Universidade Tiradentes
CIÊNCIA DA COMPUTAÇÃO

Ruan Víctor Alves da Costa
Roberto Maia Menezes
Raul Rodrigues de Souza
Maria Eduarda Souza Lima
Mateus Fraga Santos
Raphael Martins Carneiro Barbosa

PROJETO UNIDADE II

CADASTRO E GERENCIAMENTO DE AULAS

Aracaju - SE
2025

Ruan Víctor Alves da Costa
Roberto Maia Menezes
Raul Rodrigues de Souza
Maria Eduarda Souza Lima
Mateus Fraga Santos
Raphael Martins Carneiro Barbosa

PROJETO UNIDADE II

CADASTRO E GERENCIAMENTO DE AULAS

Atividade sobre Cadastro e Gerenciamento de Aulas apresentado como requisito parcial da avaliação da disciplina Projeto de Programação, ministrada pela Prof. Layse Santos Souza, no 2º semestre de 2025.

Aracaju - SE
2025

Sumário

1 INTRODUÇÃO

Este documento tem como finalidade apresentar o desenvolvimento do módulo "Cadastro e Gerenciamento de Aulas", componente integrante do sistema Plataforma de Gerenciamento de Cursos e Treinamentos de Véridia, elaborado como parte da disciplina Projeto de Programação.

O sistema foi planejado para atender à demanda da Prefeitura de Véridia, que busca digitalizar e otimizar a oferta de cursos e treinamentos oferecidos por instituições públicas e privadas. A plataforma geral engloba diversos módulos: Usuários, Cursos, Aulas, Inscrições e Pagamentos, Agenda, Notificações e Relatórios. O Grupo 3 foi responsável pela implementação integral do módulo de Aulas.

O módulo desenvolvido tem como principal propósito gerenciar a criação, edição, visualização e cancelamento de aulas, controlando a disponibilidade de instrutores, locais e horários e garantindo a integridade dos dados por meio de validações automáticas. Além disso, implementa um sistema de notificações que mantém os usuários informados sobre alterações na programação.

O projeto foi implementado em Java, utilizando JavaFX para a interface gráfica, FXML para a definição das telas, Scene Builder para o design visual, JPA (Java Persistence API) para o mapeamento objeto-relacional e banco de dados H2 para persistência de dados. A arquitetura adotada foi o padrão MVC (Model–View–Controller), garantindo modularidade, coesão e facilidade de manutenção.

2 JUSTIFICATIVA

A necessidade de um sistema que automatize a criação e o controle de aulas é evidente em contextos educacionais complexos, onde múltiplos cursos e instrutores compartilham recursos. O módulo de aulas surge como uma resposta direta à demanda da cidade de Véridia por um sistema capaz de eliminar sobreposições de horários, organizar recursos e otimizar a comunicação entre os envolvidos.

Durante o desenvolvimento, o grupo buscou alinhar boas práticas de engenharia de software com princípios de usabilidade e eficiência, integrando conceitos estudados na disciplina**,** como integração com Banco de Dados, Programação Orientada a Objetos e Interface Gráfica. Essa abordagem favoreceu uma aplicação modular e escalável, que pode ser facilmente mantida e expandida por outros grupos de desenvolvimento.

A escolha da linguagem Java se justifica por sua robustez, portabilidade e ampla adoção tanto no meio acadêmico quanto corporativo. O uso do JavaFX permitiu o desenvolvimento de uma interface moderna e interativa, enquanto o FXML foi empregado para a estruturação das telas e o CSS para a estilização visual, separando de forma clara o design da lógica de programação.

O Scene Builder foi utilizado como ferramenta de apoio à construção das interfaces, o que permitiu criar janelas, botões e painéis de forma visual, reduzindo a complexidade do código e aumentando a produtividade. O FXML, em conjunto com os controllers Java, segue o padrão MVC (Model–View–Controller), garantindo baixo acoplamento entre camadas e facilitando a manutenção e os testes.

Para garantir a persistência dos dados, foi utilizado o banco de dados H2, acessado através do JPA (Java Persistence API), o que proporcionou uma comunicação eficiente entre o sistema e o banco de dados relacional por meio de mapeamento objeto-relacional. Essa configuração permitiu registrar, consultar e atualizar informações sobre aulas, instrutores, cursos e locais de forma segura e integrada.

O projeto foi gerenciado e construído com o auxílio do Maven, que organizou dependências e builds do sistema, além de facilitar a integração entre bibliotecas externas. Todo o desenvolvimento foi versionado no GitHub, o que possibilitou controle de versões, rastreamento de alterações e colaboração simultânea entre os integrantes do grupo.

3 OBJETIVOS

3.1 OBJETIVO GERAL

O objetivo geral deste projeto é desenvolver um módulo completo de Cadastro e Gerenciamento de Aulas integrado à Plataforma de Gerenciamento de Cursos e Treinamentos de Véridia, atendendo às necessidades de controle, organização e automação das atividades pedagógicas.

O sistema tem como propósito permitir que administradores e instrutores realizem a visualização, edição e cancelamento de aulas de forma eficiente, enquanto os alunos podem consultar as aulas disponíveis, verificando horários, cursos e instrutores correspondentes.

O módulo foi planejado para eliminar sobreposições de horários, garantir a consistência de dados e oferecer uma experiência de uso clara e intuitiva. Além disso, busca consolidar os conceitos de programação orientada a objetos, persistência de dados e arquitetura MVC, utilizando Java, JavaFX, FXML, H2 e JPA como principais tecnologias.

Por meio deste desenvolvimento, pretende-se também proporcionar ao grupo uma vivência prática em engenharia de software, aplicando princípios de modularização, versionamento de código e documentação acadêmica.

3.2 OBJETIVOS ESPECÍFICOS

1. Implementar o gerenciamento de aulas, permitindo o cadastro, edição, visualização e cancelamento, vinculando-as a cursos, instrutores e locais cadastrados no sistema.

2. Aplicar validações automáticas de conflito de horários, assegurando que um mesmo instrutor ou local não seja alocado em duas aulas simultaneamente, utilizando algoritmos de verificação temporal desenvolvidos em Java.

3. Desenvolver uma interface gráfica moderna e responsiva, utilizando JavaFX em conjunto com FXML e CSS, garantindo uma navegação intuitiva e uma experiência de usuário agradável.

4. Integrar o módulo de Aulas com as entidades do sistema principal, como Curso, Local e Inscrição, assegurando comunicação consistente entre as camadas de dados e controle.

5. Implementar a persistência de dados por meio do banco H2, utilizando JDBC para gerenciar operações de inserção, atualização e exclusão, garantindo segurança e integridade nas informações.

6. Adotar boas práticas de programação e arquitetura, estruturando o código no padrão MVC (Model–View–Controller), com baixo acoplamento e alta coesão entre as classes.

7. Utilizar ferramentas profissionais de apoio ao desenvolvimento, como Maven para gerenciamento de dependências e GitHub para versionamento e colaboração em equipe.

8. Gerar documentação técnica e acadêmica, incluindo diagramas UML (classes e casos de uso), descrição detalhada de métodos e entidades, e relatório final no formato ABNT.

9. Validar o sistema por meio de testes funcionais, assegurando o correto funcionamento das operações principais e a integração entre os módulos da plataforma.

