Creating the pagination component

Now that we have a working mechanism to fetch the data, let's implement the actual pagination itself. Let's start by creating a new Pagination.razor template in the Components folder... and a partial class as well and call it Pagination.razor.cs. After creation, we are going to modify the class first… We need the MetaData parameter; we need a parameter called Spread… and we are going to create a parameter called SelectedPage which is actually an EventCallback… We know what MetaData is for but what is Spread for? Spread defines how many pagination buttons we see before and after the currently selected page button. We'll see how it impacts our implementation later. EventCallback SelectedPage is there to help us execute the method from the parent component. In this case, we'll be executing a method from the Products component every time we select a new page in this component. We also need a list of links to populate since we want to navigate from one page to another... So, let's create a new PagingLink class in the Features folder… We need four properties in total. First, we need a Text property… we are going to use the Text property for every button text in the pagination component (Previous, Next, 1,2,3…). Then we need a Page property that will hold the value of the current page. Finally, we are going to use the Enabled... and Active properties... to decide whether to add the disabled and active CSS classes to the pagination button. We are also going to create a constructor to make the initialization easier... Now, let's get back to the Pagination class and add the links. First, we want to create a private list of links… and then we want to populate these links in the OnParametersSet method… Remember, this method triggers whenever the component receives a parameter input. We're just going to call the CreatePaginationLinks method in here… and then create it outside… In the method itself, we want to initialize the links first… and then add the "Previous" link… we're going to set the Page to current page minus one… it will be enabled if the current page has a previous page, and the text will just be "Previous"... Now we need to add all the links in between the Previous and the Next buttons… We're going to create a simple loop… and iterate as many times as there are pages… and create the links for values that are in the range of the Spread. For example, if the current page is 3, and the spread is 1, we are going to generate the links for pages 2,3, and 4. These links are always active, and we can use our index to populate their page and text. Finally, we're going to add the "Next" link too... For the Next link, we need to use the current page plus one, check the HasNext value, and call it "Next". We're also going to implement the OnSelectedPage method that has one parameter, the PagingLink link. In the

method body, we're going to check if the Page from that link is the same as the current page from the MetaData, or if the link is disabled… in that case, we're just going to exit. In case it's not, we're going to set the current page to the link page, and then invoke the event callback with the page number. Now that we've implemented the logic behind the scenes, we want to display our component. Let's head to the Pagination.razor component… and then we want to create a nav element that is used to define a set of navigation links… and then we need a list itself… and inside it, we're going to loop through all the links… and create a list element… each element has an @onclick event that will call our OnSelectedPage method... We're going to change the cursor style for each link to the pointer so we can see that those are indeed the links when we hover the cursor over them… and we're going to create some classes conditionally… first, we'll add the disabled class... if the link.Enabled property is false, and we'll add the "active" class... if the link.Active property is true. We also need a simple text span that contains the link text… And now we need to add the Pagination component to the Products component… And we're going to pass the MetaData parameter... hardcode Spread to 1... and we're going to set the SelectedPage to the SelectedPage local method... The only thing left now is to create a local SelectedPage method which will get called when the EventCallback from the Pagination component triggers… So, let's go to the Products class and define the SelectedPage method… it has one parameter, page… inside the method we'll set the page number to that page… and we'll call GetProducts… We also need to define the GetProducts method… and we want to move our logic from the OnInitializedAsync to this method… and then call it in the OnInitializedAsync too... Excellent... Now, let's start our client application… if the server is not started, start it... and inspect the result... And there we go; we can see that everything works as we want it to. We can go to the next page… and then back… We can see that the previous and next buttons get grayed out when we're on the first and the last pages… Fantastic…