LISTA_ESPERA`); • Após a operação, o edital não deve manter inscrições com status `CLASSIFICADO` ou `LISTA_ESPERA` associadas ao resultado anterior.
US-08	**História:** Como secretaria, quero efetivar a concessão de bolsa a partir do resultado final para que o vínculo do bolsista seja criado e rastreável no sistema. **Critérios de aceitação:** • A criação do bolsista deve ser permitida apenas para inscrições com status `CLASSIFICADO` no resultado final do edital; • Ao criar o bolsista, o sistema deve atualizar o status da inscrição correspondente para `BOLSISTA`; • O sistema deve registrar o edital de origem e os dados essenciais do vínculo (por exemplo, orientador, data de início e data prevista de término); • Ao concluir, o sistema deve exibir uma mensagem de confirmação.

Tabela 2 – Histórias de usuário – Secretaria (cont.), Aluno / Bolsista.

ID	História e critérios de aceitação
US-09	**História:** Como secretaria, quero encerrar o vínculo de bolsa de um bolsista registrando data e motivo para manter o histórico e a prestação de contas do programa. **Critérios de aceitação:** • Campos obrigatórios no encerramento: data de término efetivo e motivo de encerramento; • O sistema deve impedir o encerramento caso algum campo obrigatório não seja informado; • Ao concluir o encerramento, o sistema deve atualizar o status da bolsa para `ENCERRADA`; • O sistema deve manter o histórico do bolsista disponível para consulta posterior.
US-10	**História:** Como aluno do PPGI, quero criar uma conta no Marvin e ser habilitado com o perfil adequado para acessar o formulário de inscrição em editais de bolsa. **Critérios de aceitação:** • O sistema deve permitir que o aluno crie uma conta e realize autenticação no Marvin; • Após autenticado, o aluno deve possuir permissão para acessar a tela de inscrição do módulo de bolsas; • A tentativa de acessar o formulário de inscrição sem autenticação deve ser bloqueada, exibindo mensagem apropriada; • A conta criada deve estar associada ao perfil/papel necessário para realizar inscrições no módulo.
US-04	**História:** Como aluno do PPGI, quero me inscrever em um edital de bolsa selecionando o edital e preenchendo meus dados para concorrer à vaga. **Critérios de aceitação:** • O formulário de inscrição deve listar apenas editais abertos; • A matrícula deve ser validada para evitar múltiplas inscrições ativas no mesmo edital; • Ao concluir a inscrição com sucesso, o sistema deve exibir uma mensagem de confirmação.
US-05	**História:** Como bolsista, quero que meu orientador e a data prevista de término da bolsa estejam corretos no sistema para que a coordenação acompanhe meu vínculo adequadamente. **Critérios de aceitação:** • A tela de informações do bolsista deve exibir orientador, início, término previsto e edital de origem; • Alterações nesses campos só podem ser feitas por secretaria ou coordenação; • O sistema deve registrar a atualização com sucesso e manter a consistência das informações.

Tabela 3 – Histórias de usuário – Equipe de TI / Administração do Marvin.

ID	História e critérios de aceitação
US-01	**História:** Como membro da equipe de TI, quero cadastrar e abrir um edital de bolsa para que os alunos possam se inscrever no período definido. **Critérios de aceitação:** • Campos obrigatórios: título, período de inscrição, agência de fomento; • Ao salvar o edital com status `ABERTO`, ele deve aparecer na lista de editais; • O edital aberto deve ser exibido como opção no formulário de inscrição dos alunos.
US-11	**História:** Como equipe de TI / administração do Marvin, quero que o módulo carregue automaticamente os dados-base do subsistema de bolsas na inicialização para que o ambiente fique pronto para demonstração, testes e uso inicial sem intervenção manual no banco. **Critérios de aceitação:** • Ao iniciar a aplicação, o sistema deve criar os dados-base do módulo quando não existirem (por exemplo, editais, orientadores e notificações iniciais vinculadas a editais); • A inicialização deve ser **idempotente**: reinicializações não devem duplicar registros já existentes; • O sistema deve registrar em *log* as decisões de criação ou de reaproveitamento (por exemplo, "criado" vs. "ignorado"); • Ao término da inicialização, as telas do módulo devem estar operacionais com dados mínimos para navegação e execução dos fluxos centrais.
US-12	**História:** Como equipe de TI / administração do Marvin, quero manter por código os cadastros de orientadores, notificações e classificadores por edital para que o módulo suporte variações de regras entre editais sem depender de telas administrativas. **Critérios de aceitação:** • Deve ser possível definir e alterar por código os orientadores disponíveis (nome, e-mail e dados institucionais necessários); • Deve ser possível definir e alterar por código as notificações vinculadas a um edital (tipo, assunto e mensagem); • Deve ser possível definir e alterar por código classificadores específicos por edital (estratégias/métodos de cálculo e ordenação), permitindo variações por agência/-modalidade; • As alterações por código devem refletir no comportamento do sistema após implantação/reinicialização, sem necessidade de intervenção direta no banco.

3.7 Modelos de caso de uso

Os casos de uso capturam, em nível mais formal, as interações entre atores e o sistema. A Figura 2 ilustra um diagrama de casos de uso em alto nível para o módulo de bolsas, com os atores principais:

Tabela 4 – Descrição dos atores envolvidos nos casos de uso (Elaboração própria).

Ator	Descrição
Comissão de Bolsas	Define critérios e conduz o processo de classificação/homologação das inscrições, acompanhando resultados e regras aplicáveis aos editais.
Aluno	Realiza inscrição em editais, acompanha o status da candidatura e, quando bolsista, consulta informações do vínculo e notificações relacionadas.
Equipe de TI	Mantém a infraestrutura e a disponibilidade do sistema, apoiando implantação, configuração, monitoramento e manutenção técnica do módulo.

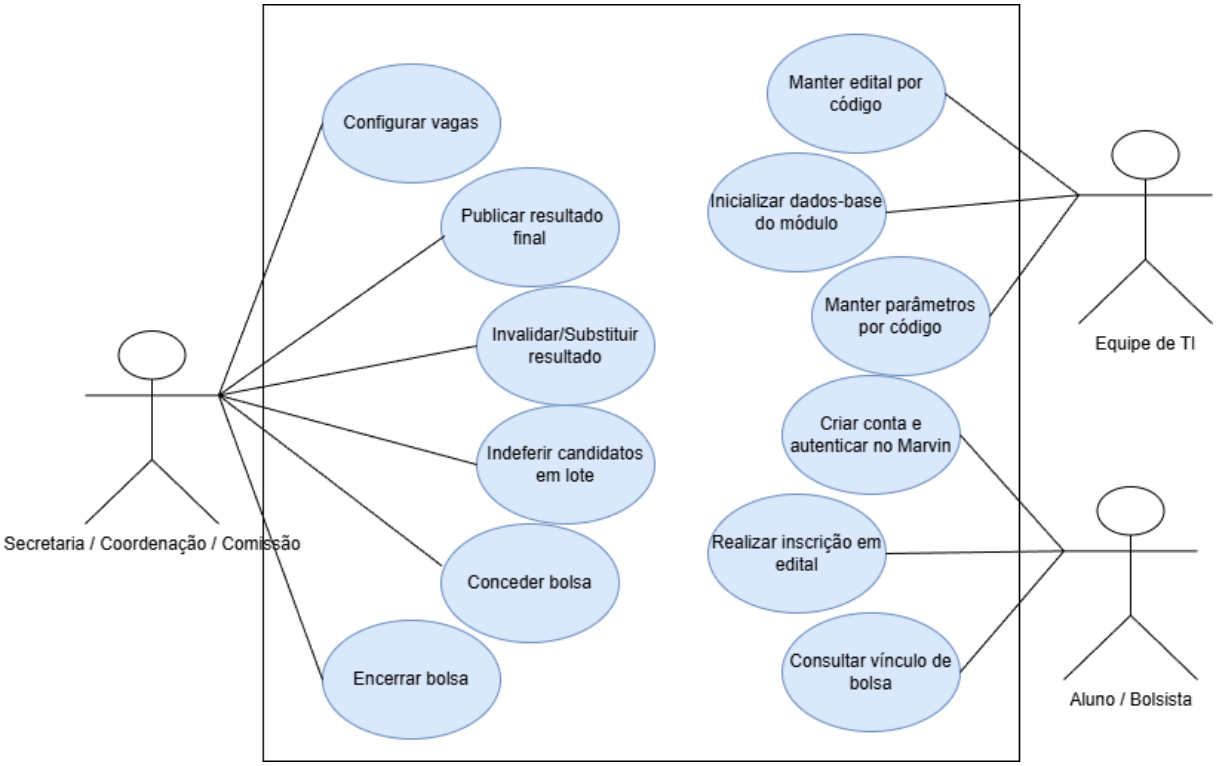

Figura 2 – Diagrama de Casos de Uso do subsistema Gestão de Bolsas

Os casos de uso centrais incluem, entre outros:

- **UC01 – Gerenciar edital de bolsa**: inclui cadastrar, editar, abrir e encerrar editais;
- **UC02 – Realizar inscrição em edital**: aluno preenche formulário, selecionando edital e fornecendo dados acadêmicos;
- **UC03 – Configurar cotas e gerar classificação**: coordenação define vagas por nível–categoria e aciona o processo de classificação;

- **UC04 – Conceder bolsa a candidato**: criação de registro de bolsista a partir de uma candidatura classificada;
- **UC05 – Consultar e gerenciar bolsistas**: secretaria e coordenação consultam e atualizam informações de vínculo;
- **UC06 – Encerrar bolsa**: registro de término efetivo e motivo de encerramento;
- **UC07 – Indeferir candidatos**: operação em lote para marcar candidatos como `INDEFERIDO` em um edital;

3.8 Subsistemas e interdependências

O módulo de Gestão de Bolsas é concebido como um subsistema dentro do Marvin, cujas funcionalidades são organizadas de forma modular e reutilizam uma base comum provida pelo **Marvin**. O Marvin Núcleo concentra as capacidades fundamentais da plataforma (infraestrutura e serviços compartilhados), de modo que módulos específicos possam focar na implementação de regras e processos de seus domínios sem replicar mecanismos transversais.

Em termos de interdependência, o **subsistema de Gestão de Bolsas** depende do **Núcleo do Marvin** para: (i) autenticação e autorização de usuários (controle de perfis/papéis), (ii) acesso a dados institucionais já mantidos pela plataforma (como usuários acadêmicos), e (iii) serviços transversais, como o envio de e-mails por meio do componente *Mailer*. Em contrapartida, o Núcleo não depende do módulo de bolsas, mantendo-se independente e reutilizável por outros subsistemas.

A Figura 3 apresenta, em alto nível, os subsistemas relevantes e suas dependências.

O módulo de Gestão de Bolsas é concebido como um subsistema dentro do Marvin, com dependências claras para componentes já existentes:

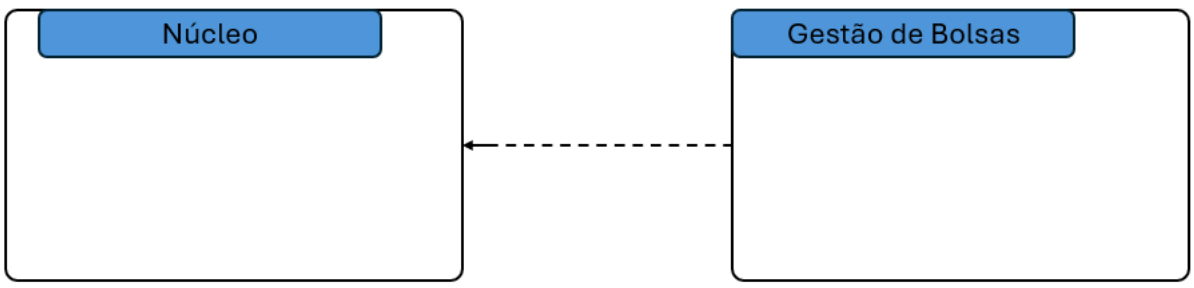

Figura 3 – Diagrama de Pacotes e os Subsistemas Identificados.

Tabela 5 – Subsistemas identificados e suas interdependências.

Subsistema	**Descrição**
Núcleo	Módulo base da plataforma Marvin, responsável por funcionalidades transversais e estruturantes, como cadastro e autenticação de usuários acadêmicos, além de serviços compartilhados utilizados pelos demais subsistemas.
Gestão de Bolsas (PPGI)	Subsistema proposto neste trabalho, responsável por apoiar o ciclo de vida das bolsas do PPGI: definição/publicação de editais, recebimento de inscrições, classificação, concessão do vínculo, acompanhamento de vigência, registro de encerramentos e notificações associadas.

Do ponto de vista de independência, o módulo de bolsas não redefine conceitos já presentes no Marvin Core (usuário, permissões, configuração de banco de dados), utilizando-os como base. Por outro lado, introduz novas entidades de negócio (por exemplo, Edital de Bolsas (*ScholarshipNotice*), Inscrição (*ScholarshipApplication*), Vínculo de bolsa / Bolsista (*ScholarshipStudent*)) e serviços especializados que podem, no futuro, ser reutilizados por outros módulos.

3.9 Modelo conceitual

O modelo conceitual do módulo de Gestão de Bolsas é composto por entidades que representam os principais elementos do processo: editais, inscrições, classificações, bolsistas, notificações e orientadores.

A Figura 4 apresenta o diagrama de classes em alto nível. Embora a modelagem conceitual seja própria da fase de requisitos, este diagrama inclui alguns atributos para tornar explícitas as informações relevantes registradas ao longo do processo.

Em alto nível, destacam-se as seguintes classes de domínio:

- **Edital de bolsa (*ScholarshipNotice*)**: representa o instrumento administrativo pelo qual o programa define uma oferta de bolsas e suas regras. Nele constam informações como título e descrição, o período em que as inscrições podem ser realizadas, parâmetros operacionais do processo (por exemplo, periodicidade de relatórios) e um apontamento para o documento do edital (PDF). Ao longo do ciclo de vida, um edital passa por estados (por exemplo, aberto, encerrado), recebe inscrições de candidatos, dá origem a um resultado de classificação e, a partir desse resultado, pode gerar vínculos de bolsistas. O edital também serve de referência para as comunicações enviadas durante o processo (por exemplo, avisos de aceite, lembretes e encerramentos).

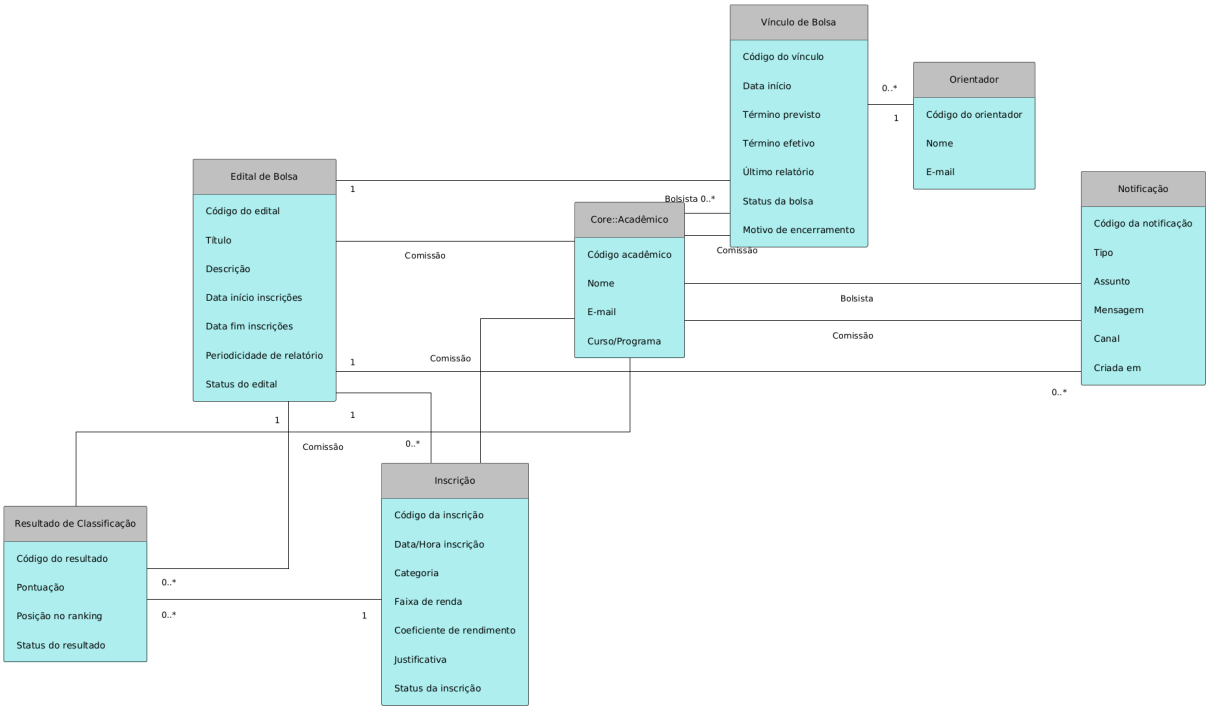

Figura 4 – Diagrama de Classes de Domínio.

- **Inscrição (*ScholarshipApplication*)**: modela a inscrição de um candidato em um edital. Inclui dados do aluno (nome, e-mail, matrícula), informações acadêmicas (curso, nível de formação), faixa de renda, categoria de participação, coeficiente de rendimento, justificativa e `status` da inscrição;
- **Resultado de classificação da inscrição (*ScholarshipApplicationClassification*)**: registra o desfecho de uma inscrição em um processo de classificação associado ao edital, incluindo a pontuação calculada, a posição no ranking e o status do resultado (por exemplo, `CLASSIFICADO`, `LISTA_ESPERA`, `INDEFERIDO`);
- **Vínculo de bolsa / Bolsista (*ScholarshipStudent*)**: representa a materialização de uma bolsa concedida a um aluno a partir do resultado do edital. Mantém as datas de início, término previsto e término efetivo, informações de acompanhamento (por exemplo, último relatório), o status do vínculo e a associação ao orientador docente e ao edital de origem;
- **Orientador (*ScholarshipInstructor*)**: representa o docente responsável por orientar bolsistas vinculados ao programa, armazenando dados de identificação e contato, além de informações institucionais necessárias para caracterização do vínculo;
- **Notificação (*ScholarshipNotification*)**: representa mensagens emitidas no contexto de um edital ao longo do processo (por exemplo, comunicação de aceite, lembrete de relatório, aviso de encerramento). Cada notificação registra seu tipo,

assunto, conteúdo textual e data/hora de criação, sempre associada ao edital que motivou sua emissão;

- **Configuração de vagas por nível e categoria (*ScholarshipQuotaConfig*)**: estrutura de apoio utilizada na etapa de classificação para registrar, para cada combinação *(nível, categoria)*, o número de vagas definido no edital.

As relações entre essas classes permitem rastrear o fluxo completo de um candidato: da inscrição em um edital (Inscrição (*ScholarshipApplication*)), passando pela classificação (Resultado de classificação da inscrição (*ScholarshipApplicationClassification*)), até a eventual concessão e acompanhamento da bolsa (Vínculo de bolsa / Bolsista (*ScholarshipStudent*)) sob responsabilidade de um orientador (Orientador (*ScholarshipInstructor*)). De forma complementar, o módulo mantém notificações vinculadas ao edital (Notificação (*ScholarshipNotification*)), o que facilita a rastreabilidade de comunicações e lembretes ao longo do processo.

4 Projeto do Sistema e Implementação

Este capítulo descreve o projeto e a implementação do módulo de *Gestão de Bolsas do PPGI* integrado ao Marvin. São apresentadas as tecnologias utilizadas, a arquitetura de software, a modelagem do subsistema, o projeto de dados, a implementação dos principais casos de uso e uma visão geral das telas implementadas.

4.1 Tecnologias e ferramentas utilizadas

O módulo de Gestão de Bolsas foi implementado utilizando o mesmo *stack* tecnológico do Marvin, de modo a facilitar sua integração e manutenção a longo prazo. As principais tecnologias e ferramentas adotadas são:

- **Linguagem de programação:** Java, seguindo o paradigma orientado a objetos e boas práticas de modularização e reutilização de componentes;
- **Plataforma:** Jakarta EE, explorando recursos como EJB (Enterprise Java Beans), CDI, JPA (Jakarta Persistence), Bean Validation e Jakarta Faces;
- **Servidor de aplicação:** WildFly, utilizado para empacotar e executar o *war* do Marvin, incluindo o módulo de bolsas;
- **Gerenciador de dependências e build:** Maven, responsável por gerenciar bibliotecas, empacotar o projeto e integrar com o plugin de implantação no WildFly;
- **Banco de dados:** PostgreSQL, reutilizando a instância já configurada para o Marvin, com novas tabelas específicas para editais, inscrições, classificações, bolsistas, orientadores e notificações;
- **Framework de interface web:** Jakarta Faces com PrimeFaces, para construção das páginas *XHTML* e componentes ricos (tabelas, diálogos, selects, calendário, etc.);
- **Template de interface:** AdminFaces, já utilizado pelo Marvin, fornecendo o *layout* principal, cabeçalhos, barras de navegação e estilos padronizados;
- **Bibliotecas e utilitários do ecossistema Marvin:** o módulo segue as mesmas convenções e componentes compartilhados já empregados em outros subsistemas do Marvin, como o utilitário JButler para suporte a classes persistentes e DAOs base, além de componentes transversais (por exemplo, infraestrutura de e-mail e configuração do sistema);
- **Ferramentas de apoio:** IDE com suporte a Java e Maven (por exemplo, IntelliJ IDEA), Git para controle de versão e clientes de banco de dados para inspeção das tabelas do módulo.

A adoção dessas tecnologias garante coerência com os demais módulos do Marvin, reduz o esforço de operação por parte da equipe de TI e favorece a evolução incremental do sistema.

4.2 Arquitetura de software do módulo

A arquitetura do módulo segue o padrão em **camadas** utilizado no Marvin, aderente ao estilo *Service Layer*, no qual o software é organizado em camadas bem definidas e o acesso às regras de negócio é centralizado em serviços de aplicação.

Em alto nível, podem ser destacadas as seguintes camadas e responsabilidades:

- **Camada de apresentação (View / Controller):** composta por páginas *XHTML* com componentes PrimeFaces e controladores JSF anotados com `@Named` e `@ViewScoped`. O objetivo é manter a lógica de interface e o estado de tela encapsulados no escopo de visão, seguindo o padrão MVC (View e Control) e evitando acoplamento com a camada de persistência;
- **Camada de aplicação (Services / Use Cases):** implementada com EJB `@Stateless`, encapsulando a lógica de cada caso de uso, coordenação de DAOs e integração com serviços transversais do Marvin. Esta camada atua como fronteira transacional e de segurança para o módulo, orquestrando operações como abertura/encerramento de editais, classificação com quotas, criação de bolsistas e encerramento de vínculo;
- **Camada de domínio:** formada por entidades JPA que representam conceitos centrais do módulo, tais como *ScholarshipNotice*, *ScholarshipApplication*, *ScholarshipApplicationClassification*, *ScholarshipStudent*, *ScholarshipInstructor* e *ScholarshipNotification*, além de *value objects* como *ScholarshipQuotaConfig* e dos *enums* de apoio. Por convenção do ecossistema Marvin, as classes de domínio seguem o mesmo padrão de persistência e reuso de infraestrutura já empregado em outros módulos;
- **Camada de persistência:** constituída por DAOs especializados (interfaces e implementações) que estendem utilitários base do ecossistema Marvin (via JButler), concentrando operações de CRUD e consultas (JPQL) em um ponto dedicado. Essa decisão reduz duplicação de código e mantém a lógica de consulta desacoplada dos serviços de aplicação.

A Figura 5 sintetiza a organização da arquitetura em camadas e partições adotada no trabalho, evidenciando a separação entre o módulo de *Gestão de Bolsas* e o *core* do Marvin. Em cada partição, os componentes são agrupados nas camadas de Interface com o Usuário (CIU), Lógica de Negócio (CLN) e Gerência de Dados (CGD), com dependências direcionadas das camadas superiores para as inferiores.

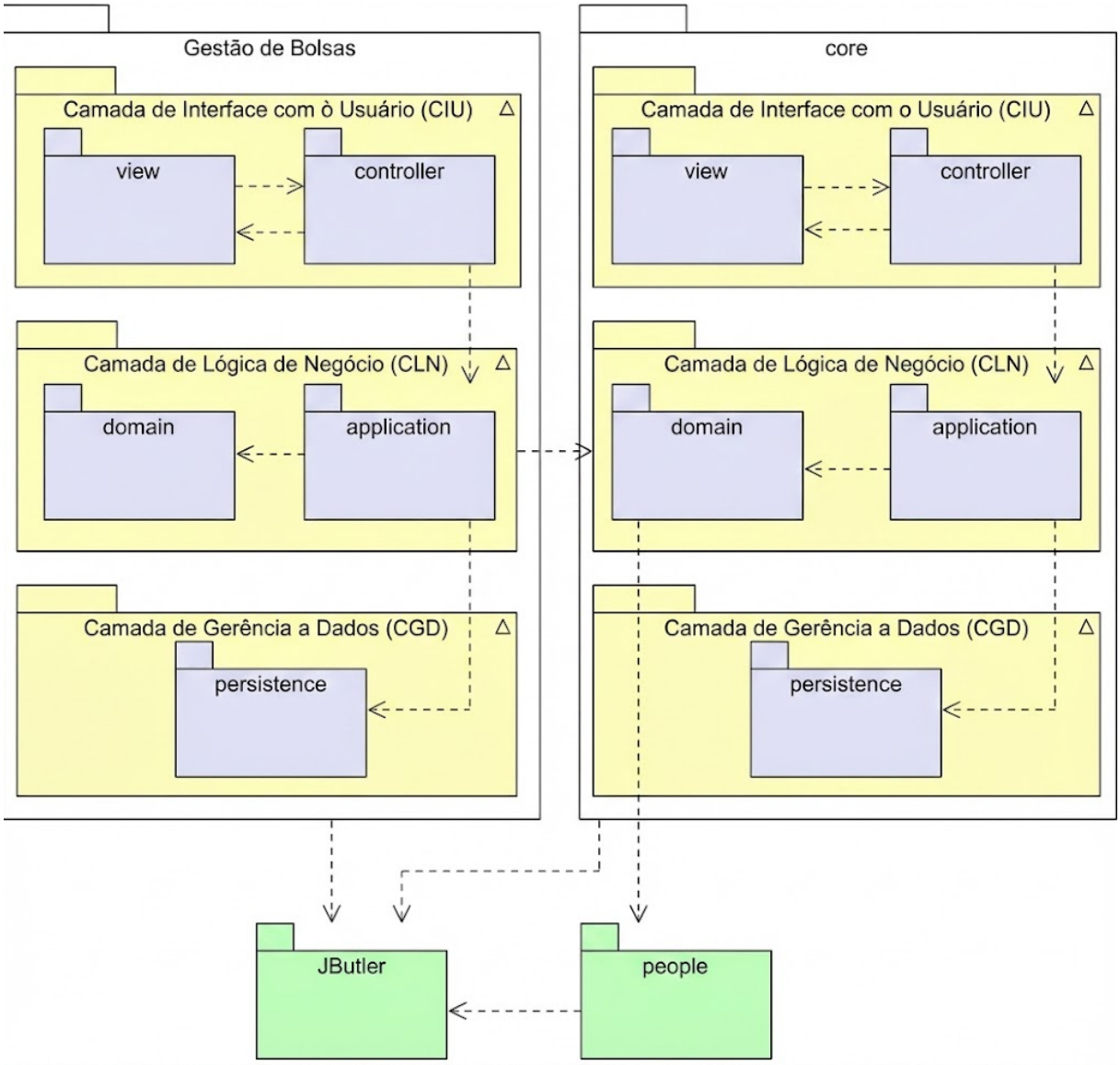

Figura 5 – Diagrama demostrando organização da arquitetura do Projeto

Integração estrutural com o Marvin.

O módulo é empacotado no mesmo *war* do Marvin e segue a organização de pacotes por subsistema, mantendo o pacote base do sistema e separando arquivos de apresentação em pastas específicas do módulo (páginas XHTML) e a implementação Java em pacotes com o prefixo `br.ufes.inf.labes.marvin.scholarship.`. Essa estratégia preserva a padronização estrutural do Marvin e facilita manutenção e localização de artefatos (view, controller, application, persistence e domain).

Integração com serviços transversais do Marvin (e-mail).

O módulo reutiliza a infraestrutura de e-mail do Marvin baseada em eventos. Em vez de enviar e-mails diretamente a partir dos serviços de bolsa, o módulo dispara um evento

de envio (com destinatário, template e modelo de dados), observado por um componente central (*Mailer*) responsável por compor a mensagem e enviá-la via Jakarta Mail com base na configuração SMTP do sistema. Esse acoplamento indireto reduz dependências entre módulos e padroniza o envio de notificações na plataforma.