4 METODOLOGIA

O desenvolvimento do módulo Cadastro e Gerenciamento de Aulas ocorreu usando uma estratégia gradual e repetitiva, respeitando as fases descritas no planejamento de execução criado pela professora Ma. Layse Souza na disciplina Projeto de Programação. Essa abordagem permitiu que cada entrega semanal representasse uma evolução funcional do sistema, chegando à versão completa unida à Plataforma de Cursos e Capacitações de Véridia.

O projeto teve uma construção em equipe, feita pelos membros do Grupo 3, levando em conta as responsabilidades individuais apresentados no primeiro módulo e melhorados no segundo módulo. Cada membro ficou responsável por partes específicas, mas todos opinaram nas escolhas de formato, estrutura e junção. Essa organização do trabalho estimulou a união e a experiência de um cenário de criação verdadeiro, com controle de versões constante pelo GitHub.

4.1 Etapas de Desenvolvimento

O processo de desenvolvimento do sistema foi dividido em cinco fases principais: análise, modelagem, implementação, integração e testes.

Na fase de análise, foram definidos os requisitos funcionais e não funcionais do módulo, com base na proposta do projeto da cidade de Véridia. Nessa etapa, foram identificados os atores envolvidos, administrador, instrutor e aluno, as operações principais, criar, editar, visualizar e cancelar aulas, e as regras de negócio, como validação de horários, limite de capacidade e vínculos entre cursos e instrutores.

A etapa de modelagem envolveu a criação dos diagramas UML, representando classes, relacionamentos e casos de uso. O diagrama de classes apresentou a hierarquia das entidades principais (Usuário, Administrador, Instrutor, Aluno, Curso, Aula, Local e Inscrição), enquanto o diagrama de casos de uso definiu as permissões e interações de cada tipo de usuário.

Na fase de implementação, as classes foram codificadas em Java, estruturadas segundo o padrão MVC (Model–View–Controller). O grupo utilizou o JavaFX para a construção da interface gráfica, com telas declaradas em FXML, estilizadas com CSS e integradas a controladores (Controller) responsáveis pela lógica e eventos. Essa separação entre as camadas de visualização e lógica garantiu uma arquitetura organizada e de fácil manutenção.

Durante a integração, os módulos foram conectados entre si e ao banco de dados H2, utilizando JPA (Java Persistence API) para realizar o mapeamento objeto-relacional e a manipulação das tabelas e registros. O gerenciamento das dependências foi feito com o Maven, garantindo compatibilidade e controle das bibliotecas utilizadas.

Por fim, na etapa de testes, foram executadas simulações e execuções reais no ambiente JavaFX, verificando a integridade dos dados, o comportamento das telas e as respostas do sistema às ações do usuário. Os testes buscaram validar o correto funcionamento dos algoritmos de verificação de conflitos e de atualização automática de informações.

4.2 Arquitetura e Estrutura de Pastas

O projeto foi estruturado de forma modular, visando clareza e reuso de código. A organização de pastas segue o padrão MVC, conforme descrito a seguir:

```
Projeto-de-Programação_N02_Grupo3/
├── src/
│   ├── model/
│   │   ├── Aula.java
│   │   ├── Curso.java
│   │   ├── Local.java
│   │   ├── Usuario.java
│   │   ├── Instrutor.java
│   │   ├── Administrador.java
│   │   ├── Aluno.java
│   │   ├── Inscricao.java
│   │   ├── Notificacão.java
│   │   └── Permlssao.java
│   ├── service/
│   │   ├── AulaContrroller.java
│   │   ├── CursoController.java
│   │   ├── InstrutorController.ja~
│   │   ├── UsuãrioController.java
│   │   ├── LocalController.java
│   │   ├── InscricaoController.java
│   │   ├── NotificacãoController.javva
│   │   ├── ValidacaoController.java
│   │   ├── AgendaController.java
│   │   └── GlobalExceptionHandleler.java
│   ├── view/
│   │   ├── main.fxml
│   │   ├── cadastroAula.fxml
│   │   ├── editarAula.fxml
│   │   ├── aulas.fxml
│   │   ├── cursos.fxml
│   │   ├── locals.fxml
│   │   └── instrutores.fxml
│   ├── resources/
│   │   ├── styles.css
│   │   ├── layout.css
│   │   ├── main.css
│   │   ├── table.css
│   │   ├── database.sgl
│   │   └── application.proprties
```

A camada Model contém as entidades e suas regras de negócio, enquanto a camada Controller gerencia as ações dos usuários e realiza a comunicação com o banco de dados. A camada View abriga os arquivos FXML e CSS, responsáveis pela interface visual.

Essa divisão permitiu que a aplicação mantivesse baixo acoplamento entre módulos, facilitando futuras expansões, como a integração com APIs externas, relatórios e autenticação.

