Projeto Integrador 2º Semestre - DSM

**Disciplinas:**

Banco de Dados

Desenvolvimento Web II

Engenharia de Software II

**Professores:**

Bruno

Nilton

Orlando

**Grupo 4 / Nome da Empresa: Vision**

**Sistema: ERP (Planejamento de recursos empresariais**)

|  |  |
| --- | --- |
| Integrante | **Papel Principal** |
| GUSTAVO CARVALHO | BackEnd |
| JULIANO SANTOS | Product Owner e BackEnd |
| KAUÊ RASHINI | BackEnd |
| LUIZ HENRIQUE | Documentação e Diagramas |
| MELISSA BEATRIZ | FrontEnd |

**Fatec Araras**

**2024**

FICHA DE CONTROLE - PROJETO INTERDISCIPLINAR

DISCIPLINA CHAVE: Engenharia de Software II - PI II

PROFESSOR: Bruno Henrique de Paula Ferreira

GRUPO: 04 SEMESTRE: 2/2024

TÍTULO DO PROJETO: Título do projeto

DATA DA APRESENTAÇÃO: 03/12/2024

NOTA:

INTEGRANTES DO GRUPO: Nome grupo

|  |  |
| --- | --- |
| Nome | Nota Individual |
| GUSTAVO CARVALHO DA SILVA |  |
| JULIANO APARECIDO DOS SANTOS |  |
| KAUÊ RASHINI DE SOUZA GOMES |  |
| LUIZ HENRIQUE DE LUCCAS |  |
| MELISSA BEATRIZ VIEIRA |  |

Araras, 03 de dezembro de 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor Bruno Henrique de Paula Ferreira

**Sumário**

[1. Apresentação da Empresa 4](#_Toc176466743)

[1.1 Missão 4](#_Toc176466744)

[1.4 Link Repositório 4](#_Toc176466745)

[2. Escopo do sistema 5](#_Toc176466746)

[2.1 Objetivos do projeto 5](#_Toc176466747)

[2.2 Técnica de levantamento de requisitos 5](#_Toc176466748)

[2.3 Requisitos funcionais 5](#_Toc176466749)

[2.4 Requisitos não funcionais 5](#_Toc176466750)

[2.5 Comparativo entre sistemas 6](#_Toc176466751)

[Descrever resumidamente e apresentar uma tabela de comparação do sistema desenvolvido versus 1 ou 2 outros sistemas de mercado. 6](#_Toc176466752)

[2.6 Cronograma 6](#_Toc176466753)

[3. Documentação do Sistema 7](#_Toc176466754)

[3.1 Metodologia de Desenvolvimento 7](#_Toc176466755)

[3.2 Diagramas UML 7](#_Toc176466756)

[Diagrama de caso de uso (1º) 7](#_Toc176466757)

[Diagrama de classes (2º) 7](#_Toc176466758)

[Diagrama de sequência (2º) 7](#_Toc176466759)

[3.3 Modelo Conceitual (1º) 8](#_Toc176466760)

[3.4 Modelo lógico (1º) 8](#_Toc176466761)

[3.5 Modelo Físico (2º) 8](#_Toc176466762)

[3.6 Recursos e ferramentas (1º e 2º) 8](#_Toc176466763)

[3.7 Etapas / Sprints realizados (1º e 2º) 8](#_Toc176466764)

[3.8 Interface do usuário 8](#_Toc176466765)

[4. Testes e Qualidade 9](#_Toc176466766)

[4.1 Estratégia de Testes: Descrever a estratégia de testes adotada 9](#_Toc176466767)

[4.2 Resultados dos Testes: Apresentar os resultados dos testes realizados 9](#_Toc176466768)

[4.3 Garantia da Qualidade: Descrever as práticas adotadas para garantia da qualidade 9](#_Toc176466769)

[4.4 Requisitos mínimos de hardware e software para o sistema 9](#_Toc176466770)

[4.5 Contrato para desenvolvimento de software 9](#_Toc176466771)

[5. Considerações Finais 9](#_Toc176466772)

[6 Referências 9](#_Toc176466773)

[Anexo I - Diário de bordo 9](#_Toc176466774)

[Anexo II – Cronograma efetivo 9](#_Toc176466775)

[Anexo III – Evidências 9](#_Toc176466776)

# Apresentação da Empresa

A Codares é uma empresa com soluções objetivas, com o intuito de atender empresas de pequeno porte, oferecendo soluções minimalistas e eficientes. Seu foco é simplificar e integrar processos essenciais para o funcionamento de diversos tipos de sistemas.





