UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO DE ENGENHARIA DE SOFTWARE

MARIA LANUZA DOS SANTOS SILVA

UMA ARQUITETURA DE UM SISTEMA PARA O *MENTORING* NA UFERSA

PAU DOS FERROS - RN
2025

MARIA LANUZA DOS SANTOS SILVA

UMA ARQUITETURA DE UM SISTEMA PARA O MENTORING NA UFERSA

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Engenharia de Software.

Orientador: Reudismam Rolim de Sousa, Prof. Dr.

PAU DOS FERROS - RN

2025

S586a Silva, Maria Lanuza dos Santos.
Uma arquitetura de um sistema para o mentoring na UFERSA / Maria Lanuza dos Santos Silva. - 2025.
63 f. : il.

Orientador: Reudismam Rolim de Sousa.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia de Software, 2025.

1. Mentoria. 2. Arquitetura de software. 3. Modelagem de sistema. 4. Gestão do conhecimento. 5. Engenharia de software. I. Sousa, Reudismam Rolim de, orient. II. Título.

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

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

UMA ARQUITETURA DE UM SISTEMA PARA O MENTORING NA UFERSA

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Engenharia de Software.

Defendida em: **17 / 12 / 2025**.

BANCA EXAMINADORA

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

_____
Hortência Pessoa Rêgo Gomes, Prof. Ma. (UFERSA)
Membro Examinador

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

DEDICATÓRIA

Francisca Lopes dos Santos (In Memoriam).

“Um coração gentil vence todas as tempestades da vida.” - Ajahn Chah

AGRADECIMENTOS

Em primeiro lugar, expresso minha profunda gratidão a Deus, a Buda e ao Universo, pelas forças invisíveis que me guiaram e sustentaram ao longo desta jornada. Sua presença me deu força, inspiração e resiliência para superar todos os desafios e concluir este trabalho.Reconheço, com humildade, que cada passo dado foi iluminado por essa energia maior, que acolheu minhas dúvidas, acalmou meus medos e renovou minha esperança nos momentos mais difíceis. Como ensina Buda, “a paz vem de dentro; não a procure fora”, e foi nessa serenidade interior que encontrei clareza para seguir em frente.

Agradeço imensamente à minha família, que sempre esteve ao meu lado, oferecendo apoio incondicional, incentivo e amor. Sem vocês, nada disso seria possível. Agradeço especialmente minha mãe Maria de Lourdes e minha tia Maria do Socorro por toda dedicação e ajuda durante minha jornada. Também sou grata aos meus amigos próximos, Marcos Jhonaths de Oliveira, Caio Vinicius Pessoa Gomes e Maria Helena Ferreira de Oliveira, por serem pilares de amizade, companheiros de risadas e discussões que enriqueceram minha vida acadêmica e pessoal. Obrigado por acreditarem em mim e por compartilharem momentos que tornaram tudo mais leve, sem o apoio de vocês não teria sido possível.

Um agradecimento especial ao meu orientador, Prof. Dr. Reudismam, pela orientação dedicada, pelos conselhos valiosos e pelo apoio constante que foram fundamentais para o desenvolvimento deste TCC. Por fim, agradeço a todos os professores, colegas e demais colaboradores que contribuíram direta ou indiretamente para este trabalho. Este TCC é dedicado a todos vocês.

RESUMO

Este trabalho propõe e descreve uma Arquitetura de Software para a plataforma do projeto Mentoring da UFERSA, cujo objetivo é apoiar e gerenciar programas de mentoria acadêmica. Baseado em pesquisa-ação, o estudo realizou levantamento de requisitos (funcionais e não funcionais), modelagem UML e a proposição de três visões arquiteturais (visão de módulo, visão componente & conector e visão de alocação), conforme a abordagem de Bass, Clements e Kazman. A arquitetura foi avaliada por meio de cenários representativos que abordaram atributos de qualidade como segurança, escalabilidade e compatibilidade. Os artefatos produzidos como casos de uso, diagramas de classes, componentes, cenários de avaliação e especificações de requisitos, validam que a proposta atende às necessidades de registro, comunicação e gestão das atividades de mentoria, além de oferecer base sólida para implementação futura. Conclui-se que a solução proposta contribui para a formalização, automação e escalabilidade do projeto Mentoring, promovendo melhores práticas de engenharia de software em contexto educacional.

Palavras-chave: mentoria; arquitetura de software; modelagem de sistema; gestão do conhecimento; engenharia de software.

ABSTRACT

This work presents and describes a Software Architecture proposal for the Mentoring project platform at UFERSA, whose purpose is to support and manage academic mentoring activities. Based on an action-research methodology, the study conducted a requirements elicitation process (functional and non-functional), UML modeling, and the development of three architectural views, module, component & connector, and allocation, following the approach proposed by Bass, Clements, and Kazman. The architecture was evaluated through representative scenarios addressing quality attributes such as security, scalability, and compatibility. The produced artifacts, including use cases, class diagrams, component diagrams, evaluation scenarios, and requirements specifications, demonstrate that the proposed solution meets the needs of registration, communication, and management within the mentoring program, providing a robust foundation for future implementation. The results indicate that the proposed architecture contributes to the formalization, automation, and scalability of the Mentoring project, promoting good software engineering practices in an educational context.

Keywords: mentoring; software architecture; modeling system; knowledge management; software engineering.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

GC	Gestão do Conhecimento
Dr	Doutor
CSS3	Cascading Style Sheets
SOLID	Conjunto de princípios de design (SRP, OCP, LSP, ISP, DIP)
RF	Requisito Funcional
HTML5	HyperText Markup Language
SBGC	Sociedade Brasileira de Gestão e Conhecimento
PBL	Aprendizagem baseada em problemas
TICs	Tecnologias da Informação e Comunicação
UFERSA	Universidade Federal Rural do Semi-árido
UML	Linguagem de Modelagem Unificada

SUMÁRIO

1 INTRODUÇÃO

A mentoria tem suas raízes na Antiguidade, sendo associada à prática de transmissão de conhecimento entre mestres e aprendizes em diferentes áreas do saber, como o artesanato, a filosofia e a formação de líderes, configurando-se como elemento essencial para a preservação e evolução cultural (Homero, 2000).

O termo “mentoria” deriva da obra Odisseia, de Homero, em que Ulisses, ao partir para a Guerra de Tróia, confia a educação de seu filho Telêmaco ao amigo Mentor. Com isso, Mentor consolidou-se como símbolo de guia e conselheiro e o conceito de mentoria passou a designar o processo de orientação e aconselhamento entre alguém mais experiente e outro em formação (Homero, 2000).

Com o decorrer do tempo, essa prática evolui, abrangendo diferentes áreas, como o universo acadêmico e corporativo, evidenciando sua importância para o crescimento individual e profissional do indivíduo (Kram, 1983). No contexto universitário, a mentoria vem se consolidando como uma estratégia para acolhimento e integração dos estudantes, especialmente os ingressantes. Além de fornecer adaptação, a mentoria promove o desenvolvimento de competências como pensamento crítico, comunicação e escuta empática (Sória; Macuch, 2025).

Pesquisas recentes mostram que a mentoria amplia o sentimento de pertencimento à comunidade universitária, fortalecendo os vínculos interpessoais e contribuindo na permanência dos discentes em cursos de graduação (Calsing; Heidemann, 2023). Neste contexto, a prática não é limitada ao auxílio acadêmico, mas integra dimensões psicossociais e éticas, configurando-se como um espaço de diálogo e acolhimento que potencializa o processo formativo (Carvajal, 2024).

O projeto *Mentoring* consiste em uma estratégia institucional de apoio e desenvolvimento acadêmico baseada no modelo de peer mentoring, no qual estudantes mais experientes (mentores) acompanham e orientam alunos ingressantes (mentees) durante sua adaptação ao ambiente universitário. A iniciativa busca promover o acolhimento, a integração e a permanência discente, atuando de forma preventiva frente a desafios como evasão, retenção e dificuldades emocionais nos semestres iniciais. Por meio de encontros planejados e da mediação colaborativa, o projeto estimula o desenvolvimento de competências acadêmicas, profissionais e socioemocionais, fortalecendo a autonomia, o senso de pertencimento e o protagonismo estudantil. Dessa forma, o *Mentoring* configura-se como uma ferramenta

complementar à formação acadêmica, contribuindo para uma trajetória universitária mais saudável, integrada e eficaz.

No contexto das mentorias, o *Mentoring* é um projeto executado na Universidade Federal Rural do Semi-Árido (UFERSA), que consistente de encontros formativos ministrados por mentores para os discentes ingressantes em cursos de graduação, sob a supervisão de professores associados aos temas tratados nos encontros (Meneses *et al*., 2021). Os conteúdos ministrados pelos mentores são direcionados à resolução dos principais problemas dos mentorados, de modo a minimizar as taxas de insucesso dentro da graduação.

Ademais, o avanço das Tecnologias da Informação e Comunicação (TICs) têm impulsionado a transformação digital na educação. Segundo Moran (2015), a tecnologia pode transformar o aprendizado ao oferecer novas formas de ensino e interação. A busca por aprendizado e aperfeiçoamento é essencial para o desenvolvimento acadêmico e profissional dos estudantes.

No entanto, muitos profissionais e estudantes encontram dificuldades com suporte personalizado de maneira eficiente. Diante disso, este projeto propõe o desenvolvimento de uma Arquitetura de Software para o projeto *Mentoring*, oferecendo um ambiente estruturado para aprendizagem e orientação, que conecte alunos a mentores.

O desenvolvimento da Arquitetura do Software, foi baseada na proposta de Bass, Clements e Kazman (2021), que consiste no projeto da arquitetura por meio de três visões principais do software: a visão de módulo, que apresenta os principais elementos estruturais do software; a visão componente & conector, que busca apresentar os elementos em tempo de execução; e a visão de implantação, que apresenta o sistema em termos dos recursos a serem empregados para o desenvolvimento do sistema.

A arquitetura foi avaliada utilizando uma abordagem baseada em cenários, uma das etapas importantes nos métodos de avaliação de Arquiteturas de Software (Bass; Clements; Kazman, 2021). Para Pressman e Maxim (2021), a engenharia de software aplica métodos sistemáticos para transformar requisitos em soluções de qualidade, escaláveis e manuteníveis. Nesse cenário, a modelagem e a arquitetura são etapas essenciais para garantir que o sistema de mentoria universitária proposto neste trabalho seja desenvolvido de forma profissional, alinhando boas práticas de engenharia às necessidades educacionais identificadas.

1.2 Objetivo geral

O objetivo principal deste trabalho é o de desenvolver uma Arquitetura de *Software* para guiar o desenvolvimento de uma plataforma para o projeto *Mentoring*.

1.2.1 Objetivos específicos

Para o alcance do objetivo geral deste trabalho, definem-se como objetivos específicos o levantamento dos requisitos funcionais e não funcionais do sistema, a proposição de uma Arquitetura de *Software* para o projeto *Mentoring* e a avaliação da arquitetura proposta por meio de uma abordagem baseada em cenários. Ademais, contempla-se a elaboração da documentação necessária ao desenvolvimento do projeto, bem como a redação e consolidação do Trabalho de Conclusão de Curso.

1.3 Justificativa

A falta de suporte personalizado é um dos principais desafios enfrentados pelos estudantes na Educação Superior. Um sistema de mentorias pode suprir essa necessidade, garantindo acesso a conteúdo de qualidade, interação com especialistas e acompanhamento individualizado, promovendo maior inclusão educacional e melhor desempenho acadêmico.

Segundo Sommerville (2011), um planejamento estruturado é essencial para o sucesso do desenvolvimento de sistemas. Assim, a utilização de modelagem e Arquitetura de *Software* permitirá um planejamento adequado para a implementação futura, garantindo escalabilidade, usabilidade e eficiência.

O projeto *Mentoring* foi destaque pelos impactos positivos que tem causado na graduação. Gradualmente, a UFERSA vem expandindo a atuação do projeto para alcançar mais cursos, embora atualmente esteja concentrada no Campus sede, em Mossoró. Essa expansão reforça a necessidade de um sistema que acompanhe, registre e gerencie as atividades do projeto, permitindo maior controle e continuidade das ações. Além disso, programas de mentoria online em universidades brasileiras promovem o desenvolvimento de habilidades como empatia e escuta ativa (LIMA; COSTA, 2021).

Dessa forma, o desenvolvimento de um ambiente automatizado e bem estruturado torna-se fundamental para apoiar, gerenciar e fortalecer as ações do projeto, potencializando seus resultados acadêmicos e sociais.

1.4 Apresentação do Trabalho

Este trabalho propõe a modelagem arquitetural do sistema Mentoring, uma solução voltada à gestão da mentoria acadêmica mediada por tecnologia. A Seção 2 – Fundamentação Teórica apresenta os conceitos de Engenharia de Software, diagramas UML, Ensino Mediado por Tecnologia, Gestão do Conhecimento e Arquitetura de Software, incluindo os princípios SOLID que fundamentam a proposta.

Na Seção 3 – Trabalhos Relacionados, são discutidas iniciativas semelhantes ao tema abordado. A Seção 4 – Metodologia descreve os procedimentos adotados para o desenvolvimento da pesquisa, enquanto a Seção 5 – Planejamento apresenta os requisitos do sistema, seus atores, restrições e casos de uso. A Seção 6 – Implementação detalha as visões arquiteturais do sistema, abrangendo as visões de módulo, componente & conector e alocação. Na Seção 7 – Avaliação, a arquitetura proposta é analisada por meio de cenários representativos. Por fim, a Seção 8 – Considerações Finais apresenta as conclusões do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho está embasada em quatro pilares essenciais: Engenharia de Software, Ensino Mediado por Tecnologia, Gestão do Conhecimento e Arquitetura de Software. Esses conceitos fornecem suporte teórico para o desenvolvimento do sistema proposto, garantindo que ele seja bem estruturado, eficiente e alinhado às necessidades dos usuários.

2.1 Engenharia de Software

De acordo com Sommerville (2007) e o material didático de Engenharia de *Software* da UAB/UFAL (Domínguez *et al*., 2010), o *software* não se limita ao programa executável, abrangendo também a documentação e os arquivos de configuração necessários ao seu funcionamento. Um sistema de *software* é composto por programas correlacionados, arquivos de configuração, documentação técnica que descreve sua estrutura interna, e documentação do usuário, que orienta quanto a sua utilização.

Segundo Pressman (2010), o *software* difere essencialmente do *hardware*, pois é um elemento lógico, desenvolvido e não fabricado, não sofrendo desgaste físico ao longo do tempo. Ainda assim, pode tornar-se obsoleto devido à evolução tecnológica e às mudanças de requisitos. Essas características evidenciam a necessidade de processos de engenharia que garantam sua qualidade e capacidade de adaptação.

O conceito de Engenharia de *Software* surgiu em 1968, em um contexto de instabilidade na produção de sistemas, marcado por atrasos, custos elevados e insatisfação com o desempenho das aplicações (Domínguez *et al*., 2010). Essa “crise do *software*” levou à criação de uma disciplina voltada à aplicação de princípios de engenharia ao desenvolvimento de programas.

Assim, a Engenharia de *Software* consolidou-se como uma área que abrange desde a especificação dos requisitos até a manutenção do sistema, envolvendo práticas técnicas, gerenciais e metodológicas (Sommerville, 2007). O processo de *software,* nesse contexto, é compreendido como um conjunto estruturado de atividades: especificação, desenvolvimento, validação e evolução. Tendo como objetivo principal assegurar a entrega de produtos confiáveis, eficientes e alinhados às necessidades do cliente.

