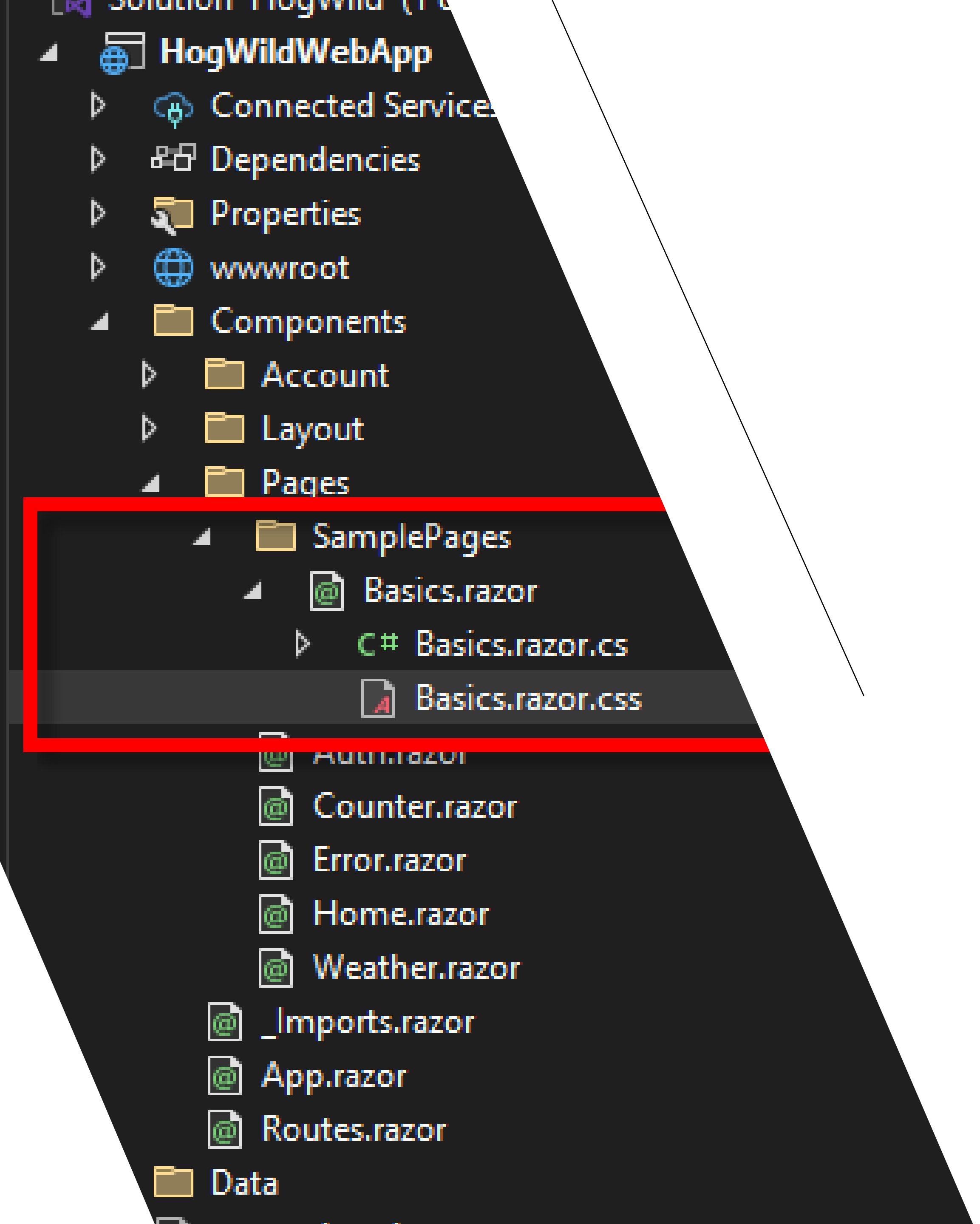
SAMPLE PAGES WITH MUDBLAZOR



```
Solution 'HogWild' (
  HogWildWebApp
    Connected Service
    Dependencies
    Properties
    wwwroot
    Components
      Account
      Layout
      Pages
        SamplePages
          Basics.razor
            Basics.razor.cs
            Basics.razor.css
        Auth.razor
        Counter.razor
        Error.razor
        Home.razor
        Weather.razor
      _Imports.razor
      App.razor
      Routes.razor
    Data
```

SAMPLE PAGES FOLDER AND BASICS PAGE

Follow the instructions from Blazor (Session 2 & 3) to create the SamplePages folder and Basics.razor, Basics.razor.cs, and Basics.razor.css files.

REFACTORING BASICS.RAZOR

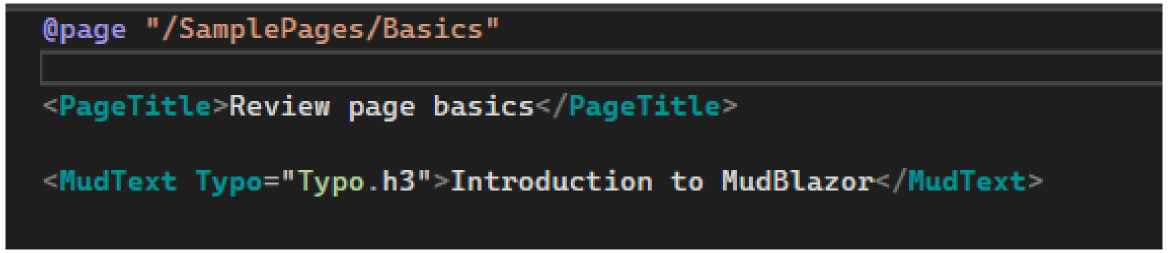
1. Add the @page directive and <PageTitle>
2. Instead of the <h1> HTML tag use a Component:

<MudText>

3. Use the Attribute Typo to indicate the text size that will be rendered. To create a <h3>

Typo="Typo.h3"

Basics.razor



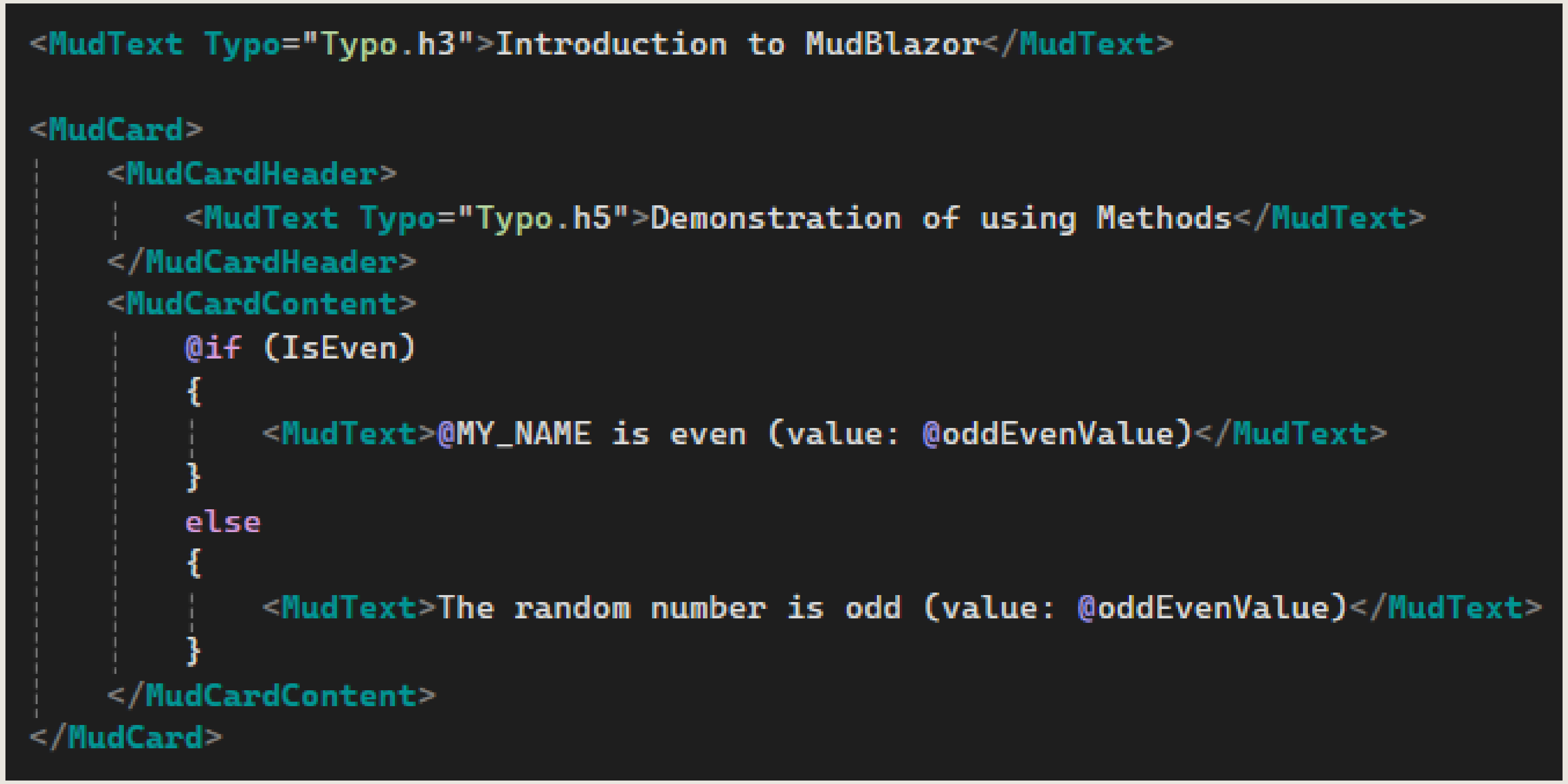
```
@page "/SamplePages/Basics"

<PageTitle>Review page basics</PageTitle>

<MudText Typo="Typo.h3">Introduction to MudBlazor</MudText>
```

UPDATING BASICS.RAZOR

Basics.razor



```
<MudText Typo="Typo.h3">Introduction to MudBlazor</MudText>

<MudCard>
    <MudCardHeader>
        <MudText Typo="Typo.h5">Demonstration of using Methods</MudText>
    </MudCardHeader>
    <MudCardContent>
        @if (IsEven)
        {
            <MudText>@MY_NAME is even (value: @oddEvenValue)</MudText>
        }
        else
        {
            <MudText>The random number is odd (value: @oddEvenValue)</MudText>
        }
    </MudCardContent>
</MudCard>
```

Use a **<MudCard>** component to create a quick and simple layout.

Wrap the Header in a **<MudCardHeader>** component.