## 1.1 Missão

Proporcionar sistemas acessíveis e eficientes, capacitando pequenos empreendedores a gerenciar seus negócios de forma prática e inteligente.

1.2 Visão

Expandir a oferta de soluções inovadoras que atendam às necessidades de empresas de diferentes setores, promovendo sua evolução e crescimento.

1.3 Valores

Nosso compromisso é entregar soluções de alto valor, garantindo a satisfação dos clientes por meio de sistemas eficientes, práticos e ajustáveis a orçamentos diversos.

## 1.4 Link Repositório

<https://github.com/PI-2-semestre-DSM/titanium-acad-app>

# Escopo do sistema

A empresa Codares tem como objetivo realizar a gestão da academia ‘Titanium’, onde será implementado o sistema web que permite o acompanhamento da frequência dos alunos, gestão de produtos e pagamentos.

O administrador será o proprietário da academia, onde será responsável pelas permissões do sistema de seus funcionários. Os alunos devem fazer o login através de seu CPF, para que haja um controle de sua frequência e pagamento, sendo possível também o acompanhamento de pagamentos e treinos realizados.

A aplicação é acessada exclusivamente pela web (sendo necessário o uso de internet), será de fácil acesso e explicada caso necessário pelos funcionários.

## Objetivos do projeto

Login dos clientes e acompanhamento de frequência;

Inventário de estoque;

Sistema prático e intuitivo.

## 2.2 Técnica de levantamento de requisitos

* Entrevista com o proprietário do estabelecimento profissional
* Questionários

## Requisitos funcionais

|  |  |
| --- | --- |
| **Ref.** | RF01 |
| **Requisito** | Matrícula de usuário |
| **Ator** | Administrador |
| **Prioridade** | Essencial |
| **Pré-condições** | Acesso com internet |
| **Requisitos não funcionais associados** |  |
| **Descrição** | O sistema deve cadastrar um novo usuário ao fazer uma nova matrícula. |
| **Detalhes do requisito** | O Administrador criando a matrícula automaticamente gerará um cadastro no sistema |

|  |  |
| --- | --- |
| **Ref.** | RF02 |
| **Requisito** | Login de Usuário |
| **Ator** | Usuário |
| **Prioridade** | Essencial |
| **Pré-condições** | Acesso com internet |
| **Requisitos não funcionais associados** |  |
| **Descrição** | O sistema deve ter uma tela com campos de usuário de acesso e senha. |
| **Detalhes do requisito** | O usuário após preencher os campos de CPF e senha corretamente terá acesso aos treinos, pagamentos e frequência. |

|  |  |
| --- | --- |
| **Ref.** | RF03 |
| **Requisito** | Controle Financeiro |
| **Ator** | Administrador |
| **Prioridade** | Importante |
| **Pré-condições** | Acesso com internet |
| **Requisitos não funcionais associados** |  |
| **Descrição** | O sistema deve ter controle sobre os pagamentos |
| **Detalhes do requisito** | O Administrador solicita ao sistema |

