100 XP

Bind controls to data in Blazor applications

7 minutes

Blazor lets you bind HTML controls to properties so that changing values are automatically displayed in the user interface (UI).

Suppose you're developing a page that collects information from customers about their pizza preferences. You want to load the information from a database and enable customers to make changes, such as recording their favorite toppings. When there's a change from the user or an update in the database, you want the new values to display in the UI as quickly as possible.

In this unit, you'll learn how to use data binding in Blazor to tie UI elements to data values, properties, or expressions.

What is data binding?

If you want an HTML element to display a value, you can write code to alter the display. You'll need to write extra code to update the display when the value changes. In Blazor, you can use data binding to connect an HTML element to a field, property, or expression. This way, when the value changes, the HTML element is automatically updated. The update usually happens quickly after the change, and you don't have to write any update code.

To bind a control, you would use the `@bind` directive:

razor

```razor
@page "/"

<p>
    Your email address is:
    <input @bind="customerEmail" />
</p>

@code {
    private string customerEmail = "user@contoso.com"
}
```

In the preceding page, whenever the `customerEmail` variable changes its value, the `<input>` value updates.

Note

Controls, such as `<input>`, update their display only when the component is rendered and not when a field's value changes. Because Blazor components render after any event handler code executes, in practice, updates are usually displayed quickly.

Bind elements to specific events

The `@bind` directive is smart and understands the controls it uses. For example, when you bind a value to a textbox `<input>`, it binds the `value` attribute. An HTML checkbox `<input>` has a `checked` attribute instead of a `value` attribute. The `@bind` attribute automatically uses this `checked` attribute instead. By default, the control is bound to the DOM `onchange` event. For example, consider this page:

razor

```razor
@page "/"

<h1>My favorite pizza is: @favPizza</h1>

<p>
    Enter your favorite pizza:
    <input @bind="favPizza" />
</p>

@code {
    private string favPizza { get; set; } = "Margherita"
}
```

When the page is rendered, the default value **Margherita** is displayed in both the `<h1>` element and the textbox. When you enter a new favorite pizza in the textbox, the `<h1>` element doesn't change until you tab out of the textbox or select `Enter` because that's when the `onchange` DOM event fires.

Often, that's the behavior you want. But suppose you want the `<h1>` element to update as soon as you enter any character in the textbox. You can achieve this outcome by binding to the `oninput` DOM event instead. To bind to this event, you must use the `@bind-value` and `@bind-value:event` directives:

razor

```razor
@page "/"

<h1>My favorite pizza is: @favPizza</h1>

<p>
    Enter your favorite pizza:
    <input @bind-value="favPizza" @bind-value:event="oninput" />
</p>

@code {
    private string favPizza { get; set; } = "Margherita"
}
```

In this case, the title changes as soon as you type any character in the textbox.

Format bound values

If you display dates to the user, you might want to use a localized data format. For example, suppose you write a page specifically for UK users, who prefer to write dates with the day first. You can use the `@bind:format` directive to specify a single date format string:

razor

```razor
@page "/ukbirthdaypizza"

<h1>Order a pizza for your birthday!</h1>

<p>
    Enter your birth date:
    <input @bind="birthdate" @bind:format="dd-MM-yyyy" />
</p>
```

```razor
@code {
    private DateTime birthdate { get; set; } = new(2000, 1, 1);
}
```

ⓘ Note

At the time of writing, format strings are only supported with date values. Currency formats, number formats, and other formats might be added in the future. To check the latest information on binding formats, see Format strings in the Blazor documentation.

As an alternative to using the `@bind:format` directive, you can write C# code to format a bound value. Use the `get` and `set` accessors in the member definition, as in this example:

razor

```razor
@page "/pizzaapproval"
@using System.Globalization

<h1>Pizza: @PizzaName</h1>

<p>Approval rating: @approvalRating</p>

<p>
    <label>
        Set a new approval rating:
        <input @bind="ApprovalRating" />
    </label>
</p>

@code {
    private decimal approvalRating = 1.0;
    private NumberStyles style = NumberStyles.AllowDecimalPoint | NumberStyles.AllowLeadingSign;
    private CultureInfo culture = CultureInfo.CreateSpecificCulture("en-US");

    private string ApprovalRating
    {
        get => approvalRating.ToString("0.000", culture);
        set
        {
            if (Decimal.TryParse(value, style, culture, out var number))
            {
                approvalRating = Math.Round(number, 3);
            }
        }
    }
}
```

In the next unit, you'll apply what you've learned.

Next unit: Exercise - Bind controls to data in Blazor applications

Continue ›