A UML (Linguagem de Modelagem Unificada) é uma linguagem padrão amplamente utilizada na engenharia de *software* para representar graficamente a estrutura e o comportamento de sistemas. Segundo Fowler (2005), a UML não é uma metodologia de desenvolvimento, mas uma notação que pode ser aplicada em diferentes abordagens, principalmente aquelas orientadas a objetos. Seu objetivo central é facilitar a comunicação entre os profissionais envolvidos, garantindo um entendimento comum sobre o sistema em desenvolvimento.

Conforme Booch, Rumbaugh e Jacobson (2005), a UML 2 foi criada para oferecer uma visão completa do sistema, abrangendo tanto seus aspectos estruturais quanto comportamentais. Ela é composta por diversos diagramas que auxiliam na representação e análise do *software* em diferentes etapas do projeto. Entre os principais, destacam-se os diagramas de casos de uso, de classes e de sequência, cada um responsável por descrever uma perspectiva específica do sistema.

No contexto deste trabalho, que trata de um sistema educacional, a UML foi fundamental para representar de forma clara as interações entre administradores, professores, mentores e mentorados. De acordo com Booch, Rumbaugh e Jacobson (2005), o uso da UML reduz ambiguidades e melhora a comunicação durante o desenvolvimento, tornando-se uma ferramenta essencial para o planejamento e a manutenção de sistemas de software.

2.1.1 Diagrama de Casos de Uso

Segundo a especificação oficial da *Object Management Group* (OMG, 2017), o diagrama de casos de uso é um diagrama comportamental da UML cujo propósito é “modelar as funcionalidades fornecidas por um sistema, descrevendo o conjunto de interações possíveis entre os atores externos e o próprio sistema”. Visando representar o comportamento observável do sistema a partir da perspectiva do usuário, sem detalhar sua implementação interna.

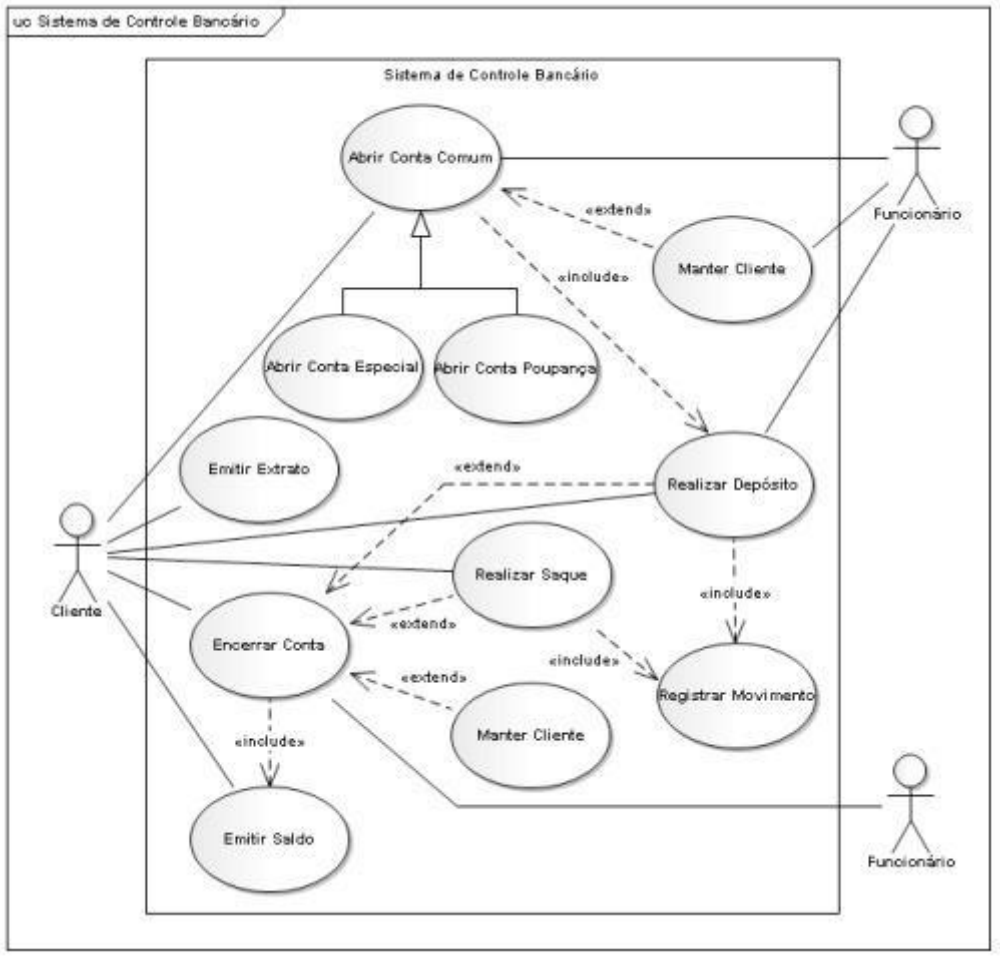
Conforme discutido por Booch, Rumbaugh e Jacobson (2005), esse tipo de diagrama tem papel essencial nas fases iniciais do desenvolvimento, uma vez que organiza e descreve as funcionalidades previstas do sistema e os atores que interagem com ele, auxiliando no alinhamento entre necessidades dos usuários e o que o sistema deverá oferecer. Assim, ele contribui para a compreensão compartilhada entre analistas e *stakeholders* acerca do que se espera do sistema em termos de comportamento.

Os elementos principais do diagrama de casos de uso incluem:

- Atores: entidades externas que interagem com o sistema (usuários, dispositivos ou outros sistemas);
- Casos de uso: representam os serviços ou funcionalidades oferecidas pelo sistema;
- Relações: associação, generalização, inclusão («*include*») e extensão («*extend*»).

A Figura 1 apresenta um exemplo de diagrama de casos de uso que ilustra as interações entre os atores e as principais funcionalidades do sistema de controle bancário, conforme discutido neste tópico.

Figura 1 - Exemplo de diagrama de Casos de Uso



Fonte: Guedes (2011)

2.1.2 Diagrama de Classes

O diagrama de classes é um diagrama estrutural da UML que "mostra as classes do sistema, seus atributos, métodos e os relacionamentos estáticos entre elas" (Larman, 2007, p. 147). É considerado um dos diagramas mais importantes, pois descreve a estrutura estática do sistema e serve como base para a implementação orientada a objetos.

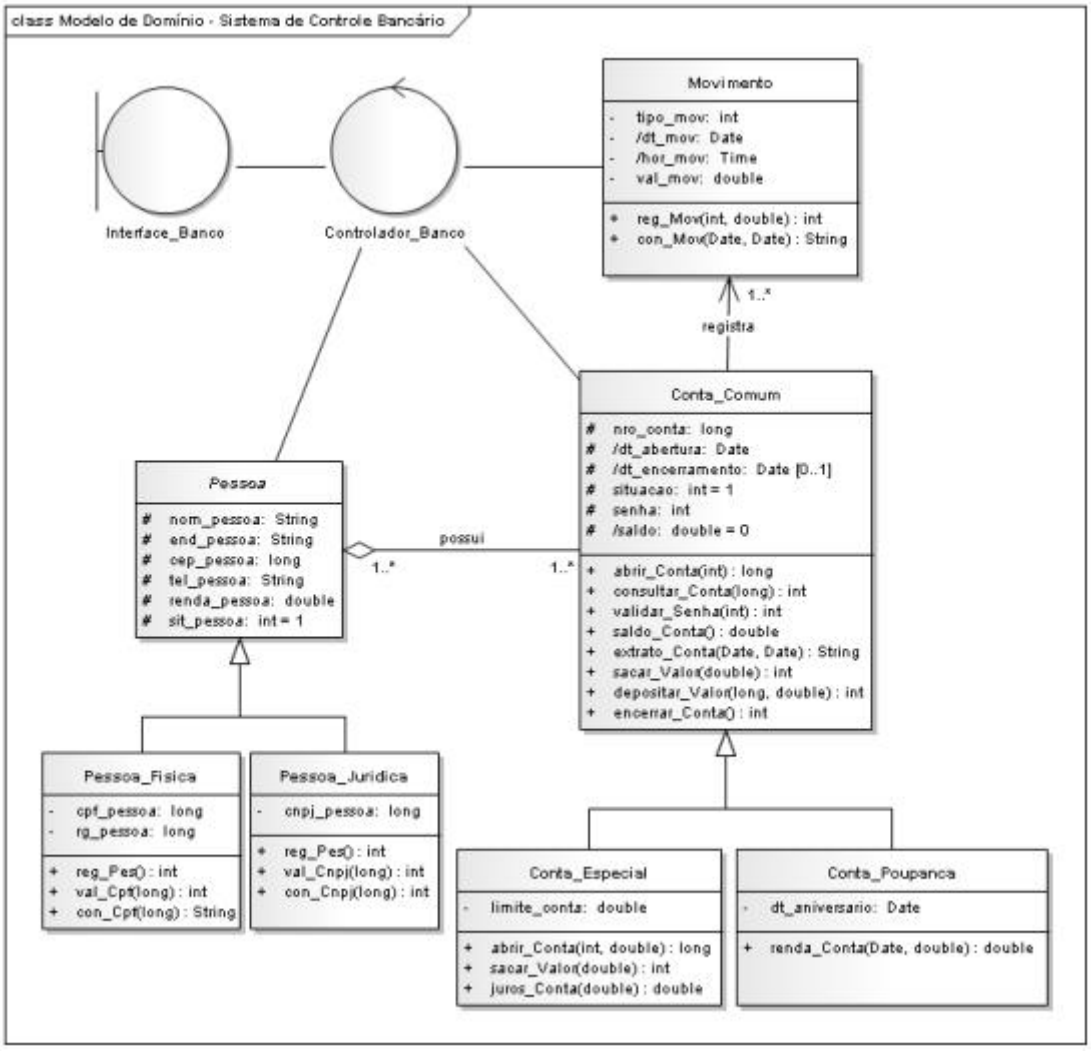
Além disso, esse diagrama revela como as classes se conectam e formam o "esqueleto" do sistema, sendo essencial para compreender a arquitetura e o reuso de componentes.

Os elementos principais de um diagrama de classes incluem os seguintes:

- Classes: representadas por retângulos divididos em três partes (nome, atributos e operações);
- Relacionamentos: associação, generalização (herança), dependência, agregação e composição;
- Visibilidade: pública (+), privada (-) e protegida (#).

A Figura 2 apresenta um modelo de diagrama de classes que representa a estrutura estática do sistema bancário, evidenciando suas classes, atributos, operações e relacionamentos.

Figura 2 - Exemplo de Diagrama de Classes



Fonte: Guedes (2011)

2.1.3 Diagrama de Componentes

O diagrama de componentes é um diagrama estrutural da UML cuja função é representar a organização e a dependência entre os componentes de *software* que compõem um sistema. Segundo Booch, Rumbaugh e Jacobson (2005, p. 605), esse diagrama "modela a estrutura física dos componentes e como eles se conectam por meio de interfaces providas e requeridas". Dessa forma, fornece uma visão clara sobre a modularidade do *software*, permitindo identificar responsabilidades, delimitar fronteiras entre subsistemas e apoiar decisões de arquitetura.

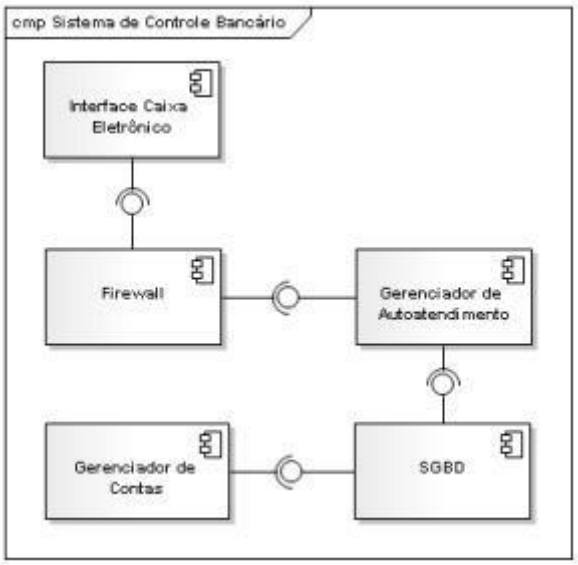
Conforme Larman (2007), a modelagem por componentes contribui para o reuso de módulos e facilita a manutenção, uma vez que estabelece explicitamente as interfaces responsáveis pela comunicação entre partes do sistema. Além disso, Bass, Clements e Kazman (2021) destacam que diagramas de componentes são essenciais na documentação arquitetural, pois evidenciam a decomposição funcional do *software* e seu alinhamento com os requisitos de qualidade, como escalabilidade e independência entre módulos.

Os principais elementos do diagrama de componentes incluem:

- Componentes: unidades modulares de implementação, representadas como blocos com o estereótipo «*component*»;
- Interfaces: pontos de acesso definidos por contratos;
- Dependências: relações que indicam que um componente utiliza a interface de outro;
- Portas e conectores: utilizados para especificar a comunicação entre componentes.

Sendo assim, esse tipo de diagrama permite compreender a arquitetura lógica e modular do sistema, complementando outros diagramas estruturais e garantindo uma visão clara do arranjo interno do *software* antes de sua implementação. A Figura 3 apresenta um diagrama de componentes que demonstra a organização modular do sistema, destacando seus principais componentes e as interfaces que estabelecem a comunicação entre eles.

Figura 3 - Exemplo de Diagrama de Componentes



Fonte: Guedes (2011)

2.1.4 Diagrama de Sequência

O diagrama de sequência pertence à categoria dos diagramas de interação e é utilizado para mostrar como os objetos interagem entre si ao longo do tempo, trocando mensagens em sequência cronológica. Conforme Fowler (2005, p. 94), esse diagrama "mostra como os objetos interagem entre si por meio da troca de mensagens ao longo do tempo".

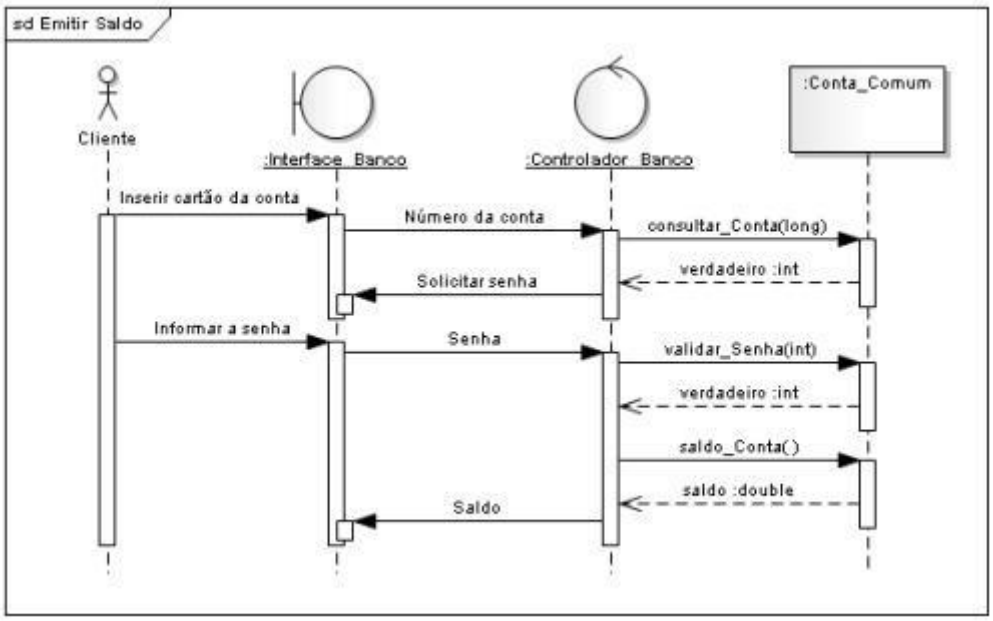
Ele é particularmente útil para detalhar o comportamento dinâmico de um sistema, permitindo a visualização do fluxo de execução associado a um caso de uso específico. Dessa forma, complementa os diagramas estruturais ao evidenciar a lógica de funcionamento do sistema.

