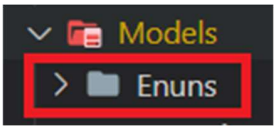
ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

DESENVOLVIMENTO DE SISTEMAS – 2023/1

Luiz Fernando Souza / Quitéria Danno

Aula 14 – Projeto Front-end MVC consumindo API – Parte 1: GetAll, Post e Get

1. Crie uma pasta chamada Enuns dentro da pasta Models



2. Dentro da pasta models crie uma classe chamada ClasseEnum. Após isso altere a palavra *class* por *public enum.* Dentro do corpo do enum programaremos os itens conforme abaixo

```
public enum ClasseEnum
{
    Cavaleiro = 1,
    Mago = 2,
    Clerigo = 3
}
```

3. Crie a classe **PersonagemViewModel.cs** dentro da pasta Models com as mesmas propriedades da classe Personagens do projeto de API. Comente as propriedades do tipo **Usuario, Arma** e **List<PersonagemHabilidade>**, além de adicionar o using para o enumeration (RpgMvc.Models.Enuns). Remova as notações *NotMapped* e *JsonIgonore* que vierem na cópia. Resumindo as propriedades para este momento: Id, Nome, Forca, PontosVida, Defesa, Inteligencia, Classe, FotoPersonagem, Disputas, Vitorias e Derrotas.

- Para o enum será necessário o using de RpgMvc.Models.Enums

4. Crie a classe **PersonagensController.cs** dentro da pasta Controllers. Faça a herança de *controller* conforme sinalizado

```
public class PersonagensController : Controller
```

- Requer o *using Microsoft.AspNetCore.Mvc*

5. Crie uma variável global para o endereço base da API

```
public class PersonagensController : Controller
{
    public string uriBase = "xyz/Personagens/";
    //xyz tem que ser substituído pelo nome do seu site da API.
```

6. Crie o método **IndexAsync** conforme abaixo, será comentado após a imagem os *usings* que serão necessários: Task → *System.Threading.Tasks,* AuthenticationHeaderValue → *System.Net.HttpHeaders* httpClient → *System.Net.Http,* List → *System.Collections.Generic,* PersonagemViewModel → *RpgMvc.Models,* JsonConvert → *Newtonsoft.Json,* Session → *Microsoft.AspNetCore.Http*

```
[HttpGet]
0 references
public async Task<ActionResult> IndexAsync()
{
    try
    {
      1 string uriComplementar = "GetAll";
      2 HttpClient httpClient = new HttpClient();
      3 string token = HttpContext.Session.GetString("SessionTokenUsuario");
      4 httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

      5 HttpResponseMessage response = await httpClient.GetAsync(uriBase + uriComplementar);
      6 string serialized = await response.Content.ReadAsStringAsync();

      7 if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            List<PersonagemViewModel> listaPersonagens = await Task.Run(() =>
                JsonConvert.DeserializeObject<List<PersonagemViewModel>>(serialized));

            return View(listaPersonagens);
        }
      8 else
            throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

(1) Variável *string* para conter o nome método para concatenar com o endereço base da *api*.
(2) Variável do tipo *http* que fará toda transição da requisição dos dados na web
(3) Token recuperado da Sessão para variável string
(4) Carregamento do cabeçalho (header) da requisição com o token
(5) Variável que vai guardar a resposta da requisição e que guardar várias informações
(6) Etapa em que o conteúdo da requisição (*json*) vai ser guardado em uma *string* para próxima etapa.
(7) Se retornar Ok (200) será feita uma desserialização do que era *string* para que vire um objeto. Neste caso está sendo transformado em uma lista de Personagens. A operação é bem-sucedida desde que os campos contidos no *json* tenham identificação parecida com os campos da classe Personagem.
(8) Se der erro cairá no else, e o que estiver na variável “serilalized” será uma mensagem de erro que será lançada como exceção.

---

7. Crie uma pasta chamada **Personagens** dentro da pasta Views e dentro desta pasta crie um arquivo chamado **Index.cshtml**. Tudo o que tem um @ na frente representa programação razor, ficando explicito que estamos querendo usar algo que está nas classes C# (.cs). O restante é o bom e velho *html* para formar o layout da página.

