Aula 06

Prof: Henrique Augusto Maltauro

Codificar Back-end de Aplicações Web

JSON

Na maioria das tecnologias, para trabalharmos com o JSON precisamos fazer um processo de serialização e deserialização.

Algumas tecnologias vão possuir um atalho para realizar esse processo, como o C# com o { get; set; } do atributos, mas mesmo assim vai existir outras formas mais precisas de trabalharmos com esse processo.

Mas primeiramente, vamos entender o que é serialização e deserialização.

JSON

- Serialização

A serialização é o processo de converter um objeto em um formato que possa ser armazenado ou transmitido por meio de uma conexão.

Dentro do contexto do JSON, é basicamente pegarmos um objeto e transformá-lo em um JSON.

JSON

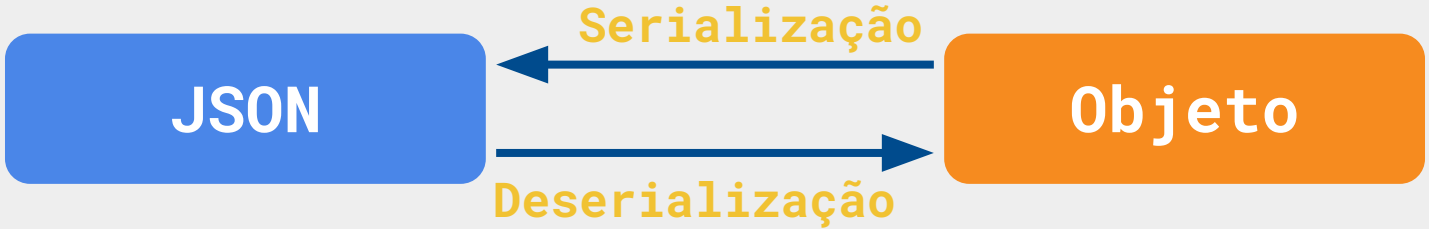
- Deserialização

A deserialização é o processo inverso, onde se converte esse outro formato em um objeto.

Dentro do contexto do JSON, é basicamente pegarmos um JSON e transformá-lo em um objeto.

JSON

- Serialização e Deserialização



JSON

- C#: Serialização e Deserialização

No C#, para trabalharmos com os processos de serialização e deserialização no JSON, basicamente precisamos definir cada um dos atributos dos nossos objetos como propriedades de JSON.

E para isso, nós fazemos uso de alguns atributos de configuração.

JSON

- C#: Serialização e Deserialização

O próprio C# já possui uma série de atributos de configuração para realizar esse processo. Contudo, existe um pacote externo que é bem mais completo e entrega algumas funcionalidades que os atributos de configuração que o C# já possui não entrega.

E para fazermos uso dele, precisamos primeiro entender o NuGet.

NuGet

NuGet

O NuGet é um gerenciador de pacotes, utilizado para compartilhar e gerenciar bibliotecas de código.

Ou seja, eu consigo instalar no meu projeto, funcionalidades que não são nativas do C# e que foram desenvolvidas por outros desenvolvedores.

Isso é feito com o intuito de facilitar alguns processos de desenvolvimento, uma vez que essas bibliotecas tendem a resolver algum problema de forma genérica.

NuGet

Lembrando que, a instalação e gerenciamento de pacotes é feita por projeto, e caso eu queira usar o mesmo pacote em mais de um projeto, eu preciso instalar nos dois projetos.

O acesso a esse processo é bem simples, basta clicar com o botão direito no projeto, e selecionar a opção de Gerenciar Pacotes NuGet:

JSON

- C#: Serialização e Deserialização

Para as nossas aulas, vamos instalar o pacote Newtonsoft.Json, que vai fornecer diversos atributos de configuração para trabalharmos com o processo de serialização e deserialização de diversas formas.

JSON

- C#: Serialização e Deserialização

Existem outros, mas vamos primeiramente focar em três atributos de configuração desse pacote:

- [JsonObject]
- [JsonProperty]
- [JsonIgnore]

JSON

- C#: Serialização e Deserialização([JsonObject])

O atributo de configuração [JsonObject], vai definir uma configuração para a classe/objeto, determinando uma regra para a serialização e deserialização dos seus atributos.

Com isso, ele vai receber um parâmetro do tipo MemberSerialization, que vai definir essa regra.

JSON

- C#: Serialização e Deserialização([JsonObject])

Esse MemberSerialization possibilita três opções diferentes de configuração:

- OptOut
- OptIn
- Fields

JSON

- C#: Serialização e Deserialização([JsonObject])

MemberSerialization.OptOut

- Todos os atributos públicos serão serializados e deserializados por padrão, exceto aqueles marcados como ignorados.

JSON

- C#: Serialização e Deserialização([JsonObject])

MemberSerialization.OptIn

- Somente os atributos marcados para serialização e deserialização serão considerados no processo.
- Será essa opção que faremos mais uso nas nossas aulas.

JSON

- C#: Serialização e Deserialização([JsonObject])

MemberSerialization.Fields

- Todos os atributos públicos e privados serão serializados e deserializados por padrão, exceto aqueles marcados como ignorados.

JSON

- C#: Serialização e Deserialização([JsonObject])

```
using Newtonsoft.Json;

[JsonObject(MemberSerialization.OptIn)]
public class Pessoa
{
    // Bloco de código da classe
}
```

JSON

- C#: Serialização e Deserialização([JsonProperty])

O atributo de configuração [JsonProperty], vai marcar cada atributo da classe/objeto como serializável e deserializável.

Ele também permite receber uma string de parâmetro que irá definir o nome daquela propriedade no JSON, pois podemos ter situações em que a propriedade do JSON tenha um nome diferente do atributo da nossa aplicação.

JSON

- C#: Serialização e Deserialização([JsonProperty])

```
using Newtonsoft.Json;

[JsonObject(MemberSerialization.OptIn)]
public class Pessoa
{
    [JsonProperty("id_pessoa")]
    public long Id;
}
```

JSON

- C#: Serialização e Deserialização([JsonIgnore])

E por fim, o atributo de configuração [JsonIgnore] é meio auto explicativo, ele vai marcar um atributo da classe/objeto como ignorado.

Ou seja, ele é totalmente desconsiderado na estrutura do JSON.

JSON

- C#: Serialização e Deserialização([JsonIgnore])

```
using Newtonsoft.Json;

[JsonObject(MemberSerialization.OptIn)]
public class Pessoa
{
    [JsonIgnore]
    public long Id;
    // Esse atributo é ignorado
    // na estrutura do JSON
}
```

JSON

- C#: Serialização e Deserialização

Por motivos de força maior, em outras palavras, não está funcionando do jeito esperado, vamos precisar fazer uso dos atributos de configuração nativos do C#.

JSON

- C#: Serialização e Deserialização([JsonPropertyName])

O atributo de configuração [JsonPropertyName], vai marcar cada atributo da classe/objeto como serializável e deserializável.

Ele também permite receber uma string de parâmetro que irá definir o nome daquela propriedade no JSON, pois podemos ter situações em que a propriedade do JSON tenha um nome diferente do atributo da nossa aplicação.

JSON

- C#: Serialização e Deserialização([JsonIgnore])

E por fim, o atributo de configuração [JsonIgnore] é meio auto explicativo, ele vai marcar um atributo da classe/objeto como ignorado.

Ou seja, ele é totalmente desconsiderado na estrutura do JSON.

Exercício

Exercício

1 - Implementar o [JsonPropertyName] nos atributos da classe Pessoa do exercício da aula 04, com nomes diferentes para cada um dos atributos.