📘 Documentação - Projeto EduConnect

Autor: Carlos Henrique Nunes

1. VISÃO GERAL DO SISTEMA

1.1 Nome do Sistema

EduConnect — Plataforma Inteligente de Gestão Escolar

1.2 Objetivo Geral da Plataforma

O **EduConnect** é uma plataforma integrada de gestão escolar desenvolvida para facilitar e otimizar a rotina acadêmica de **alunos, professores, responsáveis e administradores**.
O sistema conecta todos os perfis em um ecossistema digital único, garantindo organização, eficiência, comunicação transparente e acompanhamento contínuo do desempenho escolar.

A plataforma foi projetada para ser:

- **Escalável**
- **Responsiva**
- **Segura**
- **Intuitiva**
- **Fácil de operar**

Atendendo tanto instituições de pequeno porte quanto redes educacionais maiores.

1.3 Desafios que a Plataforma Resolve

O EduConnect resolve problemas frequentes do ambiente escolar:

- Falta de comunicação entre escola, alunos e responsáveis

- Dificuldade no acompanhamento de desempenho
- Falta de centralização de informações
- Desorganização de atividades, provas e cronogramas
- Ausência de uma ferramenta simples para gestão de turmas e matrículas
- Pouca visibilidade de métricas educacionais para decisões estratégicas

---

1.4 Público-Alvo

👨‍🏫 Professores

- Criar e gerenciar atividades
- Registrar desempenho
- Acompanhar alunos
- Planejar provas e trabalhos
- Comunicar-se facilmente com turmas

🧑‍🎓 Alunos

- Acessar conteúdos, atividades, calendário e notas
- Visualizar sua evolução
- Receber alertas e notificações educativas
- Acompanhar trilha de aprendizagem

👨‍👩‍👦 Responsáveis

- Visualizar notas e desempenho
- Acompanhar faltas

- Ser notificado de eventos, reuniões e provas
- Ver progresso dos filhos em tempo real

🏫 Administradores

- Gerenciar toda a instituição
- Criar turmas, usuários e permissões
- Observar dashboards de métricas gerais
- Tomar decisões baseadas em dados (BI)

1.5 Principais Funcionalidades da Plataforma (Completa)

Abaixo estão todas as funcionalidades planejadas para a versão final (EduConnect 2.0), divididas por módulo:

📌 Módulo 1 — Acesso e Perfis de Usuário

- Login integrado (simulado no MVP, real na versão final)
- Perfis com permissões específicas
- Fluxo de onboarding
- Tema claro/escuro preservado por usuário

📌 Módulo 2 — Dashboard Inteligente

Cada perfil possui uma visão própria.

Professores:

- Turmas
- Entregas pendentes
- Desempenho médio
- Próximas aulas/eventos

Alunos:

- Atividades da semana
- Progresso da trilha de estudos
- Próximas provas
- Notificações

Administradores:

- Número de matrículas
- Turmas ativas
- Indicadores gerais da instituição

📌 Módulo 3 — Calendário Educacional Avançado

✔ Visual mensal
✔ Marcadores de dias com eventos
✔ Notificações verdes para compromissos
✔ Cards de compromissos abaixo do calendário
✔ Eventos por perfil
✔ Exibição da data selecionada (“Dia X de Mês Y”)

Na versão final:

- CRUD de eventos por perfil
- Eventos recorrentes
- Vinculação com atividades e provas

- Exportação para Google Calendar

📌 Módulo 4 — Gestão Acadêmica

Professores:

- Criar atividades e avaliações
- Gerenciar turmas
- Registrar notas e faltas
- Upload de materiais

Alunos:

- Entregar atividades
- Ver notas
- Ver progresso
- Baixar materiais

Admin:

- Criar e editar turmas
- Gestão de matrículas
- Controle de permissões

📌 Módulo 5 — Comunicação Interna

- Notificações por evento
- Mensagens entre usuários

- Alertas automáticos (performances, faltas, prazos)
- Canal professor ↔ aluno
- Canal professor ↔ responsáveis

📌 Módulo 6 — BI e Indicadores

- Dashboard com gráficos (Chart.js na versão final)
- Média de notas por turma
- Distribuição de desempenho
- Acompanhamento de frequência
- Ranking gamificado (versão final)

📌 Módulo 7 — Trilhas de Aprendizagem (Avançado)

Para diferenciar o projeto:

- Plano de estudo baseado no aluno
- Recomendações automáticas
- Medalhas e conquistas
- Progresso visual de módulos

1.6 MVP Atual (Entregue no Módulo Web)

O MVP contém:

- Login funcional (simulação)

- Dashboard inicial
- Menu lateral responsivo
- Tema claro/escuro
- Gestão inicial de alunos e professores (CRUD com dados simulados)
- Filtros dinâmicos
- Calendário interativo com eventos
- Cards de compromissos
- Notificações (incluindo modo especial para eventos)
- Responsividade completa (mobile-friendly)

Esse MVP já demonstra:

- domínio de HTML, CSS e JS
- controle de estado
- manipulação de DOM
- componentização básica
- UX consistente
- boas práticas de estrutura

1.7 Visão Evolutiva (Roadmap)

Versão 1.0 — MVP (pronto)

Interface + funcionalidades front-end com dados simulados.

Versão 1.5 — Backend Fake

- JSON Server

- Estrutura de rotas REST
- CRUD real via fetch()
- Simulação de banco de dados

Versão 2.0 — Plataforma Completa

- Todos os módulos avançados
- Perfis reais com permissões
- Notificações inteligentes
- Calendário com CRUD
- Fluxos de matrícula
- Trilhas de aprendizagem
- BI avançado
- Gamificação

1.8 Diferenciais Estratégicos do Projeto

- Documentação completa
- Pensamento de produto
- Visão sistêmica
- Funcionalidades avançadas e inovadoras
- Códigos limpos, organizados e padronizados
- Arquitetura escalável

---

2. REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

2.1. Requisitos Funcionais (RF)

A seguir, estão listados os requisitos funcionais do sistema EduConnect.
Cada requisito possui um código identificador (RFxx), uma descrição objetiva e, quando necessário, dependências.

🔐 RF01 — Autenticação de Usuários

O sistema deve permitir que usuários façam login utilizando credenciais válidas (usuário e senha).
Perfis permitidos: Administrador, Professor, Aluno.

👤 RF02 — Perfis de Acesso

O sistema deve oferecer níveis de permissão distintos para cada tipo de usuário:

- **Administrador:** Acesso completo ao sistema; gerenciamento total.
- **Professor:** Acesso às funcionalidades pedagógicas e de turmas.
- **Aluno:** Acesso às funcionalidades acadêmicas, trilhas e calendário pessoal.

👩‍🎓 RF03 — Gestão de Alunos

O sistema deve permitir ao Administrador e ao Professor:

- Cadastrar alunos
- Editar informações
- Listar alunos
- Filtrar alunos

- Excluir alunos (futuro)
- Atribuir alunos a turmas

👩‍🏫 RF04 — Gestão de Professores

Permitir:

- Cadastrar professores
- Editar informações
- Listar professores
- Filtrar professores
- Excluir professores (futuro)
- Associar professor a disciplinas

📚 RF05 — Gestão de Disciplinas

O sistema deve permitir:

- Criar disciplinas
- Listar e editar disciplinas
- Associar disciplinas a turmas e professores

🏫 RF06 — Gestão de Turmas

O sistema deve permitir:

- Criar novas turmas
- Listar turmas e seus respectivos alunos
- Associar professores a turmas

- Gerenciar capacidade das turmas (futuro)

📅 RF07 — Calendário Escolar Interativo

O sistema deve exibir um calendário completo com:

- Navegação entre meses
- Identificação visual de dias com eventos
- Exibição de compromissos por dia
- Criação de eventos (admin/professor – futuro)
- Visualização de eventos do aluno (aluno)

🔔 RF08 — Sistema de Notificações

O sistema deve permitir:

- Notificações de ações internas (sucesso, erro, sistema)
- Notificações de eventos do calendário
- Notificações especiais personalizadas
- Diferenciação por cor conforme o tipo (sucesso, alerta, evento)

📊 RF09 — Relatórios

O sistema deve permitir:

- Visualização de gráficos de desempenho (Chart.js)
- Relatórios por turma/disciplina (futuro)

🧭 RF10 — Dashboard Inteligente

Ao fazer login, o usuário deve visualizar um painel inicial com informações personalizadas:

- Administrador → visão geral do sistema
- Professor → turmas e avisos
- Aluno → trilhas, calendário, desempenho

🌐 RF11 — Tema Claro/Escuro

O sistema deve permitir alternância entre modos de exibição:

- Tema claro
- Tema escuro
  Salvar preferência no localStorage.

