04

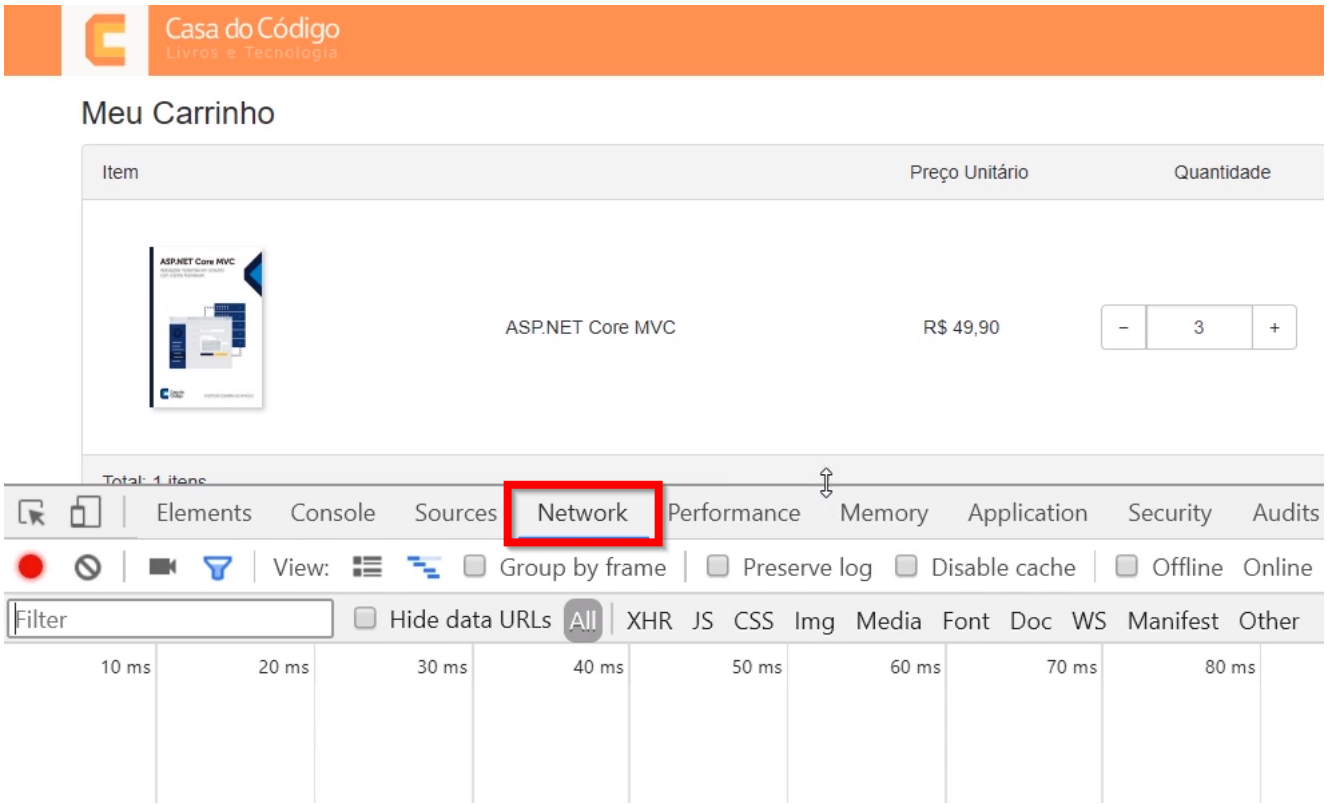
Protegendo a Action do AJAX Contra Ataques Maliciosos

Transcrição

Conseguimos proteger com sucesso a action de resumo contra o ataque de algum agente malicioso, mas ainda falta um outro ponto critico da aplicação: o método que recebe a requisição da chamada `ajax()` .

Quando clicamos nos botões "+" ou "-" para alterarmos a quantidade de itens no carrinho de compras, temos na `PedidoController.cs` o método `UpdateQuantidade()` que recebe essa requisição e faz a alteração no número de itens. Precisamos proteger esse método contra ataques externos. É importante que sempre validemos o token anti-falsificação. Antes disso, simularemos um ataque feito contra o método `UpdateQuantidade()` .

Na página de carrinho aberta no browser, pressionaremos o atalho "F12" e teremos acesso às ferramentas do desenvolvedor. Selecionaremos a aba "Network", desse modo poderemos ver todas as requisições feitas dentro da página.