Os elementos principais de um diagrama de sequência incluem:

- Objetos: instâncias das classes envolvidas na interação;
- Linhas de vida (*lifelines*): indicam o tempo de existência dos objetos;
- Mensagens: representam chamadas de métodos, respostas e interações entre objetos;
- Barras de ativação: indicam o período de execução de uma operação.

A Figura 4 apresenta um diagrama de sequência que ilustra o fluxo de mensagens trocadas entre os objetos durante a execução do caso de uso selecionado.

Figura 4 - Exemplo de Diagrama de Sequência



Fonte: Guedes (2011)

2.1.5 Diagrama de Implantação

O diagrama de implantação (ou *deployment diagram*) é um diagrama estrutural da UML voltado para a representação da arquitetura física do sistema (OMG, 2017). Seu objetivo é modelar como os componentes de *software* são distribuídos entre os nós de *hardware*, sendo essencial em sistemas distribuídos.

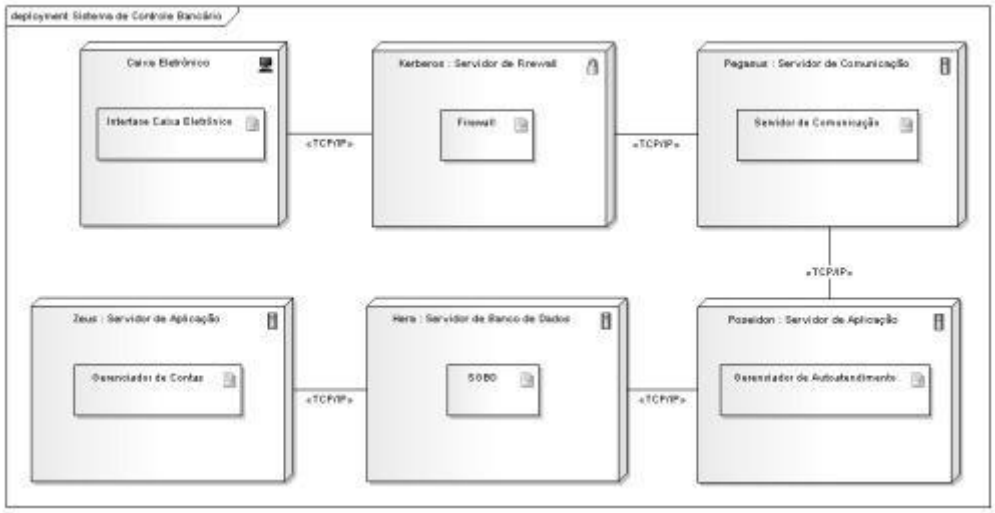
Segundo Booch, Rumbaugh e Jacobson (2005, p. 619), esse diagrama "mostra a configuração do *hardware* (nós) e a distribuição dos componentes de *software* sobre essa infraestrutura". Assim, ele é indispensável no planejamento da infraestrutura de produção, ajudando a tomar decisões relacionadas a desempenho, escalabilidade e segurança.

Os elementos principais de um diagrama de implantação incluem os seguintes:

- Nós: dispositivos físicos ou virtuais (como servidores, computadores ou dispositivos móveis);
- Componentes e artefatos: *softwares* implantados nos nós;
- Ligações de comunicação: conexões físicas ou lógicas entre nós que representam a troca de dados e mensagens.

A seguir, a Figura 5 apresenta um diagrama de implantação que demonstra a distribuição dos componentes de *software* entre os diferentes nós de *hardware* do sistema bancário.

Figura 5 - Exemplo de diagrama de Implantação



Fonte: Guedes (2011)

Essas abordagens garantem uma visão abrangente do funcionamento do sistema antes da sua implementação. Em especial, os diagramas de classes, componentes e de implantação são essenciais para a modelagem das diferentes visões de um *software* proposto por Bass, Clements e Kazman (2021). Por sua vez, os diagramas de sequências são essenciais na modelagem de cenários, úteis para a avaliação de uma arquitetura (Bass; Clements; Kazman, 2021).

2.2 Ensino Mediado por Tecnologia

O uso da tecnologia na educação contemporânea tem sido um fator transformador e significativo no processo de ensino-aprendizagem. Moran (2015) destaca que o uso de plataformas interativas pode tornar o aprendizado mais dinâmico e acessível, promovendo uma maior autonomia dos discentes.

Kenski (2003, p.47) enfatiza que as tecnologias “não são apenas instrumentos auxiliares, mas elementos que transformam profundamente os modos de ensinar e aprender, criando novas formas de interação, comunicação e construção de saberes”.

Neste contexto, ambientes virtuais de aprendizagem desempenham um papel fundamental, proporcionando interatividade e colaboração entre docentes e discentes. Valente (1999) reforça essa perspectiva ao afirmar que a mediação tecnológica favorece a construção ativa do conhecimento, incentivando o engajamento e a participação ativa.

Na Educação Superior, as pesquisas evidenciam que a tecnologia também atua como mediadora essencial da aprendizagem, promovendo práticas mais dinâmicas. Richter, Cleitom,

Bernardi e Cordenonsio (2019), em estudo realizado pela Universidade Federal do Rio Grande do Sul (UFRGS), identificaram que metodologias como oficinas, aprendizagem baseada em problemas (PBL) e aprendizagem baseada em projetos (ABP) têm contribuído para maior motivação dos alunos, especialmente em disciplinas com elevado grau de abstração, como a programação de computadores.

Além disso, Pesquisas com programas de mentoria on-line em instituições brasileiras têm demonstrado contribuições significativas para a formação docente, especialmente no que se refere ao desenvolvimento de competências interpessoais e reflexivas. Estudos que investigam iniciativas de mentoria on-line em contexto universitário indicam que esse tipo de prática favorece a construção de vínculos de confiança, a troca de saberes pedagógicos e a reflexão crítica sobre a própria prática educativa, contribuindo para ampliar a empatia, escuta ativa e o senso de pertencimento entre docentes e participantes do programa (Ferreira et al., 2023).

Portanto, o ensino mediado por tecnologia não se restringe ao uso instrumental de ferramentas digitais, mas se consolida como um modelo de aprendizagem colaborativo, criativo e personalizado, que integra aspectos cognitivos, afetivos e sociais no processo educativo. Essa mediação amplia o papel do professor como orientador e do estudante como protagonista da construção do conhecimento.

2.3 Gestão de Conhecimento (GC)

A Gestão do Conhecimento (GC) é um campo essencial para o desenvolvimento organizacional, pois promove o aprendizado contínuo, a inovação e a eficiência institucional. Nonaka e Takeuchi (1997) afirmam que o conhecimento organizacional é criado pela interação entre o conhecimento tácito, pessoal e subjetivo, e o explícito, que pode ser registrado e compartilhado. Essa relação ocorre no modelo SECI, composto pelas etapas de socialização, externalização, combinação e internalização, representando o ciclo de criação e conversão do conhecimento.

Nas Instituições de Educação Superior (IESs), a GC assume papel estratégico ao preservar e difundir o saber institucional. Braga (2022) ressalta que a construção de uma memória organizacional reduz o retrabalho, facilita a tomada de decisão e assegura a continuidade das atividades, mesmo diante da rotatividade de servidores. Dessa forma, o conhecimento passa a ser um ativo coletivo, fundamental para o fortalecimento das práticas administrativas e acadêmicas.

Para Davenport e Prusak (1998), gerir conhecimento vai além do simples armazenamento de informações: é necessário criar uma cultura organizacional que valorize a colaboração e o compartilhamento de experiências. Nessa perspectiva, as Tecnologias da Informação e Comunicação (TICs) desempenham papel relevante ao permitir a disseminação do conhecimento e a integração entre os diferentes atores da instituição (Choo, 2003).

No contexto da UFERSA, a aplicação dos princípios da GC pode fortalecer projetos como o *Mentoring*, que promove a troca de saberes entre mentores e mentorados. A incorporação de mecanismos de GC nesse sistema permitirá registrar boas práticas e lições aprendidas, criando uma base de conhecimento coletivo. Assim, a GC contribui para a consolidação de um ciclo contínuo de aprendizagem e para o aprimoramento da gestão educacional e institucional.

2.4 Arquitetura de Software

A arquitetura de *software* pode ser entendida como a estrutura fundamental de um sistema, composta pelos seus principais componentes, pelas relações entre eles e pelos princípios que orientam seu projeto e evolução. Fowler (2003) a define como o conjunto das decisões mais importantes do sistema, aquelas que determinam sua estrutura, comportamento e capacidade de adaptação. De forma semelhante, Brooks (1995) ressalta que a arquitetura é o resultado da conciliação entre as exigências técnicas e as necessidades humanas, sendo essencial para reduzir a complexidade do desenvolvimento.

O objetivo da arquitetura de *software* é garantir que o sistema atenda de forma eficiente aos requisitos funcionais e não funcionais, promovendo qualidade, manutenibilidade e escalabilidade. Segundo Valente (2022), a arquitetura deve proporcionar uma visão de alto nível do sistema, permitindo que as decisões técnicas sejam tomadas de modo colaborativo e fundamentado. Essa visão auxilia na comunicação entre os membros da equipe e orienta o desenvolvimento desde as fases iniciais do projeto.

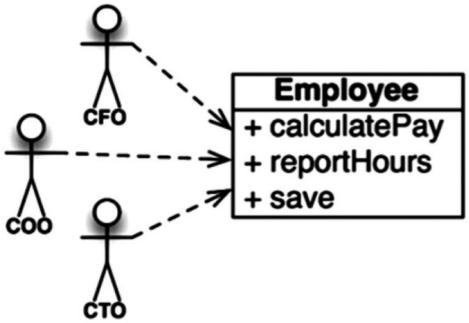
Entre os principais princípios arquiteturais, destacam-se a modularidade, o baixo acoplamento, a alta coesão e a separação de responsabilidades. Esses aspectos facilitam o entendimento e a evolução do sistema, reduzindo custos de manutenção. No projeto arquitetural descrito por Silva e Sousa (2022) para o sistema STAAPCMR, por exemplo, o uso de módulos, componentes e camadas visou garantir a clareza estrutural e a modificabilidade do sistema, especialmente por meio de princípios como a Responsabilidade Única e a Inversão de Dependência, do conjunto SOLID.

Os padrões e princípios de *design*, como os propostos por Martin (2019), desempenham papel fundamental na qualidade da arquitetura. Esse conjunto de princípios do SOLID estabelece diretrizes que promovem a flexibilidade e a robustez do software, alinhando-se às ideias de modularidade e baixo acoplamento discutidas por autores como Valente (2022) e Bass, Clements e Kazman (2021). A seguir, exploramos cada princípio SOLID, com definições concisas, exemplos práticos do livro de Martin e aplicações no contexto arquitetural, incluindo o sistema STAAPCMR.

2.4.1 Princípio da Responsabilidade Única (SRP - *Single Responsibility Principle*)

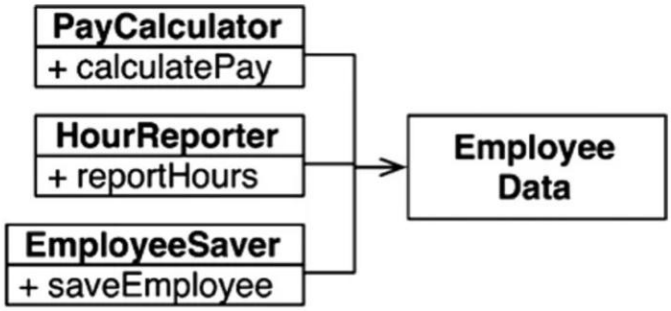
O SRP define que cada módulo ou classe deve ter uma única razão para mudar, ou seja, deve ser responsável por apenas uma tarefa específica. Isso evita que alterações em uma funcionalidade afetem outras partes do sistema. Martin (2019) ilustra esse princípio com o exemplo de uma classe possui método associado a diferentes atores (eg., contabilidade, gestão de pessoas e administração de banco de dados), ilustrado na Figura 6: separá-las em classes distintas (i.e., voltadas a cada um dos atores) torna o código mais coeso e fácil de testar, conforme pode ser visto na Figura 7. Aplicado à arquitetura, o SRP promove camadas independentes, como a separação entre domínio e infraestrutura. No sistema STAAPCMR, conforme Silva e Sousa (2022), esse princípio foi usado para dividir responsabilidades em módulos específicos, facilitando a manutenção e reduzindo o impacto de mudanças.

Figura 6 - Exemplo de um código que fere o SRP



Fonte: Martin (2019)

Figura 7 - Reorganização do código seguindo o SRP

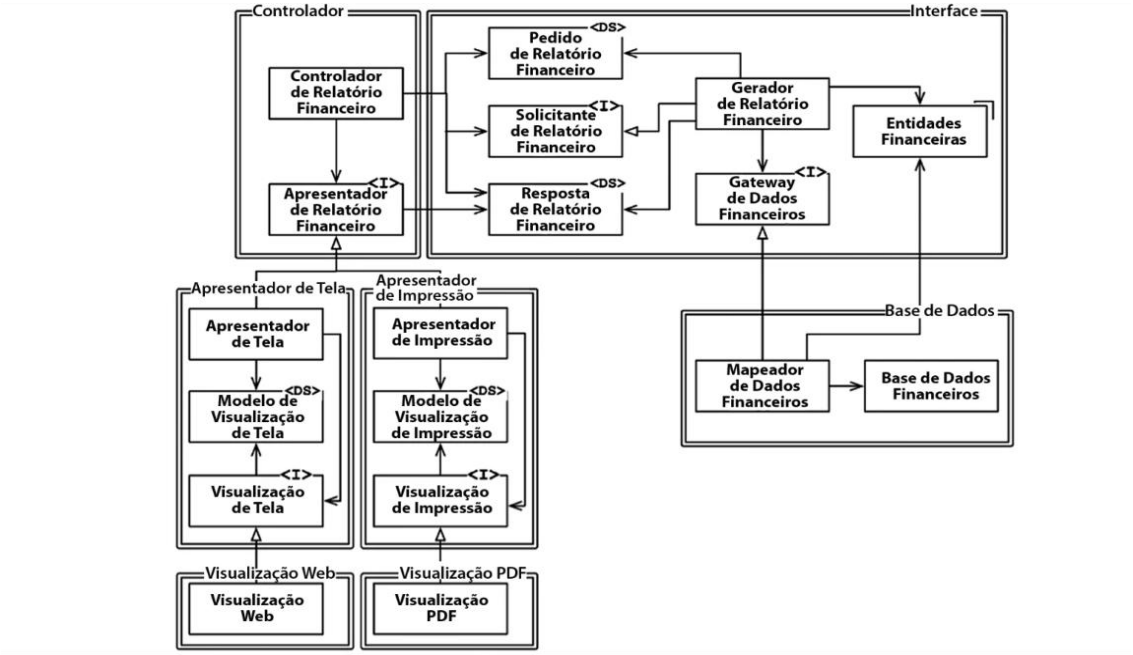


Fonte: Martin (2019)

2.4.2 Princípio Aberto-Fechado (OCP - *Open-Closed Principle*)

O OCP estabelece que entidades de *software* (classes, módulos) devem estar abertas para extensão, mas fechadas para modificação. Isso significa ter um *software* que permite adicionar novas funcionalidades sem alterar o código existente, reduzindo o risco de *bugs*. Martin (2019) ilustra o OCP com um exemplo de um sistema de relatórios que pode ser estendido com novos formatos (como PDF ou Excel), sem modificar a classe base, através de herança ou interfaces. Na Figura 8, pode-se ver que novos tipos de apresentação podem ser incluídos sem reestruturar o restante do código. Na arquitetura, isso se traduz em *designs* modulares, onde *plugins* ou novos componentes podem ser integrados sem reescrever o núcleo do sistema. Esse princípio foi aplicado no STAAPCMR para permitir extensões futuras, como novos tipos de análise, sem comprometer a estrutura existente.