4.3 Projeto dos Componentes da Arquitetura

A modelagem do módulo de bolsas foi organizada em visões complementares, articulando a perspectiva de entidades de domínio, serviços de aplicação, persistência e navegação.

A arquitetura do módulo utiliza *frameworks* já consolidados no desenvolvimento Web com Jakarta EE, incluindo JSF como controlador frontal, CDI para injeção de dependências, JPA para mapeamento objeto-relacional e PrimeFaces como biblioteca de componentes de interface.

4.3.1 Camada de Lógica de Negócio

Esta seção apresenta o projeto da **Componente de Domínio do Problema** do módulo (pacote `domain`), responsável por representar os principais conceitos do processo real de gestão de bolsas (por exemplo, edital, inscrição, resultado de classificação, vínculo de bolsa, orientador e notificações).

Assim como ocorre no Marvin, as classes persistentes do pacote `domain` reutilizam o utilitário JButler. Em particular, as entidades do domínio herdam de uma classe base que provê identificador e funcionalidades comuns a objetos persistentes. Para evitar poluir o diagrama principal, as relações de herança com as classes do JButler podem ser apresentadas em um diagrama separado (Figura 7), mantendo o foco do leitor nos conceitos do domínio.

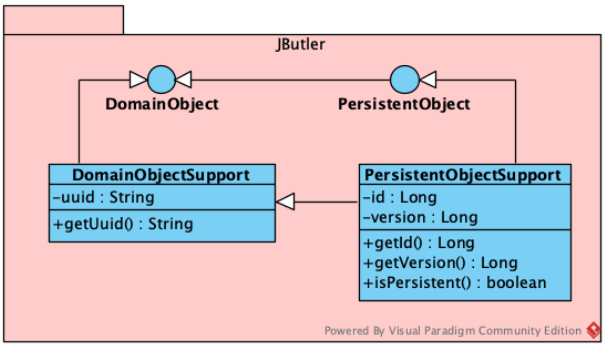

Figura 6 – Classes base do JButler utilizadas por entidades persistentes do módulo.

No módulo de Gestão de Bolsas, destacam-se as seguintes entidades de domínio (nomes técnicos em itálico apenas como referência à implementação):

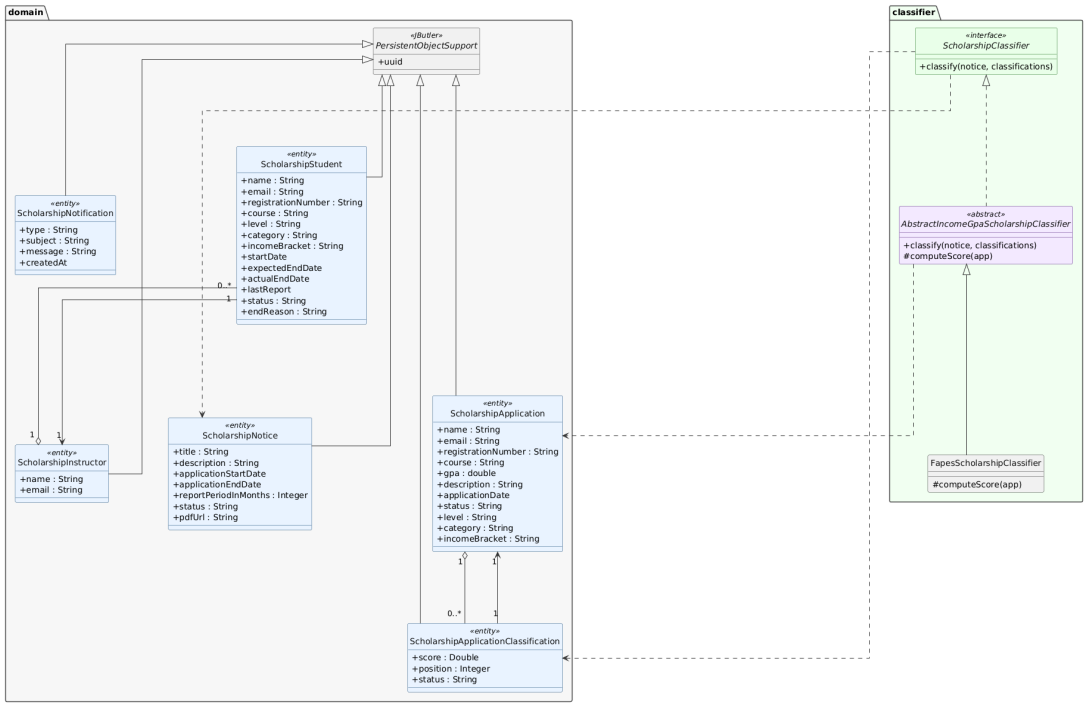

Figura 7 – Modelo de Classes de Domain

- **Edital de bolsas(*ScholarshipNotice*)**: representa a chamada de bolsa, registrando informações essenciais como título, período de inscrição, status e referência ao documento do edital. O edital agrega inscrições e mantém histórico de notificações relacionadas ao processo.
- **Inscrição (*ScholarshipApplication*)**: representa a candidatura de um aluno um edital, armazenando dados do candidato e informações necessárias para avaliação(por exemplo, nível, categoria, faixa de renda e coeficiente de rendimento), além do status da inscrição.
- **Resultado de classificação (*ScholarshipApplicationClassification*)**: registra o resultado de uma inscrição em um processo de classificação, armazenando pontuação e posição no ranking, e mantendo associação tanto com o edital quanto com a inscrição.
- **Vínculo de bolsa / Bolsista (*ScholarshipStudent*)**: representa a concessão efetiva da bolsa a um aluno, contendo dados do vínculo (datas, status e encerramento) e mantendo associação com o edital de origem e com o orientador.
- **Orientador (*ScholarshipInstructor*)**: representa o docente que orienta bolsistas, permitindo relacionar orientadores e seus orientandos no módulo.
- **Notificação (*ScholarshipNotification*)**: representa comunicações vinculadas a um edital (por exemplo, aceite, lembrete e encerramento), registrando tipo, assunto, mensagem e data/hora de criação.

Perfis de acesso (Academic e papéis do Marvin Core).

O módulo de Gestão de Bolsas reutiliza o mecanismo de autenticação/autorização provido pelo Marvin Core, no qual usuários do sistema são representados pela entidade *Academic* e seus perfis de acesso são atribuídos por meio de um conjunto de papéis (*Role*).

Para atender às necessidades do módulo, foram adicionados ao núcleo três novos papéis, definidos em *Role* como constantes de nome: *Bolsista*, *Gestão de Bolsas* e *Secretaria do PPGI*. Esses papéis permitem separar responsabilidades entre (i) o aluno/bolsista, (ii) os usuários com permissão de operar funcionalidades administrativas do módulo, e (iii) a secretaria do programa.

A implementação do módulo se apoia nas capacidades já existentes do Marvin Core para manipulação desses papéis, incluindo a recuperação de um papel pelo nome e a listagem de usuários que possuem um determinado papel, além da própria estrutura de persistência que representa a associação *Academic–Role* por tabela associativa. Dessa forma, o módulo estende o conjunto de perfis disponíveis sem redefinir o mecanismo de controle de acesso, mantendo o núcleo independente e promovendo reuso de infraestrutura.

Estratégia de classificação (componente de domínio/aplicação).

A etapa de classificação é parametrizada por uma estratégia, permitindo encapsular regras específicas por tipo de edital/agência. Essa estratégia é definida por uma interface (*ScholarshipClassifier*) e uma implementação abstrata que organiza o fluxo de cálculo de pontuação e ranqueamento (por exemplo, agrupando candidatos por nível e categoria, atribuindo posições e aplicando limites por grupo quando configurado).

Estruturas auxiliares (não persistentes).

A configuração de vagas por combinação *(nível, categoria)* é representada por um *value object* (*ScholarshipQuotaConfig*) utilizado como DTO em memória entre a interface (JSF/PrimeFaces) e a camada de aplicação. Por não representar um conceito persistente do domínio, essa estrutura não é mapeada como entidade e não gera tabela no banco.

4.3.2 Camada de Interface com o Usuário

A Camada de Interface com o Usuário do módulo de Gestão de Bolsas segue o padrão MVC adotado no ecossistema Marvin. As páginas Web (XHTML) representam a *View* e são organizadas no pacote `view`, enquanto as classes Java responsáveis por intermediar as ações do usuário representam o *Controller* e encontram-se no pacote `controller`. A porta de entrada para o *Model* do MVC é fornecida pelas interfaces/serviços da camada `application`, mantendo o *back-end* independente do *front-end* e preservando dependências

unidirecionais dos controladores para os serviços.

A Figura 8 apresenta o diagrama UML da Camada de Interface com o Usuário do módulo, distinguindo os elementos por **tipo, cor e estereótipo**: as **Views** (páginas XHTML) são representadas por círculos azuis e anotadas com «view» (V_MENU, V_FORM, V_NOT, V_RES, V_STU, V_INF); os **Controllers** JSF são representados por círculos em cor diferenciada e anotados com «controller» (C_FORM, C_NOT, C_RES, C_STU, C_INF); e as **interfaces de serviço** da camada application são representadas por caixas e anotadas com «service» (S_FORM, S_NOT, S_CLS, S_RES, S_STU, S_INF). As siglas utilizadas no diagrama são definidas na Tabela 6.

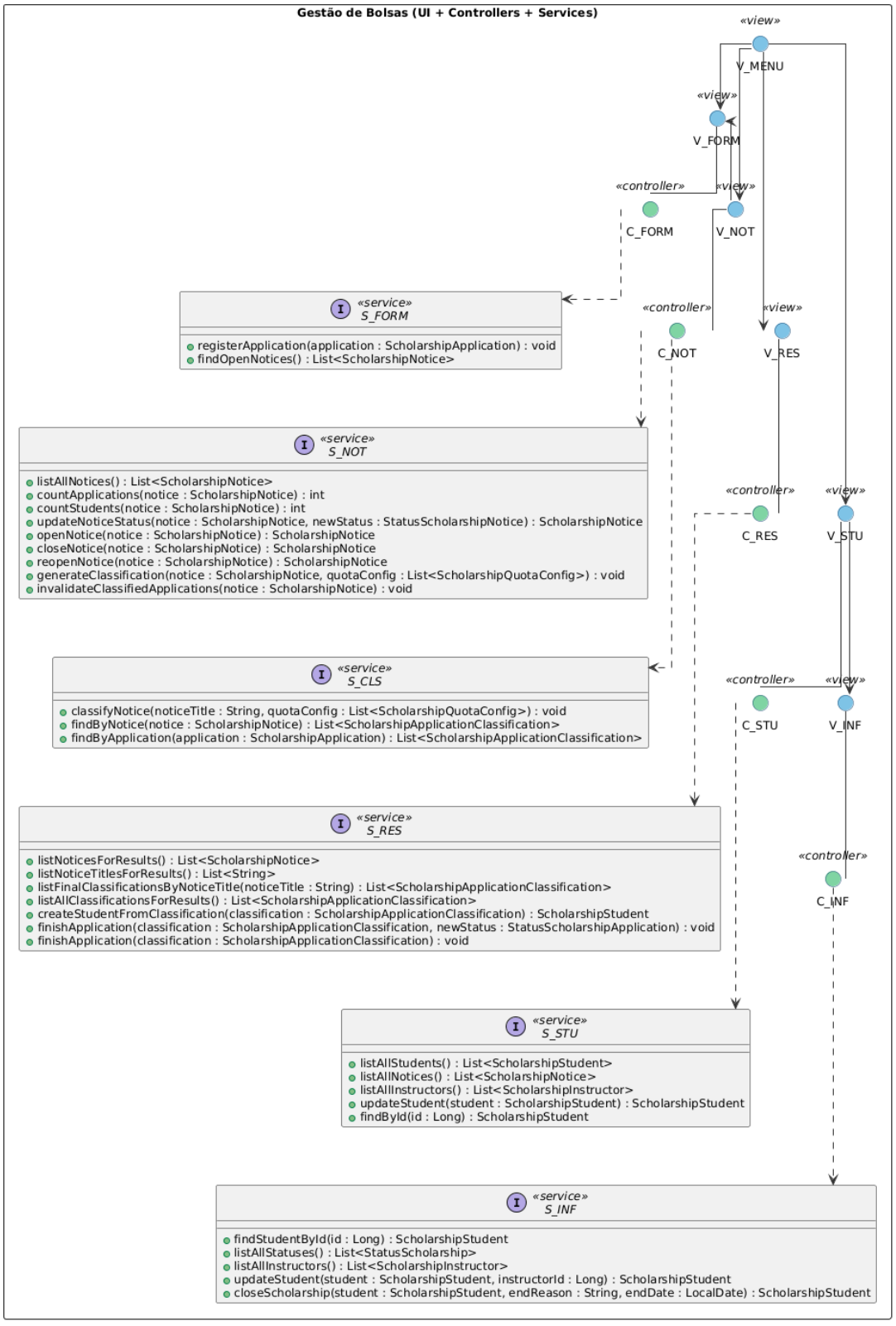

Figura 8 – Projeto da Camada de Interface com o Usuário do módulo de Gestão de Bolsas (view e controller), em dependência com interfaces da camada application.

Tabela 6 – Abreviações utilizadas nos diagramas do módulo de Gestão de Bolsas.

Categoria	**Abreviação**	**Elemento (nome completo)**
View (XHTML)	`V_MENU`	`menu_scholarship.xhtml`
View (XHTML)	`V_FORM`	`applicationForm/index.xhtml`
View (XHTML)	`V_NOT`	`manageScholarshipNotice/index.xhtml`
View (XHTML)	`V_RES`	`manageScholarshipApplication/index.xhtml`
View (XHTML)	`V_STU`	`manageScholarshipStudent/index.xhtml`
View (XHTML)	`V_INF`	`manageScholarshipStudentInformation/index.xhtml`
Controller	`C_FORM`	`ApplicationFormController`
Controller	`C_NOT`	`ManageScholarshipNoticeController`
Controller	`C_RES`	`ManageScholarshipApplicationController`
Controller	`C_STU`	`ManageScholarshipStudentController`
Controller	`C_INF`	`ManageScholarshipStudentInformationController`
Service	`S_FORM`	`ApplicationFormService`
Service	`S_NOT`	`ManageScholarshipNoticeService`
Service	`S_CLS`	`ScholarshipApplicationClassificationService`
Service	`S_RES`	`ManageScholarshipApplicationService`
Service	`S_STU`	`ManageScholarshipStudentService`
Service	`S_INF`	`ManageScholarshipStudentInformationService`
Service	`S_NTF`	`ManageScholarshipNotificationService`

Controladores JSF (pacote `controller`).

Os controladores do módulo são *managed beans* anotados com `@Named` e escopo de visão (`@ViewScoped`), mantendo o estado da tela e delegando a lógica transacional para EJBs da camada `application`. A inicialização das telas é centralizada em métodos `@PostConstruct`, nos quais os controladores carregam listas auxiliares (por exemplo, editais, status e orientadores), preparam modelos para *binding* e tratam parâmetros de navegação.

Em particular, destacam-se:

- **Formulário de inscrição (`ApplicationFormController`):** controla a tela de inscrição do candidato. Inicializa uma nova inscrição e carrega os editais abertos para o seletor; também suporta pré-seleção de edital via parâmetro de requisição (por exemplo, `noticeTitle`), facilitando navegação a partir de outras telas. A submissão do formulário delega ao serviço de aplicação o registro da inscrição e apresenta mensagens de feedback ao usuário;

- **Overview de editais (`ManageScholarshipNoticeController`):** fornece os dados e ações da tela de acompanhamento dos editais. Atua como fachada de apresentação sobre o serviço responsável por listar editais, computar métricas agregadas, abrir/encerrar editais e acionar operações relacionadas à classificação (incluindo o diálogo de configuração de vagas);
- **Resultados/Classificação (`ManageScholarshipApplicationController`):** controla a tela de consulta e operação sobre listas de classificação final. Carrega os títulos de editais com resultados disponíveis, aplica filtros em memória (status/ano/categoria/nível/busca textual) e expõe ações como criar bolsista a partir de um classificado e finalizar candidatura em caso de desistência;
- **Listagem de bolsistas (`ManageScholarshipStudentController`):** controla a tela de listagem e filtragem de vínculos de bolsa. Carrega a lista de bolsistas e listas auxiliares (editais/status/orientadores), aplicando filtros em memória e oferecendo navegação para a tela de detalhes do bolsista;
- **Detalhes do bolsista (`ManageScholarshipStudentInformationController`):** controla a tela de visualização/edição de um bolsista específico. Recupera o `studentId` via parâmetro de requisição, carrega o bolsista e as listas auxiliares para edição, e delega ao serviço a validação/persistência de alterações (status, datas, orientador) e o encerramento com motivo e data efetiva.

Páginas XHTML (pacote `view`).

Neste parágrafo , descrevem-se as principais páginas *XHTML* do módulo e como elas interagem com seus respectivos controladores (camada `controller`) e serviços (camada `application`), seguindo o padrão MVC adotado no Marvin.

- **Formulário de inscrição (applicationForm/index.xhtml):** a página apresenta um formulário para o candidato selecionar um edital aberto e preencher seus dados pessoais e acadêmicos. Ao acessar a página, o controlador *ApplicationFormController* inicializa uma nova inscrição com a data corrente, carrega a lista de editais abertos por meio do serviço de aplicação e, quando presente, pré-seleciona um edital via parâmetro de requisição `noticeTitle`. Ao submeter o formulário, é acionado o método `ApplicationFormController.submit()`, que delega o registro ao serviço `ApplicationFormService.registerApplication(application)`, responsável por validar e persistir a inscrição;
- **Overview de editais (manageScholarshipNotice/index.xhtml):** a tela concentra o acompanhamento dos editais e ações administrativas do ciclo do processo. Ao abrir a página, o controlador *ManageScholarshipNoticeController* carrega os

editais via `ManageScholarshipNoticeService.listAllNotices()` e expõe contadores agregados para exibição na interface. Ações como abrir e encerrar edital são executadas por botões que acionam `openNotice()` e `closeNotice()`, delegando ao serviço a transição de estado e persistência. A partir dessa tela também é acionada a etapa de classificação, na qual o controlador prepara a configuração de vagas (quotas) e, após confirmação, dispara a geração do resultado;

- **Resultados/Classificação (manageScholarshipApplication/index.xhtml):** a página é utilizada pela comissão e secretaria para consultar listas finais e executar ações operacionais. Ao abrir a tela, o controlador *ManageScholarshipApplicationController* carrega os títulos de editais com resultados e a lista inicial de classificações por meio do serviço de aplicação, além de popular os filtros (status, categoria, nível e anos). Quando o usuário seleciona um edital, o listener `onNoticeChange()` atualiza a tabela consultando apenas o resultado do edital escolhido; e, ao alterar filtros, o controlador recalcula a lista exibida. A tabela expõe ações como `createStudent(classification)` para criar um bolsista a partir de um candidato aprovado e `finishApplication(classification)` para finalizar a inscrição (por exemplo, desistência), delegando essas operações ao serviço `ManageScholarshipApplicationService`;

- **Listagem de bolsistas (manageScholarshipStudent/index.xhtml):** a tela exibe os vínculos de bolsa em forma de tabela, com filtros por edital de origem, status, orientador e busca textual. Ao inicializar, o *ManageScholarshipStudentController* carrega a lista de bolsistas e as listas auxiliares de filtros (editais e orientadores) chamando o serviço `ManageScholarshipStudentService`. Para navegar aos detalhes de um bolsista selecionado, o controlador constrói o *outcome* para a página de informações, incluindo o parâmetro `studentId` na URL;

- **Informações do bolsista (manageScholarshipStudentInformation/index.xhtml):** a tela apresenta os dados do vínculo e permite atualização controlada. Ao acessar a página, o *ManageScholarshipStudentInformationController* lê o parâmetro `studentId` da requisição, carrega o bolsista via `ManageScholarshipStudentInformationService.findStudentById(id)` e inicializa listas auxiliares de status e orientadores. Alterações gerais do vínculo (status, orientador, datas) são salvas por `saveStudent()`, que delega ao serviço `updateStudent(student, instructorId)`. O encerramento do vínculo é executado por um diálogo: o controlador prepara campos auxiliares em `openCloseDialog()` e, ao confirmar, `closeScholarship()` exige motivo e data de término efetivo, delegando ao serviço `closeScholarship(student, reason, date)` para registrar encerramento e aplicar consistência;

Em conjunto, essas páginas implementam o fluxo operacional do módulo sob a

ótica do usuário: submissão de inscrições por candidatos; acompanhamento e manutenção de editais; consulta e gestão do resultado de classificação; criação e acompanhamento de vínculos de bolsa; e encerramento com registro obrigatório de motivo e data, mantendo a lógica concentrada em serviços e a interface responsável por coletar dados, acionar operações e apresentar feedback.

4.3.3 Camada de Gerência de Dados (persistência)

A Camada de Gerência de Dados do módulo de Gestão de Bolsas segue o padrão *DAO* adotado no ecossistema Marvin, concentrando operações de acesso ao banco em um conjunto dedicado de interfaces e implementações no pacote `persistence`. Essa decisão mantém a lógica de negócio (camadas `application` e `domain`) desacoplada da tecnologia de persistência, além de padronizar operações de CRUD e consultas.

Conforme o padrão do Marvin, o projeto da persistência organiza-se em duas hierarquias complementares: (i) uma hierarquia de **interfaces** com raiz em `BaseDAO` (do utilitário JButler), que declara operações genéricas; e (ii) uma hierarquia de **classes** com raiz em `BaseJPADAO`, que implementa essas operações genéricas por meio de JPA. Para cada entidade do domínio do módulo, existe uma interface DAO local (EJB `@Local`) que estende `BaseDAO<T>` e declara operações específicas; sua implementação concreta especializa `BaseJPADAO` e implementa a interface correspondente.

A Figura 9 apresenta o diagrama UML da componente de persistência do módulo, evidenciando as interfaces DAO e suas principais operações. A relação com as classes-base do JButler pode ser apresentada em um diagrama separado quando necessário, para evitar poluir o diagrama principal.

DAOs do módulo.

No módulo de Gestão de Bolsas, destacam-se os seguintes DAOs (interfaces locais), que encapsulam consultas típicas de cada conceito do domínio:

- ***ScholarshipNoticeDAO*:** provê consultas para recuperar edital por título, listar editais por status, consultar editais cujo período de inscrição intersecta um intervalo de datas e obter listagem geral de editais, suportando as operações de manutenção e consulta;
- ***ScholarshipApplicationDAO*:** oferece recuperação de inscrições por edital e status, busca textual por fragmento (e-mail ou matrícula) e consulta por edital e matrícula. Essa última é utilizada para reforçar a unicidade lógica de inscrições (evitando múltiplas inscrições ativas do mesmo aluno no mesmo edital);

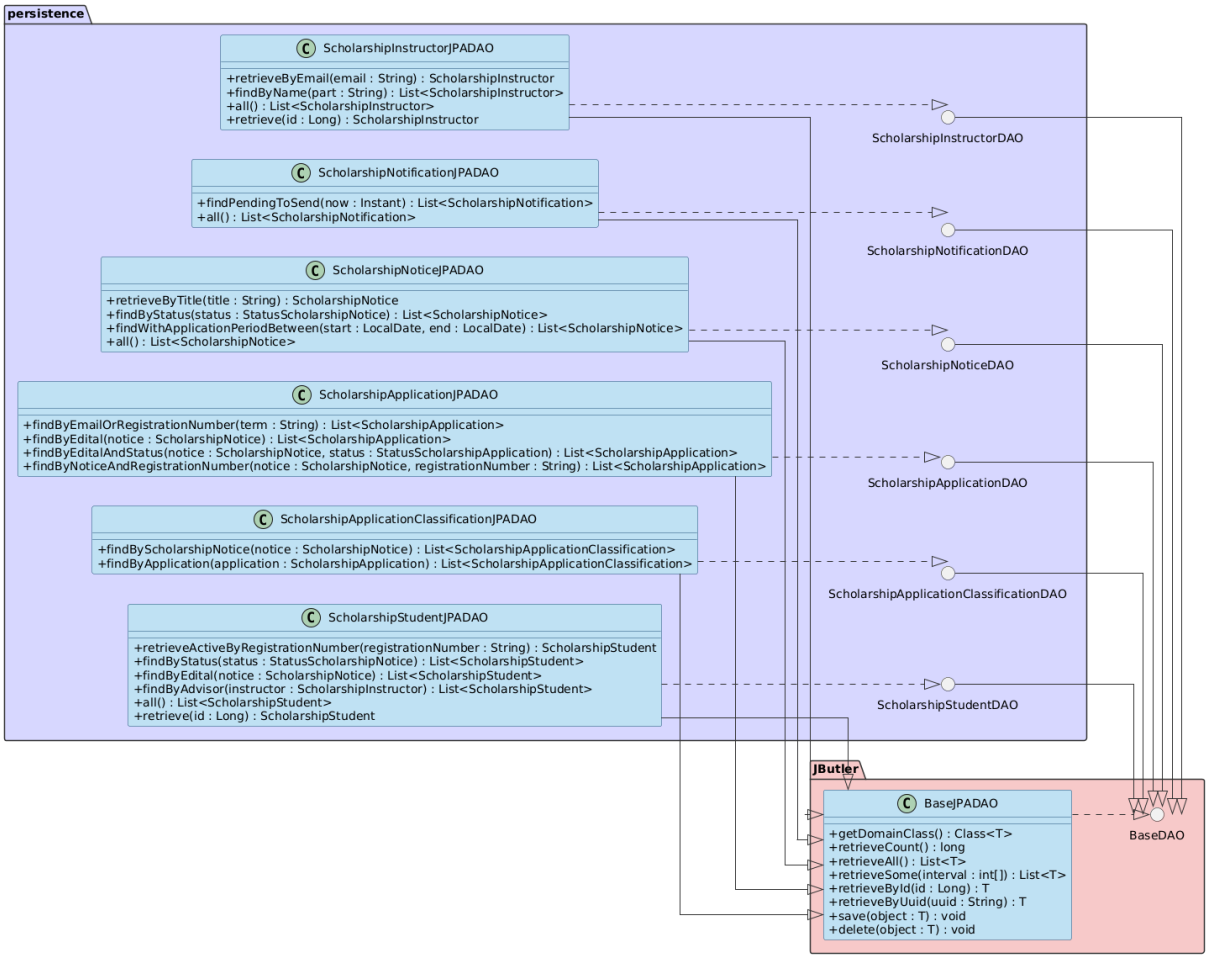

Figura 9 – Projeto da Camada de Gerência de Dados (pacote `persistence`) do módulo de Gestão de Bolsas.

- ***ScholarshipApplicationClassificationDAO*:** concentra a recuperação de resultados de classificação por edital e por inscrição, viabilizando a consulta do ranking e a persistência/reprocessamento do resultado;
- ***ScholarshipStudentDAO*:** encapsula consultas de vínculos por edital, status e orientador, além de recuperação do bolsista ativo por matrícula, apoiando verificações de consistência no ciclo de vida do vínculo de bolsa;
- ***ScholarshipInstructorDAO*:** permite recuperar orientador por e-mail e listar por fragmento do nome, apoiando a vinculação obrigatória de bolsistas a orientadores;
- ***ScholarshipNotificationDAO*:** disponibiliza consultas para recuperar notificações associadas a um edital e, quando aplicável, localizar notificações pendentes a serem enviadas até um instante de referência, mantendo rastreabilidade das comunicações do processo.

Consultas (JPQL) e uso pelas camadas superiores.

As operações específicas declaradas nas interfaces DAO são implementadas nas classes concretas por consultas JPQL (ou equivalentes via JPA), mantendo as decisões de consulta confinadas à camada `persistence`. Na camada `application`, os serviços orquestram essas operações por injeção de dependência (`@EJB`), compondo consultas e atualizações necessárias aos casos de uso (por exemplo, listagens do *overview*, consulta de resultados, criação de bolsista a partir de uma classificação e verificações de integridade como duplicidade de inscrição por matrícula).

Nota sobre os *data initializers*.

Durante a inicialização do sistema, alguns dados-base do módulo podem ser previamente carregados por componentes de inicialização (*initializers*) executados no *startup* da aplicação. Em particular, são criados exemplos/padrões de editais, orientadores e notificações, de forma a facilitar testes, demonstrações e o uso inicial do módulo. Para evitar duplicação entre reimplantações, esses inicializadores aplicam verificações de existência antes de persistir novos registros (por exemplo, buscando por título do edital, e-mail do orientador ou por tipo de notificação no contexto de um edital).

Além disso, a presença dos *initializers* se tornou especialmente relevante no contexto do requisito levantado junto ao cliente de permitir **variações de critérios de classificação por edital/agência**. Como cada edital pode demandar ajustes no cálculo do resultado, todos os dados que dependem de um edital específico precisam ser cadastrados manual e cuidadosamente (por exemplo, parâmetros e metadados do edital, conjunto de notificações associadas e textos de comunicação, e a configuração do tipo de classificação associado ao edital). Os *data initializers*, portanto, reduzem o esforço de configuração inicial e garantem um conjunto mínimo e coerente de dados para execução do módulo, sem substituir o cadastramento manual que ocorre quando um novo edital real é criado.