## 2.4 Requisitos não funcionais

|  |  |
| --- | --- |
| **Ref.** | RNF01 |
| **Requisito** | Tempo de Resposta |
| **Ator** | Usuário |
| **Prioridade** | Essencial |
| **Pré-condições** | Acesso a uma conexão estável de internet |
| **Requisitos não funcionais associados** |  |
| **Descrição** | O sistema deve responder às solicitações dos usuários em até 1 segundo. |
| **Detalhes do requisito** | O tempo de resposta deve ser medido em cenários normais de uso e com volume médio de acessos simultâneos. |

|  |  |
| --- | --- |
| **Ref.** | RNF02 |
| **Requisito** | Conexão com a Internet |
| **Ator** | Sistema |
| **Prioridade** | Essencial |
| **Pré-condições** | Acesso à internet estável |
| **Requisitos não funcionais associados** |  |
| **Descrição** | O sistema deve garantir que todos os processos de comunicação com o banco de dados funcionem corretamente, desde que a conexão com a internet esteja ativa e estável. |
| **Detalhes do requisito** | O sistema depende de uma conexão estável com a internet para realizar operações de leitura e escrita no banco de dados. Caso a conexão seja interrompida, o sistema deve exibir uma mensagem de erro ao usuário, informando sobre a falha na comunicação. |

|  |  |
| --- | --- |
| **Ref.** | RNF03 |
| **Requisito** | Compatibilidade com PHP |
| **Ator** | Sistema |
| **Prioridade** | Essencial |
| **Pré-condições** | Versão de PHP maior ou igual a 8.2 |
| **Requisitos não funcionais associados** |  |
| **Descrição** | O sistema deve ser compatível com a versão 8.2 ou superior do PHP. |
| **Detalhes do requisito** | O sistema deve ser desenvolvido e testado para funcionar corretamente nas versões 8.2 ou superiores do PHP. |

## 2.5 Comparativo entre sistemas

## Descrever resumidamente e apresentar uma tabela de comparação do sistema desenvolvido versus 1 ou 2 outros sistemas de mercado.

**Comparativo das funcionalidades da aplicação**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funcionalidades | Excel | | Zorro CRM | Trello | TitaniumApp |
| Gerenciar vendas | | **X** | **X** | **-** | **X** |
| Gerenciar produtos | | **X** | **X** | **-** | **X** |
| Relatórios automáticos de vendas | | **-** | **X** | **-** | **X** |
| Controle de estoque | | **-** | **-** | **-** | **X** |
| Autenticação de usuário | | **-** | **X** | **X** | **X** |
| Ser multiplataforma | | **X** | **X** | **X** | **-** |

## 2.6 Cronograma

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Tarefas** | **22/08** | **29/08 a 05/09** | **08/10 a 18/10** | **01/10 a**  **22/10** | **29/10 a 12/11** | **13/11 a 22/11** | **03/12** |
| Definição Grupos | X |  |  |  |  |  |  |
| Criação Empresa |  | X |  |  |  |  |  |
| Escopo Sistema |  |  | X |  |  |  |  |
| Requisitos |  |  | X |  |  |  |  |
| Diagramas |  |  | X | X | X |  |  |
| Protótipo |  |  |  |  |  |  |  |
| Documentação |  |  | X | X | X | X |  |
| Entrega |  |  |  |  |  |  |  |
| Apresentação |  |  |  |  |  |  | X |