Figura 8 - Organização do software para seguir o OCP

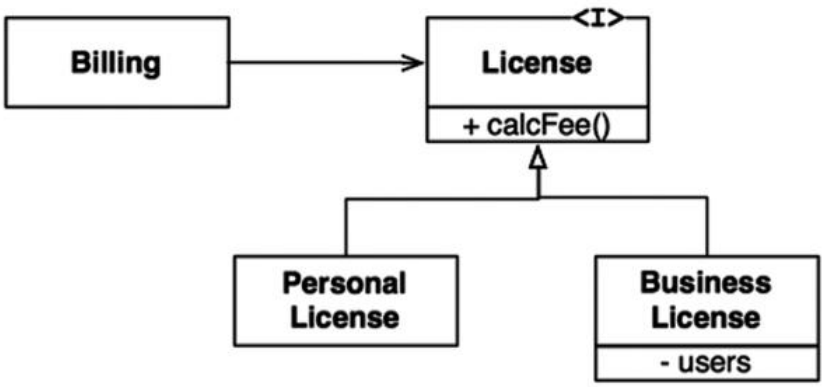


Fonte: Martin (2019)

2.4.3 Princípio da Substituição de Liskov (LSP - *Liskov Substitution Principle*)

O LSP estabelece que subtipos devem ser substituíveis por seus tipos base sem alterar o comportamento esperado do programa. Violá-lo leva a comportamentos inesperados, como uma subclasse que lança exceções não previstas. Martin (2019) ilustra o princípio com uma classe que representa uma cobrança (*Billing*), ilustrado na Figura 9. O valor da cobrança é baseado em dois tipos de licenças: Licença Comercial (*Business Licence*) e Licença Pessoal (*Personal Licence*). Para cumprir o LSP, o software não pode requerer tratamento especial, independente do tipo de licença e de suas bases de cálculo. Aplicado amplamente, o LSP reforça a confiabilidade em sistemas orientados a objetos, evitando dependências frágeis. No contexto do STAAPCMR, ele assegurou que subclasses de componentes pudessem ser trocadas sem afetar a integridade do sistema, promovendo reutilização e testabilidade.

Figura 9 - Exemplo do LSP

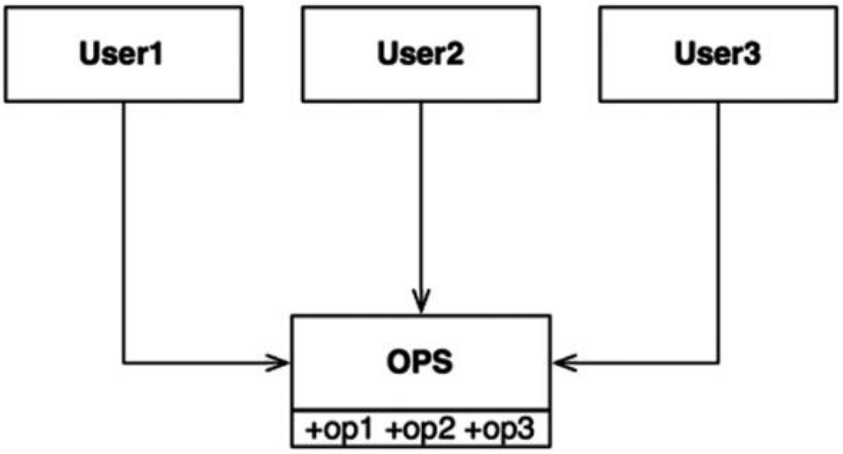


Fonte: Martin (2019)

2.4.4 Princípio da Segregação de Interfaces (ISP - *Interface Segregation Principle*)

O ISP estabelece que clientes não devem ser forçados a depender de interfaces que não usam. Em vez de uma interface "gorda" com muitos métodos, crie interfaces específicas e pequenas. Martin explica em Arquitetura Limpa que isso previne acoplamentos desnecessários: imagine uma interface Impressora com métodos para imprimir, escanear e faxear; clientes que só precisam imprimir não devem implementar os outros. Na Figura 10, é apresentado uma representação visual de um *design* que pode ser reestruturado, em que embora os usuários (*Users*) usem apenas uma operação específica, conhecem as operações de todos os usuários ao acessar a classe OPS.

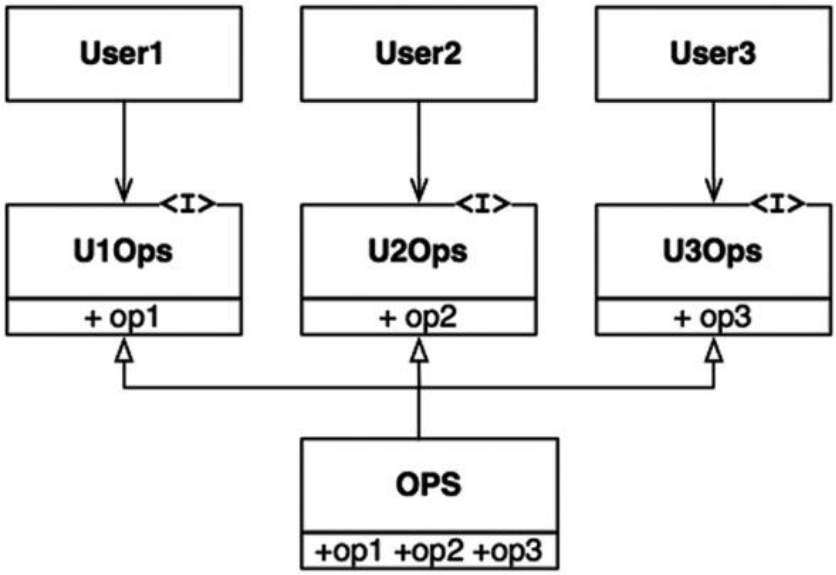
Figura 10 - Exemplo de organização que fere o ISP



Fonte: Martin (2019).

Por sua vez, na Figura 11, é apresentada uma organização do código, que emprega o ISP, em que cada usuário possui acesso apenas aos seus métodos específicos, por meio da segregação do código em três interfaces distintas. Na arquitetura, ISP leva a *designs* mais granulares, facilitando testes e evoluções independentes de componentes. Para o STAAPCMR, interfaces segregadas foram usadas para desacoplar módulos, permitindo que diferentes partes do sistema evoluíssem sem interferências.

Figura 11 - Estruturação do software seguindo o ISP



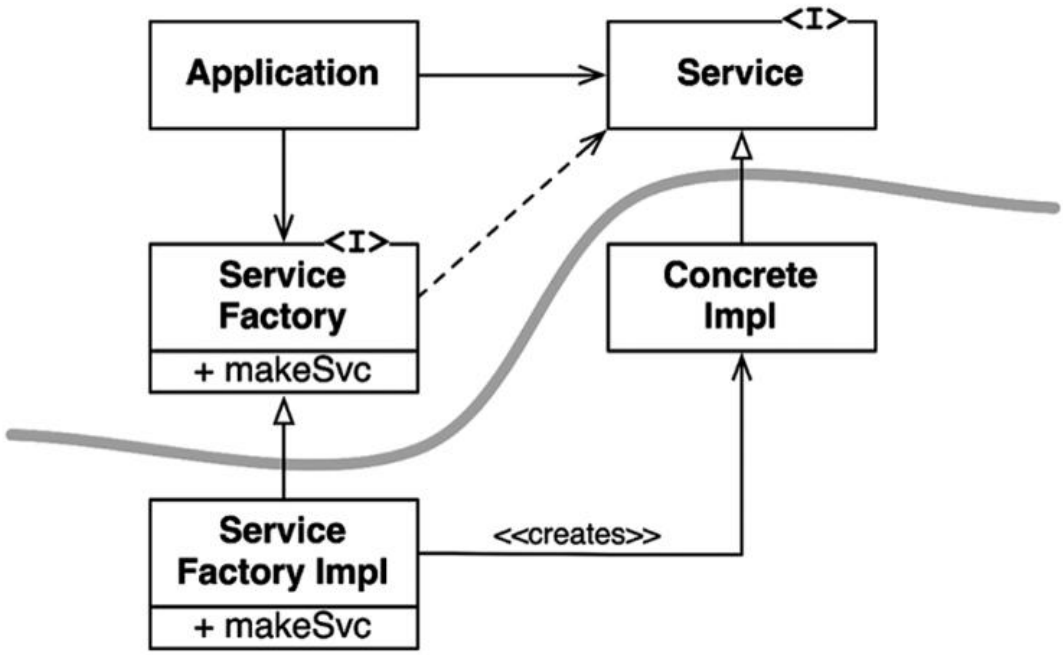
Fonte: Martin (2019)

2.4.5. Princípio da Inversão de Dependência (DIP - *Dependency Inversion Principle*)

O DIP estabelece que módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações. Além disso, abstrações não devem depender de detalhes, mas o contrário. O livro enfatiza que isso inverte o fluxo de dependência, usando injeção de dependência para desacoplar camadas: por exemplo, uma classe de negócio não depende diretamente de um banco de dados, mas de uma interface abstrata. Isso resulta em arquiteturas limpas onde mudanças em tecnologias (como trocar um banco SQL por NoSQL) afetam apenas as camadas externas, preservando o núcleo do sistema. De forma concreta, Martin (2019) apresenta, como exemplo, o Padrão de Projeto Fábrica Abstrata, que ao invés de criar objetivos diretamente, criar por meio de fábrica (Figura 12). No STAAPCMR, o DIP foi crucial para reduzir acoplamentos, permitindo modificações na infraestrutura sem impactar

a lógica de negócio, alinhando-se às recomendações de Bass, Clements e Kazman (2021) sobre modificabilidade.

Figura 12 - Uso do padrão *Abstract Factory* para lidar com a dependência



Fonte: Martin (2019)

Os princípios SOLID, como detalhados em Martin (2019), são ferramentas essenciais para construir *software* sustentável, integrando-se perfeitamente aos conceitos de arquitetura discutidos por Fowler (2003), Brooks (1995) e Valente (2022). Eles promovem uma arquitetura que separa preocupações, facilita a evolução e reduz custos de manutenção a longo prazo, como evidenciado na aplicação prática no sistema STAAPCMR por Silva e Sousa (2022). Ao aplicá-los, desenvolvedores criam sistemas mais resilientes, alinhados com práticas ágeis e escaláveis. Em síntese, a arquitetura de *software* é a base conceitual e técnica sobre a qual se sustenta a engenharia de sistemas complexos. Ela traduz as decisões estruturais que viabilizam a qualidade, a segurança e a manutenção do *software* ao longo do tempo. Como enfatiza Brooks (1995), embora não exista uma "bala de prata" capaz de eliminar toda a complexidade do desenvolvimento de *software*, uma arquitetura bem projetada, guiada por princípios como o SOLID, é a ferramenta mais eficaz para controlá-la e garantir o sucesso do projeto.

3 TRABALHOS RELACIONADOS

Esta seção apresenta e discute os principais trabalhos correlatos ao desenvolvimento de sistemas e programas de mentoria e monitoria na Educação Superior, com ênfase em iniciativas voltadas à integração acadêmica, acompanhamento discente e apoio à permanência estudantil. A análise contempla estudos desenvolvidos no contexto da UFERSA e em outras instituições de ensino, além de sistemas tecnológicos que inspiram e subsidiam a modelagem e a arquitetura proposta neste trabalho.

O projeto *Mentoring* constitui o principal antecedente desta pesquisa e teve como foco inicial o desenvolvimento de um protótipo de plataforma Web para mentoria acadêmica entre pares na UFERSA, com o objetivo de apoiar a adaptação de estudantes ingressantes à vida universitária. O sistema foi concebido com base em princípios de usabilidade, modularidade e escalabilidade, utilizando tecnologias como *HTML5, CSS3, TypeScript* e *Vite*. Contudo, essa etapa restringiu-se à prototipagem da interface, sem contemplar *back-end*, persistência de dados ou uma arquitetura de software formal. A presente pesquisa dá continuidade à iniciativa, avançando na modelagem e definição arquitetural do sistema, visando torná-lo uma solução completa para a gestão de mentorias acadêmicas na UFERSA.

Outro estudo de destaque é o trabalho de Menezes *et al*. (2021), também na UFERSA, no âmbito do Curso de Medicina. O projeto descreve a aplicação da mentoria entre pares (*peer mentoring*) como estratégia de acolhimento e suporte emocional aos ingressantes, especialmente diante das metodologias ativas de ensino, como o *Problem-Based Learning* (PBL). A pesquisa evidenciou benefícios significativos quanto à redução da ansiedade, ao desenvolvimento da autogestão e à melhoria da adaptação acadêmica, reforçando a importância de práticas estruturadas de acompanhamento discente. Esses resultados sustentam a relevância de sistemas digitais que formalizam e ampliam o alcance dessas ações no ambiente universitário.

De forma complementar, o estudo de Sória e Macuch (2025) apresenta uma visão abrangente sobre as experiências de mentoria acadêmica no país. As autoras analisaram 21 artigos publicados entre 2018 e 2023, identificando que a maioria dos programas ocorre em cursos da área da saúde, especialmente Medicina, sendo estruturados de forma formal, híbrida e entre pares. Os resultados destacam o papel da mentoria como prática de acolhimento, integração e fortalecimento de vínculos, mas apontam lacunas em relação à sistematização tecnológica, acompanhamento contínuo e mensuração de resultados, reforçando a pertinência

de soluções digitais baseadas em arquitetura de software escalável, como a proposta neste trabalho.

Além desses estudos, é pertinente mencionar o SADIS (Sistema de Acompanhamento e Mentoria Discente), desenvolvido na Universidade Federal de Santa Catarina (UFSC), conforme apresentado por Silva e Pereira (2021). O SADIS foi concebido para gerenciar programas de tutoria e mentoria acadêmica, permitindo o registro de encontros, o acompanhamento de desempenho e a emissão de relatórios administrativos. Embora tenha obtido sucesso na gestão institucional de mentorias, o sistema não contemplou aspectos de interação social, comunicação entre pares ou integração com metodologias pedagógicas ativas, limitações que o projeto atual busca superar por meio de uma abordagem orientada à experiência do usuário e ao suporte colaborativo.

As evidências encontradas nos trabalhos de Sabino (2025), Menezes *et al*. (2021), Sória e Macuch (2025) e Silva e Pereira (2021) demonstram que as iniciativas de mentoria e monitoria acadêmica apresentam impactos positivos no acolhimento, desempenho e permanência discente, mas ainda carecem de sistemas integrados e arquiteturas robustas que permitam sua gestão automatizada. Nesse contexto, a presente pesquisa propõe uma etapa importante para a construção de um sistema de mentoria acadêmica por meio de uma arquitetura modular e escalável, visando consolidar as práticas já existentes na UFERSA e oferecer suporte tecnológico à coordenação de programas institucionais.

Quadro 1 - Comparativo funcional entre os trabalhos

Autor/Ano	**Sistema/Programa**	**Funcionalidades**	**Limitações**	**Contribuição para o Presente Trabalho**
Menezes *et al*. (2021)	Programa de Peer Mentoring (UFERSA – Medicina)	Encontros presenciais e relatórios manuais; apoio à adaptação ao PBL.	Falta de sistema informatizado; ausência de registros automatizados.	Fundamenta a importância da mentoria como suporte institucional; base para módulos de acompanhamento.

