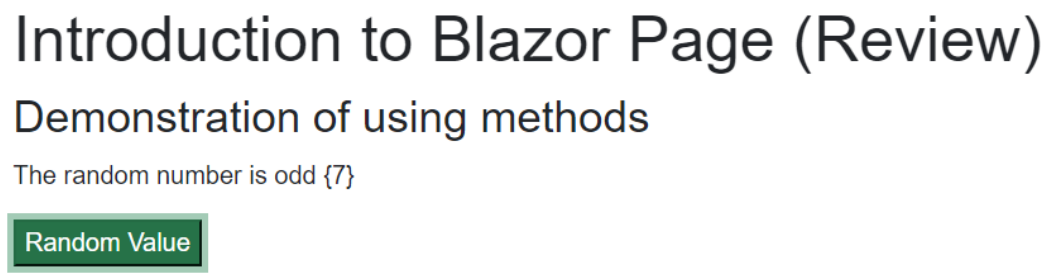
Session 3 - Building a Blazor Application with Simple Controls

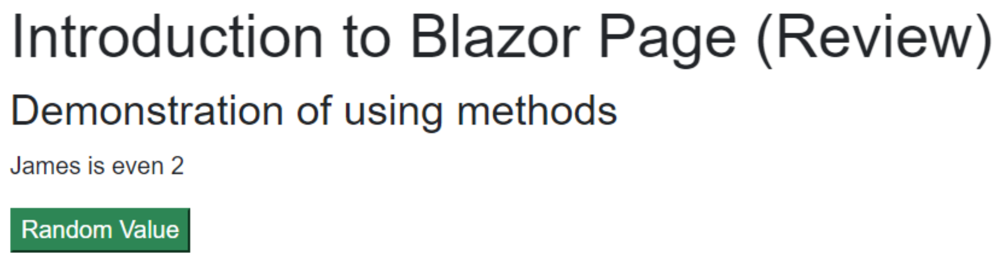
Demonstration of using Methods

- Conditional rendering based on values.
- Button to trigger the method call.


Introduction to Blazor Page (Review)

Demonstration of using methods

The random number is odd {7}


Introduction to Blazor Page (Review)

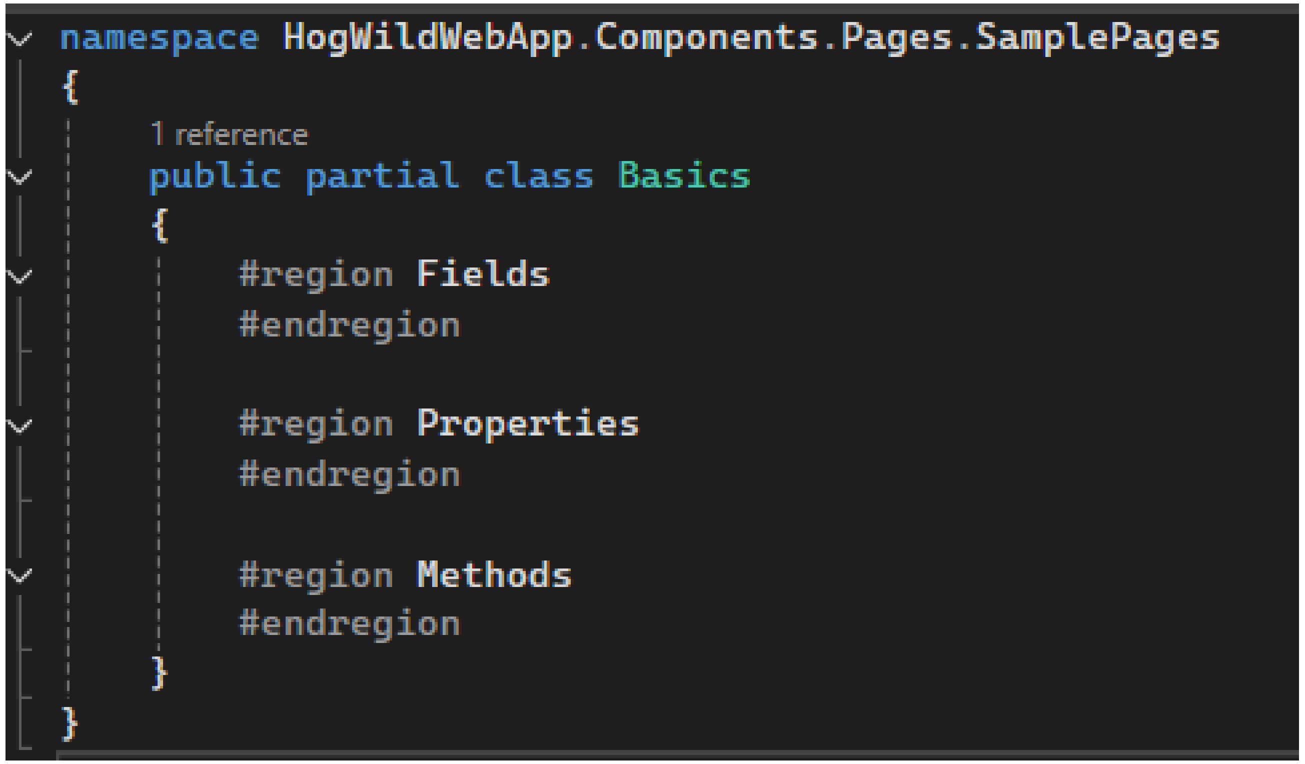
Demonstration of using methods

James is even 2

Setting up Regions

- It is best practice to include #regions within your code. #regions allows for clean organization
- Create Three (3) initial regions for any code behind .cs page.
    - **Fields** – Holds private variables used on the page
    - **Properties** – Holds Public or read-only (get) properties for use on the page
    - **Methods** – Holds private and/or public methods that can be used by the page

Basics.razor.cs

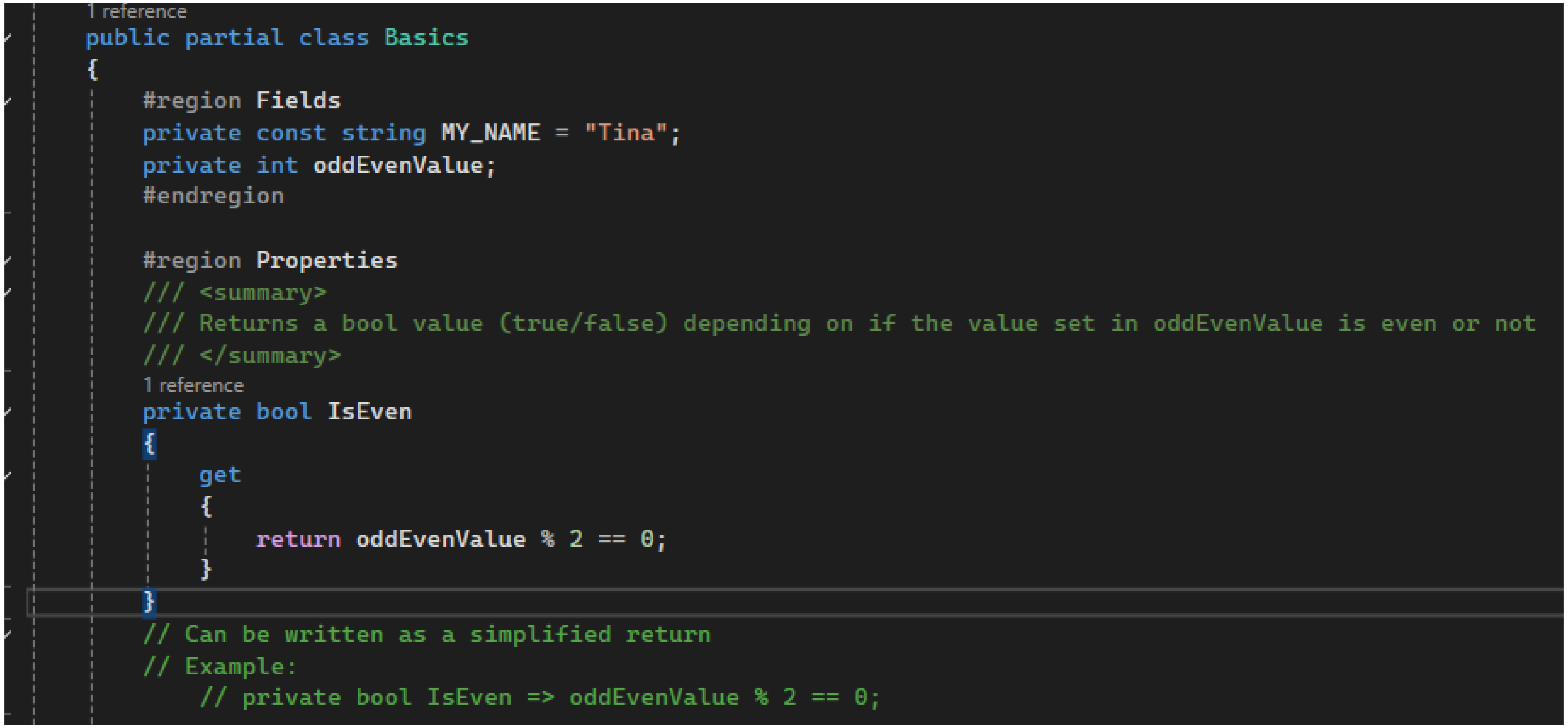


```
namespace HogWildWebApp.Components.Pages.SamplePages
{
    1 reference
    public partial class Basics
    {
        #region Fields
        #endregion

        #region Properties
        #endregion

        #region Methods
        #endregion
    }
}
```

Random Number Generation - Setup

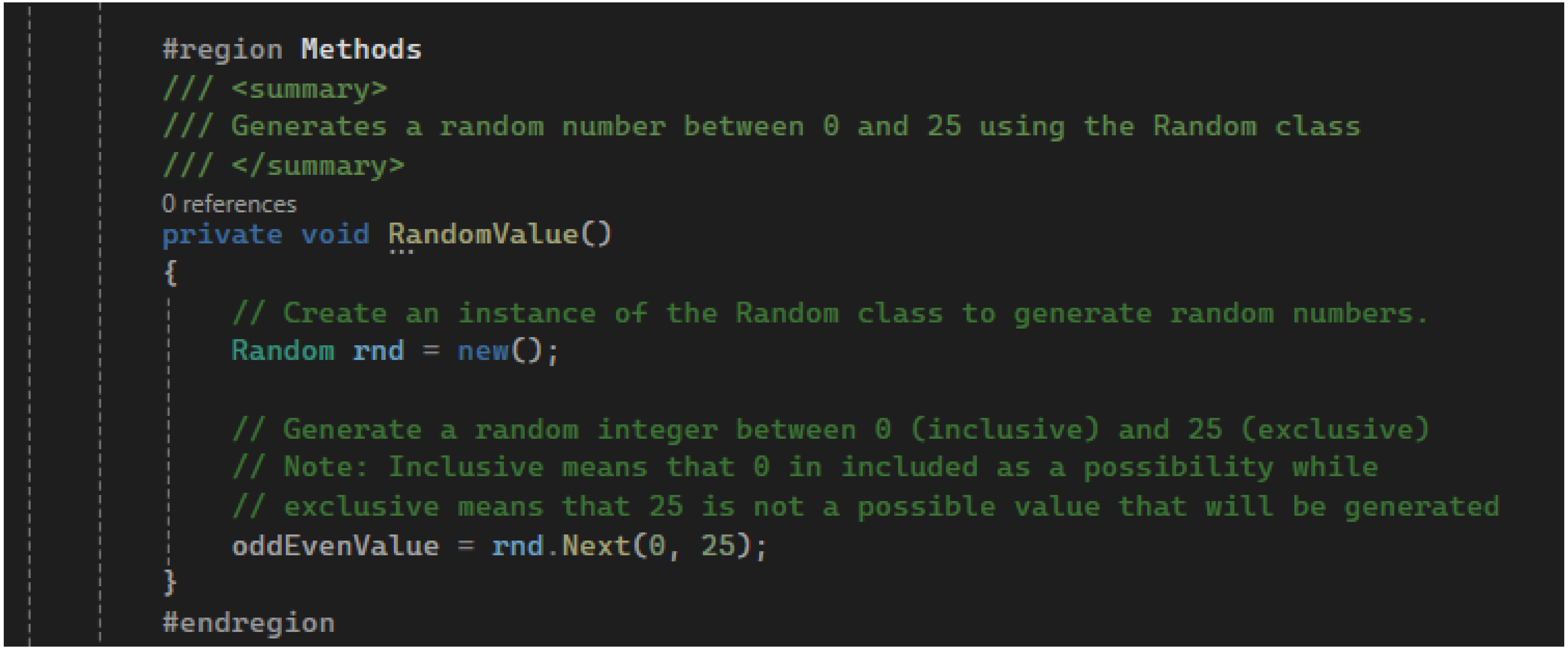
- This code generates a random number between 0 and 25, checks if it's even, sets the myName variable accordingly, and then triggers a component state update.
- Create a field to store the generated number
- Create a property to determine in the generated value is Even or Odd
- Create a Constant value to store your name
  - Please change this to [your name]

Basics.razor.cs



```
1 reference
public partial class Basics
{
    #region Fields
    private const string MY_NAME = "Tina";
    private int oddEvenValue;
    #endregion

    #region Properties
    /// <summary>
    /// Returns a bool value (true/false) depending on if the value set in oddEvenValue is even or not
    /// </summary>
    1 reference
    private bool IsEven
    {
        get
        {
            return oddEvenValue % 2 == 0;
        }
    }
    // Can be written as a simplified return
    // Example:
        // private bool IsEven => oddEvenValue % 2 == 0;
```

Basics.razor.cs



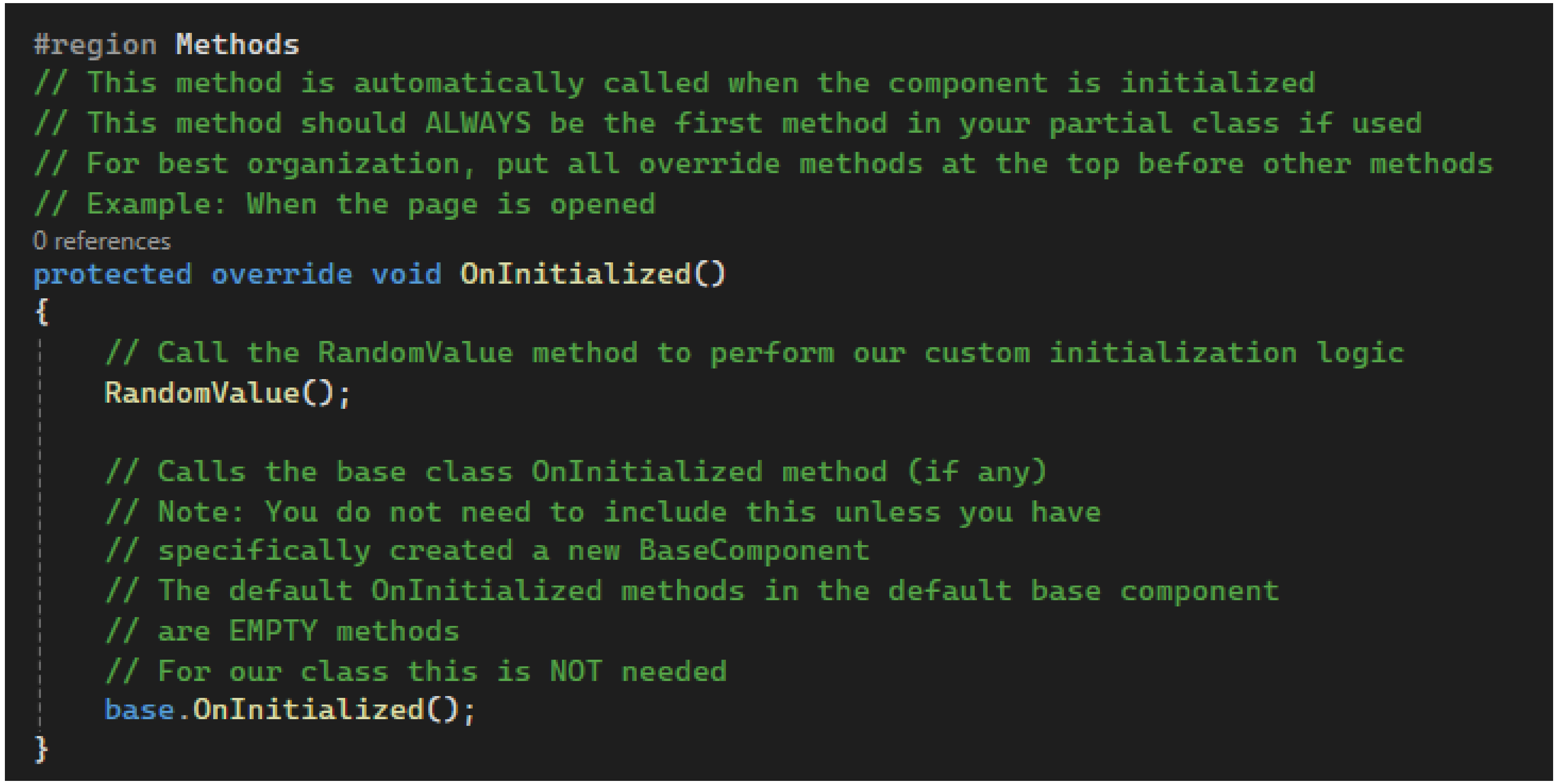
```
#region Methods
/// <summary>
/// Generates a random number between 0 and 25 using the Random class
/// </summary>
0 references
private void RandomValue()
{
    // Create an instance of the Random class to generate random numbers.
    Random rnd = new();

    // Generate a random integer between 0 (inclusive) and 25 (exclusive)
    // Note: Inclusive means that 0 in included as a possibility while
    // exclusive means that 25 is not a possible value that will be generated
    oddEvenValue = rnd.Next(0, 25);
}
#endregion
```

“Random Value” method

OnInitialized() override – Basics.razor.cs

- The **OnInitialized()** method is critical to a Blazor component's lifecycle. It allows you to execute custom initialization logic while ensuring that any base class initialization is not overridden but extended. It can be asynchronous as well by using **async Task OnInitializedAsync()** which enables non-blocking operations, which is particularly useful in web applications where responsiveness is crucial or when you need to await another async method.
- This is like the constructor of a C# class.

