INSTITUTO FEDERAL
CATARINENSE
Câmpus Videira

Turma: Ciência da Computação
Aluno: Bruno Pergher
Disciplina: Linguagem de programação II
Professor: Fabricio Bizotto

Antes de começar de fato a trabalhar com os arquivos do MVC, foi criado um data model que a partir dele geramos a tabela no banco, sendo um model bem parecido com o model que irei usar no MVC.

```
1   using System.ComponentModel.DataAnnotations;
2
3   namespace LP2MVCWithAuth.Data.Models
4   {
        3 references
5       public class ListaTarefas
6       {
7           [Key]
            2 references
8           public int Id { get; set; }
9
10          [StringLength(100)]
11          [Required]
            3 references
12          public string Titulo { get; set; }
13
14          [StringLength(1000)]
            3 references
15          public string? Descricao { get; set; }
16
17          [Required]
            7 references
18          public DateTime DataEntrega { get; set; }
19      }
20  }
21
```

class LP2MVCWithAuth.Data.Models.ListaTarefas

Após ter criado o banco de dados para o sistema, foi começado a ser trabalhado no sistema em MVC, agora abordando cada parte do MVC temos no M - Model:

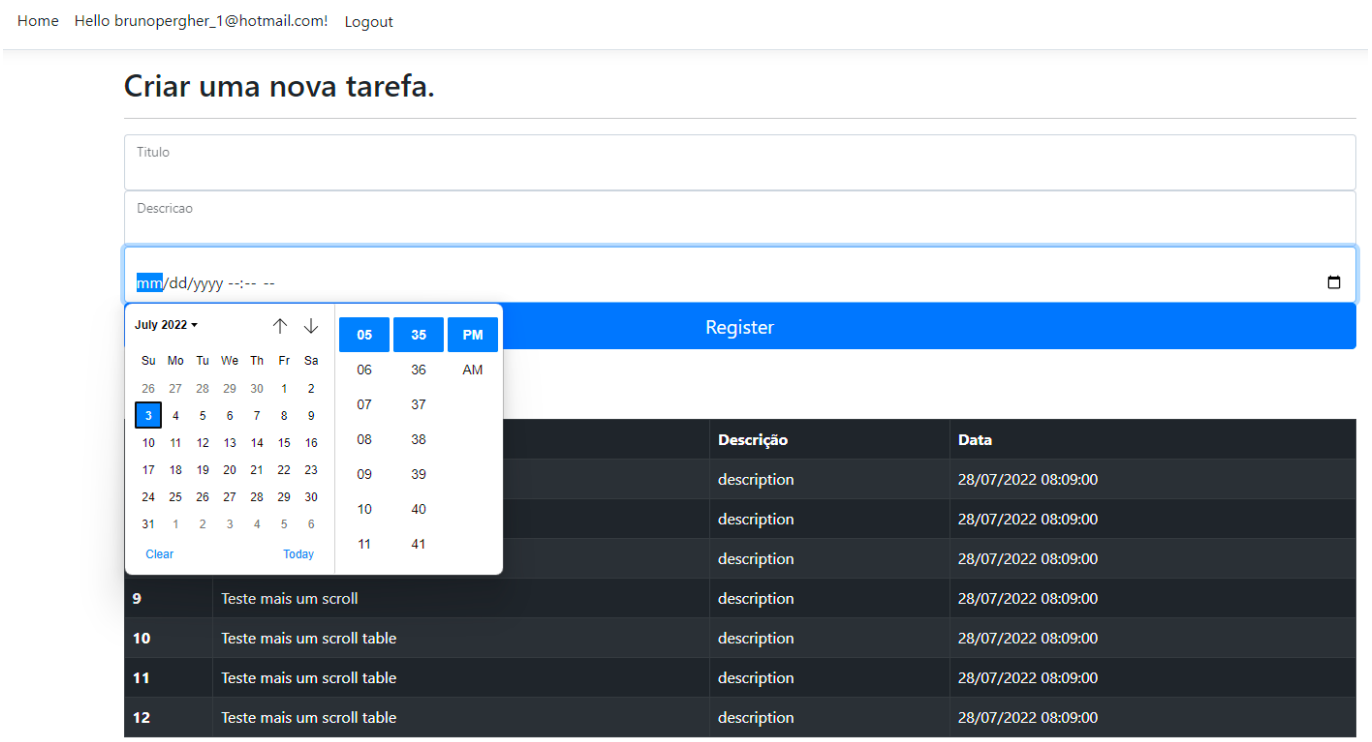
Esse Model foi feito para a lista de tarefas, ele fica dentro da pasta de models, e possui as informações de uma tarefa como também a lista de todas as tarefas que será usada para mostrar a lista de tarefas. Também é usado decorators como o StringLength que vai validar o tamanho máximo para esse dados no formulário, também é usado o Required, que mostra que esse dado é obrigatório, e também temo o DisplayFormat que deixa a data em um padrão comum para nós.

```cs
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Mvc.Rendering;
using System;
using System.Collections.Generic;

namespace LP2MVCWithAuth.Models
{

    public class ListaTarefas
    {

        public ListaTarefas()
        {
            this.Tarefas = new List<ListaTarefas>();
        }


        public int Id { get; set; }

        [StringLength(100)]
        [Required]

        public string Titulo { get; set; }

        [StringLength(1000)]

        public string? Descricao { get; set; }

        [Required]
        [DisplayFormat(ApplyFormatInEditMode = true, DataFormatString = "{0:dd/MM/yyyy HH:MM}")]

        public DateTime DataEntrega { get; set; }

        5 references
        public List<ListaTarefas> Tarefas { get; set; }
    }
}

