Walkthrough 21 - Blazor WebAssembly with EF Core

Setup

This lab will explore Blazor WebAssembly using EF Core.

1. Start Visual Studio.
2. Click Create a new project.
3. Set language to **C#** and project type to **Web**.
4. Select the Blazor WebAssembly App template, click Next.
5. Set Project name to **MovieTrackerBlazor**.
6. Set Location to a **folder of your choosing**.
7. Ensure Place solution and project in the same directory is not selected, click Next.
8. Set version to **.NET 5.0**, unselect Configure for HTTPS, select ASP.NET Core hosted, unselect Progressive Web Application, click Create.

WeatherForecast.cs

1. Delete MovieTrackerBlazor.Shared / WeatherForecast.cs.

WeatherForecastController.cs

1. Delete MovieTrackerBlazor.Server / Controllers / WeatherForecastController.cs.

SurveyPrompt.razor

1. Delete MovieTrackerBlazor.Client / Shared / SurveyPrompt.razor.

Counter.razor

1. Delete MovieTrackerBlazor.Client / Pages / Count.razor.

FetchData.razor

1. Delete MovieTrackerBlazor.Client / Pages / FetchData.razor.

Genre.cs

1. Create a **Genre** class in the MovieTrackerBlazor.Shared folder. Add the **using System.ComponentModel.DataAnnotations;** directive.
2.
```
public class Genre
{
    [Key]
    public int Id { get; set; }

    [StringLength(25)]
    public string GenreDescription { get; set; }
}
```

- **Save the file.**

Movie.cs

1. Create a **Movie** class in the MovieTrackerBlazor.Shared folder. Add the **using System.ComponentModel.DataAnnotations;** directive.
2.
```
public class Movie
{
    [Key]
    public int Id { get; set; }

    [Required, StringLength(100)]
    public string Title { get; set; }

    [DataType(DataType.Date)]
    public DateTime? DateSeen { get; set; }

    [ForeignKey("Genre")]
    public int? GenreId { get; set; }

    public Genre Genre { get; set; }

    [Range(1, 10)]
    public int? Rating { get; set; }
}
```

3. Save the file.

GenresController.cs

1. Right-click the Controllers folder and select Add / Controller..., select the API templates and select the API Controllers with actions, using Entity Framework. click Add.
2. Set Model class to **Genre (MovieTrackerBlazor.Shared)**.
3. For Data context class, click the **+** button. Accept the name **MovieTrackerBlazor.Server.Data.MovieTrackerBlazorServerContext**, click Add.
4. Accept the default Controller name, click Add.
5. Update the routing to remove api. Delete the GET with id, PUT, POST and DELETE endpoints. Also delete GenreExists.
6.

```
[Route("~~api~~/[controller]")]
[ApiController]
public class GenresController : ControllerBase
{
    private readonly MovieTrackerBlazorServerContext _context;

    public GenresController(MovieTrackerBlazorServerContext context)
    {
        _context = context;
    }

    // GET: api/Genres
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Genre>>> GetGenre()
    {
        return await _context.Genre.ToListAsync();
    }

    // GET: api/Genres/5
    [HttpGet("{id}")]
    public async Task<ActionResult<Genre>> GetGenre(int id)
    {
        var genre = await _context.Genre.FindAsync(id);

        if (genre == null)
        {
            return NotFound();
        }

        return genre;
    }

    // PUT: api/Genres/5
    // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPut("{id}")]
    public async Task<IActionResult> PutGenre(int id, Genre genre)
    {
        if (id != genre.Id)
        {
            return BadRequest();
        }

        _context.Entry(genre).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!GenreExists(id))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }

        return NoContent();
    }

    // POST: api/Genres
    // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
    [HttpPost]
    public async Task<ActionResult<Genre>> PostGenre(Genre genre)
    {
        _context.Genre.Add(genre);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetGenre", new { id = genre.Id }, genre);
    }

    // DELETE: api/Genres/5
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteGenre(int id)
    {
        var genre = await _context.Genre.FindAsync(id);
        if (genre == null)
        {
            return NotFound();
        }

        _context.Genre.Remove(genre);
        await _context.SaveChangesAsync();

        return NoContent();
    }
```

```
    private bool GenreExists(int id)
    {
        return _context.Genre.Any(e => e.Id == id);
    }
}
```

