**ExpressYourSelf (REST API)**

Neste documento, tentarei sintetizar a proposta do projeto **ExpressYourSelf**, além de documentar todas as decisões e abordagens adotadas por mim para atingir seus objetivos.

**1. Objetivo do Projeto**

O objetivo é criar uma **REST API** que segue os seguintes critérios:

1. Deve ser feita usando a **última versão do ecossistema .NET Core**.
2. Todas as interações com o banco de dados devem ser feitas com o uso do **Entity Framework**, exceto na resolução da terceira tarefa.
3. O banco de dados utilizado deve ser **Sequencial (SQL)**. O esquema e a população estarão disponíveis, mas o uso dos mesmos é facultativo.

Esses são os objetivos e as restrições gerais. Agora vamos detalhar as funcionalidades da API.

**2. Descrição das Tarefas**

A aplicação está dividida em **Tasks**, que podem ser entendidas como requisitos principais.

**2.1 Função Principal**

* **Descrição:** Um endpoint que recebe um **IP** como entrada e retorna um conjunto de dados estruturados:
  + CountryName
  + TwoLetterCode
  + ThreeLetterCode
* **Ordem de Busca dos Dados:**
  + Primeiro, na **memória cache**.
  + Depois, no **banco de dados** (caso não esteja no cache).
  + Por fim, na **API pública externa IP2C** (caso não esteja no banco).
* **Ações Adicionais:**
  + Dados adquiridos pela API externa devem ser armazenados no banco.
  + Consultas devem ser armazenadas no cache para melhorar performance.

**2.2 Atualização de Dados**

* **Descrição:** Automação que verifica periodicamente a veracidade dos dados no banco e os atualiza, se necessário.
* **Restrições:**
  + Frequência: A cada **1 hora**.
  + Verificar todos os IPs do banco, mas em lotes de 100 por vez.
  + Referência: A **API pública externa IP2C**, assumida como confiável.
* **Ações Adicionais:**
  + Dados inconsistentes devem ser atualizados no banco e invalidados no cache.

**2.3 Report Endpoint**

* **Descrição:** Um endpoint que fornece relatórios sobre o banco de dados.
* **Requisitos:**
  + **Entrada:** Um array de TwoLetterCountryCode. Um valor null retorna dados completos.
  + **Saída:** Um array de objetos estruturados com:
    - CountryName (string)
    - AddressesCount (int)
    - LastAddressUpdate (DateTime)
  + Não deve usar o **Entity Framework**.

**3. Decisões de Contexto**

**3.1 Função Principal**

* **Adição do IP na Resposta:** Por padrão, a aplicação vai sempre retornar um JSON com três valores: Nome do País, Código de duas Letras e Código de três letras. Porém eu decidi colocar a opção de retornar um JSON com 4 valores, os 3 anteriores mais o próprio IP, pois:
  1. Isso melhora a transparência para o usuário.
  2. O impacto no custo e na performance é insignificante.

Para tornar isso uma opção, implementarei um endpoint (toggleip) que sempre que acessado liga ou desliga a opção de receber o IP junto com os dados estruturados.

* **Framework de Cache:** Escolhi o **IMemoryCache** por ser nativo do .NET e fácil de implementar.
* **TTL:** Defini o TTL como **1 hora**, compatível com o intervalo de atualização da automação e a finalidade da API.
* **Reset de TTL:** Quando um dado é acessado no cache, seu TTL será resetado automaticamente.
* **Formato da Resposta da IP2C:** A resposta é em **CSV**, o que exige tratamento especial nos dados. Além disso, a API não permite requisições personalizadas, o que poderia otimizar custo e performance.
* **Banco de Dados:** Decidi usar os comandos SQL fornecidos pelo próprio documento, pois, tendo em vista o alinhamento com os requisitos, a escalabilidade, e praticidade, eles atendem perfeitamente as necessidades dessa Task e das Demais. Porém, a sua sintaxe terá de ser formatada, pois a ferramenta utilizada será o Postgree SQL.

**3.2 Atualização de Dados**

* **Automação:** Usarei o **HostedService** do ASP.NET pela simplicidade e eficiência.
* **Premissas:**
  1. Os dados CountryName, TwoLetterCode e ThreeLetterCode têm uma relação intrínseca e devem ser consistentes.
  2. A API externa **IP2C** será considerada confiável.
* **Critério de Comparação:** Apenas os campos CountryName e TwoLetterCode serão comparados. Caso haja diferenças, o objeto será atualizado no banco.
* **Tratamento no Cache:** Dados atualizados no banco também serão atualizados no cache, em vez de removidos, para evitar custos extras.

**3.3 Report Endpoint**

* **Estrutura do Banco:** O endpoint exige múltiplas tabelas relacionadas, incluindo um atributo LastAddressUpdate no schema.
* **Entrada e Saída:** Apenas null deve retornar todos os dados, e os tipos de entrada e saída não são flexíveis.
* **Abordagem Técnica:** Usarei apenas **SQL bruto**, já que o uso do Entity Framework é restrito nesta tarefa.

**4. Ferramentas, Frameworks e Técnicas**

* **Ecossistema de Desenvolvimento:** .NET Core **9.0.101** (versão mais atual no momento do desenvolvimento).
* **Abordagem:** Minimal API.
* **Banco de Dados:** PostgreSQL **15.4**.
* **Cache:** IMemoryCache.
* **Automação:** HostedService.
* **Versionamento:** Git e GitHub.
* **Teste de Requisição:** Postman.
* **Metodologia:** Scrum.

**Nota:** Para mais detalhes sobre as Sprints e classes desenvolvidas para compor o sistema e atender a todos os requisitos, veja os documentos Sprints e Classes.