



## About Michael Washington



Microsoft  
Most Valuable  
Professional  
Award



## Blazor Books By Michael Washington

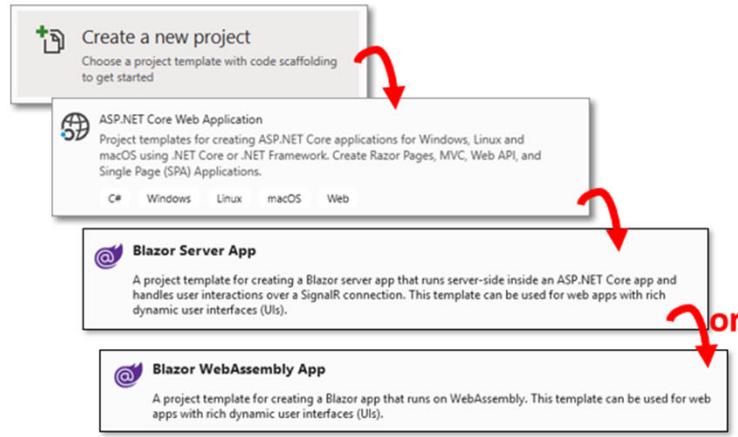


## Agenda

- Blazor basics
- Creating A Step-By-Step End-To-End Database Server-Side Blazor Application
- Creating A Step-By-Step End-To-End Database Client-Side (WebAssembly) Blazor Application
- Creating A Blazor Server Azure B2C App



## What is Blazor?



The client-side:

- Has n
- Has a
- Fully l
- Offload
- Suppo

There are d

- Restri
- Requi
- Has a
- Has le
- supp

# WebAssembly

Web Browser

standard

ion

30<sup>th</sup> ANNIVERSARY  
VS Live!  
1993 - 2023

The server-side hosting model offers several benefits:

## Server Side Blazor

Web Browser

Server

- Requires an ASP.NET Core server to serve the app. Deployment without a server (for example, from a CDN) isn't possible.

©Microsoft Corporation

30<sup>th</sup> ANNIVERSARY  
VS Live!  
1993 - 2023

## Sample Applications



## Blazing Pizza

A screenshot of a web-based pizza ordering application. At the top, there's a navigation bar with 'BLAZING PIZZA', 'GET PIZZA', 'MY ORDERS', and 'SIGN IN'. Below the navigation, there are several pizza options: 'MARGHERITA', 'BASIC CHEESE PIZZA', 'THE BACONATORIZOR' (selected), 'BUFFALO CHICKEN', and 'VEGGIE DELIGHT'. The 'THE BACONATORIZOR' section shows a large image of a bacon-topped pizza, a size selection slider set to '12" (£81.49)', and an 'Extra Toppings' dropdown menu with 'Lobster on top - (£64.50)' selected. Buttons for 'Buffalo chicken £5.00 x' and 'Lobster on top £64.50 x' are shown. The total price is listed as 'Price: £ 81.49'. On the right side, a 'YOUR ORDER' summary shows a '12" Margherita' with 'Blue cheese', 'Buffalo chicken', and 'Pepperoni', totaling '£ 18.49'. A green 'Order &gt;' button is at the bottom right of the order summary. At the very bottom, a footer bar shows 'Total: £18.49' and another 'Order &gt;' button.



## Visual Studio Live! Las Vegas 2023

# Flight Finder

The screenshot shows a Blazor application titled "Flight Finder". The main area displays flight search results from HND to SEA. Two flights are listed:

- Delta First: Departure at 3:00 AM on Sat Jan 18 (HND) and arrival at 2:25 PM on Sat Jan 18 (SEA). Total duration: 3 hours.
- JetBlue First: Departure at 3:20 AM on Sat Jan 25 (SEA) and arrival at 8:40 PM on Sat Jan 25 (HND). Total duration: 2 hours.

The total cost for the Delta flight is \$1,365. An "Add" button is present next to the price. A sidebar titled "Shortlist (2)" lists two additional flight options:

- LHR - SEA: Sat Jan 18 Economy flight from LHR to SEA at 6:05 PM for \$841.
- HND - SEA: Sat Jan 18 First class flight from HND to SEA at 3:00 AM for \$841.

A "Quickest" dropdown menu is visible on the right. The URL in the browser bar is https://localhost:44387.

## Features

This section is a placeholder for listing features, indicated by a large blue rectangular box containing the word "Features".

30<sup>th</sup> ANNIVERSARY  
VS Live!  
1993 – 2023

# Components

The screenshot shows the Visual Studio interface with the following components:

- Solution Explorer:** Shows a project named "Blazor" with files like Counter.razor, FetchData.razor, and Index.razor.
- Counter.razor:** UI Markup (HTML-like code) and Processing Logic (C# code). The UI includes an H1 title, a paragraph showing the current count, and a button to increment it. The logic defines a currentCount variable and an IncrementCount method.
- Code Editor:** Shows the C# code for the Counter component.
- Components:** A callout points to the Counter.razor file in the Solution Explorer.
- VS Live! 30th Anniversary Logo:** A logo in the bottom right corner.

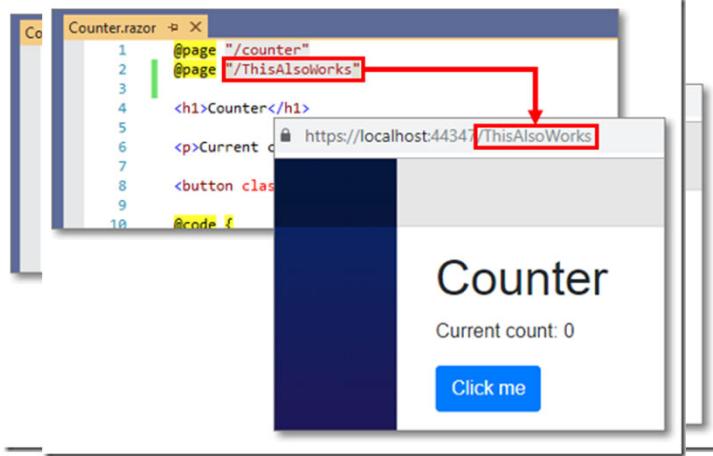
# Data Binding

## Two way binding

The diagram illustrates two-way data binding between UI elements and code. It shows:

- UI Elements:** An input range slider and a paragraph displaying the slider value.
- Code:** C# code defining a SliderValue variable and its initial value.
- Diagram Flow:** Blue arrows show the flow from the UI input to the code variable, and red arrows show the flow back from the code variable to the UI output.
- Callout:** A box labeled "Two Way Binding" contains a slider with the value "39".
- VS Live! 30th Anniversary Logo:** A logo in the bottom right corner.