Como exemplo, a adaptação por edital é evidenciada pela existência de um classificador específico para FAPES, o que reforça que a classificação é dirigida por elementos associados ao edital e que, por consequência, a base de dados precisa estar corretamente configurada para cada chamada.

4.4 Implementação a partir das telas do sistema

Esta seção descreve a implementação do módulo de Gestão de Bolsas a partir da perspectiva das **telas disponibilizadas ao usuário**, guiando o leitor pelas funcionalidades que foram entregues.

4.4.1 Principais telas do módulo

A seguir, descrevem-se as telas principais e a forma como cada uma aciona controladores e serviços para realizar as funcionalidades do módulo.

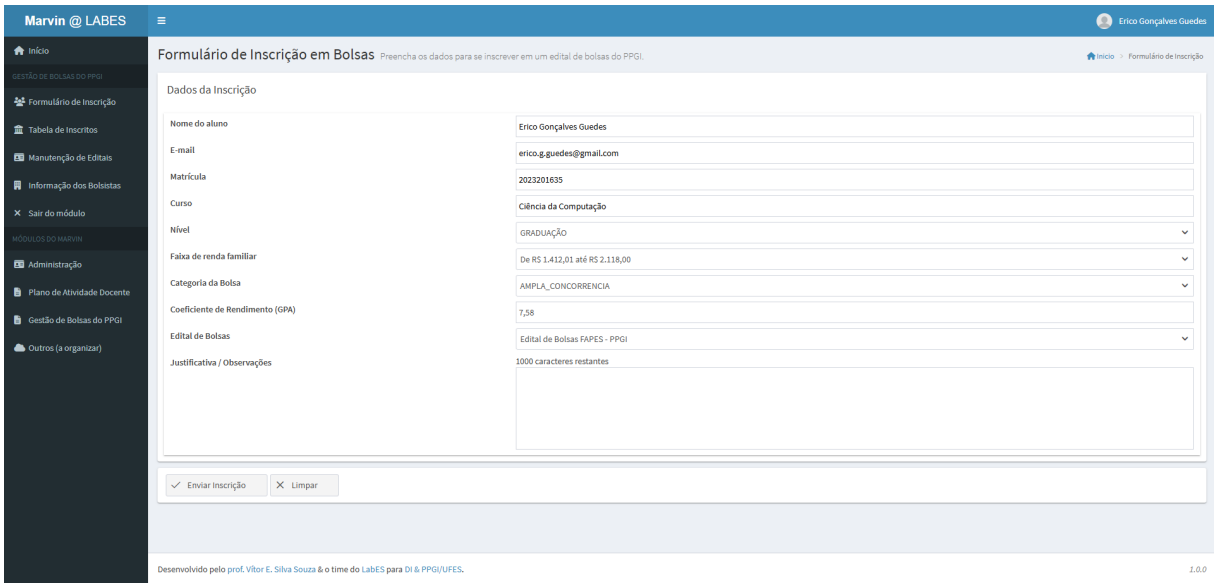

Figura 10 – Tela de formulário de inscrição.

- **Formulário de inscrição (applicationForm/index.xhtml)** (Figura 10): ao acessar a página, o *ApplicationFormController* inicializa uma nova inscrição, carrega os editais abertos para o seletor e, quando presente, interpreta o parâmetro `noticeTitle` para pré-selecionar o edital. Ao submeter o formulário, o controlador delega ao *ApplicationFormService* o registro da inscrição, que valida a elegibilidade do edital (status/período) e aplica verificações de integridade (por exemplo, duplicidade por matrícula no mesmo edital), persistindo os dados por meio dos DAOs do módulo. Em caso de sucesso, o usuário recebe feedback e o formulário é reinicializado para nova submissão.

- **Overview de editais (manageScholarshipNotice/index.xhtml)** (Figura 11): esta é a tela central de acompanhamento dos editais. Ao abrir a página, o *ManageScholarshipNoticeController* carrega a lista de editais e expõe indicadores consolidados para cada edital (por exemplo, contagem de inscrições e bolsistas). Ações de *abrir* e *encerrar* edital acionam métodos do controlador que delegam ao *ManageScholarshipNoticeService* a transição de estado e a persistência. A partir dessa tela também é iniciado o processo de classificação, precedido pela configuração de vagas (quotas) em um diálogo específico.

- **Diálogo de configuração de vagas (quotas)** (Figura 12): ao acionar "Gerar Classificação", é exibido um *pop-up* com uma tabela contendo todas as combinações

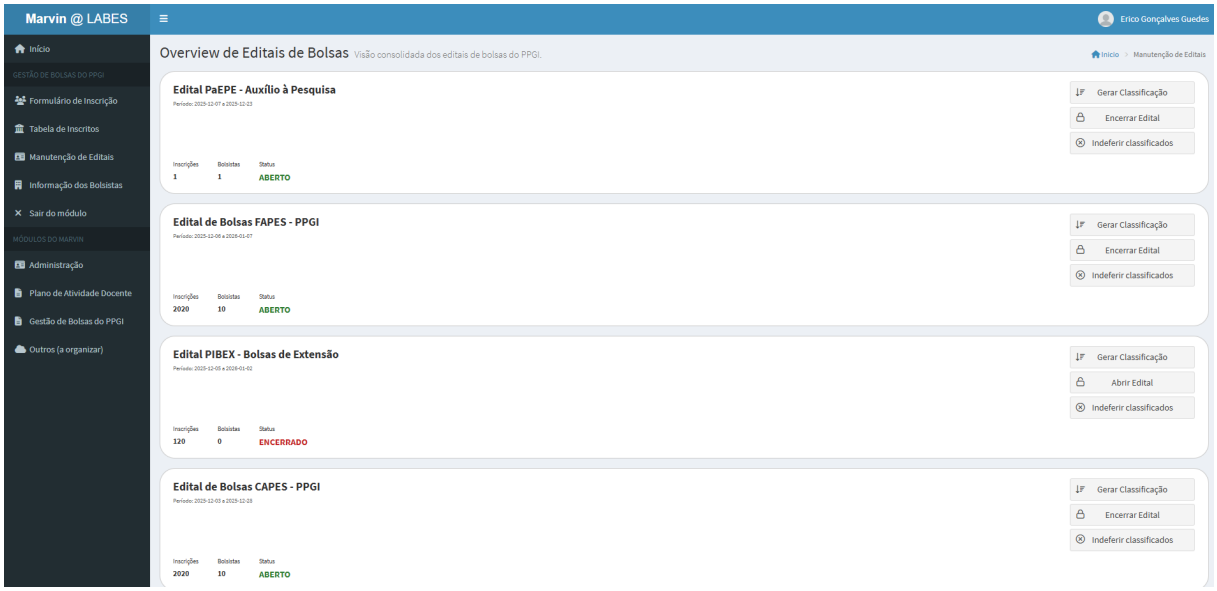

Figura 11 – Tela de overview de editais.

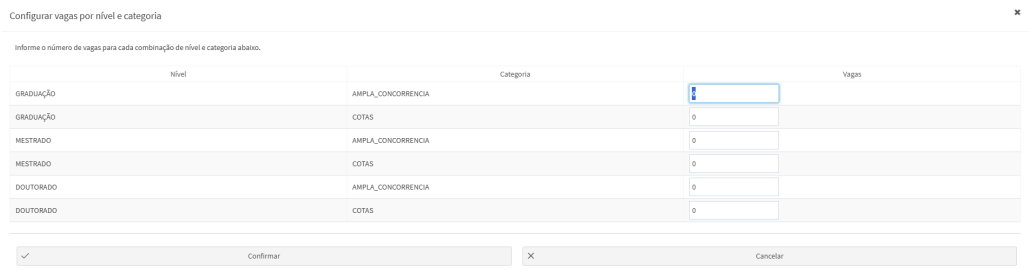

Figura 12 – Diálogo de configuração de vagas por nível e categoria.

de nível e categoria, com campos numéricos para informar o número de vagas por combinação. Os botões “Confirmar” e “Cancelar” controlam o envio (ou descarte) das configurações, permitindo aplicar quotas no processamento da classificação.

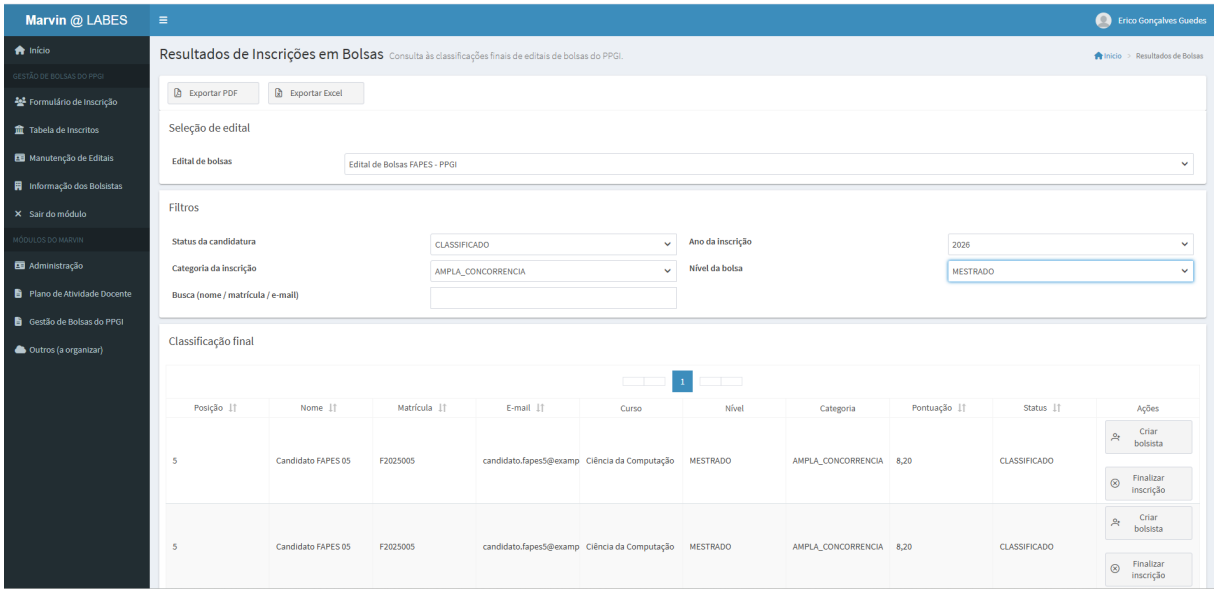

Figura 13 – Tela de resultados e classificação final.

- **Resultados/Classificação (manageScholarshipApplication/index.xhtml)** (Figura 13): a tela é utilizada pela comissão e secretaria para consultar o resultado final de um edital e executar ações operacionais. Ao abrir a página, o *ManageScholarshipApplicationController* prepara as listas auxiliares (por exemplo, títulos de editais e filtros de status/categoria/nível/ano) e exibe as classificações conforme o edital selecionado. A tabela permite aplicar filtros e realizar operações por linha: (i) *criar bolsista* a partir de um candidato aprovado e (ii) *finalizar candidatura* em caso de desistência. Essas operações são delegadas ao *ManageScholarshipApplicationService*, que atualiza os estados das entidades, persiste as alterações e, quando aplicável, aciona notificações do módulo.

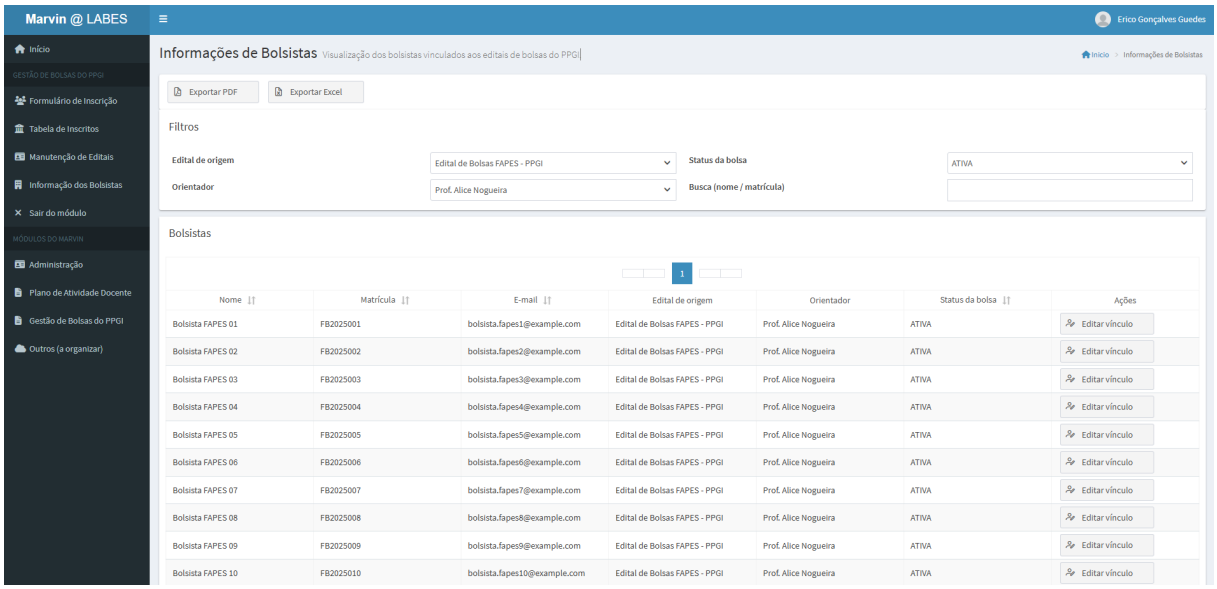

Figura 14 – Tela de listagem de bolsistas.

- **Listagem de bolsistas (manageScholarshipStudent/index.xhtml)** (Figura 14): a tela apresenta os vínculos de bolsa em forma de tabela, com filtros por edital de origem, status e orientador. O *ManageScholarshipStudentController* carrega a lista de bolsistas e os dados auxiliares de filtragem utilizando serviços do módulo e DAOs correspondentes. Ao selecionar um bolsista, o controlador navega para a tela de detalhes, incluindo o parâmetro `studentId` na URL.

- **Informações do bolsista (manageScholarshipStudentInformation/index.xhtml)** (Figura 15): ao acessar a página, o *ManageScholarshipStudentInformationController* lê o parâmetro `studentId`, carrega o bolsista via serviço e inicializa listas auxiliares (por exemplo, status e orientadores). A tela permite atualizar informações administrativas do vínculo (orientador e datas) e executar o encerramento formal por meio de um diálogo. No encerramento, o usuário deve informar motivo e data efetiva de término; o serviço responsável valida os dados, registra o encerramento e atualiza o status da bolsa para `ENCERRADA`, preservando rastreabilidade.

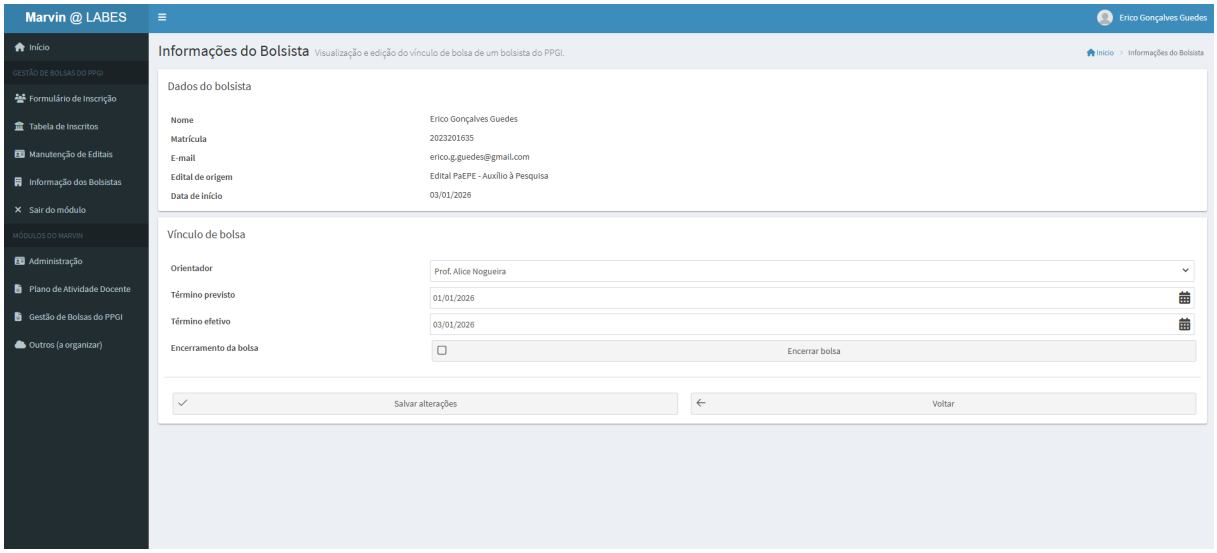

Figura 15 – Tela de informações do bolsista.

- **Diálogo de encerramento de bolsa** (Figura 16): acionado a partir da tela de informações do bolsista, permite informar o motivo de encerramento e a data efetiva de término. O serviço valida os campos e, em caso de sucesso, atualiza o status da bolsa para `ENCERRADA` e registra o motivo para fins de auditoria e rastreabilidade do vínculo.

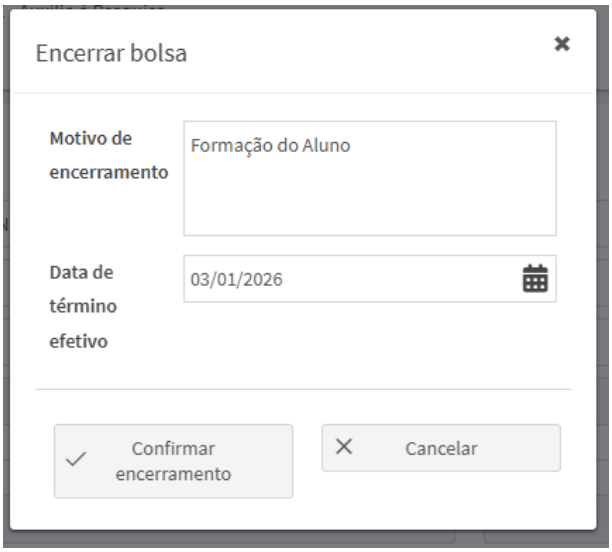

Figura 16 – Diálogo de encerramento de bolsa.

4.4.2 Funcionalidades transversais

Além das telas principais, o módulo inclui funcionalidades transversais integradas ao ecossistema Marvin. Em particular, as notificações por e-mail são disparadas pelos serviços do módulo por meio de eventos (*MailEvent*) e enviadas pelo componente *Mailer* do Marvin Core, mantendo o módulo desacoplado do mecanismo de entrega. De modo

complementar, rotinas agendadas podem acionar o serviço de notificações para envio de lembretes de relatório conforme parâmetros associados ao edital.

Essa organização baseada em telas permite compreender de forma progressiva as funcionalidades implementadas e sua interação com controladores, serviços e persistência, sem exigir do leitor um entendimento detalhado do código. Os macrofluxos completos, com passos internos de implementação, foram mantidos como material complementar no Apêndice A.

Os macrofluxos completos de implementação estão disponíveis no Apêndice A.

5 Avaliação e validação

Este capítulo descreve a estratégia de avaliação e validação planejada para o módulo de *Gestão de Bolsas do PPGI* integrado ao Marvin. O sistema foi implementado com base nos requisitos levantados junto ao cliente, estando apto à implantação em ambiente de produção. Assim, a validação descrita nesta seção possui como o objetivo assegurar a aderência aos requisitos e registrar evidências funcionais que sustentem o aceite pela secretaria e pela comissão de bolsas.

5.1 Objetivo da avaliação

A avaliação do sistema tem como objetivo principal **confirmar a aderência do módulo aos requisitos do cliente e a completude dos fluxos centrais entregues**, fornecendo evidências para a homologação e aceitação formal.

De forma mais específica, pretende-se:

- verificar se os casos de uso centrais — gestão de editais, inscrições, classificação com quotas, gestão de bolsistas, encerramento de bolsas e indeferimento em lote — podem ser executados de ponta a ponta em ambiente controlado;
- checar se **regras de negócio críticas** estão corretamente implementadas e consistentes entre camadas, incluindo: período de inscrição, transições de status de inscrição e bolsa, aplicação de quotas por nível e categoria, impedimento de duplicidades por matrícula no mesmo edital, e registro de motivo e data no encerramento do vínculo;
- confirmar se a **interface** está coerente com o restante do Marvin em termos de layout, navegação, mensagens de feedback e terminologia, reduzindo a curva de adoção;
- produzir insumos para a **homologação com o cliente**, estruturando um roteiro de testes e um checklist que possa ser aplicado em reunião, gerando evidências para ajustes finais e aceite posterior.

Portanto, a avaliação aqui descrita é predominantemente *funcional* e *de validação da entrega*: o foco é registrar evidências de que a versão entregue ao cliente está alinhada às expectativas levantadas durante o levantamento de requisitos.

5.2 Procedimento de avaliação

A avaliação foi executada na etapa final do trabalho pelo próprio autor, durante implementação do módulo em ambiente de desenvolvimento e testes do Marvin, utilizando

uma instância do WildFly e banco de dados PostgreSQL configurado para o módulo. Nesta fase, foram conduzidos testes de execução manual (orientados a tarefas) e validação técnica do fluxo, com dados de exemplo gerados por inicializadores (*Sample data initializers*) e por inserções controladas no banco.

Cenários de teste orientados a tarefas

Foram definidos cenários de teste com base nos casos de uso e nas estórias de usuário especificadas nos capítulos anteriores. Os cenários principais são:

1. **Manutenção de edital:** criação de edital, definição de período de inscrição e alternância do status (abrir/encerrar) na tela de *overview*;
2. **Inscrição de candidato:** submissão do formulário em edital aberto, preenchendo dados pessoais e acadêmicos, e verificação de validações obrigatórias;
3. **Classificação com quotas:** configuração de vagas por nível e categoria no diálogo de quotas e geração da classificação para o edital, verificando consistência do ranking e dos status resultantes;
4. **Indeferimento em lote de classificados/lista de espera:** execução da funcionalidade de indeferimento em lote, garantindo que apenas inscrições em `CLASSIFICADO` e `LISTA_ESPERA` sejam atualizadas para `INDEFERIDO`;
5. **Criação e acompanhamento de bolsistas:** criação de *ScholarshipStudent* a partir de candidatura/classificação aprovada, consulta da listagem de bolsistas e navegação para a tela de informações do vínculo;
6. **Encerramento de bolsa com motivo e data:** uso do diálogo de encerramento para informar motivo e data de término, verificando validações e atualização do status do vínculo;
7. **Notificações por e-mail (verificação técnica):** checagem do disparo de eventos de e-mail e do envio pelo componente *Mailer* do Marvin (por exemplo, aceite, encerramento e lembretes), confirmando que os templates são renderizados e que o servidor SMTP está corretamente configurado para o ambiente de testes.

Checklist de homologação (pré-entrega)

Como o sistema foi entregue ao cliente, foi elaborado um **checklist de homologação** para ser utilizado na reunião de validação com a secretaria e a comissão. O checklist organiza os itens em blocos que refletem o fluxo real de trabalho:

- *Fluxo da secretaria:* abertura/encerramento de edital, conferência de inscrições, criação/edição de bolsistas, encerramentos e consistência de dados do vínculo;
- *Fluxo da comissão:* configuração de quotas, geração de classificação, publicação de resultados, indeferimento em lote e tratamento de desistências;
- *Consistência e rastreabilidade:* transições de status e registros obrigatórios (por exemplo, motivo/data de encerramento), além de coerência entre edital, inscrição, classificação e bolsista;
- *Usabilidade e terminologia:* clareza de rótulos, mensagens e botões, facilidade de localizar funcionalidades-chave, e alinhamento da nomenclatura com a prática do PPGI.

5.3 Resultados preliminares de verificação

Por se tratar de uma etapa de homologação, os resultados a seguir devem ser entendidos como confirmação preliminar de aderência, sujeita a ajustes finos de terminologia e de preferências operacionais.

Evidências funcionais verificadas internamente

- **Coerência do fluxo de edital:** a alternância de status `ABERTO`/`ENCERRADO` foi validada em interface, e editais encerrados não são listados como opção de inscrição no formulário;
- **Submissão de inscrições com validações:** o formulário aplica validações de obrigatoriedade e impede submissões com dados incompletos, preservando integridade mínima do cadastro;
- **Classificação com quotas:** o diálogo de configuração de vagas permite parametrizar o processo por combinação nível–categoria, e o resultado persiste score/posição e status de classificação para consulta posterior;
- **Operações administrativas sobre resultados:** funcionalidades como indeferimento em lote e ações relacionadas à transição de estados foram verificadas em ambiente de teste, visando manter consistência do processo quando um resultado precisa ser descartado e publicado novamente;
- **Gestão do vínculo do bolsista:** a edição de orientador/datas e o encerramento com motivo e data efetiva exigidos pelo domínio foram exercitados, garantindo rastreabilidade;

- **Notificações:** foi verificado o disparo do evento de e-mail e a integração com o *Mailer* do Marvin em ambiente controlado, com registro em *logs* e validação de configuração SMTP.

Pendências esperadas para homologação

Alguns aspectos dependem diretamente da validação do cliente e devem ser confirmados na entrega:

- **Ajustes de terminologia e políticas de operação:** nomes de botões, mensagens e regras de exceção (por exemplo, que status são considerados "ativos" para bloqueio de duplicidade) podem exigir alinhamento fino com a prática do PPGI;
- **Relatórios e formatos de publicação:** embora existam exportações e listagens operacionais, a secretaria e a comissão podem solicitar modelos específicos de relatório e layouts padronizados para divulgação de resultados;
- **Parâmetros e exceções de editais por agência:** detalhes de regras de pontuação, desempate e pesos podem variar entre chamadas e devem ser confirmados pelo cliente ao comparar com editais reais recentes.

6 Conclusão

Este capítulo apresenta as considerações finais sobre o desenvolvimento do módulo de *Gestão de Bolsas do PPGI* integrado ao Marvin, bem como oportunidades de evolução identificadas ao longo do trabalho. Retomam-se os objetivos definidos na introdução, discutem-se as contribuições alcançadas e são sugeridas direções para trabalhos futuros.

6.1 Considerações finais

O ponto de partida deste trabalho foi a constatação de que a gestão de bolsas no Programa de Pós-Graduação em Informática (PPGI) da UFES vinha sendo conduzida, em grande parte, por meios ineficientes, trocas de e-mails e documentos dispersos. Editais, inscrições, classificações e controle de bolsistas eram processos fortemente dependentes de trabalho manual, o que dificultava a rastreabilidade das decisões, a transparência para candidatos e a manutenção de um histórico organizado ao longo dos anos.

Diante desse contexto, o objetivo geral estabelecido consistiu em **conceber, projetar e implementar um módulo de software para gestão de bolsas, integrado ao Marvin**, capaz de apoiar o ciclo de vida completo de editais do PPGI: desde a publicação de chamadas, passando pela inscrição de candidatos, classificação com regras de quotas, geração de bolsistas e acompanhamento do vínculo, até o encerramento da bolsa.

Ao longo dos capítulos anteriores, esse objetivo foi desdobrado em objetivos específicos, que podem ser sintetizados nas seguintes contribuições principais:

- **Levantamento e especificação de requisitos:** foram identificados e documentados os atores envolvidos (secretaria, comissão de bolsas, equipe de TI, e candidatos), suas necessidades e restrições de negócio, incluindo regras sobre períodos de inscrição, limites de vagas, situações de conflito entre editais e rastreabilidade de decisões de concessão e encerramento de bolsas;
- **Arquitetura em camadas e integração ao Marvin:** o sistema foi projetado segundo a arquitetura em camadas já adotada pelo Marvin, utilizando Java/Jakarta EE, WildFly, JPA e PrimeFaces. Isso permitiu reutilizar *infraestrutura* existente (autenticação, padrão de DAOs, templates de interface, serviço de e-mail) e manter coerência com os demais módulos;
- **Implementação dos casos de uso centrais:** foram implementados casos de uso que cobrem o fluxo operacional da secretaria e da comissão, tais como: criação manual e manutenção de editais; formulário de inscrição de candidatos; classificação com quotas nível–categoria; criação e gerenciamento de bolsistas; encerramento de

bolsas com registro de motivo e data; e indeferimento em lote de candidatos após homologação;

- **Mecanismo de classificação com quotas:** foi desenvolvida uma solução flexível para classificação de candidatos que separa o cálculo de *score* (por meio de classificadores específicos de agência) da aplicação de quotas por combinação de nível e categoria. A parametrização das vagas é realizada em interface amigável, permitindo ajustar o número de bolsas sem alterar o código;
- **Base para evolução futura:** o projeto de dados, a arquitetura e os serviços foram concebidos de forma extensível, de modo a facilitar futuros incrementos, integrações com outros sistemas da UFES e a inclusão de funcionalidades de análise e relatórios.