7. Save the file.

MoviesController.cs

1. Right-click the Controllers folder and select Add / Controller..., select the API templates and select the API Controllers with actions, using Entity Framework. click Add.
2. Set Model class to **Movie (MovieTrackerBlazor.Shared)**.
3. Set Data context class to **MovieTrackerBlazor.Server.Data.MovieTrackerBlazorServerContext**.
4. Accept the default Controller name, click Add.
5. Update the routing to remove api.
6.
```
[Route("api/[controller]")]
[ApiController]
public class MoviesController : ControllerBase
{
    private readonly MovieTrackerBlazorServerContext _context;
...
```

7. Update GetMovie to include Genre.
8.
```
[HttpGet]
public async Task<ActionResult<IEnumerable<Movie>>> GetMovie()
{
    return await _context.Movie.Include(m => m.Genre).ToListAsync();
}
```

9. Save the file.

appsettings.json

1. Update the connection string to use SQL Server Express and simplify the database name.
2.
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MovieTrackerBlazorServerContext": "Server=(localdb)localhost\\mssqllocaldbsqlexpress;Database=MovieTrackerBlazorServerContext-6c6dd608-9a64-44be-a883-43cb474b07a1movie_tracker_blazor;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

3. Save the file.

Database Migration

1. In the Package Manager Console, ensure MovieTrackerBlazor.Server is the default project. Issue the command **Add-Migration init**.
2. Followed by **Update-Database**.
3. In SQL Server Object Explorer, expand localhost\sqlexpress / Databases. Right-click movie_tracker_blazor and select New Query..., paste the following INSERT statements into the query window.
4.
```
INSERT INTO Genre VALUES('Action');
INSERT INTO Genre VALUES('Adventure');
INSERT INTO Genre VALUES('Animation');
INSERT INTO Genre VALUES('Biography');
INSERT INTO Genre VALUES('Comedy');
INSERT INTO Genre VALUES('Crime');
INSERT INTO Genre VALUES('Documentary');
INSERT INTO Genre VALUES('Drama');
INSERT INTO Genre VALUES('Family');
INSERT INTO Genre VALUES('Fantasy');
INSERT INTO Genre VALUES('Film Noir');
INSERT INTO Genre VALUES('History');
INSERT INTO Genre VALUES('Horror');
INSERT INTO Genre VALUES('Music');
INSERT INTO Genre VALUES('Musical');
INSERT INTO Genre VALUES('Mystery');
INSERT INTO Genre VALUES('Romance');
INSERT INTO Genre VALUES('Sci-Fi');
INSERT INTO Genre VALUES('Short Film');
INSERT INTO Genre VALUES('Sport');
INSERT INTO Genre VALUES('Superhero');
INSERT INTO Genre VALUES('Thriller');
INSERT INTO Genre VALUES('War');
INSERT INTO Genre VALUES('Western');
```

5. Click the Execute button in the upper-left corner.
6. Close the query window.

index.html

1. Open wwwroot / index.html.
2. Update the title to **Movie Tracker**.
3. ```
   <!DOCTYPE html>
   <html>

   <head>
       <meta charset="utf-8" />
       <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
       <title>Movie Tracker~~Blazor~~</title>
       <base href="/" />
       <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
       <link href="css/app.css" rel="stylesheet" />
       <link href="MovieTrackerBlazor.Client.styles.css" rel="stylesheet" />
   </head>
   ...
   ```
4. Save the file.

NavMenu.razor

1. Open MovieTrackerBlazor.Client / Shared / NavMenu.razor.
2. Update the application name to **Movie Tracker**. Change the Home link text to Movies, and delete the Counter and Fetch data menu items.
3. ```
   <div class="top-row pl-4 navbar navbar-dark">
       <a class="navbar-brand" href="">Movie Tracker~~Blazor~~</a>
       <button class="navbar-toggler" @onclick="ToggleNavMenu">
           <span class="navbar-toggler-icon"></span>
       </button>
   </div>

   <div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
       <ul class="nav flex-column">
           <li class="nav-item px-3">
               <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                   <span class="oi oi-home" aria-hidden="true"></span> ~~Home~~**Movies**
               </NavLink>
           </li>
           ~~<li class="nav-item px-3">~~
               ~~<NavLink class="nav-link" href="counter">~~
                   ~~<span class="oi oi-plus" aria-hidden="true"></span> Counter~~
               ~~</NavLink>~~
           ~~</li>~~
           ~~<li class="nav-item px-3">~~
               ~~<NavLink class="nav-link" href="fetchdata">~~
                   ~~<span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data~~
               ~~</NavLink>~~
           ~~</li>~~
       </ul>
   </div>

   @code {
       private bool collapseNavMenu = true;

       private string NavMenuCssClass => collapseNavMenu ? "collapse" : null;

       private void ToggleNavMenu()
       {
           collapseNavMenu = !collapseNavMenu;
       }
   }
   ```
4. Save the file.