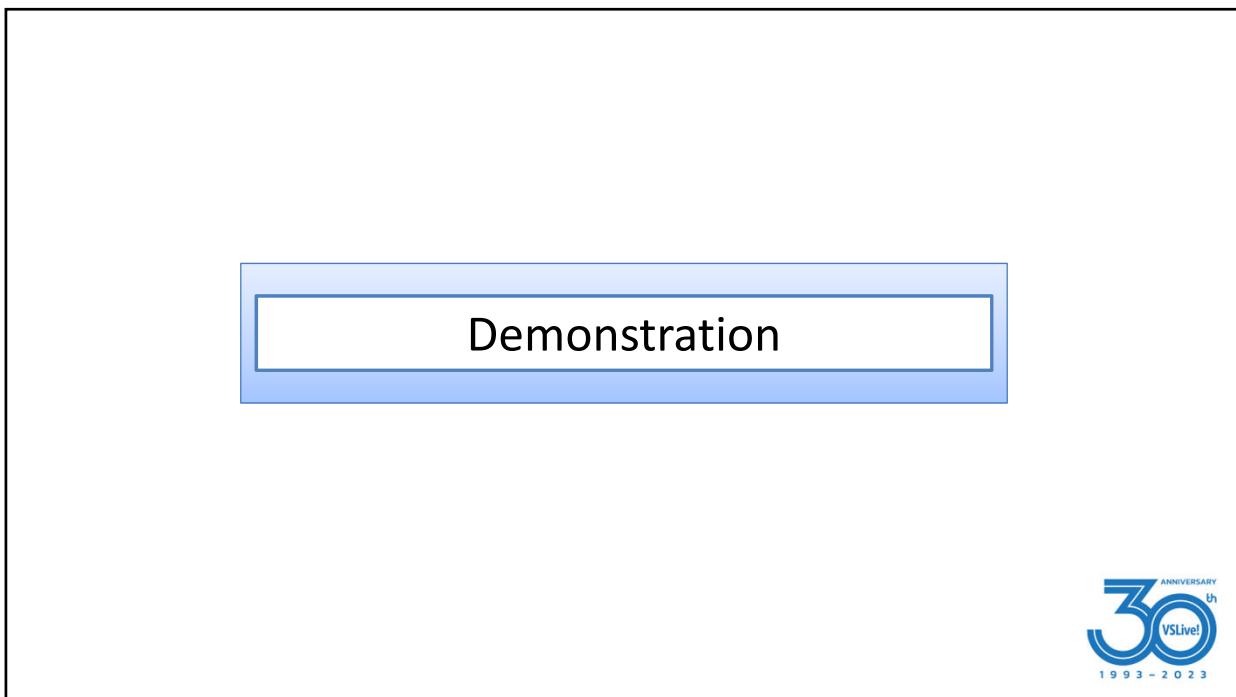
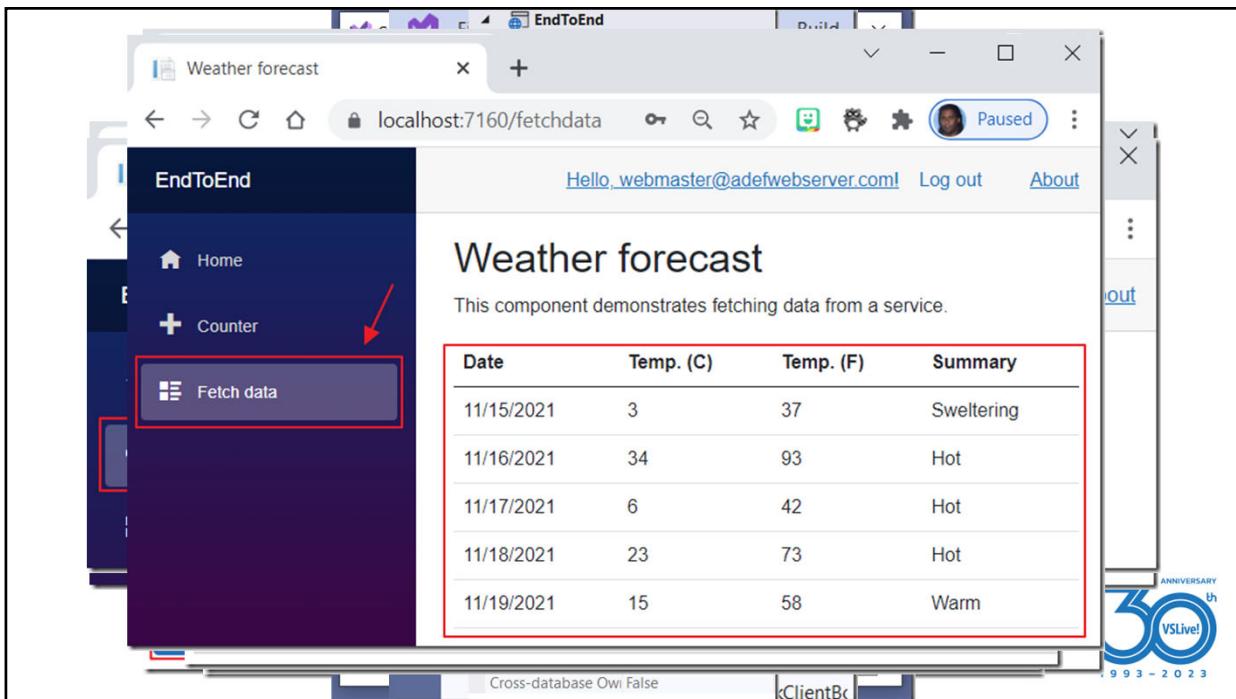
# Routing

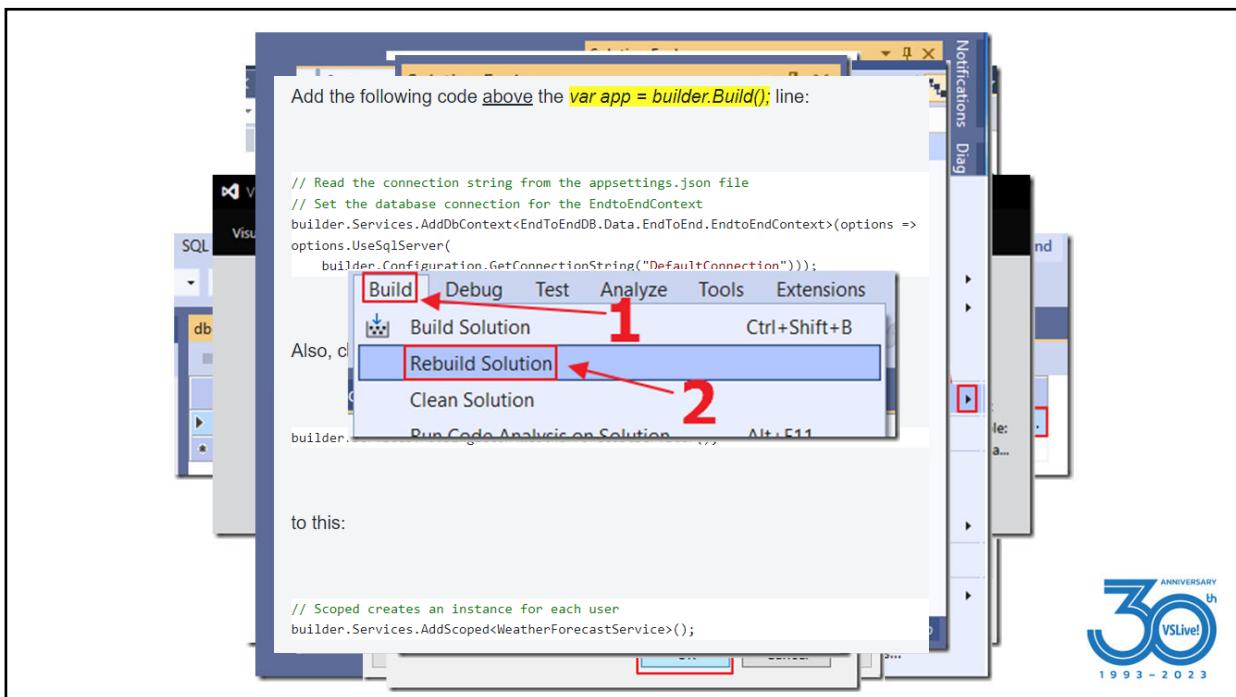
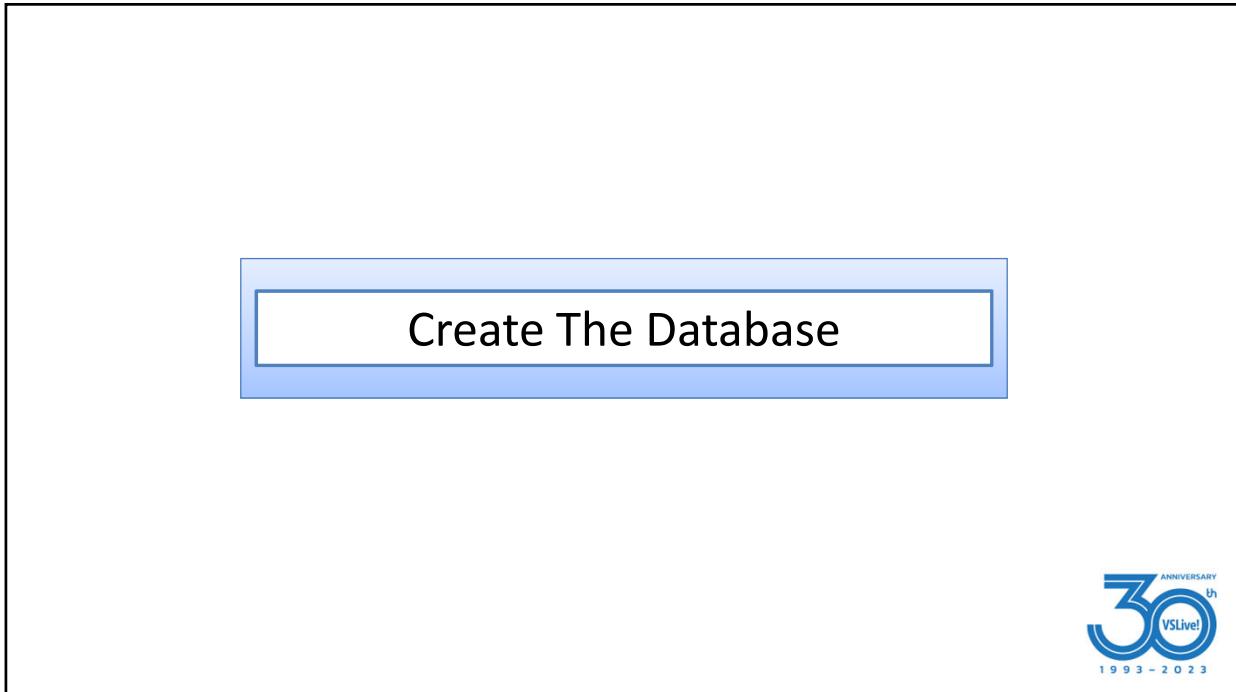