Os objetivos definidos na Seção 1.2 foram atendidos de forma coerente com o escopo deste trabalho. Em particular, o **objetivo geral** — conceber, projetar e implementar um módulo de software para gestão de bolsas, integrado ao Marvin — é sustentado pela especificação de requisitos (Capítulo 3), pelo projeto e decisões arquiteturais (Capítulo 4) e pelas evidências de verificação e discussão de limitações e próximos passos (Capítulo 5).

O **OE1** (levantar, analisar e documentar requisitos) foi atendido e tem como evidência o Capítulo 3, no qual são apresentados os atores e perfis envolvidos, os requisitos funcionais e não funcionais, as regras de negócio e as estórias de usuário com critérios de aceitação (Seções 3.2 a 3.6). O **OE2** (projetar e documentar a arquitetura do módulo integrada ao Marvin) também foi atendido, com evidências no Capítulo 4, que descreve a arquitetura em camadas e a organização do módulo e suas responsabilidades, bem como na seção de subsistemas e interdependências, onde se explicita a relação do módulo de Bolsas com o Marvin Core, com o módulo Admin e com serviços transversais (Seção 3.8; Capítulo 4). Por fim, o **OE3** (implementar e testar o código-fonte do módulo) foi atendido no escopo desta monografia: a implementação das funcionalidades centrais é evidenciada no Capítulo 4, ao descrever as telas do sistema e as operações suportadas para edital, inscrição, classificação com quotas, criação e acompanhamento de bolsistas, encerramento e notificações; a verificação interna das funcionalidades é apresentada no Capítulo 5, por meio do procedimento adotado e dos resultados observados nos cenários de teste. Limitações remanescentes e trabalhos futuros (incluindo integrações institucionais adicionais e evolução de relatórios) são discutidos no Capítulo 5, reforçando o atendimento aos objetivos com clareza sobre o escopo atingido.

Do ponto de vista acadêmico, o trabalho contribui para o estudo de projetos de sistemas *web* corporativos em contexto universitário, mostrando como práticas de engenharia de software (levantamento de requisitos, modelagem de domínio, arquitetura em camadas, uso de frameworks e padrões) podem ser aplicadas a problemas reais da gestão acadêmica. Do ponto de vista prático, o módulo oferece à secretaria e à coordenação

do PPGI uma ferramenta integrada ao ambiente já conhecido (Marvin), reduzindo esforços manuais e aumentando a consistência dos dados de bolsas.

Naturalmente, algumas limitações permanecem. Determinadas regras de negócio mais específicas de cada agência de fomento ainda precisam ser refinadas e parametrizadas de forma mais declarativa. Da mesma forma, recursos avançados de relatórios e visualização de indicadores foram apenas delineados, não tendo sido implementados em toda sua extensão. Essas limitações, contudo, abrem espaço para uma agenda clara de trabalhos futuros.

6.2 Trabalhos futuros

O módulo de Gestão de Bolsas do PPGI foi concebido como um primeiro passo em direção a uma solução mais ampla para acompanhamento de bolsas ao longo de todo o ciclo de vida de estudantes e egressos do programa. A seguir são elencadas algumas possibilidades de evolução, tanto do ponto de vista funcional quanto tecnológico.

Egressos e trajetória dos bolsistas

Uma linha natural de continuidade consiste em estender o módulo para acompanhar o percurso de **egressos** que foram bolsistas do PPGI. Isso envolve:

- criação de uma entidade de *egresso* com vínculo aos registros de *ScholarshipStudent*;
- registro de informações sobre atuação profissional, prosseguimento para doutorado ou pós-doutorado, participação em projetos de pesquisa e produção científica após a conclusão do curso;
- mecanismos para coleta periódica de dados (por exemplo, via formulários eletrônicos ou integração com plataformas de currículos) que alimentem indicadores de impacto das bolsas na trajetória dos egressos.

Essa extensão permitiria que o programa respondesse de forma mais estruturada a demandas de avaliação externa e prestação de contas sobre os resultados de suas políticas de fomento.

Integrações com sistemas institucionais

Outra direção de evolução é aprofundar a **integração com sistemas institucionais da UFES** e de agências de fomento. Entre as possibilidades:

- integração com sistemas acadêmicos para importar e validar automaticamente dados de matrícula, histórico escolar e situação de vínculo dos alunos, reduzindo retrabalho e inconsistências;
- integração com sistemas de recursos humanos e financeiros responsáveis pelo pagamento de bolsas, de modo que o status da bolsa no Marvin reflita em tempo quase real a situação de pagamento;
- exportações padronizadas para plataformas das agências de fomento, seguindo modelos de planilhas ou arquivos exigidos em prestações de contas e relatórios anuais.

Essas integrações fortaleceriam o papel do módulo como fonte única de informação sobre bolsas, ao mesmo tempo em que diminuiriam o esforço manual da secretaria na consolidação de dados dispersos.

Dashboards e análise de dados

A consolidação de dados de editais, inscrições, classificações e vínculos de bolsa cria um terreno fértil para o desenvolvimento de **dashboards e painéis analíticos**. Trabalhos futuros podem explorar:

- construção de indicadores como número de bolsas por ano, agência de fomento, linha de pesquisa, nível e categoria;
- análise de perfil socioeconômico dos bolsistas, distribuição de faixas de renda, taxas de renovação e evasão;
- integração com ferramentas de *business intelligence* (por exemplo, Power BI ou soluções em nuvem) para permitir que coordenação e comissão explorem os dados de forma interativa.

Além de apoiar a gestão interna, tais painéis podem subsidiar decisões estratégicas sobre distribuição de bolsas, definição de critérios de classificação e priorização de linhas de pesquisa.

Automações avançadas e melhorias de processo

Do ponto de vista de automação de processos, há uma série de melhorias possíveis:

- agendamento de **tarefas automáticas** (por exemplo, envio de lembretes de entrega de relatório, avisos de término de bolsa, alertas sobre atraso na entrega de documentos);

- implementação de **fluxos de aprovação** com múltiplas etapas (secretaria, comissão, coordenação), incluindo registro explícito das decisões de cada etapa e comentários;
- aprimoramento da **configuração de regras de classificação**, tornando-as mais declarativas e parametrizáveis, de modo que mudanças nos editais de agências possam ser refletidas com menos intervenção no código;
- refinamento de aspectos de **usabilidade** e acessibilidade das telas, com base em testes com usuários reais e métricas de uso.

Generalização do módulo para outros programas

Por fim, uma linha de trabalho de médio prazo envolve a **generalização do módulo** para que possa ser utilizado por outros programas de pós-graduação da UFES ou mesmo por unidades acadêmicas com processos de bolsas similares. Isso exigiria:

- parametrização de elementos específicos do PPGI (estrutura de linhas de pesquisa, tipos de bolsas, regras de relatórios);
- mecanismos de multi-*tenant* ou multi-programa, permitindo que um único módulo atenda a múltiplos cursos, mantendo separação de dados;
- documentação e *onboarding* adequados para equipes de outros programas que desejem adotar o módulo.

Em síntese, o trabalho realizou o primeiro ciclo de implantação de um módulo integrado de Gestão de Bolsas no Marvin, cobrindo o núcleo de funcionalidades necessárias à realidade atual do PPGI. Ao mesmo tempo, deixou mapeado um conjunto robusto de extensões e refinamentos que podem ser explorados em projetos futuros, tanto de pesquisa quanto de desenvolvimento, consolidando o módulo como um ativo estratégico para a gestão acadêmica e para a avaliação da pós-graduação na instituição.

Referências

ARAÚJO, R. *Arquitetura em Três Camadas para aplicações web*. 2023. Blog Gran Cursos Online. Publicado em 22 fev. 2023. Available on: <https://blog.grancursosonline.com.br/arquitetura-em-tres-camadas-para-aplicacoes-web/>. Citado na página 26.

BADALOTTI, G. M. *Introdução ao Desenvolvimento de Sistemas Web*. Indaial: UNIASSELVI, 2018. Citado na página 22.

BARCELLOS, M. P. *Análise e Projeto de Sistemas de Software*. 2. ed. Vitória: EDUFES, 2018. Citado 3 vezes nas páginas 19, 20 e 21.

CHANE, V. A. *Macs – Sistema de controle de acessos do Marvin*. Monografia (Monografia (PG)) — Universidade Federal do Espírito Santo (UFES), Vitória, ES, 2022. Curso de Ciência da Computação. Orientador: Prof. Dr. Vitor Estevão Silva Souza. Citado na página 14.

CONTE, T.; MENDES, E.; TRAVASSOS, G. H. Processos de desenvolvimento para aplicações web: Uma revisão sistemática. In: *Anais do Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)*. Brasil: [s.n.], 2005. Citado na página 22.

CRISPIN, L.; GREGORY, J. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Boston, MA: Addison-Wesley, 2009. Citado na página 21.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. *Introdução ao Teste de Software*. Rio de Janeiro: Editora Campus / SBC, 2007. Citado 2 vezes nas páginas 21 e 22.

FALBO, R. d. A. *Notas de Aula – Engenharia de Software*. 2018. <http://www.inf.ufes.br/~falbo>. Universidade Federal do Espírito Santo. Acesso em: 16 jul. 2025. Citado 2 vezes nas páginas 21 e 34.

FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. New York: John Wiley & Sons, 1999. Citado na página 24.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston, MA: Addison-Wesley, 2002. Citado na página 27.

ISO/IEC. *ISO/IEC 25010:2011 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)*. 2011. International Organization for Standardization. Modelo de qualidade de software (SQuaRE). Citado na página 13.

ISO/IEC. *ISO/IEC 27002:2022 — Information security, cybersecurity and privacy protection — Information security controls*. 2022. Controles de segurança como logging, monitoramento e controle de acesso. Citado na página 13.

ISO/IEC. *ISO/IEC 27002:2022 — Information security, cybersecurity and privacy protection — Information security controls*. 2022. Controles de acesso, registro e monitoramento centralizados para aumentar rastreabilidade e proteção de dados. Citado na página 28.

LAUDON, K. C.; LAUDON, J. P. *Sistemas de Informação Gerenciais*. 11. ed. São Paulo: Pearson, 2014. Citado na página 13.

MALL, R. *Fundamentals of Software Engineering*. 5. ed. New Delhi: PHI Learning, 2020. Citado na página 18.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The Art of Software Testing*. 3. ed. Hoboken, NJ: Wiley, 2011. Citado na página 22.

OLIVEIRA, B. G. N. d. *Módulo de controle acadêmico da Pós-Graduação do Marvin*. Monografia (Monografia (PG)) — Universidade Federal do Espírito Santo (UFES), Vitória, ES, 2023. Curso de Engenharia de Computação. Orientador: Prof. Dr. Vítor E. Silva Souza. Citado na página 14.

PFLEEGER, S. L. *Software Engineering: Theory and Practice*. 2. ed. Upper Saddle River, NJ: Prentice Hall, 2004. Citado na página 19.

PPGI/UFES. *Programa de Pós-Graduação em Informática – UFES*. <https://informatica.ufes.br/pt-br/pos-graduacao/PPGI>. Acesso em: 22 fev. 2026. Citado na página 13.

PRESSMAN, R. S.; LOWE, D. *Web Engineering: A Practitioner's Approach*. New York: McGraw-Hill, 2008. Citado 2 vezes nas páginas 22 e 25.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 8. ed. Porto Alegre: AMGH Editora, 2016. Citado 4 vezes nas páginas 18, 20, 21 e 27.

ROBERTSON, S.; ROBERTSON, J. *Mastering the Requirements Process*. 2. ed. Boston: Addison-Wesley, 2006. Citado na página 19.

SHKLAR, L.; ROSEN, R. *Web Application Architecture: Principles, Protocols and Practices*. 3. ed. Chichester: Wiley, 2009. Citado 2 vezes nas páginas 25 e 26.

SOMMERVILLE, I. *Engenharia de Software*. 8. ed. São Paulo: Pearson Addison Wesley, 2007. Citado 3 vezes nas páginas 19, 24 e 27.

SOUZA, V. E. S. *Documento de Projeto de Sistema: Marvin: núcleo*. Vitória, ES, 2022. Documento interno de projeto (design) do módulo central do Marvin. Citado 3 vezes nas páginas 14, 23 e 24.

SWACHA, J.; KULPA, A. Evolution of popularity and multiaspectual comparison of widely used web development frameworks. *Electronics*, 2023. Artigo sobre comparação de frameworks Web. Citado na página 24.

TURBAN, E.; VOLONINO, L. *Tecnologia da Informação para Gestão*. 9. ed. Porto Alegre: Bookman Editora, 2015. Citado 2 vezes nas páginas 27 e 28.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Independente, 2020. Citado 2 vezes nas páginas 26 e 27.

Apêndices

APÊNDICE A – Macrofluxos de implementação

A seguir são descritos os macrofluxos de implementação dos principais casos de uso do módulo de Gestão de Bolsas.

A.1 Gestão de editais

A gestão de editais é centralizada no serviço *ManageScholarshipNoticeService*, acionado pelo controlador *ManageScholarshipNoticeController* a partir da tela de *overview* de editais. A implementação segue a separação em camadas do Marvin: o controlador mantém o estado da visão e delega ao serviço as operações transacionais e o acesso aos DAOs.

- **Carregamento e apresentação no overview:** ao inicializar a tela (`@PostConstruct`), o controlador recupera todos os editais por meio de `listAllNotices()`, que delega ao `ScholarshipNoticeDAO` a listagem completa. A interface apresenta cada edital em formato de *card*, exibindo título, período de inscrição e status atual, além de indicadores consolidados;
- **Métricas por edital (inscrições e bolsistas):** o *overview* exibe, para cada edital, o número total de inscrições e de bolsistas vinculados. No serviço, a contagem de inscrições é obtida somando as quantidades por status, iterando sobre todos os valores de `StatusScholarshipApplication` e consultando o `ScholarshipApplicationDAO` por `(edital, status)`. A contagem de bolsistas é obtida consultando o `Scholarship-StudentDAO` por edital e computando o tamanho da lista retornada;
- **Abertura e encerramento do edital:** a mudança de estado do edital é acionada por botões na tela e executada pelos métodos `openNotice()` e `closeNotice()`. O serviço aplica a transição para `ABERTO` ou `ENCERRADO` e persiste a alteração via `ScholarshipNoticeDAO.save()`. Na camada de apresentação, o controlador exibe mensagens de sucesso após a operação, mantendo feedback imediato ao usuário;
- **Operações administrativas relacionadas ao resultado:** além da manutenção do status do edital, a tela de *overview* agrega ações típicas do fluxo de seleção, como disparar a etapa de classificação (com configuração de vagas) e permitir indeferimento em lote de candidatos já classificados/lista de espera quando for necessário publicar

um novo resultado. Essas operações são orquestradas pelo serviço de aplicação e refletem necessidades práticas da comissão/secretaria no ciclo de vida do edital.

Em conjunto, essas funcionalidades tornam a tela de *overview* o ponto central de acompanhamento do processo: ela reúne os editais ativos e encerrados, oferece indicadores consolidados e disponibiliza ações de controle do período de inscrição e de manutenção do resultado, mantendo o fluxo aderente ao padrão arquitetural do Marvin.

A.2 Inscrições em edital

O caso de uso de inscrição é suportado pelo *ApplicationFormController*, que mantém o estado da tela de formulário e expõe uma instância de *ScholarshipApplication* para ligação direta com os campos do *XHTML*. A lógica de negócio e as validações são delegadas ao serviço *ApplicationFormService* (implementado por um EJB `@Stateless`), mantendo a separação de responsabilidades adotada no Marvin.

- **Inicialização da tela e carregamento de editais abertos:** ao abrir o formulário, o controlador instancia um novo *ScholarshipApplication*, inicializa a `applicationDate` com a data corrente e carrega a lista de editais disponíveis por meio de `findOpenNotices()`. No serviço, essa consulta retorna apenas editais cujo status é `ABERTO`, garantindo que o aluno visualize somente opções elegíveis para inscrição;
- **Seleção do edital por título:** a página trabalha com um seletor cujo *value* é uma *String* (`selectedNoticeTitle`). Antes de registrar a inscrição, o controlador resolve esse título para um objeto *ScholarshipNotice* válido (a partir da lista de editais abertos) e associa o edital ao objeto *ScholarshipApplication* que está sendo preenchido. Além disso, a inicialização da tela suporta pré-seleção do edital via parâmetro de requisição (`noticeTitle`), o que facilita navegação a partir de outras telas do módulo;
- **Preenchimento de dados pelo candidato:** o aluno informa dados pessoais e acadêmicos (nome, e-mail, matrícula, curso, nível, faixa de renda, categoria, coeficiente de rendimento e justificativa). As restrições de obrigatoriedade e limites de tamanho são reforçadas tanto na camada de interface (componentes com `requiredMessage`) quanto pelas anotações de validação do domínio;
- **Validações no serviço (regras de integridade e elegibilidade):** ao submeter o formulário, o controlador invoca `registerApplication(application)`. No serviço, são aplicadas validações fundamentais:
  - a inscrição não pode ser nula e deve estar associada a um edital válido e persistido (com identificador);

  - o período de inscrição é verificado a partir de `applicationStartDate` e `applicationEndDate` do edital, evitando submissões fora da janela temporal definida;
  - a matrícula é normalizada e é realizada uma busca por inscrições existentes no mesmo edital. Caso já exista outra inscrição do aluno em *status ativo* (por exemplo, `INSCRITO`, `CLASSIFICADO`, `LISTA_ESPERA` ou `BOLSISTA`), o serviço bloqueia a operação para evitar duplicidade.

- **Persistência e padronização do estado inicial:** se a inscrição for válida, o serviço define valores padrão quando necessário (por exemplo, `status = INSCRITO` e `applicationDate = today` caso não tenham sido informados) e persiste a entidade *ScholarshipApplication* via `ScholarshipApplicationDAO.save()`;

- **Feedback ao usuário e reinicialização do formulário:** após o registro, o controlador exibe mensagens de sucesso e reinicia o formulário com uma nova instância de *ScholarshipApplication*, recarregando também a lista de editais abertos para manter a tela pronta para uma nova inscrição.

Dessa forma, o fluxo de inscrição garante consistência com o estado do edital, aplica regras mínimas de integridade (especialmente contra duplicidade por matrícula) e mantém uma experiência de uso simples e alinhada ao padrão arquitetural do Marvin, com a camada de apresentação focada em estado de tela e o serviço concentrando as validações e persistência.

A.3 Gestão de inscrições

A gestão de inscrições concentra as funcionalidades voltadas à comissão e à secretaria para acompanhar o resultado do processo seletivo, aplicar filtros, tomar decisões operacionais (por exemplo, converter um classificado em bolsista ou registrar desistência) e exportar listagens. Esse conjunto é implementado pela combinação do controlador *ManageScholarshipApplicationController*, da interface/serviço *ManageScholarshipApplicationService* e dos DAOs de inscrições/classificações.

- **Carregamento inicial e seleção de edital:** ao abrir a página de resultados, o controlador carrega os títulos de editais disponíveis por meio de `listNoticeTitlesForResults()` e, em seguida, recupera as classificações existentes usando `listAllClassificationsForResults()`. Quando um edital é selecionado, a lista é atualizada chamando `listFinalClassificationsByNoticeTitle(title)`. Internamente, o serviço resolve o edital pelo título (via DAO) e consulta as classificações associadas ao edital, retornando-as já ordenadas para apresentação no *dataTable*.

- **Filtros e busca na tabela de resultados:** sobre a lista base de classificações, a camada de apresentação aplica filtros por *status da candidatura*, *ano da inscrição*, *categoria*, *nível* e *busca textual* (nome/matrícula/e-mail). O filtro de ano é derivado da data de inscrição (`applicationDate`), permitindo análise histórica e segmentação por edição do processo. Essa estratégia melhora a responsividade da tela, pois evita refazer consultas complexas a cada interação, mantendo o recorte final consistente com a seleção do edital.

- **Conversão de inscrição em bolsista (criação do vínculo):** para candidatos aprovados, a tabela disponibiliza a ação "Criar bolsista", que delega ao serviço `createStudentFromClassification(classification)`. O serviço cria um *ScholarshipStudent* copiando dados relevantes da *ScholarshipApplication*, associa o edital de origem e define valores iniciais do vínculo (por exemplo, `startDate` e `expectedEndDate`), além de associar um orientador quando necessário. Após persistir o bolsista, o serviço atualiza o status da inscrição para `BOLSISTA`, impedindo reutilização indevida da mesma inscrição em múltiplos vínculos. Também é disparada uma notificação de aceite utilizando a infraestrutura de e-mail do Marvin.

- **Finalização de inscrição (desistência):** quando um candidato não prossegue com a bolsa, a ação "Finalizar inscrição" aplica a regra de atualização de status, delegando ao serviço `finishApplication(classification)`. Nesta implementação, o serviço usa uma sobrecarga conveniente que marca a candidatura como `DESISTENTE`, registrando que aquela inscrição não deve mais ser considerada ativa no fluxo.

- **Tratamento de erros e feedback ao usuário:** as operações críticas (criar bolsista e finalizar inscrição) são encapsuladas em *try/catch* no controlador, exibindo mensagens de sucesso em caso de conclusão e mensagens de erro quando há falhas de validação ou exceções inesperadas. Após operações bem-sucedidas, a listagem é recarregada para refletir o novo estado das candidaturas.

- **Exportação dos resultados:** a página de resultados foi projetada para suportar exportações do conteúdo filtrado em formatos amplamente utilizados pela secretaria e pela comissão. A exportação é acionada a partir de botões no topo da tela e utiliza o mecanismo de exportação do *dataTable*, produzindo arquivos em PDF e planilha (Excel), coerentes com o recorte aplicado pelos filtros.

Em síntese, a gestão de inscrições materializa a etapa operacional do processo seletivo após a classificação: ela permite localizar candidatos por edital e critérios, registrar decisões (concessão ou desistência) e produzir listagens para publicação e tramitação administrativa, mantendo o estado das inscrições sincronizado com o ciclo de vida do vínculo de bolsa.

A.4 Classificação com quotas nível × categoria

A classificação com quotas utiliza um diálogo específico, acionado na tela de *overview* de editais, e é executada pelo serviço *ScholarshipApplicationClassificationService*. O fluxo foi projetado para separar claramente: (i) configuração de vagas na camada de apresentação, (ii) cálculo de pontuação e ordenação em um componente de classificação, e (iii) aplicação das quotas e persistência do resultado.

- **Configuração de vagas via *ScholarshipQuotaConfig*:** ao clicar em "Gerar Classificação", o controlador *ManageScholarshipNoticeController* prepara uma lista de configurações *nível–categoria* por meio do *value object ScholarshipQuotaConfig*. Cada instância armazena: `level` (nível), `category` (categoria) e `slots` (número de vagas), com *bridges* `getVacancies()/setVacancies()` para compatibilidade com o código da *view*. O diálogo exibe essa lista em uma tabela com campo numérico, exigindo que o usuário informe um número inteiro ≥ 0 por combinação;
- **Gatilho do caso de uso e validação de entrada:** ao confirmar, o controlador delega a operação ao *ManageScholarshipNoticeService*, que valida o edital selecionado e a presença de uma configuração de quotas. Em seguida, encaminha ao serviço de classificação o edital e a lista de *ScholarshipQuotaConfig* preenchida pelo usuário;
- **Recuperação de inscrições elegíveis:** o serviço de classificação recupera as inscrições do edital no status `INSCRITO`, garantindo que a classificação reprocessa apenas candidatos efetivamente concorrentes e evita reavaliar inscrições já finalizadas, indeferidas ou convertidas em bolsista;
- **Seleção da estratégia de classificação (*ScholarshipClassifier*):** a pontuação e ordenação são delegadas a uma implementação de *ScholarshipClassifier*, escolhida dinamicamente de acordo com o tipo do edital. Isso permite encapsular regras específicas por agência (por exemplo, CAPES e CNPq) sem comprometer a coesão da camada de serviço;
- **Construção, pontuação e ranqueamento:** para cada inscrição elegível, o serviço cria um *ScholarshipApplicationClassification* associado ao edital e à inscrição. O classificador calcula a pontuação (`score`) e ordena os candidatos. A posição (`position`) é atribuída de forma determinística dentro do contexto do resultado, permitindo rastreabilidade do ranking e auditoria das decisões;
- **Aplicação das quotas nível–categoria:** após ordenar, o serviço aplica as quotas informadas. Para isso, a lista de *ScholarshipQuotaConfig* é convertida em um mapeamento interno (por exemplo, chave `level#category` → vagas). Em seguida, percorre-se o ranking e, para cada candidato, contabiliza-se quantos já foram aceitos no grupo correspondente:

  - se o limite de vagas do grupo for 0, candidatos daquele grupo são marcados como `INDEFERIDO`;
  - se houver vagas, os primeiros até o limite são marcados como `CLASSIFICADO` e os excedentes como `LISTA_ESPERA`.

  Esse pós-processamento garante que o resultado final respeite a distribuição de vagas por combinação *nível–categoria*, conforme parametrizado pela comissão/coordenação;

- **Persistência do resultado consolidado:** por fim, os registros de *ScholarshipApplicationClassification* são persistidos via *DAO* e os status associados às inscrições são atualizados no mesmo contexto transacional, permitindo que a comissão consulte o resultado consolidado posteriormente e que a tela de resultados reflita o estado atualizado do processo seletivo.

Em síntese, o fluxo implementado permite que a comissão/coordenação configure vagas por combinação *nível–categoria* via *ScholarshipQuotaConfig*, gere o ranking com regras específicas por edital e publique um resultado coerente com as quotas informadas, mantendo rastreabilidade por meio dos registros de classificação persistidos.

A.5 Gestão de bolsistas

A gestão de bolsistas é implementada pela combinação de uma tela de listagem e uma tela de detalhes, apoiadas por serviços de aplicação responsáveis por manter a consistência do vínculo de bolsa. Em termos arquiteturais, a camada de apresentação utiliza controladores JSF para filtrar e navegar entre telas, enquanto a camada de aplicação concentra operações como criação de bolsista, edição do vínculo e encerramento com registro de motivo e data.

- **Listagem e filtragem de bolsistas:** a tela de listagem expõe um conjunto de filtros (por edital de origem, status da bolsa, orientador e busca textual por nome/matrícula), permitindo que a secretaria e a coordenação encontrem rapidamente vínculos relevantes. O controlador responsável atualiza a lista exibida a cada alteração de filtro, mantendo paginação e ordenação na interface;

- **Criação de bolsista a partir da classificação:** quando um candidato é aprovado, o serviço de gestão cria um *ScholarshipStudent* com base nos dados da inscrição, copiando atributos essenciais (nome, e-mail, matrícula, curso e nível), vinculando o bolsista ao *ScholarshipNotice* de origem e definindo o estado inicial da bolsa como `ATIVA`. As datas são inicializadas com a data corrente como início e um término previsto calculado a partir de uma regra padrão do módulo (por exemplo, horizonte de

18 meses). Além disso, um orientador disponível (*ScholarshipInstructor*) é associado ao bolsista para satisfazer a obrigatoriedade do vínculo no domínio; por fim, o status da inscrição correspondente é atualizado para `BOLSISTA`, evitando que a mesma candidatura seja reutilizada indevidamente para criação de novos vínculos;

- **Tela de detalhes e edição do vínculo:** ao selecionar um bolsista na lista, o sistema navega para uma tela de informações detalhadas, na qual dados básicos (nome, matrícula, e-mail e edital de origem) são apresentados em modo somente leitura. Campos de vínculo (orientador, término previsto e término efetivo) podem ser ajustados pela secretaria/coordenação e persistidos pelo serviço responsável. Essa separação entre dados de identificação e dados do vínculo reduz erros de manutenção e preserva a rastreabilidade do ciclo de vida da bolsa;

- **Encerramento com motivo e data efetiva:** a finalização de um vínculo é realizada por meio de um diálogo específico na tela de detalhes. O usuário informa `endReason` (motivo) e `actualEndDate` (data efetiva). O serviço valida que ambos os campos foram preenchidos e atualiza o bolsista, registrando o encerramento e alterando o status para `ENCERRADA`. Esse procedimento garante consistência do histórico e fornece informações essenciais para auditoria, relatórios e prestação de contas (por exemplo, identificar encerramentos antecipados e seus motivos).

Em conjunto, as telas e serviços do módulo suportam o acompanhamento contínuo dos bolsistas, desde a concessão (criação do vínculo) até o encerramento com registro formal, mantendo integridade dos dados e aderência às necessidades operacionais do PPGI.