_Imports.razor

1. Open MovieTrackerBlazor.Client / _Imports.razor.
2. Add the using MovieTrackerBlazor.Shared directive.
3. ```
   @using System.Net.Http
   @using System.Net.Http.Json
   @using Microsoft.AspNetCore.Components.Forms
   @using Microsoft.AspNetCore.Components.Routing
   @using Microsoft.AspNetCore.Components.Web
   @using Microsoft.AspNetCore.Components.Web.Virtualization
   @using Microsoft.AspNetCore.Components.WebAssembly.Http
   @using Microsoft.JSInterop
   @using MovieTrackerBlazor.Client
   @using MovieTrackerBlazor.Client.Shared
   **@using MovieTrackerBlazor.Shared**
   ```
4. Save the file.

Index.razor

1. Open MovieTrackerBlazor.Client / Pages / Index.razor.
2. Update the heading and delete the SurveyPrompt.
3. ```
   @page "/"

   <h1>~~Hello, world!~~**Movie Tracker**</h1>
   ```

```
Welcome to your new app.

<SurveyPrompt Title="How is Blazor working for you?" />
```

4. Inject the **HttpClient**.
5.
```
@page "/"
@inject HttpClient Http

<h1>Movie Tracker</h1>
```

6. Add a code block. Declare an array of movies. Override OnInitializedAsync to retrieve the movies from the Web API.
7.
```
@page "/"
@inject HttpClient Http

<h1>Movie Tracker</h1>

@code {
    private Movie[] movies;

    protected override async Task OnInitializedAsync()
    {
        movies = await Http.GetFromJsonAsync<Movie[]>("movies");
    }
}
```

8. Add some markup to display one of two loading messages.
9.
```
@page "/"
@inject HttpClient Http

<h1>Movie Tracker</h1>

@if (movies == null)
{
    <p><em>Loading...</em></p>
}
else
{
    if (movies.Length == 0)
    {
        <p><em>No movies found.</em></p>
    }
    else
    {
        <p><em>@movies.Length movie(s) found.</em></p>
    }
}

@code {
    private Movie[] movies;

    protected override async Task OnInitializedAsync()
    {
        movies = await Http.GetFromJsonAsync<Movie[]>("movies");
    }
}
```

10. Run the site.

MovieForm.razor

1. Add a Razor component to the Pages folder named **MovieForm.razor**.
2. Delete the heading.
3.
```
<h3>MovieForm</h3>

@code {

}
```

4. Declare 3 parameters for the component.
5.
```
@code {
    [Parameter]
    public Movie movie { get; set; }

    [Parameter]
    public string ButtonText { get; set; }

    [Parameter]
    public EventCallback OnValidSubmit { get; set; }
}
```

6. Start the EditModel specifying the Model and OnValidSubmit properties.
7.
```
<EditForm Model="movie" OnValidSubmit="OnValidSubmit">

</EditForm>

@code {
...
```

8. Add a label and text input for Title.
9.
```
...
<EditForm Model="movie" OnValidSubmit="OnValidSubmit">
    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Title</label>
```

```
        <InputText id="Title" @bind-Value="movie.Title" />
    </div>
</EditForm>
...
```

10. Add a label and date control for DateSeen.
11.
```
...
<EditForm Model="movie" OnValidSubmit="OnValidSubmit">
    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Title</label>
        <InputText id="Title" @bind-Value="movie.Title" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Date Seen</label>
        <InputDate id="DateSeen" @bind-Value="movie.DateSeen" />
    </div>
</EditForm>
...
```

12. Inject the HttpClient.
13.
```
@inject HttpClient Http

<EditForm Model="movie" OnValidSubmit="OnValidSubmit">
...
```

14. Declare a list of genres. Override OnInitializedAsync to retrieve the genres from the Web API.
15.
```
@code {
    [Parameter]
    public Movie movie { get; set; }

    [Parameter]
    public string ButtonText { get; set; }

    [Parameter]
    public EventCallback OnValidSubmit { get; set; }

    private List<Genre> genres = new();

    protected override async Task OnInitializedAsync()
    {
        genres = await Http.GetFromJsonAsync<List<Genre>>("genres");
    }
}
```

16. Add a label and select control for Genre. Loop through the genres collection to create the drop-down list options.
17.
```
...
<EditForm Model="movie" OnValidSubmit="OnValidSubmit">
    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Title</label>
        <InputText id="Title" @bind-Value="movie.Title" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Date Seen</label>
        <InputDate id="DateSeen" @bind-Value="movie.DateSeen" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Genre</label>
        <InputSelect id="GenreId" @bind-Value="movie.GenreId">
            @foreach (var genre in genres)
            {
                <option value="@genre.Id">@genre.GenreDescription</option>
            }
        </InputSelect>
    </div>
</EditForm>
...
```

