Blazor: Getting Started

BUILDING YOUR FIRST BLAZOR APPLICATION

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

Overview

- What will you learn from this course?
- Hello Blazor
- The different hosting models of Blazor
- Understanding File → New Project
- Creating a first Blazor app
- Debugging a Blazor app

What Will You Learn from This Course?

This course will teach you how to build your first Blazor application.

How We Will Approach This Course

File → New Project...

Quite some code...

And then, a full Blazor application!

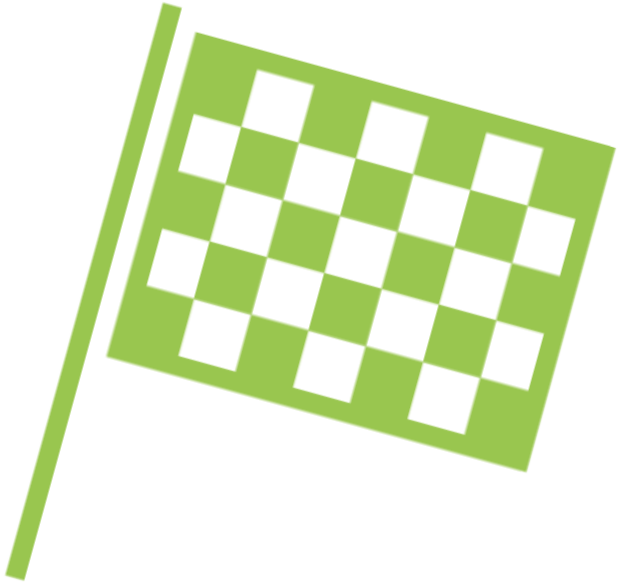
What I Assume You Know

HTML and CSS

C#

Some Razor is recommended

What You Need to Have Installed



Visual Studio 2019 (16.6 or higher)

- https://visualstudio.microsoft.com/vs/

.NET Core 3.1 SDK

- https://dotnet.microsoft.com/download/dotnet-core/3.1

.NET 5 compatible

A browser

The Scenario: Bethany's Pie Shop HRM

- List of employees
- Detail page
- Add new employee
- Navigation

Demo

Taking a look at the finished application

Blazor: Getting Started

Start Course

Bookmark

Add to Channel

Download Course

Table of contents | Description | Transcript | Exercise files | Discussion | Recommended

Hello Blazor

Blazor is a framework
to build interactive web UIs
using C# and HTML.

Hello Blazor

Based on WebAssembly or run on server

No plugin, based on web standards

Integrate with JavaScript

Benefits of Visual Studio and .NET including performance and libraries

Leverage your C# skills
to build interactive web applications.

Blazor Roadmap

2018
Experimental project

April 2019
Blazor preview

September 2019
Blazor released

May 2020
Client-side Blazor

November 2020
Part of .NET 5

Later
Other platforms

Demos are built using .NET Core 3.1

All code is compatible with .NET 5

.NET 5 module at the end of this course

Blazor: The Big Picture

by Barry Luijbregts

Blazor is an incredible framework that you can use to develop rich and interactive client-side web UIs with C# instead of JavaScript. In this course, you'll learn what Blazor is and why you should use it.

Start Course | Bookmark | Add to Channel | Download Course

Table of contents | Description | Transcript | Exercise files | Discussion | Recommended