```
<!--Namespace da classe de Modelo para esta view-->
@model IEnumerable<RpgMvc.Models.PersonagemViewModel>

<!--Inclua os TempData para Sucesso e Erro aqui, conforme exemplo na view de Autenticação-->

@{ViewBag.Title = "Personagens"; }<!--Título da página para o navegador-->
<h2>Relação de Personagens</h2><!--Título da página-->
<p>    <!--Links apontando para views na mesma pasta-->
    @Html.ActionLink("Criar Novo Personagem", "Create")
</p>
<table class="table">
    <tr><!--Títulos das colunas da tabela-->
        <th>@Html.DisplayNameFor(model => model.Id)</th>
        <th>@Html.DisplayNameFor(model => model.Nome)</th>
        <th>@Html.DisplayNameFor(model => model.Classe)</th>
        <th>@Html.DisplayNameFor(model => model.PontosVida)</th>
        <th>@Html.DisplayNameFor(model => model.Disputas)</th>
        <th>@Html.DisplayNameFor(model => model.Vitorias)</th>
        <th>@Html.DisplayNameFor(model => model.Derrotas)</th>
        <th></th>
    </tr>
    <!--Looping para escrever os dados na tabela-->
    @foreach (var item in Model)
    {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.Id)</td>
            <td>@Html.DisplayFor(modelItem => item.Nome)</td>
            <td>@Html.DisplayFor(modelItem => item.Classe)</td>
            <td>@Html.DisplayFor(modelItem => item.PontosVida)</td>
            <td>@Html.DisplayFor(modelItem => item.Disputas )</td>
            <td>@Html.DisplayFor(modelItem => item.Vitorias)</td>
            <td>@Html.DisplayFor(modelItem => item.Derrotas)</td>
            <td><!--Coluna para Links/botões-->
                @Html.ActionLink("Editar", "Edit", new { id = item.Id } ) |
                @Html.ActionLink("Detalhes", "Details", new { id = item.Id }) |
                @Html.ActionLink("Deletar", "Delete", new { id = item.Id })
            </td>
        </tr>
    }
</table>
```

8. Insira na *view* **_Layout.cshtml,** que se encontra na pasta Views/Shared, um item de menu para a controller e view que criamos anteriormente. Depois execute o programa e navegue para a página de Personagens para certificar que os dados vão ser carregados.

```
<ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Personagens" asp-action="Index">Personagens</a>
    </li>
```

9. Retorne à c*ontroller* e crie um método *HttpPost* com o nome "**CreateAsync**". Esse método postará para a *Api* enviando um objeto serializado. MediaTypeHeaderValue requer o using *System.Net.Http.Headers*

```
[HttpPost]
0 references
public async Task<ActionResult> CreateAsync(PersonagemViewModel p)
{
    try
    {
      1 HttpClient httpClient = new HttpClient();
      2 string token = HttpContext.Session.GetString("SessionTokenUsuario");
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);

      3 var content = new StringContent(JsonConvert.SerializeObject(p));
      4 content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
      5 HttpResponseMessage response = await httpClient.PostAsync(uriBase, content);
      6 string serialized = await response.Content.ReadAsStringAsync();

      7 if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            TempData["Mensagem"] = string.Format("Personagem {0}, Id {1} salvo com sucesso!", p.Nome, serialized);
            return RedirectToAction("Index");
        }
      8 else
            throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Create");
    }
}
```

- Resumo das ações:

(1) Declaração de Objeto HttpClient responsável pelo tráfego de dados na internet.
(2) Declaração de varável para armazenar o token que está na string e passagem do token recuperado para a propriedade Authorization do objeto httpClient
(3) O objeto **p** está sendo *serializado*, ou seja, está sendo transformado numa cadeia de caracteres em série dentro da variável *content*, o famoso formato *json*.

(4) Está sendo informado no cabeçalho do conteúdo que ele é do tipo *json*.
(5) É declarada uma variável para armazenar o resultado da requisição que está sendo postada com o conteúdo serializado para o endereço base, já que o post o utiliza.
(6) O conteúdo da resposta da requisição é armazenado numa variável no formato de *string*.
(7) Sendo Ok (200) a mensagem será armazenada num TempData para que a View Index possa apresentar o conteúdo.
(8) Caso não seja Ok, lançará uma exceção que será guardada no TempData para a View Index exibir.

10. Retorne à c*ontroller* e crie um método *HttpGet* com o nome "**Create**". Esse método será acionado quando o usuário clicar em "Novo Personagem" e não exige nenhum parametro, apenas carregará a View.