An End-To-End Sample  
Server Side Blazor



## Visual Studio Live! Las Vegas 2023





Demonstration



Read From The Database

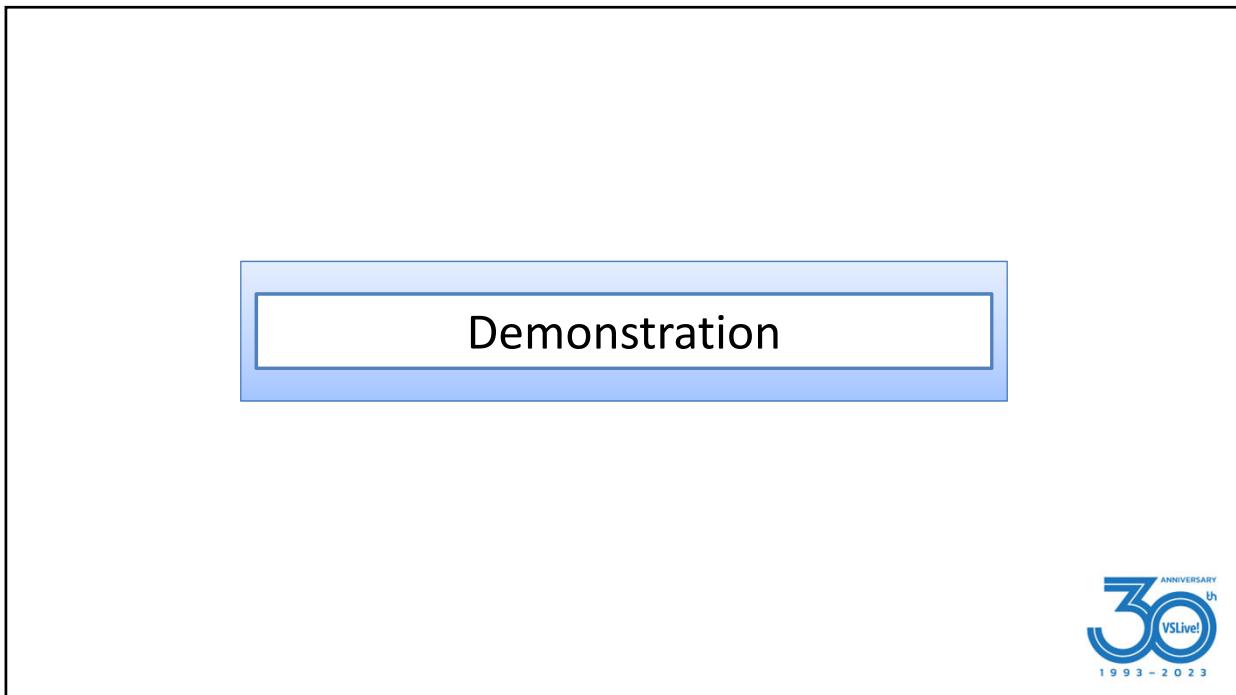


Visual Studio Live! Las Vegas 2023

The screenshot shows a .NET Core application interface. At the top, there is a navigation bar with 'EndToEnd' on the left and 'Register' and 'Login' on the right. Below this, a modal window titled 'Log in' is displayed. The modal has a dark header with 'EndToEnd' and a light body containing a form. The form includes fields for 'Email' (Hello, webmaster@adefwebserver.com!), 'Password', and 'Remember me?' (unchecked). A large blue 'Log in' button is at the bottom, highlighted with a red box and a red arrow pointing to it from below. To the right of the modal, the main content area shows a 'Weather forecast' section with a table:

| Date      | Temp. (C) | Temp. (F) | Summary |
|-----------|-----------|-----------|---------|
| 6/27/2019 | 37        | 99        | Hot!    |

At the bottom of the page, there is a code editor window showing C# code for an 'EndToEndController' class. The code includes annotations like `[@Page("Fetchdata")]`, `[@Using EndToEndData]`, and `[@Using EndToEndDb.Data.EndToEnd]`. It also contains logic for authentication and authorization using `AuthenticationStateProvider`.



## Inserting Data Into The Database



```
@if (ShowPopup async Task SaveForecast())
{
    {
        < // Close the Popup
        < ShowPopup = false;
        // Get the current user
        var user = (await authenticationStateTask).User;
        // A new forecast will have the Id set to 0
        if (objWeatherForecast.Id == 0)
        {
            // Create new forecast
            WeatherForecast objNewWeatherForecast = new WeatherForecast();
            objNewWeatherForecast.Date = System.DateTime.Now;
            objNewWeatherForecast.Summary = objWeatherForecast.Summary;
            objNewWeatherForecast.TemperatureC =
                Convert.ToInt32(objWeatherForecast.TemperatureC);
            objNewWeatherForecast.TemperatureF =
                Convert.ToInt32(objWeatherForecast.TemperatureF);
            objNewWeatherForecast.UserName = user.Identity.Name;
            // Save the result
            var result =
                @Service.CreateForecastAsync(objNewWeatherForecast);
        }
        else
        {
            // This is an update
        }
        // Get the forecasts for the current user
        forecasts =
            await @Service.GetForecastAsync(user.Identity.Name);
    }
}
```