4.3 Diagramas

Casos de Uso

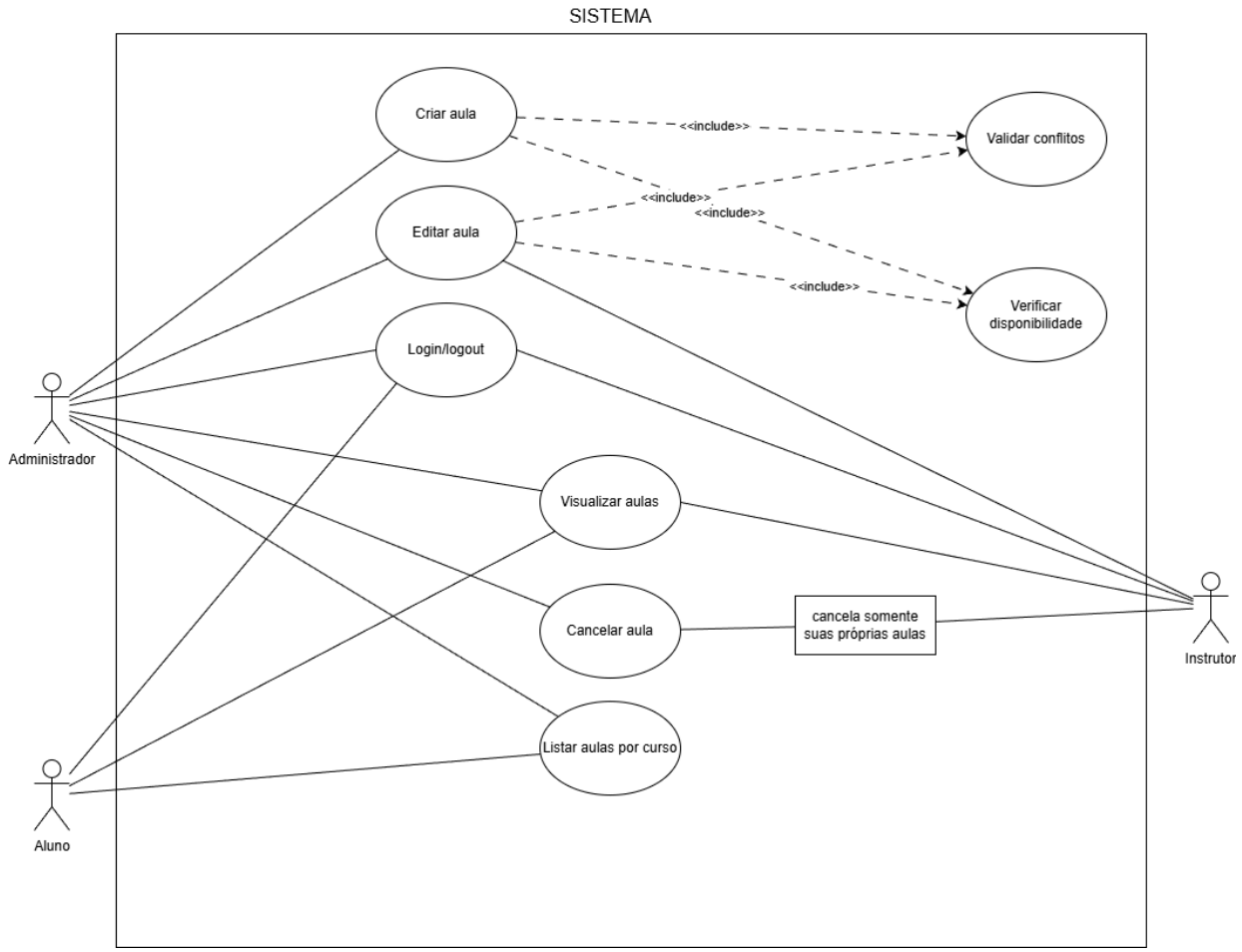

Figura 1 - Diagrama de Casos de Uso

UC01 – Login/Logout

Permite que o usuário entre e saia do sistema de forma segura.

- Atores: Administrador, Instrutor e Aluno.
- Pós-condição: Sessão iniciada ou encerrada.
- Fluxo principal: O usuário informa login, o sistema valida e libera o acesso.
- Exceção: Credenciais incorretas → mensagem de erro.

UC02 – Criar Aula

Permite cadastrar uma nova aula no sistema.

- Atores: Administrador e Instrutor.
- Pré-condição: Curso, instrutor e local já cadastrados.
- Pós-condição: Aula criada e salva no banco de dados.
- Fluxo principal: O usuário preenche os dados da aula e confirma o cadastro.
- Inclui: Verificar disponibilidade e Validar conflitos.

- Exceção: Caso exista conflito de horário ou local indisponível, o sistema não permite o cadastro.

UC03 – Editar Aula

Permite alterar informações de uma aula existente.

- Atores: Administrador e Instrutor (somente suas próprias aulas).
- Pré-condição: Aula já cadastrada.
- Pós-condição: Aula atualizada.
- Fluxo principal: O usuário seleciona a aula e edita os dados desejados.
- Inclui: Validar conflitos e Verificar disponibilidade.

UC04 – Visualizar Aulas

Permite visualizar a lista de aulas cadastradas.

- Atores: Administrador, Instrutor e Aluno.
- Pré-condição: Usuário logado.
- Pós-condição: Lista de aulas exibida na tela.
- Fluxo principal: O usuário acessa a tela de aulas e o sistema mostra todas as aulas disponíveis.

UC05 – Cancelar Aula

Permite cancelar uma aula já marcada.

- Atores: Administrador e Instrutor.
- Pré-condição: Aula cadastrada e ainda não iniciada.
- Pós-condição: Aula marcada como cancelada.
- Fluxo principal: O usuário seleciona a aula, informa o motivo e confirma o cancelamento.

UC06 – Listar Aulas por Curso

Permite visualizar aulas específicas de um curso.

- Atores: Administrador, Instrutor e Aluno.
- Pré-condição: Curso existente.
- Pós-condição: Aulas do curso exibidas
- Fluxo principal: O usuário escolhe um curso e o sistema mostra as aulas relacionadas.

UC07 – Validar Conflitos

Verifica se há sobreposição de horários para o mesmo local ou instrutor.

- Atores: Sistema (interno).
- Fluxo principal: O sistema compara horários e retorna aviso caso haja conflito.

UC08 – Verificar Disponibilidade

Checa se o instrutor e o local estão livres no horário escolhido.

- Atores: Sistema (interno).
- Fluxo principal: O sistema valida a disponibilidade e informa se o cadastro pode prosseguir.

Código	Nome do Caso de Uso	Atores Envolvidos	Descrição / Objetivo	Pré-condição	Pós-condição
UC01	Login / Logout	Administrador, Instrutor, Aluno	Permite que o usuário entre e saia do sistema de forma segura.	Usuário cadastrado.	Sessão iniciada ou encerrada.
UC02	Criar Aula	Administrador, Instrutor	Cadastrar uma nova aula vinculada a curso, instrutor e local.	Curso, instrutor e local existentes.	Aula criada e salva no sistema.
UC03	Editar Aula	Administrador, Instrutor	Alterar dados de uma aula já cadastrada.	Aula existente.	Aula atualizada.
UC04	Visualizar Aulas	Administrador, Instrutor, Aluno	Consultar aulas disponíveis conforme filtros.	Usuário logado.	Lista de aulas exibida.
UC05	Cancelar Aula	Administrador, Instrutor	Cancelar uma aula ainda não iniciada e registrar o motivo.	Aula cadastrada.	Aula cancelada.
UC06	Listar Aulas por Curso	Administrador, Instrutor, Aluno	Exibir aulas de um curso específico.	Curso existente.	Aulas listadas na tela.
UC07	Validar Conflitos	Sistema (interno)	Verificar sobreposição de horários entre aulas.	Dados da aula informados.	Resultado da validação.
UC08	Verificar Disponibilidade	Sistema (interno)	Checar se instrutor e local estão disponíveis no horário informado.	Dados da aula informados.	Resultado da verificação.