🧩 RF12 — Trilhas de Aprendizagem Inteligentes (IA)

O sistema deve gerar trilhas personalizadas com base em:

- Perfil do aluno
- Histórico de desempenho
- Preferências de aprendizagem
- Materiais recomendados

(Obs: Funcionalidade avançada, aplicada após módulo .NET + integração backend)

💬 RF13 — Sistema de Feedback & Chat Interno

O sistema deve permitir:

- Envio de mensagens entre professor ↔ aluno
- Feedback rápido sobre entregas

- Histórico de conversas (banco de dados)

🧪 RF14 — Sistema de Avaliações e Entregas

O sistema deve permitir:

- Professores cadastrarem provas/trabalhos
- Alunos enviarem entregas (upload)
- Correção e feedback

🎓 RF15 — Portal do Aluno

O aluno deve poder visualizar:

- boletim
- frequências
- trilhas
- compromissos
- avisos
- notificações pendentes

2.2 Requisitos Não Funcionais (RNF)

⚡ RNF01 — Desempenho

- O sistema deve carregar qualquer página em menos de **3 segundos** em condições normais.
- O calendário deve atualizar sem recarregar a página.

📱 RNF02 — Responsividade

O sistema deve funcionar corretamente em:

- Desktop
- Tablet
- Mobile
  Usando grid e estrutura responsiva.

🔒 RNF03 — Segurança

- Dados sensíveis devem ser protegidos.
- As senhas devem ser armazenadas com hashing (no backend).
- A API deve ser protegida com JWT (futuro).

🧩 RNF04 — Arquitetura

- Front modular e componentizado.
- Backend baseado em arquitetura de camadas.
- Padrão REST para API.

🛠️ RNF05 — Manutenibilidade

- Código organizado em módulos.
- Comentários e docstrings quando necessário.
- Documentação completa (antes, durante e depois).

🔄 RNF06 — Versionamento

- Todo desenvolvimento deve seguir boas práticas de Git Flow.
- PRs obrigatórias para funcionalidades novas.

🎨 RNF07 — Usabilidade

- Interface intuitiva.
- Padrões de cores e ícones consistentes.
- Feedback visual para ações do usuário.

🔊 RNF08 — Acessibilidade

- Contraste adequado.
- Navegabilidade por teclado.
- Elementos descritivos (alt em imagens).

🌐 RNF09 — Compatibilidade

Suporte aos navegadores:

- Chrome
- Edge
- Firefox
- Opera

🧪 RNF10 — Testes

- O sistema final deve possuir testes automatizados (unitários e integração).
- Front (React) → Jest + React Testing Library
- Back (API .NET) → xUnit

---

3. MODELAGEM DE DADOS

A modelagem de dados do **EduConnect** define a estrutura lógica e relacional do sistema, garantindo consistência, integridade e escalabilidade das informações.
Ela foi planejada para atender aos diferentes tipos de usuários, operações do ambiente escolar e futuras expansões (como API .NET, autenticação real e dashboards personalizados).

A modelagem foi dividida em:

- **3.1 Entidades Principais**
- **3.2 Relacionamentos entre Entidades**
- **3.3 Modelo Conceitual (DER — Diagrama Entidade Relacionamento)**
- **3.4 Modelo Lógico (Tabelas e Colunas)**
- **3.5 Regras de Integridade**
- **3.6 Considerações sobre Segurança e Escalabilidade do Banco**

3.1 Entidades Principais

A seguir estão as principais entidades previstas no sistema EduConnect:

1. Usuário

Representa qualquer pessoa que acessa a plataforma.

Atributos:

- `id_usuario`

- `nome`
- `email`
- `senha_hash`
- `tipo_usuario` (aluno, professor, administrador)
- `data_criacao`
- `status` (ativo/inativo)
- `ultimo_acesso`

---

2. Aluno

Dados específicos de um aluno.
Cada aluno é um *usuário* do sistema.

Atributos:

- `id_aluno`
- `id_usuario` (FK → Usuário)
- `matricula`
- `data_nascimento`
- `turma_atual`
- `responsavel_nome`
- `responsavel_contato`

---

3. Professor

Dados específicos de um professor (também é um usuário).

Atributos:

- `id_professor`
- `id_usuario` (FK)
- `formacao`
- `especialidade`
- `carga_horaria`
- `disciplinas_ministradas` (lista, descrita na tabela Disciplinas)

---

4. Disciplina

Matérias que podem ser atribuídas aos professores e alunos.

Atributos:

- `id_disciplina`
- `nome`
- `descricao`
- `carga_horaria`

---

5. Turma

Turmas do ano letivo.

Atributos:

- `id_turma`
- `nome_turma` (ex.: 1ºA, 3ºB)

- `ano_letivo`
- `turno` (manhã / tarde / integral)

---

6. Calendário / Eventos

Eventos escolares como provas, reuniões, avisos gerais, simulados etc.

Atributos:

- `id_evento`
- `titulo`
- `descricao`
- `data_evento`
- `tipo_evento` (prova, reunião, comunicado, simulado etc.)
- `criado_por` (FK usuário)
- `publico_alvo` (alunos, professores, turma específica etc.)

---

7. Notificação

Sistema interno de alertas para alunos, professores e administradores.

Atributos:

- `id_notificacao`
- `id_usuario_destino`
- `mensagem`
- `tipo` (informativa, alerta, evento, sistema)

- `data_envio`
- `lida` (boolean)

---

8. Registro de Presença

Frequência registrada em sala de aula.

Atributos:

- `id_presenca`
- `id_aluno`
- `id_disciplina`
- `data`
- `status` (presente, ausente, atraso, justificativa)

---

9. Avaliações / Notas

Notas e avaliações feitas pelos professores.

Atributos:

- `id_avaliacao`
- `id_aluno`
- `id_disciplina`
- `descricao`
- `data_avaliacao`
- `nota`

- `peso`

---

10. Logs do Sistema

Registro de auditoria para segurança e monitoramento.

Atributos:

- `id_log`
- `id_usuario`
- `descricao`
- `ip_origem`
- `timestamp`

3.2 Relacionamentos entre Entidades

- **Um Usuário pode ser:**
    - 1 aluno **ou**
    - 1 professor **ou**
    - 1 administrador
      (relação 1 → 1 com Aluno/Professor)
- **Um Professor ministra várias disciplinas**
  (1 → N)
- **Uma Disciplina pode ser ministrada por vários professores**
  (N → 1)
  Ou N ↔ N, caso a escola permita co-docência.
- **Uma Turma possui vários alunos**
  (1 → N)

- **Um Aluno pode ter várias avaliações**
  (1 → N)
- **Um Evento pode ser destinado a vários usuários**
  (N → N)
- **Um Aluno possui diversos registros de presença**
  (1 → N)
- **Um Usuário recebe várias notificações**
  (1 → N)

3.3 Modelo Conceitual (DER – Diagrama Entidade Relacionamento)

(Essa estrutura será representada em um diagrama visual no Draw.io).

Representação textual do DER:

```
USUARIO (1) ----- (1) ALUNO
USUARIO (1) ----- (1) PROFESSOR

PROFESSOR (1) ----- (N) DISCIPLINA
ALUNO (1) ----- (N) TURMA

ALUNO (1) ----- (N) AVALIACAO
ALUNO (1) ----- (N) PRESENCA

EVENTO (N) ----- (N) USUARIO  (EVENTO_USUARIO)

USUARIO (1) ----- (N) NOTIFICACAO

DISCIPLINA (1) ----- (N) AVALIACAO
```

3.4 Modelo Lógico (Tabelas e Colunas)

Tabela: Usuario

Campo	Tipo	Descrição
id_usuario	INT PK	Identificador
nome	VARCHAR(120)	Nome completo
email	VARCHAR(120) UNIQUE	Login
senha_hash	VARCHAR(255)	Hash da senha
tipo_usuario	ENUM(aluno, professor, admin)	Perfil
status	BOOLEAN	Ativo/Inativo
ultimo_acesso	DATETIME	Último login

Tabela: Aluno

Campo	Tipo
id_aluno	INT PK
id_usuario	INT FK
matricula	VARCHAR(20)
data_nascimento	DATE
turma_atual	VARCHAR(20)
responsavel_nome	VARCHAR(120)
responsavel_contato	VARCHAR(20)

Tabela: Professor

Campo	Tipo
id_professor	INT PK
id_usuario	INT FK
formacao	VARCHAR(120)
especialidade	VARCHAR(120)

carga_horaria	INT

Tabela: Evento

Campo	Tipo
id_evento	INT PK
titulo	VARCHAR(120)
descricao	TEXT
data_evento	DATE
tipo_evento	VARCHAR(50)
criado_por	INT FK
publico_alvo	VARCHAR(50)

Tabela: Eventos_Usuarios *(tabela associativa)*

Campo	Tipo
id_evento	INT FK
id_usuario	INT FK

Tabela: Avaliacao

Campo	Tipo
id_avaliacao	INT PK
id_aluno	INT FK
id_disciplina	INT FK
descricao	VARCHAR(120)
data_avaliacao	DATE
nota	DECIMAL(4,2)
peso	DECIMAL(3,1)

👉 E segue assim para todas as tabelas.

