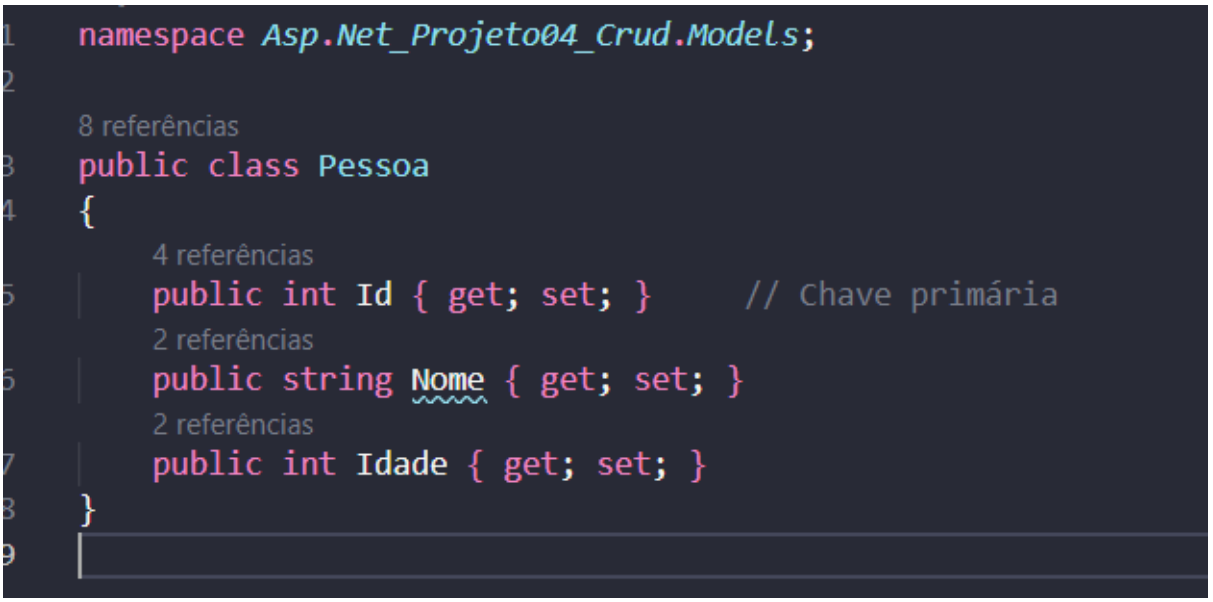
1 - Arquivo: Views/Home/Index.cshtml

Explicação: Criação da classe que pega a **chave** Id , Nome , Idade ; Responsável por pegar informações obtida do formulário front-end usando a propriedade Getter e Setter.



```
namespace Asp.Net_Projeto04_Crud.Models;

8 referências
public class Pessoa
{
    4 referências
    public int Id { get; set; }     // Chave primária
    2 referências
    public string Nome { get; set; }
    2 referências
    public int Idade { get; set; }
}
```

Novidades: O MySQL cria automaticamente o auto-incremento da chave primária, então nós não precisamos criar uma lógica para isso.

2 - Arquivo: Models/BancoDados.cs

Explicação: Conexão Do Banco de Dados , Configuração e criação de tabela do Banco de Dados

```
using Microsoft.EntityFrameworkCore;    // Importa o namespace do Entity Framework Core

namespace Asp.Net_Projeto04_Crud.Models;

2 referências
public class BancoDados : DbContext     // DbContext é a classe que gerencia a conexão com o banco.
{

    // Criação da tabela Pessoas
    6 referências
    public DbSet<Pessoa> PessoaDB { get; set; }  // Transfere as informações da classe Pessoa Model/Pessoa.cs para a Variável
    PessoaDB

    0 referências
    protected override void OnConfiguring(DbContextOptionsBuilder options)
    {
        // Aqui você coloca sua senha do MySQL e outras configurações
        options.UseMySQL("server=localhost;database=projeto_crud;user=usuario_crud;password=123456");
    }
}
```

Novidades:

O Entity Framework atua como um tradutor. Ele pega o seu comando em **C#** e transforma automaticamente na linguagem **SQL** que o MySQL entende.

Ação	Comando C# (Entity Framework)	O que faz no MySQL
Create (Criar)	`banco.PessoaDB.Add(p)`	`INSERT INTO PessoaDB...`
Read (Ler)	`banco.PessoaDB.ToList()`	`SELECT * FROM PessoaDB`
Update (Editar)	`banco.PessoaDB.Update(p)`	`UPDATE PessoaDB SET... WHERE Id = X`
Delete (Excluir)	`banco.PessoaDB.Remove(p)`	`DELETE FROM PessoaDB WHERE Id = X`

`public DbSet<Pessoa> PessoaDB { get; set; }`
Este comando cria a ponte de transferência de dados. A classe **Pessoa** (o molde) captura as informações vindas do formulário HTML. Esses dados são então transferidos para o **PessoaDB**, que representa a tabela real dentro do banco de dados MySQL.

3 - Arquivos: Controllers/HomeController.cs

Explicação: Manipulação de Dados do Banco de Dados

```
10      // Conexão com o banco de dados
        10 referências
11      private BancoDados banco = new BancoDados();
12
        0 referências
13      public IActionResult Index()
14      {
15
16          // Garante que o banco de dados e as tabelas existam
17          banco.Database.EnsureCreated();
18
19          // Busca a lista diretamente da tabela do MySQL
20          var lista = banco.PessoaDB.ToList();
21
22          // Variavel usada para mandar informações para o arquivo HTML
23          ViewBag.Pessoas = lista;
24
25          return View();
26      }
```

Novidade:

`private BancoDados banco = new BancoDados();`
Instância da classe BancoDados.cs (do arquivo Models) que permite ter acesso ao atributo PessoaDB; sendo assim, você consegue manipular os dados.