Diagrama de Classes

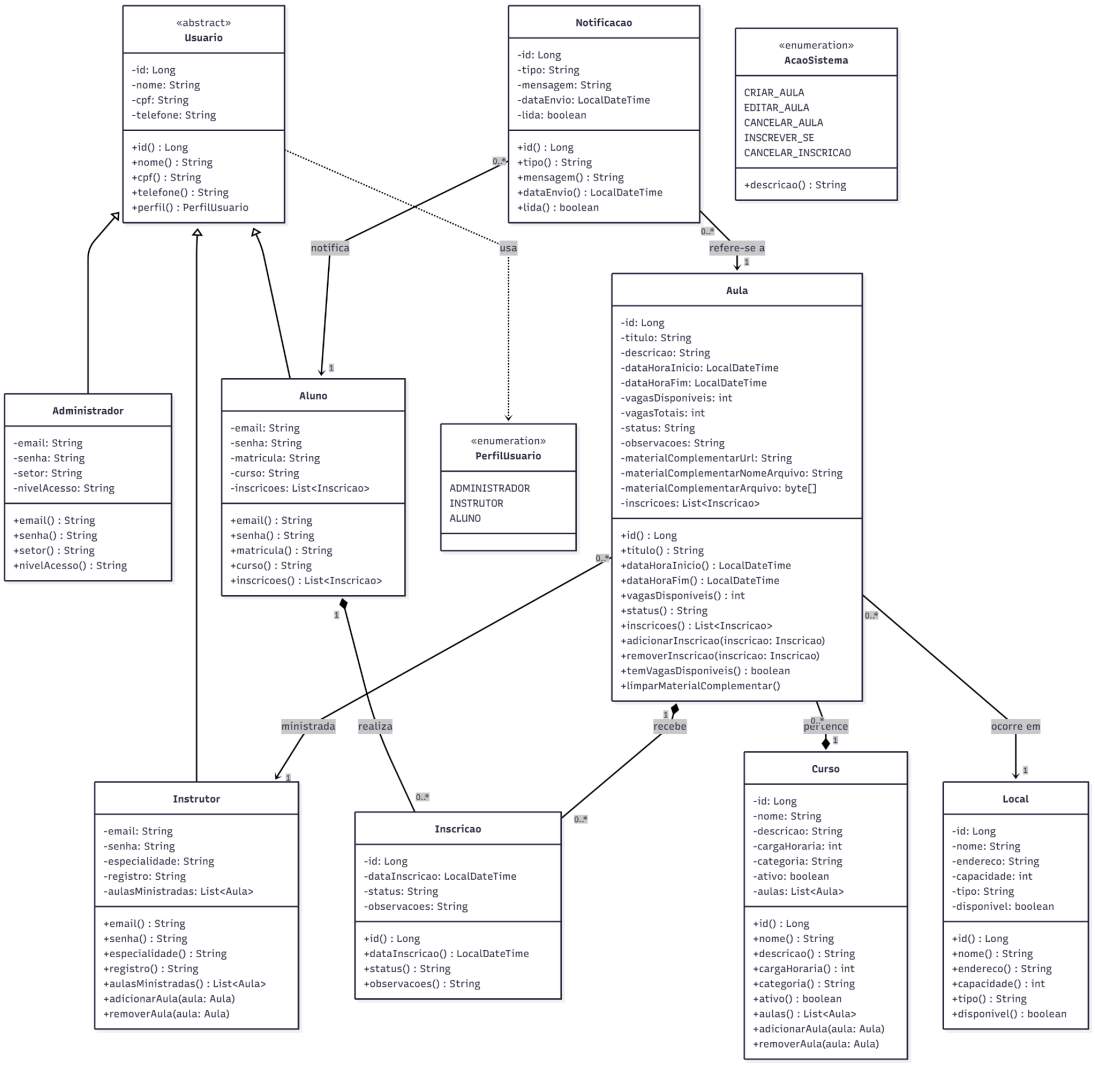

Figura 2: Diagrama de Classes

O diagrama de classes elaborado representa a estrutura estática do sistema de gerenciamento de aulas, evidenciando as principais entidades, seus atributos, operações e os relacionamentos entre elas. O objetivo desse diagrama é demonstrar como os elementos do sistema estão organizados e de que forma interagem entre si dentro do modelo orientado a objetos.

O sistema possui como base a classe abstrata Usuário, que reúne os atributos e comportamentos comuns a todos os perfis, como identificação, nome, CPF e telefone. A partir dela, derivam as subclasses Administrador, Instrutor e Aluno, que representam os diferentes papéis no sistema e especializam suas funções.

- O Administrador é o usuário responsável pelo controle e pela gestão geral do sistema. Ele possui atributos específicos, como setor e nível de acesso, e tem autoridade para realizar ações administrativas, como criar, editar ou cancelar aulas.
- O Instrutor contém informações profissionais, como especialidade e registro, além de manter uma lista de aulas ministradas.
- O Aluno possui matrícula, vínculo com um curso e uma lista de inscrições, que registram sua participação nas aulas disponíveis.

Essa estrutura hierárquica facilita o reaproveitamento de código e permite o uso de polimorfismo, tornando o sistema mais flexível e modular.

As demais classes representam as entidades de domínio relacionadas ao processo de ensino. A classe Curso agrupa um conjunto de aulas e contém atributos como nome, descrição, carga horária e categoria. O relacionamento entre Curso e Aula é de composição, indicando que as aulas estão diretamente associadas ao curso e deixam de existir se ele for excluído.

A classe Aula é o elemento central do sistema, reunindo informações como título, descrição, data e horário, quantidade de vagas e status. Ela também pode armazenar materiais complementares, que podem ser representados por uma URL ou por um arquivo enviado diretamente ao sistema. Cada aula está vinculada a um Local, representando o ambiente físico ou virtual em que ocorre, e mantém uma lista de Inscrições, que associam os alunos às aulas.

A classe Local contém informações sobre o espaço onde as aulas acontecem, como nome, endereço, capacidade e tipo, além de um atributo que indica sua disponibilidade. Essa relação é unidirecional, uma vez que a aula depende do local, mas o local não precisa conhecer as aulas associadas a ele.

A classe Inscrição é responsável por registrar a participação do aluno em uma aula específica, armazenando dados como a data da inscrição, o status e eventuais observações. Essa entidade é fundamental para o controle de vagas e acompanhamento da participação dos alunos.

A classe Notificação complementa o modelo ao representar as mensagens enviadas aos usuários, geralmente alunos, sobre eventos ou alterações no sistema. Cada notificação possui atributos como tipo, mensagem, data de envio e status de leitura, e está associada tanto à aula que originou o aviso quanto ao aluno que a recebe.

O diagrama também inclui duas enumerações: PerfilUsuario e AcaoSistema. A primeira define os tipos de usuários existentes (administrador, instrutor e aluno), enquanto a segunda descreve as ações que podem ser realizadas no sistema, como criar, editar e cancelar aulas, ou inscrever-se e cancelar inscrições. Essas enumerações contribuem para a padronização das operações e auxiliam no controle de permissões.

Os relacionamentos entre as classes são definidos conforme sua natureza:

- A herança foi aplicada entre a classe Usuário e suas subclasses, refletindo a especialização dos perfis do sistema.
- As composições estão presentes entre Curso e Aula, e entre Aula e Inscrição, demonstrando dependência forte entre esses elementos.
- As associações bidirecionais foram utilizadas entre Instrutor e Aula, permitindo que ambos mantenham referências mútuas.
- As associações unidirecionais aparecem nas ligações de Aula com Local, e de Notificação com Aluno e Aula, representando dependências em apenas um sentido.

De modo geral, o diagrama de classes oferece uma visão estruturada e detalhada da arquitetura lógica do sistema de gerenciamento de aulas. Ele descreve a forma como os dados e as responsabilidades estão distribuídos, servindo de base conceitual para o desenvolvimento das funcionalidades do software.

