ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2023/1

Luiz Fernando Souza / Quitéria Danno

Aula 15 – Projeto Front-end MVC Consumindo API – Controller e Views para disputa entre os personagens

1. Na view Personagens/Index, crie o primeiro botão de disputas abaixo da tag de fechamento da table.

```
@Html.ActionLink("Clique aqui para um embate com armas!!!", "Index", "Disputas",
    null, new { @class = "btn btn-warning" })
```

- Execute o projeto e verifique se os dados serão apresentados abaixo da tabela.

2. Crie uma classe chamada **DisputaViewModel.cs** dentro da pasta Models, com as propriedades a seguir

```
public int Id { get; set; }
0 references
public DateTime? DataDisputa { get; set; }
0 references
public int AtacanteId { get; set; }
0 references
public int OponenteId { get; set; }
0 references
public string Narracao { get; set; }
0 references
public int HabilidadeId { get; set; }
0 references
public List<int> ListaIdPersonagens { get; set; } = new List<int>();
0 references
public List<string> Resultados { get; set; } = new List<string>();
```

- Será necessário o using System.Collections.Generic e System

3. Crie a classe chamada **DisputasController.cs** dentro da pasta Controllers, programando a herança à classe Controller e deixando o endereço da controller da sua API definido na variável uriBase

```
public class DisputasController : Controller
{
    0 references
    public string uriBase = "xyz/Disputas/";
    //xyz tem que ser substituído pelo nome do seu site na API.

    //Próximos métodos ficarão aqui
}
```

- Sintaxe Controller exigirá o using Microsoft.AspNetCore.Mvc

4. Ainda na controller criada, programe o método do tipo HttpGet que vai gerar carregar a tela de disputas. Adicione os usings necessários nos locais em que o código indicar erros de referência.

```
[HttpGet]
public async Task<ActionResult> IndexAsync()
{
    try
    {
        HttpClient httpClient = new HttpClient();
    A   string token = HttpContext.Session.GetString("SessionTokenUsuario");
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

        string uriBuscaPersonagens = "http://xyz.somee.com/RpgApi/Personagens/GetAll";
    B   HttpResponseMessage response = await httpClient.GetAsync(uriBuscaPersonagens);
        string serialized = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            List<PersonagemViewModel> listaPersonagens = await Task.Run(() =>
                JsonConvert.DeserializeObject<List<PersonagemViewModel>>(serialized));
        C
            ViewBag.ListaAtacantes = listaPersonagens;
            ViewBag.ListaOponentes = listaPersonagens;
            return View();
        }
        else
        D   throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

- Usings: System.Net.Http.Headers; RpgMvc.Models ; Newtonsoft.Json; System.Collections.Generic; System.Net.Http; System.Threading.Tasks e Microsoft.AspNetCore.Http

(A) Criação da variável http e obtenção do token guardado na session

(B) Definição da rota da API que buscará a lista de personagens na API, retornando uma lista se o método tiver êxito ou uma mensagem caso dê erro. Isso tudo guardando ainda serializado.

(C) Se o status da requisição for 200 (Ok) então deserializamos para transformar numa lista de personagens, e depois geramos duas *ViewBags* a partir da lista de personagens, uma como os atacantes e outra como os oponentes. *ViewsBags* são maneiras de trafegar dados entre a controller e views e isso é que fará o carregamento do dropdownlist aparecer.

(D) Mensagem de erro sendo lançada no else e capturada no cacht sendo guardada através do *TempData*. *TempData* é outra maneira de trafegar dados entre uma controller e uma View.

5. Crie uma pasta chamada **Disputas** dentro da pasta Views e dentro da pasta Disputas, crie o arquivo do chamado **Index.cshtml** com o layout abaixo

```
@model RpgMvc.Models.DisputaViewModel
@{ViewBag.Title = "Index";}

@if (@TempData["Mensagem"] != null) {
    <div class="alert alert-success" role="alert">
        @TempData["Mensagem"]</div>
}
@if (@TempData["MensagemErro"] != null) {
    <div class="alert alert-danger" role="alert">
        @TempData["MensagemErro"]</div>
}
<h2>Ataque com Arma</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <hr />
        <div class="form-group">
            @Html.DisplayName("Atacante")
            <div class="col-md-6">
  @Html.DropDownListFor(model => model.AtacanteId, new SelectList(@ViewBag.ListaAtacantes, "Id
", "Nome"),
                    "---Selecione---", new { @class = "form-control" })</div>
        </div>
         <div class="form-group">
            @Html.DisplayName("Oponente")
            <div class="col-md-6">
@Html.DropDownListFor(model => model.OponenteId, new SelectList(@ViewBag.ListaOponentes, "Id",
 "Nome"),
                    "---Selecione---", new { @class = "form-control" })</div>
        </div><br/>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-6">
          <input type="submit" value="Atacar com Arma!!!" class="btn btn-primary" /></div>
        </div>
    </div>
}
<div>@Html.ActionLink("Retornar", "Index", "Personagens")</div>
```

- Execute e verifique se a view será carregada ao clicar no botão de disputa com arma
- Observe como utilizar uma caixa de seleção usando DropDownListFor

6. Volte à controller de disputas e crie o método que realizará a postagem do ataque com armas para a API, acrescente os usings necessários nos locais em que for apresentado erros de referência.