3.5 Regras de Integridade

Algumas regras importantes:

1. **Usuários só podem existir se estiverem vinculados a um perfil (Aluno/Professor/Admin).**
2. **Um evento só pode ser marcado por um usuário administrador ou professor.**
3. **Avaliações só podem ser cadastradas por professores.**
4. **Cada aluno deve pertencer a uma única turma.**
5. **Eventos podem ser visualizados somente por usuários autorizados (visão por público-alvo).**
6. **Notificações são exclusivas por usuário (não compartilhadas).**
7. **Notas devem ser entre 0 e 10.**
8. **Datas de eventos não podem ser retroativas (exceto histórico).**

3.6 Considerações sobre Segurança e Escalabilidade

- Senhas serão armazenadas com **hash forte (bcrypt)**.
- Banco utilizará **índices** em:
    - email
    - id_aluno
    - id_professor
    - data_evento

- Separação do banco em:
    - **Tabelas de leitura rápida** (logs, notificações)
    - **Tabelas transacionais** (aluno, professor, notas)
- Futuro suporte a:
    - Cache
    - Filas de mensagens
    - Integração com serviços

---

4. ARQUITETURA DO SISTEMA

4.1 Visão Geral da Arquitetura

O EduConnect será desenvolvido seguindo uma arquitetura em **camadas**, separando responsabilidades para garantir:

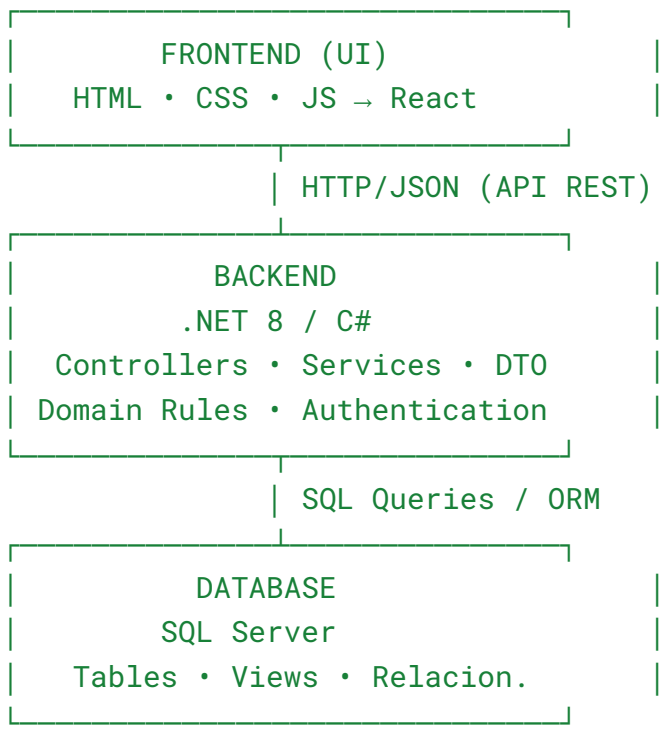
- Clareza
- Escalabilidade
- Facilidade de manutenção
- Organização modular
- Evolução futura sem reescrever tudo

A arquitetura completa será organizada em **três grandes camadas**:

1. **Frontend (Interface do Usuário)**
2. **Backend (Serviços e Lógica de Negócio)**
3. **Banco de Dados (Persistência de Dados)**
4. **Integrações Futuras (Opcional — APIs Externas)**

4.2 Arquitetura em Camadas

Abaixo, o diagrama conceitual da arquitetura:



4.3 Detalhamento das Camadas

4.3.1 Frontend (Interface do Usuário)

Tecnologias:

- HTML5
- CSS3 (modular: style, dashboard, interactivity)
- JavaScript ES6+
- Chart.js (gráficos)

- Interatividade personalizada
- React.js (fase final)

Papel do Frontend:

- Apresentar a interface gráfica ao usuário
- Enviar requisições ao backend via API
- Exibir dashboards, notificações, eventos
- Gerenciar navegação e telas
- Validar formulários antes do envio

Transição HTML/CSS/JS → React

Ao final do programa, o frontend será migrado para React, com estrutura:

```
src/
 ├── components/
 ├── pages/
 ├── hooks/
 ├── services/
 ├── context/
 └── utils/
```

4.3.2 Backend (.NET + C#)

Tecnologias:

- .NET 8 Web API
- C#
- Entity Framework Core
- JwtBearer Authentication
- SQL Server

- AutoMapper
- FluentValidation

Padrão de Arquitetura Utilizado

Clean Architecture + Domain Driven Design (simplificado)

Camadas:

```
Application/
Domain/
Infrastructure/
API/
```

Responsabilidades principais:

Camada	Função
Domain	Entidades, regras centrais do negócio
Application	Casos de uso, validações, serviços
Infrastructure	Banco, repositórios, EF Core
API	Controllers, autenticação, rotas

Principais Serviços do Backend

- Serviço de Autenticação (login, roles, tokens JWT)
- Serviço de Matrículas (CRUD completo)
- Serviço de Eventos Acadêmicos (calendário)
- Serviço de Desempenho Escolar (médias, gráficos)
- Serviço de Comunicação (mensagens entre usuários)
- Serviço de Gestão Administrativa

4.3.3 Banco de Dados (SQL Server)

O banco será relacional, modelado com **princípios ACID** e normalização até 3FN.

Tecnologias:

- Microsoft SQL Server
- Entity Framework Core
- Migrations para versionamento

Principais tabelas (detalhadas na Seção 3):

- Usuarios
- Alunos
- Professores
- Administradores
- Cursos
- Disciplinas
- Turmas
- Matriculas
- Eventos
- Avaliacoes
- Comunicados

4.3.4 Integrações Futuras (Opcional)

O sistema será preparado para se integrar a:

- API oficial do MEC (dados escolares)
- Integração com plataformas de pagamento para mensalidades
- API de calendário (Google Calendar)

- API de envio de e-mail (SendGrid)
- Autenticação via OAuth (Google / Microsoft)

Essas integrações poderão ser aplicadas pelo uso de serviços isolados:

```
/Infrastructure/Integrations/
    - GoogleCalendarService.cs
    - EmailService.cs
    - OAuthService.cs
```

4.4 Fluxo Geral de Comunicação

Exemplo: aluno acessa o calendário

```
Frontend → (GET /api/eventos?data=2025-11-12) → Backend
Backend → Consulta SQL → Banco → Retorna JSON
Frontend → Renderiza os eventos + notificações
```

4.5 Decisões de Arquitetura (ADRs)

Será documentado decisões importantes, como:

- Uso de JWT para autenticação por segurança
- Uso de SQL Server por robustez e facilidade de integração
- Migração planejada para React no front
- Adoção de Clean Architecture para organizar camadas

---

5. ARQUITETURA DA API

Esta seção descreve **como a API será organizada**, quais camadas existirão, como os módulos conversam entre si e qual padrão arquitetural será utilizado. Essa arquitetura foi pensada para:

- ser escalável
- ser fácil de manter
- funcionar bem com .NET + C#
- suportar novos módulos no futuro
- isolar regras de negócio
- fornecer segurança e permissões robustas

🔷 5.1. Padrão Arquitetural Adotado

A API do EduConnect seguirá a abordagem **Clean Architecture + DDD simplificado**, composta por camadas bem definidas:

1. Domain (Núcleo do sistema)

Contém:

- entidades
- interfaces
- regras de negócio
- validações

2. Application

Contém:

- casos de uso
- DTOs
- mapeamentos

- validação de entrada
- orquestração de fluxos

3. Infrastructure

Contém:

- acesso ao banco (EF Core)
- repositórios
- implementações externas
- envio de email
- logging
- autenticação (JWT)

4. API

Contém:

- controladores
- endpoints
- filtros
- autenticação/autorização
- documentação Swagger

📌 Esse modelo isola as responsabilidades e permite trocar o banco, atualizar regras de negócio ou mudar o front-end sem mexer no núcleo.