Diagrama de Entidade Relacionamento

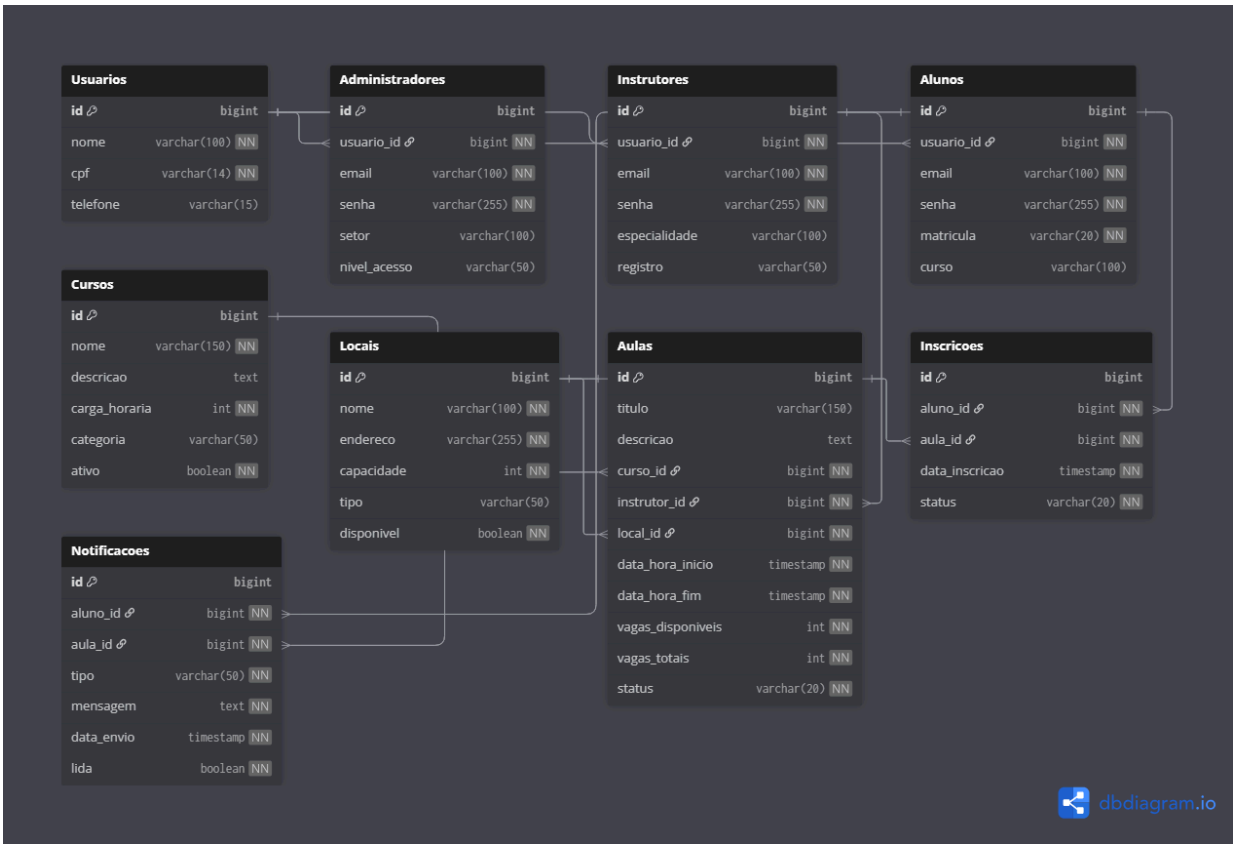

Figura 3: Diagrama de entidade relacionamento

1. Introdução

O presente documento descreve o Diagrama de Entidade-Relacionamento (DER) projetado para o sistema de gerenciamento de aulas. Este diagrama modela a arquitetura lógica do banco de dados, definindo as entidades de domínio, seus respectivos atributos, e os relacionamentos de cardinalidade entre elas. O objetivo é estabelecer uma estrutura de dados coesa e íntegra que servirá como base para o armazenamento e a recuperação das informações do sistema.

2. Descrição das Entidades e Relacionamentos

O modelo de dados é centrado na gestão de usuários, aulas e suas interações, implementando um padrão de especialização para os perfis de usuário e utilizando entidades associativas para gerenciar inscrições e notificações.

2.1. Entidades de Usuário (Generalização e Especialização)

O sistema utiliza um padrão de generalização/especialização para gerenciar os diferentes tipos de usuários:

- Usuarios: Esta é a entidade base (superclasse) que armazena os dados comuns a todos os perfis, como id, nome, cpf e telefone.
- Administradores, Instrutores e Alunos: São entidades especializadas (subclasses) que herdam as características de Usuarios. Cada uma delas possui um relacionamento um-para-um (1:1) com a entidade Usuarios, estabelecido pela chave estrangeira (FK) usuario_id.
    - Administradores: Armazena dados específicos de gestão, como setor e

nivel_acesso.
- Instrutores: Contém informações profissionais, como especialidade e registro.
- Alunos: Mantém dados acadêmicos, como matrícula e curso.
- Nota: As entidades especializadas também armazenam dados de autenticação (email e senha), o que sugere que apenas esses perfis podem se autenticar no sistema.

2.2. Entidades Centrais do Domínio

Estas entidades representam os principais conceitos do negócio de gerenciamento de aulas:

- Cursos: Agrupa um conjunto de aulas. Define o nome do curso, descricao, carga_horaria, categoria e um indicador ativo.
- Locais: Descreve os locais físicos ou virtuais onde as aulas ocorrem. Inclui nome, endereco, capacidade máxima, tipo de local e um indicador disponivel.
- Aulas: É a entidade central do diagrama, conectando os diversos elementos do sistema. Ela armazena informações detalhadas sobre cada sessão de ensino, como título, descricao, data_hora_inicio, data_hora_fim, vagas_totais, vagas_disponiveis e status.

2.3. Relacionamentos da Entidade Aulas

A entidade Aulas se relaciona com as demais entidades centrais através de relacionamentos um-para-muitos (1:N):

- Cursos (1) -> Aulas (N): Um curso pode ter várias aulas, mas cada aula pertence a um único curso (via curso_id).
- Instrutores (1) -> Aulas (N): Um instrutor pode ministrar várias aulas, mas cada aula é ministrada por um único instrutor (via instrutor_id).
- Locais (1) -> Aulas (N): Um local pode sediar várias aulas (em horários diferentes), mas cada aula ocorre em um único local (via local_id).

2.4. Entidades Associativas e de Junção

Estas entidades são usadas para modelar relacionamentos complexos e registrar interações no sistema:

- Inscricoes: Esta é uma entidade associativa (ou tabela de junção) que implementa o relacionamento muitos-para-muitos (N:M) entre Alunos e Aulas.
  - Um aluno pode se inscrever em múltiplas aulas.
  - Uma aula pode ter múltiplos alunos inscritos.
  - A entidade Inscricoes armazena os atributos desse relacionamento, como data_inscricao e o status (ex: "Confirmada", "Pendente", "Cancelada"). Ela se conecta às duas entidades através das chaves estrangeiras aluno_id e aula_id.
- Notificacoes: Esta entidade registra as mensagens enviadas aos usuários. Ela possui dois relacionamentos muitos-para-um (N:1):
  - Alunos (1) -> Notificacoes (N): Um aluno pode receber múltiplas notificações (via aluno_id).

4.4 Requisitos

REQUISITOS FUNCIONAIS

RF01 - Gerenciamento de Aulas

- RF01.1: O sistema deve permitir o cadastro de aulas com título, descrição, data/hora de início e fim.
- RF01.2: O sistema deve permitir vincular aulas a cursos específicos.
- RF01.3: O sistema deve permitir vincular aulas a instrutores.
- RF01.4: O sistema deve permitir vincular aulas a locais.
- RF01.5: O sistema deve permitir visualizar, editar e excluir aulas cadastradas.
- RF01.6: O sistema deve permitir definir número de vagas totais e vagas disponíveis por aula.
- RF01.7: O sistema deve permitir adicionar observações às aulas.

RF02 - Controle de Datas e Horários

- RF02.1: O sistema deve validar que a data/hora de fim seja posterior à data/hora de início.
- RF02.2: O sistema deve permitir filtrar aulas por período (futuras, passadas).
- RF02.3: O sistema deve exibir aulas em ordem cronológica.

RF03 - Reagendamento de Aulas

- RF03.1: O sistema deve permitir alterar data e horário de aulas.
- RF03.2: O sistema deve permitir alterar o local de aulas.
- RF03.3: O sistema deve notificar automaticamente alunos inscritos sobre reagendamentos.
- RF03.4: O sistema deve validar conflitos ao reagendar aulas.