https://www.pluralsight.com/library/courses/blazor-big-picture

The Different Hosting Models of Blazor

Different Flavors of Blazor

Client-side

Server-side

Client-side Blazor

Blazor app
WebAssembly

Runs on all modern browsers

No .NET required on server

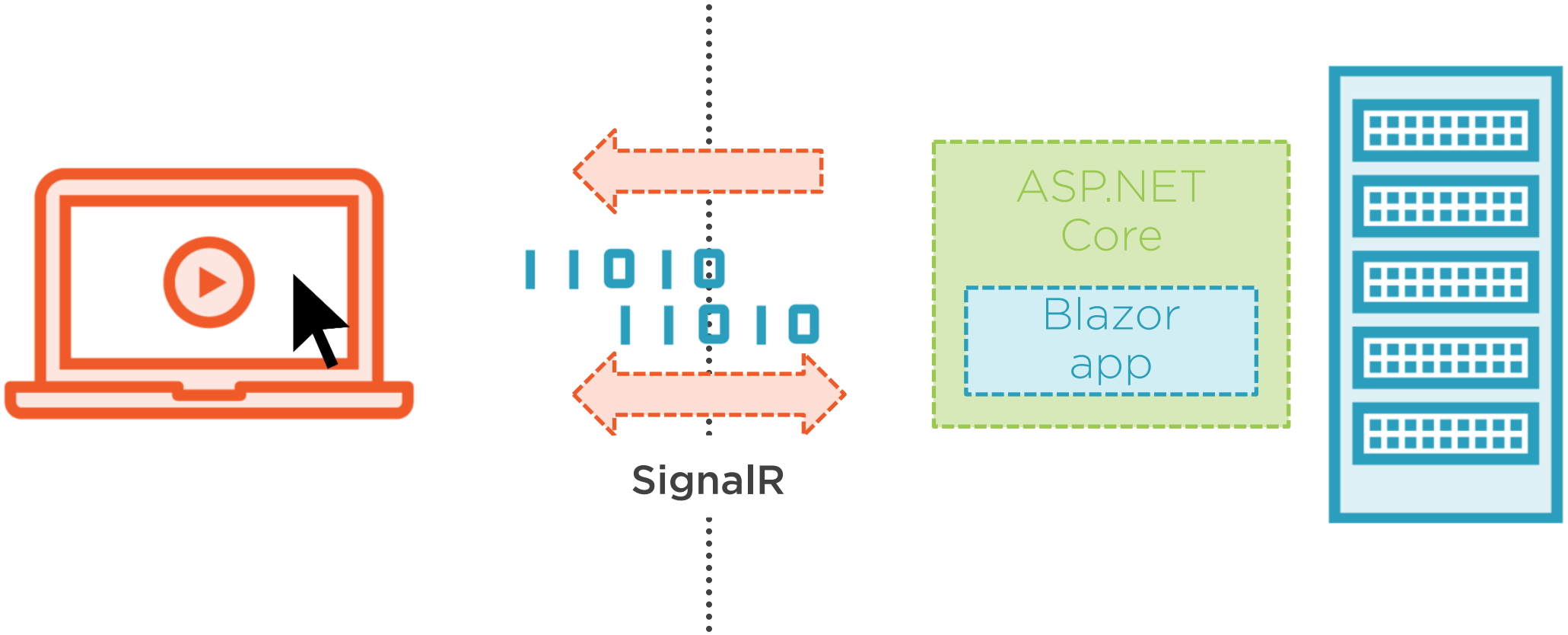
SPA user experience

Older browsers might not be supported

Initial app download is larger

Debugging support

Server-side Blazor



Small download

Works with all server-side APIs

Full debugging support

Blazor apps in non-supported browsers

No offline support

Network delay

Scalability, although not a big problem

We'll use client-side Blazor and convert to server-side later.

Let’s now dive into Blazor code!