FUNÇÃO ADICIONAR

```
// Função para Adicioar
[HttpPost]
0 referências
public IActionResult Adicionar(Pessoa input)
{
    // Adiciona o objeto 'input' na tabela Pessoas do banco
    banco.PessoaDB.Add(input);

    // Salva as mudanças de fato no arquivo do banco de dados (COMMIT)
    banco.SaveChanges();

    return RedirectToAction("Index");   // Redireciona para a ação Index
}
```

FUNÇÃO EXCLUIR

```
// Função para Excluir
0 referências
public IActionResult Excluir(int id)
{
    // Busca a pessoa pelo ID diretamente no banco de dados
    var pessoa = banco.PessoaDB.Find(id);

    if (pessoa != null)
    {
        banco.PessoaDB.Remove(pessoa); // Marca para remover
        banco.SaveChanges();          // Executa o DELETE no MySQL
    }

    return RedirectToAction("Index");
}
```

FUNÇÃO EDITAR

```
// Função para Editar (Atualizar)
0 referências
public IActionResult Editar(int id)
{
    // Busca os dados da pessoa para preencher o formulário de edição
    var pessoa = banco.PessoaDB.Find(id);
    return View(pessoa);
}
```

FUNÇÃO CONFIRMAR EDIÇÃO

```
// Função para Confirmar a Edição (Salva no banco)
[HttpPost]
0 referências
public IActionResult ConfirmarEdicao(Pessoa input)
{
    // O comando Update avisa que este objeto (com o mesmo ID) tem novos dados
    banco.PessoaDB.Update(input);
    banco.SaveChanges(); // Executa o UPDATE no MySQL

    return RedirectToAction("Index");
}
```

Novidades :

Esta tabela resume como o Entity Framework traduz suas intenções de programação em ações reais no banco de dados.

Função	Lógica no C#	Impacto no MySQL
Listar	`ToList()`	Transforma as linhas da tabela em uma lista de objetos.
Localizar	`Find(id)`	Busca um registro específico usando a Chave Primária.
Inserir	`Add(input)`	Prepara uma nova linha para ser inserida na tabela.
Atualizar	`Update(input)`	Identifica o ID e substitui os valores das colunas (Nome, Idade).
Remover	`Remove(pessoa)`	Deleta permanentemente o registro correspondente ao ID.

Essas funções são responsáveis pela parte lógica e, após a execução, elas procuram automaticamente o arquivo HTML que possui o mesmo nome da função para renderizar a página.

Ex: Assim que o usuário clica em **'Editar'**, o sistema executa a função **Editar** no `HomeController` e, logo em seguida, procura o arquivo HTML de mesmo nome (**Editar.cshtml**) para exibir na tela.

4 - Arquivo: Home/Index.cshtml

Explicação: Onde possui o formulário que pega as informações e enviar para classe Pessoa no arquivo Models/Pessoa.cs

```
@using Asp.Net_Projeto04_Crud.Models
@model IEnumerable<Pessoa>  // serve para avisar ao HTML que ele deve se preparar para lidar com vários registros vindos do banco de dados, e não apenas um.

<h2>Cadastro de Pessoas (MySQL)</h2>

<form asp-action="Adicionar" method="post">
    <input name="Nome" placeholder="Digite seu Nome" required />
    <input name="Idade" type="number" placeholder="Digite sua Idade" required />
    <button type="submit">Cadastrar no Banco</button>
</form>
```

Parte do código responsável por mostrar as informações obtida pelo back-end e por ativar as funções do Arquivo Controllers/HomeControllers.cs —> Funções Editar e Excluir

```
<table border="1">
    <tr>
        <th>ID</th>
        <th>Nome</th>
        <th>Idade</th>
        <th>Ações</th>
    </tr>
    @foreach (var Pessoa p in (List<Pessoa>)ViewBag.Pessoas)
    {
        <tr>
            <td>@p.Id</td>
            <td>@p.Nome</td>
            <td>@p.Idade</td>
            <td>
                <a asp-action="Editar" asp-route-id="@p.Id">Editar</a>  <!--Vai para função Editar e depois o arquivo HTML -->
                <a asp-action="Excluir" asp-route-id="@p.Id">Excluir</a>
            </td>
        </tr>
    }
</table>
```

Novidade : o `asp-route-id` já captura o ID da pessoa automaticamente. Assim, quando o botão é clicado, o sistema já sabe exatamente qual registro deve ser manipulado no banco de dados sem que você precise criar rotas manuais.

```
@model Pessoa

<h2>Editar Registro</h2>

<form asp-action="ConfirmarEdicao" method="post">
    <input type="hidden" asp-for="Id" />

    <div>
        <label>Nome Atual:</label>
        <input asp-for="Nome" />     <!--Envia para classe Pessoa.cs chave Nome-->
    </div>

    <div>
        <label>Idade Atual:</label>
        <input asp-for="Idade" />   <!--Envia para classe Pessoa.cs chave Idade-->
    </div>

    <button type="submit">Salvar no Banco</button>
    <a asp-action="Index">Cancelar</a>
</form>
```

Função de ConfirmarEdição (Controllers/HomeControllers.cs) no formulário (Home/Editarcshtml)