(Continua)

Quadro 1 - Comparativo funcional entre os trabalhos (Conclusão)

Sória e Macuch (2025)	Revisão de Programas de Mentoria	Identificação de práticas de mentoria formal e híbrida; análise de resultados nacionais.	Falta de sistematização digital e padronização de dados.	Direciona a necessidade de um sistema automatizado e interoperável.
Silva e Pereira (2021)	SADIS – Sistema de Acompanhamento e Mentoria Discente (UFSC)	Cadastro de usuários e turmas; registro de encontros; relatórios administrativos.	Interface limitada; ausência de comunicação em tempo real e personalização.	Inspira a implementação de módulos administrativos e relatórios integrados.
Presente Trabalho (2025)	Sistema de Monitoria Acadêmica (UFERSA)	Autenticação e perfis distintos (monitor/monitorado); comunicação interna; agendamento; relatórios; integração institucional.	Em fase de modelagem e arquitetura; implementação futura.	Integra funcionalidades acadêmicas, sociais e administrativas em arquitetura escalável.

Fonte: Autoria própria (2025)

Assim, observa-se que os trabalhos revisados apresentam evidências empíricas e tecnológicas convergentes quanto à relevância de ferramentas de mentoria e monitoria acadêmica. Todavia, o diferencial da presente proposta reside na integração arquitetural, que unifica acompanhamento institucional, comunicação entre pares e gestão pedagógica em um ambiente digital único, adaptável às necessidades da UFERSA e de seus programas de apoio estudantil.

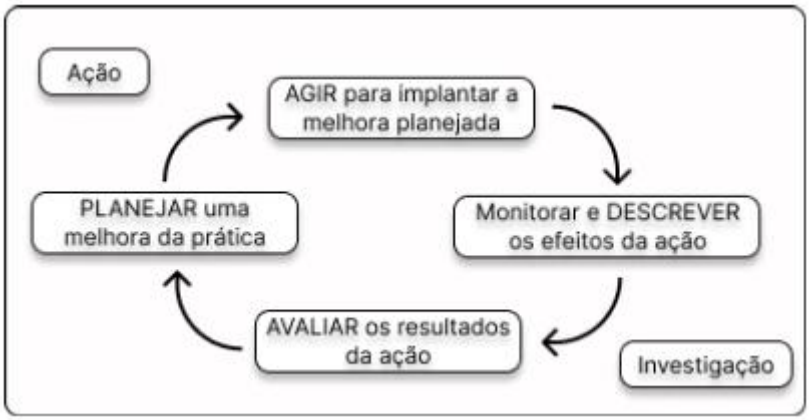
4 METODOLOGIA

A presente pesquisa adota a abordagem metodológica da pesquisa-ação, um método que integra a ação prática com a reflexão sistemática, visando não apenas compreender um fenômeno, mas também promover mudanças e melhorias no contexto estudado. Segundo Thiollent (2011), a pesquisa-ação é caracterizada por ciclos iterativos de planejamento, ação, observação e reflexão, permitindo ao pesquisador intervir diretamente na realidade para resolver problemas práticos enquanto gera conhecimento científico.

Esse método é particularmente adequado ao desenvolvimento de uma arquitetura de software para o projeto Mentoring da UFERSA, pois envolve a participação de *stakeholders* (como administradores, professores, mentores e mentorados) e a implementação incremental de soluções tecnológicas, alinhando-se aos objetivos de engenharia de software e gestão do conhecimento. Neste trabalho, foi escolhida a abordagem por visões arquiteturais apresentada por Bass, Clements e Kazman (2012), que vem sendo aplicadas em outros trabalhos da literatura (Silva et al., 2023; Pereira e Santos, 2022; Silva e Souza, 2022).

Como mencionado anteriormente, a pesquisa-ação é composta por quatro diferentes etapas (Figura 13). A primeira etapa envolve a compreensão do contexto e a definição de um plano para desenvolver uma proposta de intervenção da problemática discutida, tendo denominação de planejamento. Dando continuidade, na segunda etapa, os pesquisadores discorrem acerca do problema para viabilizar uma solução, seguindo o plano elaborado na etapa anterior, sendo denominado de implementação. Em seguida, temos a etapa de monitoramento da solução para entender os efeitos da melhoria prática. Por último, a etapa de avaliação que tem como objetivo analisar os impactos da solução desenvolvida (Tripp, 2005). Essas quatro etapas geralmente são sumarizadas em três fases: planejamento, implementação e avaliação (Tripp, 2005).

Figura 13 - Etapas do método pesquisa-ação



Fonte: Tripp (2005)

Com base nisso, a seguir no Quadro 2 será apresentado uma estruturação das fases que compõem o método, sendo descritivo o processo e o ciclo de vida da pesquisa.

Quadro 2 - Representação do ciclo de vida da pesquisa

Etapa	Prática	Investigação
Planejamento	Levantamento dos Requisitos	(a) das necessidades dos usuários (b) dos elementos a serem incluídos no ambiente.
Implementação	Desenvolvimento da arquitetura	do software desenvolvido
Avaliação	Apresentação dos cenários do ambiente	(a) dos processos

Fonte: Autoria própria (2025)

5 PLANEJAMENTO

A fase de planejamento constitui uma etapa fundamental na pesquisa-ação, pois estabelece as diretrizes que orientarão todo o processo investigativo e interventivo. De acordo com Tripp (2005), o planejamento nessa abordagem metodológica envolve a definição clara do problema, a identificação dos sujeitos envolvidos e a estruturação das ações que visam transformar a realidade estudada.

Partindo deste pressuposto, na etapa de planejamento do presente trabalho foram realizadas ações de elicitação dos requisitos do sistema, visando propor soluções para atender as necessidades dos usuários e levantar artefatos do ambiente proposto.

5.1 Requisitos do Sistema

A definição dos requisitos constitui uma das fases mais relevantes do processo de desenvolvimento de software, permitindo compreender as necessidades dos usuários e estabelecer as funcionalidades previstas para o sistema. De acordo com Sommerville (2019), “requisitos de software definem o que o sistema deve fazer e as restrições sob as quais ele deve operar”, sendo o alicerce para as etapas posteriores de modelagem, arquitetura e implementação.

Além disso, um processo sistemático de requisitos contribui para a qualidade e evolução do software. Bass, Clements e Kazman (2021) afirmam que “a arquitetura é fortemente influenciada pelos requisitos, especialmente os não funcionais”, evidenciando que uma boa elicitação possibilita maior eficiência e menor retrabalho.

Os requisitos apresentados nesta seção foram definidos com base no estudo do processo do Projeto Mentoring da UFERSA e análises de casos similares. Adotaram-se recomendações metodológicas como entrevistas com potenciais usuários, observação de atividades do programa e análise documental, conforme orientações de Valente (2022), que destaca que “o processo de elicitação deve envolver compreensão contextual, técnicas variadas e validação contínua dos requisitos”.

5.2.1 Atores do Sistema

Os atores do sistema podem ser vistos no Quadro 3.

Quadro 3 - Usuários do Sistema

Ator	**Descrição**
Administrador	Gerencia perfis, usuários, turmas e funcionalidades do sistema.
Estudante (Mentor/Mentorado)	Acessa mentorias, agenda encontros, participa das sessões e interage.
Professor	Atua como supervisor, acompanha métricas e monitora grupos.

Fonte: Autoria própria (2025)

5.2.2 Requisitos Funcionais

Os requisitos funcionais descrevem os serviços e comportamentos que o sistema deve oferecer para atender às necessidades dos usuários. De acordo com Wiegers e Beatty (2013), esses requisitos representam “capacidades que o sistema deve possuir para satisfazer as necessidades dos usuários e apoiar os objetivos do negócio”. De maneira complementar, Robertson e Robertson (2012) destacam que os requisitos funcionais definem, de forma objetiva, as ações que o software deve executar para cumprir sua finalidade. Dessa forma, os requisitos apresentados no Quadro 4 especificam as principais funcionalidades necessárias para permitir o gerenciamento das atividades de mentoria,em que P denota requisitos prioritários, E requisitos essenciais, I requisitos importantes e D requisitos desejáveis.

Quadro 4 - Requisitos funcionais

Requisito	**Descrição**	**P.**
RF001 – Cadastrar Administrador	O sistema deve permitir que o usuário administrador realize um cadastro no sistema. Ele deve informar os dados: nome/*user*, *e-mail* e senha.	E
RF002 – Editar Administrador	O sistema deve permitir que o administrador possa editar as informações contidas nos campos do cadastro do usuário admin.	E
RF003 – Excluir Administrador	O sistema deve permitir que o usuário exclua o cadastro de um administrador.	E

(Continua)

Quadro 4 - Requisitos funcionais (Continuação)

RF004 – Cadastrar usuário	O sistema deve permitir o cadastro de usuários na plataforma. Contendo as seguintes informações: nome, *e-mail*, matrícula, campus, curso e senha.	E
RF005 – Editar usuário	O sistema deve permitir que o administrador possa editar as informações contidas nos campos do cadastro do usuário.	E
RF006 – Excluir usuário	O sistema deve permitir que o usuário estudante ou professor possa realizar a exclusão da sua conta.	E
RF007 – Login usuário	O sistema deve permitir que o usuário realize login no sistema. Para isso, ele deve informar e-mail e senha.	E
RF008 – Gerenciar Perfil	O sistema deve permitir que o usuário possa gerenciar diferentes perfis com permissões distintas: estudante (mentor ou mentorado) e professor.	E
RF009 – Formar Grupos de Mentoria	O sistema deve permitir que um usuário possa criar um grupo de mentoring. Para isso, ele deve informar: título, descrição, curso e área de interesse.	E
RF010 – Planejar Encontros	O sistema deve permitir que o usuário possa agendar um encontro semanal e registrar presença. Para isso, ele deve informar: Título, temas de discussão, descrição, data e horário.	E
RF011 – Registrar Atividades	O sistema deve permitir o registro das atividades realizadas em cada sessão/encontro, com relatórios descritivos e anotações. Para isso, ele deve informar: título, descrição, data e arquivo/link.	I
RF012 – Biblioteca de Conteúdos	O sistema deve permitir que o usuário possa disponibilizar materiais de apoio temáticos. Para isso, ele deve informar: curso, disciplina, tema e arquivo/link.	E
RF013 – Comunicação Interna	Ferramentas de comunicação interna, como mensagens e notificações.	D

(Continua)

Quadro 4 - Requisitos funcionais (Conclusão)

RF014 – Avaliação do Processo de Mentoring	O sistema deve permitir que o usuário realize feedbacks sobre a plataforma. Para isso, ele deverá informar: título, curso, campus (opcional), descrição e nota (entre 1 e 10),	I
RF015 – Gerar Relatórios	O sistema deve permitir que o usuário gere relatórios sobre participação, desenvolvimento e indicadores institucionais.	I

Fonte: Autoria própria (2025)

5.2.3 Requisitos Não-Funcionais

Além dos requisitos funcionais, torna-se imprescindível a definição de atributos de qualidade que garantam o desempenho, a segurança e a capacidade de evolução do sistema. Conforme exposto por Bass, Clements e Kazman (2021), os requisitos não funcionais constituem propriedades essenciais para o correto funcionamento do software e exercem influência direta sobre decisões arquiteturais. De modo complementar, Wiegers e Beatty (2013) enfatizam que tais requisitos devem ser claramente mensuráveis e verificáveis, assegurando controle e avaliação contínua ao longo do processo de desenvolvimento.

No âmbito deste trabalho, os requisitos de qualidade buscam assegurar confiabilidade, usabilidade, disponibilidade e segurança ao sistema de mentoria acadêmica, favorecendo uma experiência consistente e adequada para estudantes, docentes e administradores. De acordo com Bass, Clements e Kazman (2021), requisitos de qualidade podem ser especificados por meio de cenários estruturados, compostos por estímulo, fonte, ambiente, resposta e métrica de avaliação. Nos Quadros 5, 6 e 7 são apresentados os cenários elaborados para o sistema proposto.

Quadro 5 - Especificação do atributo de qualidade Segurança

Segurança	
Estímulo	Um usuário tenta acessar o sistema.
Fonte	Usuário autenticado ou visitante.
Ambiente	Operação normal do sistema.
Resposta	O sistema verifica credenciais e só permite acesso a perfis válidos. Caso sejam inválidas, exibem mensagem e impede o acesso.

Segurança	
Medida	O processo de autenticação deve ocorrer em até 5 segundos.

Fonte: Autoria própria (2025)

Quadro 6 - Especificação do atributo de qualidade Escalabilidade

Escalabilidade	
Estímulo	Solicitação de nova funcionalidade para o sistema.
Fonte	Equipe de desenvolvimento ou usuário administrador.
Ambiente	Período de manutenção regular do sistema.
Resposta	A nova funcionalidade é desenvolvida, integrada e testada sem necessidade de alteração substancial em módulos existentes.
Medida	A implementação deve ocorrer em até 90 dias.

Fonte: Autoria própria (2025)

Quadro 7 - Especificação do atributo de qualidade Compatibilidade

Compatibilidade	
Estímulo	Um usuário acessa o sistema utilizando um dispositivo ou navegador diferente (ex.: celular Android, iOS, Chrome, Firefox, Edge).
Fonte	Estudante, professor ou administrador utilizando diferentes plataformas.
Ambiente	Ambiente normal de operação do sistema, com acesso via Web em desktop ou dispositivos móveis.
Resposta	O sistema renderiza corretamente a interface, executa as funcionalidades sem falhas e mantém a consistência visual e funcional.
Medida	As principais funcionalidades devem operar sem erros e a interface deve permanecer utilizável em pelo menos 90% dos navegadores e dispositivos suportados, sem quebra de layout ou perda de função.

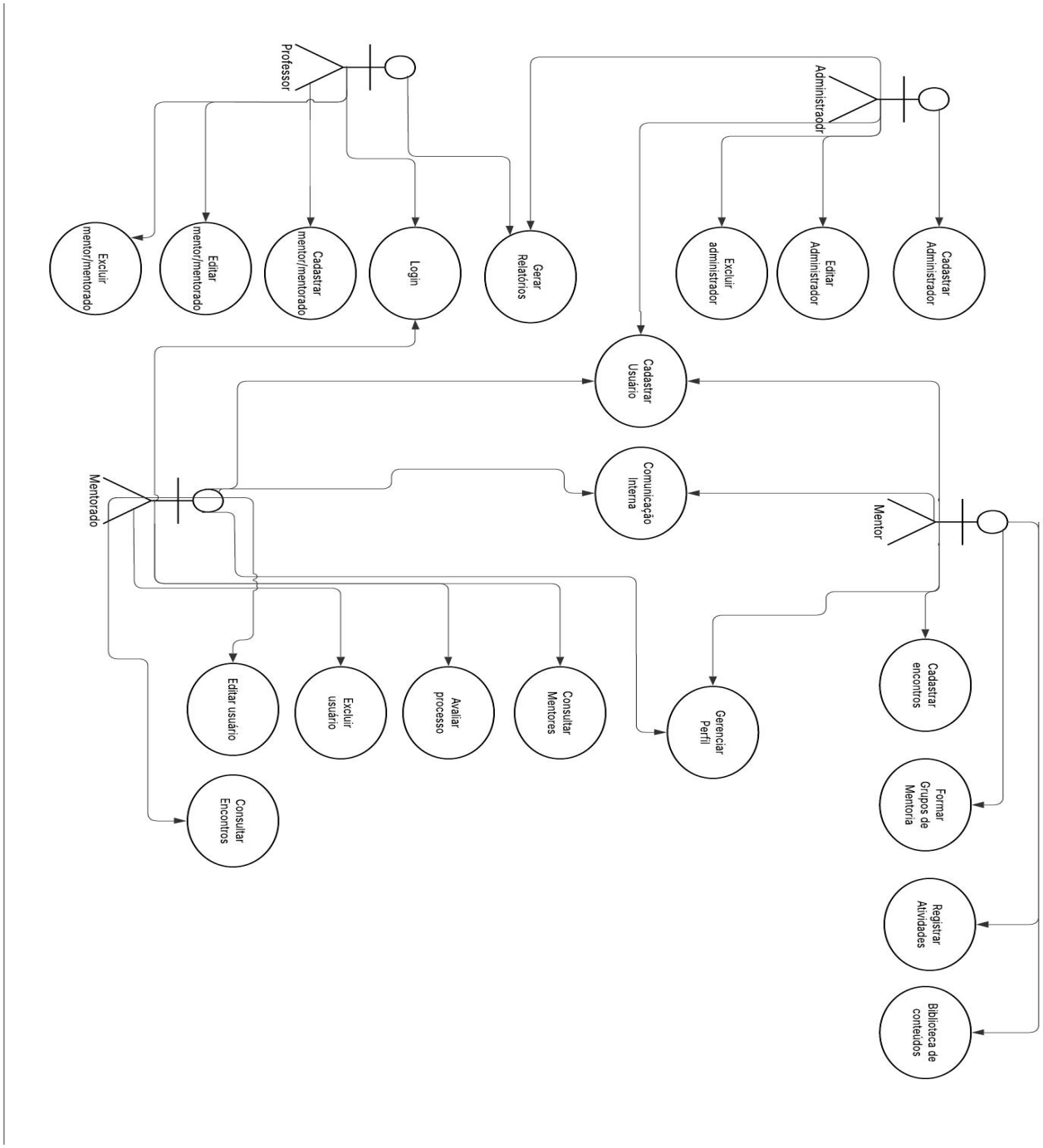
Fonte: Autoria própria (2025)

5.2.4 Casos de Uso

Os diagramas de casos de uso são importantes na modelagem de sistemas de software, pois estruturam as interações entre atores e funcionalidades do sistema. Segundo Guedes (2011), eles oferecem uma representação clara das capacidades principais sob a perspectiva do usuário, ajudando a identificar limites funcionais e serviços necessários. Essa modelagem facilita a visualização do comportamento esperado e a delimitação de responsabilidades dos participantes com o software, organizando os elementos arquiteturais.