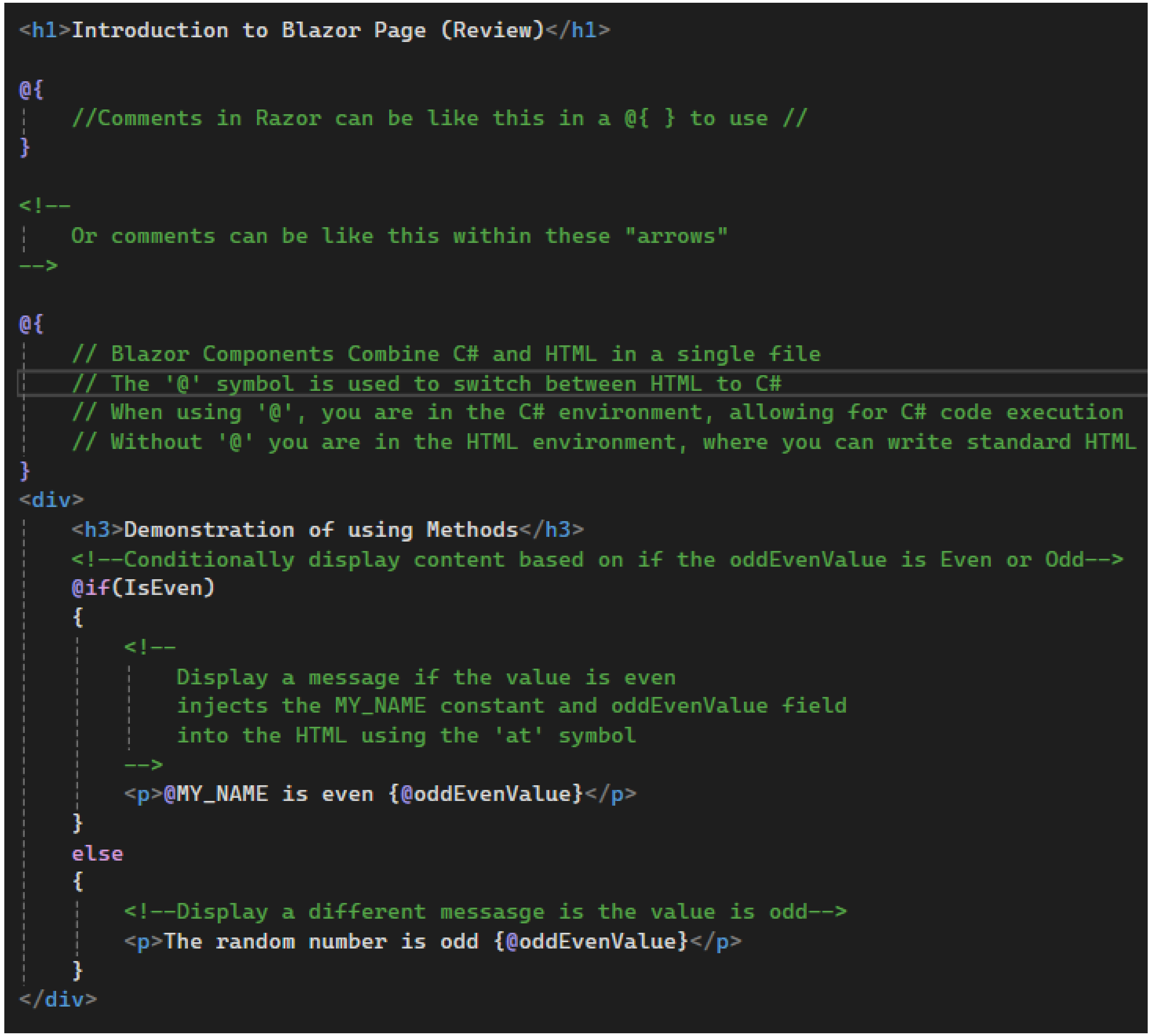
Basics.razor.cs



```
#region Methods
// This method is automatically called when the component is initialized
// This method should ALWAYS be the first method in your partial class if used
// For best organization, put all override methods at the top before other methods
// Example: When the page is opened
0 references
protected override void OnInitialized()
{
    // Call the RandomValue method to perform our custom initialization logic
    RandomValue();

    // Calls the base class OnInitialized method (if any)
    // Note: You do not need to include this unless you have
    // specifically created a new BaseComponent
    // The default OnInitialized methods in the default base component
    // are EMPTY methods
    // For our class this is NOT needed
    base.OnInitialized();
}
```

Updating the Basics.razor Page

Basics.razor

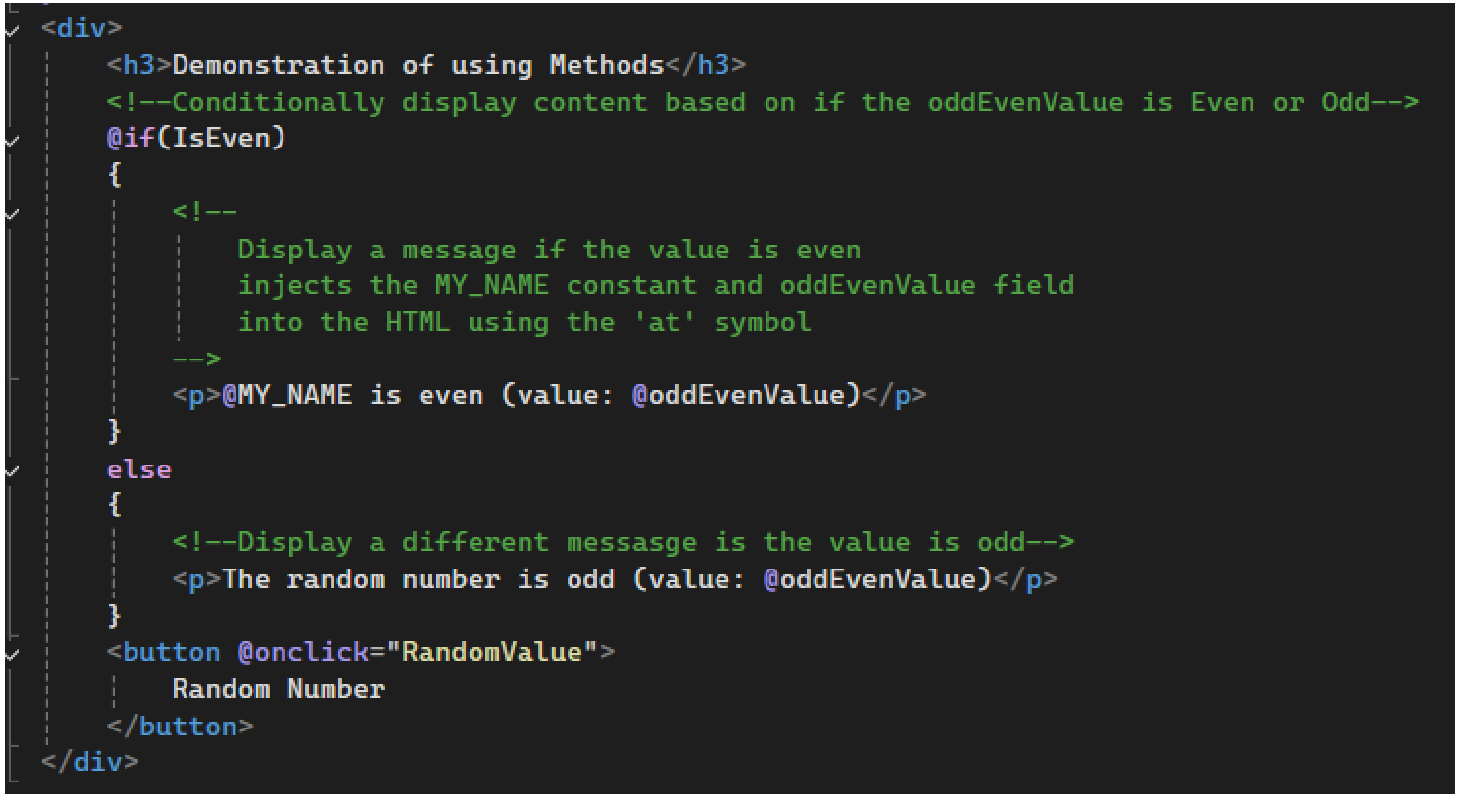


```
<h1>Introduction to Blazor Page (Review)</h1>

@{
    //Comments in Razor can be like this in a @{ } to use //
}

<!--
    Or comments can be like this within these "arrows"
-->

@{
    // Blazor Components Combine C# and HTML in a single file
    // The '@' symbol is used to switch between HTML to C#
    // When using '@', you are in the C# environment, allowing for C# code execution
    // Without '@' you are in the HTML environment, where you can write standard HTML
}
<div>
    <h3>Demonstration of using Methods</h3>
    <!--Conditionally display content based on if the oddEvenValue is Even or Odd-->
    @if(IsEven)
    {
        <!--
            Display a message if the value is even
            injects the MY_NAME constant and oddEvenValue field
            into the HTML using the 'at' symbol
        -->
        <p>@MY_NAME is even {@oddEvenValue}</p>
    }
    else
    {
        <!--Display a different messasge is the value is odd-->
        <p>The random number is odd {@oddEvenValue}</p>
    }
</div>
```

Button to call Method

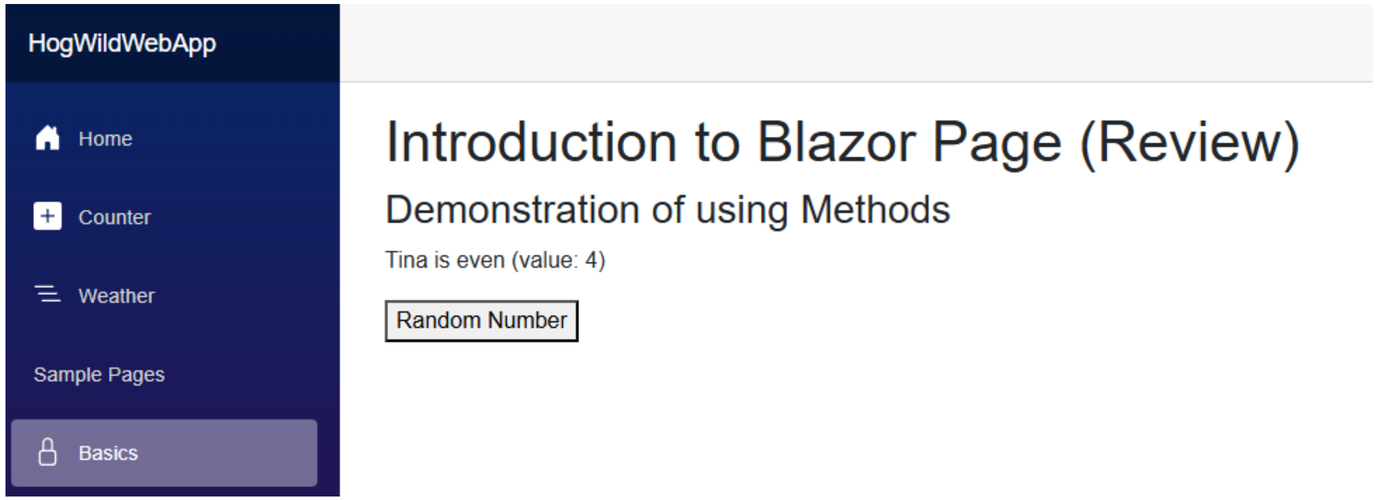
- Add a button to the **<div>** that has an @onclick that calls our RandomValue method to update the value and message.

Basics.razor



```
<div>
    <h3>Demonstration of using Methods</h3>
    <!--Conditionally display content based on if the oddEvenValue is Even or Odd-->
    @if(IsEven)
    {
        <!--
            Display a message if the value is even
            injects the MY_NAME constant and oddEvenValue field
            into the HTML using the 'at' symbol
        -->
        <p>@MY_NAME is even (value: @oddEvenValue)</p>
    }
    else
    {
        <!--Display a different messasge is the value is odd-->
        <p>The random number is odd (value: @oddEvenValue)</p>
    }
    <button @onclick="RandomValue">
        Random Number
    </button>
</div>
```

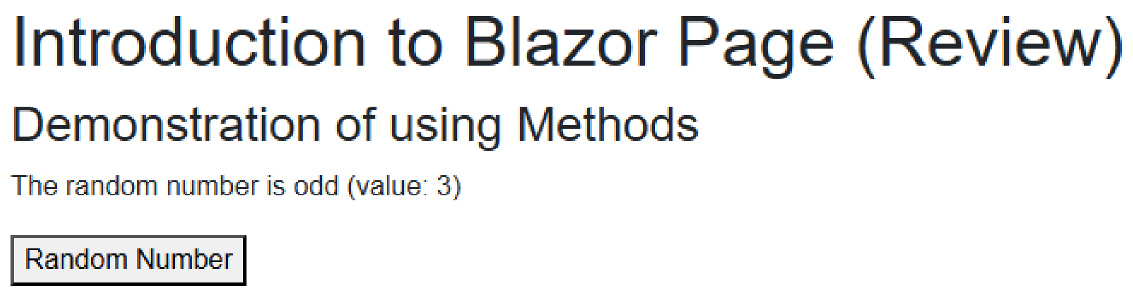
Blazor Page Output



Introduction to Blazor Page (Review)

Demonstration of using Methods

Tina is even (value: 4)



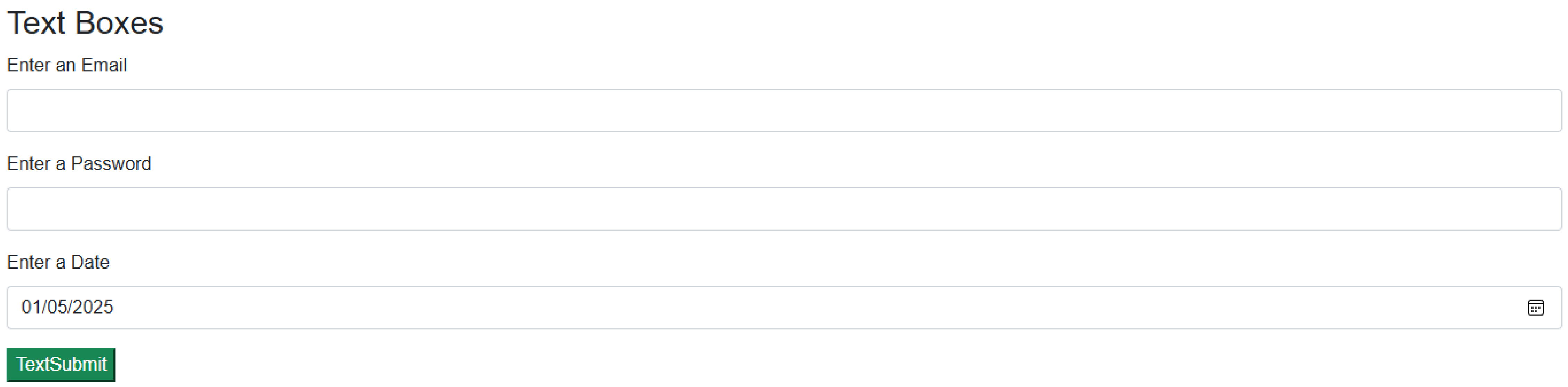
Introduction to Blazor Page (Review)

Demonstration of using Methods

The random number is odd (value: 3)

TextBoxes

- Demonstration of text input for email, password, and date.
- Button to trigger a method to update a feedback value



Text Boxes

Enter an Email

Enter a Password

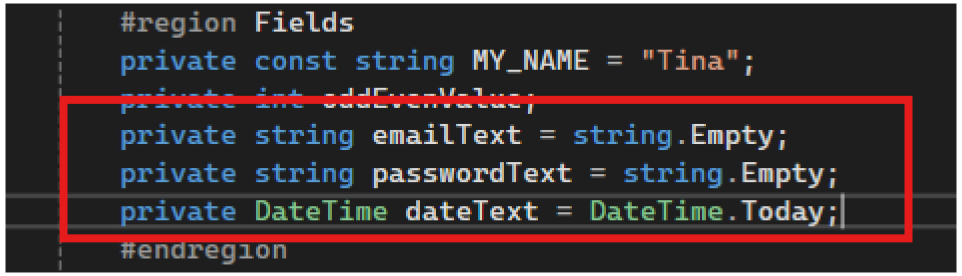
Enter a Date

01/05/2025

Fields for Holding Text Boxes Values

- **NOTE:** All variable names end with "Text" as a reference for the "Text Boxes." Do not include the "Data Type" in the name when creating variable names. Your names should look like the naming structure below.
- String values should be initialized to string.Empty not "".
  - string.Empty is less memory than an actual blank string.

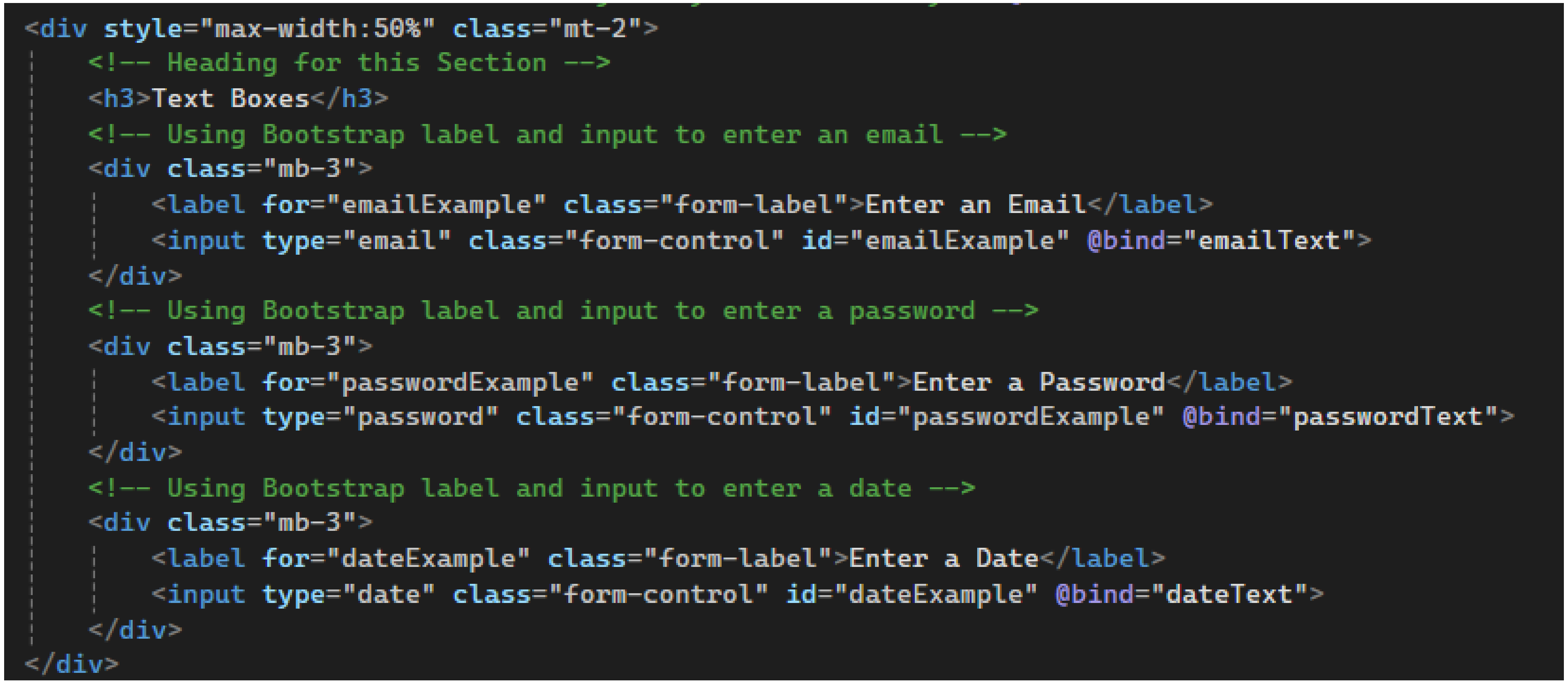
Basics.razor.cs



```
#region Fields
private const string MY_NAME = "Tina";
private int addEvenValue;
private string emailText = string.Empty;
private string passwordText = string.Empty;
private DateTime dateText = DateTime.Today;
#endregion
```

Updating the Basics.razor Page

Basics.razor

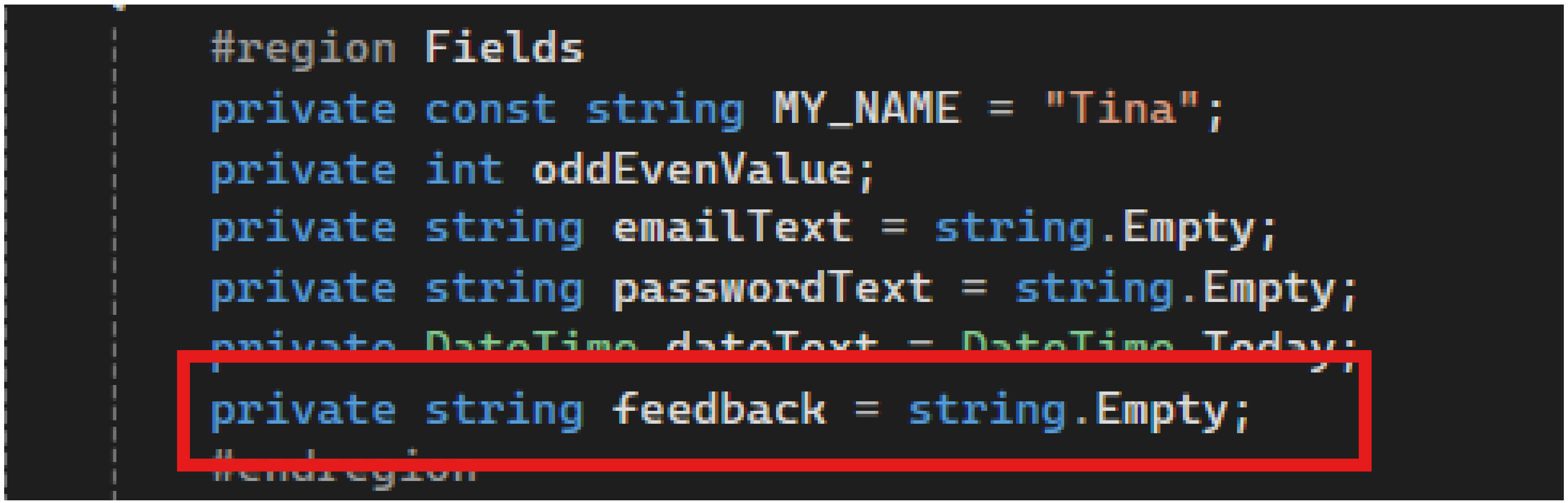


```
<div style="max-width:50%" class="mt-2">
    <!-- Heading for this Section -->
    <h3>Text Boxes</h3>
    <!-- Using Bootstrap label and input to enter an email -->
    <div class="mb-3">
        <label for="emailExample" class="form-label">Enter an Email</label>
        <input type="email" class="form-control" id="emailExample" @bind="emailText">
    </div>
    <!-- Using Bootstrap label and input to enter a password -->
    <div class="mb-3">
        <label for="passwordExample" class="form-label">Enter a Password</label>
        <input type="password" class="form-control" id="passwordExample" @bind="passwordText">
    </div>
    <!-- Using Bootstrap label and input to enter a date -->
    <div class="mb-3">
        <label for="dateExample" class="form-label">Enter a Date</label>
        <input type="date" class="form-control" id="dateExample" @bind="dateText">
    </div>
</div>
```

Updating the Basics.razor.cs Page

- Note: We will need to add a field to hold the “Feed Back“ in the code behind.