```
[HttpPost]
0 references
public async Task<ActionResult> IndexAsync(DisputaViewModel disputa)
{
    try
    {
        HttpClient httpClient = new HttpClient();
        string uriComplementar = "Arma";

A       var content = new StringContent(JsonConvert.SerializeObject(disputa));
        content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
        HttpResponseMessage response = await httpClient.PostAsync(uriBase + uriComplementar, content);
        string serialized = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            disputa = await Task.Run(() => JsonConvert.DeserializeObject<DisputaViewModel>(serialized));
B           TempData["Mensagem"] = disputa.Narracao;
            return RedirectToAction("Index", "Personagens");
        }
        else
            throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

(A) Configuração da rota para a API. Envio da requisição e armazenamento do retorno da requisição.

(B) Se o retorno for código 200 (ok) desserializa para o objeto disputa e armazena no TempData a narração do ataque e retorna para a View Index de Personagens, caso contrário uma lançará um erro.

- Execute o projeto e teste a realização do ataque

Ataque usando Habilidades

7. Crie uma classe chamada **HabilidadeViewModel** dentro da pasta Models com as propriedades Id, Nome e Dano, de forma similar ao que fizemos na API.

8. Crie um outro botão na view Index de personagens para redirecionar para o ataque usando habilidades

```
@Html.ActionLink("Clique aqui para um embate com habilidades!!!", "IndexHabilidades", "Disputas",
    null, new { @class = "btn btn-dark" })
```

9. Crie um método get na controller de disputas que será responsável por carregar a View do ataque com habilidades

```
[HttpGet]
0 references
public async Task<ActionResult> IndexHabilidadesAsync()
{
    try
    {
        //Programação aqui
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

10. Copie a programação de dentro do bloc try do método HttpGet Index (o primeiro feito na aula) e cole do corpo do método criado na etapa anterior, seguindo as orientações abaixo deste print

```
            ViewBag.ListaAtacantes = listaPersonagens;
            ViewBag.ListaOponentes = listaPersonagens;
A           return View(); //Remova esta linha
        }
        else
            throw new System.Exception(serialized);

        string uriBuscaHabilidades = "http://xyz.somee.com/RpgApi/PersonagemHabilidades/GetHabilidades";
        response = await httpClient.GetAsync(uriBuscaHabilidades);
        serialized = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
B       {
            List<HabilidadeViewModel> listaHabilidades = await Task.Run(() =>
                JsonConvert.DeserializeObject<List<HabilidadeViewModel>>(serialized));
            ViewBag.ListaHabilidades = listaHabilidades;
        }
        else
            throw new System.Exception(serialized);

C       return View("IndexHabilidades");
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

(A) Remova a linha sinalizada.
(B) Codificação que buscará a lista de habilidades e guardará na ViewBag
(C) Com a ViewBag carregada o usuário será direcionado para a View que criaremos a seguir

11. Dentro da pasta Views/Disputas crie o arquivo IndexHabilidades.cshtml. Copie todo o conteúdo do arquivo Views/Disputas/Index.cshtml e cole na view recém-criada.

12. Na view recém-criada (IndexHabilidades.cshtml) altere o título na tag h2 para “Ataque com Habilidade”. Altere também o texto do input.

```
<h2>Ataque com Habilidade</h2>
@using (Html.BeginForm())
{
```

13. Insira a codificação da seleção de ataques abaixo do DropDownList que contém os dados do atacante

```
<div class="form-group">
            @Html.DisplayName("Habilidade")
            <div class="col-md-6">
 @Html.DropDownListFor(model => model.HabilidadeId, new SelectList(@ViewBag.ListaHabilidades,
"Id", "Nome"), "---Selecione---", new { @class = "form-control" })</div></div>
```

