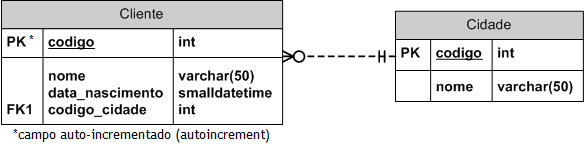
|  |  |  |  |
| --- | --- | --- | --- |
| ICI.png | **Prova de ASP.NET MVC – C#** | | |
| Nome: Arthur Pereira | | **Data: ­­31/08/2021** |
| **Tempo da prova:** 01h30min | **Inicio:** 18h00min | **Término:** 19h30min |

1. **Prática**

Olá candidato,

* Você tem uma “Solution” criada no Visual Studio 2015 com dois projetos (ProvaCandidato.Data e ProvaCandidato.Web), nesta solução há classes já desenvolvidas e funcionando para acesso ao banco de dados utilizando o EntityFramework. Utilize esta estrutura para resolver os exercícios abaixo.
* Caso não consiga utilizar as classes já prontas, você pode criar suas próprias classes para acessar o banco de dados.
* Crie uma base de dados SQL Server local chamada provacandidato. A string de conexão está na classe ContextoPrincipal, utilize-a ou configure com base nas configurações da sua base de dados.
* As tabelas necessárias para realizar os exercícios serão criadas pelo próprio EntityFramework, utilizando o conceito code-first.
* O layout, HTML e folhas de estilo não são relevantes nesta avaliação.
* Se você sentir dificuldade em algum item ou em parte de um item, pule para o próximo. Os itens são avaliados separadamente e será considerado desenvolvimento parcial.
* Ao tentar rodar o projeto, você poderá receber um erro. Identifique o problema descrito pela mensagem de erro e corrija o problema para prosseguir.

Considere o DER (Diagrama de Entidade e Relacionamento) abaixo para resolver os exercícios.



* 1. Solucione o problema de inicialização do projeto e faça a referência do projeto ProvaCandidato.Data no projeto ProvaCandidato.Web para utilizar as classes que farão acesso ao banco de dados.
  2. No leiaute, tornar o texto "Parametrize o Nome da Empresa Aqui" parametrizável, sendo que seu conteúdo deverá ser obtido a partir do Web.Config através da chave NomeEmpresa do AppSettings. A classe ConfigurationManager do namespace System.Configuration pode lhe ajudar.
  3. Crie os controllers das classes Cidade e Cliente e desenvolva as funcionalidades básicas de listagem, inclusão, alteração e exclusão dos dados. Você poderá utilizar o scaffolder do EntityFramework para lhe auxiliar no processo.
  4. Inclua uma validação obrigando que o tamanho do Nome da Cidade tenha no mínimo 3 caracteres e no máximo 50 caracteres. Aproveite e aplique a mesma regra ao campo Nome do Cliente. Você não poderá permitir a gravação se a Data de Nascimento do Cliente for superior à data atual. Você poderá criar uma validação customizada ou validar manualmente. Os atributos MinLength, MaxLength ou StringLength podem ajudar a validar o tamanho e a validação da data poderá ser realizada diretamente no controller através do ModelState.
  5. Crie uma classe genérica herdando de Controller e modifique a herança dos controllers de Cliente e Cidade para que passem a usar essa nova classe. As classes de Cliente e Cidade deverão ser passadas como argumento para a classe genérica.
  6. Corrigir a formatação da Data de Nascimento do Cliente para o formato dia, mês e ano. Talvez você precise utilizar os atributos DataType(DataType.Date) e DisplayFormat para concluir esta atividade.
  7. A seleção da Cidade no CRUD de Cliente deve ocorrer através de uma lista de seleção (DropDown ou Combo). **Será considerado um bônus** se você utilizar WebAPI e popular a combo via AJAX. Não se lembra qual é a sintaxe para retornar os dados via AJAX? $.get("ajax/test.html", function(data) {

$(".result").html(data);

alert("Load was performed.");

});

* 1. Implementar na Lista de Clientes uma funcionalidade que permita pesquisar (filtrar) os Clientes por qualquer parte do Nome.
  2. Após a inclusão, alteração e exclusão de um Cliente ou Cidade, exibir uma mensagem de sucesso usando a classe MessageHelper.
  3. Modificar a Lista de Clientes para que exiba o Nome da Cidade do Cliente. Será necessário fazer o Include da propriedade Cidade ao montar a query no DBSet do Cliente.
  4. **Este item é opcional** e você poderá utilizá-lo para demonstrar seu conhecimento com o EntityFramework, Migrations e Javascript.   
     Você deverá criar uma classe chamada ClienteObservacao cujo conteúdo será apenas uma propriedade para armazenar a observação e uma referência para o Cliente.  
     Você deverá alterar a classe Cliente para que esta contenha uma lista de observações do cliente. Não se esqueça do atributo InverseProperty e que a propriedade deve ser do tipo ICollection.  
     Você deverá alterar a visualização de detalhe do cliente para que seja possível adicionar as observações.  
     A inclusão de uma nova observação deverá ocorrer através de um popup que exibirá uma Partial View.  
      Lembre-se que o projeto de dados é o ProvaCandidato.Data e o Migrations não está habilitado. Os comandos help Enable-Migrations, help Add-Migration e help Update-Database poderão ser muito úteis para habilitar o Migrations e atualizar o banco de dados.   
     Abaixo, um exemplo para implementar a partial view...  
     

1. **Teórica**
   1. Explique o funcionamento dos blocos try{}, catch{} e finally{}.

try{ tenta executar o que deve ser executado}

catch{ se algo der errado cai aqui}

finally{ garante que independente do que aconteca vai executar o que esta nesse bloco }

* 1. Qual a função do Migrations no EntityFramework e qual problema ele tem a intenção de resolver?

Ele gera uma query que criara as entidades e os relacionamentos no banco de dados.

* 1. Qual o benefício que a ViewBag traz sendo do tipo dynamic?

Servem como variáveis vindas da controller diretas para view.