```
[HttpGet]
0 references
public ActionResult Create()
{
    return View();
}
```

11. Crie uma *View* com o nome **Create.cshtml** na pasta Personagens e realize a programação a seguir. Execute e perceba que a *div* sinalizada simboliza a *Label* e uma caixa de texto que são visualizadas na tela. Use o exemplo para criar os outros campos: Pontos de vida, Força, Defesa, Inteligência e classe.

```
@model RpgMvc.Models.PersonagemViewModel
@{
    ViewBag.Title = "Novo Personagem";
}
<!-- Insira aqui os TempData de Sucesso e Erro para evitar tela da morte -->
<h2>Criar um Novo Personagem</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <hr />
        <div class="form-group">
            @Html.LabelFor(model => model.Nome, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-6">
                @Html.EditorFor(model => model.Nome, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-6">
                <input type="submit" value="Salvar" class="btn btn-primary" />
            </div>
        </div>
    </div>
}
<div>
    @Html.ActionLink("Retornar", "Index")
</div>
```

12. Volte à *controller* e crie um método *httpGet* chamado **DetailsAsync**

```
[HttpGet]
0 references
public async Task<ActionResult> DetailsAsync(int? id)
{
    try
    {
        HttpClient httpClient = new HttpClient();
        string token = HttpContext.Session.GetString("SessionTokenUsuario");
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
        HttpResponseMessage response = await httpClient.GetAsync(uriBase + id.ToString());
        string serialized = await response.Content.ReadAsStringAsync();

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

13. Crie uma *view* chamada **Details.cshtml** dentro da pasta View/Personagens

```
@model RpgMvc.Models.PersonagemViewModel
@{
    ViewBag.Title = "Detalhes do Personagem";
}
<h2>Detalhes do Personagem</h2>
<div>
    <hr />
    <dl class="dl-horizontal">
        <dt>@Html.DisplayNameFor(model => model.Id)</dt>
        <dd>@Html.DisplayFor(model => model.Id)</dd>

        <dt>@Html.DisplayNameFor(model => model.Nome)</dt>
        <dd>@Html.DisplayFor(model => model.Nome)</dd>
    </dl>
</div>
<p>
    @Html.ActionLink("Editar", "Edit", new { id = Model.Id}) |
    @Html.ActionLink("Retornar", "Index")
</p>
```

- Execute a aplicação e perceba que o trecho sinalizado exibe o título do campo e o conteúdo do campo, no caso “Nome” e o conteúdo do campo Nome, respectivamente. Utilize o modelo para apresentar os demais dados do personagem.

Luiz Fernando Souza / Quitéria Danno

Aula 14 - Projeto Front-end MVC consumindo API – Parte 2: Put e Delete

Concluiremos as *views* para que o projeto MVC possa consumir todos os métodos CRUD da API.

1. Crie o método Get com o nome **EditAsync**. Ele carregará a view de edição após pegar da API os dados do personagem.

```
[HttpGet]
0 references
public async Task<ActionResult> EditAsync(int? id)
{
    try
    {
        HttpClient httpClient = new HttpClient();
        string token = HttpContext.Session.GetString("SessionTokenUsuario");

        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
        HttpResponseMessage response = await httpClient.GetAsync(uriBase + id.ToString());

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

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

2. Crie o método Post com o mesmo nome **EditAsync**. Esse método enviará para a api os dados para atualização na base de dados.