```

Agora que temos o banco criado, e o model que vou usar na view, foi criado a view.

```cs
@using Microsoft.AspNetCore.Identity
@using System.Globalization
@inject SignInManager<IdentityUser> SignInManager
@inject UserManager<IdentityUser> UserManager
@model LP2MVCWithAuth.Models.ListaTarefas
@{
    ViewData["Title"] = "Home Page";
}
<link rel="stylesheet" href="app.css" type="text/css" />

<div class="container">
    @if (SignInManager.IsSignedIn(User))
    {
        <div>
            <form asp-controller="Home" asp-action="Index" method="post">
                <h2>Criar uma nova tarefa.</h2>
                <hr />

                <div asp-validation-summary="ModelOnly" class="text-danger"></div>
                <div class="form-floating">
                    <input asp-for="Titulo" class="form-control" aria-required="true" />
                    <label asp-for="Titulo"></label>
                    <span asp-validation-for="Titulo" class="text-danger"></span>
                </div>

                <div class="form-floating">
                    <input asp-for="Descricao" class="form-control" aria-required="true" />
                    <label asp-for="Descricao"></label>
                    <span asp-validation-for="Descricao" class="text-danger"></span>
                </div>

                <div class="form-floating">
                    <input class="form-control" asp-for="DataEntrega" />
                </div>

                <button type="submit" class="w-100 btn btn-lg btn-primary">Register</button>
            </form>
        </div>

        <br />
        <br />
        <br />

```

Essa view vai usar a autenticação da framework para facilitar, e vai usar o model que criamos anteriormente, ao ser validado se o usuario está logado ou não, caso sim, será renderizado o formulário usado para criar uma tarefa, nele temos um form, do tipo post, que passa qual o controller que vai ser enviado a informação do form e qual o método do controller que será enviado essa informação, os asp-for e asp validation fazem as validações e mostram as mensagens de erro baseados nas decorations criadas no model, e o asp for da data cria um input tipo data pra pegar essa informação.



Continuando o código temos a parte da listagem das tarefas:

```
@if (Model.Tarefas.Count() > 0)
{
    <div class="table-wrapper-scroll-y my-custom-scrollbar scrollTable" style="overflow:auto">
        <table class="table table-dark table-bordered table-striped mb-0">
            <thead>
                <tr>
                    <th scope="col">Id</th>
                    <th scope="col">Titulo</th>
                    <th scope="col">Descrição</th>
                    <th scope="col">Data</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model.Tarefas)
                {
                    <tr>
                        <th scope="row">@item.Id</th>
                        <td>@item.Titulo</td>
                        <td>@item.Descricao</td>
                        <td>@item.DataEntrega.ToString("dd/MM/yyyy HH:mm:ss")</td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
}
else
{
    <div>
        Faça Login Para ter acesso ao site
    </div>
}
</div>
```

Após validar que existem tarefas, é feito a impressão delas em uma tabela através de um foreach. Além disso temos o else de que caso o usuário não esteja logado ele irá pedir para o usuário logar.