Understanding File → New Project

Visual Studio Templates

Client-side WebAssembly

Standalone

Client-side WebAssembly

ASP.NET Core Hosted

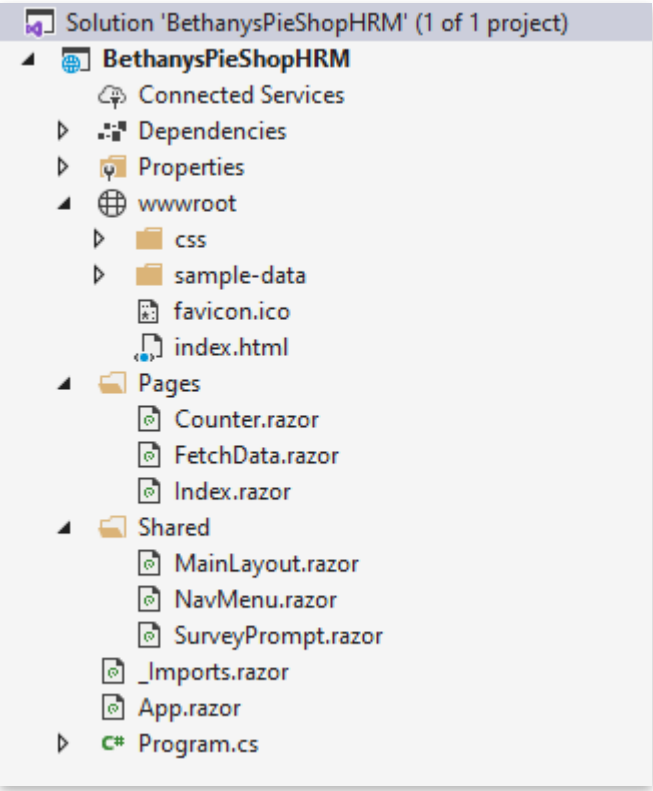
Server-side

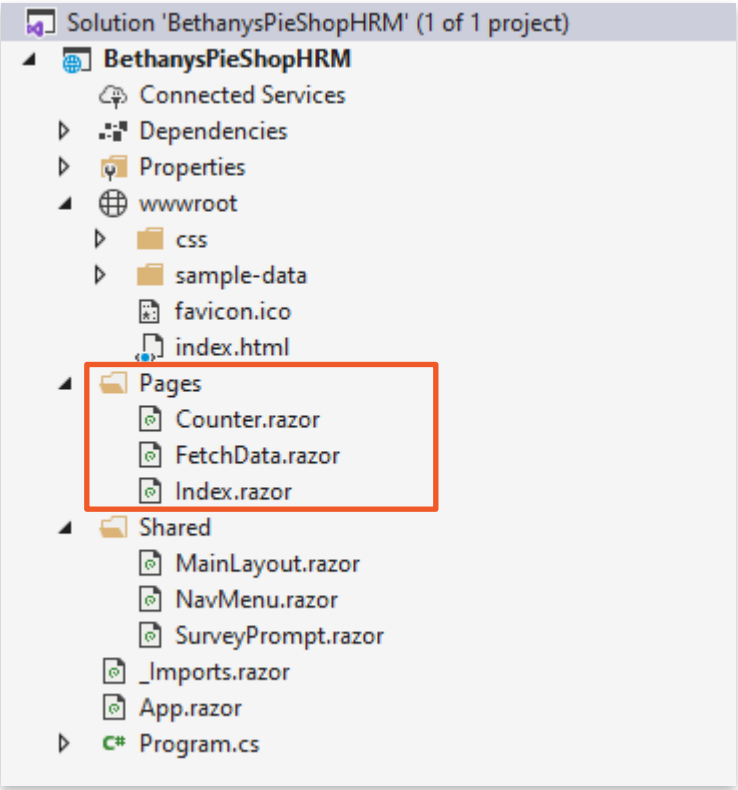
New Project

C# and .razor files

Structure similar to ASP.NET Core project

- Program.cs





*.razor files

Components are building blocks

Name must start with uppercase

Class generated upon compilation

Looking at a the First Component

```
@page "/counter"
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
@code {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

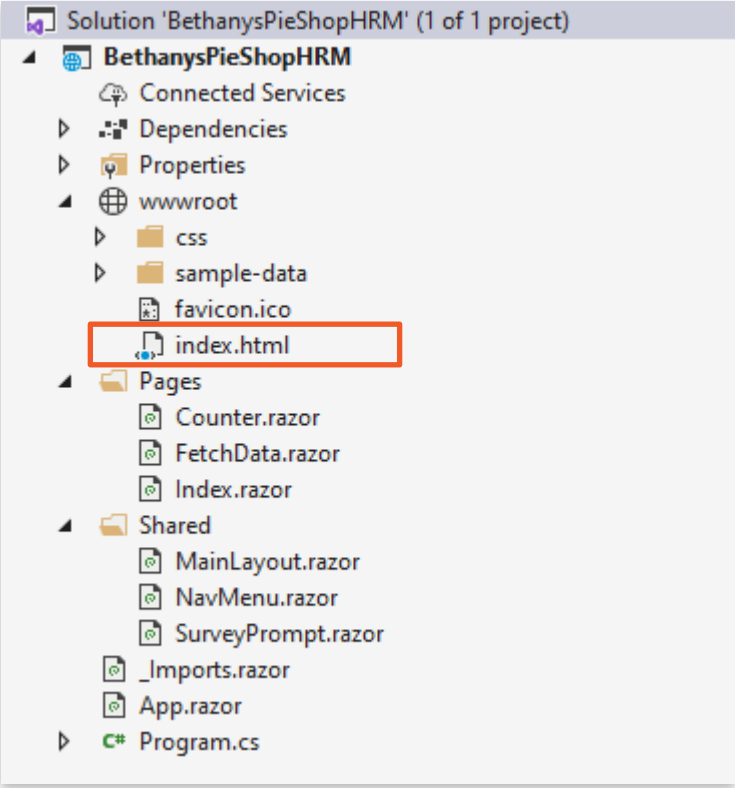
```
@page "/"

<h1>Hello, world!</h1>

Welcome to your new app.

<Counter />
```

Using a Component



Hosting page

Plain HTML

Trigger loading of your Blazor app

- blazor.webassembly.js

Demo

Creating a new project

Looking at the created files

New to ASP.NET Core?
Take a look at the
ASP.NET Core Learning Path!

Creating Your First Blazor App

Using Code

Mixed approach using @code

“Code behind” using partial

Mixed Approach

```
@page "/counter"
<h1>Counter</h1>
<p>Current count: @currentCount</p>
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
@code {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

```
public partial class EmployeeOverview
{

}
```

Using Partial Classes

Demo

Creating your first app

Demo

Adding your own layout

Demo

Debugging a Blazor app in Visual Studio

Summary

Blazor allows creating rich experiences in the browser using C# and HTML

Server-side and client-side

Components are a building block

Up next:

Working with data

Working with Data

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

Overview

Accessing real data from a REST API

Creating a form

Adding validation

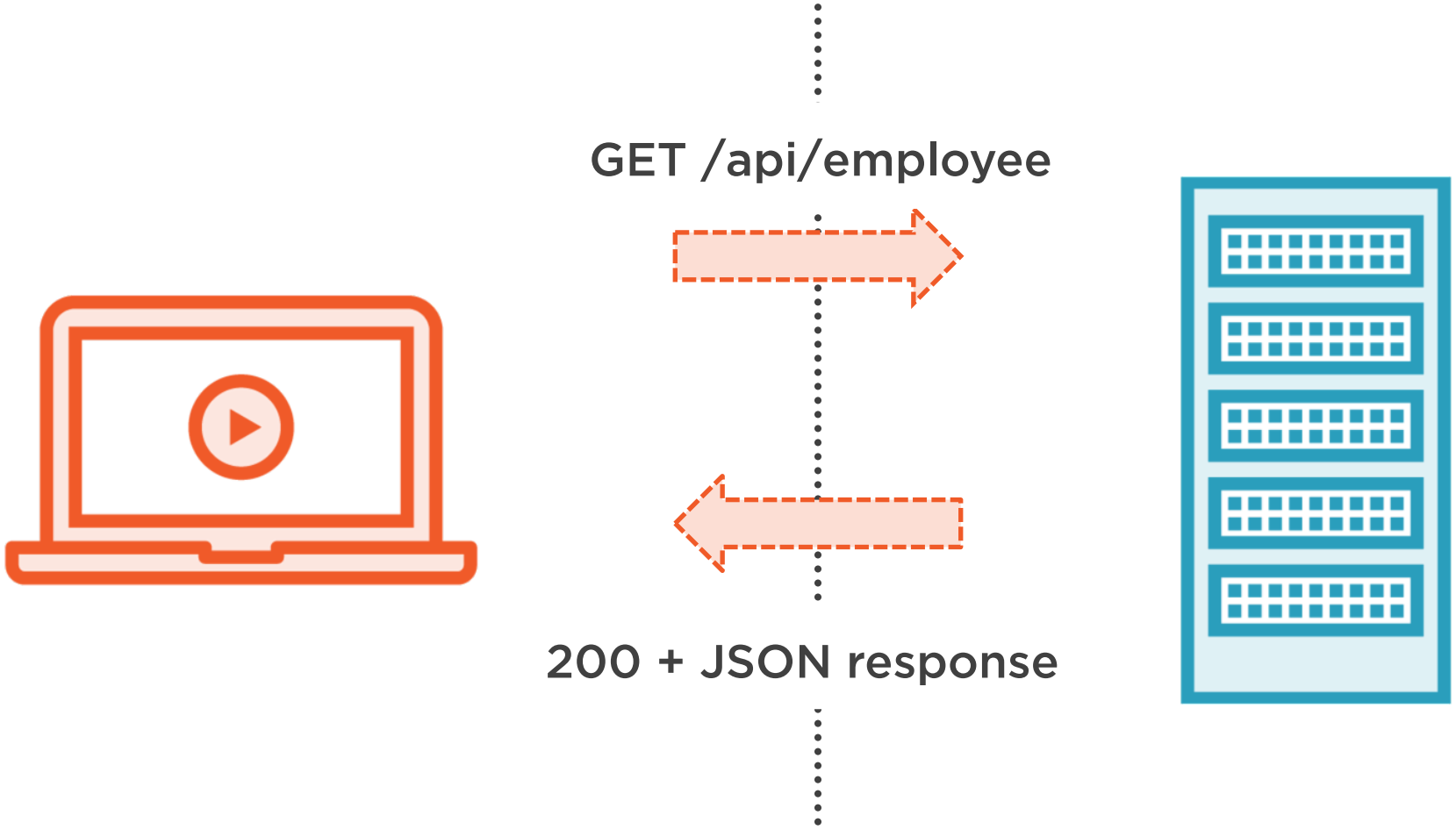
Accessing Real Data from a REST API

Data in Our Blazor App

REST API

Local storage

Accessing a REST API



Demo

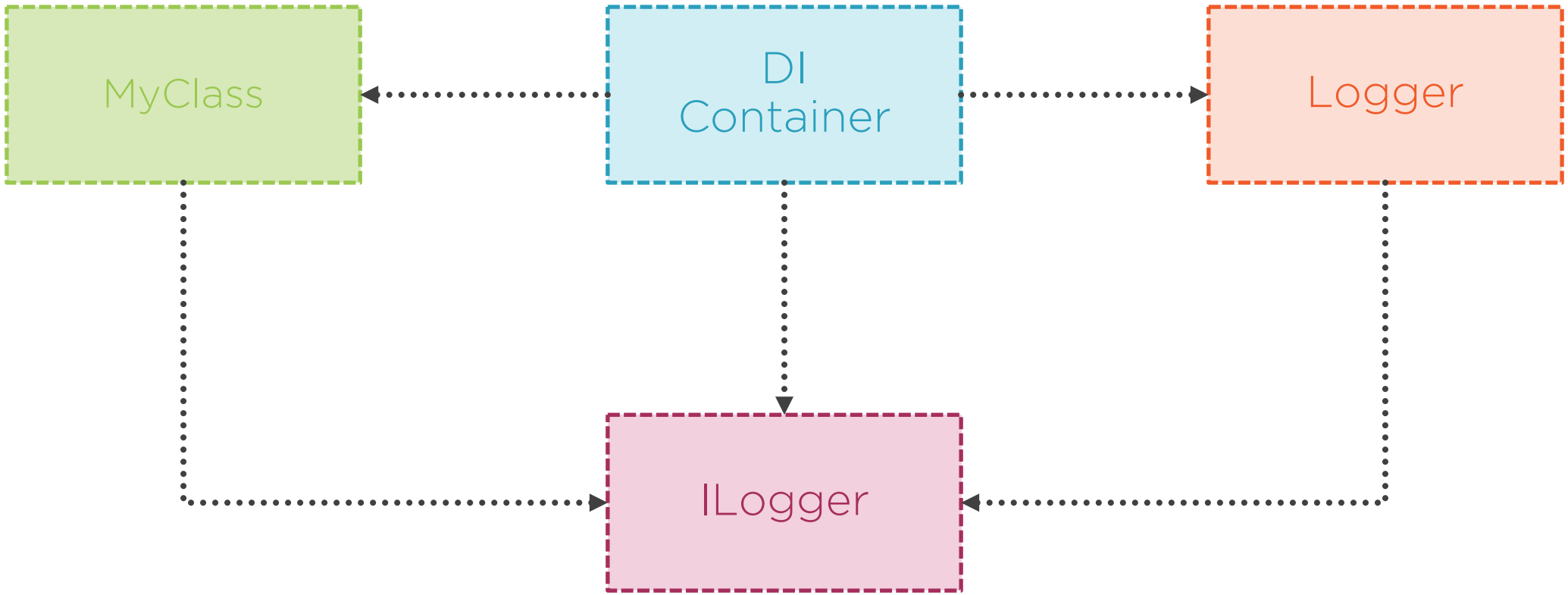
Exploring the API

Interacting with REST APIs

HttpClient

IHttpClientFactory

Sidestep: Dependency Injection



```
builder.Services.AddTransient(sp =>
    new HttpClient
    {
        BaseAddress = new Uri("http://<your-api-endpoint>")
    }
);
```

Using the HttpClient Service

```
[Inject]
public HttpClient HttpClient { get; set; }
```

Accessing the HttpClient in a Component

```
protected override async Task OnInitializedAsync()
{
    Employees = await HttpClient.GetFromJsonAsync<Employee[]>("api/employee");
}
```

Working with the JSON Helper Methods

JSON Helper Methods

GetFromJsonAsync()

PostAsJsonAsync()

PutAsJsonAsync()

DeleteAsync()

HttpClientFactory

Used to configure and create HttpClient instances in a central location

```
builder.Services.AddHttpClient
  <IEmployeeDataService, EmployeeDataService>
    (client => client.BaseAddress = new Uri("https://localhost:44340/"));
```