A.6 Gestão de informações de bolsistas

A gestão de informações de bolsistas corresponde ao conjunto de operações realizadas na tela de detalhes do vínculo, permitindo consultar e atualizar informações administrativas do bolsista (orientador, datas e status), além de registrar o encerramento com motivo e data efetiva. Essa funcionalidade é implementada pelo controlador *ManageScholarshipStudentInformationController* e pelo serviço *ManageScholarshipStudentInformationService*, mantendo o padrão arquitetural do Marvin no qual o controlador gerencia o estado da visão e o serviço encapsula regras e persistência.

- **Seleção do bolsista e carregamento do estado da tela:** ao navegar para a página de detalhes, o controlador carrega o bolsista selecionado e inicializa variáveis auxiliares de edição (por exemplo, identificador do orientador selecionado e campos temporários usados em diálogos). Essa etapa garante que a interface apresente informações consistentes e que alterações sejam feitas sobre um objeto válido;

- **Consulta e edição do vínculo (orientador e datas):** na seção de vínculo, o usuário pode selecionar um orientador a partir de uma lista carregada do serviço (ou DAO correspondente) e ajustar datas como `expectedEndDate` e `actualEndDate`. Ao acionar "Salvar alterações", o controlador delega ao serviço a atualização do bolsista, que resolve o orientador pelo identificador informado, valida campos obrigatórios e persiste o estado via *ScholarshipStudentDAO*. Esse fluxo evita que regras de domínio fiquem dispersas no *XHTML* e centraliza validações na camada de aplicação;
- **Encerramento de bolsa com registro obrigatório:** o encerramento é tratado como um macrofluxo separado, executado por um diálogo específico. Ao abrir o diálogo, o controlador prepara campos de entrada (`endReasonInput` e `actualEndDateInput`) com base no estado atual do bolsista, permitindo confirmação explícita do usuário. Ao confirmar, o serviço valida que o motivo e a data efetiva foram informados, atribui `endReason` e `actualEndDate` ao bolsista, altera o status para `ENCERRADA` e persiste a alteração. Esse procedimento é essencial para rastreabilidade, relatórios e auditoria do ciclo de vida do vínculo;
- **Tratamento de inconsistências e feedback ao usuário:** o controlador trata exceções de validação e de estado (por exemplo, bolsista inexistente, tentativa de encerrar vínculo já encerrado ou campos obrigatórios ausentes), exibindo mensagens claras ao usuário. Em caso de erro inesperado, o evento é registrado em *log* e uma mensagem genérica é apresentada, preservando a robustez da aplicação;
- **Integração com notificações do módulo:** o fluxo de encerramento pode acionar o serviço de notificações do módulo para registrar e enviar comunicações associadas ao evento (por exemplo, informar ao bolsista e/ou orientação a mudança de status), reutilizando a infraestrutura de e-mail do Marvin baseada em eventos e *templates*.

Assim, a gestão de informações de bolsistas consolida operações críticas do acompanhamento administrativo: garante atualização controlada do vínculo, formaliza encerramentos com justificativa e mantém o histórico necessário para prestação de contas e acompanhamento institucional do programa.

A.7 Notificações

O módulo de bolsas reutiliza a infraestrutura de e-mail do Marvin para enviar comunicações automáticas em pontos relevantes do fluxo. A implementação segue um padrão desacoplado: os serviços do módulo **não** enviam e-mails diretamente; em vez disso, disparam um evento de envio (*MailEvent*) com destinatário, *template* e um modelo de dados. Esse evento é observado por um componente central do Marvin (*Mailer*), responsável por renderizar o conteúdo e realizar a entrega via Jakarta Mail.

- **Padrão arquitetural: evento → Mailer:** as notificações são orquestradas pelo serviço *ManageScholarshipNotificationService* (EJB), que registra a ação em *log* e dispara um *MailEvent*. O componente *Mailer* do Marvin observa esse evento (execução assíncrona) e executa o envio efetivo via SMTP, usando a configuração central do sistema (servidor, porta, credenciais e remetente). Essa abordagem reduz acoplamento entre módulos, padroniza o envio e facilita manutenção;

- **Aceite de bolsa (template de aceitação):** ao converter uma candidatura aprovada em um *ScholarshipStudent*, o módulo chama `notifyAcceptance(student)`. Essa operação constrói um modelo de dados com informações do bolsista e do edital (por exemplo, nome, matrícula e título do edital) e dispara um *MailEvent* usando o template `SCHOLARSHIP_ACCEPTANCE`. O envio é interrompido caso o bolsista não possua e-mail válido, evitando falhas desnecessárias;

- **Encerramento/cancelamento (template de encerramento):** quando um vínculo é encerrado, `notifyClosure(student)` é chamado para comunicar a alteração. Além dos dados básicos, o modelo inclui o motivo (`endReason`) e a data efetiva de término (`actualEndDate`), e o *MailEvent* é disparado com o template `SCHOLARSHIP_CLOSURE`. Esse fluxo reforça rastreabilidade do encerramento e padroniza a comunicação;

- **Lembretes de relatório (scheduler diário + template de lembrete):** o módulo inclui um *scheduler* (EJB `@Singleton`) que executa diariamente um *job* (por padrão às 02:00 no horário do servidor) chamando `checkAndNotifyReportDeadlines()`. Nessa verificação, o serviço percorre bolsistas com status `ATIVA`, utiliza a periodicidade do edital (`reportPeriodInMonths`) e a data do último relatório (`lastReport`) para calcular a próxima entrega. Caso a data atual esteja dentro da janela de lembrete (por exemplo, sete dias antes do vencimento), o módulo dispara um *MailEvent* usando o template `SCHOLARSHIP_REPORT_REMINDER`;

- **Templates e dados de personalização:** os templates são identificados por constantes em *MailerTemplate* (por exemplo, `ScholarshipAcceptance`, `ScholarshipClosure` e `ScholarshipReportReminder`). O *Mailer* renderiza o conteúdo por meio de templates em classpath (com *data model* do tipo `Map<String,Object>`), o que permite evoluir textos e formatação sem alterar a lógica dos casos de uso.

Em síntese, a funcionalidade de notificações foi projetada para ser **transversal e desacoplada**: o módulo de bolsas define *quando* comunicar (eventos do ciclo de vida da bolsa e prazos), enquanto o núcleo do Marvin centraliza *como* enviar (renderização, configuração SMTP e entrega), mantendo consistência e reduzindo duplicação entre subsistemas.

A.8 Relatórios e exportação

Embora o foco principal deste trabalho seja a automação dos fluxos de inscrição, classificação e gestão de bolsistas, o módulo foi projetado para suportar exportações em formatos comuns:

- **Listas de inscrições por edital:** geração de relatórios com dados de candidatos, status de classificação e indicadores de ranking, com possibilidade de exportação para PDF ou planilhas Excel;
- **Listas de bolsistas:** exportação de informações de bolsistas ativos e encerrados, incluindo orientador, datas e motivos de encerramento, para uso em prestações de contas à agência de fomento ou à pró-reitoria;
- **Arquitetura preparada:** a camada de serviço e os DAOs já retornam estruturas de dados adequadas para construção desses relatórios. A implementação detalhada de todos os modelos de relatório é tratada como evolução natural do módulo.

A.9 Inicialização de dados (*Data Initializers*)

Para facilitar o uso inicial do módulo e apoiar cenários de demonstração e teste, foram implementados componentes de inicialização de dados (*data initializers*) executados no *startup* da aplicação. Esses componentes seguem o padrão já adotado no ecossistema Marvin: são EJBs `@Singleton` anotados com `@Startup` e executam sua lógica no ciclo de vida `@PostConstruct`, após a injeção de dependências.

- **Inicialização de editais (*ScholarshipNoticeInitializer*):** este inicializador garante a existência de um conjunto mínimo de editais padrão (por exemplo, CAPES, CNPq, FAPES, PaEPE, PIBEX e PIBIC). Para evitar duplicação entre reimplantações, a criação é **idempotente**: antes de persistir um edital, o componente verifica se já existe outro com o mesmo título (`retrieveByTitle`), registrando a decisão em *log* (criado vs. ignorado);
- **Inicialização de orientadores (*ScholarshipInstructorInitializer*):** este inicializador cria uma lista de orientadores de exemplo, associando `name` e `email` institucional. A verificação de duplicidade usa o e-mail como chave lógica: o componente consulta a lista de orientadores existentes e compara o e-mail de forma *case-insensitive*. Se já houver um registro com o mesmo e-mail, a criação é ignorada, evitando registros repetidos;
- **Inicialização de notificações (*ScholarshipNotificationInitializer*):** este inicializador cria notificações de exemplo vinculadas a cada edital, associando um

tipo (`ACCEPTANCE`, `REPORT_REMINDER` e `CLOSURE`), assunto (`subject`), corpo textual (`message`) e data/hora de criação (`createdAt`). A idempotência é garantida por uma verificação de existência por *(edital, tipo)*: se já houver uma notificação do mesmo tipo para o mesmo edital, o componente não recria a entrada.

Os *data initializers* reduzem o esforço de configuração inicial do módulo e tornam o protótipo reproduzível (por exemplo, após limpeza do banco ou em ambientes de homologação). Ao mesmo tempo, por criarem dados de exemplo, recomenda-se que seu uso seja controlado por política de implantação (por exemplo, habilitados em ambiente de desenvolvimento/homologação e revisados para produção), mantendo a base consistente com a realidade operacional do programa.

A.10 Telas do módulo

- **Tela de overview de editais** (Figura 17): apresenta cada edital em formato de *card*, exibindo título, período de inscrição e status, além de métricas consolidadas (quantidade de inscrições e quantidade de bolsistas vinculados). Na lateral, a tela disponibiliza ações administrativas típicas do ciclo de vida do edital, como abrir/encerrar o processo, acionar a geração de classificação (com configuração de vagas) e indeferir em lote candidatos classificados ou em lista de espera quando for necessário publicar um novo resultado.

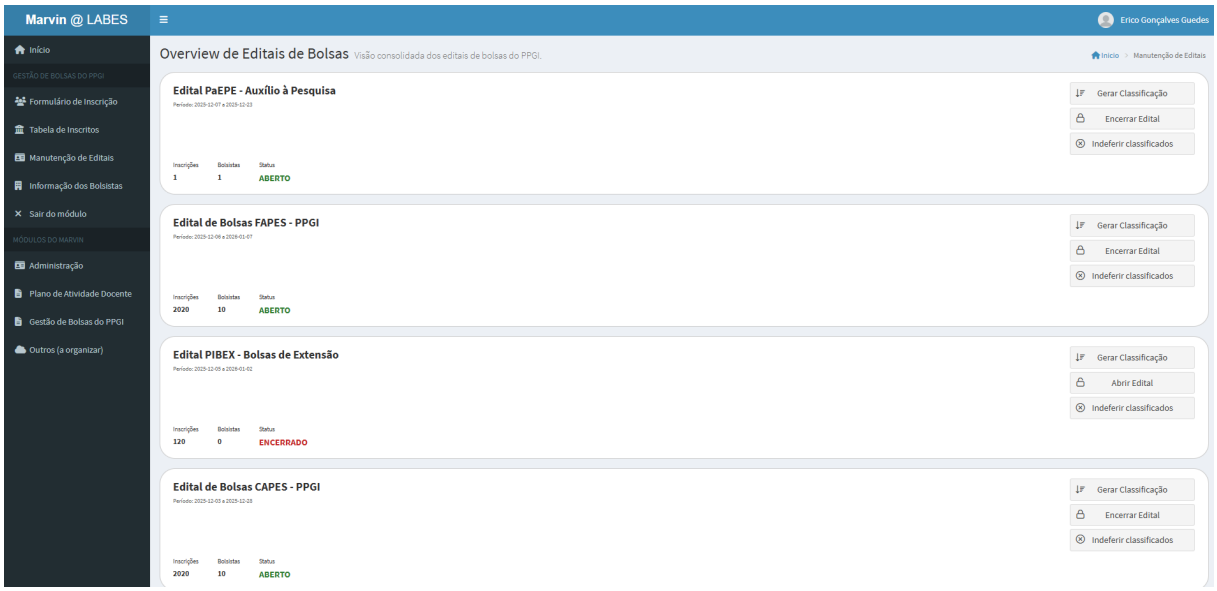

Figura 17 – Tela de overview de editais.

- **Tela de formulário de inscrição** (Figura 18): disponibiliza ao candidato um seletor de edital restrito a editais abertos e campos para preenchimento de dados pessoais e acadêmicos (por exemplo, matrícula, curso, nível, faixa de renda, categoria, coeficiente de rendimento e justificativa). Mensagens de validação são exibidas em caso de campos obrigatórios ausentes ou inconsistências de preenchimento, garantindo integridade mínima na submissão.

- **Diálogo de configuração de vagas (quotas)** (Figura 19): ao acionar "Gerar Classificação", é exibido um *pop-up* com uma tabela contendo todas as combinações de nível e categoria, com campos numéricos para informar o número de vagas por combinação. Os botões "Confirmar" e "Cancelar" controlam o envio (ou descarte) das configurações, permitindo aplicar quotas no processamento da classificação.

- **Tela de resultados/classificação** (Figura 20): apresenta a classificação final em formato de tabela com filtros por edital e atributos da candidatura (por exemplo, status, nível e categoria), além de busca textual por identificadores (nome, matrícula e e-mail). A tela oferece ações operacionais sobre candidatos (como criar bolsista

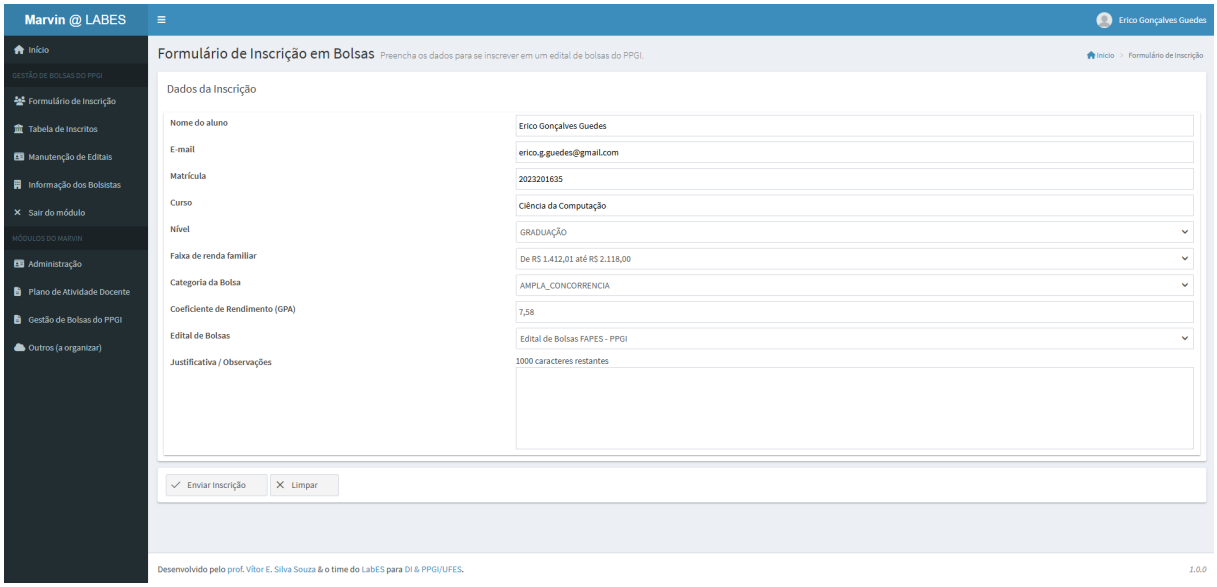

Figura 18 – Tela de formulário de inscrição.

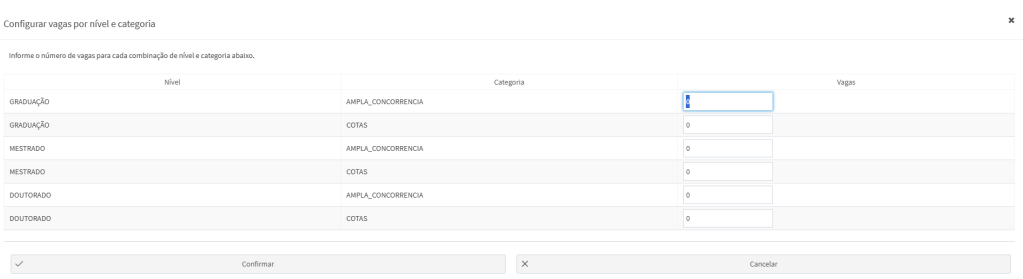

Figura 19 – Diálogo de configuração de vagas por nível e categoria.

a partir de um classificado ou finalizar candidatura em caso de desistência) e disponibiliza exportação da listagem filtrada para formatos de uso administrativo (PDF e planilha).

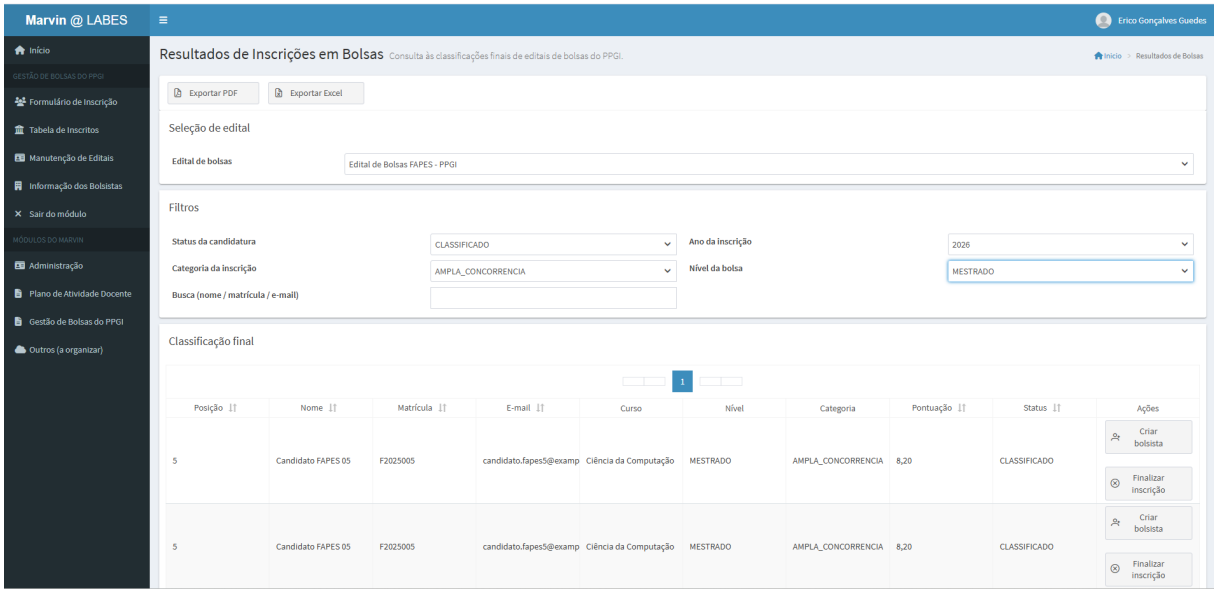

Figura 20 – Tela de resultados e classificação final.

- **Tela de listagem de bolsistas** (Figura 21): exibe uma tabela de bolsistas com filtros por edital de origem, status da bolsa, orientador e busca textual. A tela permite exportar a listagem para PDF e planilha e disponibiliza ação por linha para acesso aos detalhes do vínculo, suportando a rotina da secretaria e da coordenação.

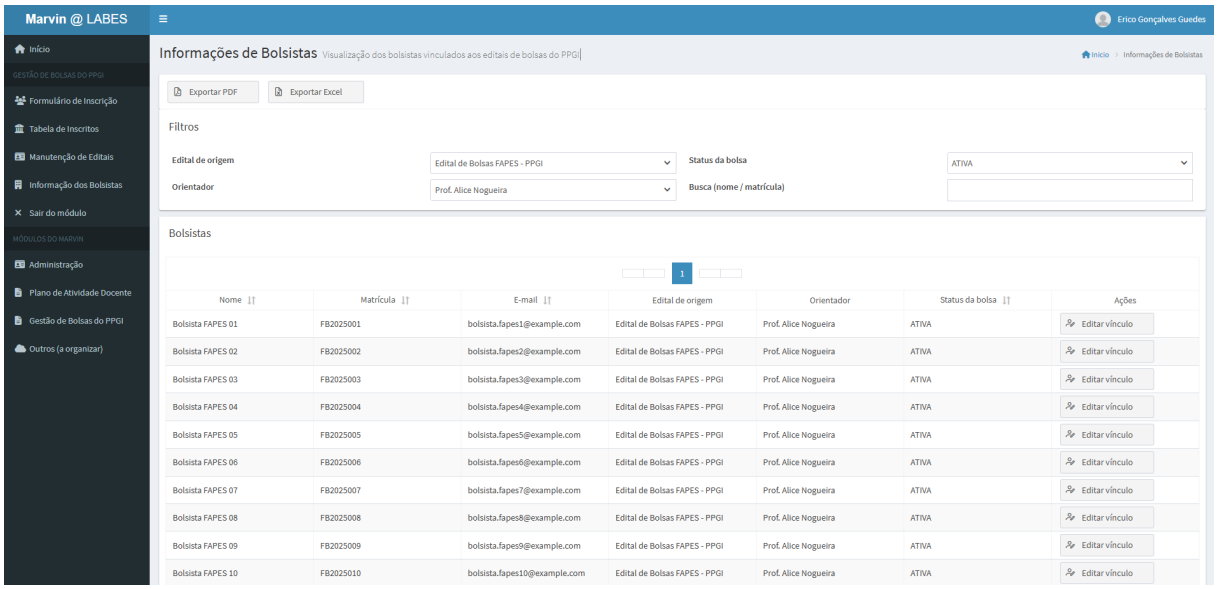

Figura 21 – Tela de listagem de bolsistas.

- **Tela de informações do bolsista** (Figura 22): apresenta dados básicos do bolsista (nome, matrícula, e-mail e edital de origem) e informações do vínculo (orientador, datas de início, término previsto e término efetivo, além do status atual da bolsa). A seção de edição permite atualizar o orientador e ajustar datas, registrando as alterações com feedback ao usuário.

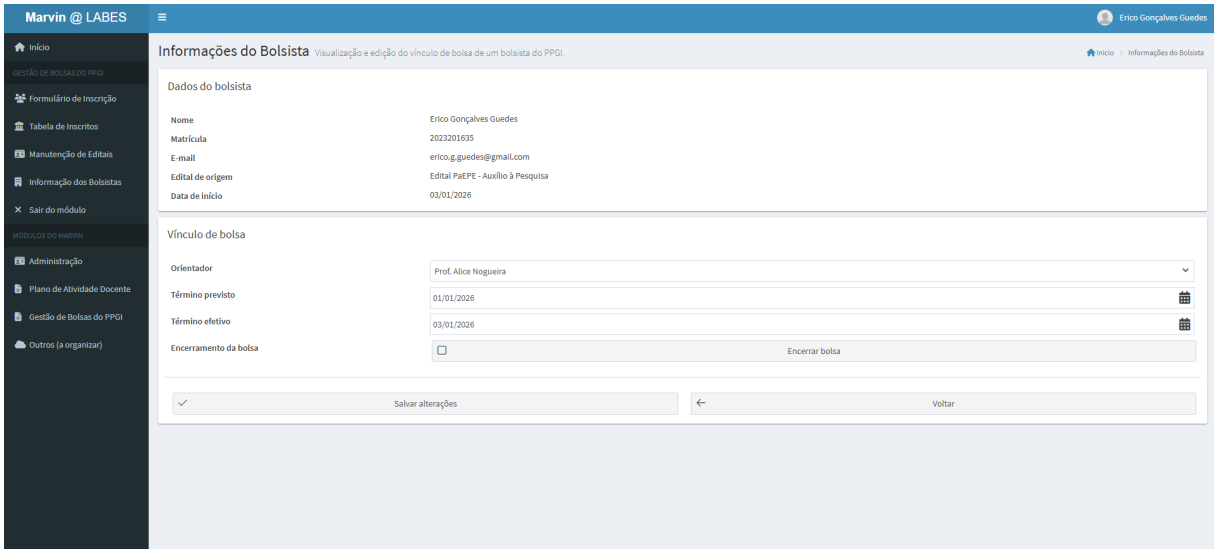

Figura 22 – Tela de informações do bolsista.

- **Diálogo de encerramento de bolsa** (Figura 23): acionado a partir da tela de informações do bolsista, permite informar o motivo de encerramento e a data efetiva

de término. O serviço valida os campos e, em caso de sucesso, atualiza o status da bolsa para `ENCERRADA` e registra o motivo para fins de auditoria e rastreabilidade do vínculo.

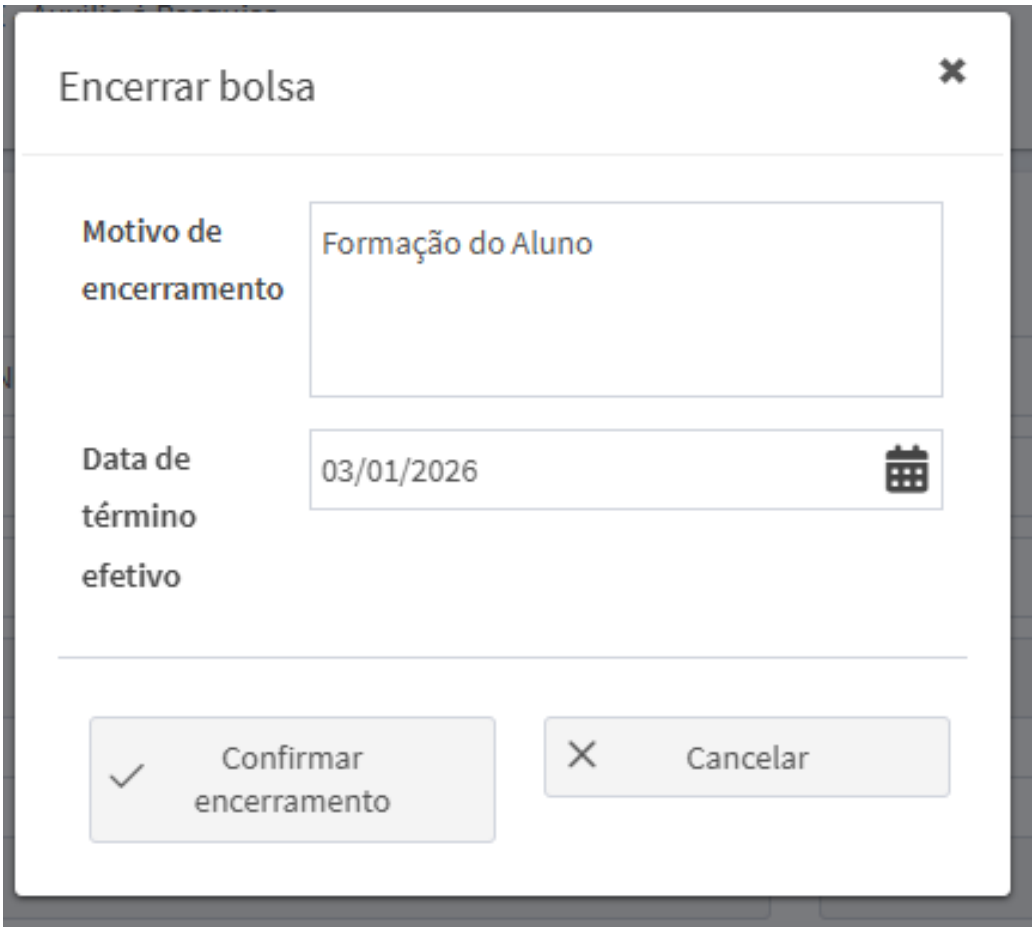

Figura 23 – Diálogo de encerramento de bolsa.

Em conjunto, essas telas implementam o fluxo de trabalho da secretaria, coordenação e comissão de bolsas do PPGI, cobrindo desde a abertura e manutenção de editais e inscrições até a classificação com quotas, a concessão e o acompanhamento de vínculos de bolsa, com suporte a exportação de resultados e registro rastreável de decisões ao longo do processo.

Documento de Requisitos de Sistema

Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin

Vitória, ES

2026

Contribuíram para esta especificação:

Nome	Contato	Contribuições
Erico Gonaçalves Guedes	LabES/erico-guedes	Escrita do documento.
Vítor E. Silva Souza	LabES/vitor-souza	Revisões.

Registro de alterações:

<https://www.overleaf.com/project/689117c666c5c6c04041c2e6>

1 Introdução

Este documento apresenta o levantamento e a definição de requisitos do **Sistema de Gestão de Bolsas do PPGI no Marvin**, módulo responsável por centralizar e apoiar o ciclo de vida das bolsas acadêmicas do Programa de Pós-Graduação em Informática (PPGI) da UFES.