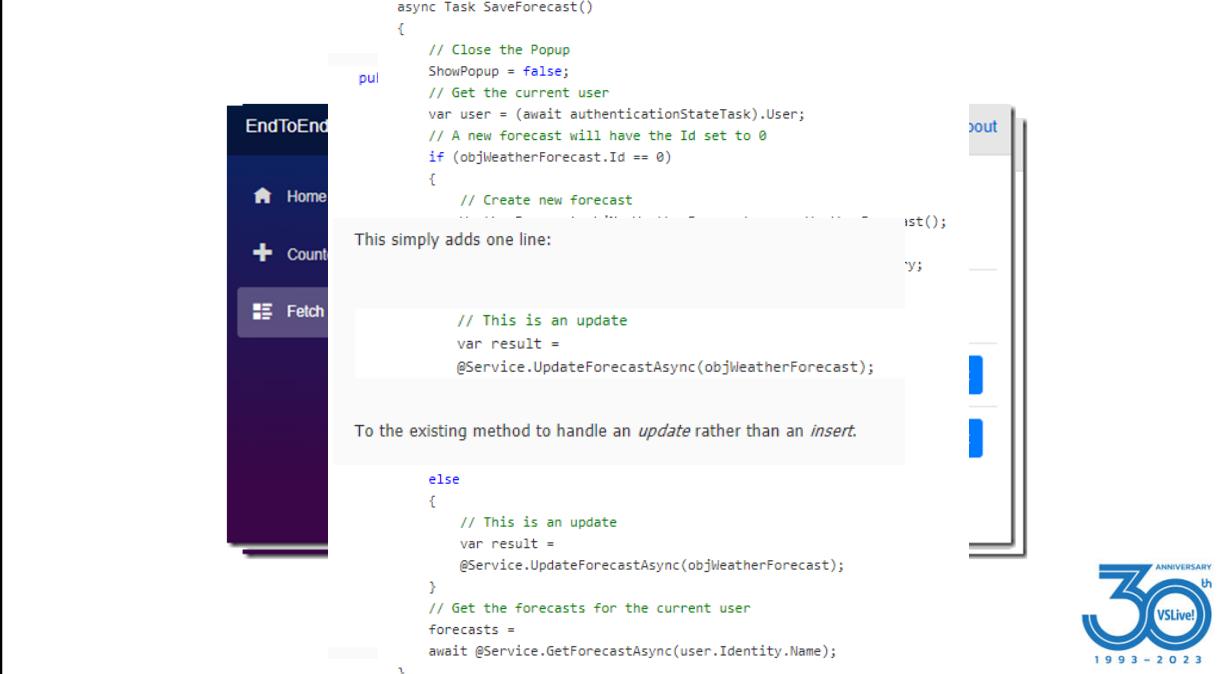
Demonstration



Updating The Data



# Visual Studio Live! Las Vegas 2023



The screenshot shows a Blazor application interface. On the left is a sidebar with navigation links: 'Home', 'Count', and 'Fetch'. The 'Fetch' link is highlighted. The main content area displays a snippet of C# code:

```
async Task SaveForecast()
{
    // Close the Popup
    ShowPopup = false;
    // Get the current user
    var user = (await authenticationStateTask).User;
    // A new forecast will have the Id set to 0
    if (objWeatherForecast.Id == 0)
    {
        // Create new forecast
        forecasts =
            await @Service.GetForecastAsync(user.Identity.Name);
    }
    else
    {
        // This is an update
        var result =
            @Service.UpdateForecastAsync(objWeatherForecast);
    }
    // Get the forecasts for the current user
    forecasts =
        await @Service.GetForecastAsync(user.Identity.Name);
}
```

A callout box points to the line 'This simply adds one line:' with the text 'To the existing method to handle an *update* rather than an *insert*.'. In the bottom right corner of the slide, there is a logo for 'VSLive! 30th Anniversary 1993 - 2023'.



A large blue rectangular box contains the word 'Demonstration' in white text, centered within a white rectangular frame.

In the bottom right corner of the slide, there is a logo for 'VSLive! 30th Anniversary 1993 - 2023'.

# Visual Studio Live! Las Vegas 2023

## Deleting The Data

A screenshot of the Visual Studio IDE. The title bar says "Solution Explorer". The main window shows C# code in a file named "Startup.cs". The code is as follows:

```
public Task<bool>
DeleteForecast()
{
    // Close the Popup
    ShowPopup = false;
    // Get the current user
    var user = (await authenticationStateTask).User;
    // Delete the forecast
    var result = @Service.DeleteForecastAsync(objWeatherForecast);    );
    // Get the forecasts for the current user
    forecasts =
        await @Service.GetForecastAsync(user.Identity.Name);
}

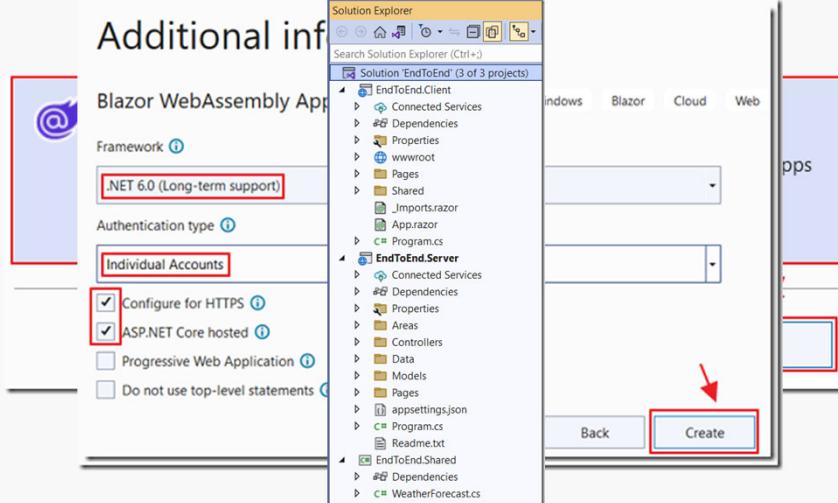
return Task.FromResult(false);
}
return Task.FromResult(true);
}
```