RF04 - Verificação de Disponibilidade

- RF04.1: O sistema deve detectar conflitos de horário para o mesmo instrutor.
- RF04.2: O sistema deve detectar conflitos de horário para o mesmo local.
- RF04.3: O sistema deve validar a capacidade do local antes de agendar aulas.
- RF04.4: O sistema deve exibir alertas visuais quando houver conflitos.
- RF04.5: O sistema deve impedir o cadastro de aulas com conflitos.

RF05 - Material Complementar

- RF05.1: O sistema deve permitir upload de arquivos PDF como material de aula.
- RF05.2: O sistema deve permitir cadastro de links externos como material complementar.
- RF05.3: O sistema deve permitir download de materiais cadastrados.
- RF05.4: O sistema deve armazenar o tipo e nome do arquivo enviado.

RF06 - Gerenciamento de Cursos

- RF06.1: O sistema deve permitir cadastrar cursos.
- RF06.2: O sistema deve permitir listar todas as aulas de um curso específico.
- RF06.3: O sistema deve permitir exportar lista de aulas por curso em formato CSV.

RF07 - Gerenciamento de Instrutores

- RF07.1: O sistema deve permitir cadastrar instrutores (herança de Usuário).
- RF07.2: O sistema deve permitir visualizar aulas atribuídas a cada instrutor.
- RF07.3: O sistema deve validar disponibilidade do instrutor antes de atribuir novas aulas.

RF08 - Gerenciamento de Locais

- RF08.1: O sistema deve permitir cadastrar locais.
- RF08.2: O sistema deve armazenar a capacidade máxima de cada local.
- RF08.3: O sistema deve verificar disponibilidade de locais por horário.

RF09 - Sistema de Inscrições

- RF09.1: O sistema deve permitir que alunos se inscrevam em aulas.
- RF09.2: O sistema deve controlar o número de vagas disponíveis.
- RF09.3: O sistema deve impedir inscrições quando não houver vagas.

RF10 - Visualização e Relatórios

- RF10.1: O sistema deve exibir dashboard com estatísticas gerais.
- RF10.2: O sistema deve permitir filtrar aulas por diversos critérios.
- RF10.3: O sistema deve listar aulas futuras.
- RF10.4: O sistema deve listar aulas com vagas disponíveis.
- RF10.5: O sistema deve exportar dados em formato CSV.

RF11 - API REST

- RF11.1: O sistema deve fornecer endpoints REST para todas as operações CRUD.
- RF11.2: O sistema deve permitir operações via API na porta 9090.
- RF11.3: O sistema deve retornar dados em formato JS

REQUISITOS NÃO FUNCIONAIS

RNF01 - Tecnologia

- RNF01.1: O sistema deve ser desenvolvido em Java 21.
- RNF01.2: O sistema deve utilizar Spring Boot 3.5.6 como framework backend.
- RNF01.3: O sistema deve utilizar JavaFX 21 para interface gráfica.
- RNF01.4: O sistema deve utilizar Spring Data JPA para persistência.
- RNF01.5: O sistema deve utilizar Maven como gerenciador de dependências.

RNF02 - Banco de Dados

- RNF02.1: O sistema deve utilizar banco de dados H2 embarcado.
- RNF02.2: O sistema deve persistir dados em arquivo.
- RNF02.3: O sistema deve manter integridade referencial entre entidades.
- RNF02.4: O sistema deve popular dados iniciais via script data.sql.

RNF03 - Interface e Usabilidade

- RNF03.1: A interface deve seguir princípios do Google Material Design.
- RNF03.2: A interface deve ser intuitiva e moderna.
- RNF03.3: O sistema deve fornecer feedback visual para todas as ações.
- RNF03.4: O sistema deve exibir alertas visuais para conflitos e erros.
- RNF03.5: As tabelas devem ter ações inline (editar, deletar, cancelar).

RNF04 - Arquitetura

- RNF04.1: O sistema deve seguir arquitetura em camadas (Controller, Service, Repository).
- RNF04.2: O código deve seguir boas práticas de Clean Code.
- RNF04.3: O sistema deve utilizar DTOs (Data Transfer Objects) para transferência de dados.
- RNF04.4: O sistema deve tratar exceções de forma customizada.
- RNF04.5: O sistema deve separar lógica de negócio da camada de apresentação.

RNF05 - Desempenho

- RNF05.1: As consultas ao banco devem ser otimizadas.
- RNF05.2: A interface deve ser responsiva e fluida.
- RNF05.3: O sistema deve carregar dados iniciais automaticamente.

RNF06 - Confiabilidade

- RNF06.1: O sistema deve validar todos os dados de entrada.
- RNF06.2: O sistema deve impedir operações que violem regras de negócio.
- RNF06.3: O sistema deve realizar validações em todas as camadas.
- RNF06.4: O sistema deve garantir consistência dos dados.

RNF07 - Manutenibilidade

- RNF07.1: O código deve ser organizado em pacotes bem definidos.
- RNF07.2: O projeto deve ter estrutura clara e documentada.
- RNF07.3: O sistema deve permitir fácil adição de novas funcionalidades.
- RNF07.4: Os arquivos FXML devem estar separados do código Java.

RNF08 - Portabilidade

- RNF08.1: O sistema deve executar em qualquer SO com Java 21 instalado.
- RNF08.2: O sistema deve ser empacotável via Maven.
- RNF08.3: O sistema deve ter banco de dados embarcado (não requer instalação .externa)

RNF09 - Segurança

- RNF09.1: O acesso ao banco deve ser protegido por credenciais.
- RNF09.2: O sistema deve validar uploads de arquivos (apenas PDF).
- RNF09.3: Os dados devem ser armazenados de forma segura.

RNF10 - Acessibilidade

- RNF10.1: O sistema deve fornecer API REST acessível via HTTP.
- RNF10.2: O sistema deve ser facilmente instalável via scripts automatizados.

5 RESULTADOS E DISCUSSÕES

O sistema desenvolvido atendeu integralmente aos requisitos estabelecidos, entregando uma solução completa e funcional para o gerenciamento de aulas da Prefeitura de Véridia. Os resultados obtidos são apresentados a seguir:

Funcionalidades Implementadas:

1. CRUD Completo de Aulas: O sistema permite criar, visualizar, atualizar e excluir aulas com interface intuitiva e responsiva. Todas as operações são validadas e persistidas corretamente no banco de dados;

2. Detecção Automática de Conflitos: Implementação robusta de validação que identifica:
   - Conflitos de instrutor (mesmo instrutor alocado em horários sobrepostos)
   - Conflitos de local (mesma sala reservada simultaneamente)
   - Validação de capacidade do local em relação às vagas oferecidas
   - Alertas visuais na interface destacando possíveis conflitos;

3. Sistema de Reagendamento: Funcionalidade completa que permite alterar data, horário ou local de uma aula, com:
   - Validação automática de novos conflitos antes de confirmar
   - Notificação automática aos alunos inscritos
   - Registro de histórico de alterações;

4. Gerenciamento de Materiais Complementares: Sistema implementado para:
   - Upload de arquivos PDF
   - Cadastro de links externos para materiais online

5. Exportação de Dados: Funcionalidade de exportação da lista de aulas por curso em formato CSV, facilitando análises e relatórios;