Id	Titulo	Descrição	Data
1002	Teste pra documentação	amanha	04/07/2022 17:54:00
6	Teste	description	28/07/2022 08:09:00
7	Teste mais um	description	28/07/2022 08:09:00
8	Teste mais um scroll	description	28/07/2022 08:09:00
9	Teste mais um scroll	description	28/07/2022 08:09:00
10	Teste mais um scroll table	description	28/07/2022 08:09:00
11	Teste mais um scroll table	description	28/07/2022 08:09:00
12	Teste mais um scroll table	description	28/07/2022 08:09:00

Agora temos o C - controller, primeiro iremos abordar o methodo do index com o get, aqui declaramos a database, onde pegamos no database as tarefas cadastradas que ainda existem prazo, e ordenamos pela mais próxima, após isso declaramos essas tarefas no model retornamos a página passando o model como variável, não é necessario definir a rota do get e da view pois temos ambas com o nome de index

```
3 references
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    private readonly ApplicationDbContext _database;

    0 references
    public HomeController(ILogger<HomeController> logger, ApplicationDbContext database)
    {
        _logger = logger;
        _database = database;
    }

    [HttpGet]
    0 references
    public IActionResult Index()
    {
        var tarefas = _database.ListaDeTarefas.ToList().Where(x => x.DataEntrega > DateTime.UtcNow).OrderBy(x => x.DataEntrega);

        Models.ListaTarefas listaTarefas = new Models.ListaTarefas();

        if (tarefas.Count() > 0)
        {
            foreach (var item in tarefas)
            {
                Models.ListaTarefas tarefa = new Models.ListaTarefas
                {
                    DataEntrega = item.DataEntrega,
                    Descricao = item.Descricao,
                    Titulo = item.Titulo,
                    Id = item.Id,
                };

                listaTarefas.Tarefas.Add(tarefa);
            }
        }

        return View(listaTarefas);
    }
```

Partindo do get, temos um método post pra quando acontecer o submit do form:

```
[HttpPost]
0 references
public IActionResult Index(Models.ListaTarefas listaTarefas)
{
    Data.Models.ListaTarefas tarefa = new Data.Models.ListaTarefas
    {
        DataEntrega = listaTarefas.DataEntrega,
        Descricao = listaTarefas.Descricao,
        Titulo = listaTarefas.Titulo,
    };

    _database.ListaDeTarefas.Add(tarefa);
    _database.SaveChanges();

    var tarefas = _database.ListaDeTarefas.ToList().Where(x => x.DataEntrega > DateTime.UtcNow).OrderBy(x => x.DataEntrega);

    if (tarefas.Count() > 0)
    {
        foreach (var item in tarefas)
        {
            Models.ListaTarefas tarefaa = new Models.ListaTarefas
            {
                DataEntrega = item.DataEntrega,
                Descricao = item.Descricao,
                Titulo = item.Titulo,
                Id = item.Id,
            };

            listaTarefas.Tarefas.Add(tarefaa);
        }
    }

    return View(listaTarefas);
}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
0 references
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
```

quando chamar o método post ele vai criar um Data model daquela tarefa para salvar essa tarefa no banco de dados, após isso ele vai pegar as tarefas do banco novamente e retornar a view com a nova tarefa registrada.

Os benefícios de usar o mvc é criar no controller métodos privados que poderiam ser usado em diversos métodos diferentes melhora muito a organização no projeto.

Sobre os defeitos da minha aplicação, acredito que não está tão otimizada como poderia ser, e que poderia ter trabalhado com um model só para tarefa e outro pra lista de tarefas em vez de uma só.