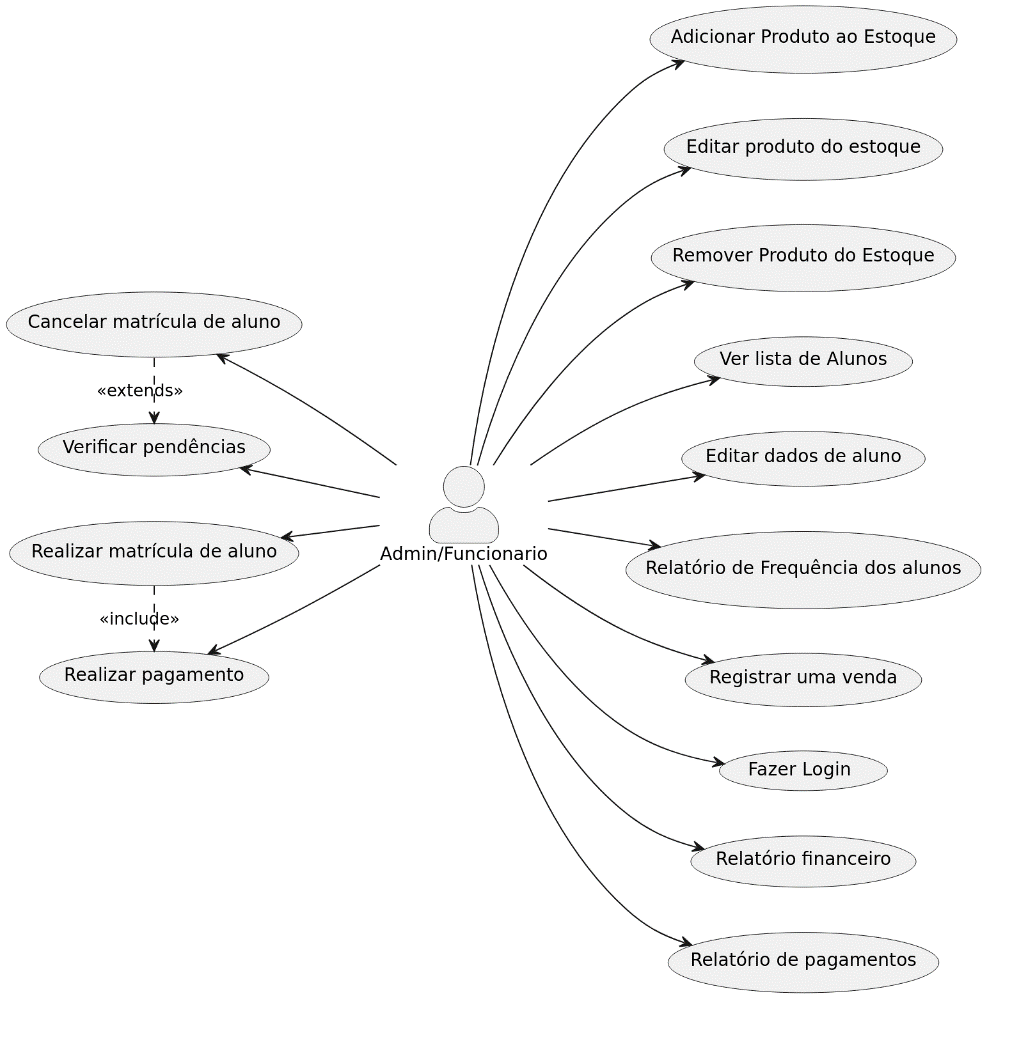
# 

# Documentação do Sistema

## Metodologia de Desenvolvimento

## 3.2 Diagramas UML

## Diagrama de caso de uso



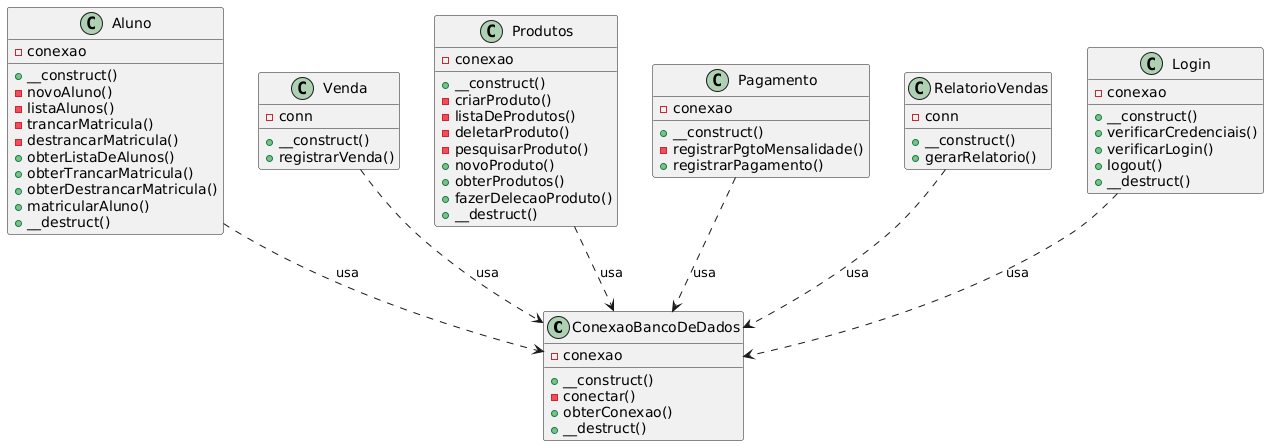
**Explicações:**

O aplicativo TitaniumApp tem a finalidade de ser utilizado pelo gestor da academia, que terá acesso a todas as funcionalidades, permitindo gerenciar alunos, estoque, vendas e pagamentos.

A funcionalidade “Cancelar matrícula de aluno” pode verificar se há pendências financeiras do aluno que solicita o cancelamento, mas o sistema também permite verificar pendências de maneira isolada.

No entanto, para realizar a matrícula do aluno é obrigatório o pagamento da mensalidade

## Diagrama de classes



**Explicações:**

O diagrama apresentado se refere a aplicação das classes que compõem o TitaniumApp, cada uma com responsabilidades bem definidas.

**Classe Alunos:**

**Métodos:**

**\_\_construct()**:  
  
O método **construct** é chamado automaticamente quando uma nova instância da classe é criada.  
 Ele inicializa a conexão com o banco de dados, criando uma instância da classe ConexaoBancoDeDados e atribuindo a conexão ao atributo $conexao.

**listarAlunos()**  
  
Este método tem a função de listar todos os alunos cadastrados no sistema.  
 Ele executa uma consulta SQL para buscar todos os registros de alunos no banco de dados. O resultado da consulta é retornado como um array associativo, permitindo que outras partes do sistema acessem os dados dos alunos para exibição ou processamento adicional.

**novoAluno()**

Este método privado insere um novo registro de aluno no banco de dados. Ele recebe as informações do aluno como parâmetros e de adiciona automaticamente a data de matrícula.

**trancarMatricula ()**

Este método privado altera o status de um aluno para "inativo" no banco de dados, baseado no ID fornecido como parâmetro. Ele utiliza uma instrução SQL para atualizar o campo ativo do registro do aluno.