Basics.razor.cs

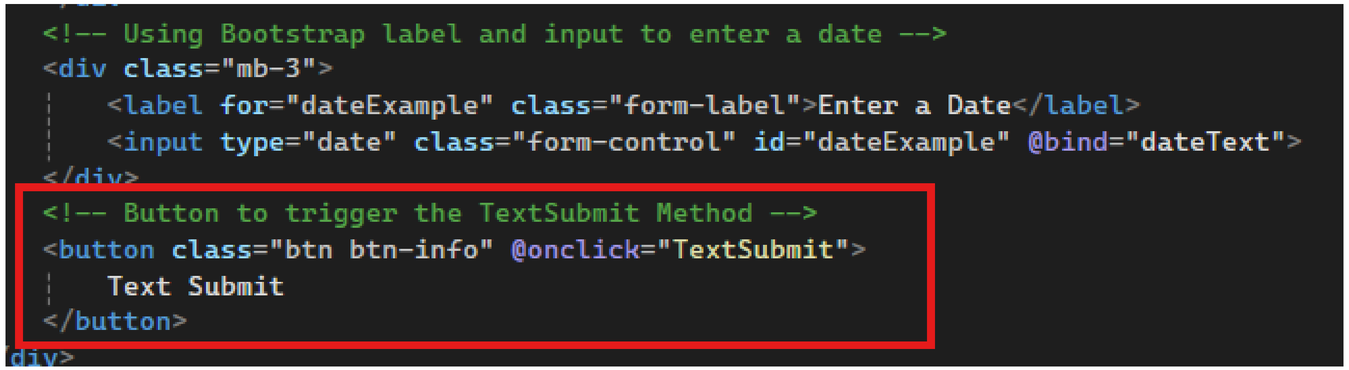


```
#region Fields
private const string MY_NAME = "Tina";
private int oddEvenValue;
private string emailText = string.Empty;
private string passwordText = string.Empty;
private DateTime dateText = DateTime.Today;
private string feedback = string.Empty;
#endregion
```

Coding the TextSubmit Method and Button

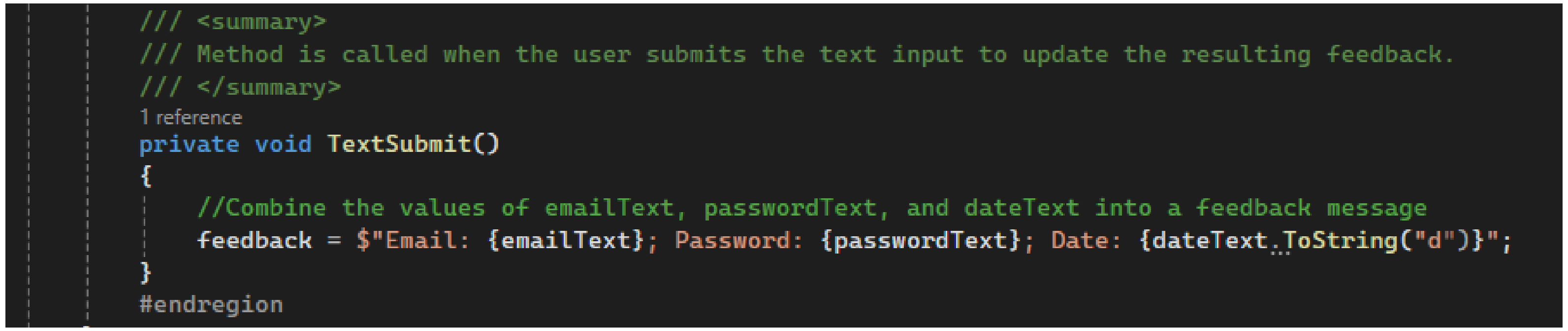
- NOTE: "Submit" is just a method name and not part of the Blazor Framework. This method could be called anything.
- Bootstrap Button Classes:
  - btn – required to control the size and general appearance
  - btn-info – makes the button the default info color
    - **Note:** If you want colors can be adjusted for Bootstrap, reference Bootstrap documentation (https://getbootstrap.com/docs/5.1/customize/sass/#maps-and-loops)

Basics.razor



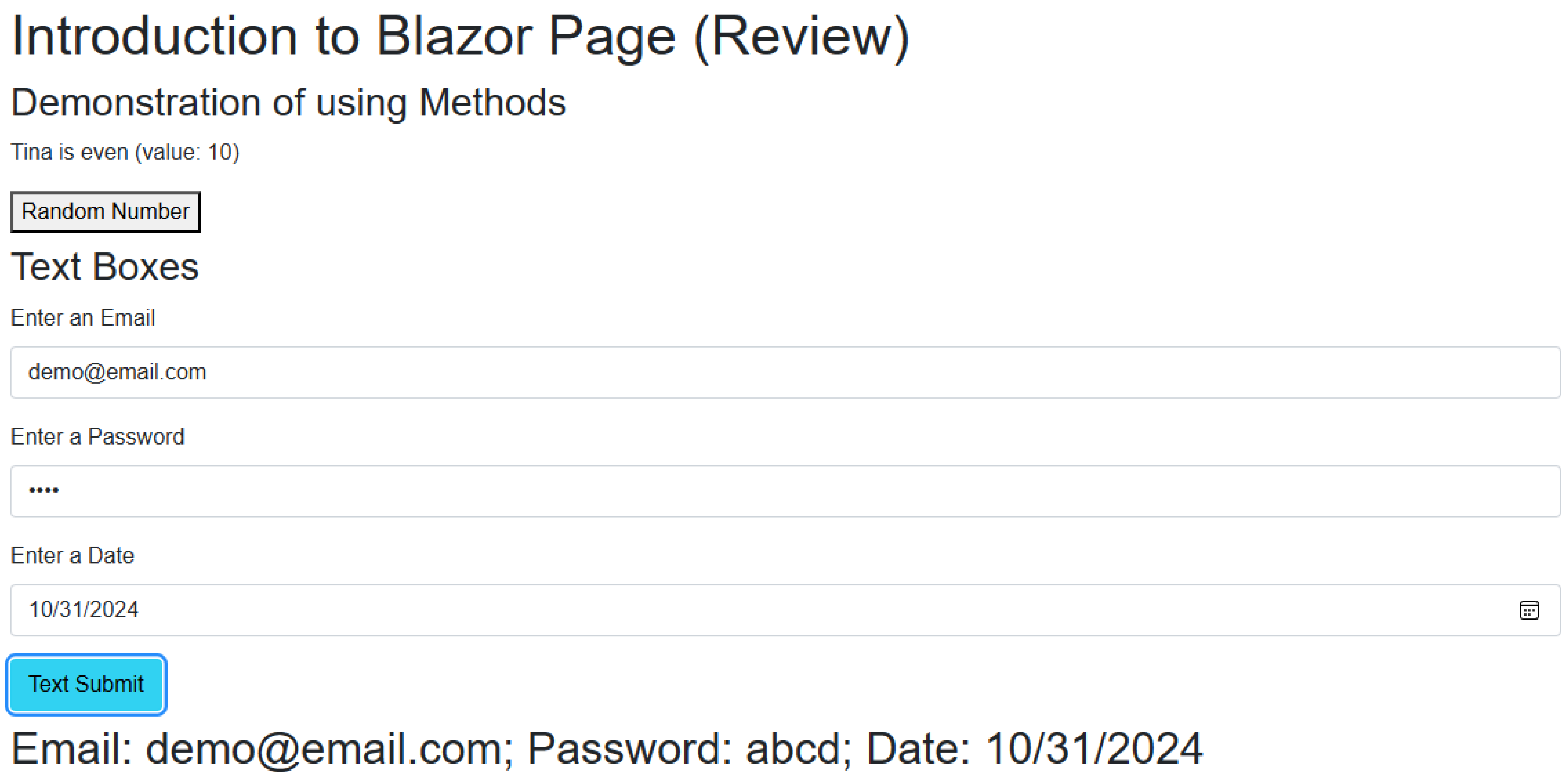
```
<!-- Using Bootstrap label and input to enter a date -->
<div class="mb-3">
    <label for="dateExample" class="form-label">Enter a Date</label>
    <input type="date" class="form-control" id="dateExample" @bind="dateText">
</div>
<!-- Button to trigger the TextSubmit Method -->
<button class="btn btn-info" @onclick="TextSubmit">
    Text Submit
</button>
```

Basics.razor.cs



```
/// <summary>
/// Method is called when the user submits the text input to update the resulting feedback.
/// </summary>
1 reference
private void TextSubmit()
{
    //Combine the values of emailText, passwordText, and dateText into a feedback message
    feedback = $"Email: {emailText}; Password: {passwordText}; Date: {dateText.ToString("d")}";
}
#endregion
```

Blazor Page Output



Introduction to Blazor Page (Review)

Demonstration of using Methods

Tina is even (value: 10)

Text Boxes

Enter an Email

demo@email.com

Enter a Password

••••

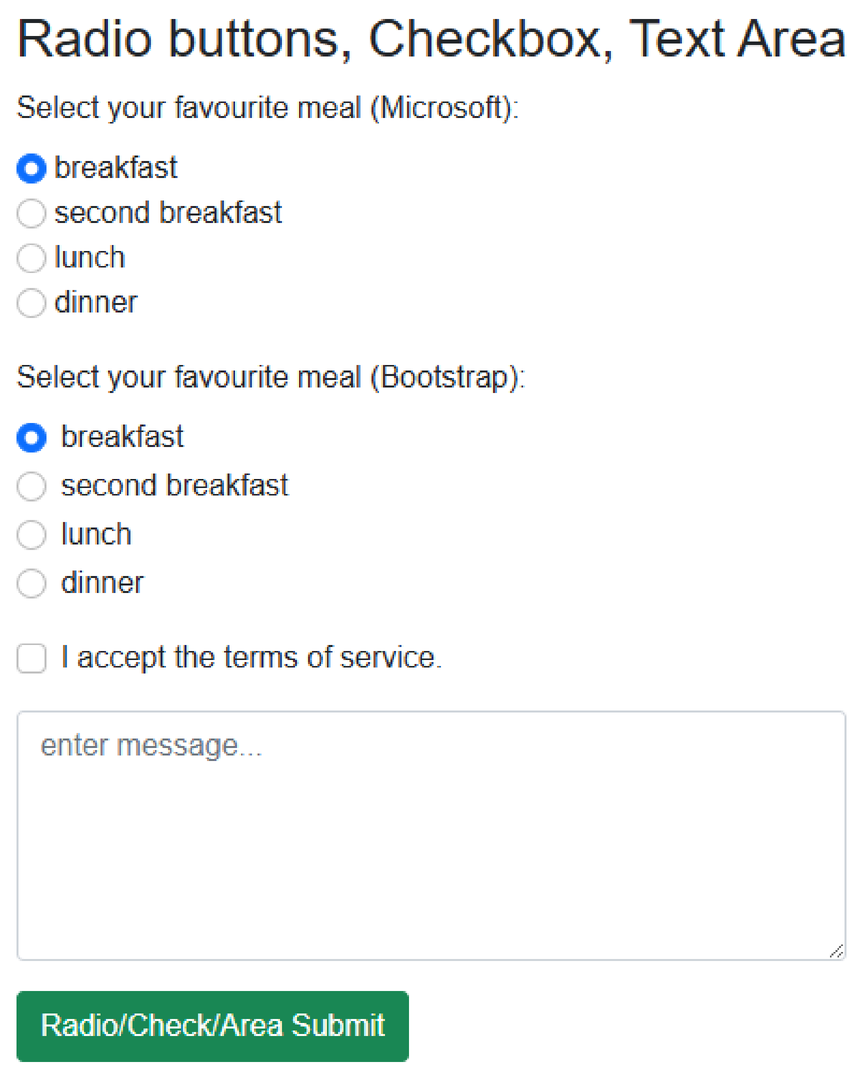
Enter a Date

10/31/2024

Email: demo@email.com; Password: abcd; Date: 10/31/2024

Radio Buttons, Checkboxes, Text Area and Bootstrap

- Select your favorite meals using radio buttons.
- Agree or disagree with the agreement using checkboxes.
- Enter messages or comments in the text area.
- Trigger the submission of radio button choices, checkbox status, and text area content with a button.
- Apply CSS styling to your Blazor application.



Radio buttons, Checkbox, Text Area

Select your favourite meal (Microsoft):

[✓ breakfast]
second breakfast
lunch
dinner

Select your favourite meal (Bootstrap):

[✓ breakfast]
second breakfast
lunch
dinner

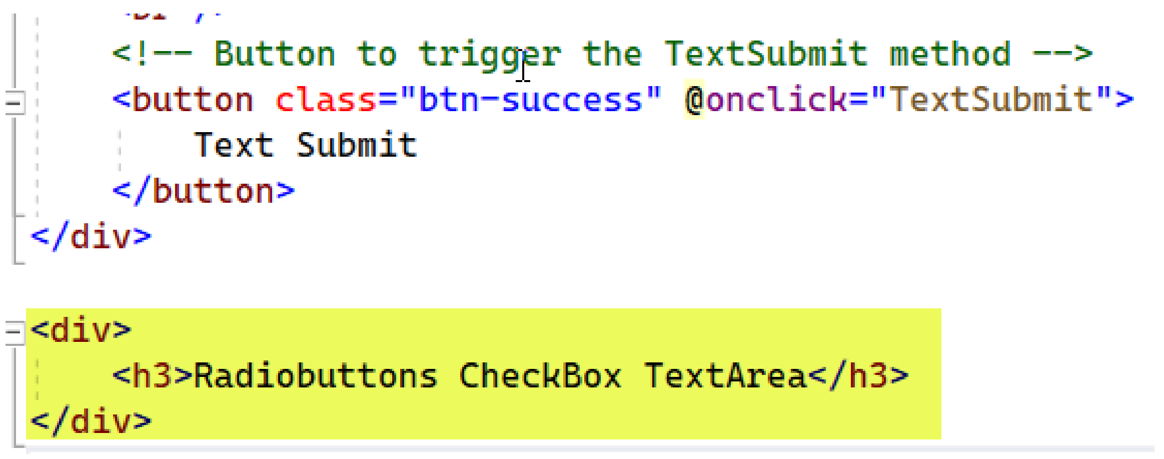
☐ I accept the terms of service.

enter message...

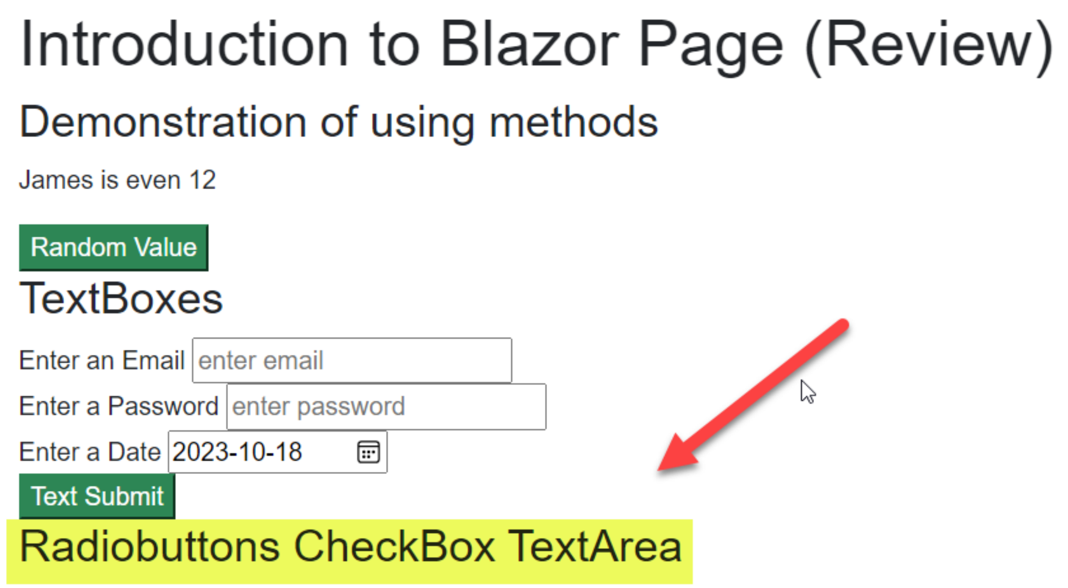
Adding New HTML without CSS or Bootstrap

- If we add a new section to our webpage without applying CSS or Bootstrap classes, we encounter a problem where the new content is added to the bottom of the page, affecting the layout.