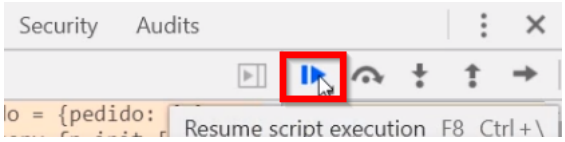
Clicaremos sobre o botão "+" do carrinho para verificarmos quais são as execuções da página e qual endereço será acessado. Somos direcionados para o método `UpdateQuantidade()` em `PedidoController.cs`

```
[Http]
public UpdadeQuantidadeResponse UpdateQuantidade([FromBody]ItemPedido itemPedido)
{
    return pedidoRepository.UpdateQuantidade(itemPedido);
}
```

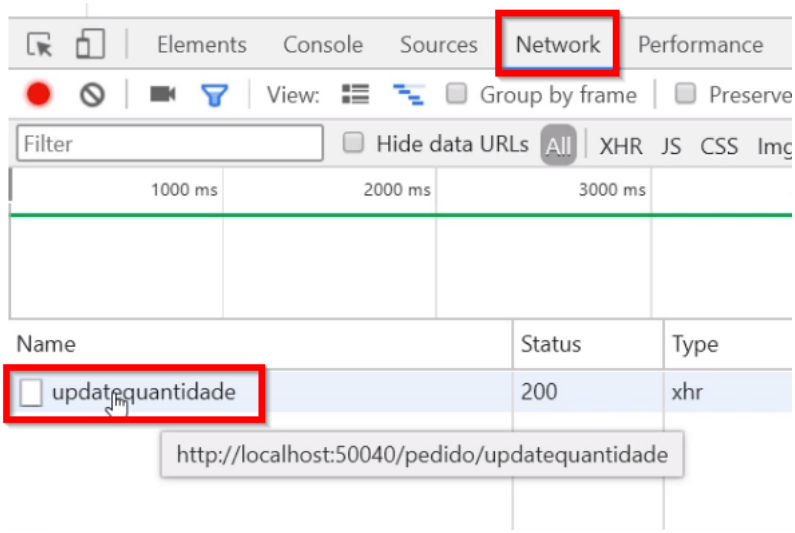
Pressionaremos "F5" e voltaremos ao navegador. Pressionaremos "F5" e voltaremos ao navegador. Verificaremos que a aplicação no cliente se interrompeu na linha `debugger` de `carrinho.js` .

```
<****!****>

 if (itemPedido.quantidade == 0) {
                linhaDoItem.remove();
            }

            debugger;
        });
```

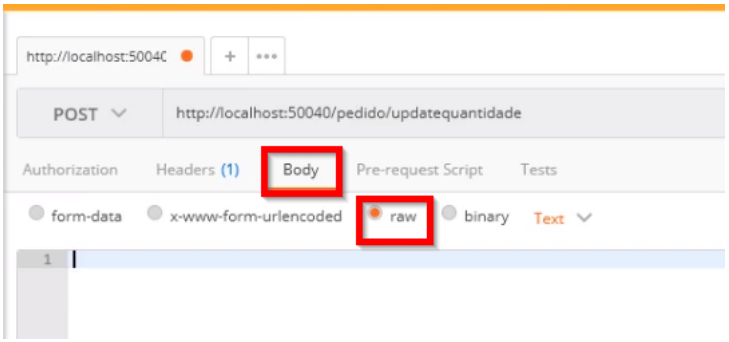
Na área de ferramentas do desenvolvedor, pressionaremos o botão "Resume Script Execution" para dar prosseguimento à execução do JavaScript.



Clicaremos sobre a aba "Network" , e encontraremos o endereço chamado, isto é, a URL de `updatequantidade` que é `http://localhost:50040/pedido/updatequantidade` . Clicaremos sobre a opção `updatequantidade` e copiraremos esta URL, para isso basta clicar com o botão direitos obre a opção e selecionar "copy > copy link address".



Voltaremos ao Postman e montaremos uma nova requisição, simulando um ataque de fora da aplicação. Modificaremos o método http para "POST" e colocaremos a URL no campo correspondente. Assim feito, selecionaremos as opções "Body" - que corresponde ao corpo da requisição - e depois marcaremos a opção "raw", que se fere ao texto puro a ser enviado.



Nesse texto puro o que será enviado? Será copiado do browser o `Request Payload` , que contém o texto enviado no corpo da requisição.

```
Request Payload
{Id: "41019", Quantidade: 4}
    Id: "41019""
    Quantidade:4
```

Copiaremos a linha `{Id: "41019", Quantidade: 4}` e colaremos no campo de texto do Postman. Assim feito, modicaremos a opção "Text" para "JSON(application/json)".

Desse modo nossa requisição está pronta e podemos pressionar o botão "Send".