```
[HttpPost]
0 references
public async Task<ActionResult> EditAsync(PersonagemViewModel p)
{
    try
    {
        HttpClient httpClient = new HttpClient();
        string token = HttpContext.Session.GetString("SessionTokenUsuario");

        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
        var content = new StringContent(JsonConvert.SerializeObject(p));
        content.Headers.ContentType = new MediaTypeHeaderValue("application/json");

        HttpResponseMessage response = await httpClient.PutAsync(uriBase, content);
        string serialized = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            TempData["Mensagem"] =
                string.Format("Personagem {0}, classe {1} atualizado com sucesso!", p.Nome, p.Classe);

            return RedirectToAction("Index");
        }
        else
            throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

3. Crie a view chamada **Edit.cshtml** na pasta Views/Personagens com o design abaixo

```
@model RpgMvc.Models.PersonagemViewModel
@{
    ViewBag.Title = "Editar Personagem";
}
<h2>Editar dados do Personagem</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <hr />
        <div class="form-group">
 @Html.LabelFor(model => model.Id, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-6">
@Html.EditorFor(model => model.Id, new { htmlAttributes = new { @class = "form-
control", @readonly = "readonly" } })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Nome, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-6">
       @Html.EditorFor(model => model.Nome, new { htmlAttributes = new { @class = "form-control" } })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-6">
                <input type="submit" value="Salvar alterações" class="btn btn-primary" />
            </div>
        </div>
    </div>
}
<div>
    @Html.ActionLink("Retornar", "Index")
</div>
```

- No exemplo temos apenas os campos de Id e Nome. Utilize o exemplo sinalizado para criar os demais campos presentes na classe Personagem.

- Execute o projeto e teste a edição de um personagem já salvo na base de dados.

4. Crie o método HttpGet **DeleteAsync** para remoção de um personagem

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

        HttpResponseMessage response = await httpClient.DeleteAsync(uriBase + id.ToString());
        string serialized = await response.Content.ReadAsStringAsync();

        if (response.StatusCode == System.Net.HttpStatusCode.OK)
        {
            TempData["Mensagem"] = string.Format("Personagem Id {0} removido com sucesso!", id);
            return RedirectToAction("Index");
        }
        else
            throw new System.Exception(serialized);
    }
    catch (System.Exception ex)
    {
        TempData["MensagemErro"] = ex.Message;
        return RedirectToAction("Index");
    }
}
```

5. Altere o link para remoção presente na view index.cshtml para exibir uma mensagem de confirmação da remoção.

```
<td>
    @Html.ActionLink("Editar", "Edit", new { id = item.Id } ) |
    @Html.ActionLink("Detalhes", "Details", new { id = item.Id }) |
    @Html.ActionLink("Deletar", "Delete", new { id = item.Id }
        , new { onclick = "return confirm('Deseja realmente deletar este personagem ?');"})
</td>
```

6. Altere a View Create.cshtml e Edit.cshtml para que ao invés de apresentar um campo EditorFor, seja apresentado um campo DropDownListFor

```
<div class="form-group">
    @Html.LabelFor(model => model.Classe, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-6">
        @Html.DropDownListFor(model => model.Classe , Html.GetEnumSelectList(typeof(RpgMvc.Models.Enuns.ClasseEnum)),
            "---Selecione---", new { @class = "form-control" })
    </div>
</div>
```

- Execute o projeto e confirme que o *DropDownList* está carregado nas Views Create e Edit através de um enumeration.

O que são TempData, ViewData e ViewBag?

TempData, *ViewData* e *ViewBag* são recursos que podemos utilizar para transitar dados de uma *controller* para uma v*iew* ou vice-versa, sendo que utilizaremos para trafegar mensagens entre estas camadas. Segue abaixo algumas referências teóricas sobre o tema que aplicaremos no projeto.

- https://www.eduardopires.net.br/2013/06/asp-net-mvc-viewdata-viewbag-tempdata/
- http://www.macoratti.net/15/06/mvc_conc1.htm
- https://pt.stackoverflow.com/questions/273504/quais-as-diferen%C3%A7as-entre-viewbag-viewdata-e-tempdata