Essa compreensão inicial orienta a identificação de componentes principais, fronteiras funcionais e camadas, conforme Bass, Clements e Kazman (2021). Além disso, os casos de uso subsidiam a criação de cenários para avaliação arquitetural, integrando requisitos funcionais e não funcionais. Assim, o planejamento estabelece uma visão coerente das interações, fundamentando o refinamento das visões estruturais e dinâmicas do sistema. Na Figura 14 pode ser visto os casos de uso para o sistema.

Figura 14 - Diagrama de Casos de Uso do Sistema



Fonte: Autoria própria (2025).

5.2 Restrições

No contexto da arquitetura de *software* e da modelagem de sistemas, as restrições são decisões pré-definidas que limitam e orientam o *design*, assegurando que a arquitetura suporta escolhas específicas, como linguagens de programação ou *frameworks*. Segundo Bass, Clements e Kazman (2021), elas definem decisões tomadas que a arquitetura deve suportar.

No ambiente proposto, foi especificado como restrição que o sistema deve ser voltado para o ambiente *Web*, implicando em tecnologias compatíveis com navegadores e protocolos HTTP, facilitando consistência e escalabilidade.

6 IMPLEMENTAÇÃO

A fase de implementação corresponde ao momento em que as soluções definidas durante o planejamento e a modelagem são efetivamente estruturadas, representadas e organizadas em artefatos concretos, conforme previsto na abordagem de pesquisa-ação. De acordo com Thiollent (2011), na etapa de ação ocorre a aplicação prática das intervenções planejadas, permitindo observar o funcionamento real da solução. Assim, nesta seção são apresentados os principais diagramas que compõem a implementação arquitetural do sistema de mentorias.

Conforme Bass, Clements e Kazman (2021), a implementação deve refletir diretamente as decisões arquiteturais descritas ao longo do projeto, garantindo rastreabilidade entre requisitos, modelagem e execução. Dessa forma, os diagramas apresentados nesta seção evidenciam a estrutura lógica, a organização funcional e a forma de implantação do sistema proposto.

6.1 Visões Arquiteturais

A arquitetura do Sistema Mentoring foi elaborada tomando como base o modelo de documentação arquitetural proposto por Bass, Clements e Kazman (2021), que define a arquitetura como a estrutura ou conjunto de estruturas que compreendem os elementos de software, suas propriedades externamente visíveis e os relacionamentos entre eles. Assim, optou-se por estruturar a arquitetura em três visões complementares, visão de módulo, visão Componente & Conector e Visão de alocação, seguindo o padrão descrito pelo método.

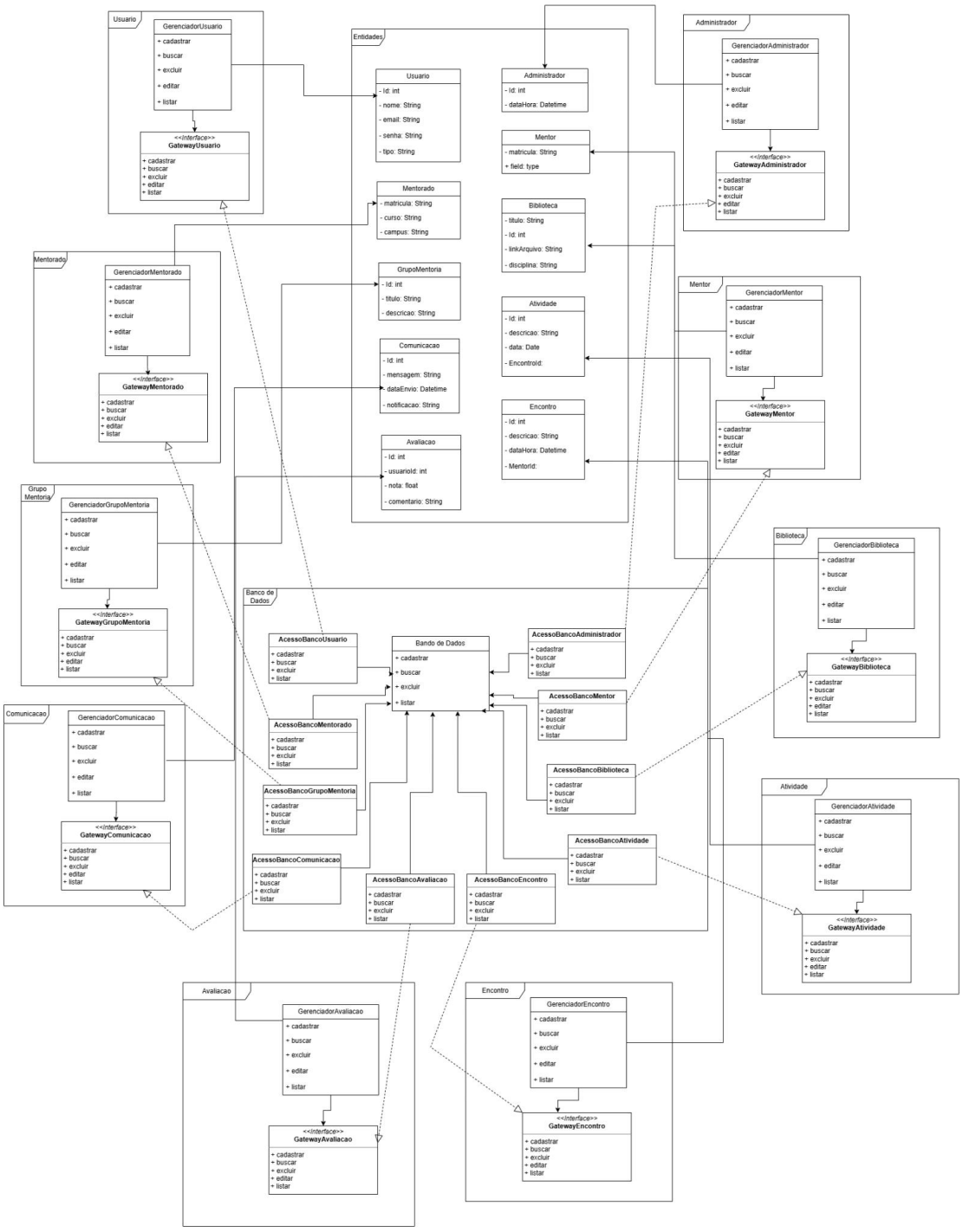
Como recomendado por Bass, Clements e Kazman (2021), cada visão representa o sistema sob uma perspectiva que torna explícitos elementos que não são aparentes nas demais. Essa separação favorece o entendimento, a comunicação entre stakeholders e a capacidade de evolução da solução. A seguir, cada visão é devidamente detalhada.

6.2 Visão de Módulo

A Visão de Módulo descreve a organização estática do sistema e permite compreender sua estrutura em termos de unidades de código e dados, conforme definido por Bass, Clements e Kazman (2021), evidenciando responsabilidades, dependências e hierarquias entre classes e camadas. Essa perspectiva torna possível identificar como o código está distribuído e como seus elementos se agrupam de forma coerente, oferecendo uma visão clara da arquitetura independentemente da execução. No Sistema *Mentoring*, essa visão foi construída

seguindo o modelo de Bass, Clements e Kazman (2021) e materializa-se no diagrama de classes apresentado na Figura 15, que fornece uma visão geral da organização do sistema. A partir desse diagrama, a arquitetura foi detalhada em três subconjuntos complementares, representados nas Figuras 16, 17 e 18, que correspondem, respectivamente, às entidades do domínio, interfaces de acesso (gateways) e aos componentes de persistência em banco de dados.

Figura 15 - Diagrama de Classes Completo



Fonte: Autoria própria (2025)

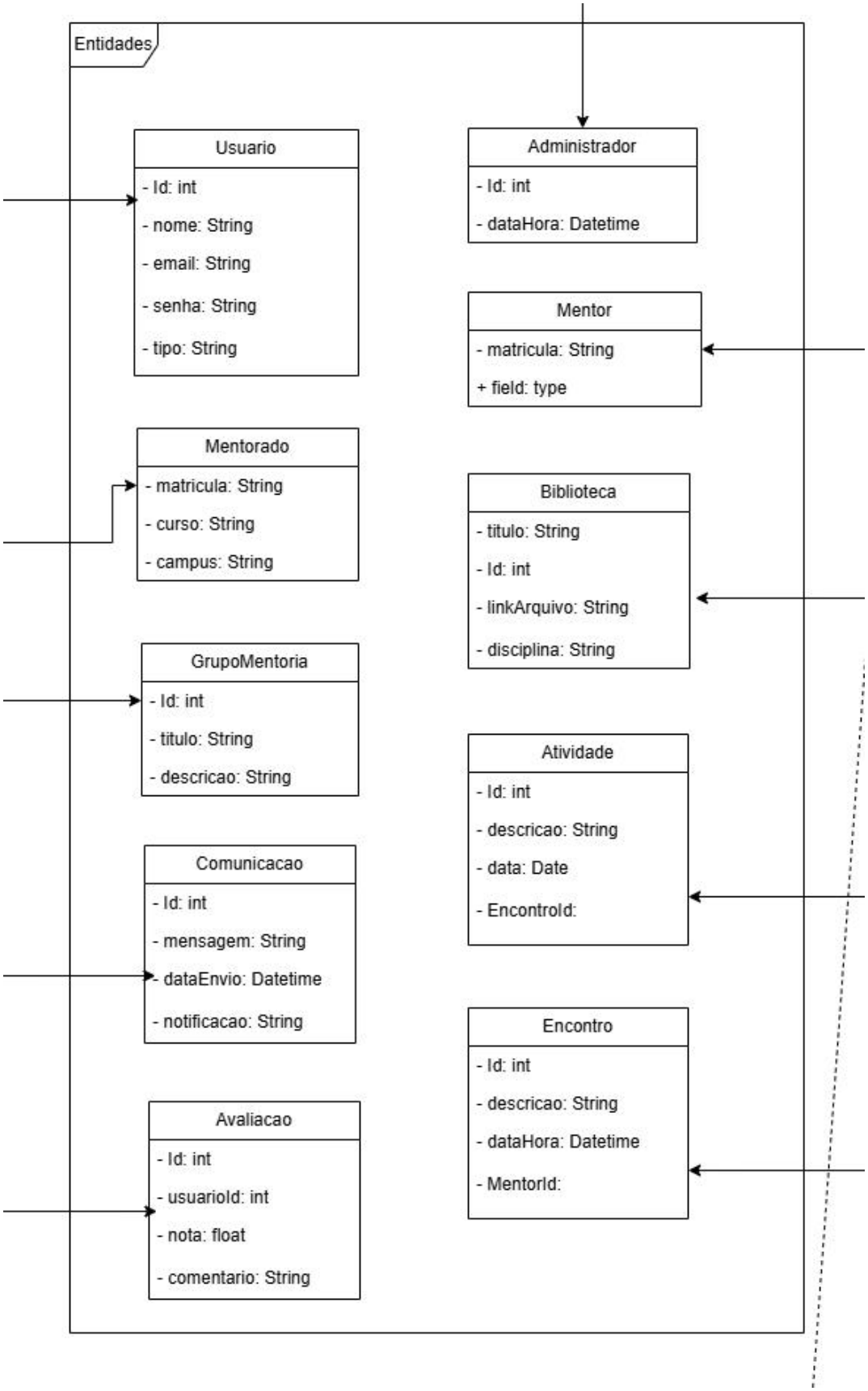
6.2.1 Entidades do Domínio

As entidades do domínio, apresentadas na Figura 16, representam os principais elementos conceituais do Sistema Mentoring, refletindo diretamente os objetos e processos do negócio. Nesse conjunto, a classe Usuário atua como abstração central para todos os perfis

que interagem com o sistema, sendo especializada pelas classes Administrador, Mentor e Mentorado. Essa hierarquia favorece o reaproveitamento de atributos comuns, como identificador, nome, e-mail e credenciais de acesso, reduzindo redundâncias e promovendo maior coesão estrutural.

Além disso, entidades como GrupoMentoria, Encontro, Atividade, Avaliação, Comunicação e Biblioteca representam funcionalidades essenciais do programa de mentoria. As associações definidas entre essas classes expressam os relacionamentos reais do domínio, como a vinculação de encontros a grupos de mentoria, o registro de atividades associadas a encontros específicos e a troca de mensagens entre usuários. Dessa forma, o diagrama de entidades fornece uma visão clara e independente de aspectos técnicos, concentrando-se exclusivamente na estrutura conceitual do sistema.