Atualmente, a gestão de bolsas é realizada predominantemente por meio de planilhas, documentos compartilhados e trocas de e-mails. Não há um repositório único e versionado que consolide informações sobre editais, inscrições, classificações e vínculos de bolsistas (ativos e egressos). Como consequência, a rastreabilidade do processo é reduzida e aumentam os riscos de inconsistências, retrabalho e perda de histórico.

O módulo proposto busca apoiar os principais atores envolvidos (secretaria, coordenação, comissão de bolsas, orientadores e alunos), oferecendo uma visão unificada do processo e registros persistentes das decisões e eventos que compõem o ciclo de concessão e acompanhamento das bolsas.

1.1 Minimundo

No contexto do PPGI, diferentes modalidades de bolsa (CAPES, CNPq, FAPES, PaEPE, PIBIC, PIBEX, entre outras) são ofertadas periodicamente por meio de editais. Cada edital define regras próprias, período de inscrição, critérios de classificação e vigência, podendo também estabelecer cotas por nível e categoria, além de obrigações do bolsista (por exemplo, entrega periódica de relatórios).

Candidatos submetem inscrições em editais abertos. Em seguida, uma comissão de bolsas avalia as inscrições e realiza a classificação e a homologação dos resultados. Os candidatos aprovados tornam-se bolsistas, ficando vinculados a um edital de origem e, quando aplicável, a um orientador.

Ao longo da vigência, o bolsista deve cumprir exigências acadêmicas e administrativas (como manutenção de desempenho e entrega de relatórios). Ao final do período, a bolsa é encerrada por término de vigência, conclusão de curso ou desligamento antecipado, sempre com o devido registro do motivo e da data de encerramento.

1.2 Problemas observados

Foram identificados os seguintes problemas no processo atual:

- **Descentralização de dados**: informações sobre editais, inscrições e bolsistas encontram-se dispersas em múltiplas planilhas e pastas, sem controle de versão, padronização e integridade referencial;
- **Falta de controle de prazos**: não há um mecanismo automatizado que auxilie secretaria, comissão, bolsistas e orientadores no acompanhamento de prazos de relatório, renovação e encerramento;
- **Dificuldade de auditoria**: decisões de classificação e homologação são frequentemente registradas apenas em documentos estáticos, sem trilha de auditoria clara e integrada ao processo;
- **Risco de sobreposição de bolsas**: inexistem validações automáticas que impeçam um aluno de manter dois vínculos de bolsa ativos simultaneamente quando isso não é permitido;
- **Relatórios trabalhosos**: a secretaria despende tempo significativo consolidando informações de fontes distintas para responder a demandas da coordenação e das agências de fomento.

1.3 Objetivos do sistema

O Sistema de Gestão de Bolsas do PPGI tem como objetivos principais:

- **Centralizar** as informações de editais, inscrições e bolsistas em um módulo integrado ao Marvin;
- **Apoiar** a secretaria e a comissão de bolsas na condução de processos seletivos, classificação e homologação;
- **Rastrear** o histórico de vínculo de cada bolsista, incluindo datas de início e término, orientador responsável (quando aplicável) e motivo de encerramento;
- **Automatizar** notificações e lembretes de prazos críticos, reduzindo atrasos e riscos de descumprimento de obrigações;
- **Fornecer** relatórios gerenciais para coordenação e comissão, facilitando a tomada de decisão e a prestação de contas.

1.4 Escopo da versão inicial

A versão inicial do módulo contempla:

- Cadastro e gestão de **editais** (criação, edição, abertura, encerramento e parametrização de periodicidade de relatórios);
- Registro e acompanhamento de **inscrições** em editais abertos, com controle de status;
- **Classificação** de candidatos, incluindo configuração de cotas por nível e categoria;
- Conversão de inscrições classificadas em **bolsistas**, com controle de vigência e vínculo com orientador (quando aplicável);
- **Notificações** relacionadas a eventos relevantes do ciclo da bolsa (aceite, início, prazos de relatórios e encerramento);
- Geração de **relatórios** em formatos PDF e CSV.

Integrações diretas com outros sistemas institucionais da UFES (como SIGA) ou com portais de agências de fomento são consideradas fora do escopo desta versão e tratadas como oportunidades de trabalho futuro.

2 Estórias de Usuário e Requisitos

Este capítulo apresenta as estórias de usuário levantadas com os principais atores do processo de bolsas, bem como os requisitos funcionais, não funcionais e regras de negócio que orientam o comportamento esperado do sistema. Os itens são identificados por IDs (US, RF, RNF e RN) para facilitar rastreabilidade e validação ao longo do desenvolvimento.

2.1 Requisitos funcionais

Os requisitos funcionais descritos a seguir representam as principais funcionalidades que o sistema deve oferecer.

RF-01 – Cadastro de editais

O sistema deve permitir que a secretaria cadastre novos editais, informando título, agência de fomento, período de inscrições, periodicidade de relatórios, anexos e observações.

RF-02 – Abertura e encerramento de editais

O sistema deve permitir que apenas usuários autorizados abram e encerrem editais, controlando o status visível aos candidatos (rascunho, aberto, recursos, homologado, encerrado).

RF-03 – Inscrição em edital aberto

O sistema deve disponibilizar um formulário de inscrição para candidatos quando o edital estiver com status "ABERTO", vinculando a inscrição ao edital e à matrícula do aluno.

RF-04 – Classificação de inscrições

O sistema deve permitir que a comissão de bolsas execute um processo de classificação, considerando critérios de renda, desempenho acadêmico e outros parâmetros definidos no edital.

RF-05 – Gestão de cotas

O sistema deve permitir configurar cotas de vagas por combinação de nível (*graduação*, *mestrado*, *doutorado*) e categoria (*ampla concorrência*, *cotas*), aplicando tais limites na classificação de candidatos.

RF-06 – Conversão de inscrição em bolsista

O sistema deve permitir que a secretaria converta inscrições classificadas em bolsistas efetivos, registrando orientador, data de início, previsão de término e agência de fomento.

RF-07 – Encerramento de bolsa

O sistema deve permitir encerrar bolsas, exigindo data efetiva de término e motivo de encerramento (formatura, reprovação, pedido do aluno, fim de vigência, entre outros).

RF-08 – Notificações e lembretes

O sistema deve enviar notificações automáticas e/ou manuais (via e-mail ou módulo interno) para candidatos, bolsistas, orientadores e comissão, de acordo com eventos relevantes (confirmação de inscrição, aceite de bolsa, prazos de relatório, encerramento de vigência).

RF-09 – Relatórios gerenciais

O sistema deve gerar relatórios que consolidem dados de editais, inscrições, bolsistas ativos e egressos, permitindo filtros por período, agência, nível e orientador.

RF-10 – Aplicação de limites por nível e categoria

O sistema deve aplicar o limite de vagas por combinação *nível–categoria*, marcando candidatos dentro do limite como `CLASSIFICADO` e excedentes como `LISTA_ESPERA`.

RF-11 – Indeferimento em lote para reprocessamento

O sistema deve permitir indeferir, em lote, candidatos com status `CLASSIFICADO` e `LISTA_ESPERA` de um edital, quando for necessário invalidar/substituir um resultado anterior.

RF-12 – Criação de bolsista a partir de candidatura classificada

O sistema deve permitir criar registros de bolsistas a partir de candidaturas classificadas, copiando dados essenciais da inscrição e vinculando o bolsista ao edital de origem.

RF-13 – Associação de orientador ao bolsista

O sistema deve associar o bolsista a um orientador docente cadastrado, permitindo seleção automática a partir de uma lista definida e/ou ajustes manuais por usuários autorizados.

RF-14 – Consulta e filtros de bolsistas

O sistema deve permitir consultar e filtrar bolsistas por edital, status, orientador, nome e matrícula.

RF-15 – Atualização do vínculo de bolsa

O sistema deve permitir editar informações do vínculo de bolsa, incluindo status, orientador, data prevista de término e data de término efetiva (quando aplicável).

RF-16 – Encerramento com registro obrigatório

O sistema deve exigir, no encerramento de bolsa, o registro do motivo e da data de término efetivo, atualizando o status para `ENCERRADA`.

RF-17 – Notificação de concessão

O sistema deve enviar notificação por e-mail ao aluno quando uma bolsa for concedida.

RF-18 – Manutenção de notificações por código

O sistema deve permitir manter por código (cadastro/edição/exclusão) notificações de bolsas, incluindo tipo, assunto e mensagem, vinculadas a um edital.

RF-19 – Manutenção de orientadores por código

O sistema deve permitir manter por código (cadastro/edição/exclusão) orientadores, incluindo nome, e-mail e informações institucionais necessárias para associação com bolsistas.

RF-20 – Manutenção de classificadores por edital por código

O sistema deve permitir manter por código (cadastro/edição/exclusão) classificadores específicos por edital (regras/métodos de cálculo e ordenação), permitindo estratégias distintas entre editais.

RF-21 – Criação de conta e habilitação de perfil

O sistema deve permitir que alunos criem uma conta no Marvin e sejam habilitados com o perfil necessário para realizar inscrições em editais do módulo de bolsas.

RF-22 – Inicialização automática de dados-base

O sistema deve inserir automaticamente, durante a inicialização da aplicação, dados-base definidos manualmente (por exemplo, editais, orientadores, notificações e classificadores), quando não existirem na base, tornando o ambiente inicial utilizável e reprodutível.

2.2 Requisitos não funcionais

Os requisitos não funcionais estão organizados na Tabela 1.

2.3 Regras de negócio

As principais regras de negócio identificadas são:

Tabela 1 – Requisitos não funcionais do módulo de bolsas.

ID	**Categoria**	**Descrição**
RNF-01	Segurança	O módulo deve reutilizar o mecanismo de autenticação e autorização do Marvin, garantindo que apenas usuários autorizados acessem operações sensíveis.
RNF-02	Auditoria	Mudanças em editais, resultados de classificação e encerramentos de bolsa devem gerar registros de auditoria com usuário, data e descrição da ação.
RNF-03	Usabilidade	As telas devem seguir o padrão visual do Marvin/AdminFaces, com navegação consistente, terminologia clara e feedback ao usuário.
RNF-04	Disponibilidade	O sistema deve permanecer disponível durante o horário administrativo da secretaria, exceto em janelas de manutenção planejadas.
RNF-05	Manutenibilidade	O código deve seguir a arquitetura em camadas já estabelecida no Marvin, facilitando evolução e correções.
RNF-06	Usabilidade	As telas para secretaria e coordenação devem apresentar visão consolidada de editais e bolsistas, com indicadores e ações principais facilmente acessíveis.
RNF-07	Usabilidade	Formulários (especialmente o de inscrição) devem ser claros e concisos, com validações em português e instruções sobre campos obrigatórios.
RNF-08	Integração	O módulo deve seguir a arquitetura em camadas do Marvin, separando responsabilidades de apresentação, aplicação e persistência.
RNF-09	Manutenibilidade	O código deve obedecer às convenções de nomenclatura e organização de pacotes do Marvin, facilitando manutenção futura.
RNF-10	Operação	O módulo deve ser implantado no mesmo servidor de aplicações (WildFly) e utilizar o mesmo banco de dados PostgreSQL do Marvin, reduzindo esforço de operação.
RNF-11	Reprodutibilidade	A inicialização automática de dados-base deve ser idempotente, evitando duplicação de registros em reimplantações e garantindo reprodutibilidade do ambiente.

RN-01 É vedado manter mais de uma inscrição *ativa* do mesmo aluno para o mesmo edital, associada ao mesmo número de matrícula.

RN-02 Inscrições somente podem ser registradas para editais com status “ABERTO” e durante o respectivo período de inscrição.

RN-03 Um mesmo aluno não pode possuir duas bolsas vigentes simultaneamente no PPGI, salvo exceções explicitamente autorizadas (fora do escopo desta versão).

RN-04 A classificação deve respeitar a configuração de vagas por combinação *nível–categoria*. Candidatos além do limite devem compor lista de espera.

RN-05 Para fins de concessão, todo bolsista deve estar associado a um orientador docente cadastrado.

RN-06 Encerramentos de bolsa devem obrigatoriamente registrar motivo e data efetiva de término, garantindo rastreabilidade institucional.

2.4 Estórias de usuário

As tabelas a seguir apresentam um conjunto de estórias de usuário relacionadas ao módulo de bolsas, agrupadas por perfil.

Tabela 2 – Estórias de usuário – Secretaria / Coordenação / Comissão.

ID	Estória de Usuário
US-01	Como **secretaria**, quero **cadastrar um edital de bolsas** com período de inscrição, agência e anexo em PDF, para divulgar oportunidades de forma padronizada.
US-03	Como **comissão de bolsas**, quero **classificar automaticamente as inscrições** com base em critérios definidos e aplicando cotas por nível e categoria, para agilizar o processo seletivo.
US-06	Como **coordenação**, quero **publicar o resultado final** da classificação de um edital, para que o processo seletivo avance com segurança.
US-07	Como **coordenação**, quero **invalidar/substituir um resultado** de classificação já gerado, para permitir correção e republicação.
US-08	Como **secretaria**, quero **converter candidatos classificados em bolsistas ativos**, registrando orientador e vigência, para iniciar a gestão do vínculo de bolsa.
US-09	Como **secretaria**, quero **encerrar o vínculo de bolsa** registrando data e motivo, para manter histórico e prestação de contas.

Tabela 3 – Estórias de usuário – Aluno / Candidato / Bolsista.

ID	Estória de Usuário
US-02	Como **candidato**, quero **realizar minha inscrição em um edital aberto**, informando meus dados e anexando documentos exigidos, para concorrer a uma bolsa.
US-04	Como **aluno do PPGI**, quero **me inscrever em um edital de bolsa** selecionando o edital e preenchendo meus dados, para concorrer à vaga.
US-05	Como **bolsista**, quero **visualizar orientador e vigência corretos** no sistema, para acompanhar meu vínculo adequadamente.
US-10	Como **aluno do PPGI**, quero **criar uma conta no Marvin e ser habilitado com o perfil adequado**, para acessar o módulo e realizar inscrições.

2.5 Critérios de aceitação

A Tabela 5 apresenta critérios de aceitação para estórias selecionadas.

Tabela 4 – Estórias de usuário – Equipe de TI / Administração do Marvin.

ID	Estória de Usuário
US-11	Como **equipe de TI**, quero **que o módulo carregue dados-base automaticamente na inicialização** (sem duplicar registros), para facilitar testes, demonstrações e uso inicial.
US-12	Como **equipe de TI**, quero **manter por código os cadastros de orientadores, notificações e classificadores por edital**, para suportar variações de regras entre editais sem depender de telas administrativas.

3 Subsistema de Gestão de Bolsas no Marvin

Este capítulo apresenta uma visão de alto nível do subsistema de Gestão de Bolsas no ecossistema Marvin. São descritos os stakeholders, os perfis de usuário, os limites de responsabilidade do subsistema e as principais integrações com o Marvin Core.

3.1 Stakeholders

Os principais stakeholders associados ao módulo são:

- **Secretaria do PPGI**: opera o módulo diariamente, cadastrando editais, acompanhando inscrições, gerenciando vínculos de bolsa e gerando relatórios;
- **Comissão de Bolsas**: define critérios do processo seletivo, acompanha classificação e homologação e valida listas preliminares e finais;
- **Coordenação do PPGI**: utiliza o módulo para obter visão consolidada de bolsas ativas, encerramentos e egressos, além de indicadores de ocupação e distribuição por agência, nível e categoria;
- **Orientadores**: acompanham bolsistas sob sua orientação, recebendo notificações e consultando informações de vigência e obrigações;
- **Alunos/Candidatos**: realizam inscrição em editais abertos, acompanham o status da candidatura e recebem comunicações do processo;
- **Equipe de TI/Marvin**: mantém a infraestrutura do sistema, garantindo disponibilidade, segurança, suporte e evolução do módulo.

Tabela 5 – Critérios de aceitação para estórias selecionadas.

Estória	**Critérios de Aceitação**
US-01	• O sistema não permite salvar edital com data de fim anterior à data de início; • Editais publicados ficam visíveis na área de inscrições dos alunos; • O usuário responsável pelo cadastro é registrado para fins de auditoria.
US-02	• Apenas editais com status "ABERTO" aparecem na lista de seleção; • O sistema impede **segunda inscrição ativa** do mesmo aluno no mesmo edital; • Após submissão, o candidato recebe e-mail de confirmação.
US-03	• A classificação respeita o limite de vagas por combinação de nível e categoria; • Candidatos além do limite são colocados em "lista de espera"; • É possível exportar a lista de classificados para PDF/CSV.
US-08	• A conversão em bolsista é permitida apenas para inscrições com status `CLASSIFICADO`; • Ao concluir, o sistema registra orientador, início e término previsto; • O status da inscrição correspondente é atualizado para `BOLSISTA`.
US-09	• Campos obrigatórios: data efetiva de término e motivo de encerramento; • O sistema impede encerramento sem preenchimento dos campos obrigatórios; • Ao encerrar, o status do bolsista é atualizado para `ENCERRADA` e o histórico permanece consultável.
US-11	• Ao iniciar a aplicação, o sistema cria dados-base do módulo quando não existirem; • A inicialização é **idempotente**: reinicializações não duplicam registros existentes; • O processo registra em *log* o que foi criado e o que foi reaproveitado.

3.2 Perfis de usuário

A partir dos stakeholders, foram definidos os seguintes perfis de usuário:

Perfil Secretaria: acesso de leitura e escrita à maior parte das funcionalidades do módulo (editais, inscrições e bolsistas), com restrições apenas em operações administrativas globais do Marvin;

Perfil Comissão: acesso às telas relacionadas ao processo seletivo, incluindo visualização de inscrições, configuração de cotas e execução/consulta de classificações e listas resultantes;

Perfil Coordenação: foco em consultas e relatórios gerenciais, com acesso predominantemente de leitura a dados consolidados e históricos;

Perfil Orientador: acesso restrito aos bolsistas sob sua orientação, permitindo consulta de vigência, histórico do vínculo e notificações associadas;

Perfil Aluno: inscrição em editais abertos, acompanhamento do status de candidatura e, quando bolsista, visualização de informações do vínculo e notificações.

3.3 Visão geral do subsistema

A Figura 1 apresenta, em alto nível, os subsistemas relevantes e suas dependências.

O módulo de Gestão de Bolsas é concebido como um subsistema dentro do Marvin, com dependências claras para componentes já existentes:

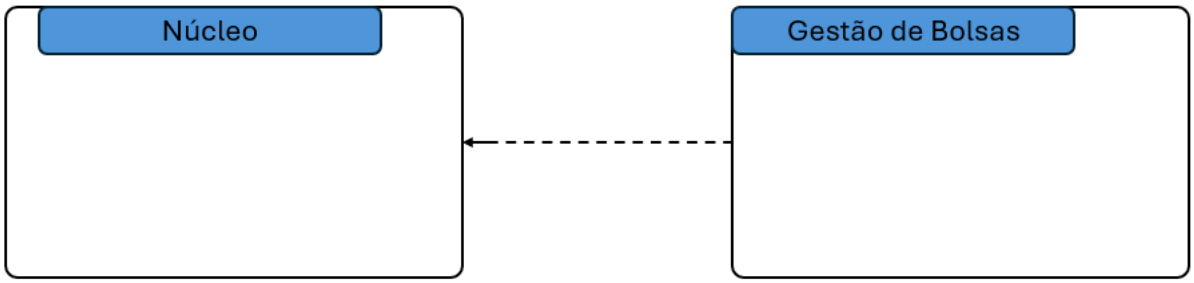

Figura 1 – Subsistemas do Marvin e interdependências do módulo de Bolsas.

O subsistema de Gestão de Bolsas organiza o ciclo de vida de bolsas acadêmicas por meio de editais. Em termos conceituais, o fluxo típico envolve: (i) criação e publicação do edital, (ii) inscrições de candidatos, (iii) classificação e homologação, (iv) concessão do vínculo de bolsa (bolsista) e (v) acompanhamento e encerramento do vínculo, preservando histórico e motivo.

Para apoiar esse fluxo, o subsistema lida com entidades específicas como *Edital*, *Inscrição*, *Classificação/Resultado*, *Bolsista* e *Notificação*, além de referências a entidades já existentes no Marvin (usuários e perfis).

3.4 Limites e responsabilidades do subsistema

O subsistema de Gestão de Bolsas é responsável por:

- modelar e persistir informações de **editais**, **inscrições**, **resultados de classificação**, **vínculos de bolsa** (bolsistas) e **notificações** do processo;
- orquestrar o **fluxo de negócio** desde a criação/publicação de um edital, passando por inscrição e classificação, até a concessão e o encerramento do vínculo de bolsa, com registro do motivo e datas relevantes;
- prover **telas e serviços** para secretaria, comissão, coordenação, orientadores e alunos, com controle de acesso baseado em perfis;
- garantir **rastreabilidade** das ações relevantes (por exemplo, alterações em editais, geração de resultados e encerramentos), apoiando auditoria e prestação de contas.

Não faz parte do escopo do subsistema:

- gerenciar o cadastro institucional completo de pessoas (alunos, docentes e técnicos), uma vez que isso é tratado por módulos centrais do Marvin;
- realizar comunicação direta e automática com sistemas externos de agências de fomento (integrações por API/portal), que são consideradas oportunidades de trabalho futuro;
- controlar dados financeiros detalhados de pagamento (folha, fichas e repasses), que permanecem sob responsabilidade de outros sistemas.

3.5 Integração com o Marvin

O módulo foi concebido como um subsistema acoplado ao Marvin Core, reutilizando componentes e serviços já existentes, com destaque para:

- **Autenticação e autorização**: login único, papéis e permissões centralizados, garantindo que operações sensíveis sejam executadas apenas por perfis autorizados;
- **Cadastro de usuários**: reutilização das informações básicas de alunos, docentes e

técnicos mantidas no núcleo do sistema, evitando duplicação de dados;

- **Infraestrutura de e-mail/notificações**: utilização do serviço existente do Marvin para envio de comunicações e lembretes associados a eventos do ciclo de bolsas (por exemplo, confirmação de inscrição, concessão de bolsa e prazos de relatórios);
- **Padrão de interface**: adoção de templates, estilos e componentes do Marvin (PrimeFaces/AdminFaces), preservando consistência visual e de navegação.

Essa integração reduz a duplicação de esforço, mantém consistência funcional e visual entre módulos e favorece a manutenibilidade ao longo do tempo, uma vez que o subsistema herda padrões e infraestrutura já consolidados no Marvin.

4 Casos de Uso

Este capítulo apresenta uma visão estruturada dos principais casos de uso do Sistema de Gestão de Bolsas do PPGI, derivados das estórias de usuário levantadas e das regras de negócio estabelecidas. Os casos de uso são apresentados em nível macro, destacando atores e objetivos, e exemplificados por uma especificação textual detalhada.

Os casos de uso capturam, em nível mais formal, as interações entre atores e o sistema. A Figura **??** ilustra um diagrama de casos de uso em alto nível para o módulo de bolsas, com os atores principais:

Tabela 6 – Descrição dos atores envolvidos nos casos de uso (Elaboração própria).

Ator	Descrição
Comissão de Bolsas	Define critérios e conduz o processo de classificação/homologação das inscrições, acompanhando resultados e regras aplicáveis aos editais.
Aluno	Realiza inscrição em editais, acompanha o status da candidatura e, quando bolsista, consulta informações do vínculo e notificações relacionadas.
Equipe de TI	Mantém a infraestrutura e a disponibilidade do sistema, apoiando implantação, configuração, monitoramento e manutenção técnica do módulo.

4.1 Visão geral

Em alto nível, o módulo contempla os seguintes macrocasos de uso:

- **UC-01 – Gerenciar Editais**;

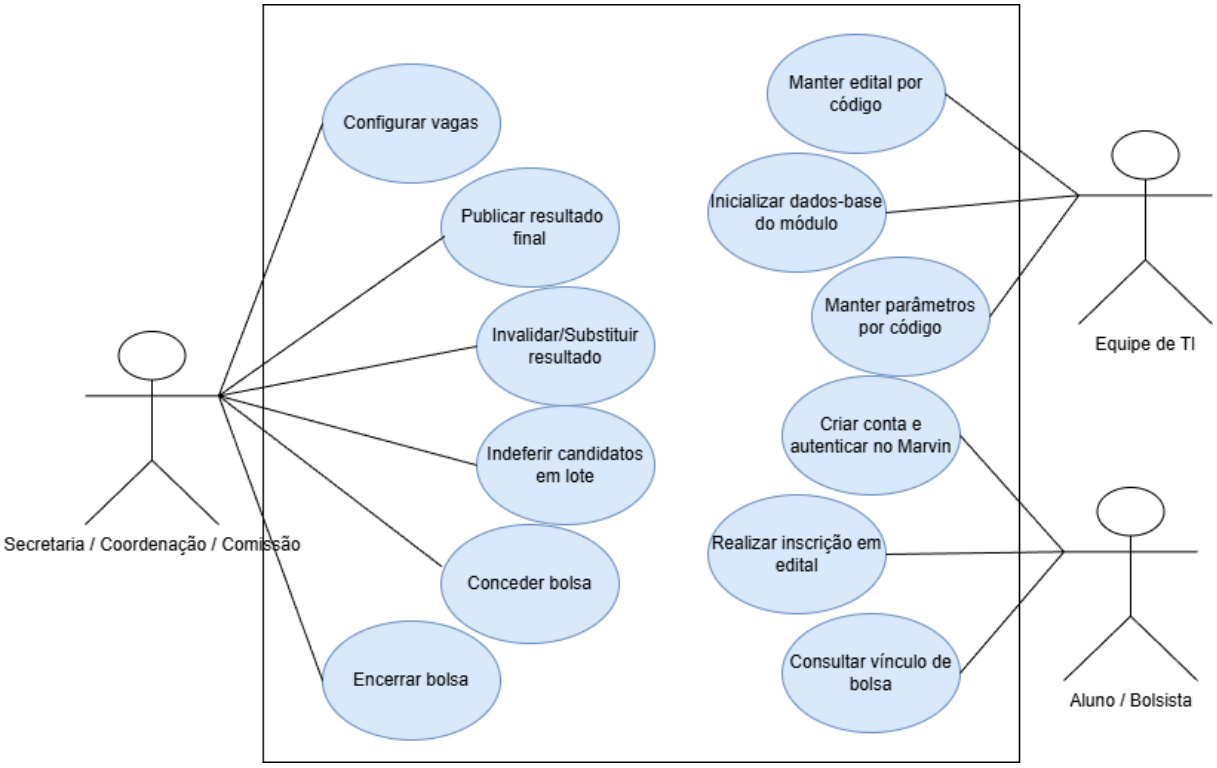

Figura 2 – Diagrama de Casos de Uso do subsistema Gestão de Bolsas.

- **UC-02 – Realizar Inscrição em Edital Aberto**;
- **UC-03 – Classificar Candidaturas**;
- **UC-04 – Gerenciar Bolsistas**;
- **UC-05 – Gerar Notificações**;
- **UC-06 – Emitir Relatórios**.

4.2 Lista de casos de uso

A Tabela 7 resume cada caso de uso, seus atores principais e uma breve descrição.

4.3 Especificação textual de casos de uso

A seguir, é apresentada uma especificação textual detalhada para o caso de uso UC-02. Especificações similares podem ser desenvolvidas para os demais casos de uso, incluindo fluxos alternativos, exceções, regras de negócio associadas e requisitos de auditoria.

UC-02 – Realizar Inscrição em Edital Aberto

Objetivo: permitir que um aluno submeta uma inscrição para concorrer a uma bolsa em um edital vigente e aberto.

Tabela 7 – Resumo dos casos de uso do módulo de bolsas.

ID	**Nome**	**Atores principais**	**Descrição**
UC-01	Gerenciar Editais	Secretaria, Coordenação	Permite criar, editar, publicar (abrir), homologar e encerrar editais, incluindo anexos, regras e parâmetros (por exemplo, período e cotas).
UC-02	Realizar Inscrição	Aluno (candidato)	Permite ao aluno submeter uma inscrição em edital aberto, preenchendo dados e anexando documentos exigidos.
UC-03	Classificar Candidaturas	Comissão de Bolsas	Permite executar a classificação das inscrições, aplicando critérios e cotas definidas no edital e gerando lista de classificados e lista de espera.
UC-04	Gerenciar Bolsistas	Secretaria	Permite converter inscrições classificadas em bolsistas, registrar orientador, acompanhar vigência e registrar encerramentos com data e motivo.
UC-05	Gerar Notificações	Secretaria, Sistema	Permite disparar notificações automáticas e/ou manuais relacionadas a eventos relevantes (confirmação de inscrição, concessão de bolsa, prazos e encerramento).
UC-06	Emitir Relatórios	Coordenação, Comissão, Secretaria	Permite gerar relatórios consolidados sobre editais, inscrições, bolsistas e egressos, com filtros e exportação (por exemplo, PDF/CSV).

Atores principais: Aluno (candidato).

Atores secundários: Serviço de autenticação/autorização do Marvin; Serviço de e-mail do Marvin.

