MBA DevXpert Full Stack .NET

[Desenvolvimento Full-Stack Avançado]

Requisitos do Projeto Módulo 02

1. Título do Projeto

Plataforma de Marketplace e Integração com Backoffice

2. Objetivo

Este projeto tem como objetivo dar continuidade ao desenvolvimento da aplicação criada no Módulo 01, evoluindo de uma plataforma de gestão de produtos para um sistema completo de loja virtual.
Nesta nova fase, os alunos deverão implementar uma aplicação de vitrine com foco no cliente final, utilizando uma SPA (Blazor ou Angular), enquanto mantêm o painel administrativo no modelo MVC e integram ambas as soluções via uma API REST centralizada.

3. Aspectos Gerais do Projeto

A nova versão do sistema será composta por três projetos principais, cada um com um papel específico:

- **Administração (MVC):** Esta aplicação será a área administrativa, onde vendedores podem gerenciar seus produtos e o administrador geral pode moderar todos os cadastros do sistema (categorias, produtos e vendedores). Esta aplicação utilizará ASP.NET Core MVC e autenticação via cookies.

- **API REST:** A API será responsável por fornecer todos os recursos para a SPA. Ela deverá expor os dados dos produtos, categorias, vendedores e clientes, possibilitar autenticação via JWT e garantir regras de segurança de forma centralizada. Deve seguir o padrão RESTful e ser documentada via Swagger.

- **Loja (SPA):** Esta aplicação será desenvolvida em Angular ou Blazor, a critério do grupo. Será a interface voltada ao cliente final, permitindo navegação pública pelos produtos cadastrados. Clientes autenticados poderão adicionar produtos à sua lista de favoritos. Esta aplicação se conecta à API via HTTP.

Cada aplicação deve ser independente em termos de front-end, mas consumir os mesmos recursos de back-end (arquitetura compartilhada entre MVC e API).

Fluxo esperado

- **Cadastro e gerenciamento de vendedores (MVC)**
    - O vendedor se cadastra no sistema administrativo e no momento do cadastro do usuário automaticamente é criado um registro na tabela de vendedores. (O registro do Vendedor recebe o mesmo ID do registro do usuário).
    - Após autenticado, o vendedor pode cadastrar produtos vinculados à sua conta, informando nome, descrição, imagem, valor, estoque, status (ativo/inativo) e categoria.
    - O vendedor visualiza, edita ou inativa **apenas os seus próprios produtos**.
    - O vendedor não cria/edita categorias, elas são disponibilizadas pelo Administrador.
    - No backoffice (MVC) não é permitido navegação anônima, qualquer ação requer um usuário logado.
- **Moderação por administrador (MVC)**
    - O perfil "Admin" pode acessar um painel com todos os vendedores e produtos cadastrados no sistema.
    - Pode criar/editar categorias.
    - O admin pode inativar produtos ou vendedores.
    - Um vendedor inativado tem todos os seus produtos ocultados da loja.
    - Não é possível excluir uma categoria que possua algum produto associado.
- **Marketplace e conta de cliente (SPA)**
    - A loja pode ser acessada anonimamente para visualizar produtos ativos.
    - O cliente pode se registrar e logar para criar uma lista de produtos favoritos.
    - O cliente é uma entidade da aplicação e deve possuir um usuário associado a ele. (mesmo padrão utilizado com vendedor).
    - O cliente pode clicar em um produto e ver:

        - Detalhes do produto
        - Informações do vendedor
        - Lista de outros produtos ativos do mesmo vendedor (como em marketplaces)

- **Lista de favoritos (SPA)**
    - Disponível apenas para clientes autenticados.
    - O cliente pode adicionar ou remover produtos da lista de favoritos.
    - Os favoritos são persistidos no banco de dados, associados ao Cliente.
    - Produtos inativos não aparecem na lista.
- **Regras gerais de segurança e integridade**
    - Somente usuários autenticados via JWT podem realizar ações de escrita na API.
    - Produtos e vendedores inativos devem ser ocultados da vitrine (SPA).
    - Produtos devem ter validações para nome, preço positivo, estoque não negativo, e imagem obrigatória.
    - Todo produto deve pertencer a uma categoria.
    - O cadastro de usuários dos clientes e vendedores devem utilizar a mesma base do Identity
    - A separação de Admin, Vendedor e Cliente no Identity devem ser feitos através de Claims.
- **Separação de Responsabilidades entre a Aplicação MVC e a API:**
    - **A aplicação MVC não deve consumir a API**, ambas devem oferecer o mesmo mecanismo de funcionalidades (arquitetura compartilhada).
- **Definição Estrutural do Modelo:**
    - A aplicação deve possuir ao menos três entidades principais: Vendedor, Categoria, Produto e Cliente.