Figura 16 - Diagrama de Classes: Entidades



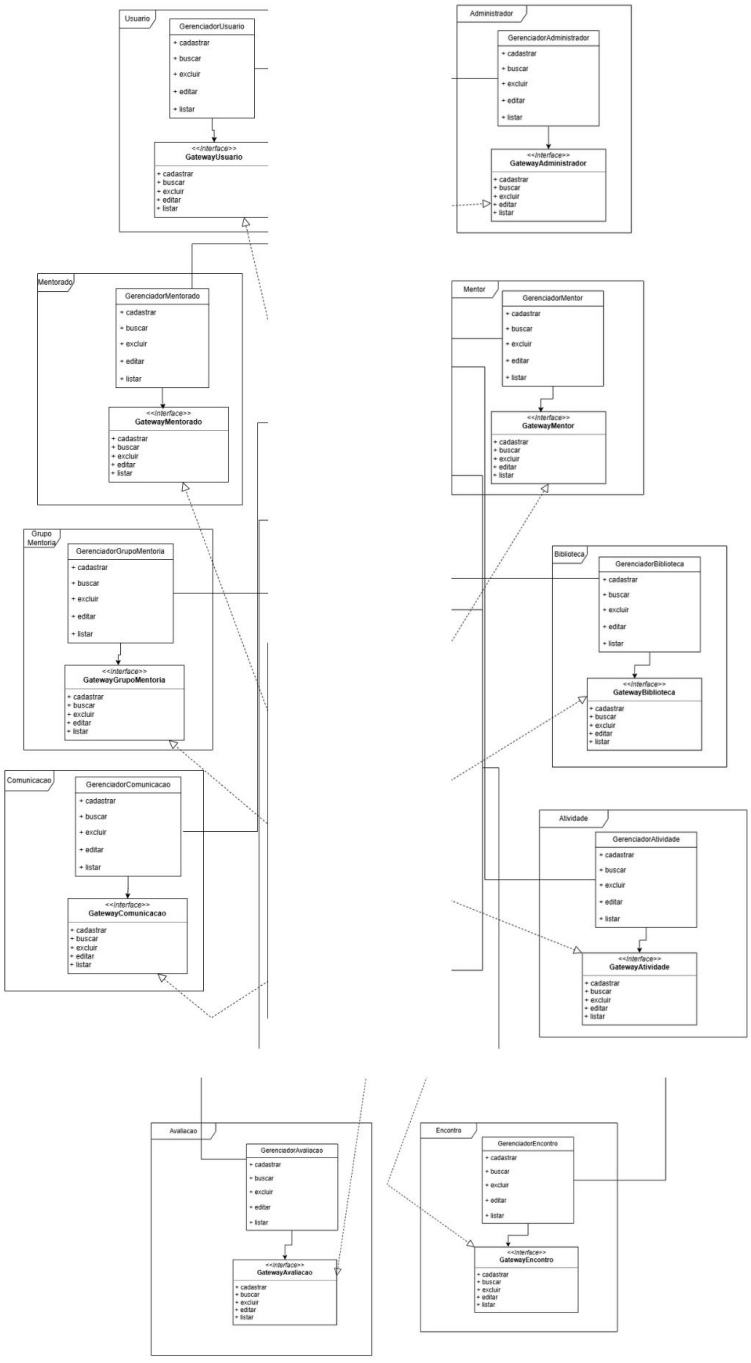
Fonte: Autoria própria (2025)

6.2.2 Interfaces (*Gateways*)

A Figura 17 apresenta o diagrama de classes das interfaces, também denominadas gateways, responsáveis por definir contratos de comunicação entre a lógica de negócio e a camada de persistência. Cada entidade do domínio possui uma interface correspondente, como GatewayUsuario, GatewayMentor, GatewayGrupoMentoria, GatewayEncontro, entre outras, que declaram operações básicas de manipulação de dados, como cadastro, consulta, edição, exclusão e listagem.

A utilização de interfaces nessa camada está alinhada ao Princípio da Inversão de Dependência, uma vez que os módulos de alto nível, representados pelos gerenciadores, passam a depender de abstrações e não de implementações concretas. Essa estratégia reduz o acoplamento entre a lógica de negócio e os mecanismos de armazenamento, favorecendo a manutenibilidade e a extensibilidade do sistema, conforme discutido por Martin (2019).

Figura 17 - Diagrama de Classes: Interfaces



Fonte: Autoria própria (2025)

6.2.3 Persistência em Banco de Dados

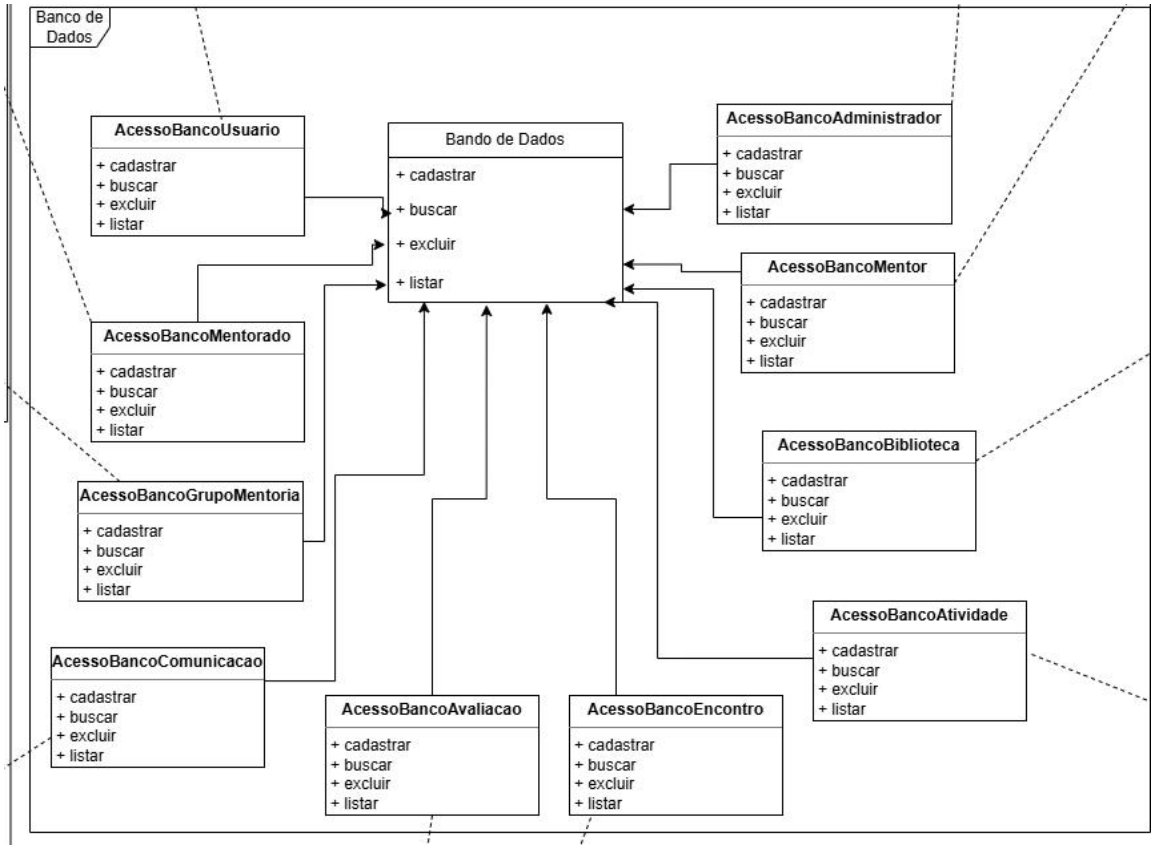
O diagrama de classes da camada de persistência, apresentado na Figura 18, representa a implementação concreta das operações de acesso ao banco de dados. Essa camada é composta pelas classes de acesso ao banco, como AcessoBancoUsuario, AcessoBancoMentor,

AcessoBancoGrupoMentoria, entre outras, que implementam as interfaces definidas na camada de gateways.

Essas classes encapsulam os detalhes técnicos relacionados à comunicação com o banco de dados, como consultas, inserções, atualizações e exclusões de registros. Ao isolar essas responsabilidades em uma camada específica, o sistema evita que as regras de negócio dependam diretamente de decisões tecnológicas, facilitando a manutenção e permitindo futuras mudanças na infraestrutura de dados sem impacto significativo nas demais camadas.

Assim, a separação da Visão de Módulo em entidades do domínio, interfaces e persistência em banco de dados estabelece uma estrutura arquitetural modular, flexível e alinhada às boas práticas de engenharia de software. Essa organização garante rastreabilidade entre requisitos, modelagem e implementação, além de oferecer uma base sólida para a evolução do Sistema Mentoring, conforme a abordagem defendida por Bass, Clements e Kazman (2021).

Figura 18 - Diagrama de Classes: Banco de Dados

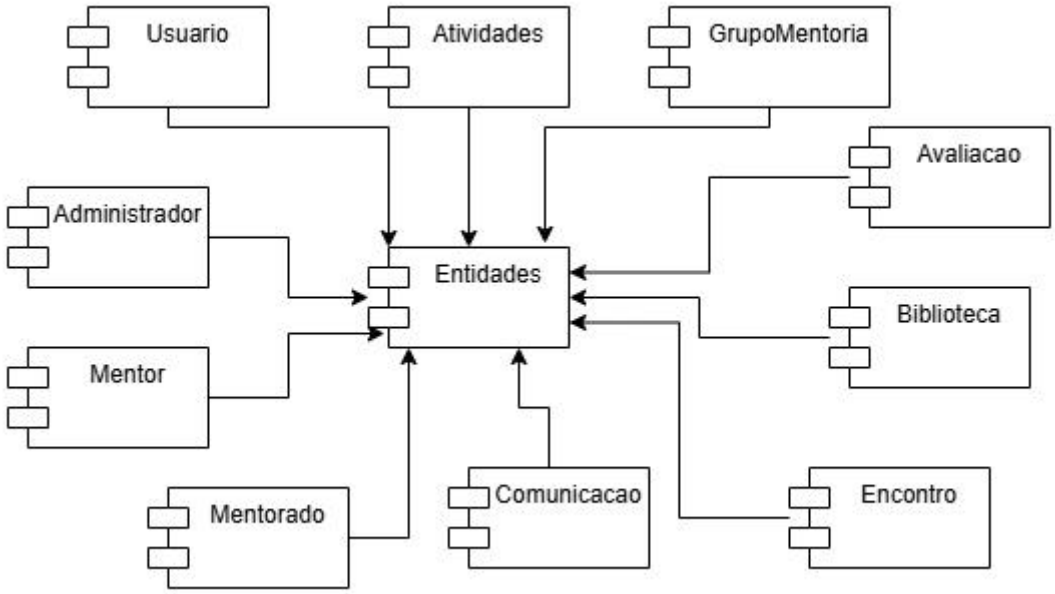


Fonte: Autoria própria (2025)

6.3 Visão Componente & Conector

A Visão Componente & Conector descreve a organização lógica do sistema em tempo de execução, tornando explícitos os componentes de software e os conectores que estabelecem comunicação entre eles. Essa visão focaliza o comportamento dinâmico e os fluxos de interação, no qual os componentes representam unidades de implementação que colaboram para realizar as funcionalidades previstas.

Figura 19 - Diagrama de Componentes do Sistema



Fonte: Autoria própria (2025)

O sistema foi organizado em camadas coerentes com o paradigma arquitetural de separação de responsabilidades. No nível superior, encontram-se os componentes associados aos perfis dos usuários (Administrador, Mentor e Mentorado) que encapsulam as funcionalidades específicas de cada papel. Esses componentes se comunicam diretamente com os componentes centrais de aplicação, em que estão localizados os gerenciadores e demais regras de negócio.

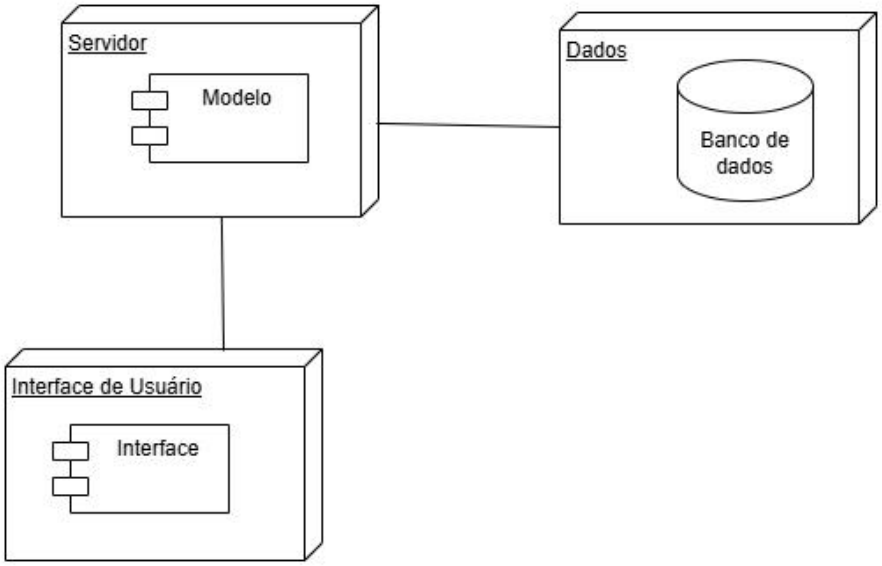
No núcleo da arquitetura, o Componente de Aplicação coordena as operações essenciais do sistema e atua como mediador entre a interface e a persistência. O Componente de Persistência agrupa os elementos que tratam do armazenamento dos dados e garante a consistência das informações. A comunicação entre esses componentes é realizada por conectores bem definidos, que promovem baixo acoplamento e facilitam a substituição ou extensão dos serviços internos.

Essa disposição segue o princípio da dependência acíclica (Martin, 2019), assegurando que os componentes centrais não dependam de componentes periféricos e evitando ciclos estruturais, sendo fundamental para garantir estabilidade e evolução gradual do sistema.

6.4 Visão de alocação

A Visão de Alocação descreve como os elementos de software são mapeados para os elementos físicos de infraestrutura, revelando a distribuição da aplicação no ambiente operacional. Essa visão é essencial para compreender questões de desempenho, comunicação e dependências externas, como salientado por Bass, Clements e Kazman (2021). A Figura 20 representa o modelo de implantação desenvolvido.

Figura 20 - Diagrama de Implantação do Sistema



Fonte: Autoria própria (2025)

A arquitetura adota uma organização típica de sistemas *Web,* distribuída em três nós principais. O primeiro é o cliente Web, acessado por meio de navegador e responsável pela interface com o usuário. O segundo é o servidor de aplicação, no qual se encontram a lógica do sistema, os modelos de domínio, os gerenciadores e os conectores utilizados para comunicação. O terceiro nó é o servidor de banco de dados, encarregado do armazenamento persistente das informações.

Essa configuração atende à restrição de operar integralmente em ambiente *Web* evidenciando a importância da separação entre interface, lógica de aplicação e dados para promover escalabilidade e facilidade de manutenção.

7 AVALIAÇÃO

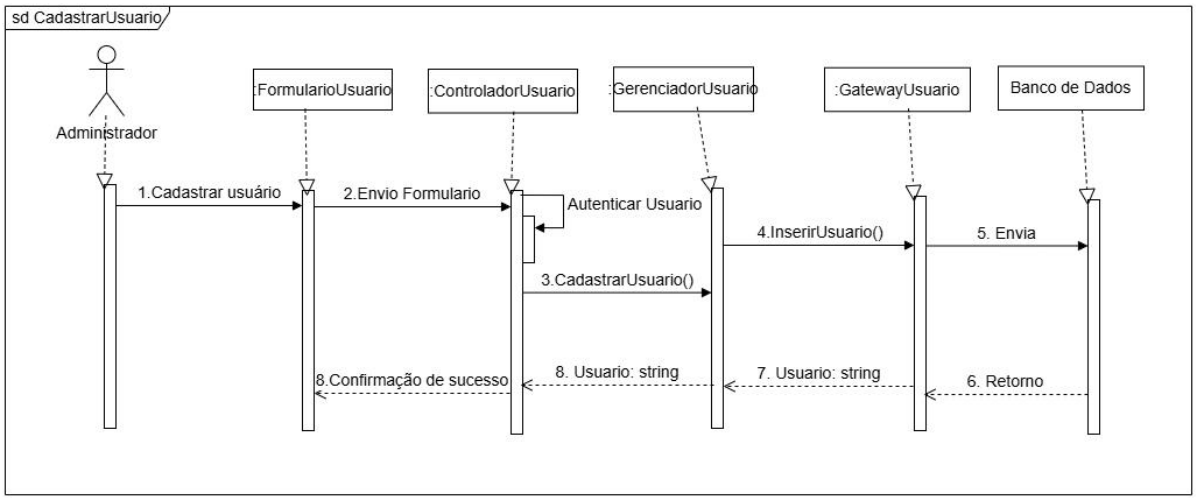
A avaliação da arquitetura foi conduzida com base na análise de cenários representativos do uso real do sistema, seguindo a abordagem de Bass, Clements e Kazman (2021), que propõem avaliar estruturas arquiteturais por meio de estímulos reais e inspeção de seus comportamentos dinâmicos. Essa abordagem, também utilizada no artigo que serviu como referência metodológica, permite verificar se a arquitetura projetada satisfaz os requisitos funcionais e não funcionais essenciais para o funcionamento do sistema de monitoria.