18. Add a label and number input for Rating.
19.
```
...
<EditForm Model="movie" OnValidSubmit="OnValidSubmit">
    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Title</label>
        <InputText id="Title" @bind-Value="movie.Title" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Date Seen</label>
        <InputDate id="DateSeen" @bind-Value="movie.DateSeen" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Genre</label>
        <InputSelect id="GenreId" @bind-Value="movie.GenreId">
            @foreach (var genre in genres)
            {
                <option value="@genre.Id">@genre.GenreDescription</option>
            }
        </InputSelect>
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Rating</label>
        <InputNumber id="Rating" @bind-Value="movie.Rating" />
    </div>
```

```
</EditForm>
...
```

20. Add validator, validation summary and a submit button.
21.
```
...
<EditForm Model="movie" OnValidSubmit="OnValidSubmit">
    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Title</label>
        <InputText id="Title" @bind-Value="movie.Title" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Date Seen</label>
        <InputDate id="DateSeen" @bind-Value="movie.DateSeen" />
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Genre</label>
        <InputSelect id="GenreId" @bind-Value="movie.GenreId">
            @foreach (var genre in genres)
            {
                <option value="@genre.Id">@genre.GenreDescription</option>
            }
        </InputSelect>
    </div>

    <div class="form-group row">
        <label class="col-sm-2 col-form-label">Rating</label>
        <InputNumber id="Rating" @bind-Value="movie.Rating" />
    </div>

    <DataAnnotationsValidator />
    <ValidationSummary />

    <button type="submit" class="btn btn-primary">@ButtonText</button>
</EditForm>
...
```

22. Save the file.

AddMovie.razor

1. Add a Razor component to the Pages folder named **AddMovie.razor**.
2. Add a page directive to specify the routing, inject the **HttpClient** and the **NavigationManager**. Add a space between Add and Movie in the heading.
3.
```
@page "/addmovie"
@inject HttpClient Http
@inject NavigationManager NavManager

<h3>Add Movie</h3>

@code {

}
```

4. Instantiate a movie. Add a method that will call the Web API to add a movie. Navigate to the home page.
5.
```
@code {
    Movie movie = new();

    async Task AddMovieAsync()
    {
        await Http.PostAsJsonAsync("movies", movie);
        NavManager.NavigateTo("/");
    }
}
```

6. Incorporate the MovieForm component and set its properties.
7.
```
@page "/addmovie"
@inject HttpClient Http
@inject NavigationManager NavManager

<h3>Add Movie</h3>

<MovieForm ButtonText="Add" movie="movie" OnValidSubmit="AddMovieAsync" />

@code {
    Movie movie = new();

    async Task AddMovieAsync()
    {
        await Http.PostAsJsonAsync("movies", movie);
        NavManager.NavigateTo("/");
    }
}
```

8. Save the file.

Index.razor

1. Add a button that links to the AddMovie component and update the no movies found text.
2.
```
...
<h1>Movie Tracker</h1>

@if (movies == null)
{
```

```
    <p><em>Loading...</em></p>
}
else
{
    <a class="btn btn-primary" href="/addmovie" title="Add"><span class="oi oi-plus" /></a>
    <p></p>

    if (movies.Length == 0)
    {
        <p><em>No movies found. Click the add button to add a movie.</em></p>
    }
    else
    {
        <p><em>@movies.Length movie(s) found.</em></p>
    }
}
...
```

3. Save the file. Run the site. Add a movie.
4. Remove the placeholder message and add a table to render the movies.
5.

```
...
<h1>Movie Tracker</h1>

@if (movies == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <a class="btn btn-primary" href="/addmovie" title="Add"><span class="oi oi-plus" /></a>
    <p></p>

    if (movies.Length == 0)
    {
        <p><em>No movies found. Click the add button to add a movie.</em></p>
    }
    else
    {
        <p><em>@movies.Length movie(s) found.</em></p>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>Title</th>
                    <th>Date Seen</th>
                    <th>Genre</th>
                    <th class="text-right">Rating</th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var movie in movies)
                {
                    <tr>
                        <td>@movie.Title</td>
                        <td>@(string.Format("{0:yyyy-MM-dd}", movie.DateSeen))</td>
                        <td>@movie.Genre.GenreDescription</td>
                        <td class="text-right">@movie.Rating</td>
                        <td></td>
                    </tr>
                }
            </tbody>
        </table>
    }
}
...
```