**destrancarMatricula ()**

Este método privado altera o status de um aluno para "inativo" no banco de dados, baseado no ID fornecido como parâmetro. Ele utiliza uma instrução SQL para atualizar o campo ativo do registro do aluno.

**Classe Venda:**

**Métodos:**

**\_\_construct()**:

Este método é automaticamente chamado quando um objeto da classe Venda é criado. Ele recebe como parâmetro a conexão com o banco de dados (geralmente obtida em outra parte do sistema) e a armazena no atributo $conn

**registrarVenda()**

O método cria uma instrução SQL que insere os dados da venda na tabela produtosvendas, associando o ID do produto e a quantidade à nova venda.

Caso a operação de inserção seja bem-sucedida, o método retorna true. Se houver um erro, ele captura a exceção e retorna false. Uma mensagem de erro também é exibida, que pode indicar um erro.

**Classe Produtos:**

**Métodos:**

**\_\_construct()**:

Este método é chamado automaticamente quando um objeto da classe Produtos é criado. Ele cria uma instância da classe ConexaoBancoDeDados e armazena a conexão com o banco de dados no atributo $conexao.

**criarProduto()**:

Antes de adicionar o novo produto, o método verifica se o produto já existe no banco de dados, chamando o método pesquisarProduto($titulo). Se o produto já estiver registrado, o processo é interrompido e uma mensagem de erro é exibida. Caso contrário, ele executa uma consulta SQL que insere o novo produto na tabela produtos.

**pesquisarProduto()**:

Este é um método privado utilizado dentro de outros métodos para verificar se um produto com o título fornecido já existe no banco de dados. Ele executa uma consulta SQL que verifica se o produto já está registrado na tabela produtos.

**deletarProduto()**:

Este método privado é responsável por deletar um produto específico do banco de dados. Ele recebe como parâmetro o **$idProduto**, que é o identificador único do produto a ser deletado. O método executa uma consulta SQL para excluir o produto com o ID correspondente na tabela produtos.