🔷 5.2. Organização dos Módulos da API

A API será dividida nos seguintes módulos (todos independentes e escaláveis):

5.2.1. Módulo de Autenticação & Autorização

Responsável por:

- cadastro de usuários
- login
- renovação de tokens
- controle de permissões por papel
- gestão de sessão
- recuperação de senha (futuro)

Papéis iniciais:

- **Aluno**
- **Professor**
- **Administrador**

Autorização via **JWT + Claims + Roles**.

5.2.2. Módulo de Usuários (User Management)

Gerencia informações dos usuários.

Endpoints essenciais:

- GET /users
- GET /users/{id}
- POST /users
- PUT /users/{id}
- DELETE /users/{id}

Cada usuário tem:

- dados pessoais
- papel
- status
- permissões específicas

5.2.3. Módulo de Alunos

Gerencia:

- perfil do aluno
- matrícula
- turma
- boletim
- calendário do aluno
- notificações do aluno

5.2.4. Módulo de Professores

Gerencia:

- disciplinas
- turmas que leciona
- registro de notas
- calendário docente
- notificações internas

5.2.5. Módulo de Turmas

Gerencia:

- criação de turmas
- associação aluno ↔ turma
- associação professor ↔ turma
- calendário de provas
- planejamento pedagógico future-proof

5.2.6. Módulo de Disciplinas

Gerencia:

- cadastro das matérias
- relação com turmas
- conteúdos

5.2.7. Módulo de Matrículas

Gerencia o fluxo de:

1. Solicitação de matrícula
2. Aprovação (Admin)
3. Atribuição do aluno à turma
4. Confirmação

5.2.8. Módulo de Calendário Acadêmico

Gerencia:

- provas
- trabalhos
- reuniões
- eventos escolares
- lembretes automatizados (push)

Há integração futura com:

- notificações
- dashboards
- painel do aluno

5.2.9. Módulo de Notas & Avaliações

Gerencia:

- notas individuais
- médias automáticas
- histórico escolar
- boletim completo
- critérios de cálculo

Gerará dados para:

- **gráficos do dashboard**

5.2.10. Módulo de Relatórios & Dashboard

Fornece dados consolidados como:

- desempenho geral da turma
- performance por disciplina
- frequência
- indicadores pedagógicos
- analytics

🔷 5.3. Estrutura de Pastas da API (.NET)

Uma organização sugerida:

```
/src
  /EduConnect.API
  /EduConnect.Application
  /EduConnect.Domain
  /EduConnect.Infrastructure

/tests
  /EduConnect.UnitTests
  /EduConnect.IntegrationTests
```

🔷 5.4. Fluxo de Autenticação

O fluxo será baseado em:

1. Usuário envia credenciais
2. Sistema valida
3. API retorna:
    - Token JWT
    - Refresh Token
    - Claims + Permissões

4. O token é enviado em cada requisição protegida

Cada módulo do sistema usará **autorização baseada em papéis**:

Módulo	Aluno	Professor	Administrador
Calendário	✔️	✔️	✔️
Notas	leitura	edição	total
Turmas	leitura	parcial	total
Usuários	❌	❌	total
Relatórios	parcial	total	total

🔷 5.5. Padrão de Resposta da API (Response Contract)

Todas as respostas devem seguir este formato:

```
{
  "success": true,
  "data": { },
  "message": "Operação realizada com sucesso",
  "errors": []
}
```

Em caso de erro:

```
{
  "success": false,
  "data": null,
  "message": "Falha ao realizar a operação",
  "errors": ["Campo Email é obrigatório"]
}
```

🔷 5.6. Documentação da API

Será utilizada:

- **Swagger / OpenAPI 3**
- versão documentada automaticamente
- exemplos de requisição e resposta
- suporte a JWT via Swagger UI

🔷 5.7. Estratégia de Segurança

A API contará com:

- Rate limiting
- Proteção contra brute force (login)
- JWT com expiração curta + refresh tokens
- Claims por usuário
- Sanitização de entrada (evitar SQL Injection)
- Logs de autenticação
- Hash de senha usando **BCrypt**

🔷 5.8. Estratégia de Escalabilidade

A arquitetura prevê:

- Deploy desacoplado do front-end
- API stateless
- Suporte a cluster com load balancer
- Caching de consultas pesadas

- Banco escalável (SQL Server)
- Possível migração futura para microsserviços (se necessário)

---

6. DIAGRAMAS DE CASO DE USO

Esta seção descreve **como cada tipo de usuário interage com o sistema**, detalhando seus objetivos, funcionalidades disponíveis e fluxos principais.

A seguir está a versão completa e organizada por atores.

🎭 6.1. Atores do Sistema

O EduConnect possui três atores principais:

Aluno

- Acessa informações pessoais
- Consulta calendário
- Visualiza boletim/notas
- Recebe notificações
- Realiza matrícula

Professor

- Gerencia notas
- Gerencia turmas
- Lança atividades
- Consulta calendário docente
- Envia comunicados

Administrador

- Gerencia usuários
- Gerencia turmas
- Gerencia disciplinas
- Aprova matrículas
- Acompanha relatórios gerais

✏️ 6.2. Lista Geral de Casos de Uso

Agrupados por ator.

👨‍🎓 6.3. Casos de Uso — Aluno

UC01 — Realizar Login

Descrição: Permite que o aluno acesse o sistema.
Atores: Aluno
Fluxo Principal:

1. Aluno informa usuário e senha.
2. Sistema valida credenciais.
3. Sistema gera token e redireciona para o painel do aluno.

UC02 — Consultar Dados Pessoais

Descrição: Exibe informações de perfil, turma e dados acadêmicos.
Atores: Aluno

UC03 — Consultar Calendário Acadêmico

Descrição: Exibe eventos do calendário.
Atores: Aluno
Fluxo Principal:

1. Aluno acessa o calendário.
2. Sistema exibe os eventos do mês.
3. Aluno clica em um dia.
4. Sistema exibe os compromissos daquele dia.

UC04 — Consultar Notas / Boletim

Descrição: Permite visualizar notas de avaliações e médias.
Atores: Aluno

UC05 — Realizar Matrícula

Descrição: Aluno solicita matrícula em uma turma ou recebe token QR para registrar matrícula.
Fluxo Alternativo (QR):

- Aluno lê QR code → Sistema valida → Matrícula aprovada automaticamente.

UC06 — Receber Notificações

Descrição: Sistema envia notificações de novos eventos, notas e avisos.

👨‍🏫 6.4. Casos de Uso — Professor

UC07 — Lançar Notas

Descrição: Permite inserir notas dos alunos da turma.
Atores: Professor
Fluxo Principal:

1. Professor seleciona a turma.
2. Escolhe atividade.
3. Insere notas.
4. Sistema salva e recalcula médias.

UC08 — Registrar Atividades / Trabalhos

Criar atividades, provas e tarefas vinculadas à turma.

UC09 — Consultar Turmas

Visualizar os alunos e disciplinas associadas.

UC10 — Enviar Avisos / Comunicados

Professor envia mensagens para:

- turma específica
- aluno específico
- todos os alunos

UC11 — Consultar Calendário Docente

Exibe reuniões, provas, eventos pedagógicos etc.

🛠️ 6.5. Casos de Uso — Administrador

UC12 — Gerenciar Usuários

CRUD de:

- Alunos
- Professores
- Administradores

UC13 — Gerenciar Turmas

CRUD:

- criar turma
- editar turma
- associar alunos
- associar professor

UC14 — Gerenciar Disciplinas

CRUD de matérias (Matemática, História, Ciências, etc.).

UC15 — Aprovar Matrículas

Recebe solicitações e aprova/nega.

