**📌 Projeto E-commerce de Roupas – Escopo Detalhado**

**1️⃣ Foco do E-commerce**

* **Produtos: Roupas, com variações de tamanho e cor.**

**2️⃣ Funcionalidades principais**

* **Cadastro de produtos (Admin).**
* **Carrinho de compras (Cliente).**
* **Finalização de pedidos com simulação de pagamento (status “Pago” ou “Falha”).**
* **Histórico de pedidos para clientes.**
* **Login e cadastro de usuários (Cliente/Admin).**
* **Página inicial com listagem de produtos.**
* **Filtragem de produtos por categoria, tamanho e cor.**

**3️⃣ Funcionalidades extras (opcionais)**

* **Reviews e avaliações de produtos.**
* **Painel administrativo com gráficos de vendas (simples).**
* **Integração com e-mail para confirmação de pedido (pode ser futuramente).**

**4️⃣ Modelagem e entidades**

**Entidades principais**

* **Produto → Id, Nome, Descrição, Preço, Estoque, CategoriaId, Tamanho, Cor, ImagemUrl**
* **Categoria → Id, Nome, Descrição**
* **Usuário → Id, Nome, Email, SenhaHash (SQL manual) ou PasswordHash (Identity), Perfil/Roles (Cliente/Admin)**
* **Pedido → Id, UsuarioId, Data, Status, Total**
* **ItemPedido → Id, PedidoId, ProdutoId, Quantidade, PreçoUnitário**
* **Carrinho → Id, UsuarioId, Lista de Itens (ProdutoId + Quantidade)**
* **Review → Id, ProdutoId, UsuarioId, Nota, Comentário, Data**

**Perfis/roles de usuário**

* **Cliente: cadastrado via formulário público.**
* **Admin: criado manual ou via seed, nunca pelo formulário público.**
* **Diferenciação de acesso é feita verificando o perfil/role após login.**

**5️⃣ Fluxo de compra**

1. **Usuário navega e filtra produtos.**
2. **Adiciona produtos ao carrinho.**
3. **Finaliza pedido → escolhe pagamento fictício (cartão, boleto, Pix).**
4. **Sistema valida dados simulados e cria o pedido com status “Pago” ou “Falha”.**
5. **Histórico de pedidos mostra todos os pedidos do usuário.**

**6️⃣ Autenticação e login**

* **Cadastro:**
  + **Clientes via formulário público, validação de email único, senha com hash.**
  + **Admin via script/seed inicial, perfil “Admin”.**
* **Login:**
  + **Único fluxo para Cliente e Admin.**
  + **Após login, sistema verifica perfil/role para liberar acesso:**
    - **Cliente: páginas de compra, histórico, reviews.**
    - **Admin: painel administrativo, cadastro/edição de produtos, relatórios.**
* **Logout: limpa sessão/cookie e redireciona para página pública.**
* **Proteção de rotas:**
  + **SQL manual → verificação de Perfil no código.**
  + **EF + Identity → [Authorize(Roles="Admin")] ou [Authorize(Roles="Cliente")].**
* **Sessão/cookies: expiração configurável, cookies HttpOnly e Secure.**
* **Regras de senha: mínimo 8 caracteres, obrigatórios maiúsculas, minúsculas e números.**
* **Situações especiais:**
  + **Usuário duplicado: impedido no cadastro.**
  + **Alteração de senha: pode ser implementada posteriormente.**
  + **Sessão expirada: redirecionar para login mantendo carrinho.**
  + **Tentativa de acesso não autorizado: redirecionar ou exibir mensagem de erro.**

**7️⃣ Arquitetura e boas práticas**

* **Camadas: Controllers → Services → Repositories → Models (versão EF).**
* **DTOs + AutoMapper aplicados na versão EF.**
* **Versão “na unha”: estrutura mais direta no início, podendo refatorar depois para Repository Pattern.**
* **Testes unitários: deixados para depois.**

**8️⃣ Infraestrutura e organização**

* **Banco: local inicialmente, depois migrar para Docker ou cloud gratuita (Railway/Supabase) para mostrar projeto online.**
* **Versão “na unha”: scripts SQL organizados em arquivos, mantendo histórico e versionamento.**
* **Versão EF: Migrations para criar/atualizar banco.**
* **Repositórios separados no GitHub → melhor para portfólio.**

**9️⃣ Front-end e APIs**

* **Front-end: Razor + Bootstrap (sem frameworks JS externos).**
* **APIs externas: não serão usadas inicialmente.**
* **Swagger: opcional, pode ser incluído para documentação dos endpoints internos, útil para mostrar back-end funcionando no portfólio.**

**🔹 Projeto 1 → “na unha” (sem EF, sem Identity)**

Esse é o que você quer para treinar bem a **base**.

* Aqui faz sentido usar o **jeito clássico**: JsonResult no back e window.open(...) no front.
* Motivo:
  + É mais simples, menos código de JS.
  + Te dá a chance de revisar bem a lógica do fluxo MVC tradicional.
  + Combina com a ideia de fazer as coisas “no braço”, entendendo como tudo se conecta.

✅ Nesse projeto, **recarregar a página inteira não é um problema**, porque o foco é exercitar CRUD, sessões, login manual etc.

**🔹 Projeto 2 → com EF e Identity**

Esse já tem uma pegada mais moderna/profissional.

* Aqui vale a pena usar o **Ajax sem reload** (com Ok(), BadRequest() e manipulação de DOM).
* Motivo:
  + Fica mais próximo do que o mercado usa hoje.
  + Dá a experiência de trabalhar no estilo “API + front dinâmico”.
  + Você já vai praticar o padrão REST (com Ok(), NotFound(), etc.).

✅ Nesse projeto, **vale investir em uma UX melhor**, sem recarregar a página inteira a cada ação.