**Prova de Conceito - API de Cadastro de Clientes em ASP.NET MVC**

**1. Objetivo**

Demonstrar a viabilidade do sistema de cadastro de clientes e logradouros, com foco no desenvolvimento do front-end em **ASP.NET MVC** e no back-end com **API** em **C#**, usando o **.NET Core 6.0** ou superior, além de banco de dados **SQL Server 2016** ou superior. A API usará um **ORM** para mapeamento objeto-relacional.

**2. Escopo**

* **Front-end**: Interface de usuário desenvolvida com **ASP.NET MVC**, usando **Razor**, **JavaScript** e **HTML**, conforme necessário.
* **Back-end**: API para realizar operações CRUD (Create, Read, Update, Delete) com clientes, incluindo upload de logotipos, relacionamento com logradouros e controle de usuários.
* **Banco de Dados**: SQL Server com tabelas para Cliente, Logradouro, Usuario. Operações de consultas serão feitas usando **ORM**, enquanto operações mais complexas ou de escrita usarão **Stored Procedures**.

**3. Componentes a Serem Desenvolvidos**

**3.1. Front-End (ASP.NET MVC com Razor)**

* **Views (Razor)**: Desenvolver views para as telas de cadastro de clientes, edição e listagem. Usar Razor para renderizar dados do cliente e incluir campos para upload de logotipos.
* **JavaScript**: Para controlar o comportamento dinâmico, como validação de formulários e pré-visualização de imagens antes do upload.
* **Controller (MVC)**: Controladores para interagir com o modelo de cliente, chamando a API REST para operações CRUD.

**3.2. Back-End (API em C# com .NET Core 6.0)**

* **Controllers**:
  + ClienteController para gerenciar clientes.
  + LogradouroController para gerenciar logradouros.
  + Implementar a autenticação via **JWT**.
* **Serviços (Services)**:
  + Serviços para lidar com a lógica de negócio de clientes e logradouros.
  + **LogotipoService** para gerenciar o upload de imagens.
* **ORM e Stored Procedures**:
  + Usar um ORM como **Entity Framework Core** para consultas simples.
  + Implementar stored procedures para operações mais complexas ou relacionadas a performance (como inserções de dados em massa ou operações complexas com múltiplas tabelas).

**3.3. Banco de Dados**

* **Modelo de Dados**:
  + Tabelas para Cliente, Logradouro, Usuario com os relacionamentos necessários.
  + Implementar **índices** e **chaves estrangeiras** para otimizar as consultas.
* **Stored Procedures**:
  + Criar stored procedures para operações específicas (como inserções e remoções de clientes e logradouros) e usar o ORM para consultas de leitura.

**4. Plano de Implementação**

**4.1. Criação do Front-End**

1. **Configurar o projeto ASP.NET MVC** com suporte a Razor.
2. Criar **views para cadastro e listagem de clientes**, incluindo um campo para upload de logotipos (pode ser feito com um <input type="file">).
3. Usar **JavaScript** para pré-visualizar o logotipo antes do envio.
4. **Controller MVC**:
   * Implementar as chamadas à API REST para operações de CRUD.
   * Exibir mensagens de sucesso ou erro.

**4.2. Implementação da API REST**

1. **Criar os Controllers** para Cliente, Logradouro e Usuario.
2. Implementar as **operações CRUD** nos controllers.
3. **JWT**:
   * Adicionar autenticação via JWT para garantir que apenas usuários autenticados possam acessar as funcionalidades da API.
4. Implementar **AutoMapper** para mapear os DTOs (Data Transfer Objects) de cliente e logradouro com as entidades.

**4.3. ORM e Stored Procedures**

1. **ORM (Entity Framework Core)**:
   * Usar para consultas simples, como GET e listagens.
2. **Stored Procedures**:
   * Usar para operações complexas de escrita no banco, como inserções e atualizações em massa.
   * Incluir as chamadas para stored procedures no serviço responsável.

**5. Plano de Testes**

1. **Testes no Front-End**:
   * Testar a interação de usuários no front-end, especialmente o upload de logotipos e operações de CRUD.
   * Verificar a comunicação entre o front-end e a API.
2. **Testes na API**:
   * Testar todas as operações CRUD, incluindo a recuperação de clientes e logradouros com e sem logotipos.
   * Testar a autenticação e autorização com JWT.
3. **Testes de Performance**:
   * Verificar a eficiência das consultas usando o ORM.
   * Avaliar a performance das stored procedures em cenários de alta carga.