**Modelagem das Classes**

As classes serão divididas em dois tipos: classes Entidade, que representam os dados em transição e manipulados, e as classes administrativas, que fazem a manipulação e gerenciam a aplicação. As classes administrativas devem ser preferencialmente estáticas.

**1. Classes Entidade**

**1.1. Classe IP**

* **Descrição:** Representa o objeto principal da aplicação, relacionado a endereços IP.
* **Atributos:**
  + IpNumber: String: Identificador único do IP.
  + Country: Country: Referência ao país relacionado a este IP.
  + LastUpdate: Date: Data da última atualização deste IP.
* **Funções:**
  + **Construtora:** Instancia a classe com os atributos definidos.
  + **Getters e Setters:** Métodos de acesso e modificação dos atributos.

**1.2. Classe Country**

* **Descrição:** Representa um país e suas características únicas.
* **Atributos:**
  + Id: int: Identificador único.
  + CountryName: String: Nome completo do país.
  + TwoLCode: String: Código ISO de duas letras.
  + ThreeLCode: String: Código ISO de três letras.
* **Funções:**
  + **Construtora:** Instancia a classe com os atributos definidos.
  + **Getters e Setters:** Métodos de acesso e modificação dos atributos.
  + **Equals:** Recebe outro objeto Country e compara seus atributos (nome e códigos ISO), retornando um booleano.

**1.3. Classe Report**

* **Descrição:** Criada para atender ao endpoint de relatórios, representando um conjunto de dados sobre países e IPs associados.
* **Atributos:**
  + Country: Country: País relacionado ao relatório.
  + AddressCount: int: Número de endereços IP associados ao país.
  + LastUpdate: Date: Data da última atualização do país no banco de dados.
* **Funções:**
  + **Construtora:** Instancia a classe com os atributos definidos.
  + **Getters e Setters:** Métodos de acesso e modificação dos atributos.

**Nota:** Uma das funções do framework EntityFramework é criar classes entidade baseadas nas tabelas do banco de dados. Por esse motivo as classes IP e Country serão criadas automaticamente. Sua edição é possível, mas deve ser evitada, porém como os documentos gerados são de classe Partial é possível realizar edições Partial com o uso de outro documento script. Isso será necessário na classe Country para a inclusão do método Equals. A classe Report não possui tabela associada, por isso, ela será criada manualmente.

**2. Classes Administrativas**

Todas as classes administrativas serão estáticas, conforme avaliado. Por esse motivo, é desejável que elas possuam nomes em siglas mínimas, para fácil acesso. Porém é extremamente importante que os códigos dessas classes sejam comentados a fim de explicitar o seu significado.

**2.1. Classe CM (Cache Memory)**

* **Descrição:** Centraliza todas as operações relacionadas à memória cache.
* **Atributos:**
  + **Nenhum:** Não armazena estado interno; utiliza o framework IMemoryCache.
* **Funções:**
  + **HasKey:** Verifica se uma chave existe no cache.
  + **GetData:** Retorna o valor associado a uma chave, caso ela exista.
  + **AddData:** Adiciona um novo par chave-valor ao cache.
  + **UpdateData:** Atualiza o valor associado a uma chave, caso ela já exista no cache.
  + **ResetTTL:** Reseta o TTL de uma entrada no cache para prolongar sua validade.

**2.2. Classe DB (Data Base)**

* **Descrição:** Centraliza todas as operações relacionadas ao banco de dados, incluindo CRUD, validação e integração com o Entity Framework.
* **Atributos:**
  + **Nenhum:** Toda lógica é encapsulada em métodos.
* **Funções:**
  + **GetCountry:** Busca um país no banco de dados pelos códigos ISO.
  + **GetIP:** Busca um IP salvo no banco de dados.
  + **AddCountry:** Adiciona um novo país ao banco de dados.
  + **AddIP:** Adiciona um novo IP ao banco.
  + **UpdateDataBase:** Atualiza os dados do banco comparando com a API externa.
  + **CRUD Genérico:** Métodos para leitura, escrita, atualização e exclusão de registros no banco.

**2.3. Classe EA (External API)**

* **Descrição:** Gerencia todas as interações com a API pública externa (IP2C).
* **Atributos:**
  + **Nenhum:** Toda lógica é encapsulada em métodos.
* **Funções:**
  + **GetCountry:** Recebe um IP, consulta a API e retorna os dados estruturados.
  + **ParseCsvData:** Converte dados em CSV retornados pela API em um formato utilizável no sistema.
  + **HandleErrors:** Trata erros de requisição, como timeouts ou respostas inválidas.

**2.4. Classe DC (Data Converter)**

* **Descrição:** Centraliza a conversão de dados externos em objetos das classes entidade, também faz o oposto, convertendo um objeto do sistema em dados que podem ser passados como uma resposta do sistema.
* **Atributos:**
  + **Nenhum:** Toda lógica é encapsulada em métodos.
* **Funções:**
  + **BuildIP:** Recebe dados estruturados e retorna um objeto do tipo IP.
  + **SerializeIP:** Recebe um objeto IP e retorna dados estruturados transferíveis.
  + **BuildCountry:** Recebe dados estruturados e retorna um objeto do tipo Country (inclui validação).
  + **SerializeCountry:** Recebe um objeto Country e retorna dados estruturados transferíveis.
  + **BuildReport:** Recebe dados estruturados e retorna um objeto do tipo Report.
  + **SerializeReport:** Recebe um Array de objetos do tipo Report e retorna um Array de dados estruturados transferíveis. Aqui se usa um array porque dentro do escopo da aplicação, não existe contexto em que um único Report seja serializado.

**2.5. Classe UD (Update Data)**

* **Descrição:** Encapsula a lógica da atualização de dados no banco e no cache, com operações em lotes.
* **Atributos:**
  + **Nenhum:** Toda lógica é encapsulada em métodos.
* **Funções:**
  + **UpdateDataBase:** Varre o banco de dados em lotes de 100 registros, compara com a API externa e atualiza dados inconsistentes.
  + **UpdateCache:** Atualiza as referências no cache para refletir as mudanças no banco.

**2.6. Classe RM (Report Manager)**

* **Descrição:** Gerencia as operações para o endpoint de relatórios, centralizando a lógica necessária para retornar os dados.
* **Atributos:**
  + **Nenhum:** Toda lógica é encapsulada em métodos.
* **Funções:**
  + **GetReports:** Recebe um array de códigos de países e retorna relatórios para os países solicitados.
  + **GetAllReports:** Retorna relatórios para todos os países no banco de dados.