Place the displayed text in a **<MudCardContent>** component:

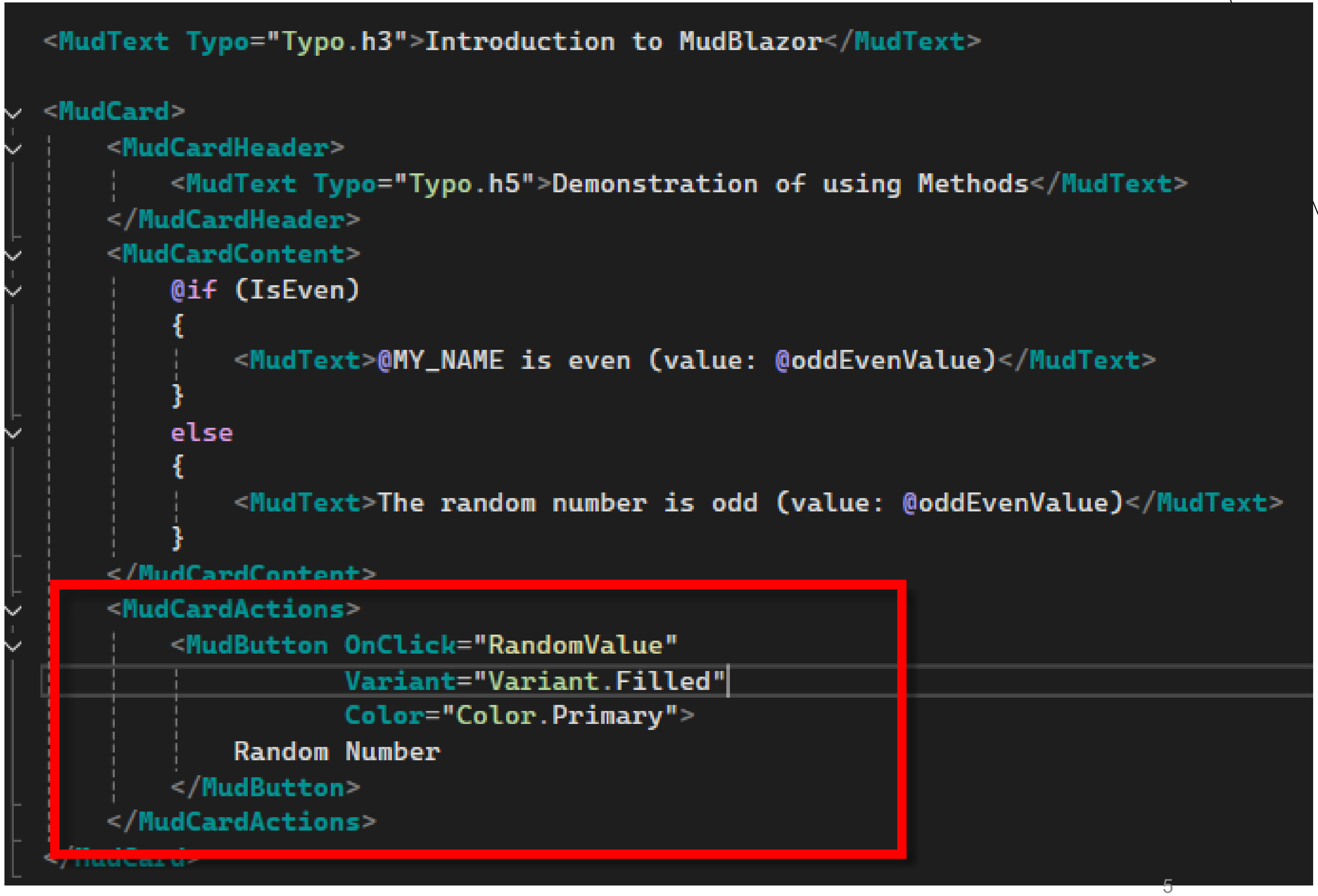
The same @if logic is used as the basic HTML project, but now we use **<MudText>** (which defaults to Typo.body1) instead of <p>

ADD A RANDOM NUMBER BUTTON

1. Add a <MudCardActions> section to your <MudCard>.
2. Place a <MudButton> inside and set the OnClick attribute to call your RandomValue method.

Notes:

- Variant – The type of button displayed. Variant.Filled is the typical thought of button with a darker background and lighter text.
- Color – Color is the color of the button, uses the default Color enum from MudBlazor.



```
<MudText Typo="Typo.h3">Introduction to MudBlazor</MudText>

<MudCard>
    <MudCardHeader>
        <MudText Typo="Typo.h5">Demonstration of using Methods</MudText>
    </MudCardHeader>
    <MudCardContent>
        @if (IsEven)
        {
            <MudText>@MY_NAME is even (value: @oddEvenValue)</MudText>
        }
        else
        {
            <MudText>The random number is odd (value: @oddEvenValue)</MudText>
        }
    </MudCardContent>
    <MudCardActions>
        <MudButton OnClick="RandomValue"
                   Variant="Variant.Filled"
                   Color="Color.Primary">
            Random Number
        </MudButton>
    </MudCardActions>
</MudCard>
```

UPDATE NAVMENU.RAZOR

Add a **MudNavGroup** with a Title attribute:

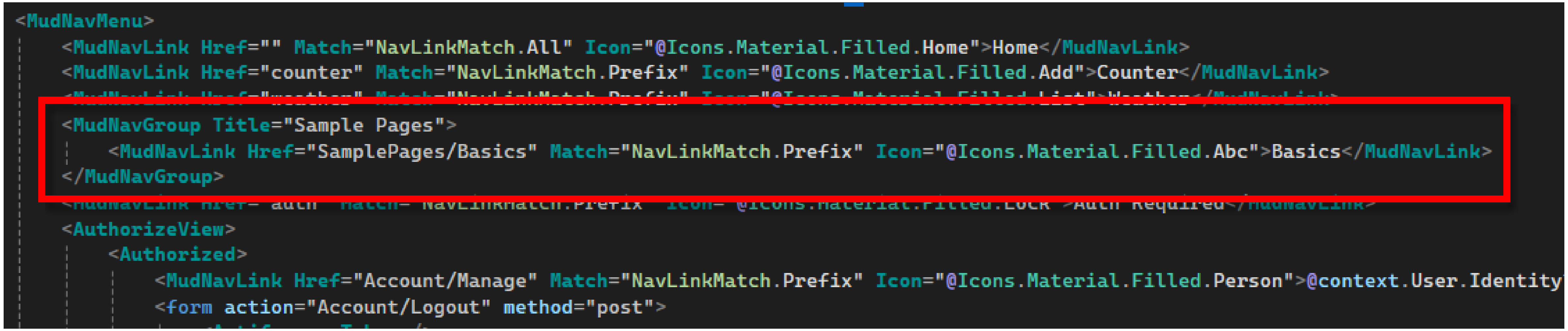
Title="Sample Pages"

Within the NavGroup add a new **MudNavLink** with the **Href** attribute linking to your Basics.razor page.

Note: You can change the icon to any icon you want within the MudBlazor Icon library:

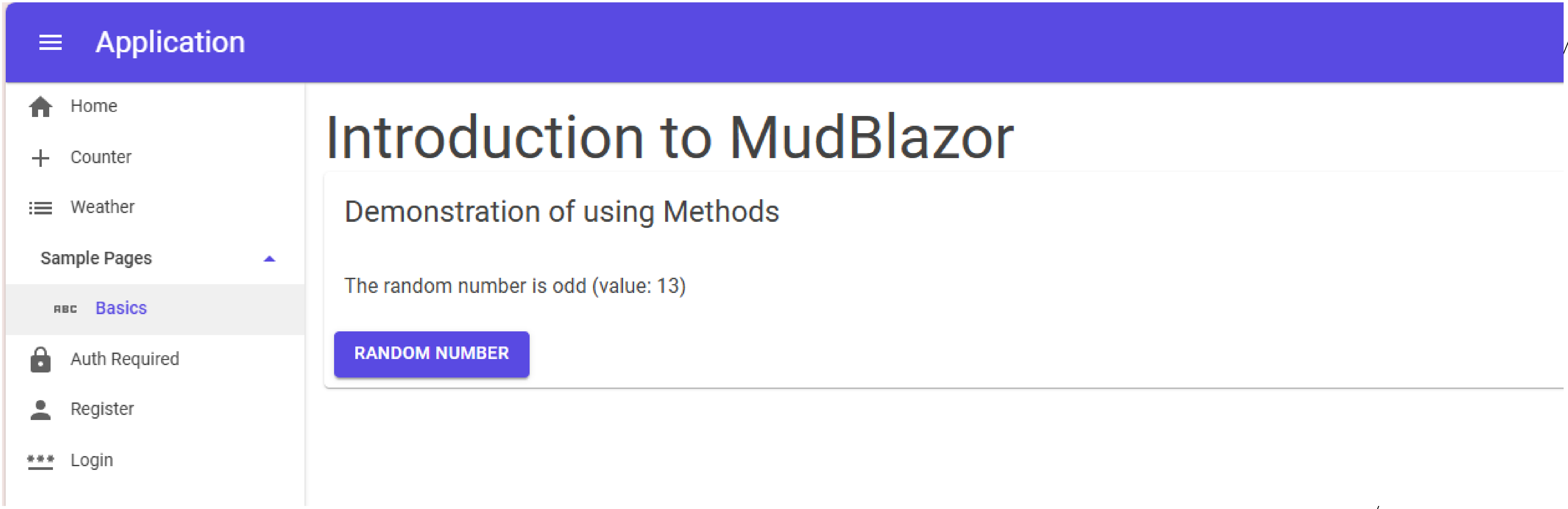
https://mudblazor.com/features/icons#icons

Layout\NavMenu.razor



```
<MudNavMenu>
    <MudNavLink Href="" Match="NavLinkMatch.All" Icon="@Icons.Material.Filled.Home">Home</MudNavLink>
    <MudNavLink Href="counter" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Add">Counter</MudNavLink>
    <MudNavLink Href="weather" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.List">Weather</MudNavLink>
    <MudNavGroup Title="Sample Pages">
        <MudNavLink Href="SamplePages/Basics" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Abc">Basics</MudNavLink>
    </MudNavGroup>
    <MudNavLink Href="auth" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Lock">Auth Required</MudNavLink>
    <AuthorizeView>
        <Authorized>
            <MudNavLink Href="Account/Manage" Match="NavLinkMatch.Prefix" Icon="@Icons.Material.Filled.Person">@context.User.Identity
            <form action="Account/Logout" method="post">
```

BLAZOR PAGE OUTPUT



Introduction to MudBlazor

Demonstration of using Methods

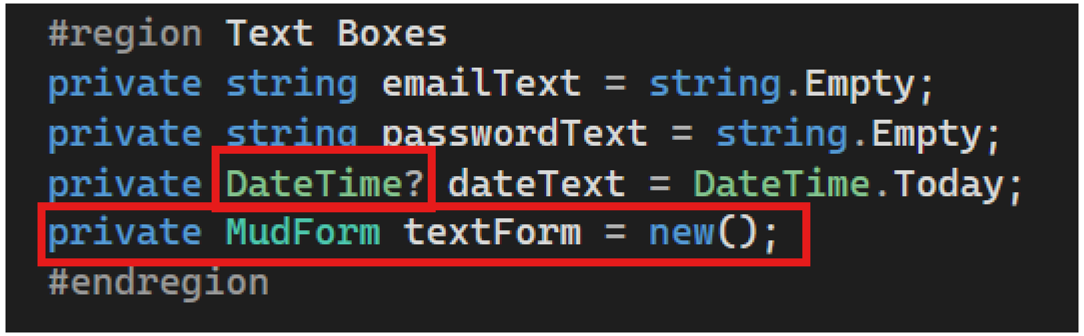
The random number is odd (value: 13)

TEXT BOX UPDATES – BASICS.RAZOR.CS

1. Update the dateText field to use DateTime?
2. Create a new Form Reference to use for form Validation
3. Update TextSubmit with a ternary operator to account for the nullable DateTime

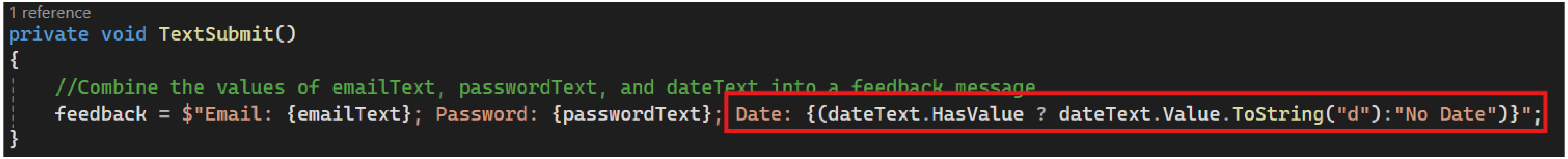
Note: When using a ternary operator inside String Interpolation, make sure to surround it with brackets inside the braces {(ternary)}