Pré-condições:

- O usuário deve estar autenticado no Marvin e possuir perfil de aluno;
- Deve existir ao menos um edital com status “ABERTO”;
- O período de inscrição do edital selecionado deve estar vigente.

Pós-condições (em caso de sucesso):

- A inscrição é persistida e vinculada ao edital e à matrícula do aluno;

- O aluno recebe confirmação de submissão (por e-mail e/ou notificação interna);
- A inscrição passa a estar disponível para avaliação e classificação pela comissão.

Fluxo principal:

1. O aluno acessa a área do módulo de bolsas e seleciona a opção de inscrições;
2. O sistema apresenta a lista de editais com status “ABERTO”;
3. O aluno seleciona o edital desejado;
4. O sistema apresenta o formulário de inscrição e preenche, quando possível, dados já conhecidos (por exemplo, nome e matrícula);
5. O aluno informa dados complementares e anexa documentos exigidos pelo edital;
6. O aluno confirma o envio da inscrição;
7. O sistema valida dados e anexos, registra a inscrição e exibe mensagem de sucesso;
8. O sistema envia e-mail (e/ou notificação interna) confirmando a submissão.

Fluxos alternativos e exceções:

- *A1 – Nenhum edital aberto:* se não houver editais com status “ABERTO”, o sistema informa que não há chamadas disponíveis no momento.
- *A2 – Inscrição duplicada:* se já existir inscrição ativa do mesmo aluno para o mesmo edital, o sistema bloqueia o envio e exibe mensagem explicativa.
- *A3 – Fora do período de inscrição:* se o edital estiver “ABERTO”, porém fora do intervalo configurado para inscrições, o sistema impede a submissão e informa o período válido.
- *A4 – Documentação incompleta ou inválida:* se anexos obrigatórios não forem enviados ou estiverem em formato inválido, o sistema impede o envio e destaca os itens pendentes.
- *A5 – Falha no envio de e-mail:* se o envio de e-mail falhar, a inscrição permanece registrada; o sistema registra o erro e apresenta mensagem informando que a confirmação poderá ser consultada no sistema.

Regras de negócio relacionadas:

- Não é permitida mais de uma inscrição ativa do mesmo aluno no mesmo edital;

- Apenas editais abertos e dentro do período de inscrição aceitam submissões.

5 Modelo Estrutural e Conceitual

Este capítulo apresenta o modelo conceitual do módulo de bolsas, destacando as entidades centrais, seus relacionamentos e restrições estruturais que orientam a implementação (por exemplo, mapeamento JPA e validações de integridade).

5.1 Visão geral do modelo de domínio

A Figura 3 apresenta o diagrama de classes em alto nível. Embora a modelagem conceitual seja própria da fase de requisitos, este diagrama inclui alguns atributos para tornar explícitas as informações relevantes registradas ao longo do processo.

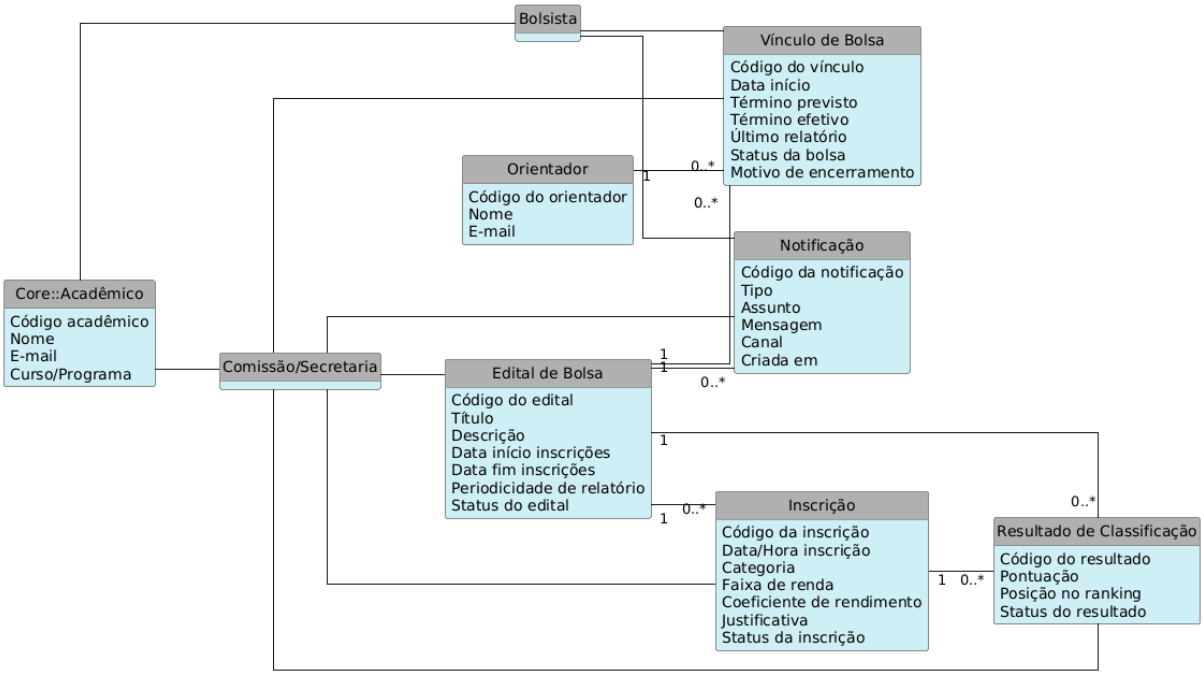

Figura 3 – Modelo conceitual do módulo de Gestão de Bolsas.

O modelo de domínio do módulo de bolsas é composto, em linhas gerais, pelas seguintes classes conceituais:

- **ScholarshipNotice** (Edital de Bolsa);
- **ScholarshipApplication** (Inscrição);

- **ScholarshipApplicationClassification** (Classificação de Inscrição);
- **ScholarshipStudent** (Bolsista);
- **ScholarshipInstructor** (Orientador);
- **Notification** (Notificação).

De forma resumida, o *Edital* organiza o processo seletivo; o *Aluno* submete uma *Inscrição*; a *Comissão* executa a *Classificação*; e a *Secretaria* concede e administra o vínculo de *Bolsista*, que pode gerar *Notificações* ao longo do tempo.

5.2 Entidades principais

ScholarshipNotice (Edital)

Representa um edital de bolsas publicado pelo PPGI. Contém informações como:

- identificador, título e descrição/observações;
- agência de fomento e tipo/modalidade de bolsa;
- período de inscrições (data de início e fim);
- status do edital (por exemplo: rascunho, aberto, recursos, homologado, encerrado);
- parâmetros do processo seletivo (por exemplo, cotas por nível e categoria);
- parâmetros operacionais (por exemplo, periodicidade/obrigações de relatórios e anexos).

ScholarshipApplication (Inscrição)

Modela a inscrição submetida por um candidato para um edital específico. Armazena:

- dados de identificação do candidato (nome, matrícula e e-mail);
- dados acadêmicos (curso e nível);
- informações relevantes para seleção (por exemplo, categoria de concorrência e faixa de renda);
- status da inscrição (por exemplo: inscrito, classificado, lista de espera, bolsista, indeferido);

- vínculo com o *ScholarshipNotice* de origem;
- metadados de submissão (data/hora e anexos/documentos exigidos).

ScholarshipApplicationClassification (Classificação de Inscrição)

Modela o resultado do processo de classificação associado a uma inscrição, incluindo:

- posição/ordem no ranking do edital;
- pontuação (score) calculada a partir de critérios definidos no edital;
- situação no resultado (por exemplo: classificado, lista de espera, indeferido);
- referência ao edital e à inscrição correspondente.

ScholarshipStudent (Bolsista)

Representa o bolsista (ativo ou egresso) e o vínculo de bolsa concedido. Registra:

- dados básicos do aluno (nome, matrícula e e-mail);
- dados acadêmicos (curso, nível) e orientador responsável;
- datas relevantes do vínculo (início, término previsto e término efetivo);
- status do vínculo (por exemplo: ativa, suspensa, encerrada);
- motivo de encerramento, quando aplicável;
- referência à inscrição e ao edital que originaram o vínculo.

ScholarshipInstructor (Orientador)

Entidade que representa o docente orientador associado ao bolsista. Em termos conceituais, reutiliza (ou estende) o cadastro de usuários do Marvin, mantendo atributos essenciais como nome e e-mail institucional, além de um identificador.

Notification (Notificação)

Armazena registros de notificações emitidas pelo módulo para candidatos, bolsistas, orientadores ou perfis administrativos, incluindo:

- tipo de notificação (por exemplo: confirmação de inscrição, concessão, lembrete, encerramento);

- canal utilizado (e-mail e/ou mensagem interna);
- assunto, conteúdo e data/hora de envio;
- destinatário e referências contextuais (por exemplo, edital, inscrição ou vínculo de bolsa).

5.3 Relacionamentos fundamentais

Em termos de relacionamentos, destacam-se:

- Um *ScholarshipNotice* (*Edital*) está associado a várias *ScholarshipApplication* (*Inscrições*) — relação **1:N**.
- Um *ScholarshipNotice* pode possuir vários resultados de *ScholarshipApplication-Classification* — relação **1:N**. Em geral, cada classificação referencia a inscrição correspondente e o edital em que ocorreu.
- Uma *ScholarshipApplication* está associada a exatamente um *ScholarshipNotice* — relação **N:1**.
- Uma *ScholarshipApplication* pode originar no máximo um *ScholarshipStudent* — relação **0..1 : 1**. Esse vínculo só ocorre após a inscrição ser classificada e convertida pela secretaria.
- Um *ScholarshipStudent* está associado a um único *ScholarshipInstructor* — relação **N:1**, considerando que um orientador pode orientar diversos bolsistas.
- Um *ScholarshipStudent* pode possuir diversas *Notification* ao longo do tempo — relação **1:N**.

5.4 Restrições conceituais e integridade

Além dos relacionamentos, algumas restrições conceituais são relevantes para o projeto e implementação do módulo:

- **Unicidade de inscrição por edital:** um aluno não deve possuir mais de uma inscrição ativa para o mesmo edital;
- **Consistência de datas do vínculo:** a data de término efetivo não pode ser anterior à data de início do bolsista;
- **Vínculo exclusivo de bolsa:** um aluno não deve manter mais de um vínculo de bolsa ativo simultaneamente no PPGI (salvo exceções fora do escopo);

- **Encerramento com motivo:** todo encerramento deve ser registrado com motivo e data, preservando histórico e rastreabilidade.

Esses elementos estruturais constituem a base para a modelagem detalhada de dados (mapeamento JPA, chaves estrangeiras e restrições), e orientam o desenho dos casos de uso e fluxos de interface discutidos nos capítulos anteriores.

6 Dicionário de Dados

Este capítulo apresenta um dicionário de dados simplificado para as principais entidades do Sistema de Gestão de Bolsas do PPGI. O objetivo é registrar, de forma tabular, atributos, tipos conceituais e descrições associados às classes de domínio, servindo como referência para implementação, testes e manutenção.

6.1 Entidade `ScholarshipNotice`

Tabela 8 – Dicionário de dados da entidade `ScholarshipNotice`.

Atributo	**Tipo (conceitual)**	**Descrição**
id	Inteiro/UUID	Identificador único do edital.
title	Texto	Título do edital de bolsas.
description	Texto	Descrição e/ou observações gerais do edital.
fundingAgency	Enum/Texto	Agência de fomento (CAPES, CNPq, FAPES etc.).
applicationStartDate	Data	Data de início do período de inscrições.
applicationEndDate	Data	Data de término do período de inscrições.
status	Enum	Situação do edital (rascunho, aberto, recursos, homologado, encerrado).
reportPeriodicity	Enum	Periodicidade de relatórios exigidos (semestral, anual etc.).
attachments	Arquivo/Lista	Anexos associados ao edital (por exemplo, PDF).
createdAt	Data/Hora	Data e hora de criação do registro.
updatedAt	Data/Hora	Data e hora da última atualização do registro.

6.2 Entidade `ScholarshipApplication`

Tabela 9 – Dicionário de dados da entidade `ScholarshipApplication`.

Atributo	**Tipo (conceitual)**	**Descrição**
id	Inteiro/UUID	Identificador único da inscrição.
applicationDateTime	Data/Hora	Momento de submissão da inscrição.
name	Texto	Nome completo do candidato.
registrationNumber	Texto	Matrícula do aluno no PPGI.
email	Texto	E-mail de contato do candidato.
course	Texto	Curso/programa ao qual o candidato está vinculado.
level	Enum	Nível (graduação, mestrado, doutorado).
incomeBracket	Enum	Faixa de renda declarada (ex.: baixa, média-baixa etc.).
category	Enum	Categoria de concorrência (ampla concorrência, cotas).
status	Enum	Status da inscrição (inscrito, classificado, lista de espera, indeferido, bolsista).
documents	Arquivo/Lista	Documentos/anexos enviados pelo candidato, quando exigidos.
scholarshipNoticeId	Referência	Referência (FK conceitual) ao edital `ScholarshipNotice`.
createdAt	Data/Hora	Data e hora de criação do registro.
updatedAt	Data/Hora	Data e hora da última atualização do registro.

6.3 Entidade `ScholarshipApplicationClassification`

Tabela 10 – Dicionário de dados da entidade `ScholarshipApplicationClassification`.

Atributo	**Tipo (conceitual)**	**Descrição**
id	Inteiro/UUID	Identificador único do resultado de classificação.
rankPosition	Inteiro	Posição do candidato no ranking do edital.
score	Decimal	Pontuação calculada no processo de classificação.
resultStatus	Enum	Situação no resultado (classificado, lista de espera, indeferido).
scholarshipNoticeId	Referência	Referência (FK conceitual) ao edital `ScholarshipNotice`.
scholarshipApplicationId	Referência	Referência (FK conceitual) à inscrição `ScholarshipApplication`.
generatedAt	Data/Hora	Data e hora em que o resultado foi gerado/publicado.

6.4 Entidade `ScholarshipStudent`

Tabela 11 – Dicionário de dados da entidade `ScholarshipStudent`.

Atributo	**Tipo (conceitual)**	**Descrição**
id	Inteiro/UUID	Identificador único do vínculo de bolsista.
name	Texto	Nome do bolsista.
registrationNumber	Texto	Matrícula do bolsista.
email	Texto	E-mail de contato.
course	Texto	Curso/programa do bolsista.
level	Enum	Nível (graduação, mestrado, doutorado).
startDate	Data	Data de início da bolsa.
expectedEndDate	Data	Data de término prevista da bolsa.
actualEndDate	Data	Data de término efetiva da bolsa (quando encerrada).
status	Enum	Situação da bolsa (ativa, suspensa, encerrada).
endReason	Texto/Enum	Motivo de encerramento da bolsa, quando aplicável.
scholarshipNoticeId	Referência	Referência (FK conceitual) ao edital de origem.
scholarshipApplicationId	Referência	Referência (FK conceitual) à inscrição de origem.
scholarshipInstructorId	Referência	Referência (FK conceitual) ao orientador responsável.
createdAt	Data/Hora	Data e hora de criação do registro.
updatedAt	Data/Hora	Data e hora da última atualização do registro.

6.5 Entidade `ScholarshipInstructor`

Tabela 12 – Dicionário de dados da entidade `ScholarshipInstructor`.

Atributo	**Tipo (conceitual)**	**Descrição**
id	Inteiro/UUID	Identificador do orientador no módulo.
name	Texto	Nome completo do orientador.
email	Texto	E-mail institucional do orientador.
active	Booleano	Indica se o orientador está ativo para vinculação.
marvinUserId	Referência	Referência ao usuário correspondente no cadastro do Marvin, quando aplicável.

6.6 Entidade `Notification`

Tabela 13 – Dicionário de dados da entidade `Notification`.

Atributo	**Tipo (conceitual)**	**Descrição**
id	Inteiro/UUID	Identificador da notificação.
type	Enum	Tipo (confirmação de inscrição, concessão, lembrete, encerramento etc.).
channel	Enum	Canal (e-mail, aviso interno).
subject	Texto	Assunto da notificação (quando aplicável).
content	Texto	Conteúdo textual da mensagem enviada.
sentAt	Data/Hora	Momento do envio da notificação.
recipientType	Enum	Tipo de destinatário (candidato, bolsista, orientador, comissão, secretaria).
recipientId	Referência	Referência ao destinatário (conforme o tipo).
scholarshipNoticeId	Referência	Referência ao edital relacionado (quando aplicável).
scholarshipApplicationId	Referência	Referência à inscrição relacionada (quando aplicável).
scholarshipStudentId	Referência	Referência ao bolsista relacionado (quando aplicável).
createdAt	Data/Hora	Data e hora de criação do registro.

O dicionário pode ser expandido conforme novos atributos e entidades sejam introduzidos no modelo. Em especial, ajustes de tipos e enums devem refletir as decisões de implementação (por exemplo, uso de UUID, enums persistidos como `String` ou `Integer`, e modelagem de anexos/documentos).

Documento de Projeto de Sistema

Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin

Vitória, ES

2026

Registro de Alterações:

Versão	Responsável	Data	Alterações
1.0	Erico Gonçalves Guedes	30/10/2025	Versão inicial.
2.0	Erico Gonçalves Guedes	25/12/2025	Versão Parcial.
3.0	Erico Gonçalves Guedes	26/1/2025	Versão Final.
3.1	Erico Gonçalves Guedes	6/1/2025	Versão Final Corrigida.

1 Introdução

Este documento apresenta o projeto (*design*) do *Sistema de Gerenciamento Integrado de Bolsas para Programas Acadêmicos no Marvin*, doravante denominado *Sistema de Gestão de Bolsas*. O subsistema foi concebido para apoiar, de forma integrada, o ciclo de vida das bolsas acadêmicas no contexto do Programa de Pós-Graduação em Informática (PPGI) da UFES, abrangendo desde a definição e publicação de editais, o recebimento e classificação de inscrições, até a formalização do vínculo de bolsistas, o acompanhamento da vigência e o registro histórico de encerramentos e egressos. :contentReference[oaicite:2]index=2

Atualmente, grande parte desses processos é conduzida por meio de planilhas, documentos compartilhados e trocas de e-mails, sem um repositório único e versionado que consolide informações sobre editais, inscrições, classificações e bolsistas. Essa fragmentação dificulta a rastreabilidade, reduz a transparência dos fluxos e aumenta o risco de inconsistências e erros administrativos. :contentReference[oaicite:3]index=3 :contentReference[oaicite:4]index=4

O Sistema de Gestão de Bolsas busca centralizar essas atividades no Marvin, reutilizando mecanismos e dados já mantidos pelo núcleo do sistema (como autenticação/autorização e cadastro de usuários) e adicionando a lógica específica do domínio de bolsas (editais, inscrições, classificação, concessão e encerramento). :contentReference[oaicite:5]index=5 Integrações diretas com outros sistemas institucionais da UFES (por exemplo, SIGA) e com portais de agências de fomento são consideradas fora do escopo desta versão, sendo tratadas como oportunidades de trabalho futuro. :contentReference[oaicite:6]index=6

O projeto do sistema foi elaborado a partir do levantamento de requisitos funcionais e não funcionais descrito em documento específico de requisitos, que apresenta minimundo, problemas observados, objetivos e escopo da versão inicial. :contentReference[oaicite:7]index=7 :contentReference[oaicite:8]index=8 Este documento, por sua vez, foca na visão de projeto: detalha a plataforma de desenvolvimento, as decisões arquiteturais e os componentes do subsistema de bolsas, em alinhamento com a arquitetura já estabelecida do Marvin.

Além desta introdução, o documento está organizado da seguinte forma:

- o Capítulo 2 descreve a plataforma de desenvolvimento e as tecnologias adotadas na implementação do sistema;
- o Capítulo 3 apresenta os requisitos não funcionais que guiam as decisões arquiteturais, incluindo táticas e critérios de aceitação;

- o Capítulo 4 descreve a arquitetura de software do subsistema de bolsas e sua integração com o núcleo do Marvin;
- o Capítulo 5 detalha o projeto dos componentes da arquitetura, organizados por camada (domínio, aplicação, interface com o usuário e persistência).

2 Plataforma de Desenvolvimento

Esta seção descreve a plataforma de implementação adotada para o desenvolvimento do Sistema de Gestão de Bolsas. O subsistema foi implementado como parte da aplicação *Marvin*, utilizando o ecossistema Jakarta EE, executando no servidor de aplicações WildFly e integrando-se ao banco de dados relacional PostgreSQL. O projeto segue uma arquitetura em camadas, com código Java no back-end (camadas de negócio e persistência) e páginas JSF/PrimeFaces na camada de interface com o usuário.

Como o módulo é integrado ao Marvin, ele reutiliza estruturas e componentes-base da plataforma já estabelecida (por exemplo, classes e serviços do JButler/Marvin na camada de persistência). A Figura **??** ilustra classes-base reutilizadas no contexto de entidades persistentes do módulo.

A Tabela 1 apresenta as principais tecnologias utilizadas e o papel de cada uma na solução.

Além dessas tecnologias principais, foram utilizados softwares de apoio para mode-

Tabela 1 – Plataforma de desenvolvimento e tecnologias utilizadas.

Tecnologia	**Versão**	**Descrição**	**Propósito**
Java (JDK)	17	Linguagem de programação orientada a objetos voltada para aplicações corporativas.	Implementação das classes de domínio, serviços de aplicação e componentes de persistência.
Jakarta EE Web Profile	10	Conjunto de especificações para desenvolvimento de aplicações corporativas Java (CDI, EJB Lite, JPA, Bean Validation, JSF, entre outras).	Base da infraestrutura da aplicação, fornecendo APIs padronizadas para controle transacional, injeção de dependências, persistência e interface web.
WildFly	29	Servidor de aplicações compatível com Jakarta EE.	Ambiente de execução, gerenciamento de ciclos de vida e serviços de infraestrutura (transações, segurança, pool de conexões etc.).
PrimeFaces	13	Biblioteca de componentes JSF para construção de interfaces web ricas.	Implementação dos formulários, listas e diálogos de interação com usuários do módulo de bolsas.
Hibernate / JPA	—	Implementação da especificação Jakarta Persistence (JPA).	Mapeamento objeto–relacional das entidades de domínio e execução de operações de acesso a dados.
PostgreSQL	15	Sistema gerenciador de banco de dados relacional de código aberto.	Armazenamento das entidades do sistema (editais, inscrições, classificações, bolsistas, notificações etc.).
Maven	3.9	Ferramenta de automação e gerenciamento de dependências para projetos Java.	Empacotamento do módulo de bolsas, gerenciamento de bibliotecas externas e integração com o processo de build e deploy.
Git / GitLab	—	Sistema de controle de versão distribuído e plataforma de hospedagem de repositórios.	Versionamento do código-fonte, colaboração entre desenvolvedores e registro de issues e merge requests.

lagem, documentação e administração da plataforma, listados na Tabela 2.

Essa combinação de tecnologias garante alinhamento com o núcleo do Marvin e com as boas práticas adotadas no laboratório, favorecendo reuso de componentes, facilidade de manutenção e suporte futuro à evolução do subsistema.

Tabela 2 – Softwares de apoio ao desenvolvimento.

Ferramenta	**Versão**	**Descrição**	**Propósito**
IntelliJ IDEA	2024.x	IDE para desenvolvimento em Java.	Edição, refatoração e depuração do código back-end.
pgAdmin	7.x	Interface gráfica para administração de bancos PostgreSQL.	Criação de bancos, execução de consultas SQL e inspeção de dados de teste.
GitLab CI/CD	—	Serviço de integração contínua no GitLab baseado em pipelines.	Automatização de builds, testes e verificação de qualidade do código do módulo de bolsas.

arquiteturais. Nesta seção, são apresentados os principais RNFs priorizados para o subsistema, organizados em categorias como desempenho, usabilidade, integração, rastreabilidade, disponibilidade e segurança.

Cada requisito é descrito com sua categoria, a tática (ou tratamento) adotada na solução, como ele será medido e os critérios de aceitação que delimitam seu atendimento.

3.1 RNF-01 – Desempenho e tempo de resposta

Categoria: Desempenho.

Tática / Tratamento:

- Utilizar consultas JPA otimizadas (projeções, *fetch* adequado e filtros) e paginação nas telas com grandes volumes de inscrições e bolsistas.
- Delegar operações mais custosas (por exemplo, geração de relatórios consolidados) a serviços específicos, evitando processamento excessivo na *thread* de requisição.
- Definir índices nas colunas mais consultadas (por exemplo: edital, status, matrícula, orientador e datas), conforme padrão adotado no Marvin.

Medida: Medição do tempo de resposta de operações típicas (listagem de editais, abertura de tela de classificação, consulta de bolsistas e exportação) em ambiente de teste com carga representativa.

Critério de aceitação: Para operações interativas (formulários e listagens), o tempo de resposta médio não deve ultrapassar 2 segundos sob carga normal; para geração de relatórios, o tempo deve ser inferior a 10 segundos para um edital com até 500 inscrições.

3.2 RNF-02 – Usabilidade e consistência de interface

Categoria: Usabilidade.

Tática / Tratamento:

- Reutilizar templates e padrões visuais consolidados no Marvin (AdminFaces e componentes PrimeFaces), preservando consistência de navegação.
- Manter rotulagem consistente de campos e ações, alinhada à terminologia utilizada pela secretaria e pela comissão de bolsas.
- Minimizar a quantidade de passos necessários para tarefas frequentes (abrir edital, classificar inscrições, converter em bolsista, encerrar bolsa).

Medida: Observação de usuários representando secretaria e comissão durante a execução de cenários de teste pré-definidos, registrando tempo para completar tarefas e número de erros.

Critério de aceitação: Usuários novos devem conseguir completar os principais cenários de trabalho sem ajuda externa, com tempo médio inferior ao dobro do tempo de um usuário familiarizado e com baixa taxa de erros por tarefa.

3.3 RNF-03 – Integração com o núcleo do Marvin

Categoria: Integração / Interoperabilidade.

Tática / Tratamento:

- Implementar o subsistema de bolsas como um módulo integrado ao Marvin, seguindo o mesmo padrão de organização por pacotes e camadas.
- Reutilizar mecanismos do núcleo (autenticação/autorização, controle de sessão, identidade de usuários), evitando cadastros paralelos de dados institucionais.
- Integrar com infraestrutura já existente (por exemplo, serviço de e-mail do Marvin) para disparo de notificações do módulo.

Medida: Revisão arquitetural e inspeção de código, verificando o uso consistente dos serviços do núcleo, bem como a ausência de duplicação desnecessária de entidades de usuários/alunos.

Critério de aceitação: Todas as telas do módulo devem respeitar autenticação/autorização do Marvin, e não deve existir nenhuma estrutura redundante que replique dados institucionais já mantidos pelo núcleo.

3.4 RNF-04 – Rastreabilidade e auditoria

Categoria: Auditabilidade.

Tática / Tratamento:

- Registrar, para entidades críticas (edital, inscrição, classificação e bolsista), informações de criação e última atualização (data/hora e usuário responsável), quando aplicável.
- Registrar mudanças de estado relevantes (abertura/encerramento de edital, publicação/homologação de resultado, concessão e encerramento de bolsa) por meio de histórico próprio e/ou log estruturado.

Medida: Inspeção dos registros gerados durante a execução de um ciclo completo (criação do edital, inscrições, classificação, conversão em bolsista e encerramento), verificando se é possível reconstruir a sequência de decisões e alterações.

Critério de aceitação: Dado um edital e um bolsista, deve ser possível identificar quem realizou cada operação crítica, em que data, e qual era o estado anterior e posterior à operação (quando aplicável).

3.5 RNF-05 – Disponibilidade e recuperação

Categoria: Confiabilidade / Disponibilidade.

Tática / Tratamento:

- Utilizar recursos transacionais do contêiner (Jakarta EE/WildFly) e validações de integridade na camada de domínio para evitar estados inconsistentes.
- Apoiar-se em mecanismos de backup e restauração do PostgreSQL para recuperação de dados em caso de falhas.

Medida: Execução de cenários de falha simulada (rollback transacional, indisponibilidade momentânea do banco) e verificação da consistência dos dados após recuperação.

Critério de aceitação: Falhas pontuais não devem deixar inscrições ou bolsas em estados parcialmente atualizados; após restauração de backup, o sistema deve retomar o funcionamento sem necessidade de correções manuais extensas.

3.6 RNF-06 – Segurança e controle de acesso

Categoria: Segurança.

Tática / Tratamento:

- Reutilizar o mecanismo de autenticação e autorização do Marvin, baseado em papéis (perfis) de usuário.
- Separar claramente permissões de secretaria, comissão, coordenação, orientadores e estudantes, restringindo operações sensíveis por perfil.
- Validar dados no servidor e evitar exposição desnecessária de informações sensíveis em URLs, parâmetros e respostas.

Medida: Execução de testes manuais de acesso indevido (tentativas de operações com perfis sem permissão) e inspeções de código focadas em pontos de entrada críticos.