Basics.razor



```
    <!-- Button to trigger the TextSubmit method -->
    <button class="btn-success" @onclick="TextSubmit">
        Text Submit
    </button>
</div>

<div>
    <h3>Radiobuttons CheckBox TextArea</h3>
</div>
```



Introduction to Blazor Page (Review)

Demonstration of using methods

James is even 12

TextBoxes

Enter an Email enter email

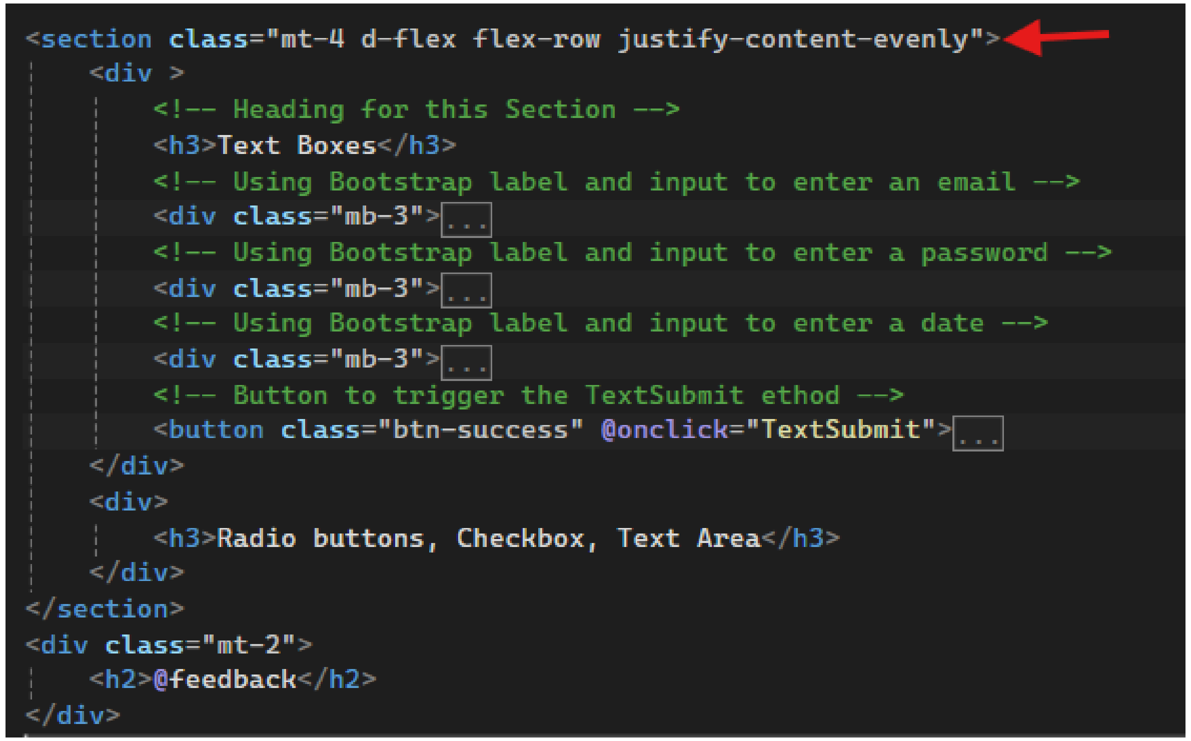
Enter a Password enter password

Enter a Date 2023-10-18

Radiobuttons CheckBox TextArea

Refactor Basics.razor using Bootstrap

Basics.razor

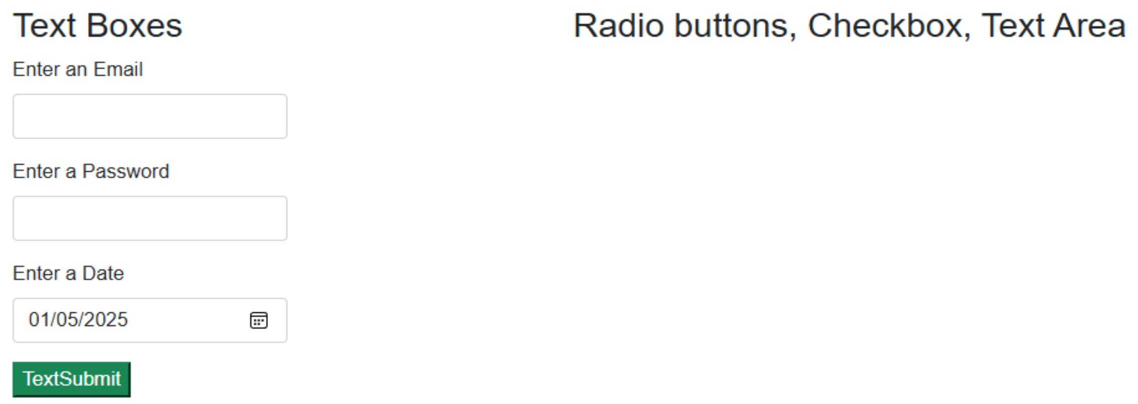


```
<section class="mt-4 d-flex flex-row justify-content-evenly">
    <div >
        <!-- Heading for this Section -->
        <h3>Text Boxes</h3>
        <!-- Using Bootstrap label and input to enter an email -->
        <div class="mb-3">...
        <!-- Using Bootstrap label and input to enter a password -->
        <div class="mb-3">...
        <!-- Using Bootstrap label and input to enter a date -->
        <div class="mb-3">...
        <!-- Button to trigger the TextSubmit ethod -->
        <button class="btn-success" @onclick="TextSubmit">...
    </div>
    <div>
        <h3>Radio buttons, Checkbox, Text Area</h3>
    </div>
</section>
<div class="mt-2">
    <h2>@feedback</h2>
</div>
```

Add a **<section>** around the two **<div>** you have for the text boxes and radio, checkbox, text area divs.

- **d-flex** – Class to make the section have display: flex;
- **flex-row** – Class to make add flex-direction: row; to the section.
- **justify-content-evenly** – Class to add justify-content: space-evenly; to the section.

Note: If you have forgotten Flexboxes in CSS remind yourself with flex-box Zombie (https://mastery.games/flexboxzombies/)



Text Boxes

Radio buttons, Checkbox, Text Area

Enter an Email

Enter a Password

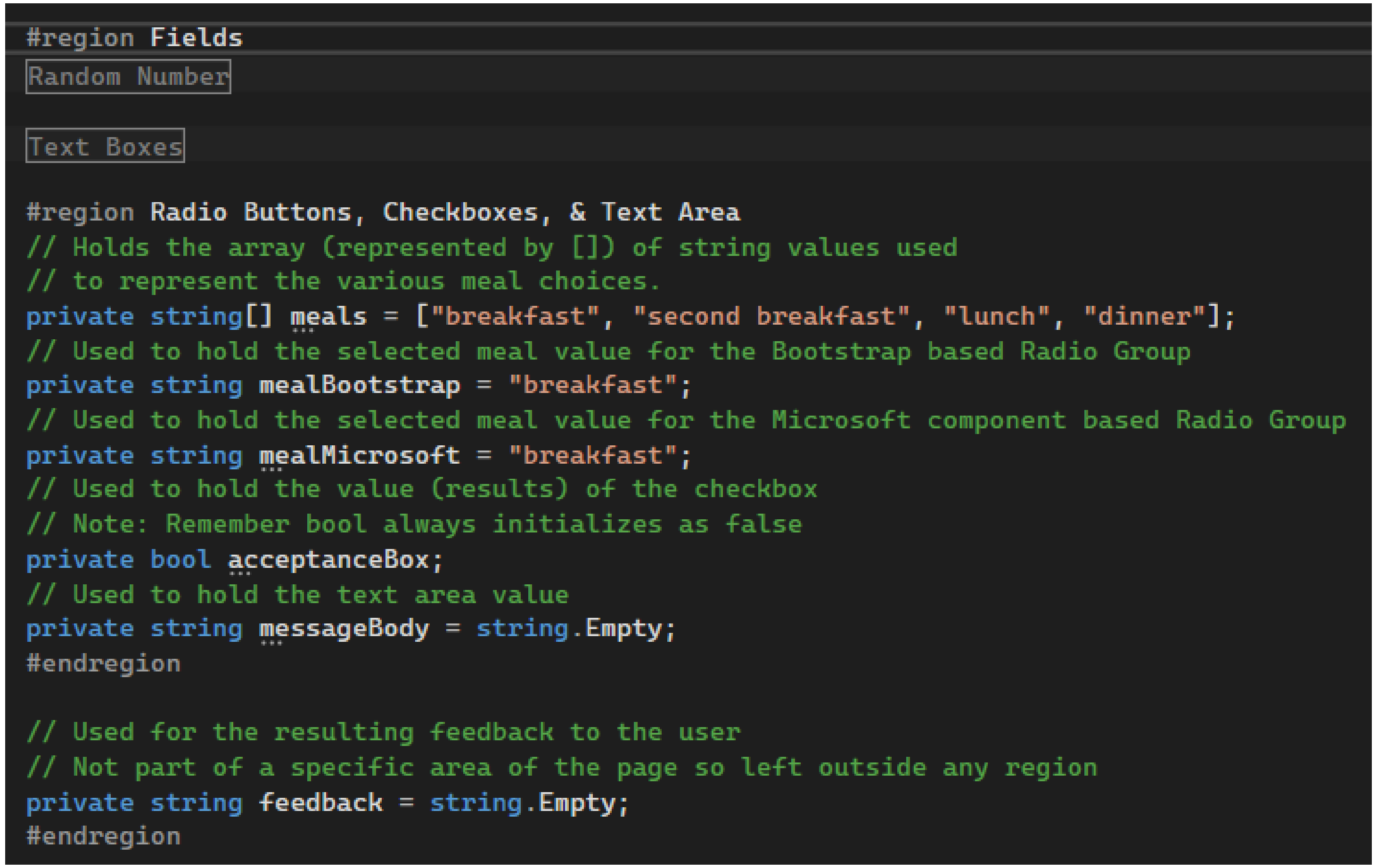
Enter a Date

01/05/2025

Fields Required for the Radio Buttons, Checkbox, and Textarea

- Since we have some distinct areas now, we can add some regions inside our **#region Fields**
- A collection is used to populate the radio button group.
    - The list can be populated from a data source, a hard-coded list, a CSV file, etc.
- We will be creating a “string array” of potential favorite meals.
- We have two fields to hold the selected radio values because we will be looking at two ways to create radio buttons (Bootstrap/HTML and using a Microsoft Component)

Basics.razor.cs

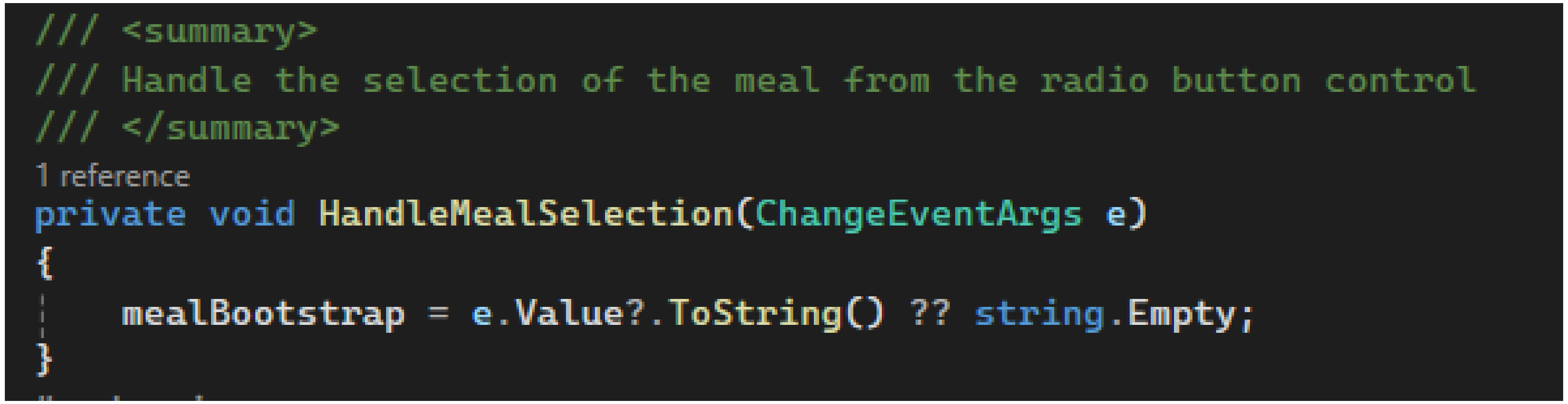


```
#region Fields
Random Number

Text Boxes

#region Radio Buttons, Checkboxes, & Text Area
// Holds the array (represented by []) of string values used
// to represent the various meal choices.
private string[] meals = ["breakfast", "second breakfast", "lunch", "dinner"];
// Used to hold the selected meal value for the Bootstrap based Radio Group
private string mealBootstrap = "breakfast";
// Used to hold the selected meal value for the Microsoft component based Radio Group
private string mealMicrosoft = "breakfast";
// Used to hold the value (results) of the checkbox
// Note: Remember bool always initializes as false
private bool acceptanceBox;
// Used to hold the text area value
private string messageBody = string.Empty;
#endregion

// Used for the resulting feedback to the user
// Not part of a specific area of the page so left outside any region
private string feedback = string.Empty;
#endregion
```

Radio Buttons using Collections

- Create a method to handle meal selection.
- HandleMealSelection method: Update the meal variable with the selected meal when a radio button changes.
- We use a null check (??) here just in case. If the ToString() value of e.Value end up null, then the value is set to a string.Empty

Basics.razor.cs

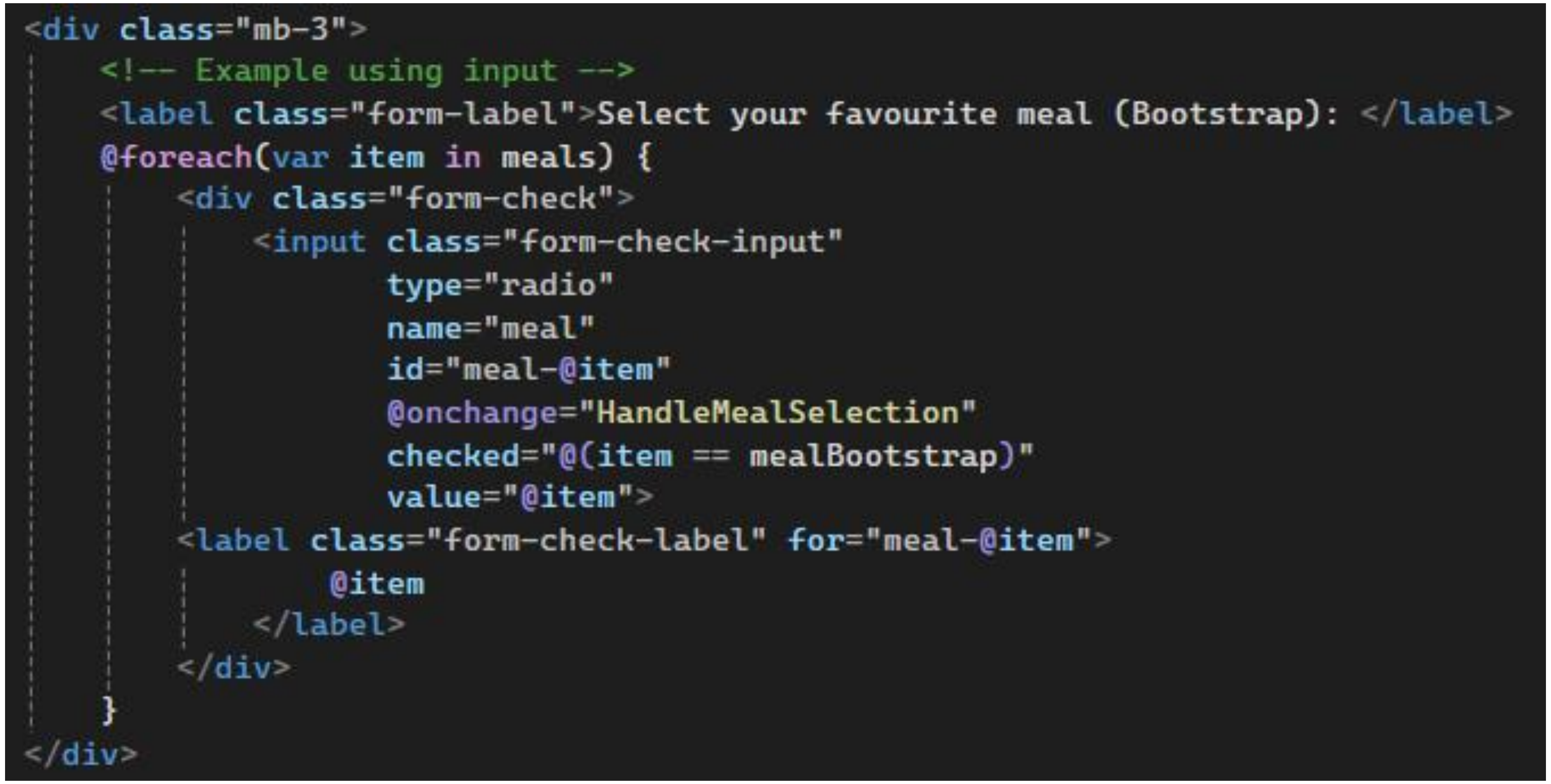


```
/// <summary>
/// Handle the selection of the meal from the radio button control
/// </summary>
1 reference
private void HandleMealSelection(ChangeEventArgs e)
{
    mealBootstrap = e.Value?.ToString() ?? string.Empty;
}
```

Radio Buttons using Collections (Bootstrap)

- Add code to the web page to generate the "Radio Buttons using Collections."
- Name attribute: This ensures that only one radio button can be selected within the same group.
- Value attribute: Holds the meal option. This will be the value bound to the meal variable.
- @onchange attribute: Calls the HandleMealSelection method when the selection changes.
- Checked attribute: Determine if a radio button should be selected based on whether its value matches the current value of the meal.

Basics.razor

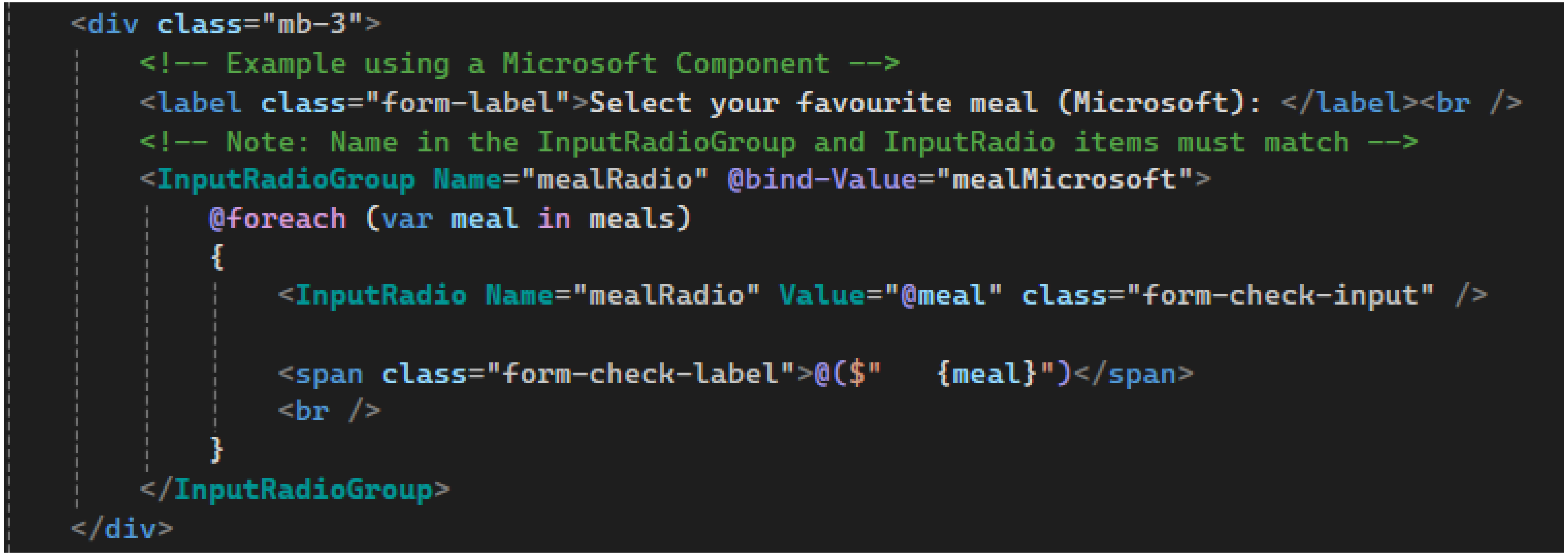


```
<div class="mb-3">
    <!-- Example using input -->
    <label class="form-label">Select your favourite meal (Bootstrap): </label>
    @foreach(var item in meals) {
        <div class="form-check">
            <input class="form-check-input"
                   type="radio"
                   name="meal"
                   id="meal-@item"
                   @onchange="HandleMealSelection"
                   checked="@(item == mealBootstrap)"
                   value="@item">
        <label class="form-check-label" for="meal-@item">
               @item
            </label>
        </div>
    }
</div>
```

Radio Buttons using Collections (Microsoft)

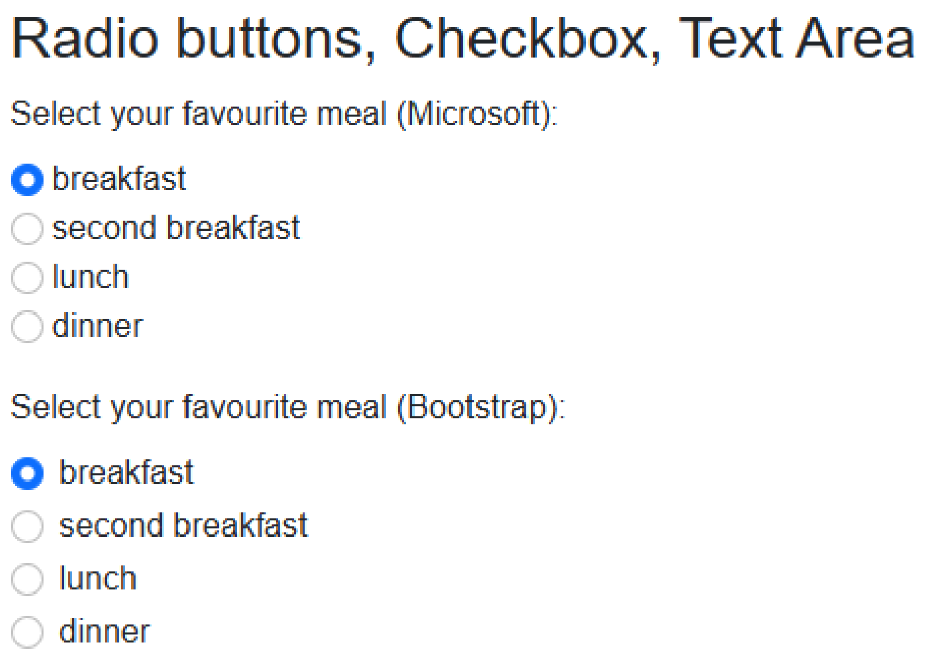
- Add code to the web page to generate the “Radio Buttons using Collections.” with a Microsoft Blazor component.
- The component lets you use @bind-Value without an additional method.