**Classe Pagamentos:**

**Métodos:**

**\_\_construct()**:

Este método é chamado automaticamente quando um objeto da classe Pagamento é criado. Ele estabelece a conexão com o banco de dados utilizando a classe ConexaoBancoDeDados.

**registrarPgtoMensalidade ()**:

Este método é responsável por registrar o pagamento de uma mensalidade para um aluno. Ele cria uma consulta SQL que insere esses dados na tabela pagamentosMensalidade no banco de dados. Isso registra oficialmente o pagamento feito por um aluno.

**registrarPagamento ()**:

Este é um método público que serve como uma interface para registrar um pagamento de mensalidade. Em vez de interagir diretamente com o método privado registrarPgtoMensalidade, ele apenas passa os parâmetros recebidos para esse método privado.

**Classe RelatorioVendas:**

**Métodos:**

**\_\_construct()**:

Este método é chamado automaticamente quando um objeto da classe RelatorioVendas é instanciado. Ele recebe uma conexão com o banco de dados ($dbConnection) como parâmetro e a armazena no atributo $conn.

**gerarRelatorio ()**:

Este método chama uma stored procedure no banco de dados para gerar um relatório de vendas em um intervalo de datas fornecido.

**Classe Login:**

**Métodos:**

**\_\_construct()**:

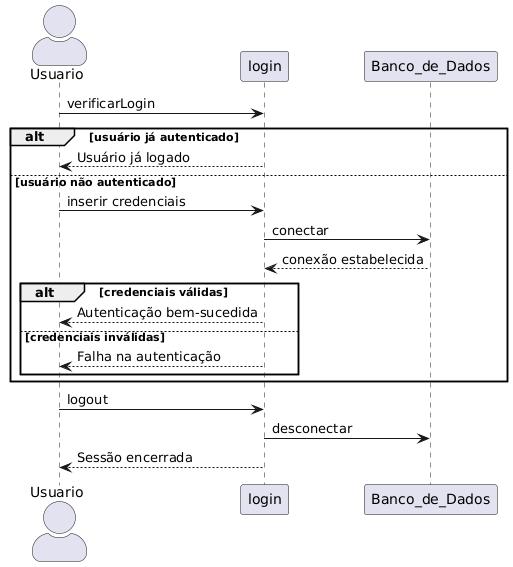
Este método é automaticamente chamado ao criar um objeto da classe. Ele inicializa a conexão com o banco de dados utilizando a classe ConexaoBancoDeDados.

**verificarCredenciais ()**:

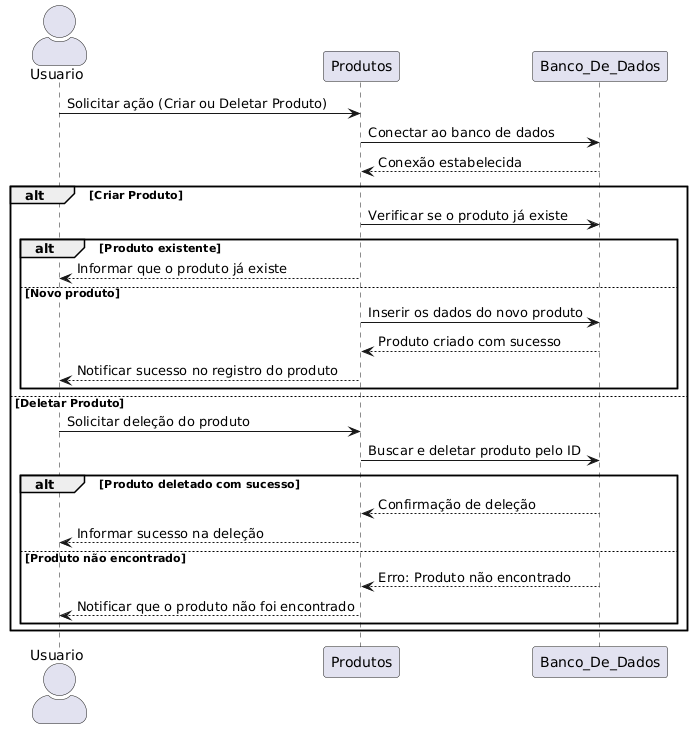
tem a função central da classe: autenticar o usuário. Ele recebe dois parâmetros, o nome de usuário e a senha, e verifica no banco de dados se há um registro correspondente.Diagramas de sequência

