🚀 LEADS MANAGEMENT SYSTEM - MVP Completo

📋 Visão Geral

Leads Management System é uma aplicação Full Stack profissional para gerenciamento de leads de vendas, desenvolvida com as melhores práticas modernas de engenharia de software.

🎯 Recursos Principais

- ✅ **Interface responsiva em React** - Tabas para "Convidados", "Aceitos" e "Recusados"
- ✅ **REST API robusta em .NET Core** - CQRS com MediatR
- ✅ **Banco de dados SQL Server** - Com migrations automáticas
- ✅ **Lógica de negócio automática** - Desconto de 10% em leads acima de $500
- ✅ **Notificações por email** - Integração simulada
- ✅ **Testes unitários** - 100% cobertura das regras de negócio
- ✅ **Clean Architecture** - Separação clara de responsabilidades
- ✅ **Docker Compose** - Ambiente pronto para produção

🚀 Quick Start (5 minutos)

Opção 1: Docker Compose (Recomendado)

```
# Clonar repositório
git clone &lt;seu-repo&gt;
cd LeadsManagement

# Rodar tudo com Docker
docker-compose up -d

# Aguardar inicialização (20-30 segundos)
sleep 30

# Acessar aplicação
# Frontend: http://localhost:3000
# API Docs: https://localhost:7000/swagger
```

Opção 2: Local (Manual)

```
# Terminal 1 - Backend
cd src/LeadsManagement.API
dotnet run
# API rodando em: https://localhost:7000
```

```
# Terminal 2 - Frontend
cd frontend
npm install
npm run dev
# Frontend rodando em: http://localhost:3000
```

Stack Tecnológico

Backend

- **Framework**: ASP.NET Core 6
- **ORM**: Entity Framework Core 6
- **Database**: SQL Server 2022
- **Padrões**: CQRS + MediatR
- **Testing**: xUnit + Moq + FluentAssertions

Frontend

- **Framework**: React 18
- **Build Tool**: Vite
- **HTTP Client**: Axios
- **UI/Notifications**: React Toastify

DevOps

- **Containerização**: Docker + Docker Compose
- **Versionamento**: Git/GitHub
- **Migrations**: Entity Framework Code First

Estrutura do Projeto

```
LeadsManagement/
├── src/
│   ├── LeadsManagement.API/              # Presentation Layer
│   │   ├── Controllers/
│   │   ├── Middleware/
│   │   ├── Program.cs
│   │   └── appsettings.json
│   │
│   ├── LeadsManagement.Application/      # Application Layer (CQRS)
│   │   ├── Features/Leads/
│   │   │   ├── Commands/
│   │   │   ├── Queries/
│   │   │   └── DTOs/
```

```
│   │   └── Common/
│   │
│   ├── LeadsManagement.Domain/           # Domain Layer
│   │   ├── Entities/Lead.cs
│   │   ├── ValueObjects/
│   │   └── Events/
│   │
│   └── LeadsManagement.Infrastructure/   # Infrastructure Layer
│       ├── Data/
│       ├── Repositories/
│       └── Services/
│
├── tests/
│   └── LeadsManagement.Tests/
│       ├── Domain/
│       ├── Features/
│       └── Integration/
│
├── frontend/
│   ├── src/
│   │   ├── api/
│   │   ├── components/
│   │   ├── hooks/
│   │   └── App.jsx
│   ├── package.json
│   └── vite.config.js
│
├── docker-compose.yml
├── Dockerfile
└── README.md
```

🔄 Fluxo da Aplicação

1. Visualizar Leads

```
Frontend (React)
    ↓ GET /api/v1/leads/status/Invited
Backend (Controller)
    ↓ MediatR
Query Handler
    ↓
Repository
    ↓
SQL Server
    ↓
JSON Response
    ↓
Frontend exibe cards em tabela
```

2. Aceitar Lead

```
Frontend: Usuário clica "Aceitar"
    ↓ POST /api/v1/leads/{id}/accept
Backend: AcceptLeadCommandHandler
    ↓
Domain Logic:
  - Se price &gt; 500: aplica 10% desconto
  - Muda status para "Accepted"
  - Dispara evento LeadAcceptedEvent
    ↓
Repository salva mudanças
    ↓
EmailService envia notificação
    ↓
Frontend recebe resposta 200
    ↓
Toast notification exibida
    ↓
Lista atualizada
```

Funcionalidades Principais

Aba "Convidados" (Invited)

- Exibe todos os leads novo
- Cartões mostram: nome, data, região, categoria, descrição, preço, ID
- Botões: **Aceitar** e **Recusar**
- Ao aceitar: desconto automático se preço > $500

Aba "Aceitos" (Accepted)

- Leads que foram aceitos
- Exibe informações adicionais: telefone, email
- Status visual diferenciado

Aba "Recusados" (Declined)

- Leads que foram recusados
- Informações de auditoria

🧪 Testes

Rodando Testes

```
# Todos os testes
dotnet test

# Com cobertura
dotnet test /p:CollectCoverage=true

# Projeto específico
dotnet test tests/LeadsManagement.Tests/
```

Cobertura de Testes

- **LeadTests.cs** - 10 testes unitários da entidade Lead
- **AcceptLeadCommandHandlerTests.cs** - 3 testes do handler de aceitação
- **DeclineLeadCommandHandlerTests.cs** - 2 testes do handler de recusa
- **GetLeadsByStatusQueryHandlerTests.cs** - 3 testes da query de listagem