14. Retorne para a controller de disputas e faça a criação do método HttpPost que enviará a disputa para a API.

```
[HttpPost]
0 references
public async Task<ActionResult> IndexHabilidadesAsync(DisputaViewModel disputa)
{
    try
    {
        HttpClient httpClient = new HttpClient();
        string uriComplementar = "Habilidade";
        var content = new StringContent(JsonConvert.SerializeObject(disputa));
        content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
        HttpResponseMessage response = await httpClient.PostAsync(uriBase + uriComplementar, content);
        string serialized = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            disputa = await Task.Run(() =>
            JsonConvert.DeserializeObject<DisputaViewModel>(serialized));
            TempData["Mensagem"] = disputa.Narracao;
            return RedirectToAction("Index", "Personagens");
        }
        else
            throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

- Teste selecionando o atacante, a habilidade ele tem e quem será o seu oponente.

15. Crie mais um botão na index de listagem de personagens

```
@Html.ActionLink("Clique aqui para um embate em grupo!!!", "DisputaGeral", "Disputas",
    null, new { @class = "btn btn-danger" })
```

16. Na controller de personagens crie o corpo do método que realizará uma disputa geral entre os personagens

```
[HttpGet]
0 references
public async Task<ActionResult> DisputaGeralAsync()
{
    try
    {
        //Próxima codificação aqui

        return RedirectToAction("Index","Personagens");
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index","Personagens");
    }
}
```

17. Programe no comentário do bloco try a primeira parte do método, onde carregaremos todos os personagens para uma lista

```
HttpClient httpClient = new HttpClient();

string token = HttpContext.Session.GetString("SessionTokenUsuario");
httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

string uriBuscaPersonagens = "http://xyz.somee.com/RpgApi/Personagens/GetAll";
HttpResponseMessage response = await httpClient.GetAsync(uriBuscaPersonagens);

string serialized = await response.Content.ReadAsStringAsync();

List<PersonagemViewModel> listaPersonagens = await Task.Run(() =>
    JsonConvert.DeserializeObject<List<PersonagemViewModel>>(serialized));
```

18. Na linha seguinte ao código anterior faremos a configuração para gerar as disputas. Confirme se será necessário using para System.Linq

```
    List<PersonagemViewModel> listaPersonagens = await Task.Run(() =>
        JsonConvert.DeserializeObject<List<PersonagemViewModel>>(serialized));

    string uriDisputa = "http://xyz.somee.com/RpgApi/Disputas/DisputaEmGrupo";
    DisputaViewModel disputa = new DisputaViewModel();
    disputa.ListaIdPersonagens = new List<int>();
    disputa.ListaIdPersonagens.AddRange(listaPersonagens.Select(p => p.Id));//using System.Linq

    var content = new StringContent(JsonConvert.SerializeObject(disputa));
    content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
    response = await httpClient.PostAsync(uriDisputa, content);

    serialized = await response.Content.ReadAsStringAsync();

    if (response.StatusCode == System.Net.HttpStatusCode.OK)
    {
        disputa = await Task.Run(() => JsonConvert.DeserializeObject<DisputaViewModel>(serialized));
        TempData["Mensagem"] = string.Join("<br/>", disputa.Resultados);
    }
    else
        throw new System.Exception(serialized);

    return RedirectToAction("Index", "Personagens");
}
catch (System.Exception ex)
{
```

- Execute o projeto e confirme que os embates serão realizados. O espero é que seja retornado todos os resultados, atualizando o número de disputas, vitórias e derrotas da lista de personagens.

19. Inclua um botão (na área de botões) para zerar o ranking e os pontos de vida de todos os personagens

```
@Html.ActionLink("Clique aqui para um embate em grupo!!!", "DisputaGeral", "Disputas",
null, new { @class = "btn btn-danger" })

@Html.ActionLink("Zerar ranking e restaurar pontos vida", "ZerarRankingRestaurarVidas", "Personagens",
null, new { @class = "btn btn-secondary", onclick = "return confirm('Deseja realmente realizar esta ação ?');" })
```

20. Abra a controller de personagens e adicione o método que vai zerar o ranking e restaurar os pontos de todos os personagens

```
[HttpGet]
        public async Task<ActionResult> ZerarRankingRestaurarVidasAsync()
        {
            try
            {
                HttpClient httpClient = new HttpClient();
                string token = HttpContext.Session.GetString("SessionTokenUsuario");
                httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
                string uriComplementar = "ZerarRankingRestaurarVidas";
    HttpResponseMessage response = await httpClient.PutAsync(uriBase + uriComplementar, null);
                string serialized = await response.Content.ReadAsStringAsync();
                if (response.StatusCode == System.Net.HttpStatusCode.OK)
                    TempData["Mensagem"] = "Rankings zerados e vidas dos personagens restauradas com
sucesso.";
                else
                    throw new System.Exception(serialized);
            }
            catch (System.Exception ex)
            { TempData["MensagemErro"] = ex.Message; }
            return RedirectToAction("Index");
        }
```