Basics.razor.cs



```
#region Text Boxes
private string emailText = string.Empty;
private string passwordText = string.Empty;
private DateTime? dateText = DateTime.Today;
private MudForm textForm = new();
#endregion
```

Basics.razor.cs

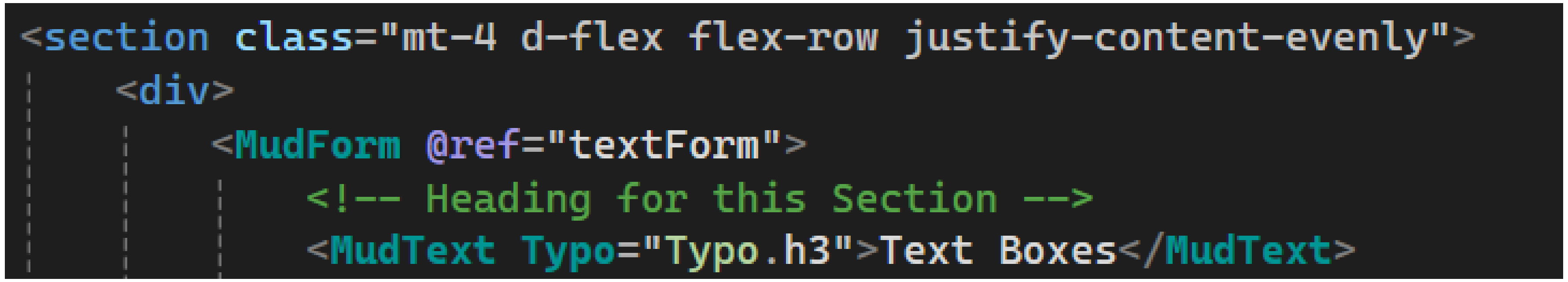


```
1 reference
private void TextSubmit()
{
    //Combine the values of emailText, passwordText, and dateText into a feedback message
    feedback = $"Email: {emailText}; Password: {passwordText}; Date: {(dateText.HasValue ? dateText.Value.ToString("d"):"No Date")}";
}
```

TEXT BOX UPDATES - MUDBLAZOR

1. Update the Heading to use <MudText> with Typo=“Typo.h3”
2. Add a <MudForm> surrounding the Fields with an attribute of @ref to reference the new textForm you created. Put this inside the existing <div>

Basics.razor



```
<section class="mt-4 d-flex flex-row justify-content-evenly">
    <div>
        <MudForm @ref="textForm">
            <!-- Heading for this Section -->
            <MudText Typo="Typo.h3">Text Boxes</MudText>
```

TEXT BOX UPDATES – CONT.

MudBlazor provides various input components for use within a form.

- MudTextField – Used for text inputs
- MudNumericField – Used for Numerical (Decimal, Int) input

Update the Text Fields to use MudTextFields, use the following Attributes

- InputType – Used to determine the type of input
- Label
- Variant – Determines how the text box is displayed
- @bind-Value – for binding a value
- Immediate =“true”
  - When set to true any validation is immediate and does not wait for the user to change focus from the field.

Optional:

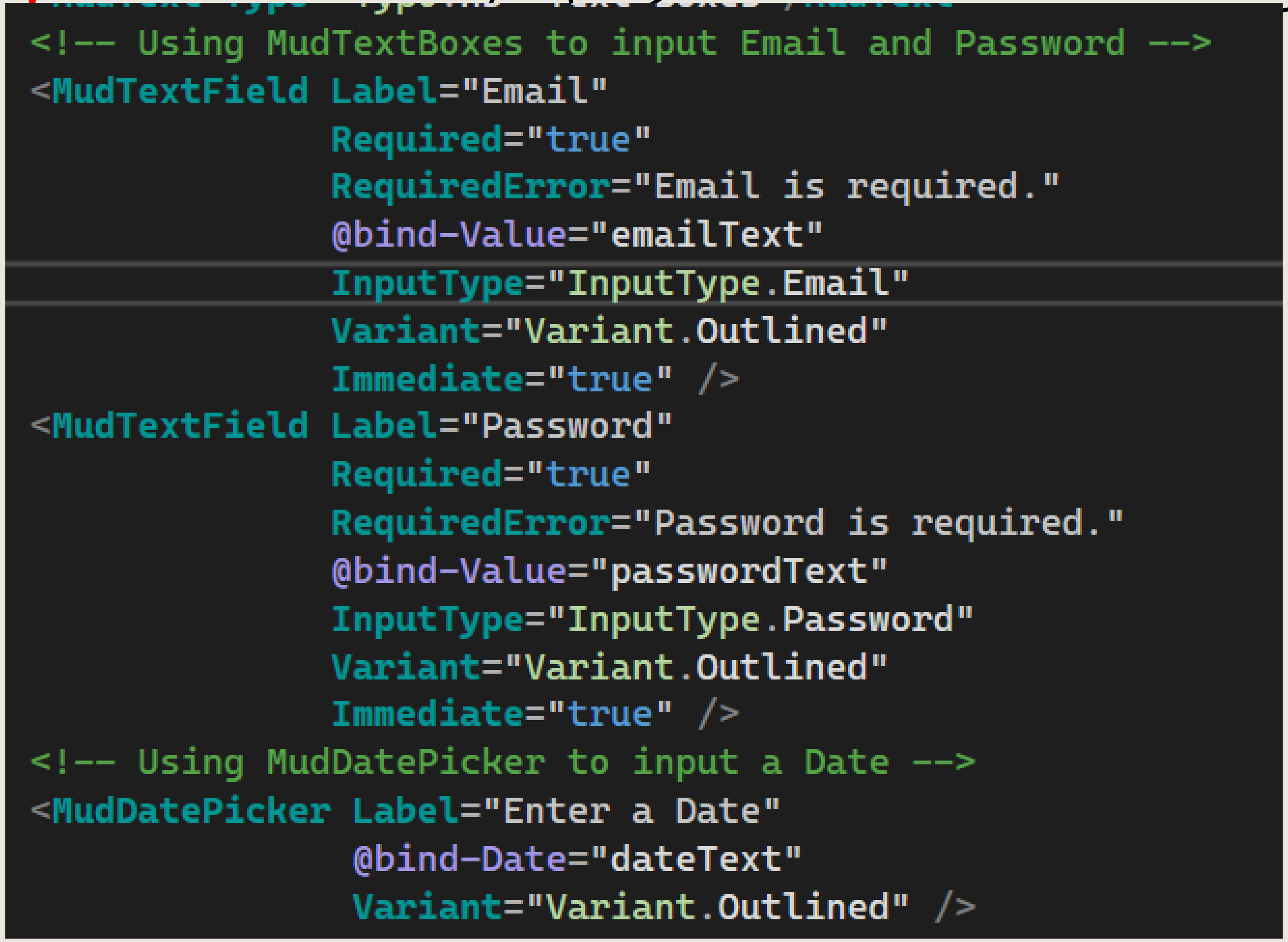
- Required=“true”
- RequiredError – The error text to display for simple validation

Update the Date Field to use MudDatePicker with the following Attributes:

- Label
- @bind-Date – for binding the selected Date, this must be bound to a DateTime? (nullable)
- Variant

TEXT BOX UPDATES – CONT.

Basics.razo
r



```
<!-- Using MudTextBoxes to input Email and Password -->
<MudTextField Label="Email"
              Required="true"
              RequiredError="Email is required."
              @bind-Value="emailText"
              InputType="InputType.Email"
              Variant="Variant.Outlined"
              Immediate="true" />
<MudTextField Label="Password"
              Required="true"
              RequiredError="Password is required."
              @bind-Value="passwordText"
              InputType="InputType.Password"
              Variant="Variant.Outlined"
              Immediate="true" />
<!-- Using MudDatePicker to input a Date -->
<MudDatePicker Label="Enter a Date"
               @bind-Date="dateText"
               Variant="Variant.Outlined" />
```

TEXT BOX UPDATES – CONT (BUTTON)

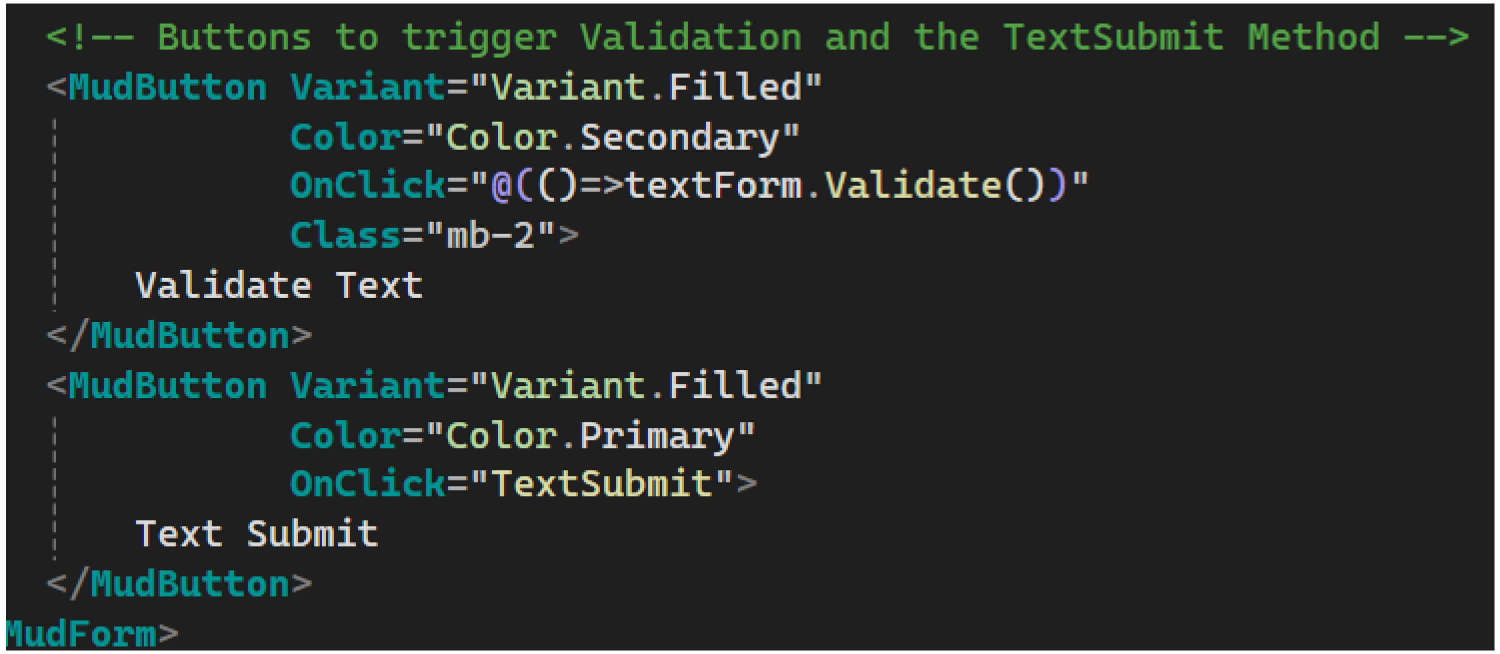
Update the button to a <MudButton> with the following attributes:

- `Variant="Variant.Filled"`
- `Color="Color.Primary“`
- `OnClick="TextSubmit“`

Add an additional button for triggering Validation above the Submit Button with:

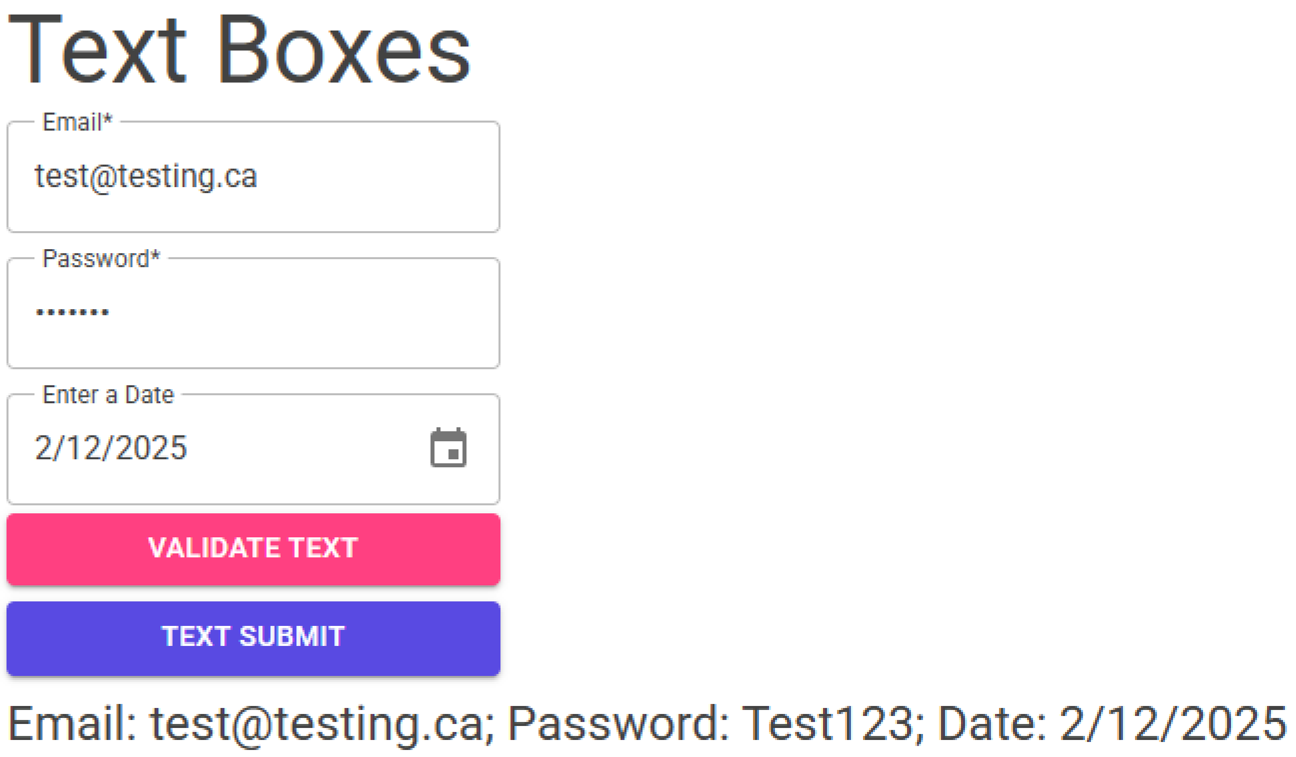
- OnClick="@(()=>textForm.Validate())“
- Class="mb-2“ (to add some spacing)

Basics.razor



```
 <!-- Buttons to trigger Validation and the TextSubmit Method -->
 <MudButton Variant="Variant.Filled"
            Color="Color.Secondary"
            OnClick="@(()=>textForm.Validate())"
            Class="mb-2">
     Validate Text
 </MudButton>
 <MudButton Variant="Variant.Filled"
            Color="Color.Primary"
            OnClick="TextSubmit">
     Text Submit
 </MudButton>
MudForm>
```

TEXT BOX UPDATES – RESULTS



Text Boxes

Email*

test@testing.ca

Password*

•••••••

Enter a Date

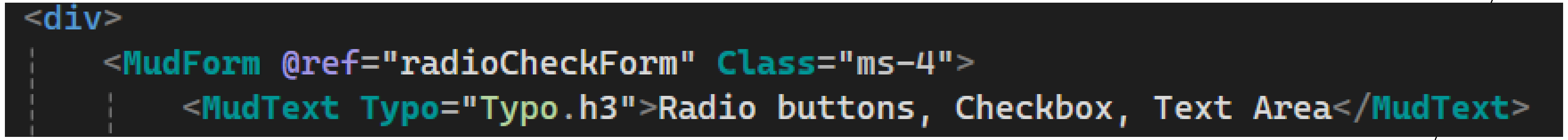
2/12/2025

Email: test@testing.ca; Password: Test123; Date: 2/12/2025

RADIO BUTTONS, CHECKBOXES, TEXT AREA - UPDATES

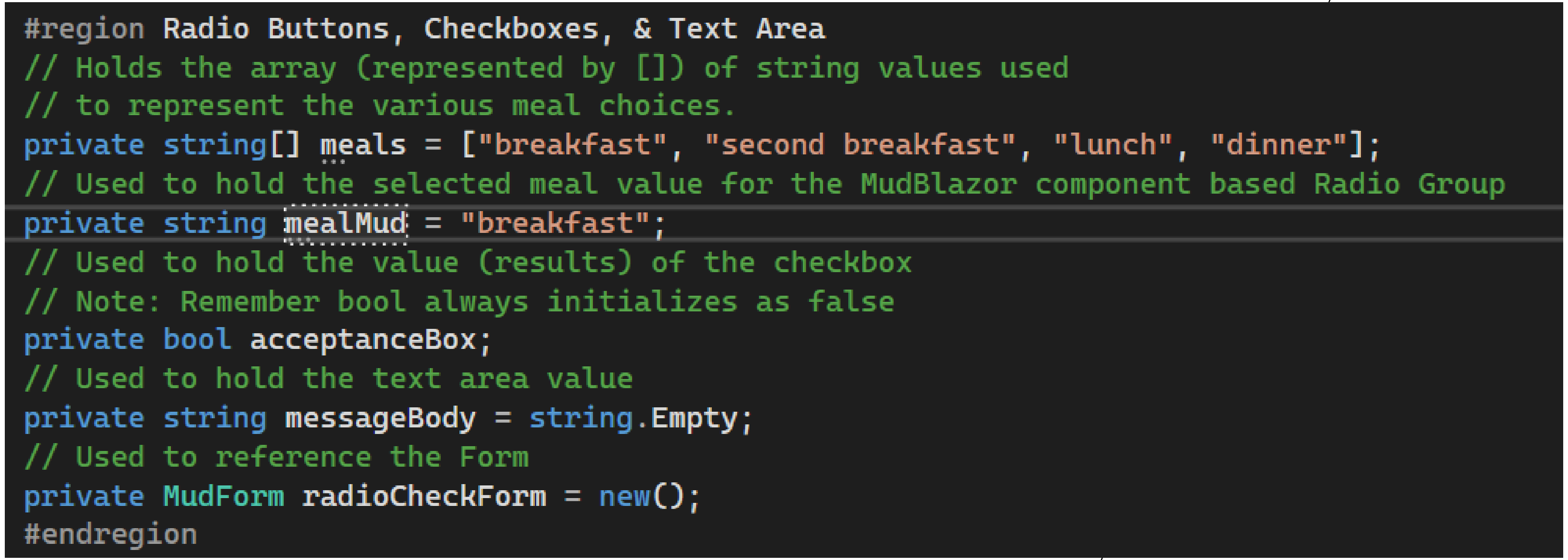
1. **Add a new form reference for this form.**
2. **Update mealMicrosoft to mealMud and remove mealBootstrap**
3. **Surround the fields with the new MudForm inside of the <div> and update the header to MudText**

Basics.razor



```
<div>
    <MudForm @ref="radioCheckForm" Class="ms-4">
        <MudText Typo="Typo.h3">Radio buttons, Checkbox, Text Area</MudText>
```

Basics.razor.cs

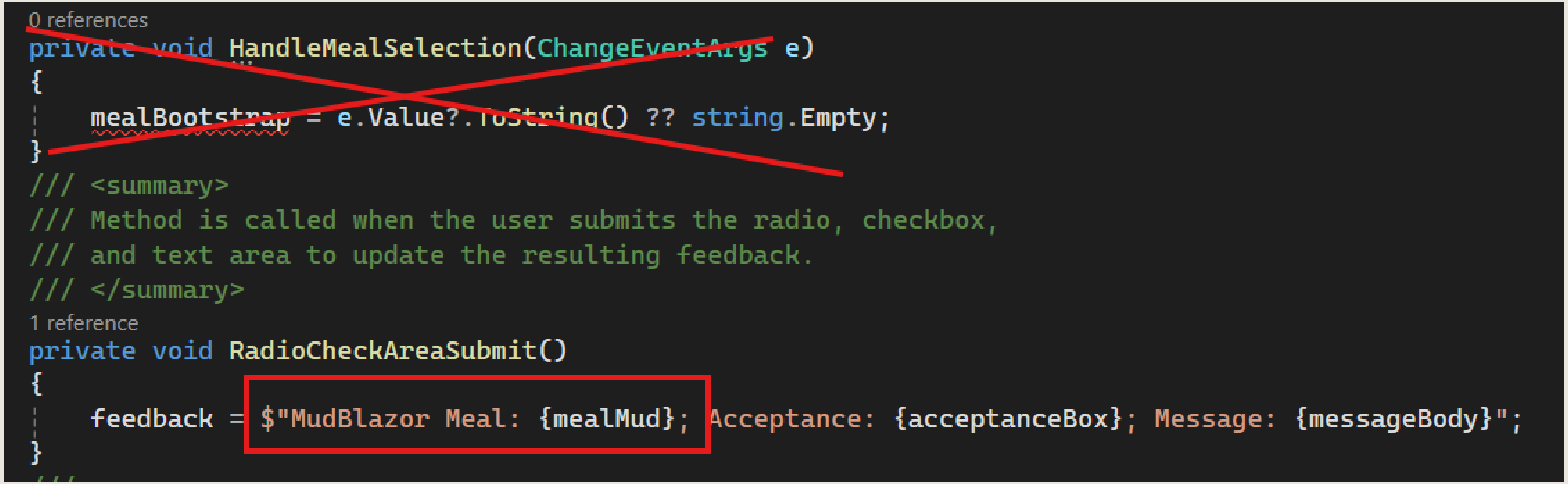


```
#region Radio Buttons, Checkboxes, & Text Area
// Holds the array (represented by []) of string values used
// to represent the various meal choices.
private string[] meals = ["breakfast", "second breakfast", "lunch", "dinner"];
// Used to hold the selected meal value for the MudBlazor component based Radio Group
private string mealMud = "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;
// Used to reference the Form
private MudForm radioCheckForm = new();
#endregion
```

RADIO BUTTONS, CHECKBOXES, TEXT AREA – CONT.