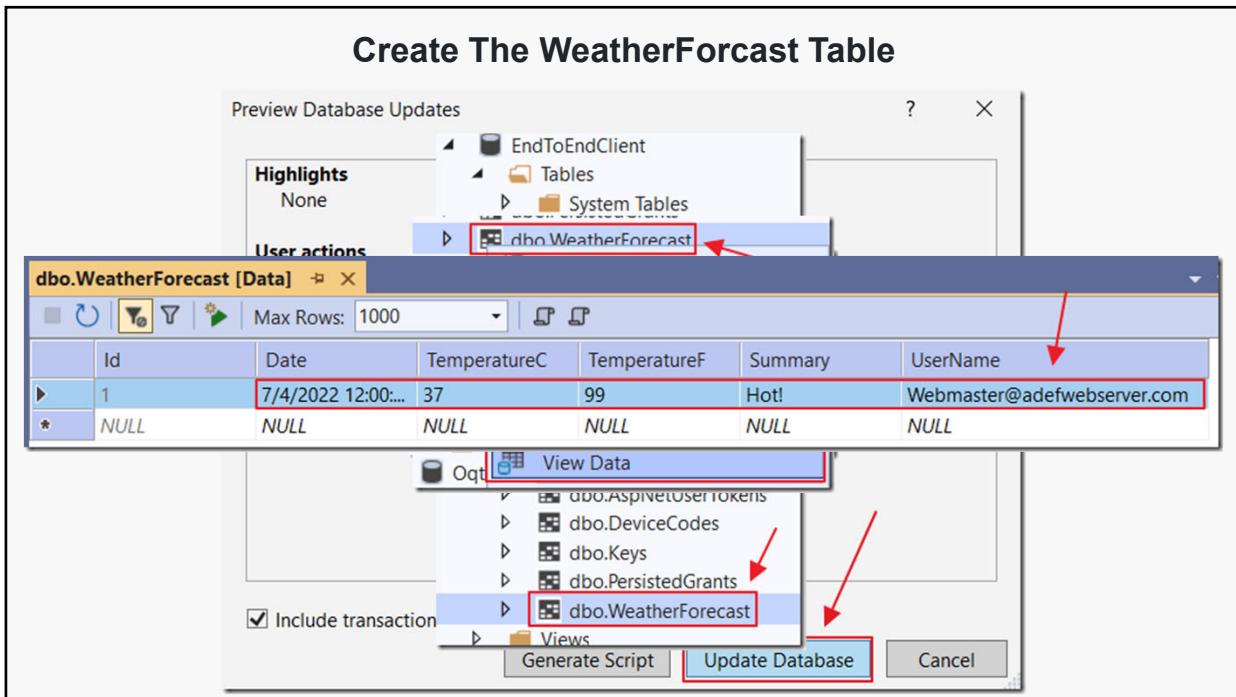
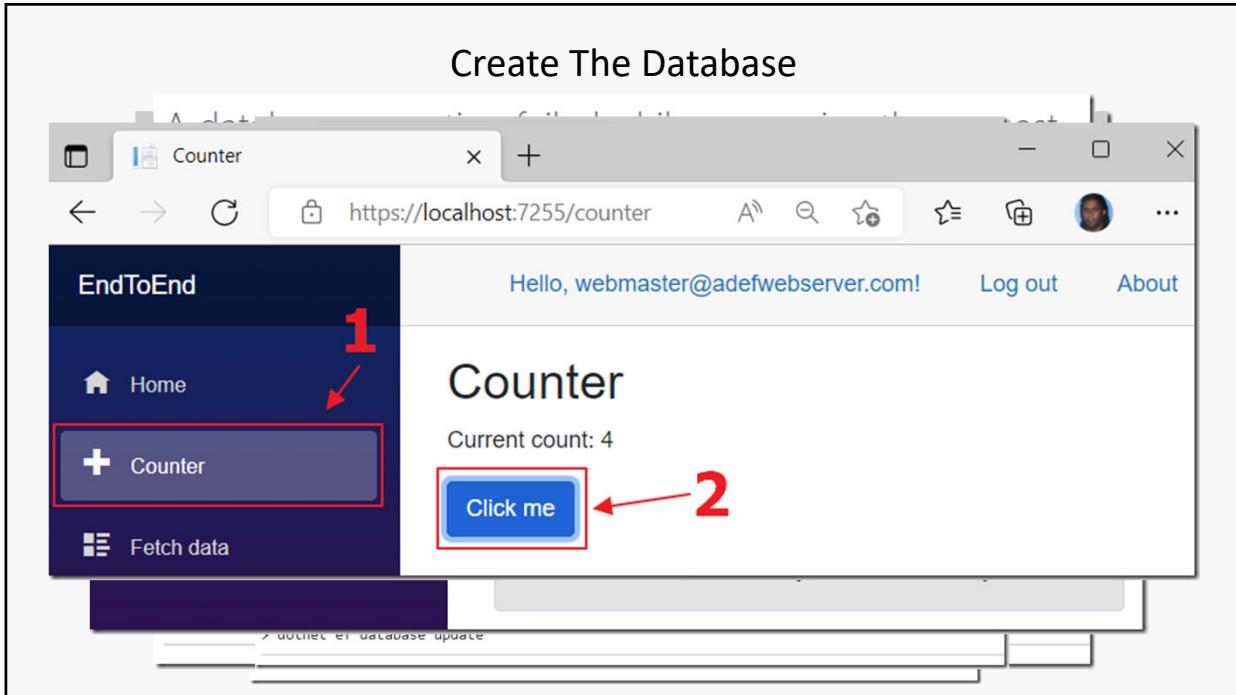
The status bar at the bottom shows "C# Startup.cs".

## An End-To-End Sample Client Side (WebAssembly) Blazor



### Create The Project





## Create The Data Context

```
// Read the connection string from the appsettings.json file
// Set the database connection for the EndToEndContext
builder.Services.AddDbContext<EndToEndDB.Data.EndToEndContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("DefaultConnection")));

```

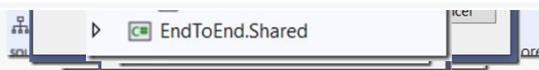
To allow the user name to be readable in the Controller methods (to be created later), change the following line:

```
builder.Services.AddIdentityServer()
    .AddApiAuthorization< ApplicationUser, ApplicationDbContext >();
```

To:

```
// This allows the username (email address) of the currently
// logged in user to be read in the server side controller
builder.Services.AddIdentityServer()
    .AddApiAuthorization< ApplicationUser, ApplicationDbContext >(options => {
        options.IdentityResources["openid"].UserClaims.Add("name");
        options.ApiResources.Single().UserClaims.Add("name");
    });

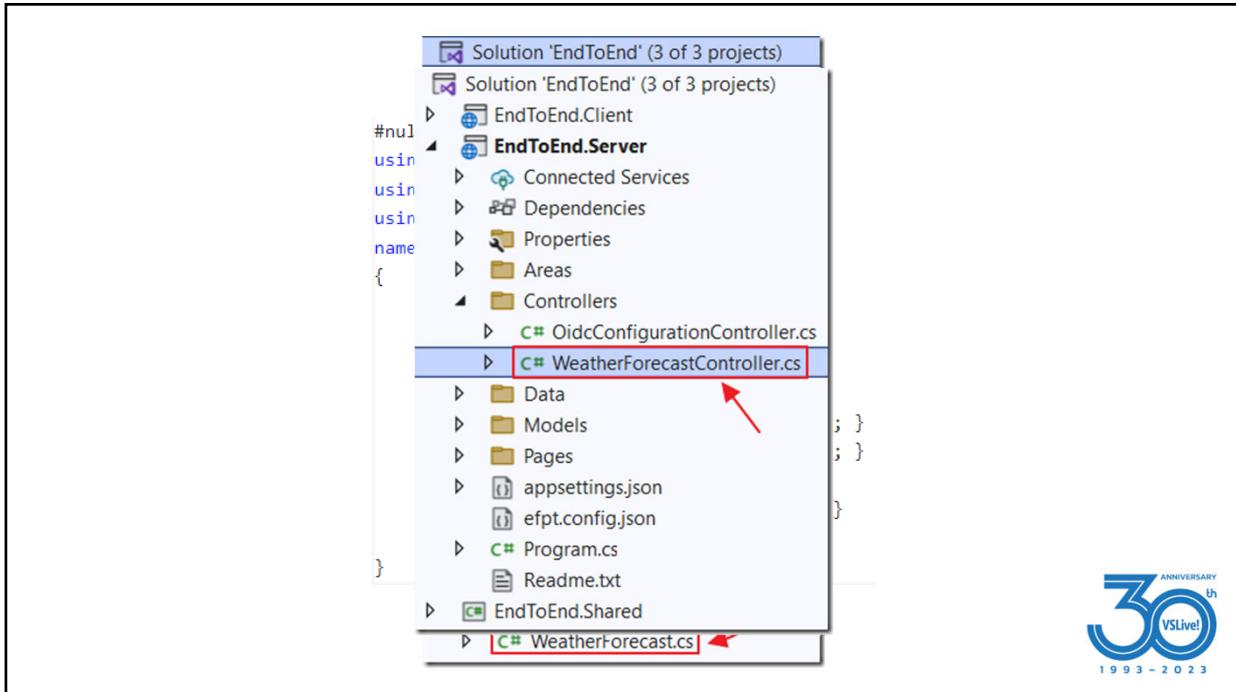
```



