Teste Prático para Desenvolvimento Web

Objetivo

O objetivo do teste é avaliar a forma como o candidato evolui a solução para o problema proposto, mais do que se está 100% correta a implementação, será avaliado de forma de tentar resolver o problema. O candidato deve entregar o teste contemplando os itens obrigatórios, caso contrário o teste será anulado.

O que é obrigatório no teste?

- Utilizar o GIT para controle de histórico de alterações na solução;
- Uso de um banco de dados;
- Disponibilizar um markdown (ex: README.md) na raiz da solução com as instruções para compilar, configurar banco e executar o projeto;
- Criar uma solução com dois projetos um frontend e outro backend conforme descrito no **Problema** apresentado mais abaixo;

Desejável

- API desenvolvida em .NETCORE utilizando a linguagem Csharp;
- Usar SQL Server;

O que não é obrigatório mas conta pontos

- Uso de conceitos de OOP e relacionados;
- Separação em camadas (aplicação, negócio e acesso a dados);
- Aplicação frontend utilizando ReactJS;
- Utilizar Procedures e Funções;

Problema / Solução

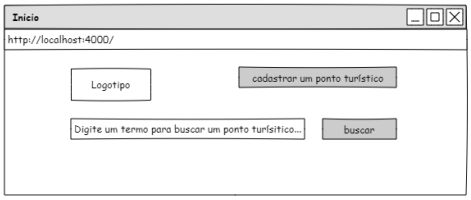
Deve ser criado uma aplicação para o cadastro e listagem de pontos turísticos do país, cada ponto turístico deve ter o nome, descrição até 100 caracteres e localização (endereço ou referência de localização), cidade e estado).

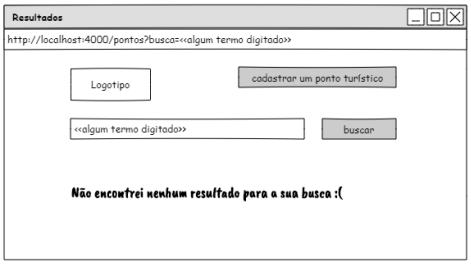
Deve ser possível na página inicial listar de forma **paginada** os cadastros ordenados de forma descrente pela data de inclusão e **permitir buscar/filtrar** digitando um termo de busca analisando tanto nome e descritivo quanto localização dos pontos turísticos cadastrados. A página deve listar os pontos com o nome e localização. Ao ser selecionado o ponto turístico deve ser demonstrada o nome, descrição e localização.

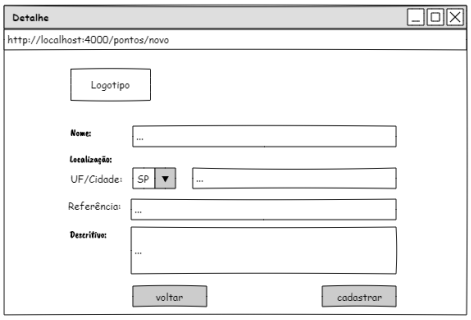
No cadastro, os estados devem ser listados como combo/dropdown e a cidade ou pode ser texto livre, ou buscar dealgum webservice online disponível na internet as cidades de acordo com o estado selecionado.

Criar um menu de navegação para alternar entre o formulário de cadastro e a listagem.

Design sugerido das telas a seguir







Arquitetura da Solução

Criar uma API REST para o cadastro e consulta dos pontos turísticos servindo de backend e uma Aplicação Web servindo de frontend para consumir a API e servir o navegador.

Modelo de Arquitetura 1 - Sugestão

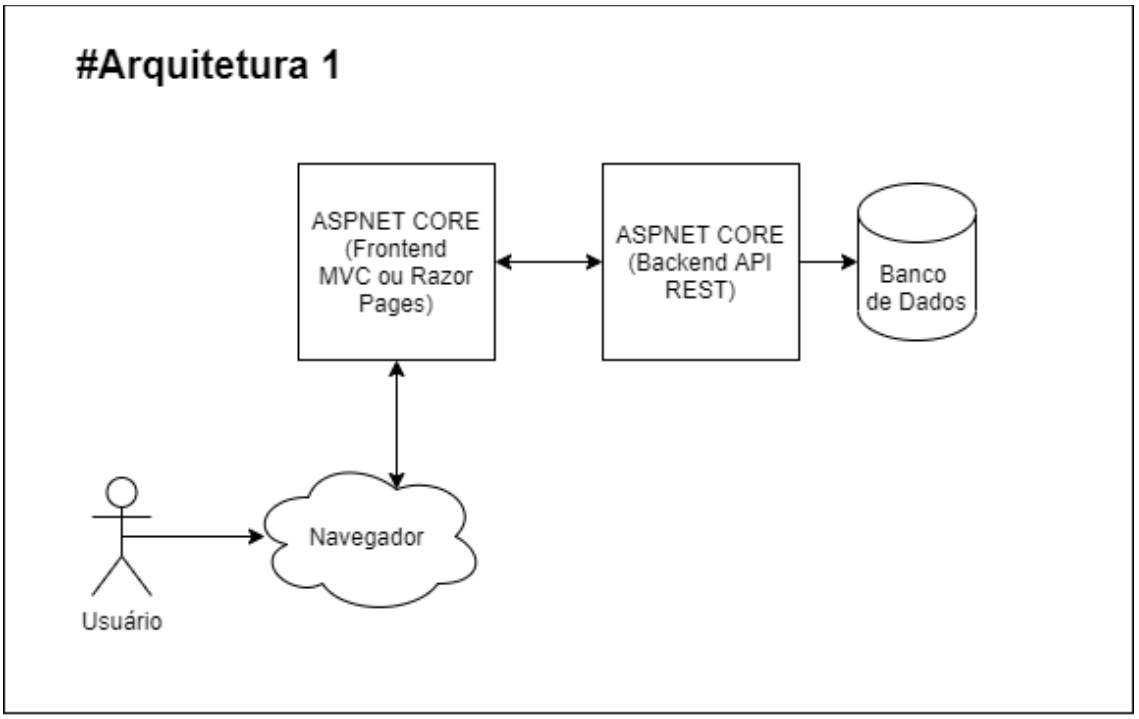
- Construa uma API REST em ASPNET Core contemplando a solução ao problema proposto.
- Construa uma Aplicação Web em ASPNET Core servindo de frontend e contemplando a solução ao problema, consumindo a API, onde:
  - Aplicação ASPNET Core MVC ou RazorPages serve as páginas conforme descrito no problema e consome os dados e operações da API REST a partir do Controllers ou Pages Csharp;