## Diagramas de sequência

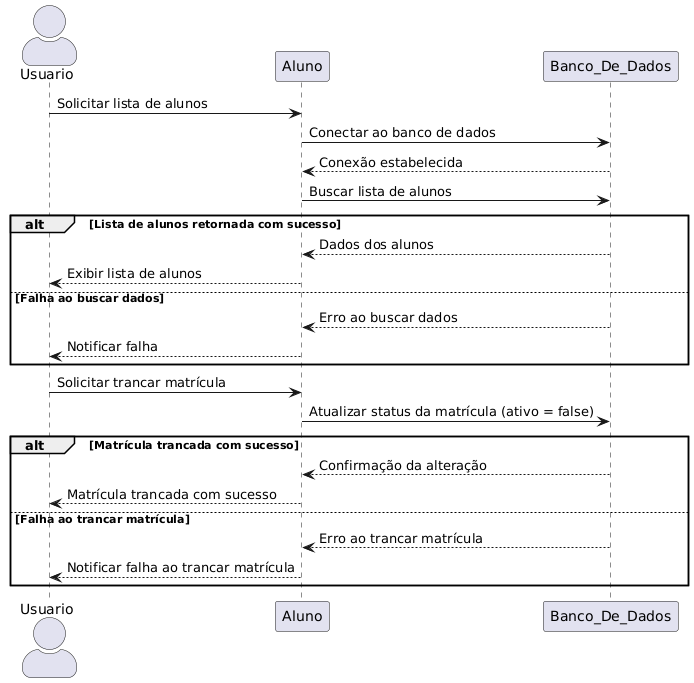
**Classe Login: Responsável pelo login do usuário**



