Esta seção da documentação refere-se à implementação da camada de **backend**, desenvolvida em **C#** com a **plataforma .NET versão 9.0**, utilizando os princípios da **programação orientada a objetos (POO)**. A arquitetura segue o padrão **MVC (Model-View-Controller)**, com foco em separação de responsabilidades, manutenção e extensibilidade do código.

O projeto utiliza recursos nativos do .NET, como **Entity Framework Core** para persistência de dados, **Dependency Injection** para gerenciamento de dependências e **middlewares** para controle de autenticação e autorização. Essa abordagem permite um desenvolvimento padronizado, testável e alinhado às boas práticas recomendadas pela Microsoft.

A estrutura de usuários do sistema foi modelada com base em **herança e polimorfismo**, utilizando uma classe base (Usuarios) para centralizar atributos e comportamentos comuns, incluindo permissões. Classes derivadas como Adm, Editor e Docente sobrescrevem métodos virtuais para definir regras específicas de acesso, promovendo reutilização e clareza.

**Classe base: Usuario**

A classe Usuario é a classe genérica que representa um usuário do sistema, contendo os atributos básicos e as permissões padrão.

**Atributos principais:**

* Nome (string): Nome completo do usuário.
* Matricula (int): Identificador único para autenticação.
* Cargo (string): Cargo ou função do usuário.
* Senha (string): Senha do usuário, armazenada com controle de acesso (private set).

**Permissões (métodos virtuais):**

São métodos booleanos que retornam false por padrão e devem ser sobrescritos nas classes filhas para definir o que cada tipo de usuário pode fazer:

public virtual bool PodeCadastrarDocente() => false;

public virtual bool PodeRemoverDocente() => false;

public virtual bool PodeCadastrarEditor() => false;

public virtual bool PodeRemoverEditor() => false;

public virtual bool PodeCadastrarAdm() => false;

public virtual bool PodeRemoverAdm() => false;

public virtual bool PodeCadastrarUsuario() => false;

public virtual bool PodeRemoverUsuario() => false;

public virtual bool PodeEditarAssociacao() => false;

**Classes filhas e regras de permissão**

* **Administrador**: Pode cadastrar e remover administradores e editores. Para cadastrar/remover administradores, é necessária confirmação de credenciais (matrícula e senha).
* **Editor**: Pode cadastrar e remover docentes, mas sem necessidade de confirmação de credenciais.
* **Docente**: Não possui permissões administrativas.

**Gerenciamento e controle**

Foi criada a classe UsuariosManager que centraliza todas as operações relacionadas a usuários, como cadastro, remoção e edição. Essas ações são acionadas a partir de um sistema de menu genérico, com submenus específicos para cada tipo de ação, garantindo uma interface organizada e modular para interação com o backend.

### ****Login e Autenticação de Acesso****

O sistema conta com um módulo de login que realiza a autenticação de usuários com base nas credenciais fornecidas (matrícula e senha). O processo de autenticação é responsável por validar a identidade do usuário e garantir que apenas usuários autorizados possam acessar o sistema e executar ações conforme suas permissões. A senha é armazenada de forma segura utilizando mecanismos de criptografia e nunca é exposta em texto plano. Após a autenticação bem-sucedida, o sistema identifica o tipo de usuário (Administrador, Editor ou Docente) e carrega dinamicamente os menus e permissões correspondentes, conforme a estrutura hierárquica implementada nas classes derivadas da classe Usuario.  
Esse processo de login é acionado no início da aplicação, funcionando como um ponto de entrada obrigatório para qualquer interação com o backend.

**Módulo de Estrutura Acadêmica**

Além do controle de usuários, o sistema também contempla o gerenciamento de **salas, turmas e cursos**, cada um representado por uma classe própria com seus atributos e comportamentos. As ações principais (criação, edição e remoção) são centralizadas na classe AcademicoManager, que atua como ponto de entrada para manipulação dessas entidades.

### Classes Individuais

#### **Sala**

Representa uma sala física ou virtual onde ocorrem atividades acadêmicas.

**Atributos comuns:**

* chave (int): Identificação da sala.
* status (bool): Sala livre (true) ou ocupada (false).
* Laboratorio (string): Ex: "laboratório de TI I", "Lab. Redes II", "Auditório".
* IdDocente (int): Pega o id do docente responsável pela sala, se houver.
* IdCurso (int): Pega o id do curso ministrado na sala, se houver.
* IdTurma (int): Pega o id da turma ocupante da sala, se houver.

**Métodos:**

* Alterar Docente
* Alterar Curso
* Alterar Laboratorio
* Alterar Turma
* Alterar Status

#### **Turma**

Representa uma turma de alunos associada a um curso e, potencialmente, a uma sala.

**Atributos comuns:**

* IdTurma (int): Id único da turma.
* NumeroTurma (int): Ex: "2024.1"
* IdCurso (referência a Curso)
* IdSala (referência a Sala)
* IdDocente (referência ao docente):

#### **Curso**

Define os dados de um curso oferecido pela instituição.

**Atributos comuns:**

* IdCurso (int): Id único da turma.
* NomeCurso (string): Ex: “Téc. Desenvolvimento de sistemas”
* IdCurso (referência a Curso)
* IdSala (referência a Sala)
* IdDocente (referência ao docente):

### Classe AcademicoManager

A classe SalaManager é responsável por todas as ações de gerenciamento entre Sala, Turma e Curso, centralizando a lógica de negócio relacionada à estrutura acadêmica. Isso inclui:

**Funções principais:**

* Cadastrar, editar e remover salas
* Criar e remover turmas
* Editar associações das salas (docente, curso, turma, laboratório.)
* Buscar sala por chave

**Observações:**

* As ações seguem o padrão orientado a objetos, mantendo encapsulamento e separação de responsabilidades.
* A lógica de verificação (como se a sala está disponível ou se uma turma pode ser alocada) está contida nos métodos da própria SalaManager, mantendo as entidades Sala, Turma e Curso com baixa acoplagem.