Total: 18+ testes validando regras de negócio críticas

Padrão de Testes (AAA)

```
[Fact]
public void Accept_WhenPriceAbove500_ShouldApplyDiscount()
{
    // Arrange: configuração
    var contact = new Contact("Maria");
    var lead = new Lead(contact, "Rio", "Imóvel", "Casa", 1000);

    // Act: execução
    lead.Accept();

    // Assert: verificação
    lead.Price.Amount.Should().Be(900); // 1000 - 10%
}
```

⚙️ Configuração

Variáveis de Ambiente

Backend (appsettings.Development.json)

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=LeadsManagementDb;Trusted_Connection=
```

```
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.EntityFrameworkCore": "Warning"
    }
  }
}
```

Frontend (.env.development)

```
VITE_API_BASE_URL=https://localhost:7000/api/v1
```

Endpoints da API

Leads

Método	Endpoint	Descrição
`GET`	`/leads/status/{status}`	Listar leads por status
`GET`	`/leads/{id}`	Buscar lead específico
`POST`	`/leads`	Criar novo lead
`POST`	`/leads/{id}/accept`	Aceitar lead (aplica desconto se necessário)
`POST`	`/leads/{id}/decline`	Recusar lead

Exemplo de Requisição

```
# Criar lead
curl -X POST https://localhost:7000/api/v1/leads \
  -H "Content-Type: application/json" \
  -d '{
    "contactFirstName": "João",
    "suburb": "São Paulo",
    "category": "Tecnologia",
    "description": "Software SaaS",
    "price": 800
  }'

# Aceitar lead (desconto de 10% aplicado automaticamente)
curl -X POST https://localhost:7000/api/v1/leads/1/accept
```

🏗️ Padrões e Princípios

Clean Architecture

- **Domain Layer**: Lógica de negócio pura, sem dependências
- **Application Layer**: Orquestra casos de uso (CQRS)
- **Infrastructure Layer**: Acesso a dados e serviços externos
- **Presentation Layer**: Controladores e entrada de dados

CQRS (Command Query Responsibility Segregation)

- **Commands**: Modificam estado (AcceptLeadCommand, DeclineLeadCommand)
- **Queries**: Apenas consultam (GetLeadsByStatusQuery, GetLeadByIdQuery)
- Processados por **MediatR** - padrão Mediator

Value Objects

- **Money**: Encapsula lógica de preços (com método ApplyDiscount)
- **Contact**: Encapsula dados de contato
- Imutáveis e autovalidáveis

Repository Pattern

- Abstração do acesso a dados
- Implementação genérica com `RepositoryBase<T>`
- Especialização com `LeadRepository`

Dependency Injection

Registrado em `ServiceCollectionExtensions`:

```
services.AddScoped&lt;LeadRepository&gt;();
services.AddScoped&lt;IEmailService, EmailService&gt;();
services.AddMediatR(cfg =&gt; ...);
```

📧 Emails

Os emails simulados são salvos em arquivos de texto na pasta:

```
src/LeadsManagement.API/emails/
```

Cada email contém:

- Para quem foi enviado

- Assunto
- Corpo em texto plano
- Versão HTML
- Timestamp

Para produção: Substituir `EmailService` por implementação real (SMTP, SendGrid, etc)

🔧 Troubleshooting

Erro: "Connection refused"

```
# Verificar se SQL Server está rodando
# Windows: Verificar Serviços (services.msc)
# Docker: docker-compose ps
```

Erro: "CORS blocked"

```
# Verificar appsettings.Development.json
# Adicionar origin se necessário
```

Frontend não conecta na API

```
# Verificar porta da API (padrão: 7000)
# Verificar VITE_API_BASE_URL no .env
# Checar console do navegador (F12) para erros
```

Certificado HTTPS inválido

```
# Limpar e confiar em certificado dev
dotnet dev-certs https --clean
dotnet dev-certs https --trust
```

📚 Recursos de Aprendizado

Backend

- Documentação MediatR: https://github.com/jbogard/MediatR
- Clean Architecture: https://blog.cleancoder.com/
- Entity Framework Core: https://learn.microsoft.com/pt-br/ef/core/

Frontend

- React: https://pt-br.react.dev/
- Vite: https://vitejs.dev/
- Axios: https://axios-http.com/

Padrões

- CQRS: https://martinfowler.com/bliki/CQRS.html
- DDD: https://www.domainlanguage.com/ddd/

Contribuindo

1. Fazer fork do projeto
2. Criar branch de feature (`git checkout -b feature/AmazingFeature`)
3. Commit mudanças (`git commit -m 'Add some AmazingFeature'`)
4. Push para branch (`git push origin feature/AmazingFeature`)
5. Abrir Pull Request

Licença

Este projeto está sob a licença MIT. Veja `LICENSE.md` para mais detalhes.

Desenvolvedor

Desenvolvido como MVP (Minimum Viable Product) para demonstrar boas práticas de engenharia de software moderno.

Stack Showcase

- ✅ Clean Architecture
- ✅ CQRS + MediatR
- ✅ Domain-Driven Design (conceitos)
- ✅ Repository Pattern
- ✅ Dependency Injection
- ✅ Entity Framework Core
- ✅ React Hooks
- ✅ Testes Unitários
- ✅ Docker/Docker Compose
- ✅ RESTful API

🆘 Suporte

Para dúvidas ou problemas:

1. Verificar esta documentação
2. Checar seção Troubleshooting
3. Examinar logs (terminal ou arquivo de log)
4. Abrir Issue no GitHub

Versão: 1.0.0
Data: Novembro 2025
Status: Production Ready ✅