6. API REST Completa: Implementação de 10 endpoints principais:
   - GET /api/aulas - Listar todas as aulas
   - GET /api/aulas/{id} - Buscar aula específica
   - POST /api/aulas - Criar nova aula
   - PUT /api/aulas/{id} - Atualizar aula
   - PATCH /api/aulas/{id}/reagendar - Reagendar aula
   - DELETE /api/aulas/{id} - Deletar aula
   - GET /api/aulas/curso/{cursoId} - Listar aulas por curso
   - GET /api/aulas/curso/{cursoId}/exportar-csv - Exportar CSV
   - GET /api/aulas/futuras - Listar aulas futuras
   - GET /api/aulas/disponiveis - Listar aulas com vagas;

7. Interface Gráfica Moderna: Desenvolvimento de interface inspirada no Google Material Design com:
  - Dashboard com estatísticas em tempo real
  - Tabelas com ações inline (editar, cancelar, deletar)
  - Filtros inteligentes para busca de aulas
  - Navegação intuitiva entre módulos (Aulas, Cursos, Instrutores, Locais, Inscrições)
  - Design responsivo e visualmente atraente.

Dados de Teste:

O sistema foi populado com dados de exemplo para demonstração:
- 5 Cursos cadastrados em diferentes áreas
- 5 Instrutores com especialidades variadas
- 5 Locais (salas e laboratórios) com capacidades diferentes
- 10 Alunos registrados
- 25 Aulas de exemplo distribuídas ao longo do semestre

Discussão sobre Desafios Enfrentados:

1. Problema de Herança JPA: Um problema que inicialmente foi identificado e estava relacionado à configuração de herança das entidades JPA. A solução foi implementar corretamente a estratégia de herança e ajustar o arquivo data.sql, conforme documentado em CORRECOES_APLICADAS.md;

2. Validação de Conflitos Complexos: Foi necessário um cuidado especial com os fusos horários e a comparação de datas na implementação da validação de sobreposição de horários. A solução final faz uso de LocalDateTime da Java Time API, que oferece maior precisão;

3. Design da Interface: Balancear funcionalidade e estética exigiu diversas iterações. A adoção do Material Design como referência forneceu diretrizes claras para a tomada de decisões de design;

4. Gerenciamento de Estado: Sincronizar o estado da interface com o banco de dados em operações concorrentes foi resolvido com a atualização automática das tabelas após cada operação.

Comparação com Soluções Existentes:

Comparado a sistemas similares no mercado, o sistema desenvolvido apresenta vantagens como:
- Instalação simplificada (sem necessidade de servidor externo)
- Interface moderna e intuitiva
- Validações robustas de conflitos
- Código aberto e customizável

Limitações Identificadas:

Apesar do sucesso geral, algumas limitações foram identificadas:
- Ausência de sistema de autenticação e autorização
- Notificações por e-mail não implementadas (apenas simuladas)
- Relatórios avançados em PDF não incluídos
- Falta de versionamento de aplicação móvel

Estas limitações foram mapeadas como possíveis melhorias futuras mas não comprometem o funcionamento principal do sistema.

6 CONSIDERAÇÕES FINAIS

O sistema de gestão de aulas criado para a Prefeitura de Véridia atingiu cada um dos objetivos traçados, gerando uma solução sólida, funcional e pronta para ser usada no dia a dia. O projeto mostrou como os conceitos da matéria de Projeto de Programação podem ser aplicados na prática, além de trazer experiência valiosa com tecnologias importantes no mercado de trabalho.

Principais Resultados:

1. Sistema finalizado com todas as funções pedidas nos requisitos da Unidade II;

2. Código organizado e fácil de entender, seguindo as melhores práticas de programação e modelos de projeto já conhecidos;

3. Visual moderno e fácil de usar, que proporciona uma ótima experiência para quem utiliza o sistema;

4. Validações fortes que garantem que os dados estejam corretos e evitam erros de uso;

5. Documentação completa, incluindo código comentado, um arquivo README detalhado e este documento técnico.

O que a Equipe Aprendeu:

O projeto ensinou muito à equipe em várias áreas:
- Trabalho em conjunto usando Git e GitHub para controlar as versões do código;
- Uso prático de modelos de projeto (Repository, Service, MVC);
- Criação de APIs RESTful seguindo as regras do protocolo HTTP;
- Salvar dados usando JPA e H2.;
- Criação de interfaces visuais com JavaFX;
- Encontrar e corrigir problemas difíceis em situações reais;
- Organizar o tempo e cumprir os prazos estabelecidos.

A experiência de criar um projeto completo em equipe mostrou a importância de falar claramente, planejar bem e dividir as tarefas de forma justa. Os desafios enfrentados, principalmente os problemas técnicos com herança JPA, mostraram que é preciso entender profundamente as tecnologias usadas e não desistir de encontrar soluções.

A escolha de tecnologias modernas e muito usadas no mercado (Spring Boot, JavaFX, JPA e H2) fez com que o projeto fosse relevante e valioso para o aprendizado da equipe, preparando os desenvolvedores para os desafios reais da programação profissional.

Conclusão:

O sistema de gestão de aulas não é só o cumprimento dos requisitos da matéria, mas também uma solução real que pode ser usada pela Prefeitura de Véridia ou outras instituições parecidas. O projeto mostra que a equipe consegue analisar problemas, criar soluções adequadas e implementar sistemas completos com tecnologias modernas.

Os resultados mostram que os objetivos de aprendizado foram alcançados completamente, com a equipe demonstrando conhecimento em programação orientada a objetos, frameworks modernos, salvar dados e criar interfaces visuais. O projeto é, portanto, uma base forte para o futuro profissional de todos os membros da equipe.

7 REFERÊNCIAS

CLARO, Daniela Barreiro; SOBRAL, João Bosco Mangueira. Programação em JAVA. **Livro programando em Java 1ª edição**, p. 12, 2008

ARAÚJO, Gabriel Fontana Junqueira et al. Desenvolvimento de um sistema de academia em JavaFX. 2022.

MARQUES, Gleice Ferreira; CARDOSO, Rafael. A importância da segurança em banco de dados. **Revista Eletrônica da Faculdade Invest de Ciências e Tecnologia**, v. 5, n. 1, p. 13-13, 2021.

DE ARRUDA, Antonio AG et al. COMPARAÇÃO DE DESEMPENHO DE OPERAÇÕES CRUD EM DIFERENTES MODELOS DE BANCOS DE DADOS: RELACIONAL, ORIENTADO A OBJETOS E OBJETO-RELACIONAL

BOAGLIO, Fernando. **Spring Boot: Acelere o desenvolvimento de microsserviços**. Casa do Código, 2017..