## Client Side (WebAssembly) Blazor CRUD Methods



Visual Studio Live! Las Vegas 2023



# Read From The Database



# Visual Studio Live! Las Vegas 2023

```
[HttpGet]
[Route("api/WeatherForecast/GetAsync")]
public async Task<List<WeatherForecastDTO>>
    GetAsync()
{
    // Get the current user
    string strCurrentUser = this.User.Claims
        .Where(x => x.Type == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name")
        .FirstOrDefault().Value;

    // Get Weather Forecasts
    var result = await _context.WeatherForecast
        .Where(x => x.UserName == strCurrentuser)
        // Use AsNoTracking to disable EF change tracking
        // UseToListAsync to avoid blocking a thread
        .AsNoTracking().ToListAsync();

    // Collection to return
    List<WeatherForecastDTO> ColWeatherForecast =
        new List<WeatherForecastDTO>();
    // Loop through the results
    foreach (var item in result)
    {
        // Create a new WeatherForecast instance
        WeatherForecastDTO objWeatherForecastDTO =
            new WeatherForecastDTO();
        // Set the values for the WeatherForecast instance
        objWeatherForecastDTO.Id =
            item.Id;
        objWeatherForecastDTO.Date =
            item.Date;
        objWeatherForecastDTO.UserName =
            item.UserName;
        objWeatherForecastDTO.TemperatureF =
            item.TemperatureF;
        objWeatherForecastDTO.TemperatureC =
            item.TemperatureC;
        objWeatherForecastDTO.Summary =
            item.Summary;
        // Add the WeatherForecast instance to the collection
        ColWeatherForecast.Add(objWeatherForecastDTO);
    }
    // Return the final collection
    return ColWeatherForecast;
}
```



## Inserting Data



# Visual Studio Live! Las Vegas 2023

```
[HttpPost]
[Route("api/WeatherForecast/Post")]
public void
Post([FromBody] WeatherForecastDTO paramWeatherForecast)
{
    // Create a new WeatherForecast instance
    WeatherForecast objWeatherForecast =
        new WeatherForecast();

    // Get the current user
    string strCurrentUser = this.User.Claims
        .Where(x => x.Type ==
            "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name")
        .FirstOrDefault().Value;

    // Set the values for the WeatherForecast instance
    objWeatherForecast.Id =
        paramWeatherForecast.Id;
    objWeatherForecast.Date =
        paramWeatherForecast.Date;
    objWeatherForecast.UserName =
        strCurrentUser;
    objWeatherForecast.TemperatureF =
        paramWeatherForecast.TemperatureF;
    objWeatherForecast.TemperatureC =
        paramWeatherForecast.TemperatureC;
    objWeatherForecast.Summary =
        paramWeatherForecast.Summary;
    // Save to the database
    _context.WeatherForecast.Add(objWeatherForecast);
    _context.SaveChanges();
}
```



Updating Data



# Visual Studio Live! Las Vegas 2023

```
[HttpPost]
[Route("api/WeatherForecast/Put")]
public void
    Put([FromBody] WeatherForecastDTO objWeatherForecast)
{
    // Get the current user
    string strCurrentUser = this.User.Claims
        .Where(x => x.Type ==
            "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name")
        .FirstOrDefault().Value;

    // Find the existing Weather Forecast record
    // using the id
    var ExistingWeatherForecast =
        _context.WeatherForecast
        .Where(x => x.Id == objWeatherForecast.Id)
        .Where(x => x.UserName == strCurrentUser)
        .FirstOrDefault();
    if (ExistingWeatherForecast != null)
    {
        // Update the values
        ExistingWeatherForecast.Date =
            objWeatherForecast.Date;
        ExistingWeatherForecast.Summary =
            objWeatherForecast.Summary;
        ExistingWeatherForecast.TemperatureC =
            objWeatherForecast.TemperatureC;
        ExistingWeatherForecast.TemperatureF =
            objWeatherForecast.TemperatureF;
        // Save to the database
        _context.SaveChanges();
    }
}
```



## Deleting Data



# Visual Studio Live! Las Vegas 2023

```
[HttpDelete]
[Route("api/WeatherForecast/Delete/{id}")]
public void
    Delete(int id)
{
    // Get the current user
    string strCurrentUser = this.User.Claims
        .Where(x => x.Type ==
            "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name")
        .FirstOrDefault().Value;

    // Find the existing Weather Forecast record
    // using the id
    var ExistingWeatherForecast =
        _context.WeatherForecast
        .Where(x => x.Id == id)
        .Where(x => x.UserName == strCurrentUser)
        .FirstOrDefault();
    if (ExistingWeatherForecast != null)
    {
        // Remove from the database
        _context.WeatherForecast
            .Remove(ExistingWeatherForecast);
        _context.SaveChanges();
    }
}
```



## Client Code - Markup



# Visual Studio Live! Las Vegas 2023

```
<!-- gif (ShowPopup) -->
<ta {
    <!-- This is the popup to create or edit a forecast -->
    <div class="modal" tabIndex="-1" style="display:block"
        role="dialog">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h3 class="modal-title">Edit Forecast</h3>
                    <!-- Button to close the popup -->
                    <button type="button" class="close"
                        @onclick="ClosePopup">
                        <span aria-hidden="true">X</span>
                    </button>
                </div>
                <!-- Edit form for the current forecast -->
                <div class="modal-body">
                    <input class="form-control" type="text"
                        placeholder="Celsius forecast"
                        @bind="objWeatherForecastDTO
                            .TemperatureC" />
                    <input class="form-control" type="text"
                        placeholder="Fahrenheit forecast"
                        @bind="objWeatherForecastDTO
                            .TemperatureF" />
                    <input class="form-control" type="text"
                        placeholder="Summary"
                        @bind="objWeatherForecastDTO.Summary" />
                    <br />
                    <!-- Button to save the forecast -->
                    <button class="btn btn-primary"
                        @onclick="SaveForecast">
                        Save
                    </button>
                    <!-- Only show button if not a new record -->
                    @if ((objWeatherForecastDTO.Id > 0)
                    {
                        <!-- Button to delete the forecast -->
                        <button class="btn btn-primary"
                            @onclick="DeleteForecast">
                            Delete
                        </button>
                    }
                </div>
            </div>
        </div>
    </div>
</p>
</p>
```



## Client Code – C# Code



Visual Studio Live! Las Vegas 2023

```
@code async Task SaveForecast()
{
    #
    // Close the Popup
    / ShowPopup = false;
    / // Get the current user
    / var user = (await authenticationStateTask).User;
    / // A new forecast will have the Id set to 0
    [   if (objWeatherForecastDTO.Id == 0)
    n
}

async Task DeleteForecast()
{
    // Close the Popup
    ShowPopup = false;
    // Delete the forecast
    await Http
        .DeleteAsync("/api/WeatherForecast/Delete/" +
    Convert.ToInt32(objWeatherForecastDTO.Id));
    // Get the forecasts for the current user
    forecasts =
        await Http
            .GetFromJsonAsync<List<WeatherForecastDTO>>(
                "/api/WeatherForecast/GetAsync");
}

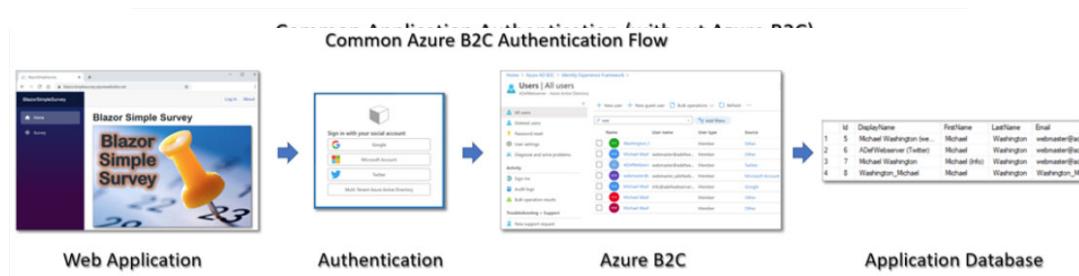
        objWeatherForecastDTO);
}
// Get the forecasts for the current user
forecasts =
    await Http
        .GetFromJsonAsync<List<WeatherForecastDTO>>(
            "/api/WeatherForecast/GetAsync");
}
}
```

