Plataforma de Educação Online

Título do Projeto

Plataforma de Educação Online

Objetivo

Desenvolver uma plataforma educacional online com múltiplos bounded contexts (BC), aplicando DDD, TDD, CQRS e padrões arquiteturais para gestão eficiente de conteúdos educacionais, alunos e processos financeiros.

Descrição Geral

A plataforma de educação e treinamento é uma aplicação disponibilizada via API para gerir cursos/matriculas/alunos/pagamentos e prover meios para que os alunos realizem os cursos.

O sistema será dividido em duas grandes responsabilidades:

1. Backend (API RESTful): Desenvolvido com ASP.NET Core WebAPI, responsável pelo processamento de dados e lógica de negócios e exposição de todos os endpoints do negócio em uma única API.

2. Bounded Contexts (BC): Cada BC deve possuir as camadas necessárias para implementar as soluções de cada problema de negócio.
    - Os BC's não precisam ser iguais (mesma complexidade, precisar usar CQRS, aplicar mesmos padrões) use conforme seus critérios.
    - TDD para modelagem e ações de negócios
    - Testes de integração para validar os casos de uso
    - Banco de Dados: SQL Server com EF Core para persistência e gerenciamento de dados,
    - Deve haver suporte para SQLite para validação sem dependência de infra.
    - O usuário logado (interativo) deve corresponder a persona do negócio (Aluno, Administrador)

Bounded Context 1: Gestão de Conteúdo

- **Aggregate Roots:**
    - **Curso** (agrega Aulas)
- **Entities e Value Objects:**
    - **Aula** (Entity)
    - **ConteudoProgramatico** (Value Object)
- **Manipulação:** Curso gerencia diretamente suas Aulas e Conteúdo Programático.

Bounded Context 2: Gestão de Alunos

- **Aggregate Roots:**
    - **Aluno** (agrega Matrículas)
- **Entities e Value Objects:**
    - **Matricula** (Entity)
    - **Certificado** (Entity)
    - **HistoricoAprendizado** (Value Object)
- **Manipulação:** Entidade Aluno gerencia diretamente suas Matrículas e Certificados.

Bounded Context 3: Pagamento e Faturamento

- **Aggregate Roots:**
    - **Pagamento**
- **Entities e Value Objects:**
    - **DadosCartao** (Value Object)
    - **StatusPagamento** (Value Object)
- **Manipulação:** Pagamento dispara eventos confirmando ou rejeitando pagamentos.

Modelo de Autenticação e Usuários

- API RESTful protegida com autenticação JWT.
- O usuário logado que executa a ação deve ser representado pela persona do negócio (Aluno, Admin), ou seja deve haver o registro da Persona e o registro do usuário de forma independente, porém compartilhando o mesmo ID.
- Tipos de usuário:
    - **Administrador:** Controle total para cadastrar cursos, aulas, gerir assinaturas, pagamentos, alunos.
    - **Aluno:** Acesso restrito para matrícula em cursos, visualização de aulas/conteúdos e gerenciamento das suas próprias informações e pagamentos.

Quem Executa as Ações

- **Aluno autenticado:** Realiza suas próprias matrículas, consulta cursos/aulas disponíveis, realiza pagamentos e acessa materiais das aulas.
- **Administrador autenticado:** Realiza operações administrativas (criação e manutenção dos cursos, gestão financeira e monitoramento dos alunos).

Casos de Uso Detalhados

Cadastro de Curso

- **Ator:** Administrador
- **Pré-condição:** Autenticado
- **Fluxo Principal:**
    1. Administrador cadastra curso informando nome e conteúdo programático.
    2. Curso validado e salvo.
- **Fluxo Alternativo:** Dados inválidos geram mensagens de erro.
- **Pós-condição:** Curso disponível para matrícula.

Cadastro de Aula

- **Ator:** Administrador
- **Pré-condição:** Curso existente
- **Fluxo Principal:**
    1. Administrador seleciona curso.
    2. Insere título, conteúdo da aula e material (se houver).
    3. Aula validada e vinculada ao curso.
- **Fluxo Alternativo:** Dados inválidos retornam erros.
- **Pós-condição:** Aula associada ao curso.

Matrícula do Aluno

- **Ator:** Aluno
- **Pré-condição:** Aluno autenticado, curso disponível
- **Fluxo Principal:**
    1. Seleciona curso e inicia matrícula.
    2. Matrícula criada com status pendente de pagamento.
- **Fluxo Alternativo:** Erro na matrícula informado.
- **Pós-condição:** Matrícula criada e aguardando pagamento.