8 ANEXOS

Anexos - Projeto de Programação

https://github.com/RaulRSouza/Projeto-de-Programa-o_N02_Grupo3.git

Figura 1 - Diagrama de Casos de Uso

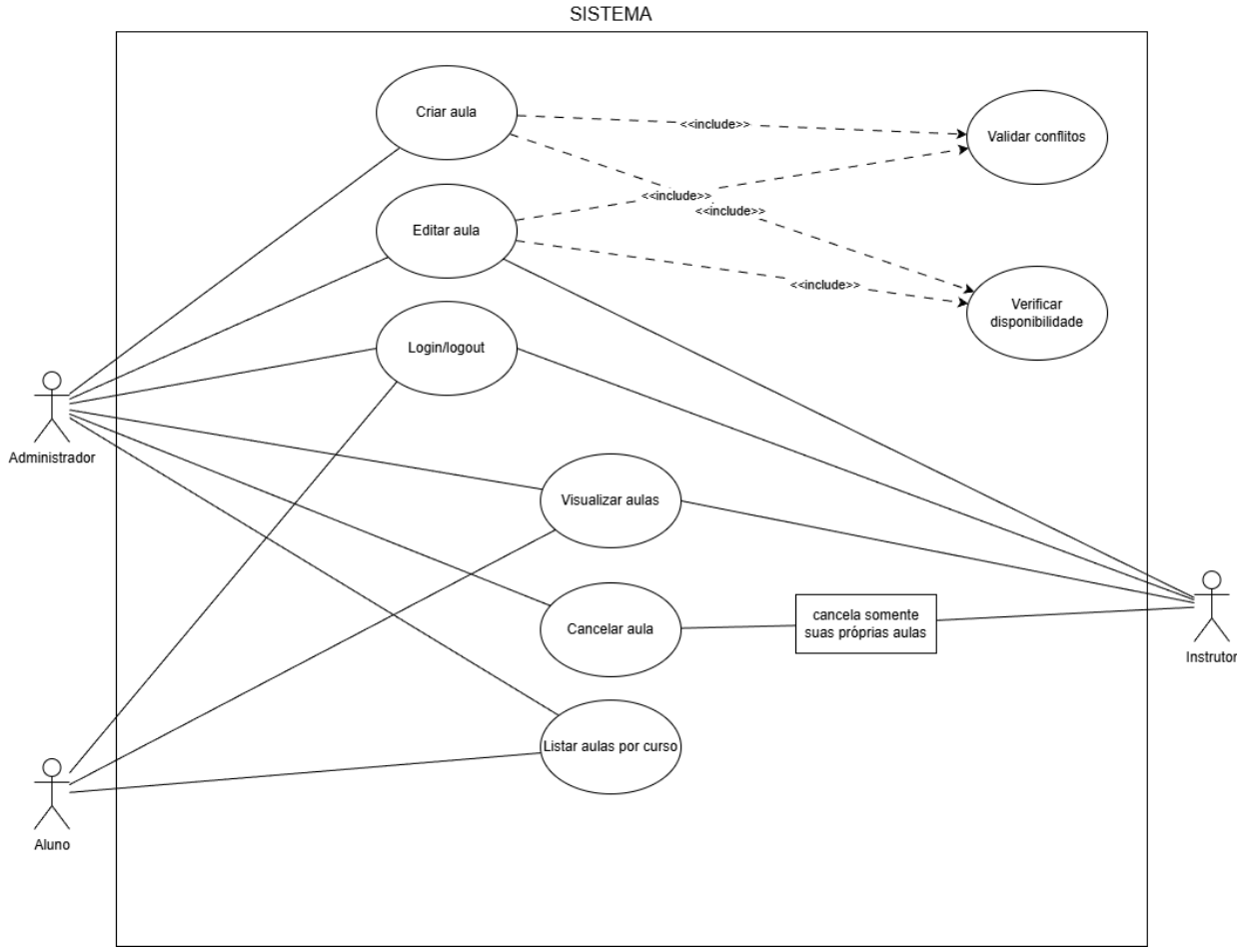

Figura 2 - Diagrama de Classes

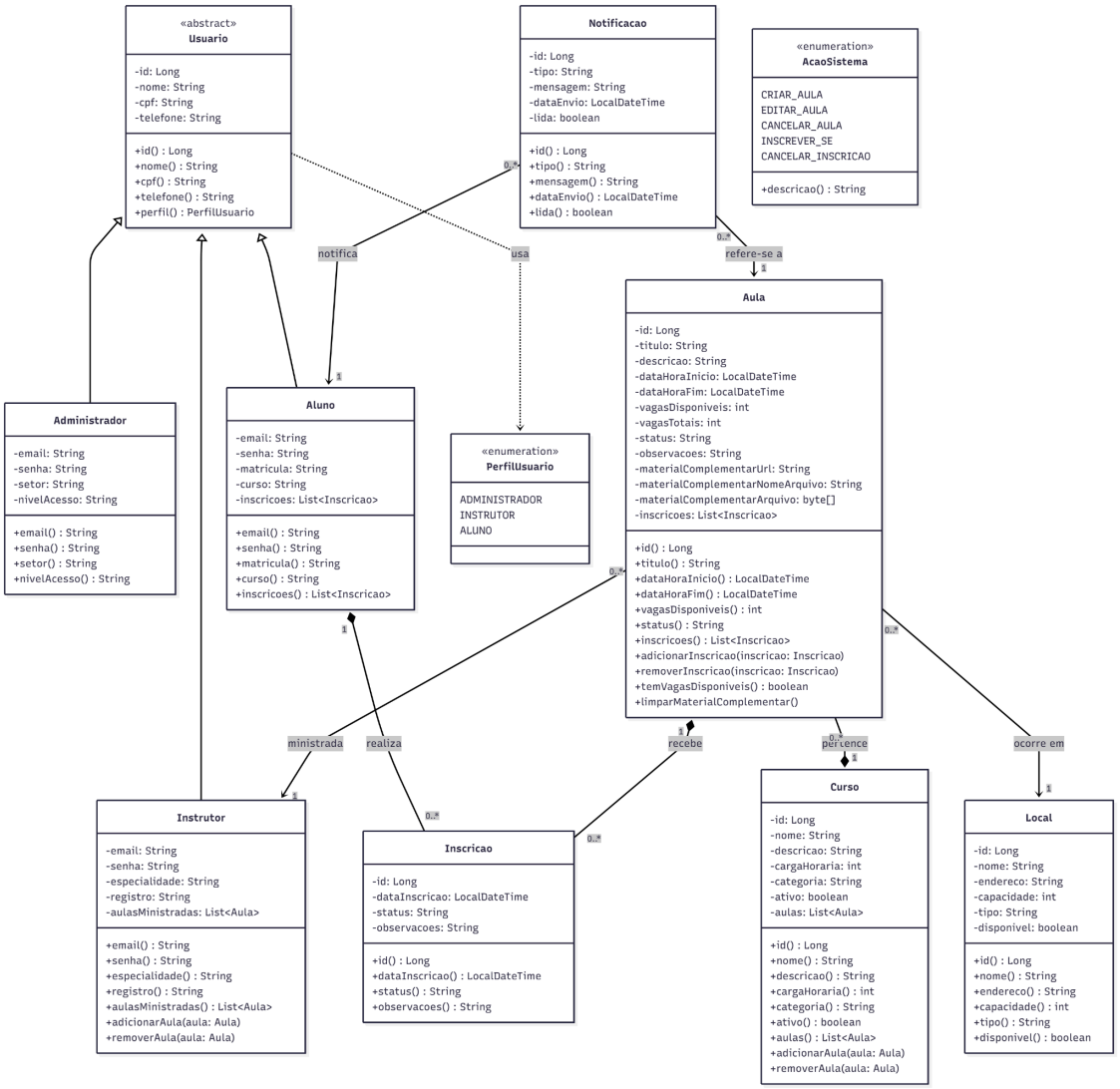

Figura 3 - Diagrama de Entidade Relacionamento