Working with the HttpClientFactory

Requires NuGet package: Microsoft.Extensions.Http

Constructor Injection in Services

```
public class EmployeeDataService : IEmployeeDataService
{
    private readonly HttpClient _httpClient;

    public EmployeeDataService(HttpClient httpClient)
    {

        _httpClient = httpClient;
    }
}
```

Demo

Adding the HttpClient

Creating a “real” data service

Updating the master and detail page

Learn more about connecting securely to APIs: Authentication and Authorization in Blazor
by Kevin Dockx

Creating a Form

Data binding support in Blazor

- One-way
- Two-way
- Component parameter

One-way Binding

```
<h1 class="page-title">
    Details for @Employee.FirstName @Employee.LastName
</h1>

public Employee Employee { get; set; }
```

One-way Binding in a Form Control

```
<label type="text" readonly class="form-control-plaintext">
    @Employee.FirstName
</label>

public Employee Employee { get; set; }
```

```
<input id="lastName" @bind="@Employee.LastName"
    placeholder="Enter last name" />
```

Two-way Binding

```
<input id="lastName" @bind-value="Employee.LastName"
    @bind-value:event="oninput"
    placeholder="Enter last name" />
```

Two-way binding on a Different Event

Demo

Testing data binding

Forms in Blazor: EditForm

- Input components
- Data binding
- Validation

Input Components

InputText

InputTextArea

InputNumber

InputSelect

InputDate

InputCheckbox

Creating a Form

```
<EditForm Model="@Employee"
    OnValidSubmit="@HandleValidSubmit"
    OnInvalidSubmit="@HandleInvalidSubmit">

    <InputText id="lastName"
        @bind-Value="@Employee.LastName"
        placeholder="Enter last name">
    </InputText>

</EditForm>
```

Demo

Adding the Add Employee form

Demo

Adding more input components

Demo

Saving the data

Learn more about
data binding in:
Creating Blazor Components
by Roland Guijt

Adding Validation

Validation in Blazor

- Similar to ASP.NET Core validations
- Data annotations
- DataAnnotationsValidator
- ValidationSummary

Demo

Adding validation

Summary

- Blazor makes working with data easy
- Data binding engine included
- Specific form components
- Validation support

Up next:

Adding more features to the app

Adding Features to the App

Gill Cleeren
CTO XPIRIT BELGIUM

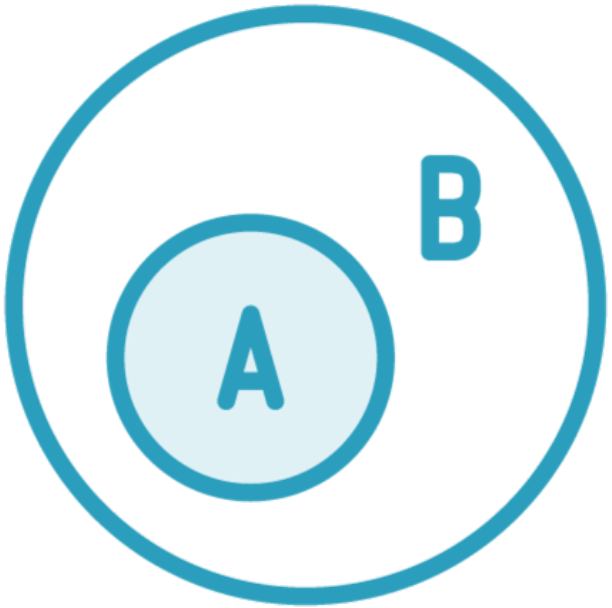
@gillcleeren www.snowball.be

Overview

Adding a dialog component

Integrating a JavaScript component

Adding a Dialog Component



Components are building block

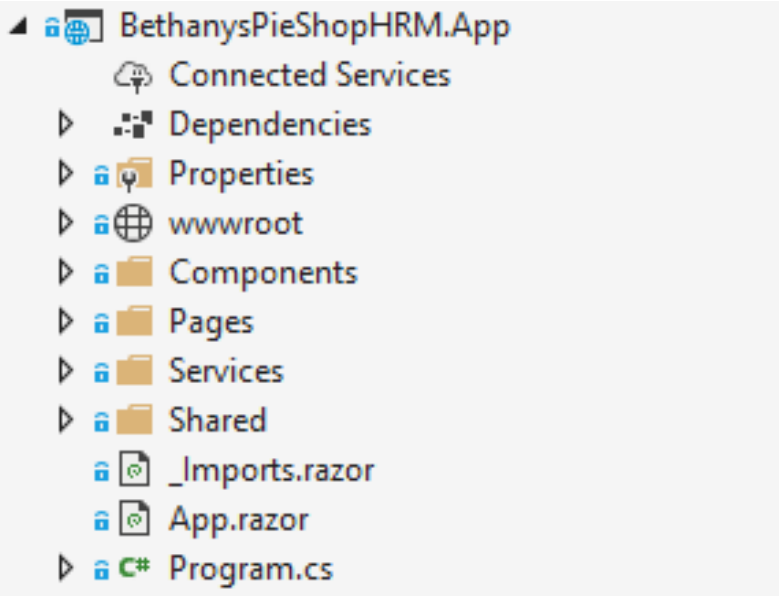
- Page
- Dialog

Reuse of functionality

Make large *pages* smaller

Can be nested

In-project or separate library



Pages

Shared

Components

```
@using BethanysPieShopHRM.Server.Components
```

Using _Imports.razor

Component Lifecycle Methods

OnInitialized

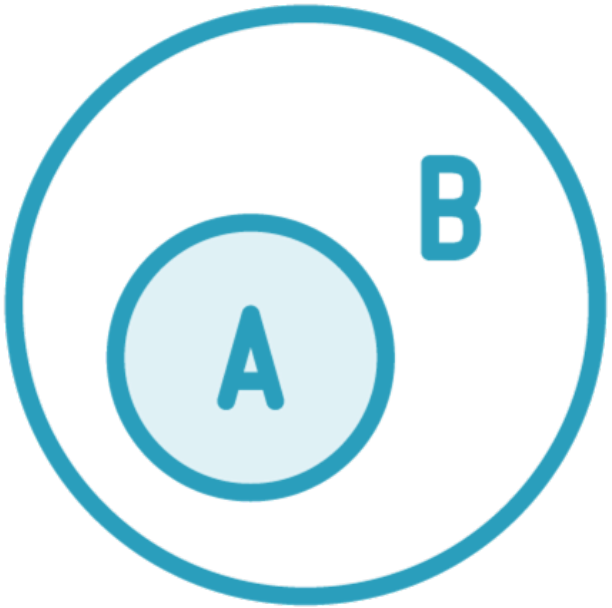
OnInitializedAsync

OnParametersSet

OnParametersSetAsync

OnAfterRender

OnAfterRenderAsync



Can receive parameters

Event binding for component communication

Demo

Creating the Add Employee dialog

Adding the component to the parent page

Component communication

Learn more about components in:
Creating Blazor Components by Roland Guijt

Integrating a JavaScript Component

Blazor apps are just web pages running on the server or client.

Not everything is possible via just .NET

JavaScript interop

- Call into JavaScript from Blazor code

Runs on the client

JavaScript Interop

.NET code calls into JavaScript

JavaScript calls into .NET code

Can be wrapped in a library

```
[Inject]
public IJSRuntime JsRuntime { get; set; }
```

Injecting IJSRuntime

Adding a Script

```
<script>
    window.DoSomething = () => {
        //do some interesting task here
    }
</script>
```

```
var result = await
JsRuntime.InvokeAsync<object>("DoSomething", "");
```

Calling into JavaScript

Demo

Adding a map via JavaScript interop

Map component:
https://aka.ms/blazorworksho
p

Learn more about this in JavaScript interop in Blazor by Thomas Claudius Huber

Summary

Blazor apps are made out of components

Can be reused across projects

Extend Blazor through JS interop

Up next:

Looking at server-side Blazor

Converting to Server-side Blazor

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

Overview

Introducing server-side Blazor

Converting the app to server-side Blazor

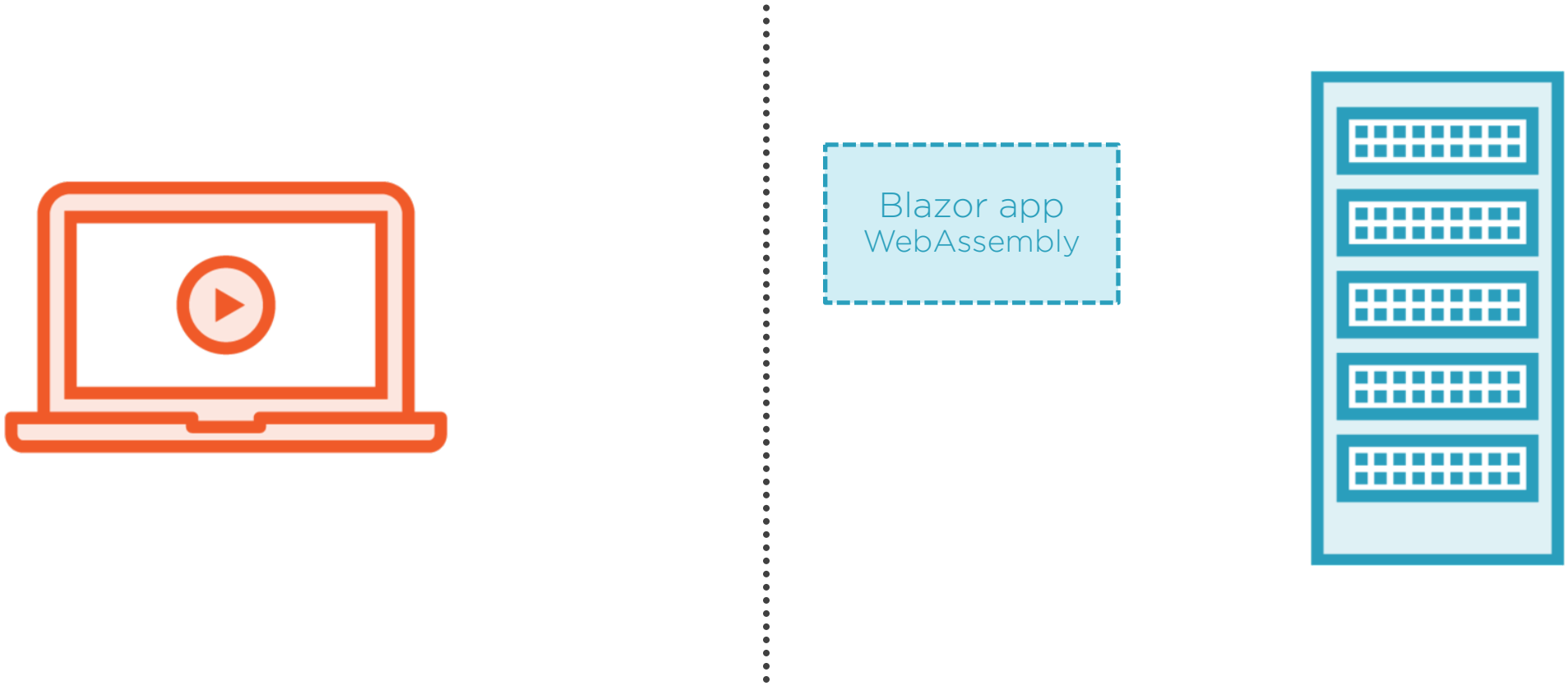
Introducing Server-side Blazor

Blazor Hosting Models

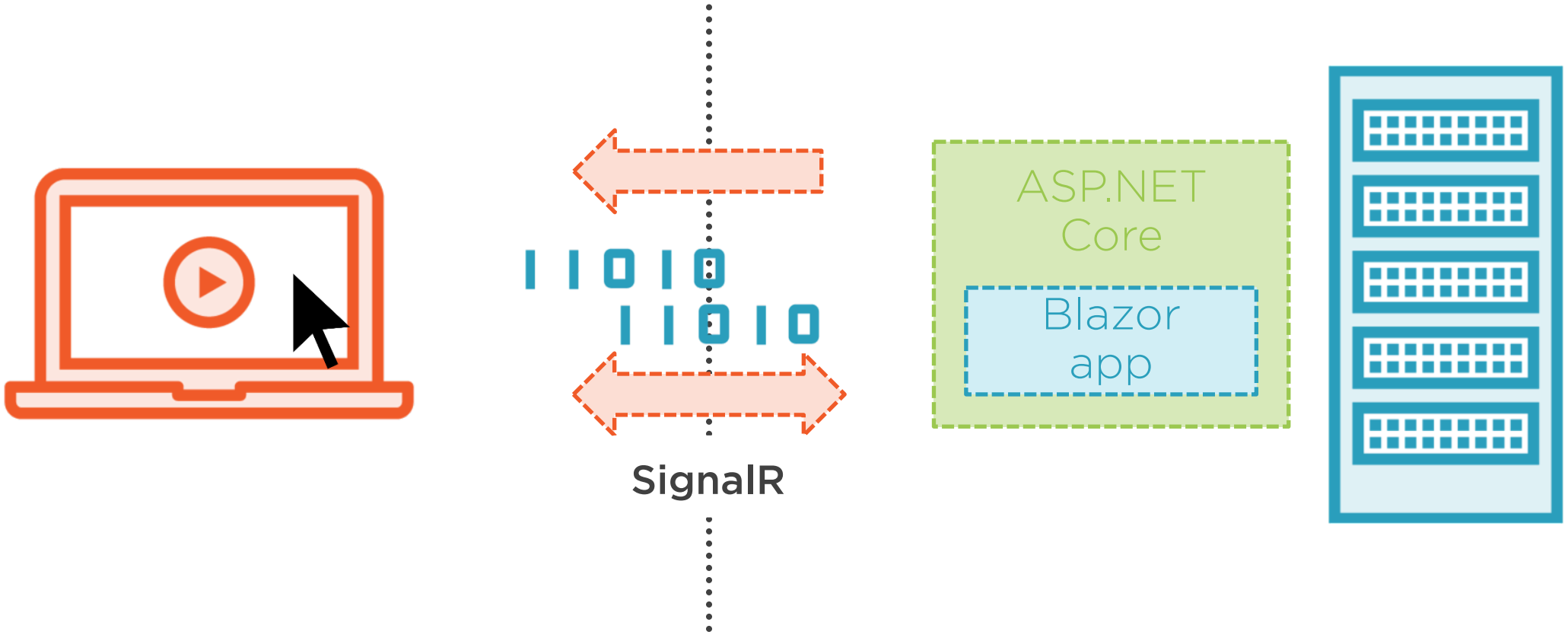
Server-side

Client-side

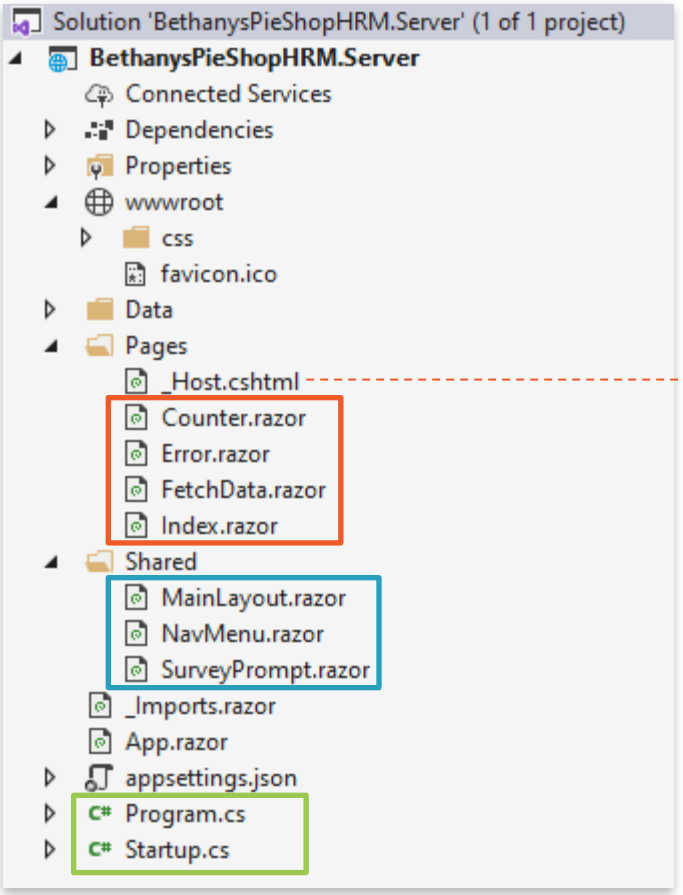
Client-side Blazor Architecture

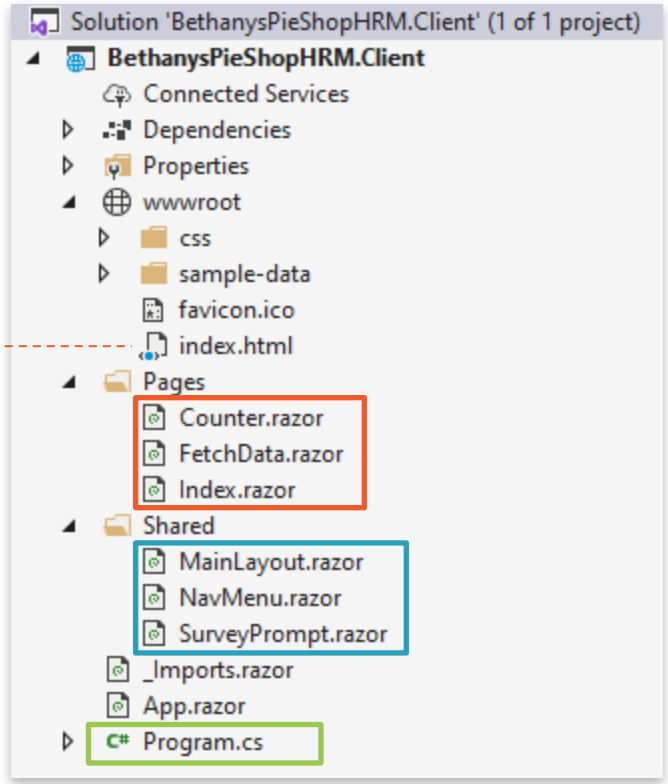


Server-side Blazor Architecture



Comparing Server-side and Client-side Blazor





Server-side Blazor

Client-side Blazor

Invoking Server-side Blazor

Different js File

_Host.cshtml

```
<html lang="en">
<body>
    <app>