**Classe Produtos: Gerencia cadastro, consulta e remoção de produtos**

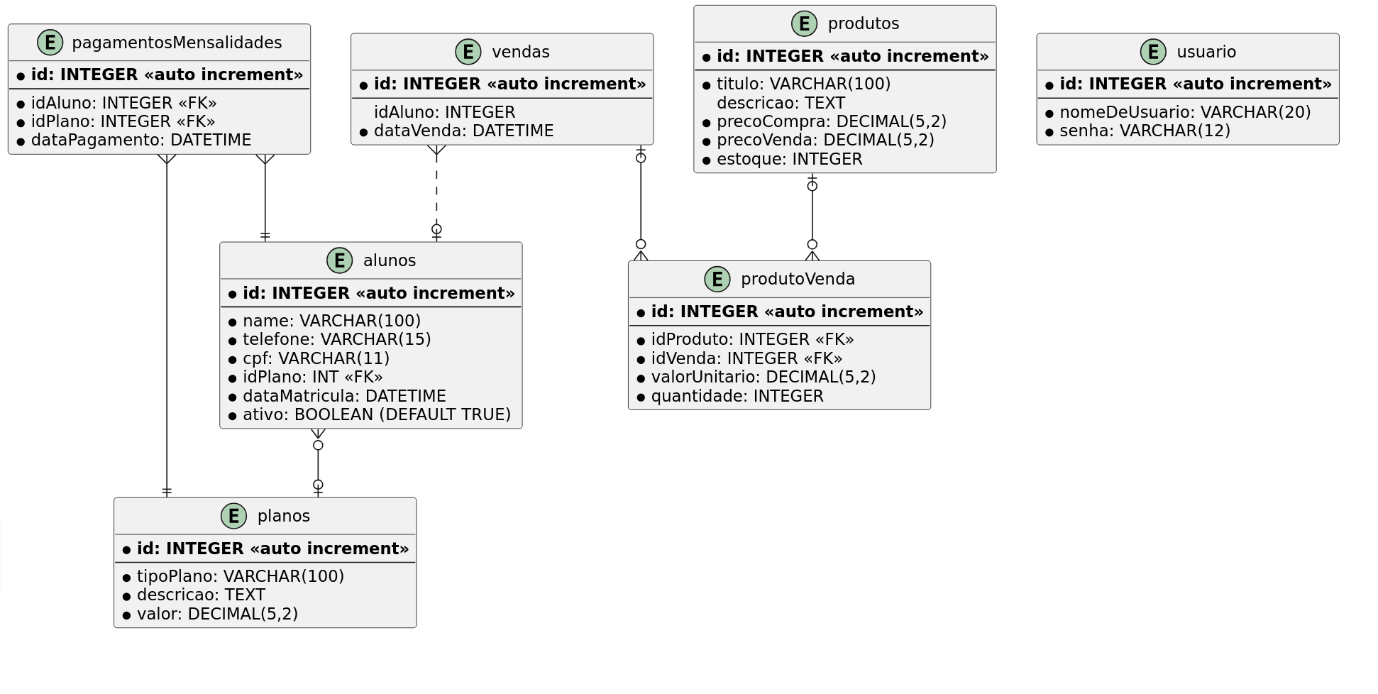
**

**Classe Alunos: Lista, consulta e tranca matrícula de alunos**



## Modelo Conceitual (1º)

## Modelo lógico (1º)



## Modelo Físico (2º)

CREATE TABLE `pagamentosMensalidades` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`idAluno` int(11) NOT NULL,

`idPlano` int(11) NOT NULL,

`dataPagamento` datetime DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `idAluno` (`idAluno`),

KEY `idPlano` (`idPlano`),

CONSTRAINT `pagamentosMensalidades\_ibfk\_1` FOREIGN KEY (`idAluno`) REFERENCES `alunos` (`id`),

CONSTRAINT `pagamentosMensalidades\_ibfk\_2` FOREIGN KEY (`idPlano`) REFERENCES `planos` (`id`)

);

CREATE TABLE `planos` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`tipoPlano` varchar(10) DEFAULT NULL,

`valor` decimal(5,2) DEFAULT NULL,

PRIMARY KEY (`id`)

)

CREATE TABLE `produtos` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`titulo` varchar(100) DEFAULT NULL,

`descricao` text DEFAULT NULL,

`precoCompra` decimal(5,2) DEFAULT NULL,

`precoVenda` decimal(5,2) DEFAULT NULL,

`estoque` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

)

CREATE TABLE `produtosVendas` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`idProduto` int(11) NOT NULL,

`idVenda` int(11) NOT NULL,

`valorUnitario` decimal(5,2) DEFAULT NULL,

`quantidade` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `idProduto` (`idProduto`),

KEY `idVenda` (`idVenda`),

FOREIGN KEY (`idProduto`) REFERENCES `produtos` (`id`),

FOREIGN KEY (`idVenda`) REFERENCES `vendas` (`id`)

)

CREATE TABLE `usuarios` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`nomeDeUsuario` varchar(20) NOT NULL,

`senha` varchar(12) NOT NULL,

PRIMARY KEY (`id`)

)

CREATE TABLE `vendas` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`idAluno` int(11) NOT NULL,

`dataVenda` datetime DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `idAluno` (`idAluno`),

FOREIGN KEY (`idAluno`) REFERENCES `alunos` (`id`)

)

## Recursos e ferramentas (1º e 2º)

## 3.7 Etapas / Sprints realizados (1º e 2º)

## 3.8 Interface do usuário

# Considerações Finais

Conclusão: Concluir o trabalho e destacar aprendizados

Contribuições Individuais: Descrever as contribuições individuais de cada membro da equipe

# 6 Referências

# Anexo I - Diário de bordo

**Físico:** Feito pelo grupo.

**Digital:** individual / Links

# Anexo II – Cronograma efetivo

# Anexo III – Evidências

Link Live, participantes externos, prints.