Criação aplicação Web em ASP.NET Core MVC que permita usuário fazer upload de arquivos, armazenar informações sobre esses arquivos em um banco de dados e depois exibir esses arquivos em uma lista

1. Criação do projeto no visual studio

ASP.NET Core Web App (Model-View-Controller). Isso porque queremos uma aplicação que siga o padrão MVC (Model-View-Controller), onde a **View** será nossa interface, o **Model** representará os dados que trabalharemos, e o **Controller** fará a lógica de controle.

1. Definir o Modelo ( model )

Model representa os dado que manipulamos no sistema. No caso, o que seria um “arquivo” enviado? Vamos armazenar informações básicas sobre esse arquivo:

* O nome do arquivo
* O caminho onde o arquivo será armazenado
* A data de envio.

Na pasta Models, criei um nova classe chamada **Arquivo.cs** . Essa classe é como uma “representação” dos arquivos enviados. Vamos colocar algumas propriedades:

|  |
| --- |
| public class Arquivo  {  public int Id { get; set; } // Um identificador único para cada arquivo.  public string NomeArquivo { get; set; } // O nome original do arquivo.  public string CaminhoArquivo { get; set; } // O caminho onde o arquivo está salvo.  public DateTime DataEnvio { get; set; } // A data em que o arquivo foi enviado.  } |

1. Configurar o Banco de Dados com Entity Framework

Como já sei como vou armazenar os dados do arquivo, preciso garantir que esses dados sejam salvos em um banco de dados. Para isso, usarei o **Entity Framework Core (EF Core)**, que me ajuda a mapear os modelos para tabelas no banco de dados.

* No arquivo appsettings.json vou adicionar uma string de conexão para o banco de dados.

|  |
| --- |
| "ConnectionStrings": {  "DefaultConnection": "Server=localhost\\sqlexpress;Initial Catalog=FileUploadDB;Integrated Security=True;TrustServerCertificate=True;"  } |

Essa configuração diz que usarei um banco de daods local chamado **FileUploadDB**.

* Agora vou criar um “contexto” que vai gerenciar a conexão com o banco e as operações que farei nele. Na pasta Data vou criar uma nova classe chamada **ApplicationDbContext.cs**:

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using FileUploadApp.Models;  public class ApplicationDbContext : DbContext  {  public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options): base(options) { }  public DbSet<Arquivo> Arquivos { get; set; } // Representa a tabela de arquivos no banco de dados.  } |

* Para garantir que o EntityFramework Core funcione corretamente, também preciso configurar o **Program.cs** para usar esse contexto:

|  |
| --- |
| builder.Services.AddDbContext<ApplicationDbContext>(options =>  options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"))); |

Agora o Sistema está conectado ao banco de dados, posso começar a gravar os arquivos nele.

1. Criar o Controller

O **Controller** é a "ponte" entre a interface (View) e os dados (Model). Ele vai gerenciar o que acontece quando o usuário faz upload de um arquivo.

Crie um controller chamado ArquivoController.cs. Ele vai ter dois métodos principais:

* **Upload**: Para receber os arquivos.
* **Lista**: Para exibir os arquivos.

|  |
| --- |
| * public class ArquivoController : Controller * { * private readonly ApplicationDbContext \_context; * public ArquivoController(ApplicationDbContext context) * { * \_context = context; * } * [HttpGet] * public IActionResult Upload() * { * return View(); * } * [HttpPost] * public async Task<IActionResult> Upload(IFormFile arquivo) * { * if (arquivo != null && arquivo.Length > 0) * { * // Salvando o arquivo na pasta 'uploads' dentro do projeto. * var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/uploads", arquivo.FileName); * using (var stream = new FileStream(path, FileMode.Create)) * { * await arquivo.CopyToAsync(stream); * } * // Salvando as informações no banco de dados. * var novoArquivo = new Arquivo * { * NomeArquivo = arquivo.FileName, * CaminhoArquivo = "/uploads/" + arquivo.FileName, * DataEnvio = DateTime.Now * }; * \_context.Arquivos.Add(novoArquivo); * await \_context.SaveChangesAsync(); * return RedirectToAction("Lista"); * } * return View(); * } * public IActionResult Lista() * { * var arquivos = \_context.Arquivos.ToList(); * return View(arquivos); * } * } |

1. Criar as Views

Agora vamos criar as telas que os usuários verão.

* **Upload de Arquivos**: Crie uma view chamada Upload.cshtml dentro da pasta Views/Arquivo. Essa será a página onde o usuário pode escolher e enviar um arquivo:

|  |
| --- |
| * <h2>Upload de Arquivo</h2> * <form asp-action="Upload" method="post" enctype="multipart/form-data"> * <input type="file" name="arquivo" /> * <button type="submit">Enviar</button> * </form> |

* **Lista de Arquivos**: Crie uma view chamada Lista.cshtml para exibir os arquivos já enviados:

|  |
| --- |
| * <h2>Lista de Arquivos</h2> * <table> * <thead> * <tr> * <th>Nome do Arquivo</th> * <th>Data de Envio</th> * <th>Visualizar</th> * </tr> * </thead> * <tbody> * @foreach (var arquivo in Model) * { * <tr> * <td>@arquivo.NomeArquivo</td> * <td>@arquivo.DataEnvio</td> * <td><a href="@arquivo.CaminhoArquivo" target="\_blank">Abrir</a></td> * </tr> * } * </tbody> * </table> |

**Passo 6: Testar a Aplicação**

Antes de rodar o projeto, não esqueça de:

* Criar a pasta wwwroot/uploads para armazenar os arquivos.
* Rodar as migrações do Entity Framework para criar a tabela no banco de dados:

|  |
| --- |
| * Add-Migration Inicial * Update-Database |

**Revisão**

* Criamos um modelo que representa os arquivos enviados.
* Configuramos o Entity Framework para salvar essas informações no banco de dados.
* Criamos um controller que recebe os arquivos e exibe uma lista.
* Criamos as views para a interface.