Critério de aceitação: Nenhuma operação de gestão de editais, classificação ou vínculo de bolsa deve ser acessível a usuários sem o perfil adequado; inscrições e dados pessoais de candidatos devem ser visíveis apenas para atores autorizados. 4

Arquitetura de Software

Esta seção apresenta a arquitetura de software do Sistema de Gestão de Bolsas e sua integração com o núcleo do Marvin. A arquitetura segue a combinação de **Camadas** e **Partições** adotada no Marvin: cada subsistema é uma partição lógica independente dentro do monólito, subdividida em camadas de Interface com o Usuário (CIU), Lógica de Negócio (CLN) e Gerência de Dados (CGD).

O subsistema de bolsas é implementado sob o pacote `br.ufes.inf.labes.marvin.scholarshi` respeitando a convenção de pacotes do *core*. Em alto nível, a solução é monolítica e modularizada, implantada como parte do `marvin.war` no servidor WildFly, compartilhando o mesmo contexto de segurança, sessão e infraestrutura de persistência do sistema.

4.1 Visão geral

A arquitetura do subsistema de bolsas está organizada em três camadas principais (além do banco de dados), com dependências unidirecionais entre camadas adjacentes:

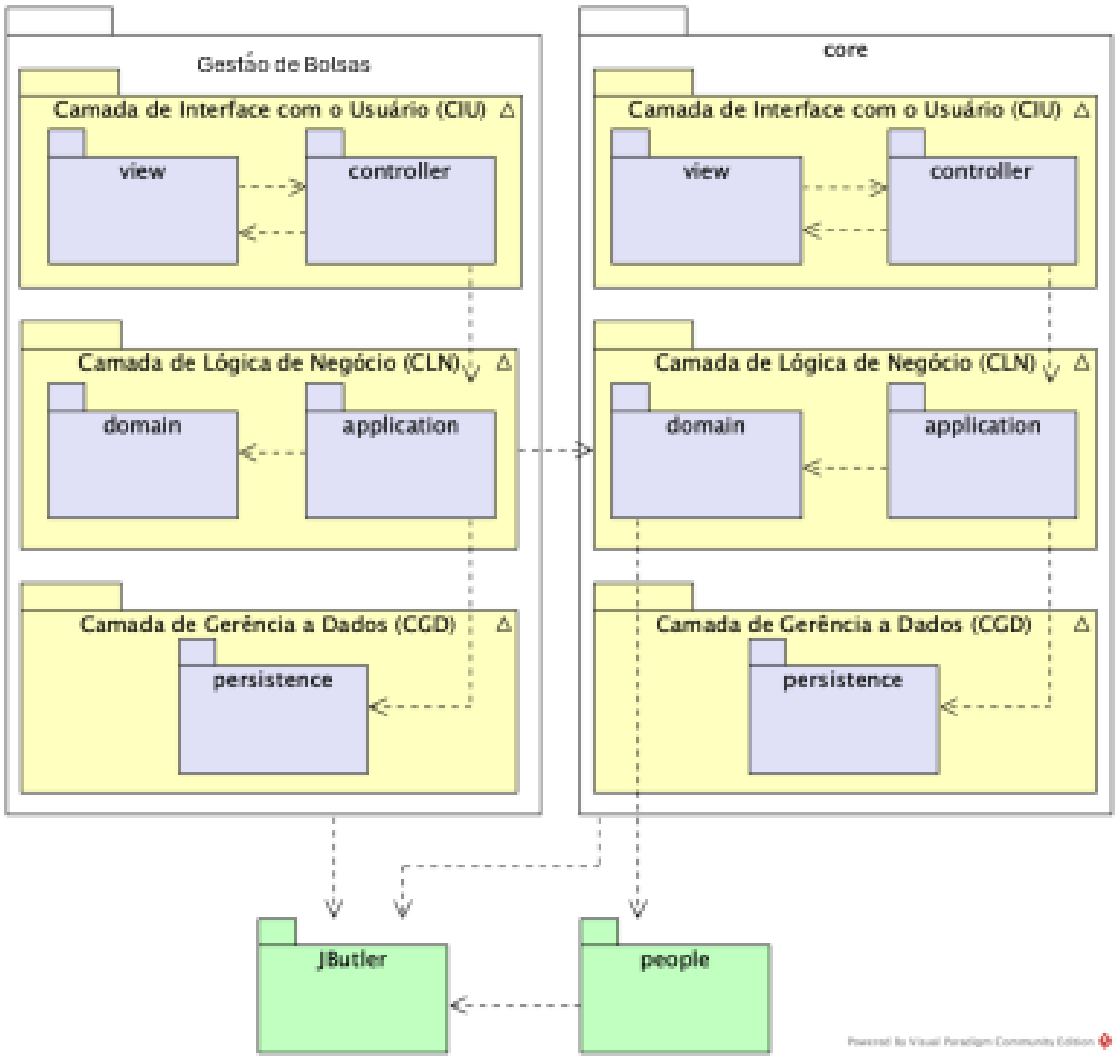

Figura 1 – Visão de partições (subsistemas) no Marvin e o posicionamento do subsistema de bolsas.

1. **Camada de Interface com o Usuário (CIU)**: composta por páginas JSF (*Facelets*) e controladores JSF (por exemplo, *managed beans view-scoped*), responsáveis pela interação com secretarias, comissões, coordenação, orientadores e estudantes.

2. **Camada de Lógica de Negócio (CLN)**: formada por classes de domínio e serviços de aplicação que encapsulam regras do processo (gestão de editais, inscrição, classificação, concessão e encerramento de bolsas), coordenando transações e validações.

3. **Camada de Gerência de Dados (CGD)**: implementada por interfaces DAO e classes *JPADAO*, responsáveis por consultas e operações de persistência no PostgreSQL via JPA/Hibernate.

Cada camada depende apenas da camada imediatamente inferior, favorecendo baixo acoplamento e alta coesão. Assim, a CIU depende exclusivamente de serviços da CLN; a CLN depende de DAOs da CGD e de serviços do núcleo do Marvin; e a CGD depende do mapeamento objeto–relacional e das configurações de persistência (`persistence.xml` e recursos do contêiner).

4.2 Partição do subsistema de bolsas

O subsistema de bolsas é tratado como uma partição dentro do pacote base do Marvin, em paralelo às demais partições (por exemplo, núcleo e administração). Essa partição é decomposta nos seguintes pacotes, alinhados às camadas:

- `scholarship.view`: recursos visuais (páginas `xhtml`, imagens, folhas de estilo e demais artefatos de interface específicos do módulo);
- `scholarship.controller`: controladores (*managed beans*) que orquestram a interação entre a interface e os serviços de aplicação;
- `scholarship.application`: serviços (EJBs) e classes auxiliares que implementam funcionalidades do sistema (casos de uso), coordenando transações e validações;
- `scholarship.domain`: entidades de domínio (por exemplo, edital, inscrição, classificação, bolsista, notificação) e *enums* associados (status e categorias);
- `scholarship.persistence`: interfaces DAO e implementações *JPADAO*, especializadas a partir das abstrações oferecidas pelo JButler.

Essa organização segue a abordagem FrameWeb adaptada ao Marvin, na qual as camadas de interface, aplicação, domínio e persistência são explicitamente separadas, porém mantidas sob um único módulo lógico. Dependências com o núcleo devem ser preferencialmente feitas na camada de lógica de negócio (serviços e/ou domínio), mantendo o núcleo independente do subsistema de bolsas.

4.3 Padrões arquiteturais e decisões de projeto

Os seguintes padrões arquiteturais orientam a implementação do subsistema:

- **Modelo–Visão–Controle (MVC)** na CIU: páginas `xhtml` atuam como Visão; controladores JSF atuam como Controle; o Modelo é representado por entidades de domínio e serviços de aplicação.

- **Service Layer** na CLN: serviços de aplicação encapsulam regras do negócio e coordenam transações, evitando concentração de lógica em controladores ou DAOs e reduzindo risco de domínio anêmico.
- **DAO/Repository** na CGD: DAOs derivadas de `BaseDAO` e implementações *JPA-DAO* derivadas de `BaseJPADAO` provêm operações genéricas (CRUD) e consultas específicas por entidade.
- **Arquitetura em camadas**: cada camada expõe interfaces estáveis para a camada superior e oculta detalhes de implementação, facilitando evolução, teste e manutenção.

Decisões específicas, como o uso de *enums* para estados de edital e bolsa, configurações de carregamento preguiçoso (*lazy loading*) em associações, definição de chaves e detalhes do mapeamento JPA, são detalhadas no Capítulo 5, ao descrever o projeto dos componentes da arquitetura. 5

Projeto dos Componentes da Arquitetura

Este capítulo detalha os principais componentes do Sistema de Gestão de Bolsas, organizados pelas camadas da arquitetura descrita no Capítulo 4. O objetivo é evidenciar como entidades, serviços, controladores e DAOs concretizam os casos de uso e requisitos levantados no documento de requisitos, preservando a organização em camadas e a integração com o núcleo do Marvin.

5.1 Camada de domínio

A camada de domínio (`scholarship.domain`) reúne as classes que representam os conceitos centrais do subsistema e as regras de consistência associadas (por exemplo, status, categorias e restrições de datas). Entre as principais entidades, destacam-se:

- `ScholarshipNotice` (Edital): representa um edital de bolsas, com atributos como título, descrição/observações, período de inscrição, periodicidade de relatórios, status e referência para o anexo (por exemplo, PDF oficial);

- **`ScholarshipApplication`** (Inscrição): registra os dados do candidato (nome, e-mail, matrícula, curso e nível), dados relevantes para seleção (por exemplo, categoria e faixa de renda), data de submissão e status da candidatura;
- **`ScholarshipApplicationClassification`** (Classificação): guarda pontuação, posição e situação no resultado (classificado, lista de espera, indeferido) para cada inscrição de um edital;
- **`ScholarshipStudent`** (Bolsista): modela o vínculo de bolsa, com datas de início, término previsto e término efetivo, status do vínculo e associação a um orientador;
- **`ScholarshipInstructor`** (Orientador): representa docentes vinculados a bolsistas, contendo dados de identificação e o vínculo institucional necessário;
- **`ScholarshipNotification`** (Notificação): registra notificações associadas a eventos do ciclo (por exemplo, concessão, lembretes e encerramento), incluindo data/hora de envio, canal e status de entrega;
- **`ScholarshipQuotaConfig`**: *value object* para representar a configuração de vagas por nível e categoria em cada edital.

Além das entidades, a camada de domínio inclui *enums* que expressam estados e categorias relevantes, como `StatusScholarshipNotice`, `StatusScholarshipApplication`, `StatusScholarship`, `ScholarshipLevel`, `ScholarshipCategory` e `ScholarshipIncomeBracket`. Esses tipos fortalecem a clareza da lógica de negócio e reduzem inconsistências de preenchimento.

5.2 Camada de aplicação

A camada de aplicação (`scholarship.application`) concentra serviços EJB que implementam os casos de uso descritos nas estórias de usuário. As interfaces de serviço definem contratos estáveis para a camada superior (CIU), e as implementações (`*ServiceBean`) coordenam transações, validações e o acesso aos DAOs. Entre os serviços principais, destacam-se:

- **`ManageScholarshipNoticeService`**: gerencia o ciclo de vida dos editais (criar, abrir, encerrar) e consultas consolidadas;
- **`ApplicationFormService`**: recebe e valida inscrições, verificando período do edital, duplicidade de matrícula por edital e consistência de dados/anexos;
- **`ScholarshipApplicationClassificationService`**: coordena a classificação de

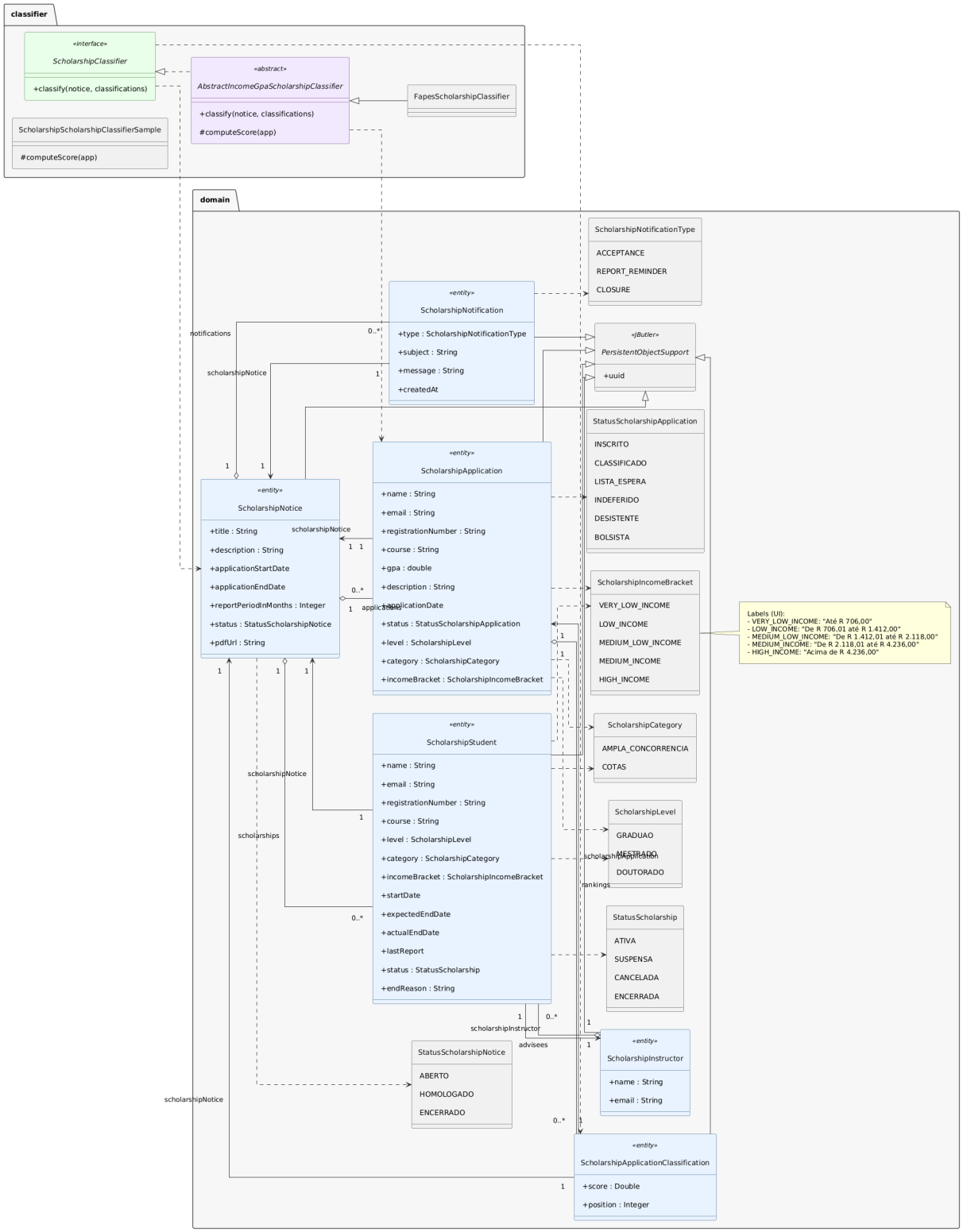

Figura 2 – Modelo de Classes de Domain

inscrições, aplicando regras e cotas por nível e categoria, e gerando listas de classificados e lista de espera;

- **`ManageScholarshipStudentInformationService`**: trata a gestão do vínculo de bolsistas (concessão a partir de inscrição classificada, atualização de status e encerramento com motivo e data efetiva);
- **`ScholarshipNotificationService`**: registra e dispara notificações associadas a eventos do ciclo (concessão, prazos e encerramento), utilizando a infraestrutura de e-mail/notificações do Marvin quando aplicável.

Esses serviços utilizam Bean Validation para garantir integridade dos dados e operam sob transações gerenciadas pelo contêiner, assegurando consistência entre a lógica de negócio e a camada de persistência.

5.3 Camada de interface com o usuário

A camada de interface (`scholarship.controller` e `scholarship.view`) é responsável pela interação com os usuários finais. Os controladores são *managed beans* JSF (tipicamente com escopo de visão), que expõem modelos e ações às páginas `xhtml`. Alguns controladores relevantes são:

- **`ManageScholarshipNoticeController`**: fornece visão geral dos editais, exibe métricas consolidadas (inscrições, bolsistas) e aciona ações como abertura/encerramento e execução da classificação;
- **`ApplicationFormController`**: controla o formulário de inscrição em editais abertos, com validações de preenchimento e seleção do edital de destino;
- **`ManageScholarshipStudentInformationController`**: permite consultar e atualizar informações do vínculo de um bolsista (atribuição de orientador, alteração de status e encerramento com motivo e data efetiva);
- **Controladores auxiliares**: dedicados a listagens, filtros, exportação de relatórios e operações de apoio.

As páginas JSF utilizam componentes PrimeFaces para formulários, tabelas, diálogos e mensagens, seguindo o padrão visual do Marvin (template base, breadcrumbs e menu). A navegação entre telas é realizada por regras declarativas e comandos JSF.

A Figura 3 apresenta o diagrama UML da Camada de Interface com o Usuário do módulo, distinguindo os elementos por **tipo, cor e estereótipo**: as **Views** (páginas

XHTML) são representadas por círculos azuis e anotadas com «view» (V_MENU, V_FORM, V_NOT, V_RES, V_STU, V_INF); os **Controllers** JSF são representados por círculos em cor diferenciada e anotados com «controller» (C_FORM, C_NOT, C_RES, C_STU, C_INF); e as **interfaces de serviço** da camada application são representadas por caixas e anotadas com «service» (S_FORM, S_NOT, S_CLS, S_RES, S_STU, S_INF). As siglas utilizadas no diagrama são definidas na Tabela 3.

Tabela 3 – Abreviações utilizadas nos diagramas do módulo de Gestão de Bolsas.

Categoria	Abreviação	Elemento (nome completo)
View (XHTML)	V_MENU	menu_scholarship.xhtml
View (XHTML)	V_FORM	applicationForm/index.xhtml
View (XHTML)	V_NOT	manageScholarshipNotice/index.xhtml
View (XHTML)	V_RES	manageScholarshipApplication/index.xhtml
View (XHTML)	V_STU	manageScholarshipStudent/index.xhtml
View (XHTML)	V_INF	manageScholarshipStudentInformation/index.xhtml
Controller	C_FORM	ApplicationFormController
Controller	C_NOT	ManageScholarshipNoticeController
Controller	C_RES	ManageScholarshipApplicationController
Controller	C_STU	ManageScholarshipStudentController
Controller	C_INF	ManageScholarshipStudentInformationController
Service	S_FORM	ApplicationFormService
Service	S_NOT	ManageScholarshipNoticeService
Service	S_CLS	ScholarshipApplicationClassificationService
Service	S_RES	ManageScholarshipApplicationService
Service	S_STU	ManageScholarshipStudentService
Service	S_INF	ManageScholarshipStudentInformationService
Service	S_NTF	ManageScholarshipNotificationService

5.4 Camada de gerência de dados

A camada de gerência de dados (scholarship.persistence) é composta por interfaces DAO e classes *JPADAO* que estendem abstrações fornecidas pelo JButler. Para cada entidade de domínio relevante existe, em geral, um par DAO/JPADAO:

- ScholarshipNoticeDAO / ScholarshipNoticeJPADAO;
- ScholarshipApplicationDAO / ScholarshipApplicationJPADAO;
- ScholarshipApplicationClassificationDAO / ScholarshipApplicationClassificationJP

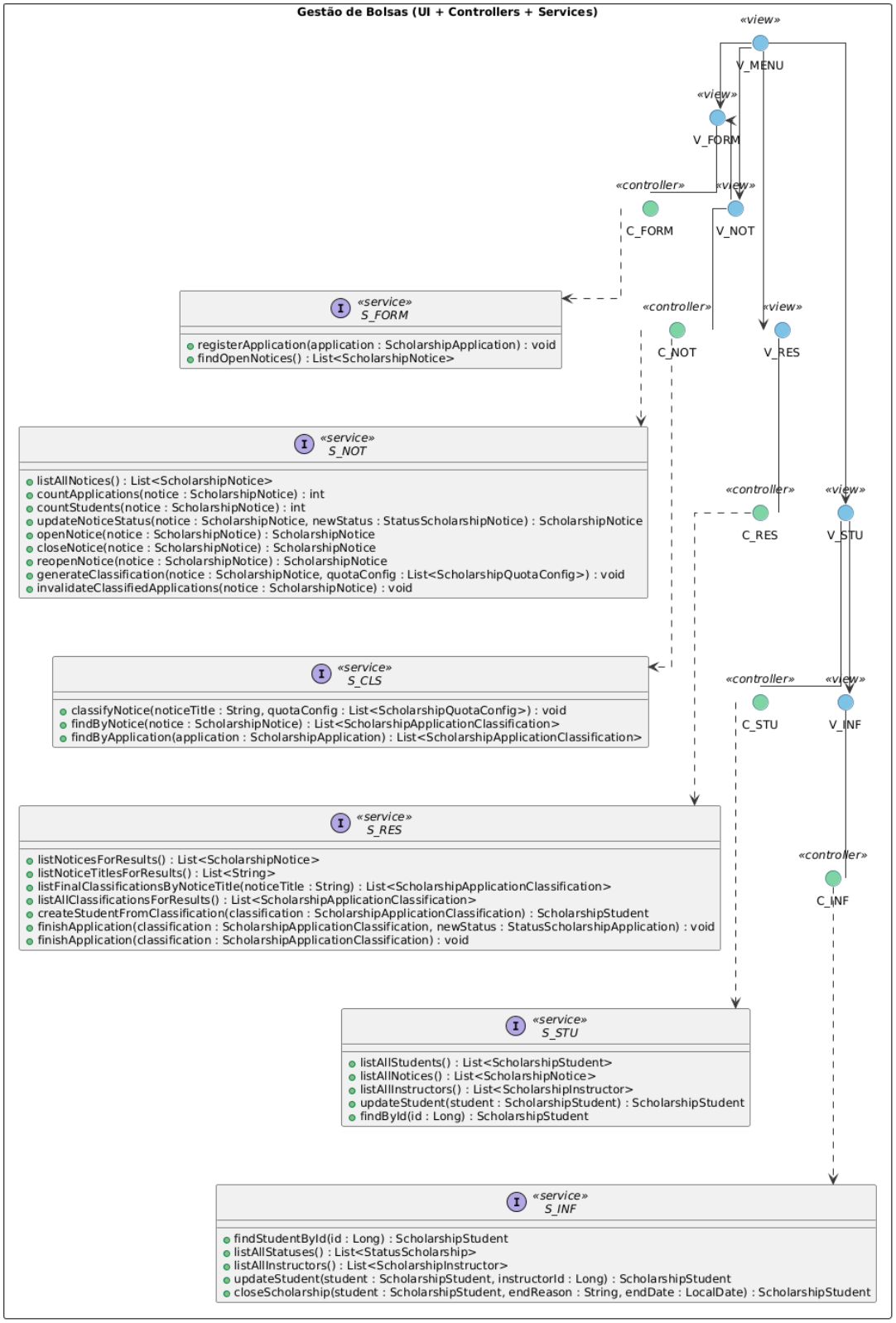

Figura 3 – Projeto da Camada de Interface com o Usuário do módulo de Gestão de Bolsas (view e controller), em dependência com interfaces da camada application.

- `ScholarshipStudentDAO` / `ScholarshipStudentJPADAO`;
- `ScholarshipInstructorDAO` / `ScholarshipInstructorJPADAO`;
- `ScholarshipNotificationDAO` / `ScholarshipNotificationJPADAO`.

As interfaces declaram operações específicas (por exemplo, listar inscrições por edital e status, recuperar bolsistas ativos, listar classificações por edital), enquanto as implementações JPADAO concretizam as consultas via JPA, reutilizando operações genéricas herdadas de `BaseJPADAO` (buscar por ID, listar, paginar, salvar e remover) e se beneficiando do contexto de persistência gerenciado pelo WildFly.

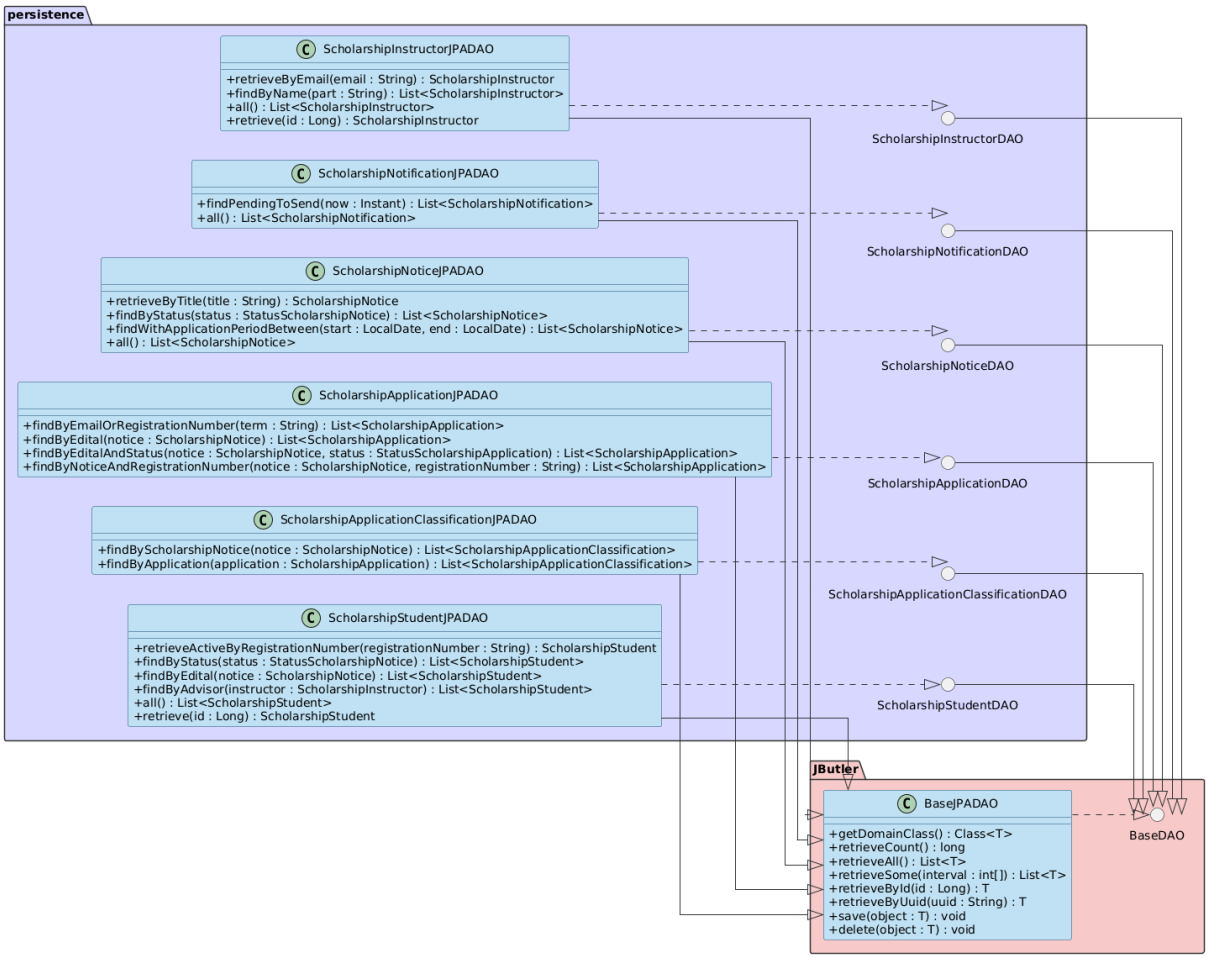

Figura 4 – Projeto da camada de persistência do módulo de bolsas (DAOs e implementações JPADAO).

5.5 Banco de dados e mapeamento JPA

O esquema relacional no PostgreSQL reflete o modelo estrutural definido para o subsistema, com tabelas correspondentes às entidades `ScholarshipNotice`, `ScholarshipApplication`, `ScholarshipApplicationClassification`, `ScholarshipStudent`, `ScholarshipInstructor` e `ScholarshipNotification`. O mapeamento JPA utiliza:

- chaves substitutas (*surrogate keys*) do tipo `Long` para a maioria das entidades;
- associações `@OneToMany` e `@ManyToOne` entre edital–inscrição e inscrição–classificação;
- relacionamentos entre bolsista e o edital/inscrição que originaram o vínculo, bem como entre bolsista e orientador;
- colunas enumeradas para estados e categorias, mapeadas via `@Enumerated`.

Esse conjunto de componentes concretiza, na implementação, as decisões arquiteturais definidas nos capítulos anteriores, garantindo alinhamento entre o modelo conceitual, a arquitetura em camadas e os requisitos de qualidade especificados para o subsistema de bolsas.

Referências