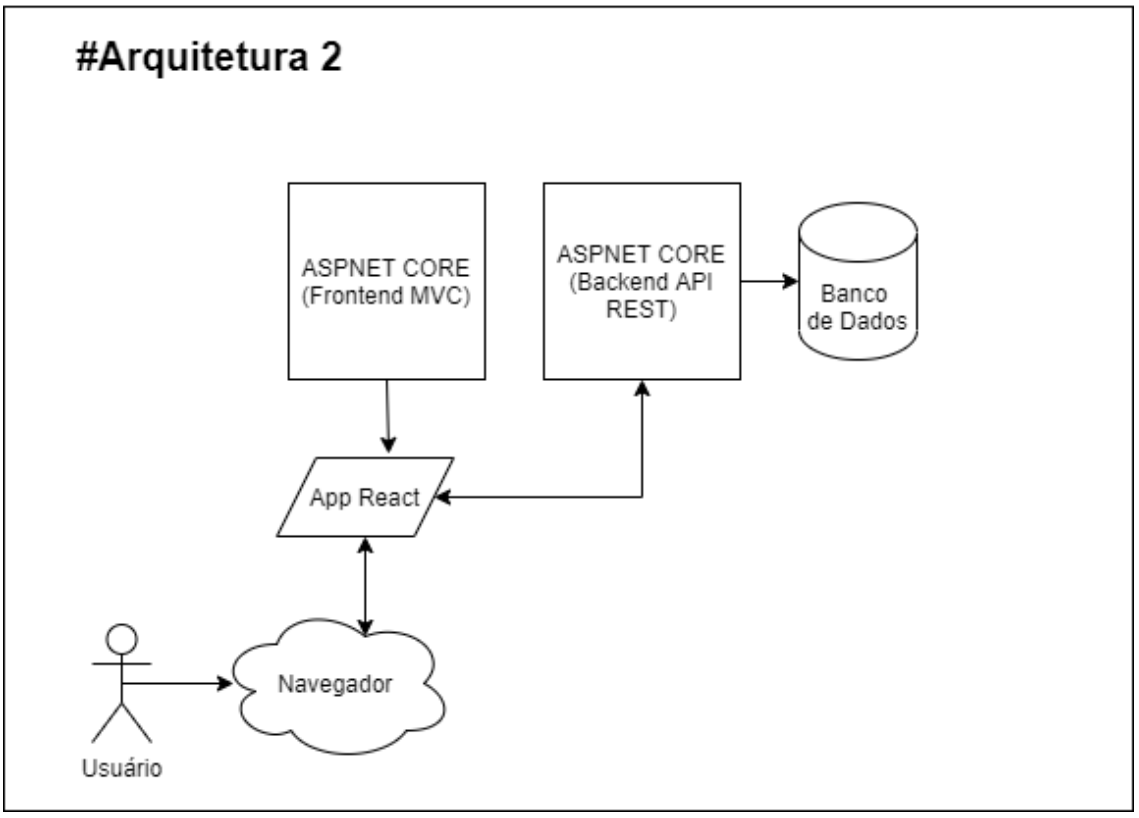
Modelo de Arquitetura 2 - Sugestão

- Construa uma API REST em ASPNET Core contemplando a solução ao problema proposto.
- Construa uma Aplicação Web em ASPNET Core servindo de frontend e contemplando a solução ao problema, consumindo a API, onde:
  - Aplicação ASPNET Core MVC serve um app ReactJS consumindo os dados e operações da API REST apartir do navegador via Javascript;

Entregar o backend e frontend em suas respectivas pastas contendo as instruções para configurar, compilar e executar cada um.

Diagrama:


#Arquitetura 1
ASPNET CORE (Frontend MVC ou Razor Pages)
ASPNET CORE (Backend API REST)
Banco de Dados
Navegador
Usuário


#Arquitetura 2
ASPNET CORE (Frontend MVC)
ASPNET CORE (Backend API REST)
Banco de Dados
App React
Navegador
Usuário