We will also need to update the RadioCheckAreaSubmit Method and remove the HandleMealSelection Method

Basics.razor.cs

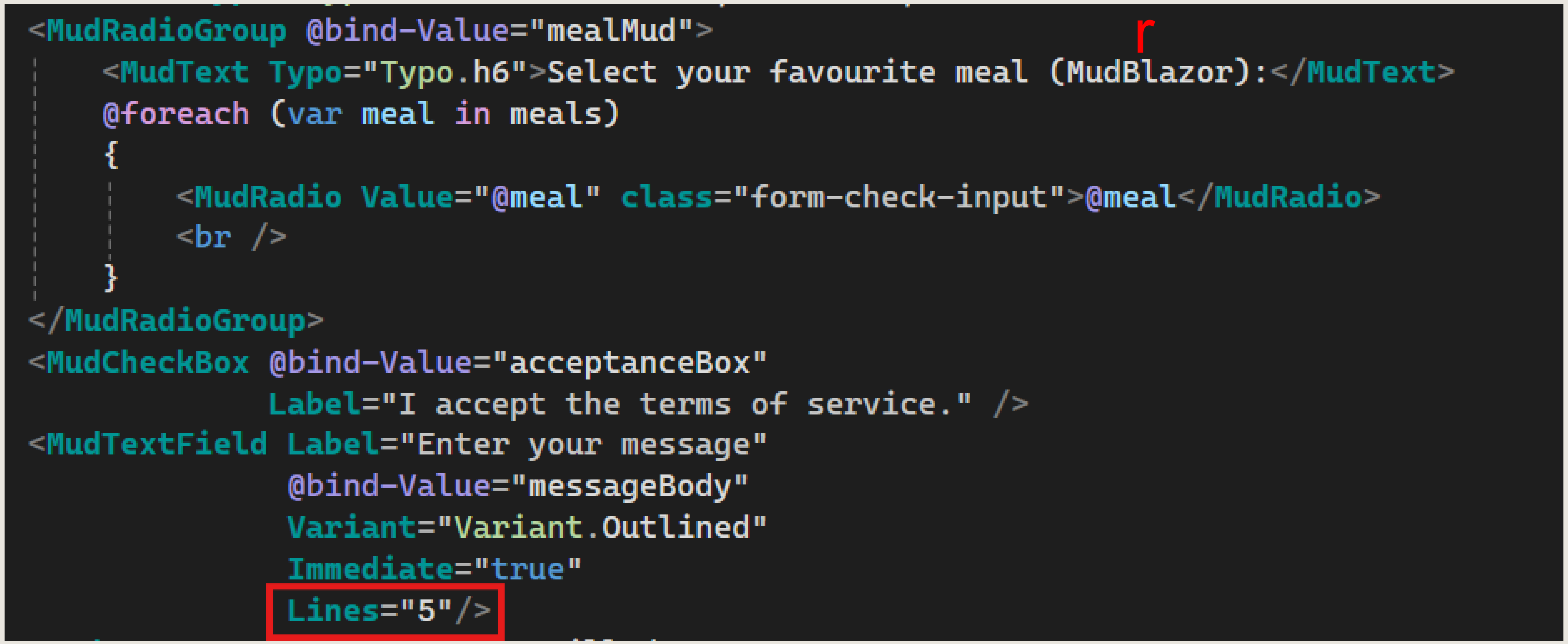


```
0 references
~~private void HandleMealSelection(ChangeEventArgs e)~~
~~{~~
~~    mealBootstrap = e.Value?.ToString() ?? string.Empty;~~
~~}~~
/// <summary>
/// Method is called when the user submits the radio, checkbox,
/// and text area to update the resulting feedback.
/// </summary>
1 reference
private void RadioCheckAreaSubmit()
{
    feedback = $"MudBlazor Meal: {mealMud}; Acceptance: {acceptanceBox}; Message: {messageBody}";
}
```

RADIO BUTTONS, CHECKBOXES, TEXT AREA – CONT.

- MudBlazor has a similar RadioButton component like the Microsoft provided one. <MudRadioGroup> functions with <MudRadio> children to make a nice Radio Group.
- Let's change the Microsoft Radio Group to a MudRadioGroup and update the Submit function to use it. We will also remove the second radio group.

Basics.razor



```
<MudRadioGroup @bind-Value="mealMud">
    <MudText Typo="Typo.h6">Select your favourite meal (MudBlazor):</MudText>
    @foreach (var meal in meals)
    {
        <MudRadio Value="@meal" class="form-check-input">@meal</MudRadio>
        <br />
    }
</MudRadioGroup>
<MudCheckBox @bind-Value="acceptanceBox"
             Label="I accept the terms of service." />
<MudTextField Label="Enter your message"
              @bind-Value="messageBody"
              Variant="Variant.Outlined"
              Immediate="true"
              Lines="5"/>
```

- MudBlazor also provides a checkbox component aptly called <MudCheckBox> which needs a binded value and a Label.
- We can also use the <MudTextField> to make multi-line text fields by adding the 'Lines' attribute.

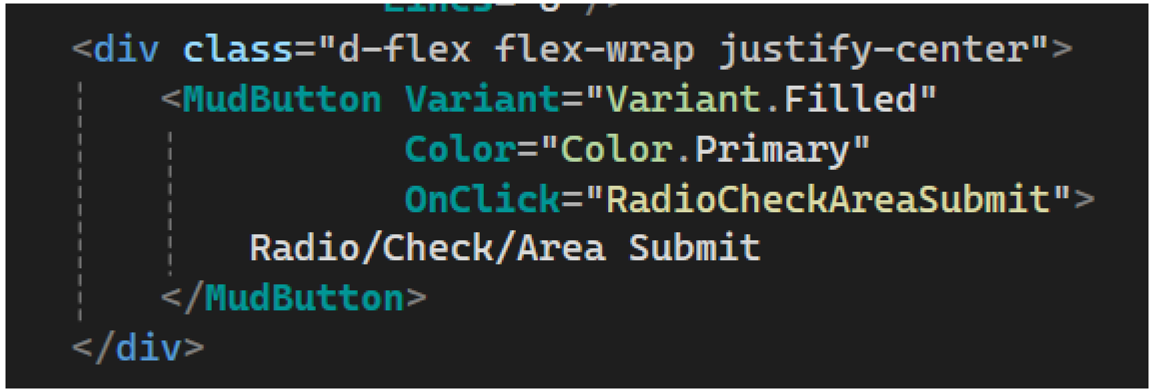
RADIO BUTTONS, CHECKBOXES, TEXT AREA - CONT

Let’s also update the button to use a <MudButton>.

We can put it in a <div> that uses a flex to center the button with:

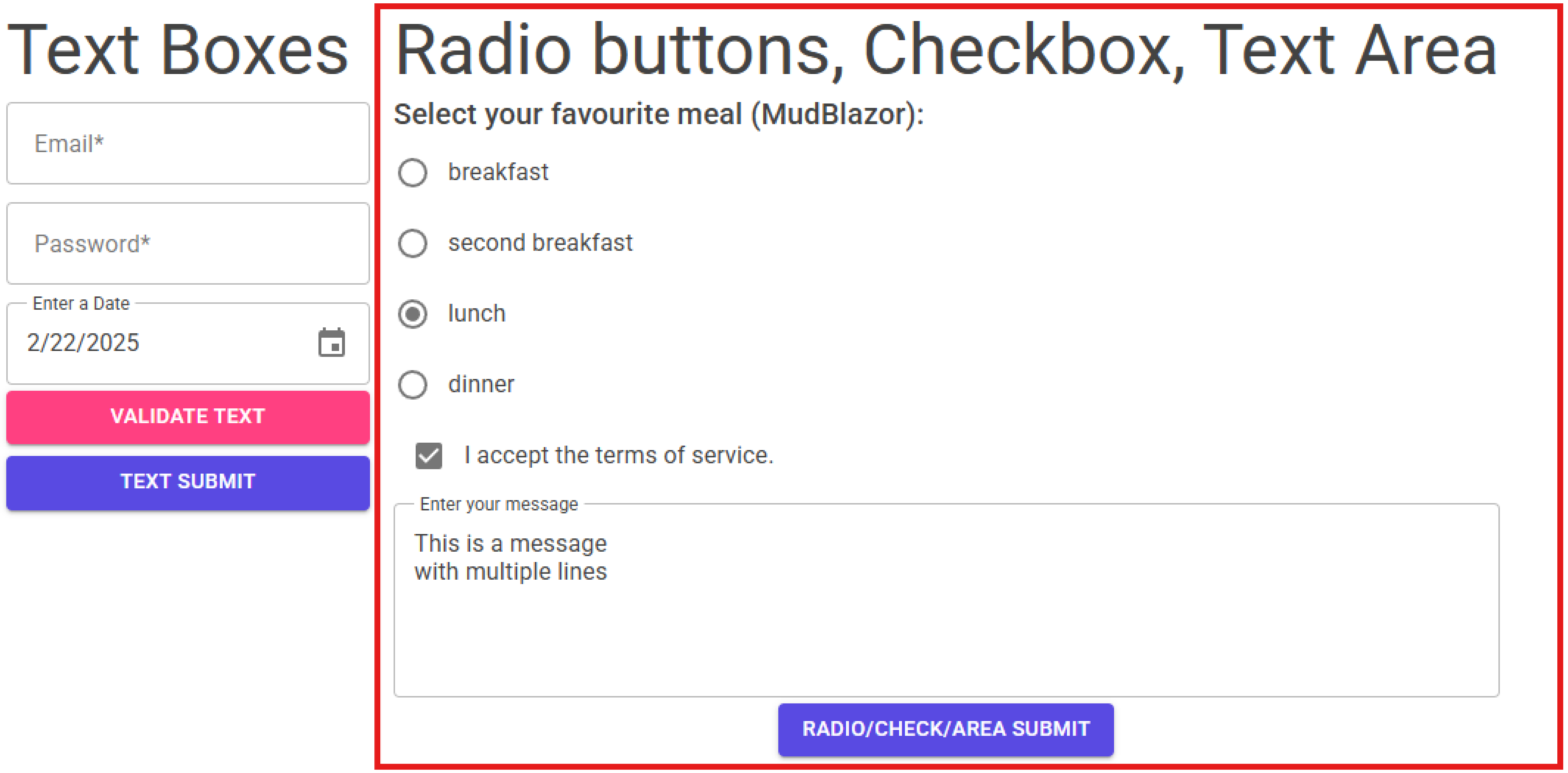
- class=“d-flex flex-wrap justify-center”

Basics.razor



```
<div class="d-flex flex-wrap justify-center">
    <MudButton Variant="Variant.Filled"
               Color="Color.Primary"
               OnClick="RadioCheckAreaSubmit">
        Radio/Check/Area Submit
    </MudButton>
</div>
```

RADIO BUTTONS, CHECKBOXES, TEXT AREA– RESULTS



MudBlazor Meal: lunch; Acceptance: True; Message: This is a message with multiple lines