Realização do Pagamento

- **Ator:** Aluno
- **Pré-condição:** Matrícula pendente
- **Fluxo Principal:**
    1. Aluno realiza pagamento informando dados do cartão.
    2. Pagamento confirmado altera status da matrícula para ativa.
- **Fluxo Alternativo:** Pagamento rejeitado notifica aluno.
- **Pós-condição:** Pagamento registrado e matrícula ativada.

Realização da Aula

- **Ator:** Aluno
- **Pré-condição:** Matrícula ativa
- **Fluxo Principal:**
    1. Acessa aula e realiza estudo.
    2. Progresso registrado automaticamente.
- **Fluxo Alternativo:** Problema de acesso notifica aluno.
- **Pós-condição:** Aula realizada e progresso registrado.

Finalização do Curso

- **Ator:** Aluno
- **Pré-condição:** Todas as aulas concluídas
- **Fluxo Principal:**
    1. Solicita finalização.
    2. Matrícula atualizada para status concluído e certificado gerado.
- **Fluxo Alternativo:** Aulas incompletas impedem finalização.
- **Pós-condição:** Certificado gerado e disponível.

Requisitos Técnicos

- Linguagem: C#
- Backend: ASP.NET Core WebAPI
- Autenticação: JWT com ASP.NET Core Identity
- Banco de Dados: SQL Server e SQLite com EF Core (o uso de SQLite deve estar sempre configurado com o Seed para que qualquer avaliador do projeto possa executar sem a infra do BD).
- Documentação: Swagger
- Controle de Versão: Github
- Testes:
    - Testes de unidades (via TDD) com cobertura mínima de 80%
    - Testes de integração completos simulando todos processos críticos (casos de uso).

Critérios de Sucesso

- Implementação correta dos bounded contexts e relações.
- Funcionalidades completas e operando dentro dos requisitos
- Código claro e coeso conforme práticas DDD, TDD e CQRS.
- Rodar todos os testes (e passar)
- Cobertura de testes de 80%
- Segurança robusta via autenticação JWT.
- Configuração (**obrigatório**):
    - O projeto deve rodar com a menor configuração de infra possível, para isso utilize a prática ensinada no vídeo a seguir: https://desenvolvedor.io/plus/criando-e-populando-automaticamente-qualquer-banco-de-dados

Prazos e Tipo de Entrega

- **Tipo de Entrega:** Projeto para desenvolvimento **individual**
- **Início do Projeto:** 10/09/2025
- **Primeira entrega (avaliação):** 13/10/2025
- **Segunda entrega (final):** 11/11/2025

Instruções Importantes

Este documento descreve os principais requisitos e funcionalidades esperadas no projeto, oferecendo uma visão geral do domínio e dos objetivos a serem atingidos.

Nem todos os detalhes técnicos e funcionais foram explicitamente descritos.

Espera-se que você compreenda profundamente o problema apresentado e utilize o conhecimento adquirido ao longo dos cursos para implementar as soluções adequadas, tomando decisões técnicas alinhadas com boas práticas.

Os casos de uso especificados devem ser implementados e funcionar conforme descrito, mas você tem autonomia para adotar abordagens técnicas e estratégias adicionais, desde que mantenha o projeto alinhado aos requisitos e objetivos apresentados neste escopo.

Entrega

Repositório no GitHub:

- O código deve ser versionado e entregue através de um repositório público no Github e dentro dos padrões especificados em: https://github.com/desenvolvedor-io/template-repositorio-mba

Documentação:

- O README.md deve seguir as diretivas e padrões informados na documentação do projeto referência.

- Incluir um arquivo FEEDBACK.md no repositório onde os feedbacks serão consolidados, o instrutor fará um PR no repositório atualizando este arquivo.

Matriz de avaliação

Os projetos serão avaliados e receberão uma nota de 0 até 10 considerando os critérios a seguir:

Critério	Peso	Comentários
Funcionalidade	30%	Avalie se o projeto atende a todos os requisitos funcionais definidos.
Qualidade do Código	30%	Considere clareza, organização, uso de padrões de codificação.
Eficiência e Desempenho	10%	Avalie o desempenho e a eficiência das soluções implementadas.
Inovação e Diferenciais	10%	Considere a criatividade e inovação na solução proposta.
Documentação	10%	Verifique a qualidade e completude da documentação, incluindo README.md.
Resolução de Feedbacks	10%	Considere como o aluno ou grupo abordou os feedbacks da revisão de código.