- **Associação de Produtos a Venderores:**
    - **Cada Produto deve estar associado a um único Vendedor**, e esta associação deve ser persistida no banco de dados. O sistema deve garantir que apenas o Vendedor proprietário pode consultar, visualizar e modificar dados do Produto.
    - O que define a propriedade do produto ao vendedor é o ID dele no momento do cadastro (ou seja o do usuário logado).
    - Admins podem apenas ativar ou inativar o produto (ou o vendedor)
- **Categorias são "Livres"**
    - **As categorias não estão restritas a um único vendedor**, todos podem utilizar as categorias disponíveis, mas apenas o perfil administrador pode criar e modificar.
- **Identidade de Vendedor/Cliente como Usuário Registrado:**
    - O sistema deve garantir que todos os Vendedores e Clientes sejam **usuários registrados**, utilizando o ASP.NET Core Identity.
- **Gerenciamento de Usuários:**
    - O sistema deve utilizar as funcionalidades padrão do ASP.NET Core Identity para o registro e autenticação de usuários.
    - **O user do Identity deve co-existir com a entidade Vendedor e Cliente**, ambos compartilham o mesmo ID e devem ser criados no mesmo processo de "Register" do usuário.
    - Não herde ou acople o AspNetUser a entidade de negócio
- **Validação e Tratamento de Erros:**
    - Deve haver uma validação robusta em todas as operações de CRUD para garantir que os dados inseridos sejam válidos e seguros.
      O sistema deve fornecer feedback de erro claro e informativo, tanto nas aplicações MVC/SPA quanto na API.
- **Recursos Extras**
    - **Recomendo que você concentre seus esforços em entregar com excelência** os requisitos obrigatórios já definidos.
    - Se desejar acrescentar algo a mais, mantenha o projeto **simples e funcional**. Um bom investimento de tempo seria **melhorar a experiência do usuário (UX)**.

4. Requisitos Funcionais

- Cadastro e Login de Vendedor (MVC com Identity)
- Cadastro e Login de Cliente (SPA com Identity JWT via API)
- CRUD de Categorias, Produtos, Vendedores (MVC)
- CRUD de Cliente (SPA + API)
- Visualização pública de produtos na SPA (anônimo)
- Visualização de detalhes e produtos por vendedor (SPA)
- Lista de favoritos do cliente (SPA + API)
- Painel de moderação (Admin)

5. Requisitos Técnicos

- **Linguagem de Programação: C# .NET 8 ou superior**
- **Frameworks:**
    - ASP.NET Core MVC
    - ASP.NET Core WebAPI
    - Angular ou Blazor (SPA)
- **Acesso a Dados:**
    - SQL Server/SQLite – O projeto deve ser compatível com SQL Server, **porém deve ser entregue configurado para SQLite em modo "Development"** para que seja executado sem dependências de infra. Use os environments do ASPNET.
    - EF Core para mapear o BD e realizar operações de CRUD
    - É obrigatório inicializar o banco de dados no start da aplicação utilize o mecanismo de seed de dados (veja mais na sessão "Critérios de Sucesso")
- **Autenticação e Autorização:**
    - Implementar autenticação usando ASP.NET Core Identity.
    - **Associar o ID do AspNetUser com o Vendedor/Cliente** (não é necessário relacionamento) basta criá-los no mesmo momento e usar o ID do AspNetUser no ID do Vendedor/Cliente.

6. Casos de Uso

Caso de Uso 1: Cadastro de Vendedor e Gerenciamento de Produtos (MVC)

Ator: Vendedor
Descrição: Um usuário acessa a aplicação de administração (MVC) e realiza seu cadastro como vendedor. Após login, ele pode cadastrar, editar, visualizar e inativar seus próprios produtos, informando nome, descrição, imagem, preço, estoque e categoria.

Caso de Uso 2: Moderação de Vendedores e Produtos (Admin - MVC)

Ator: Administrador
Descrição: O usuário administrador acessa o painel administrativo e pode visualizar a lista completa de vendedores e produtos. Ele pode inativar vendedores ou produtos. Produtos inativados deixam de aparecer na vitrine. Vendedores inativados têm todos os seus produtos ocultados.