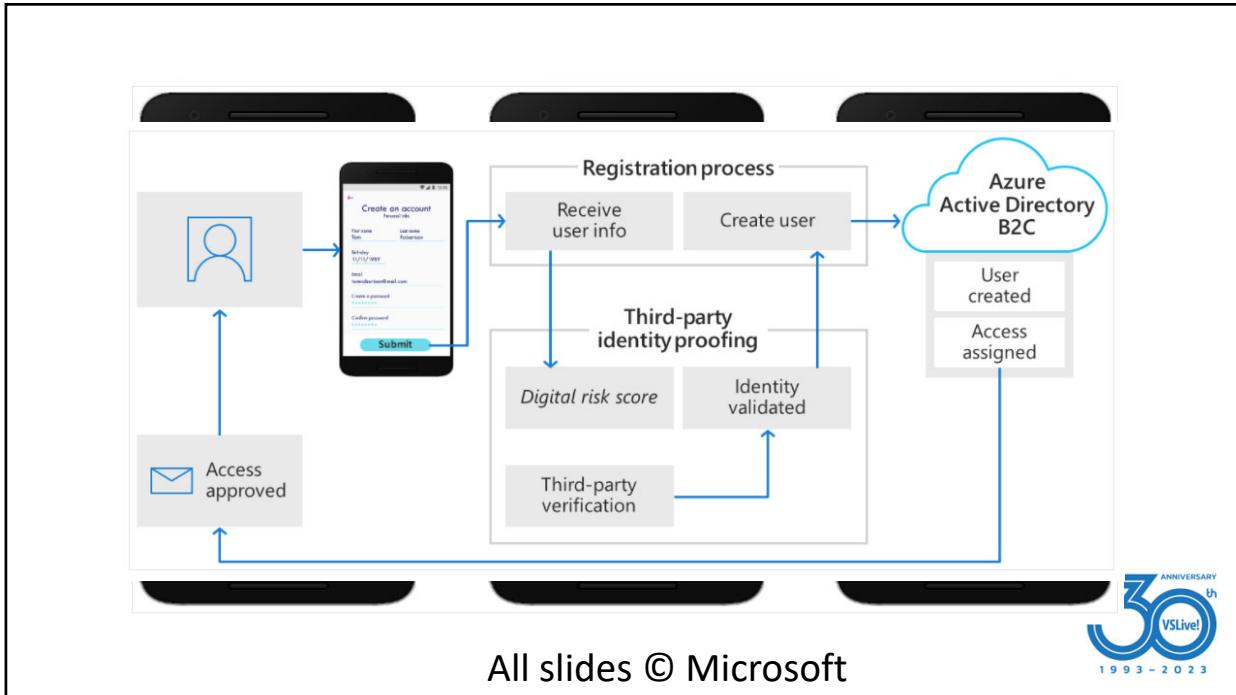


# Demonstration



## About Azure B2C





## Set-Up An Azure Active Directory B2C Tenant



## Visual Studio Live! Las Vegas 2023

The screenshot shows the Azure portal's tenant overview page. The tenant name is 'ADefWebserverB2C.onmicrosoft.com'. The 'Essentials' section displays various details: Resource group (Move) to 'AzureB2C', Location to 'United States', Subscription to 'Visual Studio Enterprise', and Pricing tier to 'PremiumP1'. The 'Azure AD B2C Tenant' section shows the tenant ID 'ADefWebserverB2C.onmicrosoft.com' and Monthly Active Users. On the left sidebar, under 'Settings', there is a 'Properties' section which includes the 'Azure AD B2C Settings' link, which is highlighted with a red arrow pointing to the 'Open B2C Tenant' button below it.



Register App in Azure AD B2C tenant



# Visual Studio Live! Las Vegas 2023

The screenshot shows the 'Register an application' page in the Azure AD B2C portal. Step 1 highlights the 'Name' field containing 'Blazor Azure B2C'. Step 2 highlights the 'Supported account types' section where the 'Accounts in any identity provider or organizational directory' option is selected. Step 3 highlights the 'Grant admin consent to openid and offline\_access permissions' checkbox. Step 4 highlights the 'Register' button at the bottom.