Basics.razor



```
<div class="mb-3">
    <!-- Example using a Microsoft Component -->
    <label class="form-label">Select your favourite meal (Microsoft): </label><br />
    <!-- Note: Name in the InputRadioGroup and InputRadio items must match -->
    <InputRadioGroup Name="mealRadio" @bind-Value="mealMicrosoft">
        @foreach (var meal in meals)
        {
            <InputRadio Name="mealRadio" Value="@meal" class="form-check-input" />

            <span class="form-check-label">@($"   {meal}")</span>
            <br />
        }
    </InputRadioGroup>
</div>
```

Blazor Page Output



Radio buttons, Checkbox, Text Area

Select your favourite meal (Microsoft):

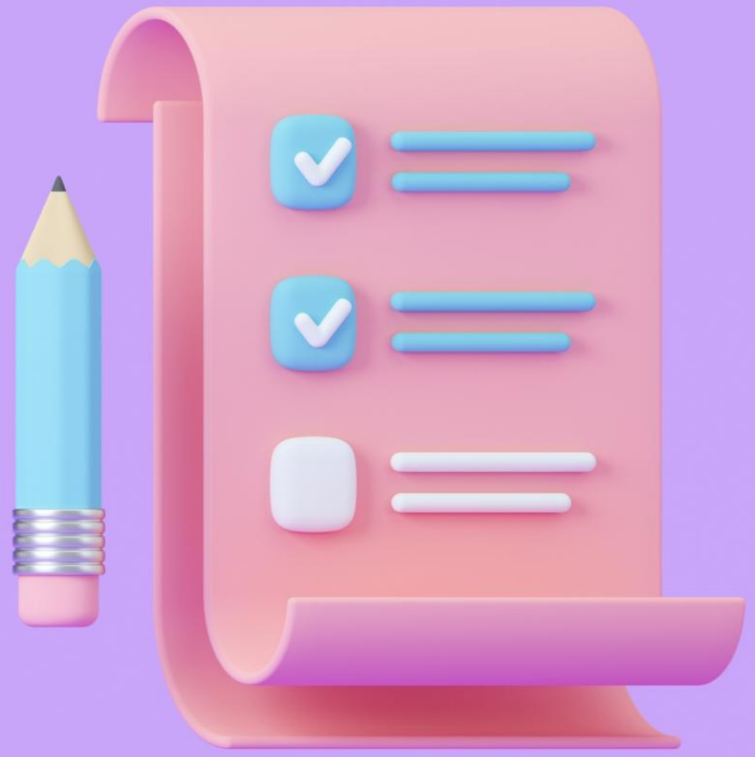
[✓ breakfast]
second breakfast
lunch
dinner

Select your favourite meal (Bootstrap):

[✓ breakfast]
second breakfast
lunch
dinner

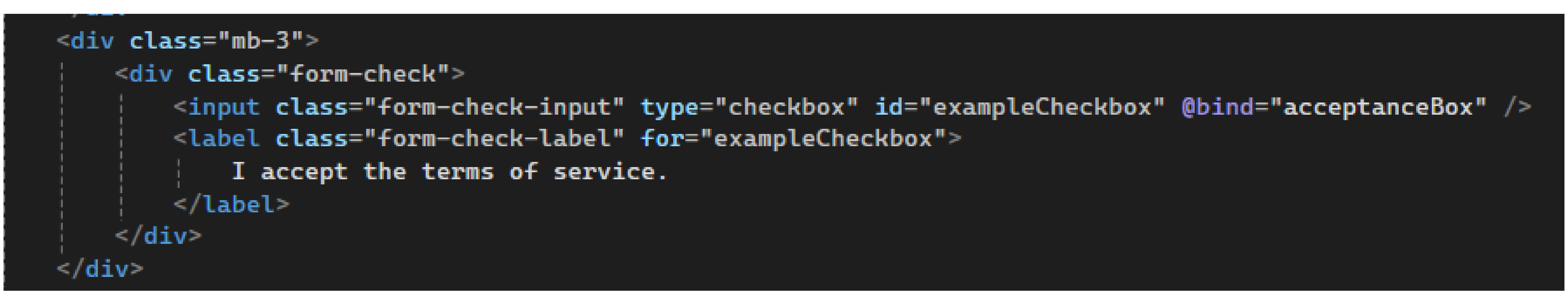
Check Box

- Checkboxes in Blazor are a user interface element that allows users to select or deselect options.
- They are commonly used for presenting binary choices or enabling multiple selections from a list of items.
- Blazor provides a way to bind the state of checkboxes to C# variables, enabling two-way data binding.
- When a user interacts with a checkbox by clicking it, the associated C# variable can be updated to reflect the checked or unchecked state.



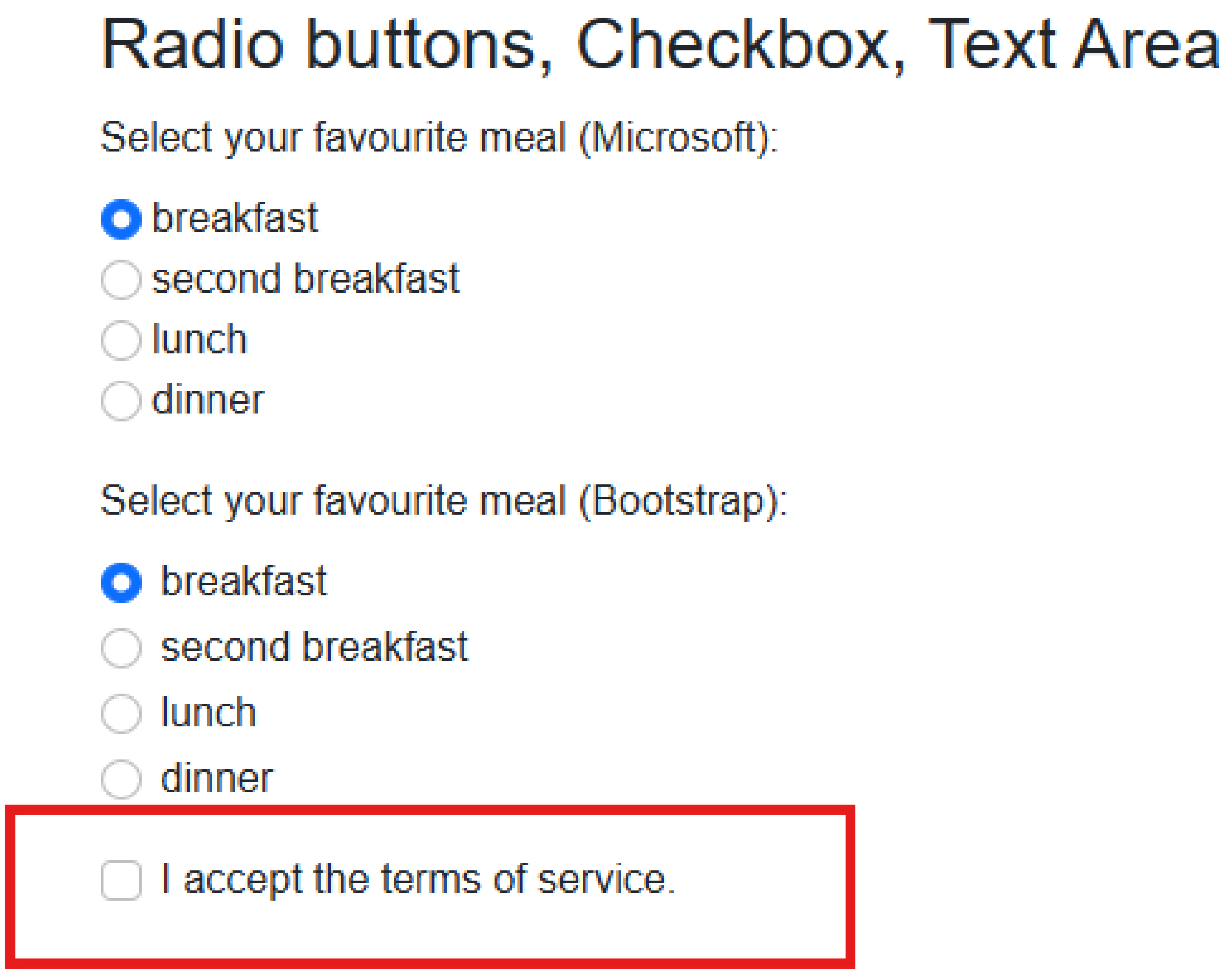
Checkbox Blazor Code

Basics.razor



```
<div class="mb-3">
    <div class="form-check">
        <input class="form-check-input" type="checkbox" id="exampleCheckbox" @bind="acceptanceBox" />
        <label class="form-check-label" for="exampleCheckbox">
            I accept the terms of service.
        </label>
    </div>
</div>
```

Blazor Page Output



Radio buttons, Checkbox, Text Area

Select your favourite meal (Microsoft):

[✓ breakfast] second breakfast lunch dinner

Select your favourite meal (Bootstrap):

[✓ breakfast] second breakfast lunch dinner

☐ I accept the terms of service.

Text Area

- Text Areas in Blazor are user interface elements used for capturing and displaying multi-line text input.
- They are suitable for gathering longer textual information from users, such as comments, descriptions, or messages.
- Blazor allows you to bind the content of Text Areas to C# variables, enabling easy data synchronization between the user interface and code behind.
- Users can enter and edit text in a Text Area; any changes are reflected in the associated C# variable.
- Text Areas are commonly used in forms, chat applications, comment sections, and anywhere extended text input is required within a Blazor application.

Text Areas Blazor Code

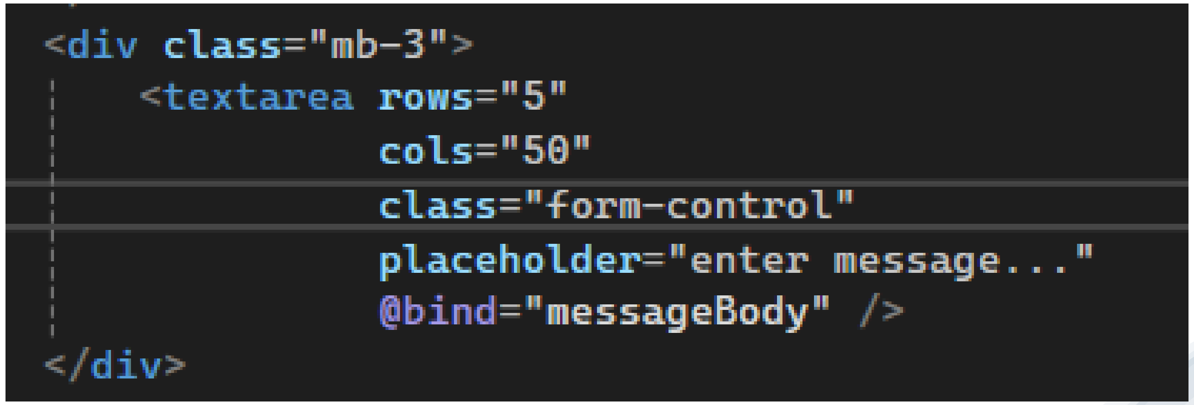
<textarea>: An HTML element designed for creating a multiline text input area where users can enter or paste text.

•**rows="5"**: An attribute that defines the number of visible text lines within the text area.

•**cols="50"**: An attribute that specifies the number of visible characters in each line of the text area.

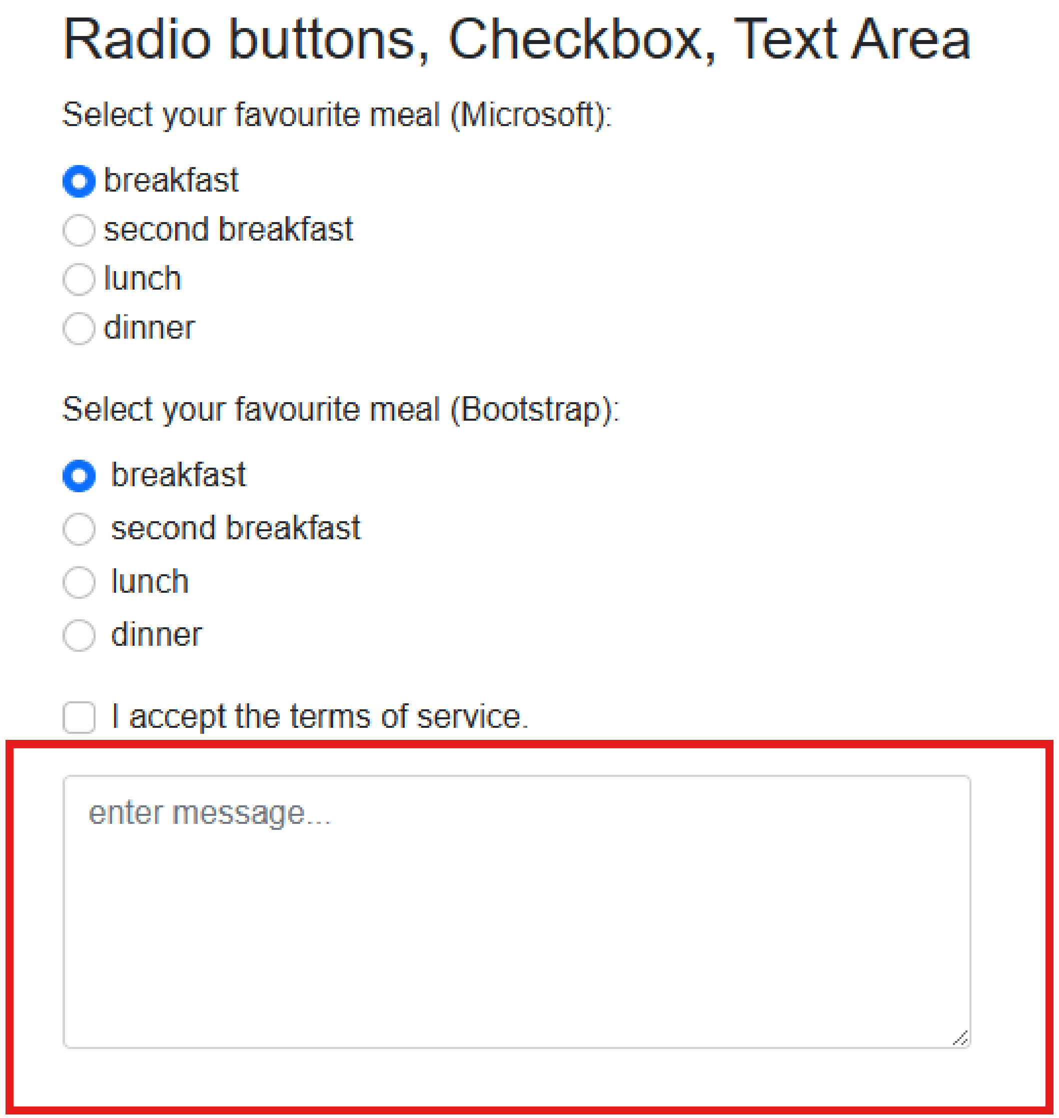
•**placeholder="enter message"**: An attribute that supplies a placeholder text. This text appears in the text area when it's empty, offering users a hint about the expected input.

Basics.razor



```
<div class="mb-3">
    <textarea rows="5"
              cols="50"
              class="form-control"
              placeholder="enter message..."
              @bind="messageBody" />
</div>
```

Blazor Page Output



Radio buttons, Checkbox, Text Area

Select your favourite meal (Microsoft):

[✓ breakfast]
second breakfast
lunch
dinner

Select your favourite meal (Bootstrap):

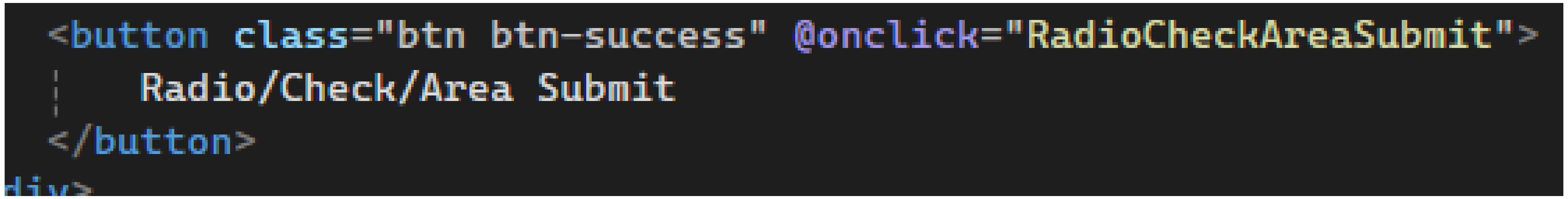
[✓ breakfast]
second breakfast
lunch
dinner

☐ I accept the terms of service.

enter message...