EditMovie.razor

1. Add a Razor component to the Pages folder named **EditMovie.razor**.
2. Add a page directive to specify the routing, inject the **HttpClient** and the **NavigationManager**. Add a space between Add and Movie in the heading.
3.

```
@page "/editmovie/{id:int}"
@inject HttpClient Http
@inject NavigationManager NavManager

<h3>Edit Movie</h3>

@code {

}
```

4. Add a parameter for the movie id and a movie property.
5.

```
@code {
    [Parameter]
    public int id { get; set; }

    Movie movie = new();
}
```

6. Implement OnParametersSetAsync to retrieve the specified movie.
7.

```
@code {
    [Parameter]
    public int id { get; set; }

    Movie movie = new();

    protected async override Task OnParametersSetAsync()
    {
```

```
        movie = await Http.GetFromJsonAsync<Movie>($"movies/{id}");
    }
}
```

8. Provide a method that will be called to update the movie.
9.
```
@code {
    [Parameter]
    public int id { get; set; }

    Movie movie = new();

    protected async override Task OnParametersSetAsync()
    {
        movie = await Http.GetFromJsonAsync<Movie>($"movies/{id}");
    }

    async Task EditMovieAsync()
    {
        await Http.PutAsJsonAsync($"movies/{movie.Id}", movie);
        NavManager.NavigateTo("/");
    }
}
```

10. Incorporate the MovieForm component and set its properties.
11.
```
@page "/editmovie/{id:int}"
@inject HttpClient Http
@inject NavigationManager NavManager

<h3>Edit Movie</h3>

<MovieForm ButtonText="Update" movie="movie" OnValidSubmit="EditMovieAsync" />

@code {
    [Parameter]
    public int id { get; set; }

    Movie movie = new();

    protected async override Task OnParametersSetAsync()
    {
        movie = await Http.GetFromJsonAsync<Movie>($"movies/{id}");
    }

    async Task EditMovieAsync()
    {
        await Http.PutAsJsonAsync($"movies/{movie.Id}", movie);
        NavManager.NavigateTo("/");
    }
}
```

12. Save the file.

Index.razor

1. Add a link in the table row that looks like a button to enable editing a movie.
2.
```
...
<tbody>
    @foreach (var movie in movies)
    {
        <tr>
            <td>@movie.Title</td>
            <td>@(string.Format("{0:yyyy-MM-dd}", movie.DateSeen))</td>
            <td>@movie.Genre.GenreDescription</td>
            <td class="text-right">@movie.Rating</td>
            <td>
                <a class="btn btn-primary" href="/editmovie/@movie.Id" title="Edit"><span class="oi oi-pencil" /></a>
            </td>
        </tr>
    }
</tbody>
...
```

3. Save the file. Refresh the browser. Edit the movie.
4. Inject the JavaScript runtime.
5.
```
@page "/"
@inject HttpClient Http
@inject IJSRuntime JS

<h1>Movie Tracker</h1>
...
```

6. Add a method to delete a movie, which will re-initialize the component.
7.
```
@code {
    private Movie[] movies;

    protected override async Task OnInitializedAsync()
    {
        movies = await Http.GetFromJsonAsync<Movie[]>("movies");
    }

    async Task DeleteAsync(int id)
    {
        var movie = movies.First(m => m.Id == id);

        if (await JS.InvokeAsync<bool>("confirm", $"Are you sure you want to delete {movie.Title}?"))
        {
            await Http.DeleteAsync($"movies/{id}");
```

```
            await OnInitializedAsync();
        }
    }
}
```

8. Add a button in the table row to call the delete method.
9.
```
...
<tbody>
    @foreach (var movie in movies)
    {
        <tr>
            <td>@movie.Title</td>
            <td>@(string.Format("{0:yyyy-MM-dd}", movie.DateSeen))</td>
            <td>@movie.Genre.GenreDescription</td>
            <td class="text-right">@movie.Rating</td>
            <td>
                <a class="btn btn-primary" href="/editmovie/@movie.Id" title="Edit"><span class="oi oi-pencil" /></a>
                <button class="btn btn-danger" @onclick="DeleteAsync(movie.Id)" title="Delete"><span class="oi oi-trash" /></button>
            </td>
        </tr>
    }
</tbody>
...
```

10. Notice that there is an error with the call to DeleteAsync. The call must provide a closure.
11.
```
@onclick="(() => DeleteAsync(movie.Id))"
```

12. Save the file. Refresh the browser. Delete the movie.