Seremos direcionados à `PedidoController.cs` e veremos que a execução se interrompeu no break point, na linha `return pedidoRepository.UpdateQuantidade(itemPedido);` .

```
[HttpPost]
public UpdadeQuantidadeResponse UpdateQuantidade([FromBody]ItemPedido)
{
    return pedidoRepository.UpdateQuantidade(itemPedido);
}
```

Temos algumas informações passadas no parâmetro `ItemPedido` : o `Id` e a `Quantidade` . Voltaremos a requisição no Postman e alteraremos o número de `Quantidade` de `4` para `4000` e enviaremos a requisição.

```
{Id: "41019", Quantidade: 4000}
```

Na página no carrinho no browser, veremos que a quantidade de itens é `4000` e o valor total da compra é de `196600,00` . Percebemos que Postman está conseguindo acessar a nossa aplicação e nosso trabalho é impedir esse acesso indevido, aplicando um atributo de validação do token anti-falsificação (`ValidateAntiForgeryToken`) em `PedidoController` .

```
[HttpPost]
[`ValidateAntiForgeryToken`]
public UpdadeQuantidadeResponse UpdateQuantidade([FromBody]ItemPedido)
{
    return pedidoRepository.UpdateQuantidade(itemPedido);
}
```

Executaremos a aplicação no browser. Escolheremos um item qualquer e seremos direcionados para a página do carrinho. No Postman enviaremos novamente a requisição `{Id: "41019", Quantidade:"4000"}` . Dessa vez, o acesso é impedido e a requisição recebe o status de erro "400 Bad Request".

Precisamos pensar em como passar esse tolken por meio do carrinho, uma vez que nessa configuração, ao pressionarmos o botão de "+" a quantidade de itens não é alterada.

Pressionaremos o atalho "F12" para acessarmos as ferramentas do desenvolvedor. Feito isso, selecionaremos a aba "Console", e verificaremos um erro `Failed toload resource: the serve responded with a status of 400 (Bad Request)` .

Ao inspecionarmos os elementos da página de carrinho, não encontraremos o token necessário para realizar a validação, como havia na tela de cadastro.

O que precisamos fazer é fornecer o token para view `Carrinho.cshtml`. Logo depois do cabeçalho criaremos uma tag helper `<form>`, com a informação de método (`method`) que será o `post`. Inseriremos, ainda, a informação `asp-action`, ou seja, para onde o formulário irá enviar as informações, no caso inseriremos o próprio `carrinho`.

```
@{

    ViewData["Title"] = "Carrinho";
}
@model CarrinhoViewModel;

<h3>Meu Carrinho</h3>

<form method="post" asp-action="carrinho">
</form>
```

Não inseriremos mais nenhuma informação dentro do formulário, pois `<form>` foi criado apenas com o objetivo de fornecer para view o tolken anti-falsificação.

De volta à página de carrinho e pressionaremos a telca "F5". Inspecionaremos os elementos e procuraremos o elemento `<form>`. Veremos que ele carrega um elemento a mais que é oculto (*hidden*), mas carrega o valor do token, uma informação bastante importante.

```
<form method="post" action="/Pedido/carrinho/035">
input name="_RequestVerificationToken" type="hidden" value=
"CfDJ8DGA9YtwEdDkFJM71llQFN_JsCz06wU.Iuks-
U_8Wgi7XM8GoelLqoLkfp12_XS_4UT19AKL2dlwodoifFSFssdemmskAI92femsowdaqwb_DpC8mFYjdsIWfzjfe
iEFIFlWOX2oDCVEFEUHWKDEI3rfjzcxfi2g"> == $0
</form>
```

Enviaremos o tolken para o servidor. Para isso, iremos para o arquivo `carrinho.js` e procuraremos a requisição `ajax()`, pois nesse monto do código nós montaremos um objeto que conterá um cabeçalho onde iremos inserir o token.

Escreveremos: `let headers`, e este objeto de cabeçalho será vazio, e atribuiremos a ele uma chave `RequestVerificationToken`. Passaremos o `token`. Em seguida, criaremos uma variável `token` que será alimentada com o conteúdo do input que vimos anteriormente, cujo nome é `RequestVerificationToken`. O obteremos por meio de um seletor do JQuery, por isso inseriremos o caractere `$`. Resta incluirmos o nome do atributo, que será `name`.

```
postQuantidade(Data) {

let token = $('[name=__RequestVerificationToken]').val()

let headers = {};
headers['RequestVerificationToken'] = token;

    $ajax({
        url:'/pedido/updatequantidade',
        type: 'POST',
        data: JSON.stringify(data)
    })done(function (response) {

<****!****>
```

Em seguida, devemos acessar o valor do token, para isso incluiremos a função do JQuery `val()`. Desse modo pegaremos o `token`, o passaremos para o objeto `headers` e atribuiremos este objeto abaixo da linha `data: JSON.stringify(data)`.

```
postQuantidade(Data) {

let token = $('[name=__RequestVerificationToken]')

let headers = {};
headers['RequestVerificationToken'] = token;

    $ajax({
        url:'/pedido/updatequantidade',
        type: 'POST',
        data: JSON.stringify(data),
        headers: headers
    })done(function (response) {

<****!****>
```

Salvaremos as modificações e voltaremos a página de carrinho. Clicaremos sobre o botão "+" e verificaremos que as quantidades do item são alteradas e o método `UpdadateQuantidade()` consegue ser acessado. Dessa forma, nossa aplicação está mais segura e funcional.