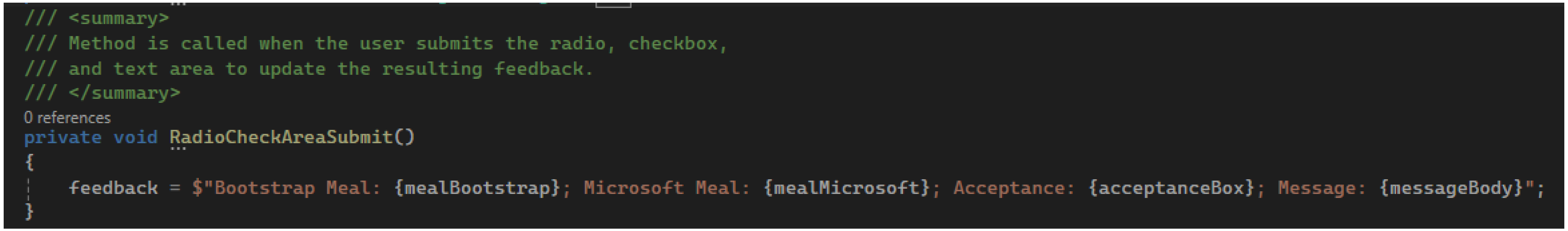
Coding the RadioCheckAreaSubmit Method

Basics.razor



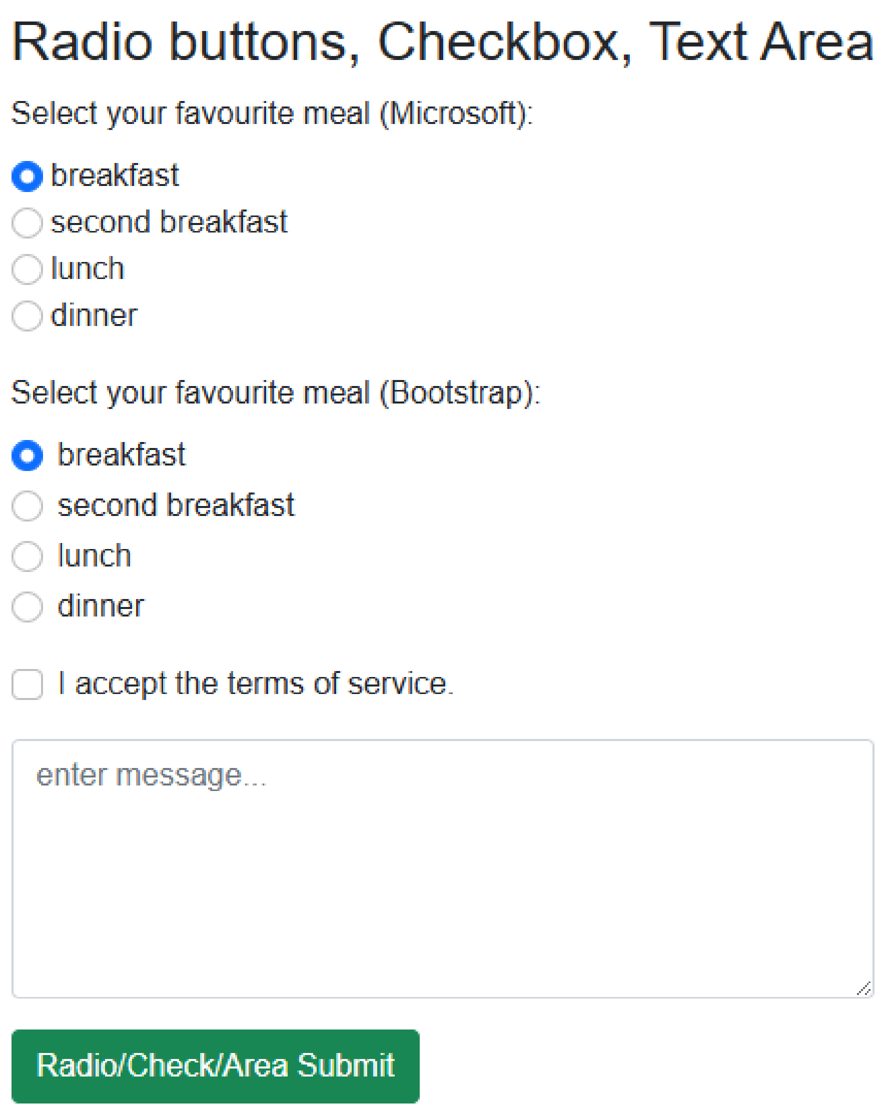
```
<button class="btn btn-success" @onclick="RadioCheckAreaSubmit">
    Radio/Check/Area Submit
</button>
```

Basics.razor.cs



```
/// <summary>
/// Method is called when the user submits the radio, checkbox,
/// and text area to update the resulting feedback.
/// </summary>
0 references
private void RadioCheckAreaSubmit()
{
    feedback = $"Bootstrap Meal: {mealBootstrap}; Microsoft Meal: {mealMicrosoft}; Acceptance: {acceptanceBox}; Message: {messageBody}";
}
```

Blazor Page Output



Radio buttons, Checkbox, Text Area

Select your favourite meal (Microsoft):

[✓ breakfast]
second breakfast
lunch
dinner

Select your favourite meal (Bootstrap):

[✓ breakfast]
second breakfast
lunch
dinner

☐ I accept the terms of service.

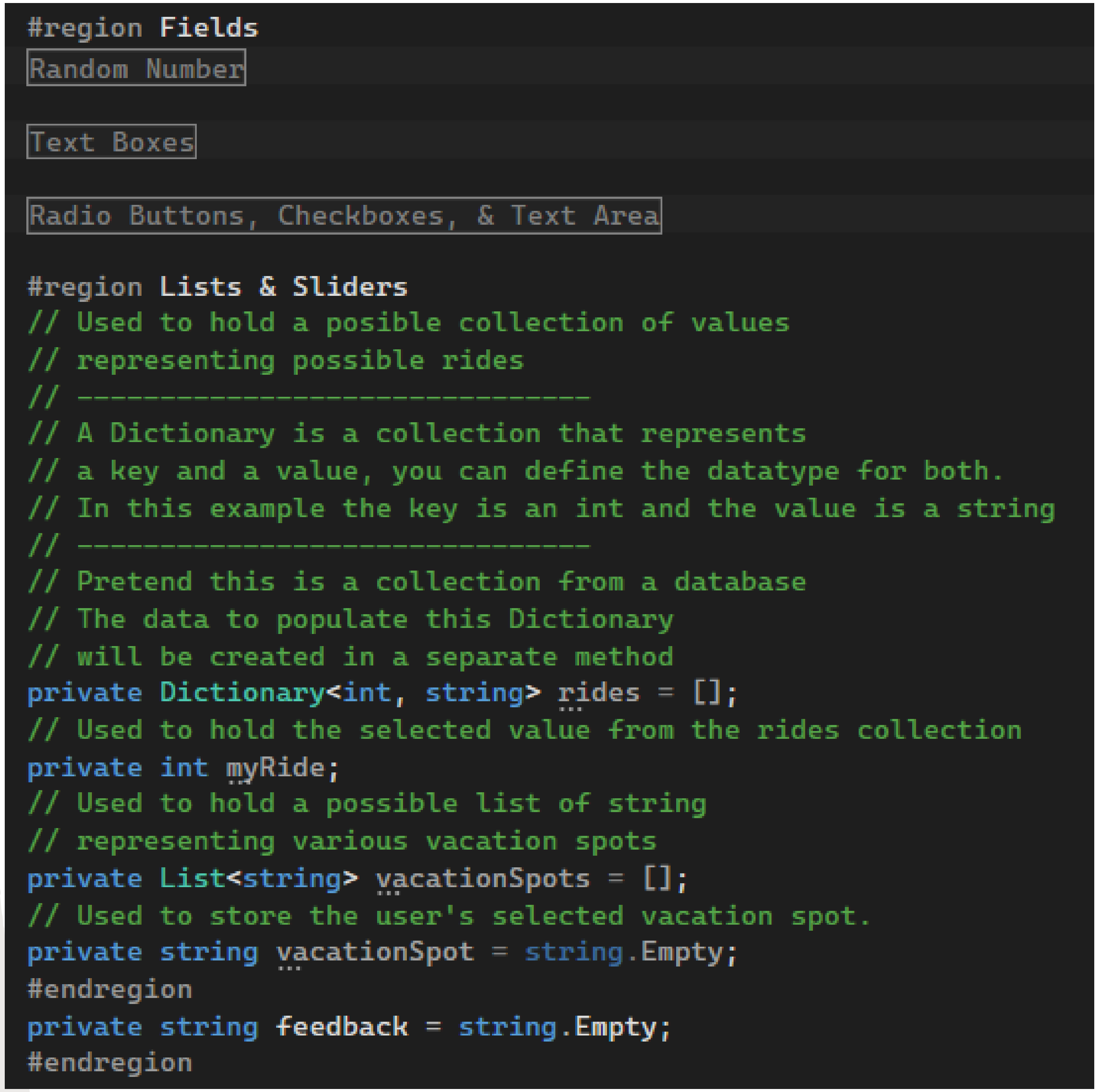
List and Sliders

- **List (Dropdown Selection):**
- The "List" in this context refers to a dropdown selection element.
- Users can open the dropdown and choose from a list of options.
- It's often used when you have a set of predefined choices, and users need to pick one.
- An example of this is a user a selection of favorite rides, such as "Car," "Bus," "Bike," etc.
- It's an interactive way to capture user preferences.

- **Slider (Rating Control):**
- The "Slider" is a form control that allows users to select a value within a specified range.
- Users can slide a knob or handle along a track to indicate their preference.
- In this case, we use it as a rating control, where users can rate something from "bad" to "good" on a scale.
- It's a visually intuitive way to collect user feedback and ratings.
- User interfaces commonly use sliders to capture continuous values within a defined range.

Field Required for List

Basics.razor.cs



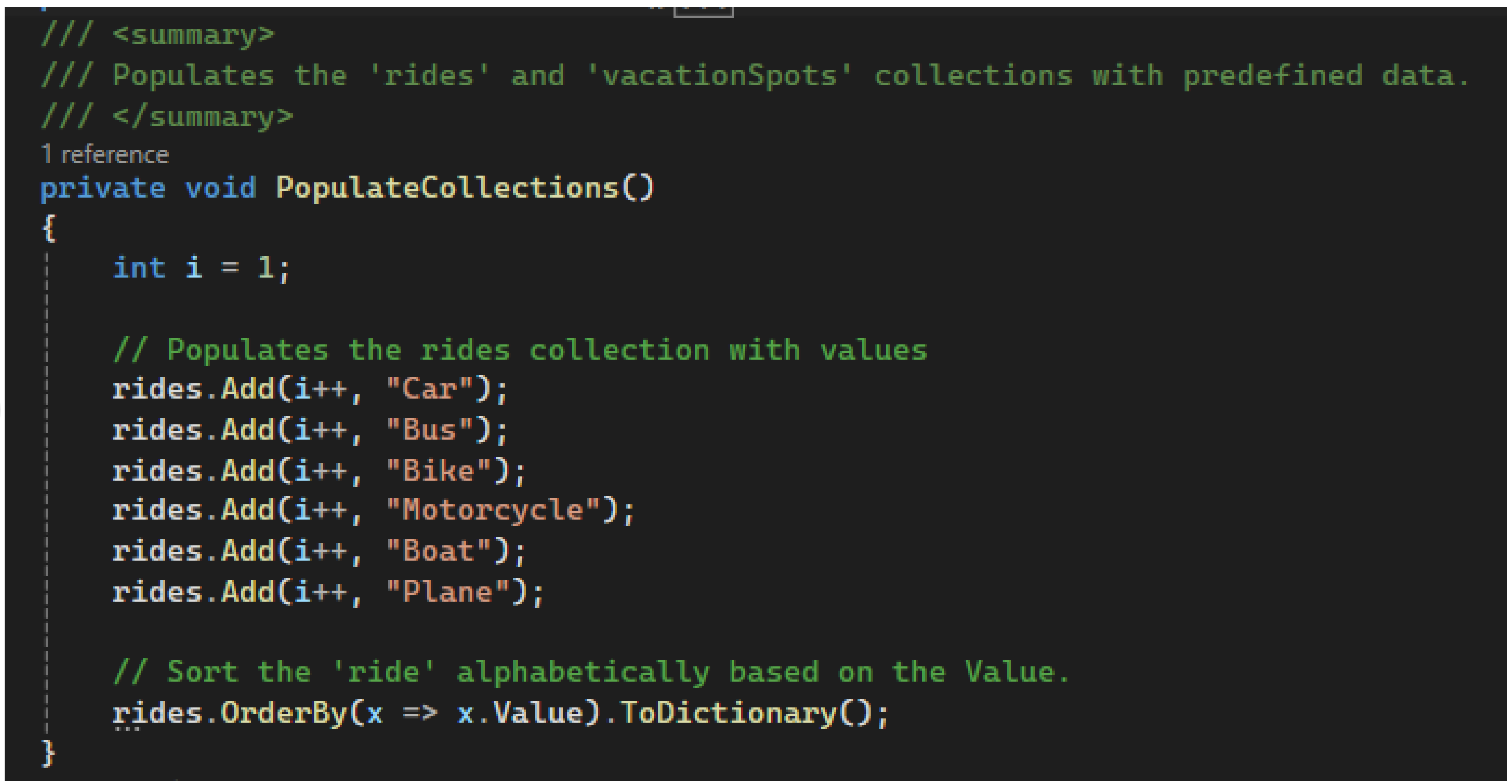
```
#region Fields
Random Number

Text Boxes

Radio Buttons, Checkboxes, & Text Area

#region Lists & Sliders
// Used to hold a posible collection of values
// representing possible rides
// -------------------------------
// A Dictionary is a collection that represents
// a key and a value, you can define the datatype for both.
// In this example the key is an int and the value is a string
// -------------------------------
// Pretend this is a collection from a database
// The data to populate this Dictionary
// will be created in a separate method
private Dictionary<int, string> rides = [];
// Used to hold the selected value from the rides collection
private int myRide;
// Used to hold a possible list of string
// representing various vacation spots
private List<string> vacationSpots = [];
// Used to store the user's selected vacation spot.
private string vacationSpot = string.Empty;
#endregion
private string feedback = string.Empty;
#endregion
```

List Using Associated Collection of Objects

- This type of list represents a web-based control where users can select their favorite ride from a dynamically generated dropdown list. The available ride options are based on a list of “Selection View” objects, each containing a unique identifier and display text. Users' selections are stored in the “My Ride” property, representing the ID of the chosen ride, allowing for easy retrieval and processing of the user's preference in a web application.

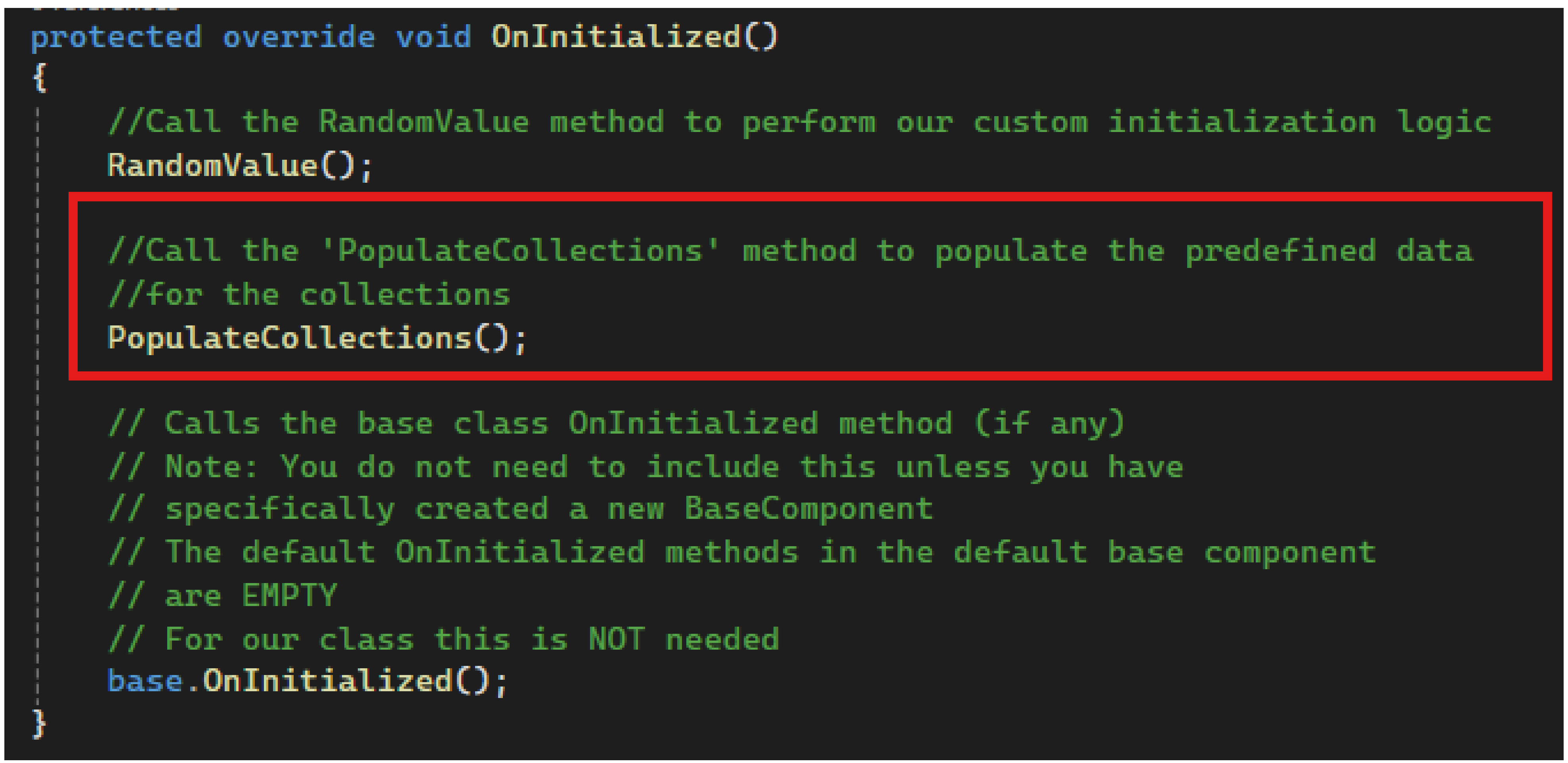
Create a Method for Populate the **rides** Collection

Basics.razor.cs



```
/// <summary>
/// Populates the 'rides' and 'vacationSpots' collections with predefined data.
/// </summary>
1 reference
private void PopulateCollections()
{
    int i = 1;

    // Populates the rides collection with values
    rides.Add(i++, "Car");
    rides.Add(i++, "Bus");
    rides.Add(i++, "Bike");
    rides.Add(i++, "Motorcycle");
    rides.Add(i++, "Boat");
    rides.Add(i++, "Plane");

    // Sort the 'ride' alphabetically based on the Value.
    rides.OrderBy(x => x.Value).ToDictionary();
}
```