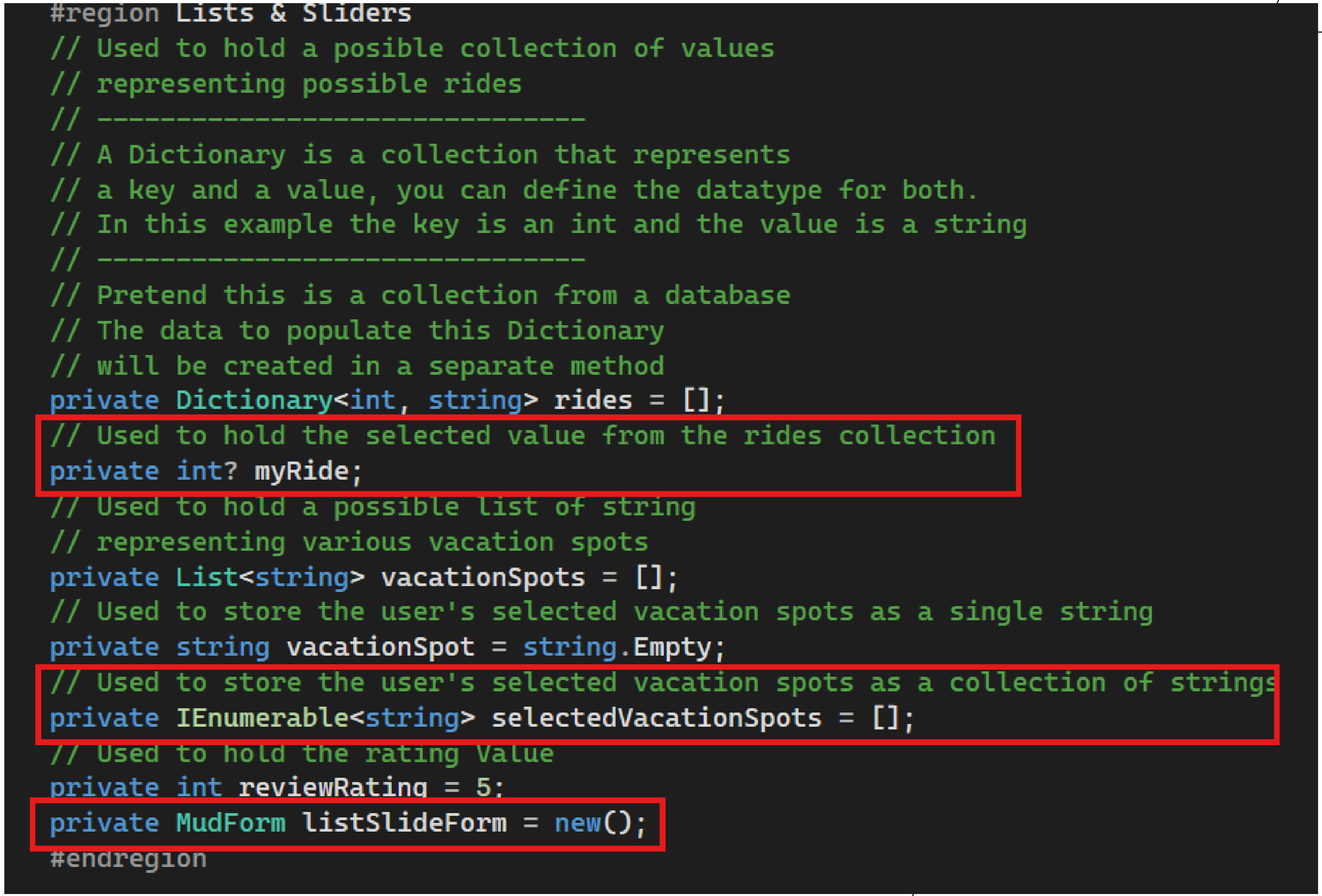
LIST AND SLIDERS - UPDATES

Updating the Lists to <MudSelect> elements will require some preparation.

1. **Add a new MudForm field called listSlideForm**
2. **Update myRide to a nullable int – add the ?**
3. **Add a selectedVacationSpots field to hold multi-select of Vacation spot values as a Ienumerable**

Note: MudBlazor for Selects does not work with List<> to hold selected values.

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 spots as a single string
private string vacationSpot = string.Empty;
// Used to store the user's selected vacation spots as a collection of strings
private IEnumerable<string> selectedVacationSpots = [];
// Used to hold the rating Value
private int reviewRating = 5;
private MudForm listSlideForm = new();
#endregion
```

LIST AND SLIDERS – CONT (MYRIDE)

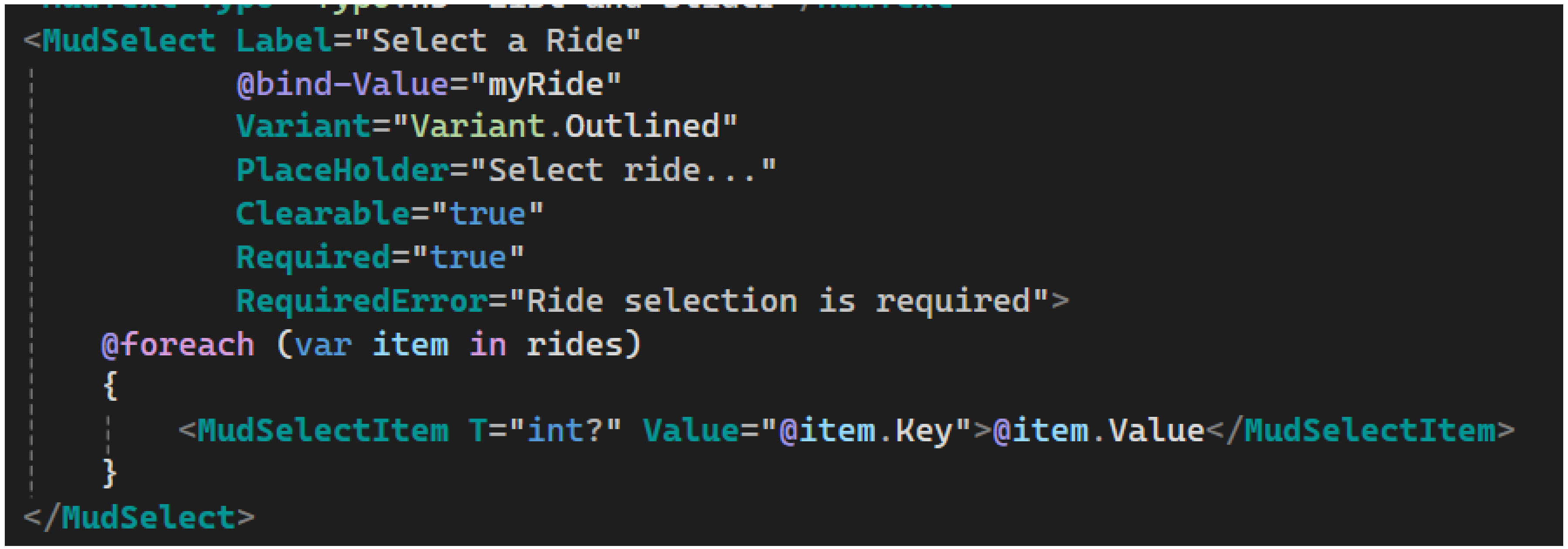
Using a <MudSelect> for myRide we can set the following attributes:

- Label="Select a Ride"
  - To give it a label
- @bind-Value="myRide"
  - To bind to the code behind field
- Variant="Variant.Outline"
  - To determine the display properties
- PlaceHolder="Select ride…"
  - To display something when there is no selection
- Clearable="true"
  - Optional: Allows the user to clear the selection
- Required="true"
  - Set to true if you want the field to be required
- RequiredError="Ride select is required"
  - The error is display when there is no selection

We also need to remember to use <MudSelectItem> and since this is a nullable int you must set T="int?" which tell the MudSelect the selection is allowed for that datatype.

LIST AND SLIDERS – CONT (MYRIDE)

Basics.razor



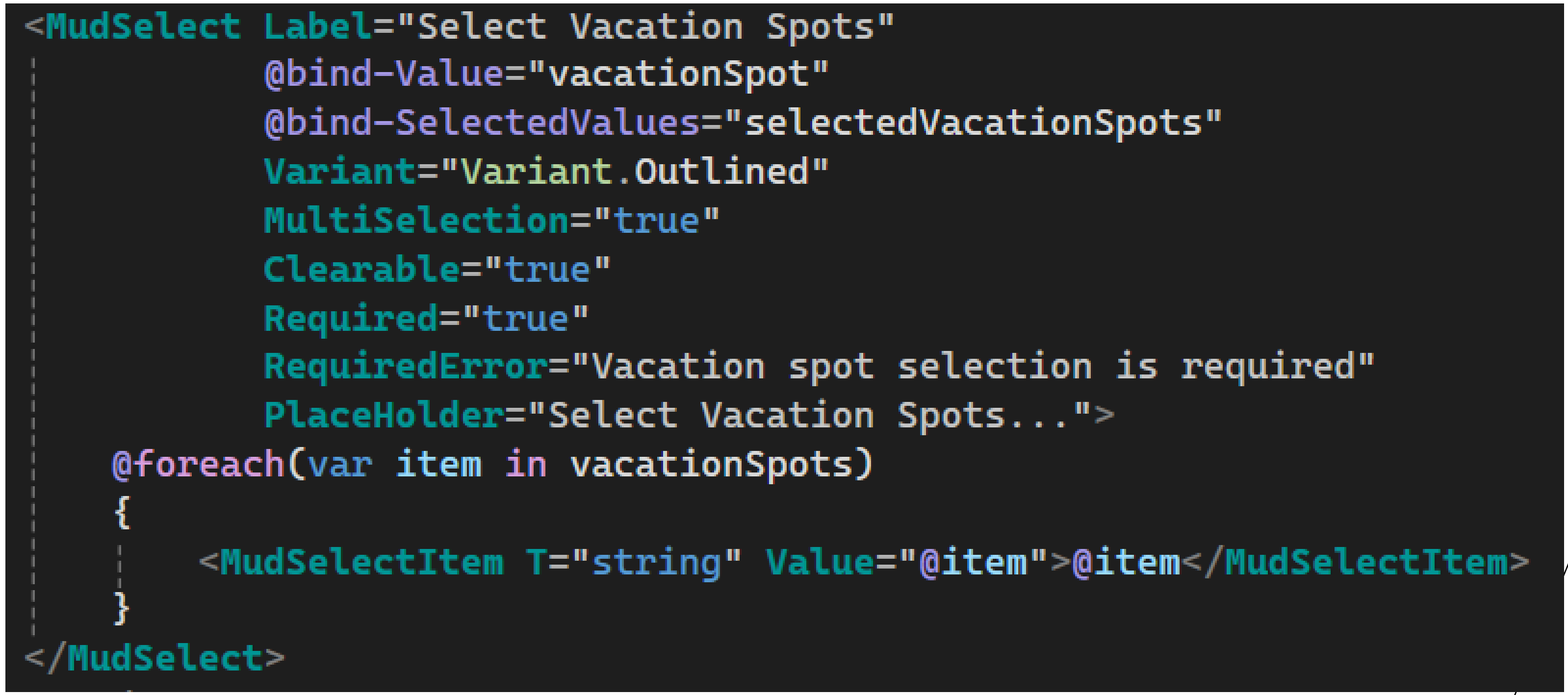
```
<MudSelect Label="Select a Ride"
           @bind-Value="myRide"
           Variant="Variant.Outlined"
           PlaceHolder="Select ride..."
           Clearable="true"
           Required="true"
           RequiredError="Ride selection is required">
    @foreach (var item in rides)
    {
        <MudSelectItem T="int?" Value="@item.Key">@item.Value</MudSelectItem>
    }
</MudSelect>
```

LIST AND SLIDERS – CONT (VACATION SPOTS)

Using a <MudSelect> for Vacation spots but we will make this Multiselect and set the following additional attributes (on top of the ones we know to set from the ride selection):

- @bind-SelectedValues=“selectedVacationSpots”
  - To bind to the IEnumerable collection
  - Note: You do not need to set @bind-Value AND @bind-SelectedValue, this depends on your use case for the multi-select field.

Basics.razor