UC16 — Gerar Relatórios

Exibe dados como:

- desempenho geral
- alunos em risco

- notas por disciplina
- engajamento do aluno

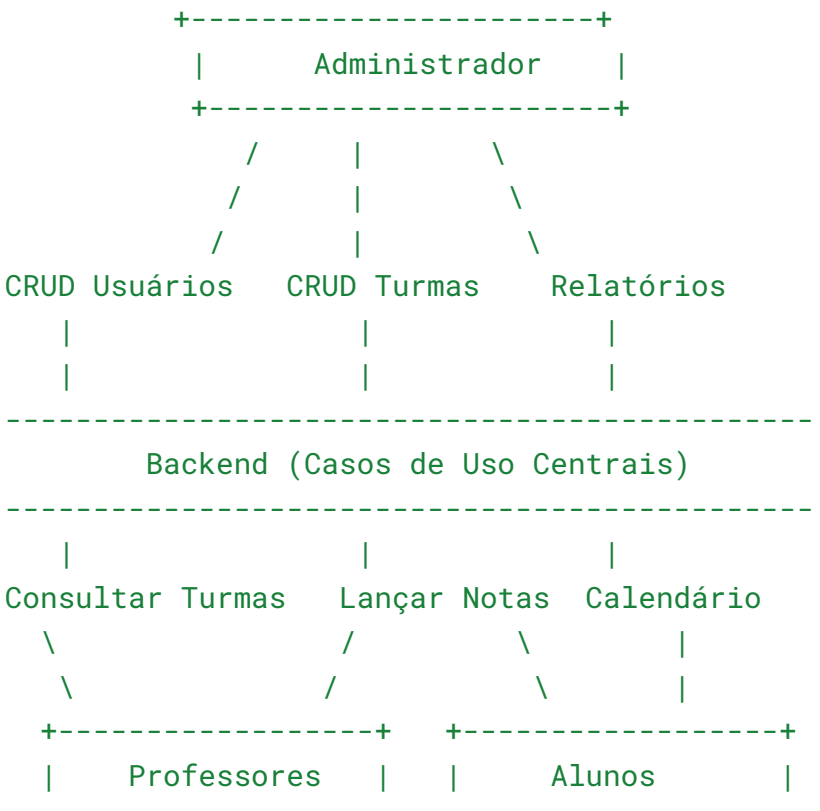
UC17 — Gerenciar Calendário Acadêmico

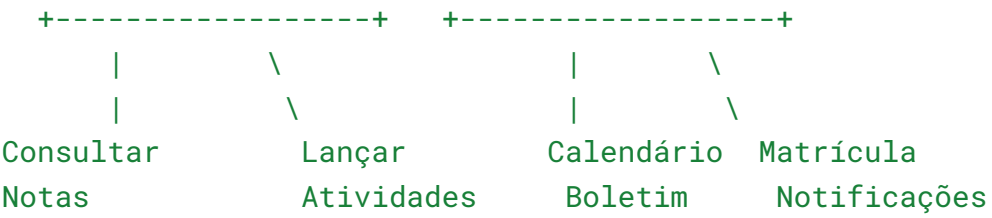
O admin pode:

- criar eventos gerais
- criar feriados
- configurar datas de fechamento

🗂️ 6.6. Diagrama de Caso de Uso — Visão Geral (Descrição)

Se fosse representado visualmente, ficaria assim:





---

7. FLUXOS DE USUÁRIO (USER FLOWS)

Os fluxos de usuário representam os caminhos que cada ator segue para realizar suas atividades dentro do **EduConnect**, desde a autenticação até a conclusão de ações específicas.

Eles são fundamentais para futuras implementações em **C# (.NET)** e depois no **React**, pois ajudam a garantir consistência entre telas, API e regras de negócio.

7.1. User Flow — Aluno

🎓 Fluxo 1: Login → Dashboard do Aluno

Objetivo: acessar o sistema.

Fluxo:

1. Aluno acessa a página inicial.
2. Insere e-mail e senha.
3. Sistema valida credenciais.
4. Redireciona para o dashboard do aluno.

🎓 Fluxo 2: Consultar Calendário e Eventos

1. Aluno acessa menu “Calendário”.
2. Sistema exibe calendário mensal.

3. Aluno seleciona um dia.
4. Sistema exibe os compromissos abaixo do calendário.
5. Aluno lê detalhes ou recebe notificação visual.

🎓 Fluxo 3: Consultar Notas / Boletim

1. Aluno abre a seção “Boletim”.
2. Sistema carrega matérias e notas.
3. Aluno visualiza médias e desempenho geral.

🎓 Fluxo 4: Realizar Matrícula

1. Aluno acessa “Matrícula”.
2. Escolhe turma disponível **ou** faz leitura de QR Code (entrega física).
3. Envia solicitação.
4. Sistema registra e envia para administrador aprovar.

7.2. User Flow — Professor

👨‍🏫 Fluxo 1: Login → Dashboard do Professor

Mesmo fluxo do aluno, mas redireciona para funcionalidades docentes.

👨‍🏫 Fluxo 2: Lançamento de Notas

1. Professor acessa “Turmas”.

2. Seleciona a turma desejada.
3. Seleciona a atividade.
4. Insere notas dos alunos.
5. Salva registros.
6. Sistema recalcula média automaticamente.

👨‍🏫 Fluxo 3: Criar Atividades / Provas

1. Seleciona turma.
2. Clica em “Nova atividade”.
3. Informações: título, data, peso, descrição.
4. Sistema registra atividade e adiciona ao calendário da turma.

👨‍🏫 Fluxo 4: Enviar Comunicados

1. Acessa “Avisos”.
2. Seleciona destino (turma / aluno / geral).
3. Escreve mensagem.
4. Envia.
5. Sistema gera notificações para o(s) aluno(s).

7.3. User Flow — Administrador

🛠️ Fluxo 1: Gerenciar Usuários

1. Admin abre “Gestão de Usuários”.
2. Escolhe criar, editar ou remover.
3. Preenche dados.
4. Sistema valida e salva.

🛠️ Fluxo 2: Aprovar Matrículas

1. Admin acessa “Matrículas Pendentes”.
2. Visualiza lista de solicitações.
3. Seleciona um aluno.
4. Escolhe Aprovar ou Rejeitar.
5. Sistema notifica o aluno.

🛠️ Fluxo 3: Gerenciar Turmas

1. Acessa menu “Turmas”.
2. Cria turma nova ou edita existente.
3. Define:
    - nome
    - série
    - disciplina
    - professor responsável
4. Sistema salva alterações.

🛠️ Fluxo 4: Manter Calendário Acadêmico

1. Acessa “Calendário Geral”.
2. Cria eventos:
    - feriados
    - provas gerais
    - fechamento de notas
3. Sistema atualiza calendário global.

7.4. Visão Geral de Navegação (Mapa Macro do Sistema)

Esta visão mostra como todos os usuários se conectam ao fluxo geral:

```
Login
  ├─ Dashboard (Aluno)
  │      ├─ Calendário
  │      ├─ Boletim
  │      ├─ Matrícula
  │      └─ Perfil
  ├─ Dashboard (Professor)
  │      ├─ Turmas
  │      ├─ Atividades
  │      ├─ Lançamento de notas
  │      └─ Comunicados
  └─ Dashboard (Admin)
         ├─ Usuários
         ├─ Turmas
         ├─ Disciplinas
         ├─ Matrículas
         └─ Relatórios
```

---

8. TELAS (UI/UX)

Esta seção descreve todas as interfaces planejadas para o **EduConnect**, abrangendo os três tipos de usuário: **Aluno**, **Professor** e **Administrador**.

Para cada tela há:

- descrição
- objetivo
- elementos principais
- requisitos de navegação
- dados consumidos/retornados pela API
- (opcional) wireframe em texto + versão futura em React

8.1. Telas Globais (Comuns a Todos os Usuários)

---

1. Tela de Login

Objetivo

Permitir autenticação segura de usuários (Aluno, Professor e Administrador).

Elementos

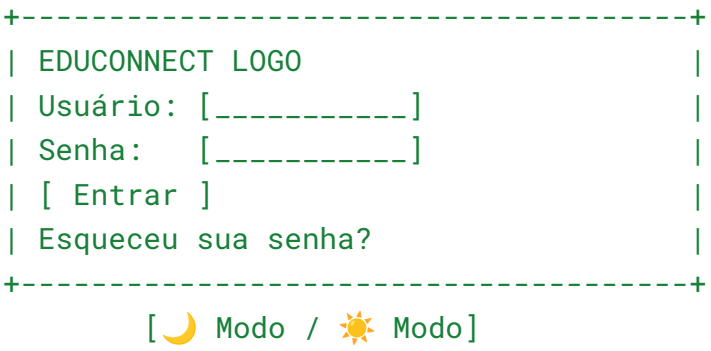
- Campo de usuário/e-mail
- Campo de senha
- Botão “Entrar”
- Mensagem de erro

- Botão “Esqueceu sua senha?”
- Alternância de tema (claro/escuro)

Regras de Negócio

- Credenciais são validadas via API.
- Se o login for válido → redireciona conforme tipo de usuário.
- Se inválido → exibe mensagem amigável.

Wireframe (simplificado)



---

2. Tela de Dashboard (com variação por tipo de usuário)

Objetivo

Centralizar acesso às funcionalidades respectivas de cada perfil.