Basics.razor.cs

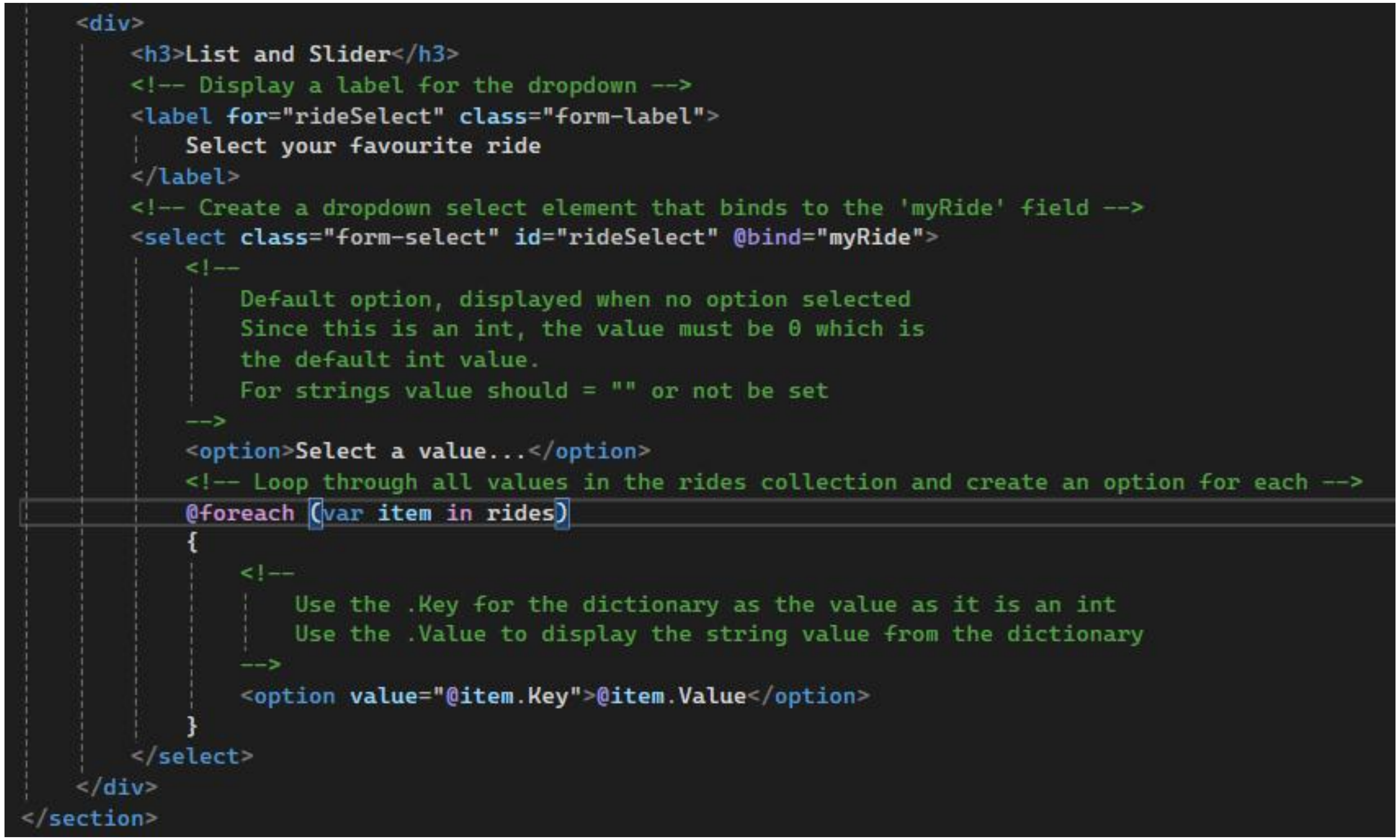


```
protected override void OnInitialized()
{
    //Call the RandomValue method to perform our custom initialization logic
    RandomValue();

    //Call the 'PopulateCollections' method to populate the predefined data
    //for the collections
    PopulateCollections();

    // Calls the base class OnInitialized method (if any)
    // Note: You do not need to include this unless you have
    // specifically created a new BaseComponent
    // The default OnInitialized methods in the default base component
    // are EMPTY
    // For our class this is NOT needed
    base.OnInitialized();
}
```

Updating the OnInitialized() Method

Adding the Rider List

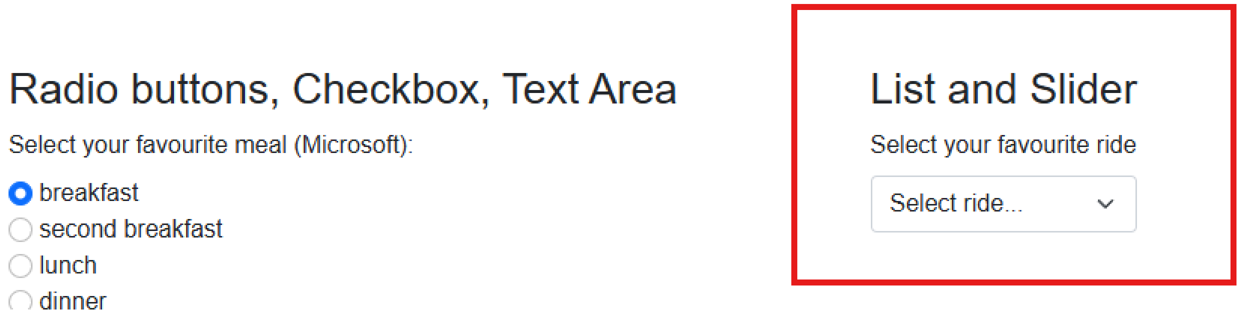
Basics.razor



```
    <div>
        <h3>List and Slider</h3>
        <!-- Display a label for the dropdown -->
        <label for="rideSelect" class="form-label">
            Select your favourite ride
        </label>
        <!-- Create a dropdown select element that binds to the 'myRide' field -->
        <select class="form-select" id="rideSelect" @bind="myRide">
            <!--
                Default option, displayed when no option selected
                Since this is an int, the value must be 0 which is
                the default int value.
                For strings value should = "" or not be set
            -->
            <option>Select a value...</option>
            <!-- Loop through all values in the rides collection and create an option for each -->
            @foreach (var item in rides)
            {
                <!--
                    Use the .Key for the dictionary as the value as it is an int
                    Use the .Value to display the string value from the dictionary
                -->
                <option value="@item.Key">@item.Value</option>
            }
        </select>
    </div>
</section>
```

Blazor Page Output

- Since we set the default value="0", "Select ride..." is selected what the page is open.
- You must ensure the value for your default option is set correctly for it to be automatically selected
  - ints default to 0
  - string do not have a default and must be set (likely to string.Empty so the value can be ""
  - Other datatypes have other default values, always consider the datatype you are using.


Radio buttons, Checkbox, Text Area

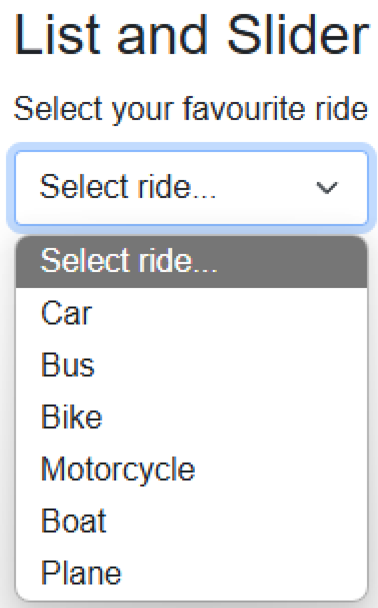
Select your favourite meal (Microsoft):

- [✓ breakfast]
- second breakfast
- lunch
- dinner

List and Slider

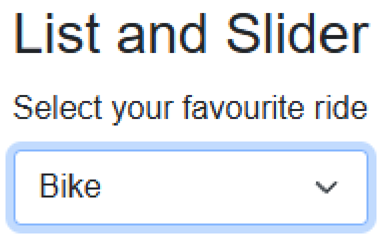
Select your favourite ride

Select ride...


List and Slider

Select your favourite ride

Select ride...

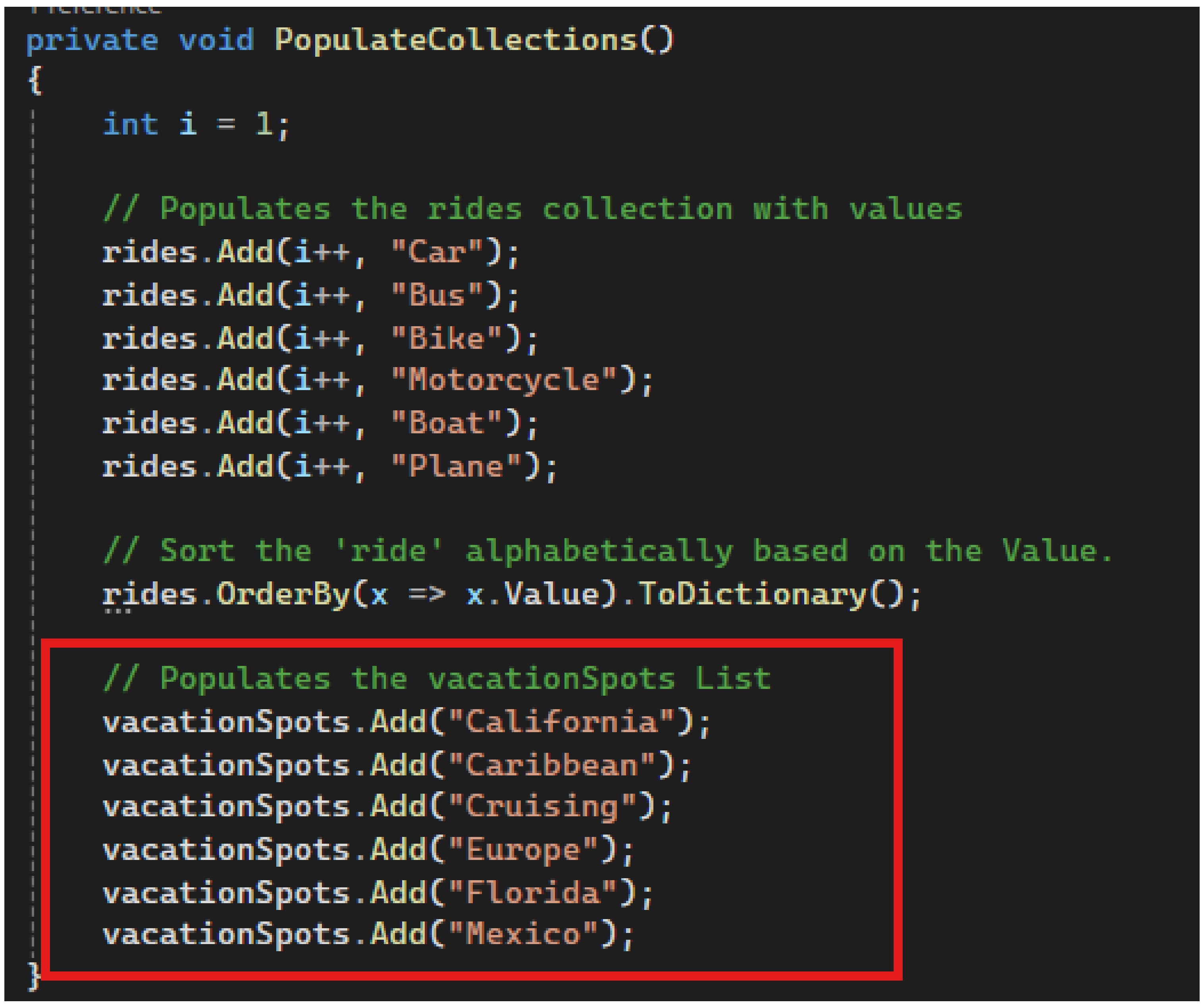

List and Slider

Select your favourite ride

Bike

Basics.razor.cs

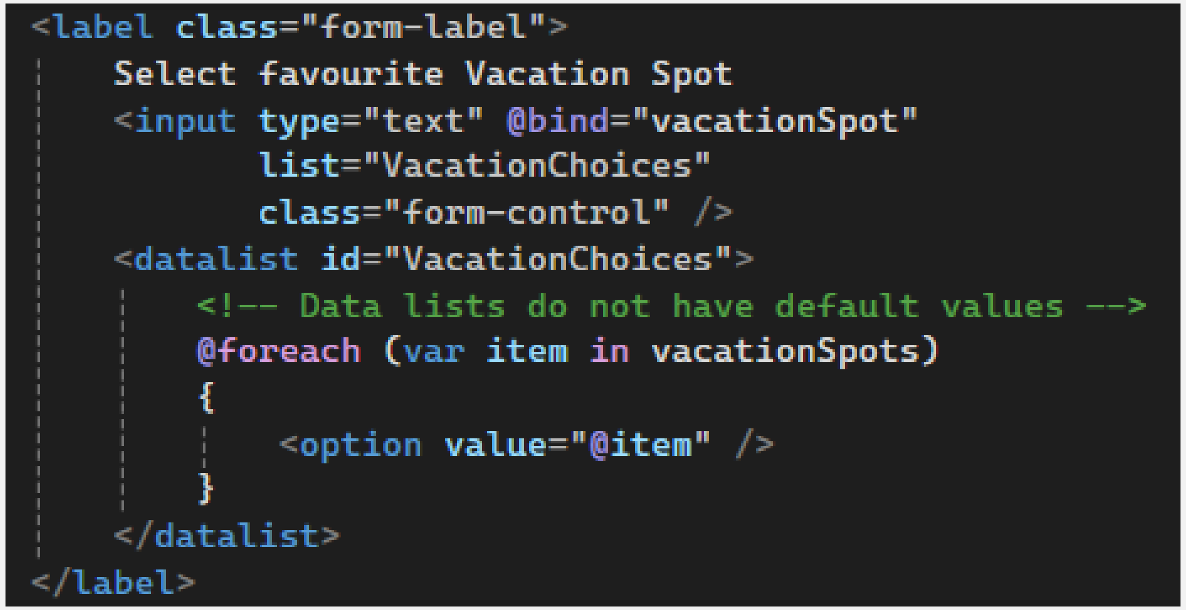
Updating the Method for Populating the Lists



```
private void PopulateCollections()
{
    int i = 1;

    // Populates the rides collection with values
    rides.Add(i++, "Car");
    rides.Add(i++, "Bus");
    rides.Add(i++, "Bike");
    rides.Add(i++, "Motorcycle");
    rides.Add(i++, "Boat");
    rides.Add(i++, "Plane");

    // Sort the 'ride' alphabetically based on the Value.
    rides.OrderBy(x => x.Value).ToDictionary();

    // Populates the vacationSpots List
    vacationSpots.Add("California");
    vacationSpots.Add("Caribbean");
    vacationSpots.Add("Cruising");
    vacationSpots.Add("Europe");
    vacationSpots.Add("Florida");
    vacationSpots.Add("Mexico");
}
```

Adding the Vacation Spot List (using a DataList)

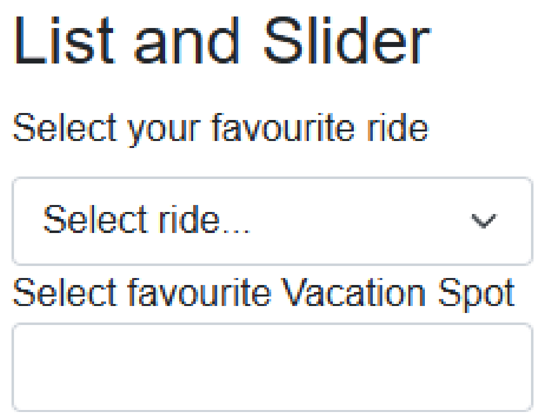
Basics.razor



```
<label class="form-label">
    Select favourite Vacation Spot
    <input type="text" @bind="vacationSpot"
           list="VacationChoices"
           class="form-control" />
    <datalist id="VacationChoices">
        <!-- Data lists do not have default values -->
        @foreach (var item in vacationSpots)
        {
            <option value="@item" />
        }
    </datalist>
</label>
```

Note: Data lists do not have a default value as they are also filter lists.