```
<MudSelect Label="Select Vacation Spots"
           @bind-Value="vacationSpot"
           @bind-SelectedValues="selectedVacationSpots"
           Variant="Variant.Outlined"
           MultiSelection="true"
           Clearable="true"
           Required="true"
           RequiredError="Vacation spot selection is required"
           PlaceHolder="Select Vacation Spots...">
    @foreach(var item in vacationSpots)
    {
        <MudSelectItem T="string" Value="@item">@item</MudSelectItem>
    }
</MudSelect>
```

LIST AND SLIDERS – CONT (SLIDER)

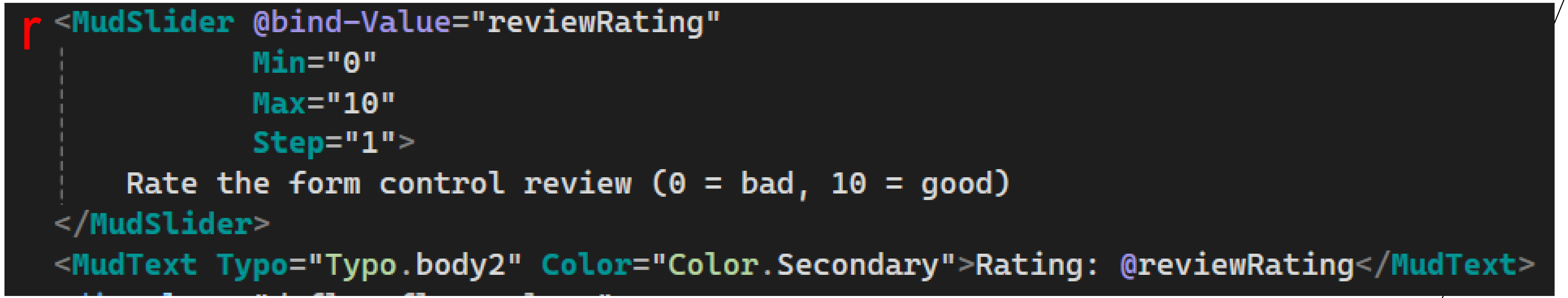
Sliders are simple and require a <MudSlider> we can set the following attributes to control min, max, and the step amount:

- Min="0"
- Max="10"
- Step="1"

Note: Step is not needed here since it is an integer, for doubles (decimals) you may want to set this to restrict the steps to 0.5 or something else.

We will also change the Review display to a MudText of body2 (for a different look) and we can even give it a colour with the Color attribute.

Basics.razo
r



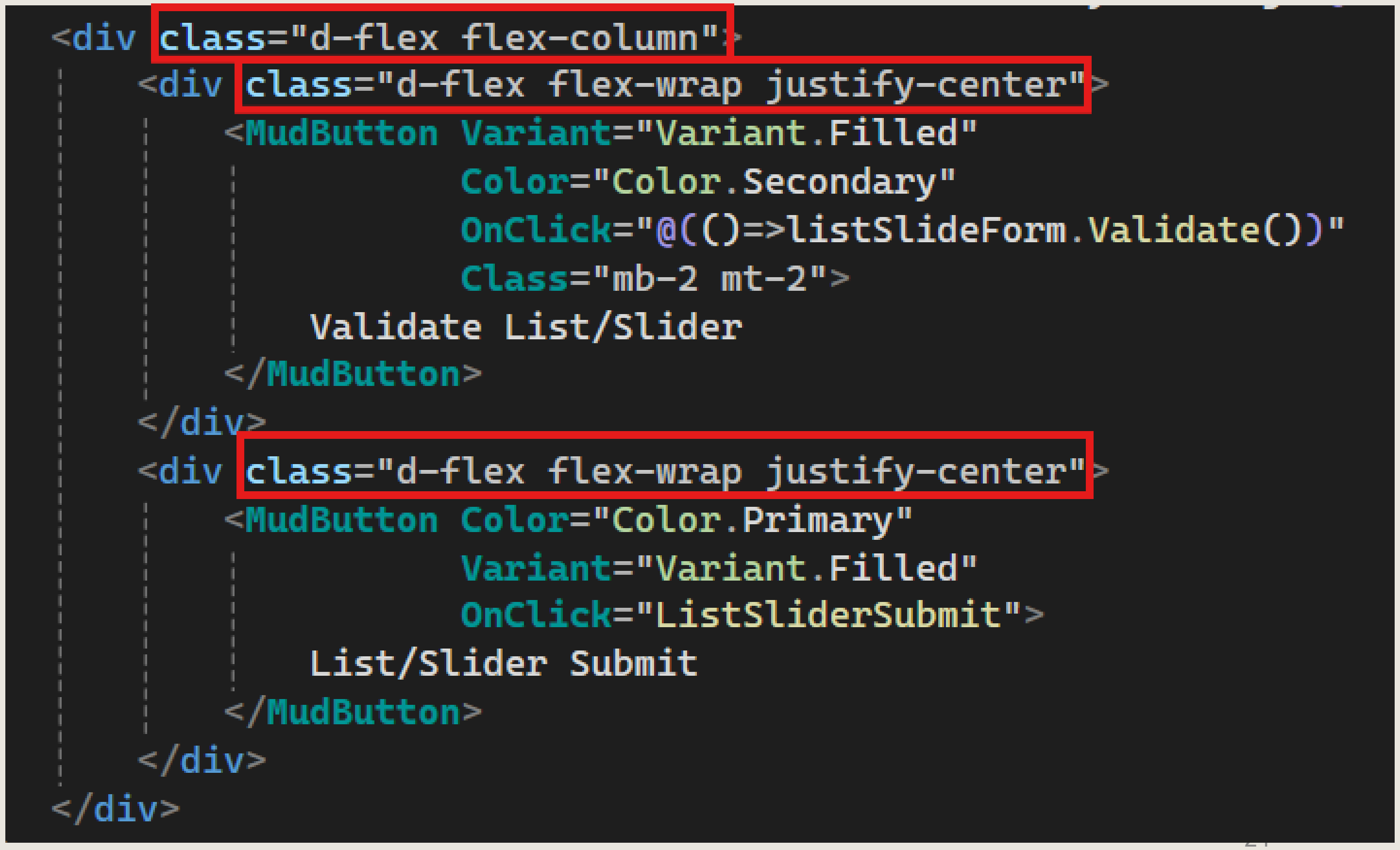
```
<MudSlider @bind-Value="reviewRating"
           Min="0"
           Max="10"
           Step="1">
    Rate the form control review (0 = bad, 10 = good)
</MudSlider>
<MudText Typo="Typo.body2" Color="Color.Secondary">Rating: @reviewRating</MudText>
```

LIST AND SLIDERS – CONT (BUTTONS)

We will include a Validate test button for this form. We can take advantage of flex to centre the display of the buttons using multiple <div> tags and different classes

- d-flex – makes the dev a flexbox
- flex-column – makes the display of block elements like divs inside the div column format (vertical)
- flex-wrap – add the wrap capabilities to the flex-box
- justify-center – makes the elements inside the div horizontally centered.

Basics.razo



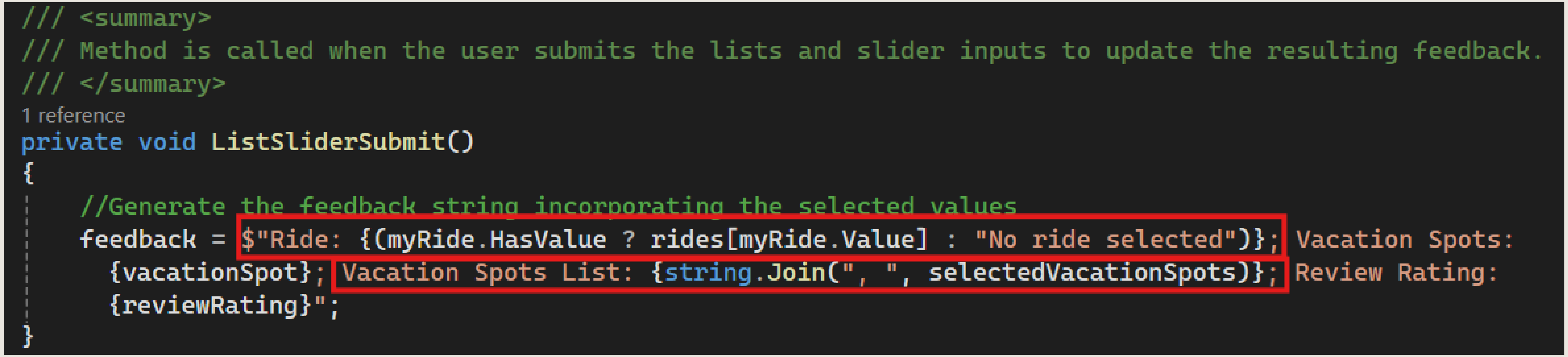
```
<div class="d-flex flex-column">
    <div class="d-flex flex-wrap justify-center">
        <MudButton Variant="Variant.Filled"
                   Color="Color.Secondary"
                   OnClick="@(()=>listSlideForm.Validate())"
                   Class="mb-2 mt-2">
            Validate List/Slider
        </MudButton>
    </div>
    <div class="d-flex flex-wrap justify-center">
        <MudButton Color="Color.Primary"
                   Variant="Variant.Filled"
                   OnClick="ListSliderSubmit">
            List/Slider Submit
        </MudButton>
    </div>
</div>
```

LIST AND SLIDERS – CONT

We need to also update the Submit Method for List and Sliders.

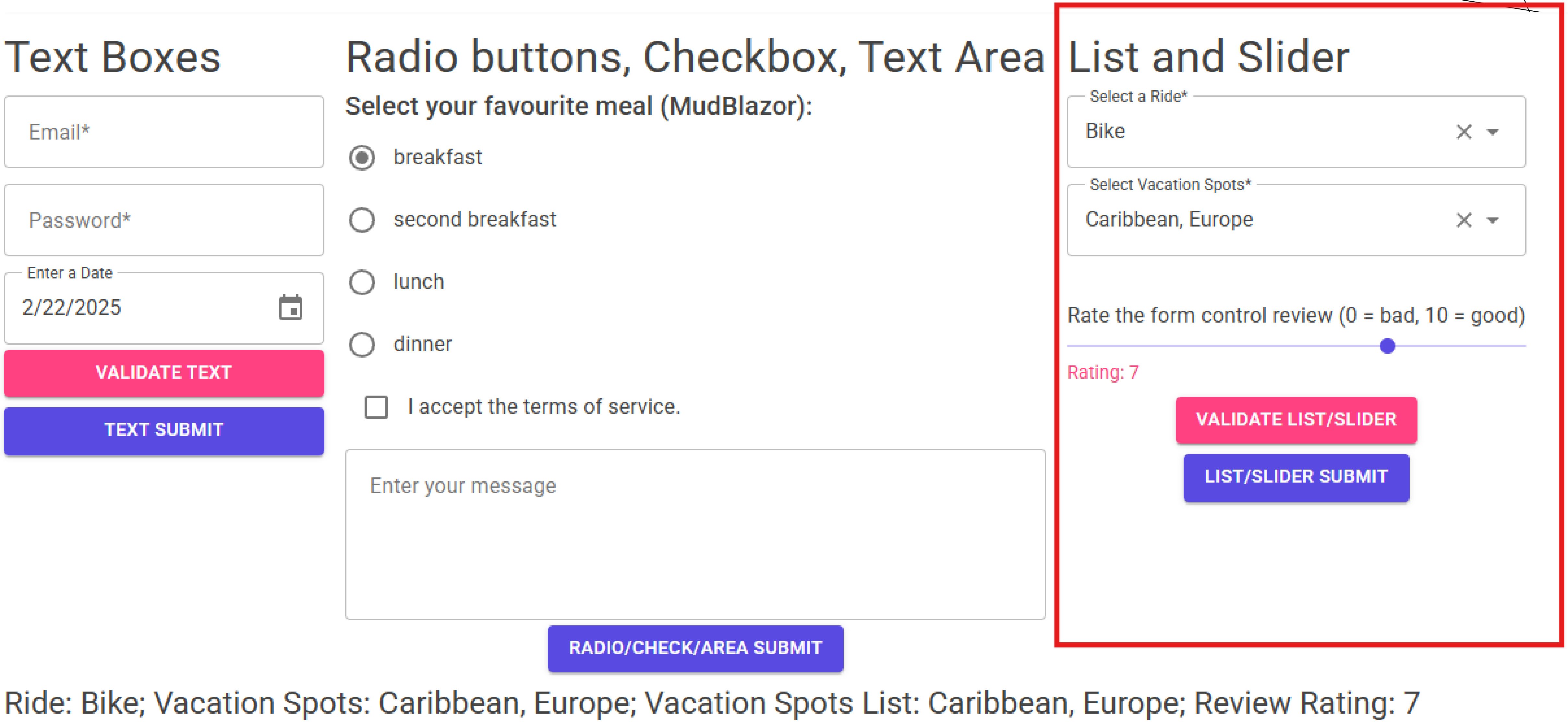
- Make sure we use a ternary operator to check myRide has a value
- Use .Value to get the value from the now nullable myRide
- Use the string.Join() method to display the Collection as a comma separated string of values for Vacation spots.

Basics.razor.cs