7.1 Cenários

Os cenários analisados envolveram aspectos de segurança, escalabilidade e compatibilidade. A avaliação demonstrou que o sistema apresenta controles adequados de validação de dados e autenticação, arquitetura modular capaz de evoluir com estabilidade e independência entre camadas que promove operação consistente em diversos dispositivos.

Entretanto, o entendimento completo da adequação arquitetural só se torna claro quando esses cenários são analisados em conjunto com os diagramas de sequência, que evidenciam o comportamento real do sistema durante a execução de suas funcionalidades mais importantes. A seguir, apresenta-se a análise detalhada das figuras, uma vez que elas constituem o principal elemento de validação dinâmica da arquitetura.

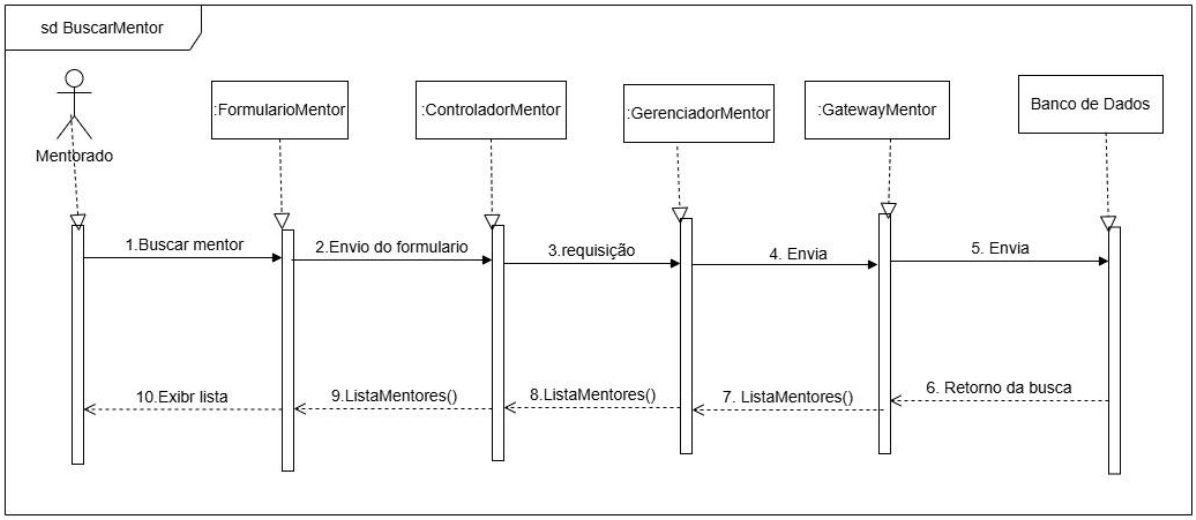
Figura 21 - Diagrama de Sequência Cadastrar Usuário



Fonte: Autoria própria (2025)

O primeiro cenário (Figura 21) ilustra o fluxo de Cadastro de Usuário que demonstra um processo rigoroso para garantir a segurança. O processo é iniciado pelo administrador na interface FormulárioUsuário, mas os dados são imediatamente repassados ao ControladorUsuario, que atua como mediador e realiza a validação e autenticação inicial. A arquitetura exige que toda regra de negócio passe pelo controlador, garantindo organização e segurança. A lógica central de criação do usuário reside no GerenciadorUsuario, que mantém a separação clara entre lógica de negócio e controle. Para a persistência, o GatewayUsuario funciona como um adaptador, isolando o sistema da tecnologia de banco de dados. O fluxo de confirmação percorre todas as camadas de volta, atestando a estabilidade, modularidade e segurança do processo e validando o cenário de segurança.

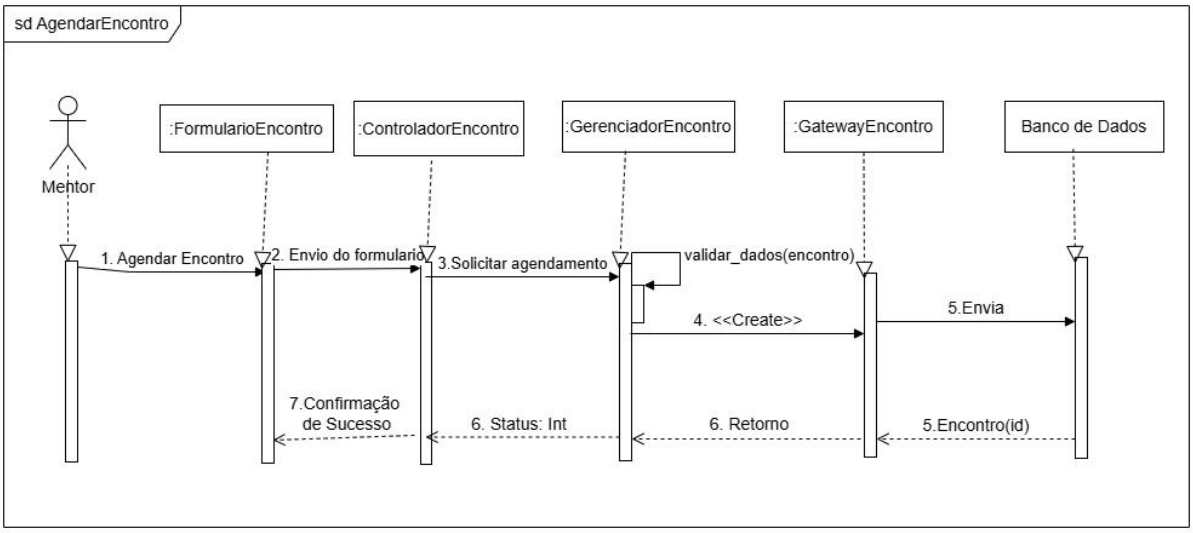
Figura 22 - Diagrama de Sequência BuscarMentor



Fonte: Autoria própria (2025)

O segundo cenário (Figura 22) ilustra uma operação de consulta focada na escalabilidade e na capacidade de adaptação a filtros. O mentorado inicia a busca com critérios específicos (curso, área, etc.) por meio do FormulárioMentor. O GerenciadorMentor é o componente responsável por processar esses critérios. A capacidade de expansão do sistema é evidente neste ponto, pois novos filtros podem ser adicionados ao Gerenciador sem impactar as camadas de apresentação ou persistência, o que atende ao cenário de escalabilidade. O GatewayMentor executa a consulta no banco de dados, e o resultado retornado ao mentorado já está processado e organizado. Este fluxo confirma que a interface é independente da lógica de consulta, reforçando também a adequação ao cenário de compatibilidade.

Figura 23 - Diagrama de Sequência AgendarEncontro



Fonte: Autoria própria (2025)

O terceiro cenário (Figura 23) detalha uma operação complexa que confirma a robustez das regras de negócio. O mentor envia a solicitação pelo FormulárioEncontro. O GerenciadorEncontro recebe a requisição e realiza verificações críticas, como a disponibilidade de horários. Em seguida, o ControladorEncontro é acionado para validar os dados obrigatórios, garantindo que o fluxo prossiga somente sob as condições corretas. Essa etapa reafirma a divisão clara entre controle e lógica de negócio. Por fim, o GatewayEncontro persiste o registro no banco de dados e retorna o identificador do encontro para que a interface possa informar o usuário com precisão.

Com base na análise dos cenários e na avaliação detalhada dos diagramas, conclui-se que a arquitetura proposta apresenta sólida separação de responsabilidades, modularidade e coerência com os requisitos do sistema. Os diagramas de sequência permitiram confirmar que a comunicação entre os componentes mantém-se estável e aderente às decisões arquiteturais tomadas, validando a solução como adequada, robusta e capaz de evoluir de forma estruturada conforme as necessidades do sistema.

8 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a modelagem e a arquitetura de software para um sistema de monitorias na UFERSA, fundamentando-se na metodologia de pesquisa-ação e nas boas práticas de engenharia de software. A investigação possibilitou compreender as necessidades reais do processo de monitoria, orientando a definição dos requisitos, a elaboração dos diagramas UML e a construção das visões arquiteturais.

A revisão teórica e a análise dos trabalhos relacionados evidenciaram a carência de soluções específicas para o gerenciamento de monitorias, reforçando a relevância da proposta. Os artefatos produzidos, incluindo casos de uso, classes, componentes, atributos de qualidade e visão arquitetural, fornecem uma base sólida para o desenvolvimento futuro do sistema, permitindo maior organização, transparência e eficiência no fluxo das atividades acadêmicas.

Embora o escopo tenha se limitado à modelagem, o estudo estabelece diretrizes claras para etapas posteriores, como implementação, testes e validação com usuários. Conclui-se que o objetivo foi atingido, oferecendo uma solução modelada de forma consistente e alinhada às demandas institucionais, capaz de contribuir para o aprimoramento das práticas de monitoria e para o suporte ao desempenho estudantil.

REFERÊNCIAS

ALVARENGA NETO, R. C. D. **Gestão do Conhecimento em Organizações**: proposta de mapeamento conceitual integrativo. São Paulo: Saraiva, 2005.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software architecture in practice**. 3. ed. Boston: Addison-Wesley, 2012.

BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software architecture in practice**. 4. ed. Addison-Wesley Professional, 2021.

BRAGA, Priscilla Even Alves. **A importância da gestão do conhecimento em instituições de ensino superior**: um estudo de caso na Universidade Federal Rural do Semiárido. 2022. Dissertação (Mestrado em Administração Pública) – Universidade Federal Rural do Semi-Árido, Mossoró, 2022.

BOOCH, Grady.; RUMBAUGH, James; JACOBSON, Ivar. **UML**: Guia do Usuário. 2. ed. Rio de Janeiro: Elsevier, 2005.

BROOKS, Frederick P. **The Mythical Man-Month**: Essays on Software Engineering. Anniversary Edition. Boston: Addison-Wesley, 1995.

CHOO, Chun Wei. **A organização do conhecimento: como as organizações usam a informação para criar significado, construir conhecimento e tomar decisões**. São Paulo: Senac, 2003.

DAVENPORT, Thomas H.; PRUSAK, Laurence. Conhecimento empresarial: como as organizações gerenciam o seu capital intelectual. Rio de Janeiro: Campus, 1998.

DOMÍNGUEZ, Arturo Hernández. **Engenharia de Software**: material didático. UAB / Universidade Federal de Alagoas, 2010. Disponível em: <https://educapes.capes.gov.br/bitstream/capes/177122/2/Material%20Didatico-Engenharia%20de%20Software.pdf> Acesso em: 5 out. 2025. Acesso em: 15 nov. 2025.

FERREIRA, L. G.; SANTOS, I. S.; SILVA, M. S. da; SILVA, B. de C. F. L. da; FERRAZ, R. D.; FERRAZ, R. de C. S. N. Aprendizagem da docência na pandemia: o Programa de Mentoria Online da UESB. **Educação & Formação**, [S. l.], v. 8, p. e10046, 2023. DOI:

10.25053/redufor.v8.e10046. Disponível em: <https://revistas.uece.br/index.php/redufor/article/view/10046>. Acesso em: 15 nov. 2025.

FOWLER, Martin. **UML Essencial**: Um Breve Guia para a Linguagem Padrão de Modelagem de Objetos. 3. ed. Porto Alegre: Bookman, 2005.

GUEDES, Gilleanes T. A. **UML 2: Uma Abordagem Prática**. São Paulo: Novatec, 2011.

KENSKI, Vani Moreira. Aprendizagem mediada pela tecnologia. **Revista Diálogo Educacional**, Curitiba, v. 4, n. 10, p. 47–56, set./dez. 2003.

LARMAN, C. Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Processo Unificado. 3. ed. Porto Alegre: Bookman, 2007.

LIMA, Ana Paula; COSTA, João. Mentoria acadêmica online: contribuições para a formação docente e o desenvolvimento de habilidades interpessoais. Educação & Sociedade, Campinas, v. 42, n. 147, p. 1125-1142, 2021.

MARTIN, Robert C. **Arquitetura Limpa**: O Guia do Artesão para Estrutura e Design de Software. Rio de Janeiro: Alta Books, 2019.

MENEZES, A. S. et al. Peer mentoring como estratégia de acolhimento ao estudante e adaptação ao método PBL. **Revista Brasileira de Educação Médica**, v. 45, n. 4, p. 1–8, 2021. DOI: 10.1590/1981-5271v45.4-20210152.

MORAN, J. M. **A Educação que desejamos**: novos desafios e como chegar lá. Papirus, 2015.

NONAKA, Ikujiro; TAKEUCHI, Hirotaka. **Criação do conhecimento na empresa**: como as empresas japonesas geram a dinâmica da inovação. Rio de Janeiro: Campus, 1997.

OMG. (2017). **Unified Modeling Language (UML) Version 2.5.1**. Object Management Group. Disponível em: <https://www.omg.org/spec/UML/2.5.1/>. Acesso: 15 nov. 2025.

PEREIRA, J.; SANTOS, M. (2022). Pesquisa-ação em engenharia de software: ciclos iterativos para inovação. RECIMA21, 13(1), 78-92. Disponível em: https://recima21.com.br/recima21/article/view/6670/4481.

PRESSMAN, Roger S. **Engenharia de Software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2010.

RICHTER, Cleitom José; BERNARDI, Giliane; CORDENONSI, André Zanki. O ensino de programação mediado por tecnologias educacionais: uma revisão sistemática de literatura. **Revista Novas Tecnologias na Educação**, v. 17, n. 1, p. 517-526, jul. 2019. DOI: 10.22456/1679-1916.95903. Disponível em: <https://seer.ufrgs.br/index.php/renote/article/view/95903/53907>. Acesso em: 15 nov. 2025.

ROBERTSON, Suzanne; ROBERTSON, James. **Mastering the Requirements Process**: Getting Requirements Right. 3. ed. Boston: Addison-Wesley, 2012.

SILVA, R. L.; PEREIRA, F. S. SADIS – Sistema de Acompanhamento e Mentoria Discente. **Revista de Gestão e Tecnologia Aplicada**, v. 12, n. 3, p. 45–59, 2021.

SILVA, A. et al. (2023). Aplicação da pesquisa-ação em contextos educacionais tecnológicos. **Temática**, 19(2), 123-145. Disponível em: https://periodicos.ufpb.br/index.php/tematica/article/view/74481/41656.

SILVA, Maria Lanuza dos Santos; SOUSA, Reudismam Rolim de. Uma Arquitetura de um Ambiente para a Associação de Apoio aos Portadores de Câncer de Mossoró e Região (AAPCMR). **Conjecturas**, vol. 22, n.13, p. 958–976, 2022.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. São Paulo: Pearson Addison Wesley, 2007.

SÓRIA, A. P.; MACUCH, R. da S. Revisão integrativa sobre programas de mentoria no ensino superior brasileiro. **Revista Brasileira de Ensino Superior**, v. 43, n. 4, p. 1–15, 2025. DOI: 10.1590/1981-5271v43n4-96936

THIOLLENT, Michel. **Metodologia da pesquisa-ação**. São Paulo: Cortez, 2011.

TRIPP, David. **Pesquisa-ação: uma introdução metodológica**. Educação e Pesquisa, São Paulo, v. 31, n. 3, p. 443-466, 2005.

VALENTE, J. A. **O computador na sociedade do conhecimento**. Campinas: UNICAMP, 1999.

VALENTE, Marco Tulio. **Engenharia de Software Moderna**. 1. ed. Minas Gerais: Independente, 2022.

WIEGERS, K.; BEATTY, J. **Software Requirements**. 3. ed. Redmond: Microsoft Press, 2013