Elementos Comuns

- Sidebar lateral (ou topbar no mobile)
- Saudação ao usuário
- Tema claro/escuro
- Botão de sair
- Área dinâmica de conteúdo

Variação por usuário

- **Aluno:** Boletim, calendário, matrícula
- **Professor:** Turmas, atividades, lançamento de notas
- **Admin:** matrículas, usuários, turmas, relatórios

8.2. Telas do Aluno

---

1. Dashboard do Aluno

Objetivo

Exibir visão geral acadêmica.

Elementos

- Próximos compromissos
- Últimas notas
- Avisos recentes do professor

Chamadas à API

- `GET /aluno/calendario/proximos-eventos`
- `GET /aluno/notas/resumo`
- `GET /avisos`

---

2. Tela de Calendário (Aluno)

Objetivo

Apresentar calendário acadêmico do aluno com eventos e compromissos.

Elementos

- Navegação entre meses
- Destaque de dias com eventos
- Cards de compromissos abaixo
- Notificações visuais (UI aplicada)

Regras

- Clique no dia mostra compromissos
- Dias com eventos possuem indicador visual (`event-dot`)

---

3. Tela de Boletim

Objetivo

Exibir notas por matéria e média geral.

Elementos

- Tabela de notas
- Cálculo de média
- Gráfico de desempenho (Chart.js → futuro React + Recharts)

API

- `GET /aluno/notas`

---

4. Tela de Matrícula

Objetivo

Permitir que o aluno solicite matrícula em uma turma.

Elementos

- Lista de turmas disponíveis
- Botão “Solicitar matrícula”
- Status da solicitação
- Possível leitura de QR Code (funcionalidade futura)

API

- `POST /matriculas/solicitar`

8.3. Telas do Professor

---

1. Dashboard do Professor

Visão geral de suas turmas, atividades e compromissos.

---

2. Tela de Turmas

Objetivo

Listar todas as turmas associadas ao professor.

Elementos

- Lista de turmas
- Pesquisa

- Acesso rápido à turma

API

- `GET /professor/turmas`

---

3. Tela de Lançamento de Notas

Objetivo

Registrar notas de atividades/provas.

UI

- Tabela com alunos
- Campo para nota
- Botão salvar
- Feedback visual

---

4. Tela de Criação de Atividade

Objetivo

Professor cria atividades avaliativas.

Elementos

- Título
- Data
- Peso
- Descrição

- Botão salvar

API

- `POST /atividades`

---

5. Tela de Comunicados

Objetivo

Enviar avisos para alunos/turmas.

Elementos

- Editor de texto
- Seleção de destinos
- Histórico de comunicados enviados

8.4. Telas do Administrador

---

1. Tela de Gestão de Usuários

Objetivo

CRUD completo de usuários.

Elementos

- Lista de usuários
- Filtros
- Botão criar

- Modal editar
- Botão excluir

---

2. Tela de Matrículas Pendentes

Objetivo

Aprovar ou rejeitar matrícula de alunos.

Fluxo

- Lista pendentes
- Visualização do aluno
- Aprovar / Rejeitar
- Sistema notifica o aluno

---

3. Tela de Gestão de Turmas

Objetivo

Criar e administrar turmas.

Elementos

- Nome da turma
- Série
- Disciplina
- Professor responsável
- Controles: criar/editar/excluir

---

4. Tela de Relatórios

Possíveis relatórios

- Quantidade de alunos por turma
- Desempenho geral por disciplina
- Taxa de aprovação
- Dados exportáveis

8.5. Identidade Visual e Layout

Tema Claro/Escuro

- Já implementado no MVP
- No React será feito via Context API ou Zustand

Design System Inicial

Elementos padrão que devem ser mantidos em toda a aplicação:

- Botões com bordas arredondadas
- Caixas com sombra leve
- Cards de conteúdo
- Tipografia base: Segoe UI
- Paleta:
    - Azul primário: `#2e86de`
    - Verde: para eventos e sucesso
    - Vermelho: para erros

Comportamentos Globais

- Feedback visual imediato em todas as ações
- Animações leves (hover, clicks, notificações)
- Mobile-first obrigatório
- Navegação clara e padronizada

8.6. Wireframes Gerais (modo texto)

Calendário

```
MÊS | <     Novembro 2025     >
-----------------------------------
 D  S  T  Q  Q  S  S
 .  .  1  2  3  4  5
 6  7  8  9 [10]* 11
...
-----------------------------------
Compromissos:
 - Prova de Matemática
```

Boletim

```
Disciplina     | Nota | Peso | Média
-------------------------------------
Matemática     | 8.0  | 3    | 8.2
Português      | 7.5  | 2    | 7.4
```

---

9. ARQUITETURA DO BACK-END (.NET)

9.1 Visão Geral

O back-end do **EduConnect** será implementado como uma **Web API** em **.NET 8 (C#)** seguindo a **Clean Architecture** (camadas separadas) e DDD simplificado. Objetivos principais:

- Isolar regras de negócio (Domain) das infra-estruturas (DB, frameworks).
- Garantir testabilidade (unit/integration).
- Ser stateless e escalável (pronto para containers).
- Facilitar migração/integração com front React e outros serviços.

9.2 Estrutura de projetos (mono-repo)

Layout (solução .NET):

```
/src
  /EduConnect.API                # Startup, controllers, swagger
  /EduConnect.Application        # Use cases, DTOs, interfaces de
serviço
  /EduConnect.Domain             # Entidades, Value Objects, regras
  /EduConnect.Infrastructure     # EF Core DbContext, Repositories,
Migrations
/tests
  /EduConnect.UnitTests
  /EduConnect.IntegrationTests
```

Cada pasta é um *project* (.csproj). Uso de `ProjectReference` entre camadas, sempre do mais alto (API) para o mais baixo (Domain).

9.3 Principais responsabilidades por camada

- **API**: Controllers, filtros, autenticação, configuração de DI, middleware, swagger.
- **Application**: Casos de uso (Services), DTOs, interfaces (IRepository, IEmailService), validações com FluentValidation.
- **Domain**: Entidades ricas, regras de negócio, exceptions específicas do domínio.
- **Infrastructure**: Implementação de repositórios (EF Core), Migrations, serviços externos (Email, GoogleCalendar), persistência.

9.4 Principais tecnologias e pacotes sugeridos

- .NET 8 SDK
- ASP.NET Core Web API
- Entity Framework Core (EF Core) + Provider (SQL Server / PostgreSQL)
- AutoMapper
- FluentValidation
- Swashbuckle (Swagger)
- Microsoft.Identity / JwtBearer
- Serilog (logging)
- xUnit / Moq (testes)
- GitHub Actions (CI)

9.5 Exemplo — DbContext (Infrastructure)

```
public class EduConnectDbContext : DbContext
{
    public EduConnectDbContext(DbContextOptions<EduConnectDbContext>
options)
        : base(options) { }

    public DbSet<Usuario> Usuarios { get; set; }
    public DbSet<Aluno> Alunos { get; set; }
    public DbSet<Professor> Professores { get; set; }
    public DbSet<Evento> Eventos { get; set; }
    public DbSet<Avaliacao> Avaliacoes { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
        // configurações Fluent API (FK, índices, constraints)
        builder.Entity<Usuario>().HasIndex(u => u.Email).IsUnique();
    }
```

```
}
```

9.6 Exemplo — Repository pattern (interface + implementação)

Interface (Application / Domain):

```
public interface IUsuarioRepository
{
    Task<Usuario?> GetByIdAsync(int id, CancellationToken ct =
default);
    Task<Usuario?> GetByEmailAsync(string email, CancellationToken
ct = default);
    Task AddAsync(Usuario user, CancellationToken ct = default);
    Task SaveChangesAsync(CancellationToken ct = default);
}
```

Implementação (Infrastructure):

```
public class UsuarioRepository : IUsuarioRepository
{
    private readonly EduConnectDbContext _db;
    public UsuarioRepository(EduConnectDbContext db) => _db = db;

    public async Task<Usuario?> GetByIdAsync(int id,
CancellationToken ct = default)
        => await _db.Usuarios.FindAsync(new object[]{ id }, ct);

    public async Task<Usuario?> GetByEmailAsync(string email,
CancellationToken ct = default)
        => await _db.Usuarios.FirstOrDefaultAsync(u => u.Email ==
email, ct);

    public async Task AddAsync(Usuario user, CancellationToken ct =
default)
    {
        await _db.Usuarios.AddAsync(user, ct);
    }
```

```
    public Task SaveChangesAsync(CancellationToken ct = default) =>
_db.SaveChangesAsync(ct);
}
```