```
/// <summary>
/// Method is called when the user submits the lists and slider inputs to update the resulting feedback.
/// </summary>
1 reference
private void ListSliderSubmit()
{
    //Generate the feedback string incorporating the selected values
    feedback = $"Ride: {(myRide.HasValue ? rides[myRide.Value] : "No ride selected")}; Vacation Spots:
      {vacationSpot}; Vacation Spots List: {string.Join(", ", selectedVacationSpots)}; Review Rating:
      {reviewRating}";
}
```

LIST AND SLIDERS– RESULTS



Text Boxes

Email*

Password*

Enter a Date

2/22/2025

Radio buttons, Checkbox, Text Area

Select your favourite meal (MudBlazor):

[✓ breakfast]

second breakfast

lunch

dinner

☐ I accept the terms of service.

Enter your message

List and Slider

Select a Ride*

Bike

Select Vacation Spots*

Caribbean, Europe

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

Rating: 7

Ride: Bike; Vacation Spots: Caribbean, Europe; Vacation Spots List: Caribbean, Europe; Review Rating: 7

IMPROVING WITH A MUDGRID

To fix the spacing and take advantage of MudBlazor’s MudGrids we can wrap all the forms inside a MudGrid and change the <div> tags to <MudItem> tags

Note: MudItem uses spacing that adds up to 12 to create columns. To make 2 columns make both MudItems use the number 6, to make 3 columns make both MudItems use the number 4

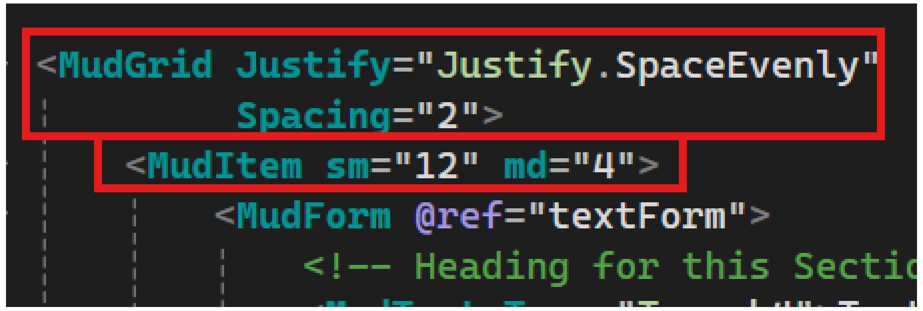
We will also use the <MudGrid> to add spacing and determine the layout.

1. Replace the <Section> with a <MudGrid>
2. Replace each <div> around the <MudForm> tags with <MudItem>

Note: MudItem takes into consideration screen sizes so we can set it to 12 when the screen is small (sm) or 4 when it’s medium (md) or larger.

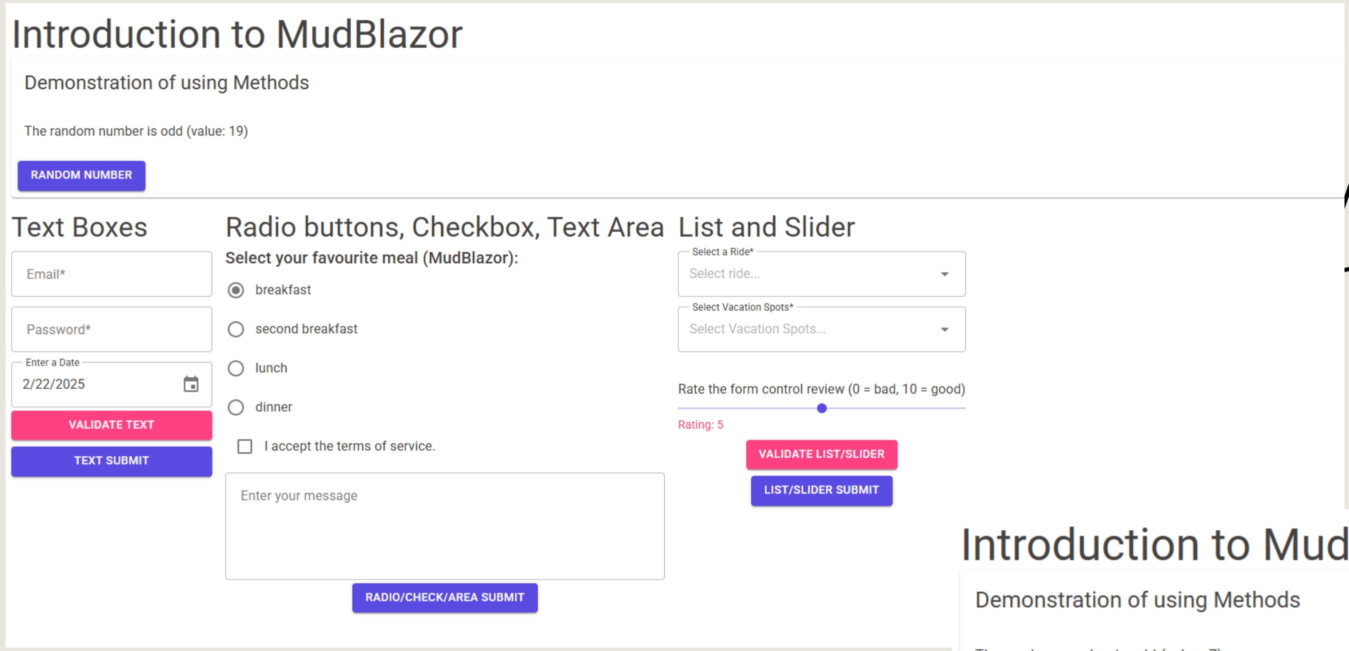
Reference the screen size breakpoints here: https://mudblazor.com/features/breakpoints#breakpoints

Basics.razor



```
<MudGrid Justify="Justify.SpaceEvenly"
         Spacing="2">
    <MudItem sm="12" md="4">
        <MudForm @ref="textForm">
            <!-- Heading for this Sectio
```

IMPROVING WITH A MUDGRID - RESULTS



Introduction to MudBlazor

Demonstration of using Methods

The random number is odd (value: 19)

Text Boxes

2/22/2025

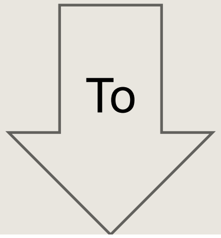
Radio buttons, Checkbox, Text Area

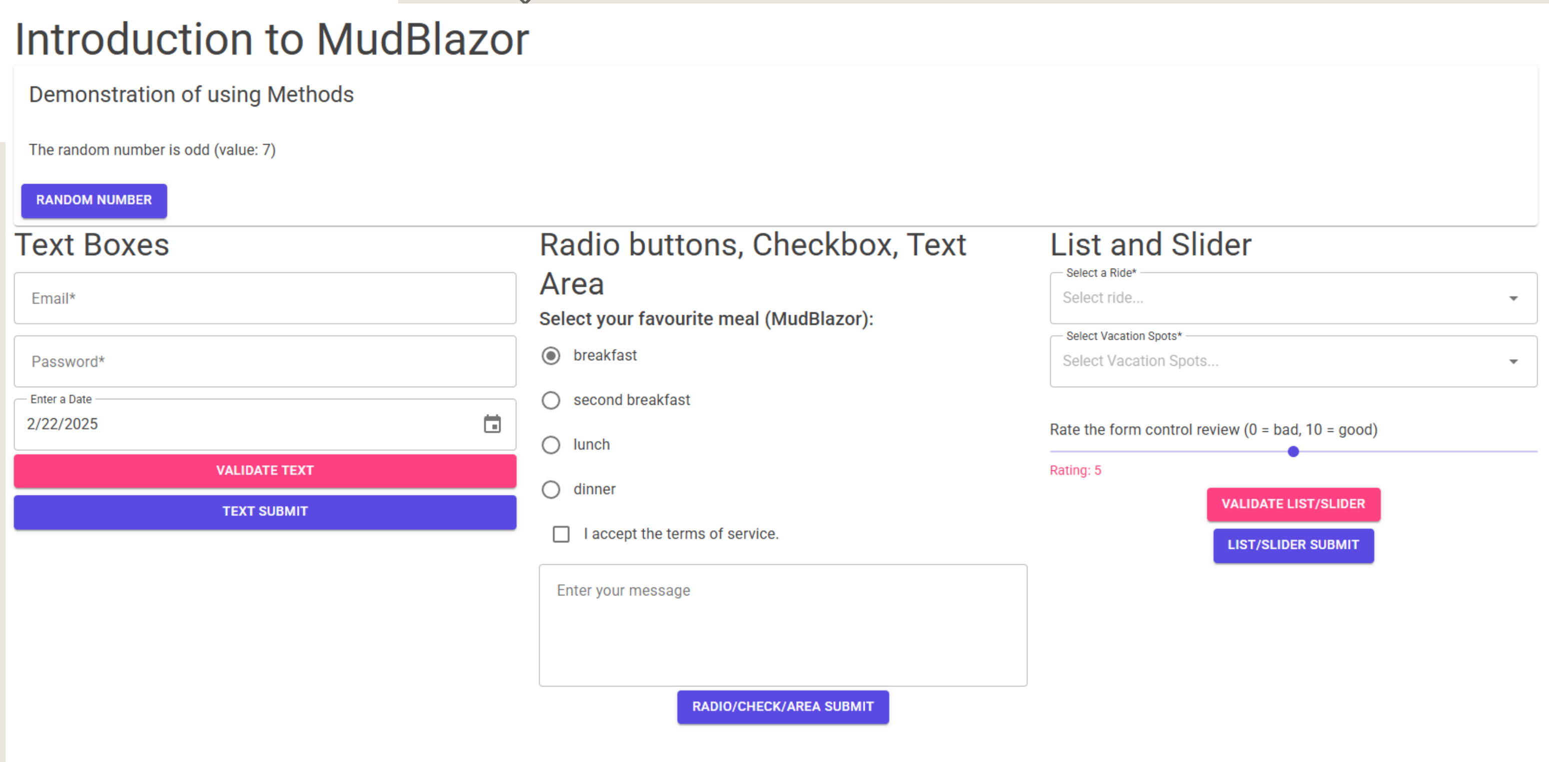
Select your favourite meal (MudBlazor):

List and Slider

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

Rating: 5





Introduction to MudBlazor

Demonstration of using Methods

The random number is odd (value: 7)

Text Boxes

2/22/2025

Radio buttons, Checkbox, Text Area

Select your favourite meal (MudBlazor):

List and Slider

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

Rating: 5

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

Copyright ©2023 Northern Alberta Institute of Technology. All rights reserved