Caso de Uso 3: Acesso Público à Loja (SPA)

Ator: Usuário anônimo
Descrição: Um visitante acessa a loja virtual (SPA) sem necessidade de autenticação. Pode navegar pelo catálogo de produtos ativos, filtrá-los por categoria e visualizar detalhes de um produto, incluindo nome, descrição, imagem, preço e dados do vendedor. Também pode acessar a vitrine com todos os produtos ativos daquele vendedor.

Caso de Uso 4: Cadastro e Login de Cliente (SPA)

Ator: Cliente
Descrição: O usuário acessa a SPA e realiza seu cadastro como cliente. Após o login, ele recebe um token JWT que permite interações autenticadas com a API, como favoritar produtos.

Caso de Uso 5: Favoritar e Desfavoritar Produtos (SPA)

Ator: Cliente
Descrição: O cliente autenticado pode adicionar ou remover produtos da sua lista de favoritos. Essa lista é persistida no banco de dados e está vinculada ao cliente. Produtos inativos não podem ser favoritados nem aparecem na lista.
Pode acessar uma página com todos os seus produtos favoritados. Ele pode clicar em qualquer um deles para ver os detalhes completos e os produtos do vendedor correspondente.

7. Critérios de Sucesso

Funcionalidades Completas:

- Requisitos funcionando corretamente e sem erros ou falhas

Qualidade do Código:

- Código claro e bem documentado, aderindo às práticas ensinadas nos cursos.

Segurança:

- Implementação correta de autenticação e autorização.
- Proteção da API com autenticação JWT.

Documentação:

- Incluir um arquivo README.md detalhado no repositório GitHub, dentro dos padrões indicados e com instruções de instalação, configuração e uso se necessário.
- Documentar a API usando Swagger com **suporte a autenticação JWT**.

Configuração:

- **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
- **Caso não seja possível rodar a aplicação por alguma dependência de infra ou setup necessário o projeto não será avaliado**

8. Proposta de Arquitetura

A arquitetura do projeto não deve exceder a complexidade do negócio. Arquiteturas complexas ou simples demais terão penalidades na avaliação.

Proposta Sugerida:

```
solution
|
├── Aplicacoes
|   ├── MVC         → Projeto de interface administrativa (ASP.NET Core MVC)
|   ├── API         → Projeto de backend (ASP.NET Core Web API)
|   └── Blazor      → Projeto SPA (caso Blazor seja o framework escolhido)
|
├── Core
|   ├── Data        → Projeto de persistência
|   |   ├── Context
|   |   ├── Repositorios
|   |   ├── Migrations
|   |   ├── Mappings
|   |   └── Outros utilitários relacionados a dados
|   |
|   └── Business    → Projeto de regras de negócio
|       ├── Models
|       ├── Interfaces
|       ├── Services
|       └── Validations
```

Diretório de projetos:

```
src
|
├── BackEnd
|   ├── MVC         → Projeto de administração
|   ├── API         → API REST
|   ├── Blazor      → SPA (caso Blazor seja escolhido)
|   ├── Data        → Persistência de dados
|   └── Business    → Regras de negócio
|
└── FrontEnd
    └── Angular     → SPA (caso Angular seja escolhido)
```

9. Prazos

- **Definição dos Grupos:** 16/06/2025
- **Início do Desenvolvimento:** 17/06/2025
- **Primeira entrega (avaliação):** 11/07/2025 (24 dias)
- **Segunda entrega (final):** 11/08/2025 (31 dias)
- **Apresentação da avaliação final e notas:** 18/08/2025

10. Entrega

- **Repositório no GitHub:**
    - O código deve ser versionado e entregue através de um repositório público no Github seguindo o padrão: 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.
- **Apresentação:**
    - O grupo deverá gravar e enviar um vídeo completo apresentando o projeto.
    - O vídeo deve abordar desde a proposta inicial, motivações e decisões arquiteturais adotadas, até a explicação da implementação de cada um dos projetos (MVC, API e SPA), finalizando com uma demonstração prática da aplicação em funcionamento, cobrindo todos os casos de uso previstos.
    - Instruções detalhadas sobre como entregar esse vídeo e os pontos obrigatórios a serem abordados serão enviadas em data próxima à apresentação final.

11. 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	20%	Considere clareza, organização, uso de padrões de codificação.
Eficiência e Desempenho	20%	Avalie o desempenho e a eficiência das soluções implementadas.
Inovação e Diferenciais	10%	Considere a criatividade e inovação na solução proposta.
Documentação e Organização	10%	Verifique a qualidade e completude da documentação, incluindo README.md.
Resolução de Feedbacks	10%	Avalie a resolução dos problemas apontados na primeira avaliação de feedback.