9.7 Exemplo — Service / Use Case (Application)

```
public class AuthService : IAuthService
{
    private readonly IUsuarioRepository _repo;
    private readonly ITokenGenerator _tokenGen;

    public AuthService(IUsuarioRepository repo, ITokenGenerator
tokenGen)
    {
        _repo = repo; _tokenGen = tokenGen;
    }

    public async Task<AuthResult> LoginAsync(string email, string
password)
    {
        var user = await _repo.GetByEmailAsync(email);
        if (user == null || !PasswordHasher.Verify(password,
user.SenhaHash))
            return AuthResult.Fail("Credenciais inválidas");

        var token = _tokenGen.GenerateToken(user);
        return AuthResult.Success(token, user.Role);
    }
}
```

9.8 Exemplo — Controller (API)

```
[ApiController]
[Route("api/v1/[controller]")]
public class AuthController : ControllerBase
{
    private readonly IAuthService _auth;
    public AuthController(IAuthService auth) => _auth = auth;
```

```
    [HttpPost("login")]
    public async Task<IActionResult> Login([FromBody] LoginDto dto)
    {
        var res = await _auth.LoginAsync(dto.Email, dto.Password);
        if (!res.Success) return Unauthorized(new { message =
res.Error });
        return Ok(new { token = res.Token, role = res.Role });
    }
}
```

9.9 DTOs e Contracts

Defina DTOs simples para request/response no Application:

```
public record LoginDto(string Email, string Password);
public record UsuarioDto(int Id, string Nome, string Email, string
Role);
```

9.10 Autenticação & Autorização

- **JWT Bearer** com claims (user id, role).
- Rotas públicas (/swagger, /health) e rotas protegidas (`[Authorize]` + `[Authorize(Roles="Admin")]`).
- Implementar Refresh Token (opcional para produção).
- Proteção contra brute force: bloqueio após X tentativas (provável via redis / memória temporária).

Configuração típica no `Program.cs`:

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationS
cheme)
  .AddJwtBearer(options => {
    options.TokenValidationParameters = new
TokenValidationParameters { /* chave, issuer, audience */ };
});
```

9.11 Validação e Fail Fast

- **FluentValidation** para DTOs no pipeline (validators para cada DTO).
- Middleware global para captura de exceções (Exception Handling Middleware) que converte exceções em respostas padronizadas.

9.12 Logging e Observability

- **Serilog** para logs estruturados (console / file / Seq).
- Expor endpoints de health (`/health`) e metrics (Prometheus, se desejar).
- Auditoria: gravar ações críticas (criação/remoção de usuários, alterações de notas) na tabela `Logs`.

9.13 Migrations, Seed e Scripts

Com EF Core:

- Criar migration:
  `dotnet ef migrations add InitialCreate -p EduConnect.Infrastructure -s EduConnect.API`
- Aplicar migration:
  `dotnet ef database update -p EduConnect.Infrastructure -s EduConnect.API`
- Seed de dados: criar método Seed no Infrastructure que cria admin inicial, turmas de exemplo, usuários mock.

9.14 Testes

- **UnitTests** (xUnit): services, validators, domain rules.

- **IntegrationTests**: usar `WebApplicationFactory<T>` (test server) e banco InMemory/Sqlite para endpoints mais importantes.
- Cobertura de testes: mínimo para regras críticas (ex.: cálculo de média, regras de matrícula).

9.15 Deploy

Deploy: Usar variáveis de ambiente para connection string e JWT secrets.

9.16 CI / CD (GitHub Actions - exemplo)

Workflow básico:

- `on: [push, pull_request]`
- Steps:
  - checkout
  - setup .NET
  - restore & build & test (`dotnet test`)

9.17 Endpoints principais (exemplo resumido)

Método	Endpoint	Descrição
POST	/api/v1/auth/login	Autenticação
GET	/api/v1/users	Listar usuários (admin)
POST	/api/v1/users	Criar usuário (admin)
GET	/api/v1/alunos/{id}/notas	Notas do aluno
GET	/api/v1/turmas	Listar turmas
POST	/api/v1/eventos	Criar evento (prof/admin)
GET	/api/v1/eventos?date=YYYY-MM-DD	Eventos por data

GET	/api/v1/relatorios/desempenho	Dados para dashboard

9.18 Observações & Boas práticas

- Preferir DTOs para entrada/saída (evitar expor entidades).
- Implementar AutoMapper profiles para mapear entidades ↔ DTOs.
- Aplicar limite de payload e validação de tamanho de arquivos (uploads).
- Documentar versionamento da API: `/api/v1/...` e planejar v2 se necessário.
- Implementar CORS restrito (domínios do front).
- Política de rollback e backups para DB (scripts e rotina).

9.19 Passo-a-passo para iniciar localmente (dev)

1. Configurar `appsettings.Development.json` com connection string local.
2. Aplicar migrations: `dotnet ef database update`
3. Rodar API: `dotnet run --project src/EduConnect.API`
4. Abrir Swagger `http://localhost:5000/swagger` e testar endpoints.

---

10. PLANO DE TESTES

10.1 Objetivo

Estabelecer uma estratégia clara e profissional de testes para o sistema **EduConnect**, garantindo:

- funcionamento correto das funcionalidades;

- segurança (especialmente autenticação e perfis de usuário);
- estabilidade e escalabilidade do back-end;
- boa experiência de uso no front-end;
- minimização de regressões durante o desenvolvimento dos próximos módulos.

O plano cobre testes desde o front atual (HTML, CSS, JS) até o back-end em .NET e o front futuro em React.

10.2 Tipos de Testes

✔ 10.2.1 Testes de Unidade (Unit Tests)

Onde serão aplicados:

- Camada **Domain**
- Camada **Application**

Objetivo:
Garantir que regras de negócio, cálculos e validações funcionem isoladamente.

Exemplos reais:

- Regra de cálculo da média do aluno.
- Validação de matrícula (turma cheia, período encerrado, etc.).
- Comparação de datas para compromissos no calendário.
- Validação de campos (e-mail, CPF, senha).

Ferramentas:

- xUnit
- Moq
- FluentAssertions

Métricas recomendadas:

- Cobertura de testes acima de 60% inicialmente (ideal +80% no futuro).

✔ 10.2.2 Testes de Integração

Onde serão aplicados:

- Infraestrutura (EF Core + Banco de Dados)
- API (controladores + serviços)

Objetivo:
Verificar se as camadas se comunicam corretamente.

Exemplos reais:

- Criar aluno → salvar no banco → recuperar na API.
- Login → gerar JWT com claims corretas.
- Criar evento → carregar no calendário.
- Lançar notas → persistir no banco → atualizar somatório do aluno.

Ferramentas:

- xUnit
- `WebApplicationFactory<T>` (Microsoft)
- Banco **SQLite InMemory** para testes
- Testcontainers (opcional para futuro, se quiser testar com Postgres real)

✔ 10.2.3 Testes de Interface (Front-end Manual / Automação no futuro)

- ***Fase atual (HTML + CSS + JS): testes manuais guiados***

Checklist para validação:

- Responsividade (320px → 1920px)
- Tema claro/escuro
- Tabela filtrando resultados corretamente
- Cadastro de alunos/professores preenchendo a tabela em tempo real
- Notificações funcionando
- Calendário navegando entre meses
- Eventos sendo carregados abaixo do calendário
- Dashboard adaptando no mobile
- Botão de logout funcionando no mobile e desktop

🔹 ***Fase futura (React): testes automatizados com Jest + Testing Library***

Exemplos que vamos criar:

- Renderização da Dashboard
- Mock de API com MSW
- Teste de formulário com validação
- Teste de rota protegida (PrivateRoute)

✔ 10.2.4 Testes de API (Postman / Swagger / Automatizados)

🔹 ***Testes manuais iniciais:***

- Coleção Postman com todos os endpoints
- Ambientes: Dev / Local
- Testes de Login (200, 401)

- Testes de criação (201)
- Testes de consulta (200)
- Testes de erro esperado (400, 404)

🔹 ***Testes automatizados com Newman (opcional futuro)***

- Execução dos testes via CLI
- Integração com GitHub Actions

✔ 10.2.5 Testes de Regressão

Objetivo:
Garantir que novas funcionalidades **não quebrem** o que já funciona.

Aplicação:

- Após adicionar React
- Após adicionar o back-end
- Ao refatorar tabelas, calendário, notificações etc.

Checklist sugerido:

- Login funciona?
- Tema persiste?
- Dashboard carrega?
- Formulário salva?
- Calendário carrega compromissos?
- API responde corretamente?

✔ 10.2.6 Testes de Segurança

Aplicados após o módulo .NET estar ativo.

Itens críticos:

- Proteção JWT (expiração, assinatura e claims)
- Bloqueio de tentativas de login (rate limit)
- Role-based authorization
- Hash seguro de senha (PasswordHasher)
- CORS configurado somente para domínios válidos
- SQL Injection (EF Core já protege, mas validamos entrada)
- Verificação de diretórios e arquivos sensíveis

Ferramentas:

- OWASP ZAP
- Dotnet Security Analyzer

10.3 Casos de Teste (Exemplos Reais)

CT001 — Login com credenciais válidas

Campo	Valor
Usuário	admin@edu.com
Senha	123456

Resultado esperado:

- Retorna 200
- Recebe token JWT
- Role = admin

- Redireciona para dashboard

CT002 — Login com credenciais inválidas

Resultado esperado:

- Exibir mensagem “usuário ou senha incorretos”
- Não redirecionar
- Notificação visual no front

CT010 — Cadastro de aluno com dados válidos

Resultado esperado:

- Item inserido na tabela
- Tabela atualizada automaticamente
- Mensagem “Aluno cadastrado com sucesso”

CT020 — Consulta de compromissos no calendário

Resultado esperado:

- Exibir cards dos eventos abaixo do calendário
- Notificações verdes indicando compromissos

CT030 — API: Criar evento

POST `/api/v1/eventos`

Resultado esperado:

- Retorna 201
- Objeto criado corretamente
- Persistido no banco

CT040 — API: Rota protegida sem token

Resultado esperado:

- Retorna 401
- “Token inválido ou ausente”

10.4 Ambiente de Testes

Ambientes previstos:

- **local** (desenvolvedor)
- **dev** (teste interno, após back-end criado)
- **test** (opcional, caso escalem o projeto)

Front atual (JS puro): navegadores
Back-end .NET:

- banco SQLite (testes)
- banco Postgres/SQLServer (produção/dev)

10.5 Ferramentas recomendadas

Área	Ferramenta

Tests .NET	xUnit
Mocks	Moq
Front React	Jest + React Testing Library
API Manual	Postman / Swagger
API Automation	Newman
Qualidade	SonarLint / SonarCloud
Segurança	OWASP ZAP

10.6 Critérios de Aceite (QA)

O projeto será considerado estável quando:

- Login funciona perfeitamente
- Dashboard carrega todas as seções
- Formulários validam campos corretamente
- Tabelas filtram sem erros
- Calendário lista e navega corretamente
- API responde com status adequado
- Requisições inválidas retornam erros claros
- Testes automatizados passam no CI

10.7 Estratégia de Testes Futuria (quando React + .NET estiverem completos)

- Testes E2E com Cypress
- Testes de snapshot com Jest
- Testes de API com Postman + Newman
- Testes de carga (k6)
- Testes de segurança (ZAP + OWASP checklist)

---

11. ROADMAP E EVOLUÇÃO DO SISTEMA

11.1 Objetivo do Roadmap

Apresentar uma visão clara e estratégica dos próximos passos no desenvolvimento do **EduConnect**, organizando as entregas por fases, priorizando o que gera mais valor no curto prazo e garantindo que o projeto evolua de maneira consistente, sustentável e demonstrável.

Este roadmap guia o crescimento do MVP atual até um sistema robusto completo, contemplando front-end avançado, back-end, banco de dados, autenticação, relatórios, dashboards e funcionalidades especiais.

11.2 Fases de Evolução do Projeto

🚀 Fase 1 — Front-end Inicial (MVP Atual) — *Concluído*

Entregas

- Dashboard estática (HTML + CSS + JS)
- Calendário com eventos

- Tema claro/escuro
- Cadastro simples via DOM
- Tabelas filtráveis
- Responsividade
- Sistema de notificações
- Melhorias de UX

🎯 **Propósito:** Demonstrar visão, design, organização e capacidade técnica inicial.

🔥 Fase 2 — Documentação Técnica Completa — *Concluído com esta seção*

Entregas:

- Documentação estruturada (modelo profissional)
- Requisitos funcionais e não funcionais
- Modelagem de dados
- Arquitetura do sistema
- Diagramas de casos de uso
- Fluxos de usuário
- Telas e wireframes
- Arquitetura do Back-end
- Plano de testes
- Roadmap final
- Organização do diretório `/docs/`

🎯 **Propósito:** Transformar o projeto em um produto real, com visão completa e clareza técnica.

🔧 Fase 3 — Back-end Inicial (API .NET + Banco de Dados)

Entregas planejadas:

- Estrutura inicial da solução .NET
- Entidades do domínio (Aluno, Professor, Admin, Turma, Evento, Nota etc.)
- Swagger configurado
- Login + JWT + Roles
- CRUDs principais:
    - Alunos
    - Professores
    - Turmas
    - Eventos
    - Notas
- Banco Postgres ou SQL Server
- Repository pattern + EF Core
- Validações + FluentValidation
- Seed inicial de dados
- Middlewares (logs, erros, CORS)

🎯 **Propósito:** Sustentar todas as telas futuras com dados reais.

⚛️ Fase 4 — Migração para Front-end Avançado (React)

Entregas:

- Setup do React + Vite
- Context API para tema e autenticação
- Integração com API (.NET)
- Rotas protegidas por role
- Componentização:
    - Header
    - Footer
    - Sidebar
    - Cards
    - Tabelas dinâmicas
    - Componentes de calendário
- Telas completas com dados reais
- Formulários com React Hook Form + Yup
- Charts com Recharts ou Chart.js

🎯 **Propósito:** Deixar o sistema moderno, escalável e pronto para empresa.

🔐 Fase 5 — Segurança e Autorização Avançada

Entregas:

- Refresh tokens
- Revogação de tokens
- Rate limiting no login
- Política de senhas fortes
- Criptografia com hashing seguro

- Logs estruturados (Serilog)
- Auditoria para admins

🎯 **Propósito:** Aumentar confiabilidade e nível profissional do projeto.

📊 Fase 6 — Dashboards Inteligentes

Entregas:

- Painel do aluno:
    - Média por disciplina
    - Status de presença
    - Próximos eventos
- Painel do professor:
    - Turmas sob responsabilidade
    - Desempenho geral dos alunos
- Painel do admin:
    - Total de alunos
    - Total de professores
    - Ocupação das turmas
    - Indicadores personalizados

🎯 **Propósito:** Mostrar maturidade de análise de dados e visualização.

🧠 Fase 7 — Funcionalidades Diferenciais

Essas funcionalidades foram discutidas anteriormente e estão entre os diferenciais mais fortes:

🔹 **Reconhecimento automático de padrões (Analytics escolar)**

- Identificar queda de desempenho
- Alertar aluno e professor
- Recomendação de estudo

🔹 **Exportação para PDF (boletim, relatórios)**

- Relatório do aluno
- Relatório da turma
- Carta de recomendação automática

🔹 **Sistema de mensagens interno**

- Aluno ↔ Professor
- Notificações instantâneas
- Registro de histórico

🔹 **Modo Offline (PWA básico)**

- Cache de arquivos
- Acesso à Dashboard mesmo sem internet

🔹 **Acessibilidade (WCAG 2.1)**

- Alto contraste
- Navegação via teclado
- Texto ajustável

🎯 **Propósito:** Surpreender o júri e mostrar domínio avançado de produto.

🚀 Fase 8 — Entrega Final + Apresentação

Entregas:

- Pitch profissional (3–6 minutos)
- Slides corporativos (tema TIVIT)
- Demonstração funcional do sistema
- Repositório organizado
- Documentação publicada no README
- Deploy do front + back (Vercel, Render, etc.)

🎯 **Propósito:** Entregar um projeto digno de contratação e destaque.

11.3 Priorização (Backlog Macro)

🥇 Alta Prioridade

- API de autenticação
- CRUDs principais (aluno/professor/turma/eventos)
- Integração React + .NET
- Dashboard funcional
- Tabelas dinâmicas com dados reais

🥈 Média Prioridade

- Sistema de mensagens
- Exportação para PDF
- Painéis separados por tipo de usuário
- Logs e auditoria

🥉 Baixa Prioridade (Diferenciais)

- Analytics escolar (IA simples)
- PWA Offline
- Gamificação
- Acessibilidade avançada

11.4 Cronograma Estimado (Flexível)

Fase	Estimativa
Construção da API .NET	2–3 semanas
Integração React	3–4 semanas
Dashboards completos	1–2 semanas
Funcionalidades avançadas	1–3 semanas
Testes finais + apresentação	1 semana