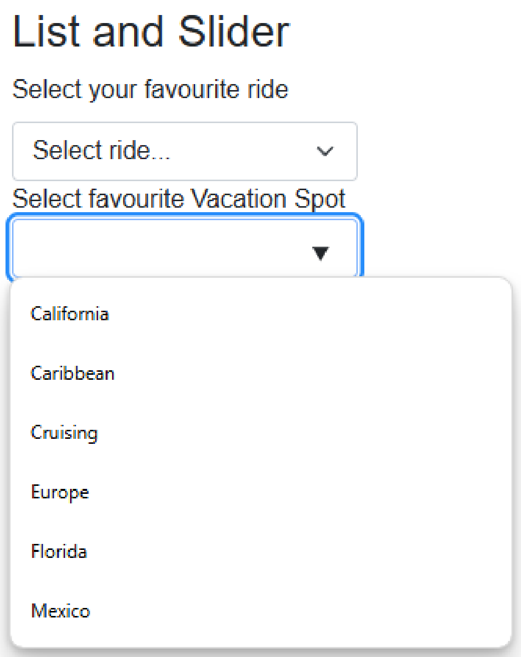
Blazor Page Output



List and Slider

Select your favourite ride

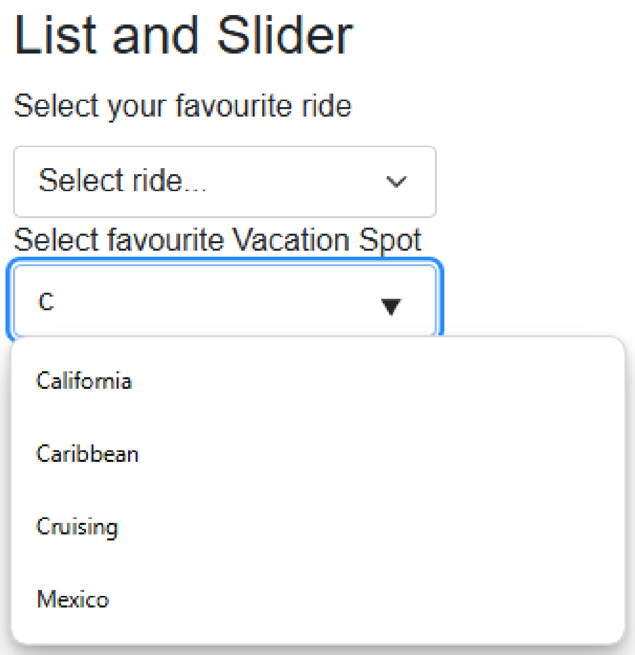
Select favourite Vacation Spot



List and Slider

Select your favourite ride

Select favourite Vacation Spot



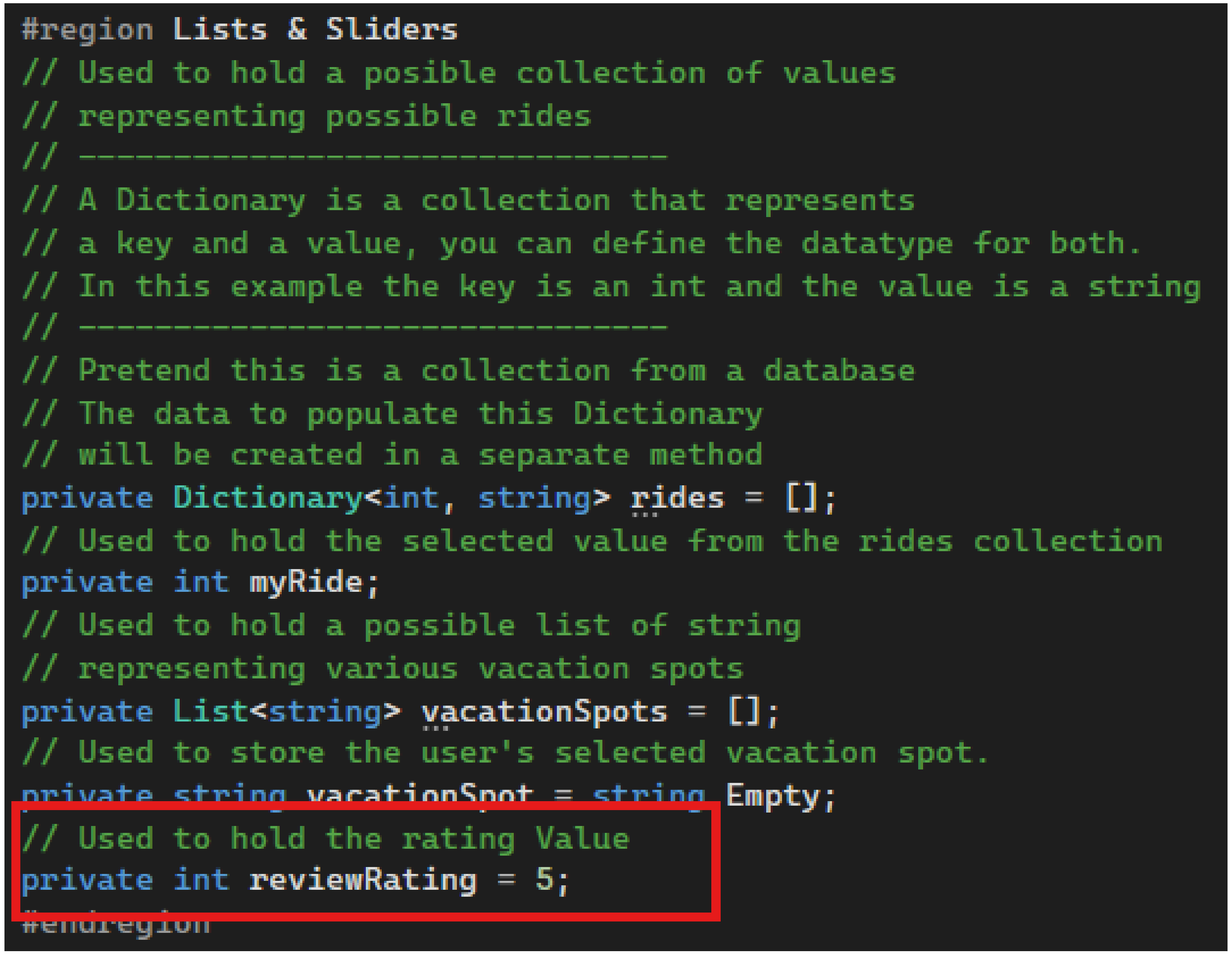
List and Slider

Select your favourite ride

Select favourite Vacation Spot

Field Required for Slider

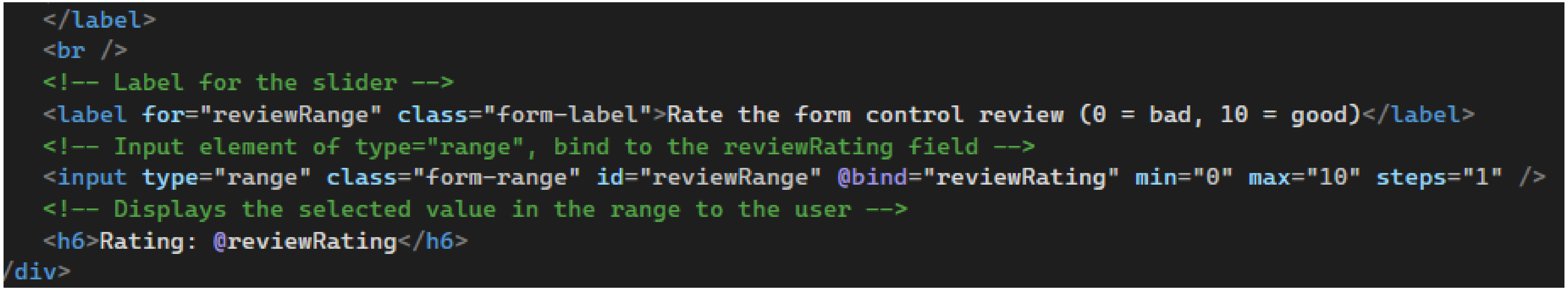
Basics.razor.cs



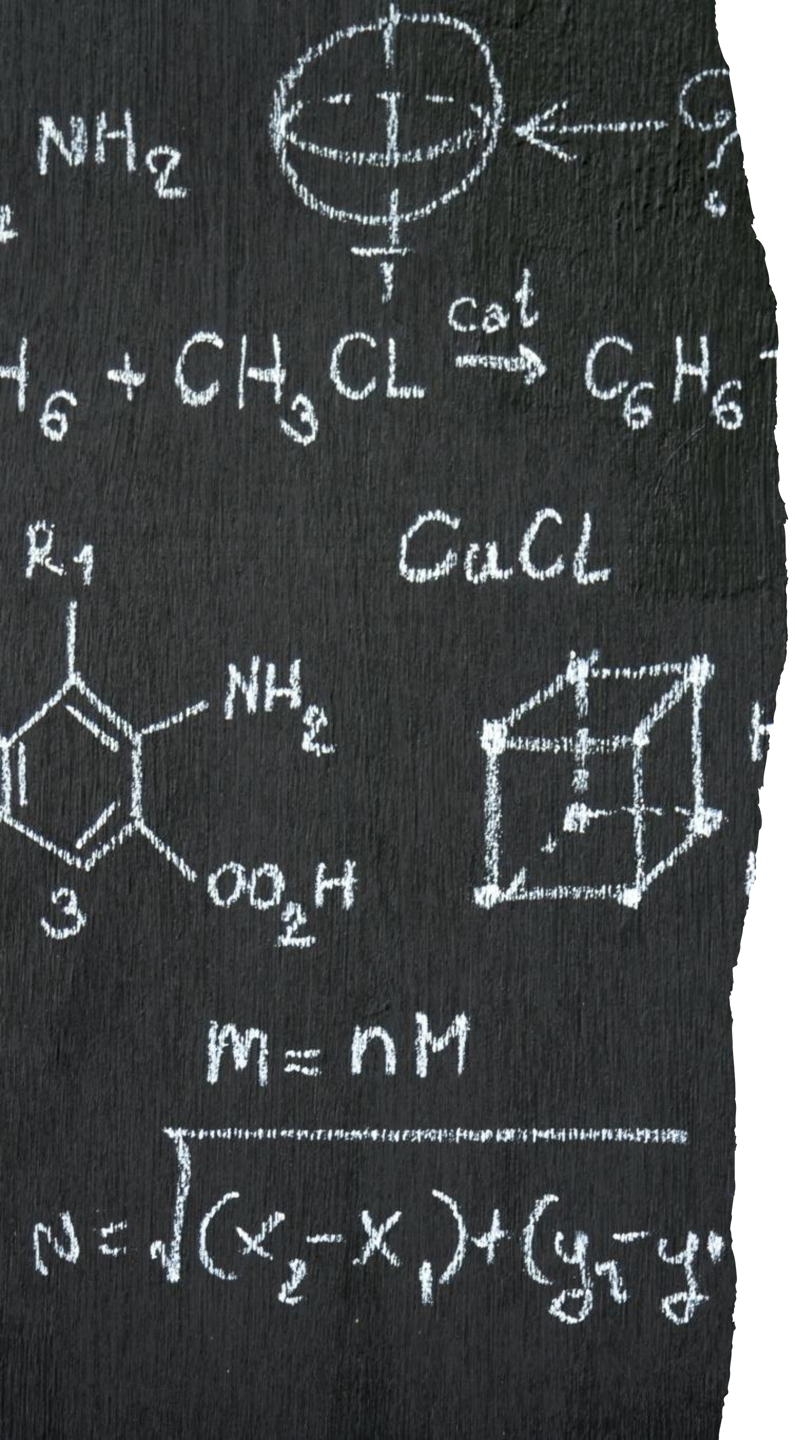
```
#region Lists & Sliders
// Used to hold a posible collection of values
// representing possible rides
// -------------------------------
// A Dictionary is a collection that represents
// a key and a value, you can define the datatype for both.
// In this example the key is an int and the value is a string
// -------------------------------
// Pretend this is a collection from a database
// The data to populate this Dictionary
// will be created in a separate method
private Dictionary<int, string> rides = [];
// Used to hold the selected value from the rides collection
private int myRide;
// Used to hold a possible list of string
// representing various vacation spots
private List<string> vacationSpots = [];
// Used to store the user's selected vacation spot.
private string vacationSpot = string.Empty;
// Used to hold the rating Value
private int reviewRating = 5;
#endregion
```

Adding the Slider

Basics.razor



```
    </label>
    <br />
    <!-- Label for the slider -->
    <label for="reviewRange" class="form-label">Rate the form control review (0 = bad, 10 = good)</label>
    <!-- Input element of type="range", bind to the reviewRating field -->
    <input type="range" class="form-range" id="reviewRange" @bind="reviewRating" min="0" max="10" steps="1" />
    <!-- Displays the selected value in the range to the user -->
    <h6>Rating: @reviewRating</h6>
/div>
```



Slider Properties

•**type="range"**: Specifies that this input element is of type "range," which is used for selecting a numeric value within a specified range.

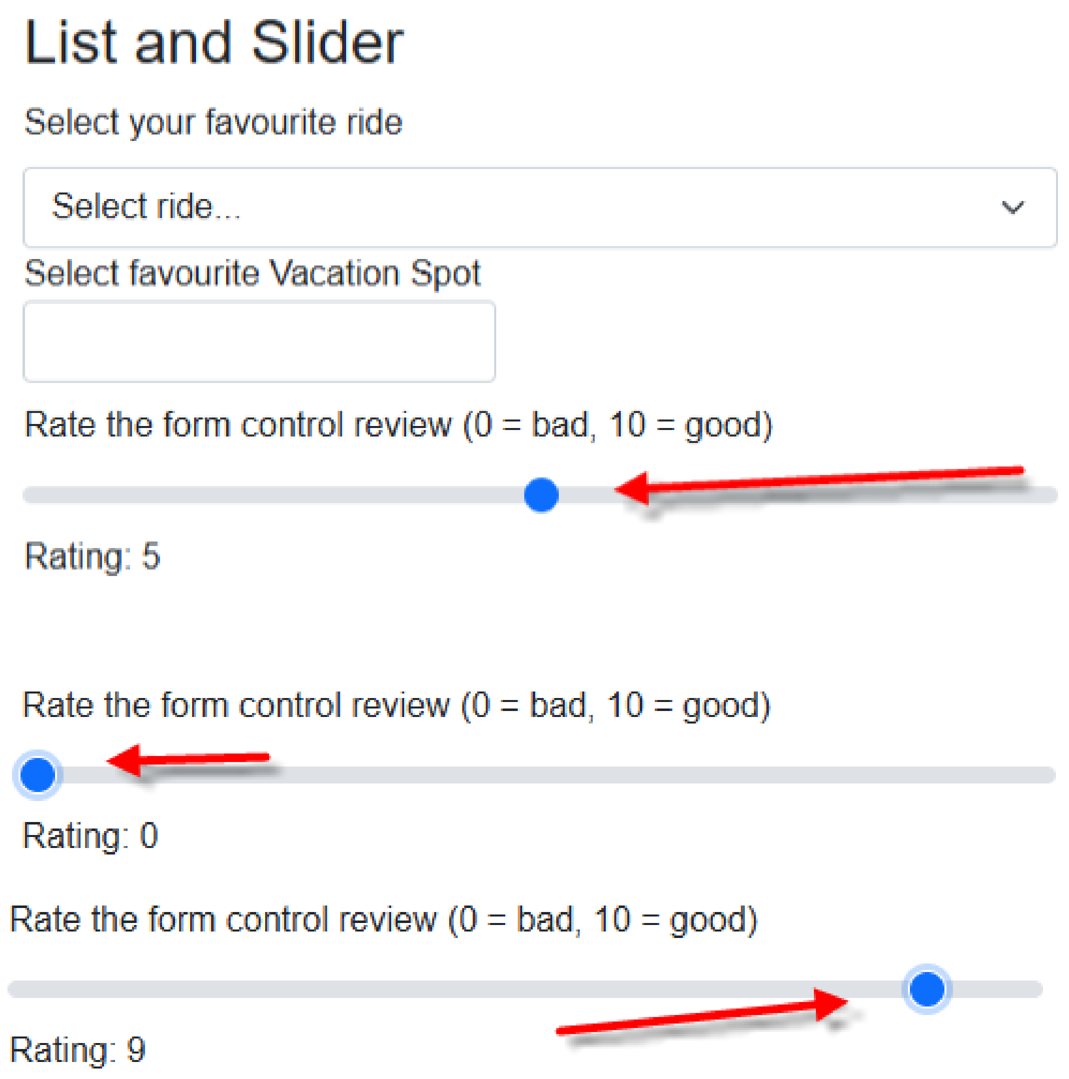
•**min="0"**: Sets the minimum value the user can select on the range. In this case, it's set to 0, indicating the lowest rating on the scale.

•**max="10"**: The maximum value the user can select on the range. Here, it's set to 10, indicating the highest rating on the scale.

•**step="1"**: Specifies the increment or step size between selectable values on the range. In this case, it's set to 1, meaning users can select whole numbers between the minimum and maximum values.

•**@bind="reviewRating"**: Specifies the field the input's value is bound to, we set this initially to equal 5 in the .cs file.

Blazor Page Output



List and Slider

Select your favourite ride

Select favourite Vacation Spot

Rate the form control review (0 = bad, 10 = good)

Rating: 5

Rate the form control review (0 = bad, 10 = good)

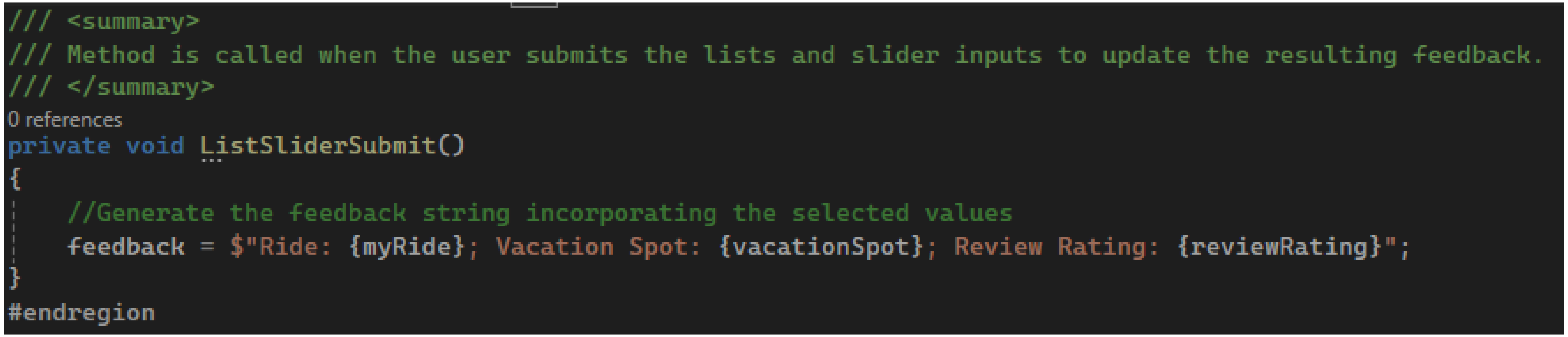
Rating: 0

Rate the form control review (0 = bad, 10 = good)

Rating: 9

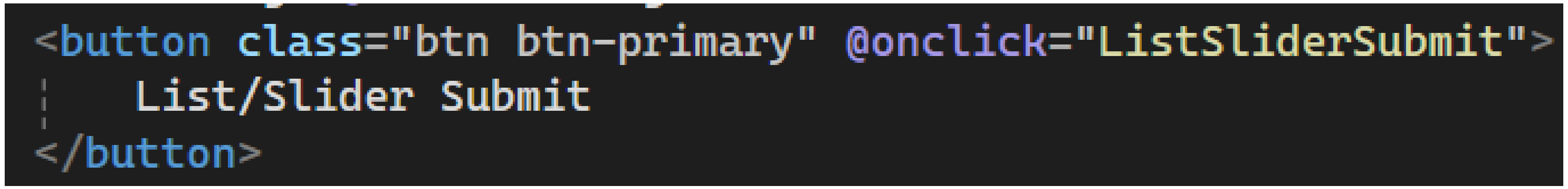
Coding the List and Slider Submit Method

Basics.razor.cs

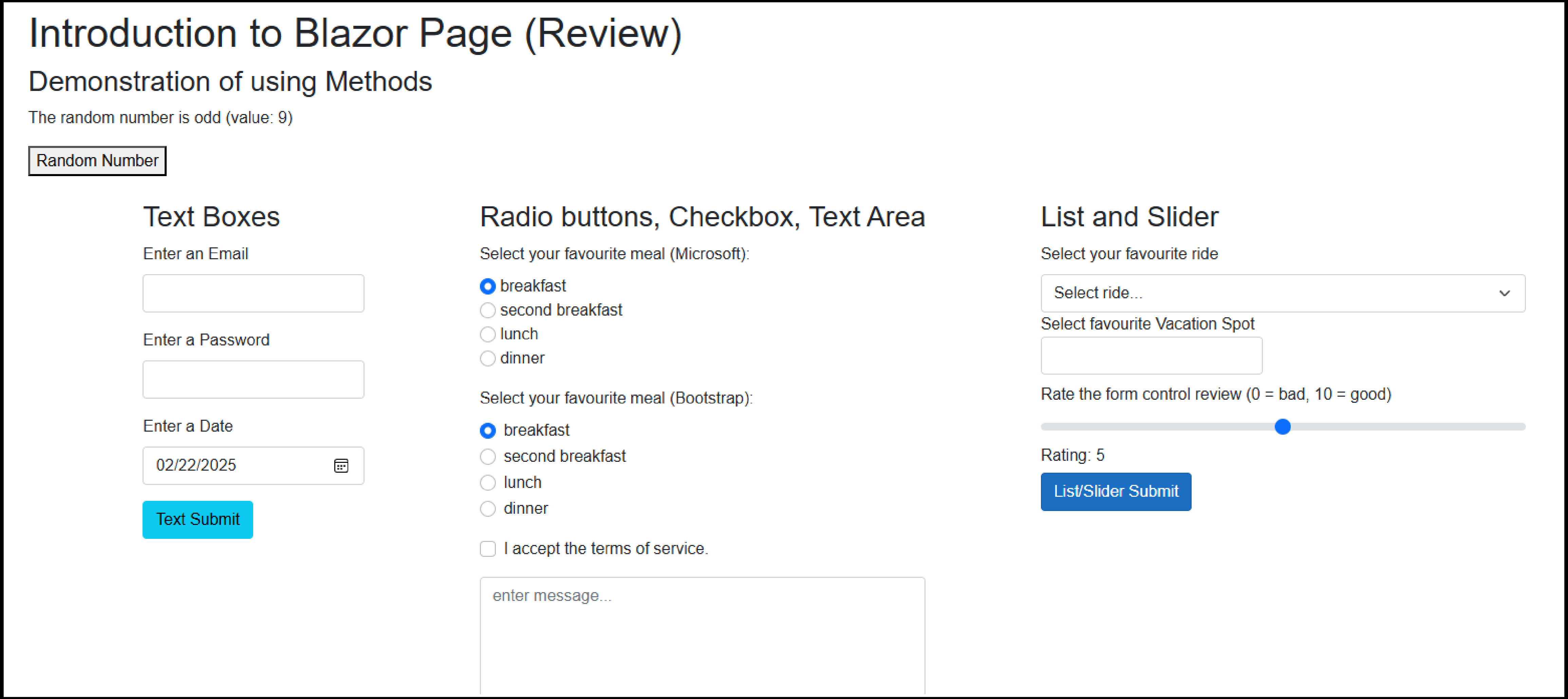


```
/// <summary>
/// Method is called when the user submits the lists and slider inputs to update the resulting feedback.
/// </summary>
0 references
private void ListSliderSubmit()
{
    //Generate the feedback string incorporating the selected values
    feedback = $"Ride: {myRide}; Vacation Spot: {vacationSpot}; Review Rating: {reviewRating}";
}
#endregion
```

Basics.razor



```
<button class="btn btn-primary" @onclick="ListSliderSubmit">
    List/Slider Submit
</button>
```



Introduction to Blazor Page (Review)

Demonstration of using Methods

The random number is odd (value: 9)

Random Number

Text Boxes

Enter an Email

Enter a Password

Enter a Date

02/22/2025

Text Submit

Radio buttons, Checkbox, Text Area

Select your favourite meal (Microsoft):

[✓ breakfast]
second breakfast
lunch
dinner

Select your favourite meal (Bootstrap):

[✓ breakfast]
second breakfast
lunch
dinner

I accept the terms of service.

enter message...

List and Slider

Select your favourite ride

Select ride...

Select favourite Vacation Spot

Rate the form control review (0 = bad, 10 = good)

Rating: 5

List/Slider Submit

Blazor Page Output

Statement Regarding Slide Accuracy and Potential Revisions

- Please note that the content of these PowerPoint slides is accurate to the best of my knowledge at the time of presentation. However, as new research and developments emerge, or to enhance the learning experience, these slides may be subject to updates or revisions in the future. I encourage you to stay engaged with the course materials and any announcements for the most current information