Configure Azure AD B2C  
Identity Provider & User Flow



# Visual Studio Live! Las Vegas 2023

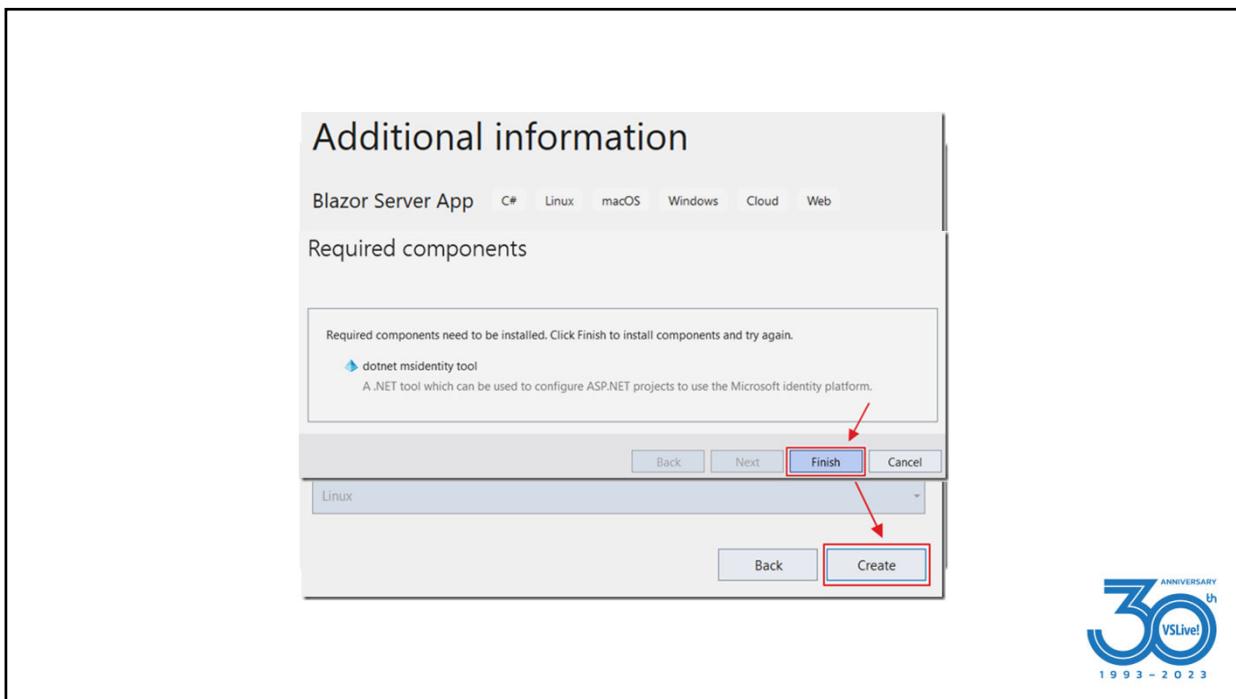
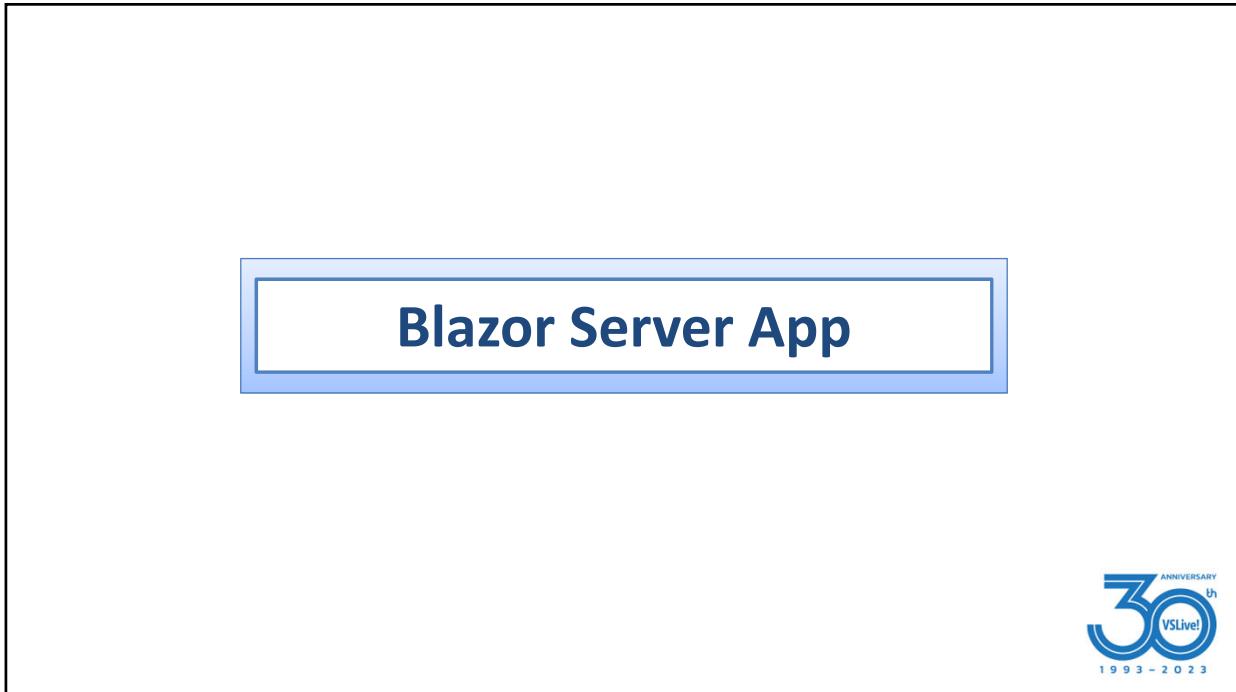
The screenshot shows the Azure AD B2C Identity providers configuration page. On the left, the 'Identity providers' section is highlighted with a red box. In the center, the 'Microsoft Account' provider is selected and highlighted with a red box. On the right, a modal window titled 'Configure social IDP' is open, showing fields for 'Name' (Microsoft Account), 'Client ID' (15b6cbeae92\_4838-b092-15b6cbeae92), and 'Client secret' (\*\*\*\*\*). A red box highlights the 'Save' button at the top right of the modal.



The screenshot shows the Azure AD B2C User flows creation page. The 'User flows' section in the sidebar is highlighted with a red box. In the main area, a user flow named 'B2C\_1\_BazorSignInSign... (Sign up and sign in (Recommended))' is listed. At the bottom, there is a 'Create' button with a red arrow pointing to it, and a 'Google' icon with a red arrow pointing to it.



## Visual Studio Live! Las Vegas 2023



Demonstration



Add the Redirect URI in the  
Azure App Registration

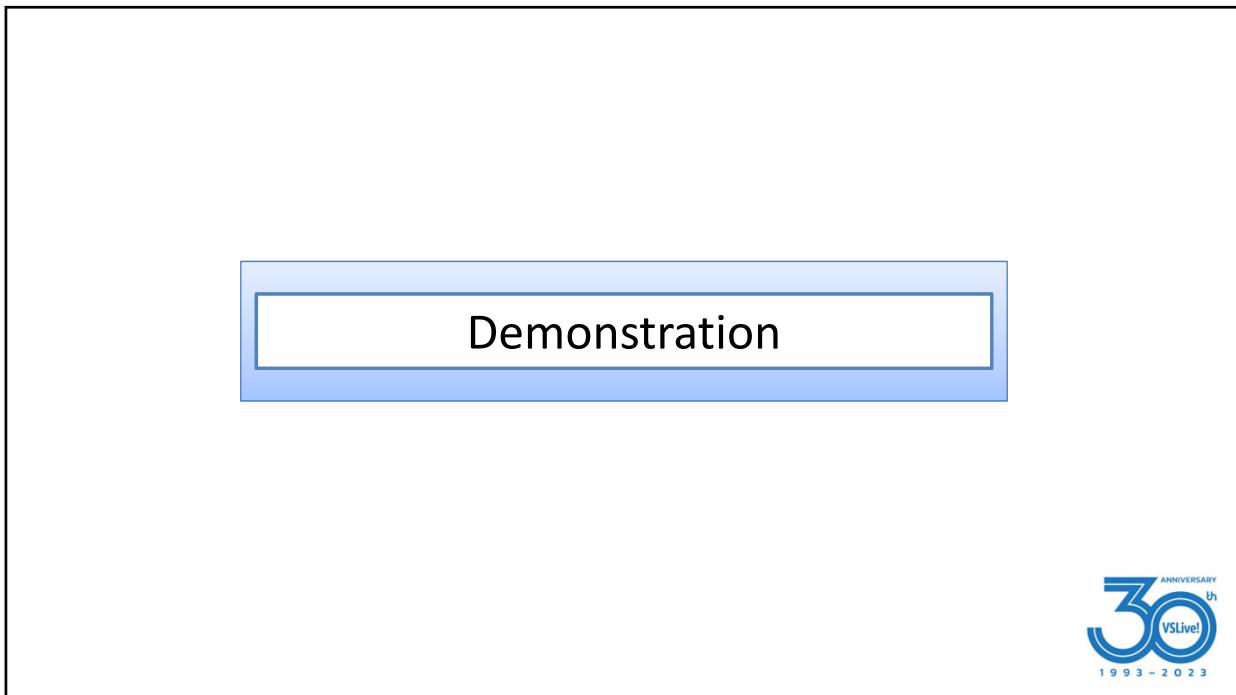


# Visual Studio Live! Las Vegas 2023

The screenshot shows the Azure AD B2C User flows interface. On the left, there is a navigation sidebar with various options like Home, App registrations, Applications (Legacy), Identity providers, API connectors, Company branding, User attributes, Users, Roles and administrators, and Policies. The 'User flows' option under Policies is highlighted with a red box and a red arrow labeled '1'. In the main content area, the title is '{Your Signup policy}' and the URL is 'Home > Azure AD B2C'. Below this, it says 'Azure AD B2C | User flows' and 'ADefWebserverB2C.onmicrosoft.com'. There is a search bar, a 'New user flow' button, and a 'Got feedback?' link. The 'Manage' section includes links for App registrations, Applications (Legacy), Identity providers, API connectors, Company branding, User attributes, Users, Roles and administrators, and Policies. Under Policies, 'User flows' is again highlighted with a red box and a red arrow labeled '1'. A form is open for creating a new user flow, with the 'Name' field containing 'B2C\_1\_BazorSignInSignUp'. A red arrow points from the 'Name' field to the 'User flows' link in the sidebar.

The screenshot shows a Blazor application's sign-in page. The header reads 'Sign-In To Azure B2C'. At the bottom right, there is a logo for the '30th VSLive! Anniversary' from 1993 to 2023.

# Visual Studio Live! Las Vegas 2023



Closing



All Resources  
Files / Tutorials



# Thank You!