        <component type="typeof(App)" render-mode="ServerPrerendered" />

    </app>

    <script src="_framework/blazor.server.js"></script>

</body>

</html>
```

Choosing for Server-side Blazor

- No initial download, all code executes on server
- All .NET Core server features become available
- Code isn't shared with clients
- No offline support
- Scalability

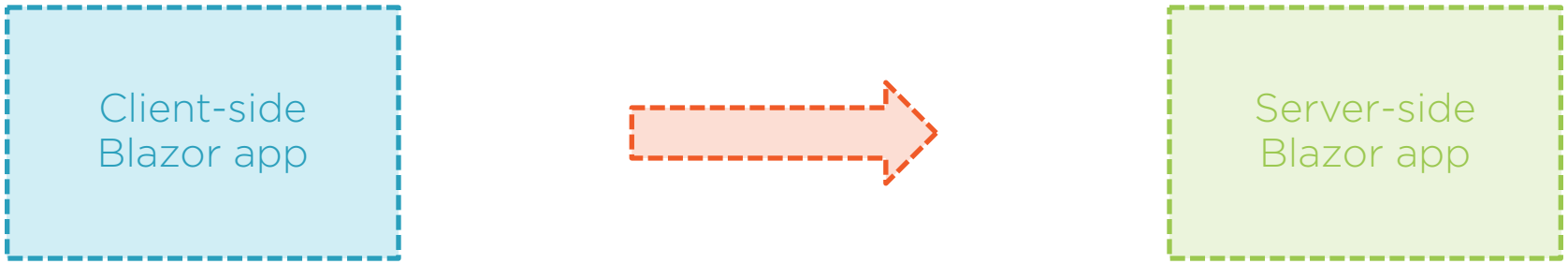
Demo

Exploring File → New Project for server-side Blazor

Looking at the running app with the browser tools

Converting the App to Server-side Blazor

Server-side to Client-side



Sharing code

- Ensure your app can be easily switched
- Extract what is different
- Share what is common

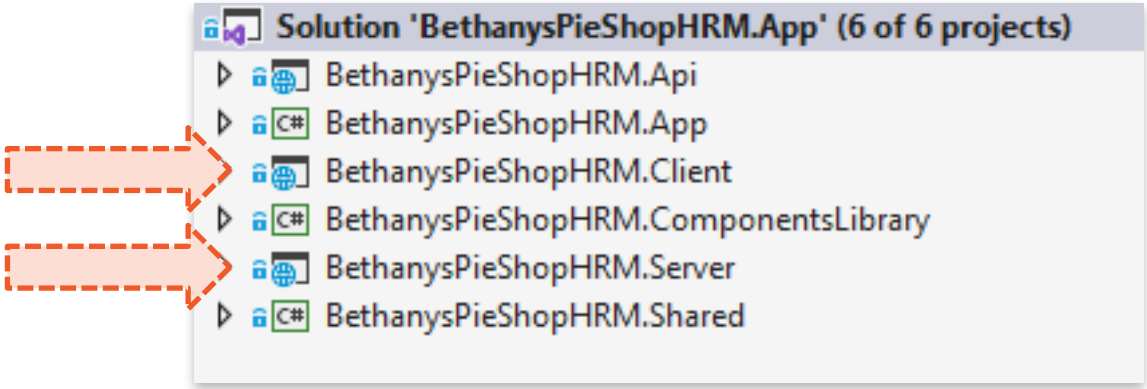
Things to Consider

EF Core

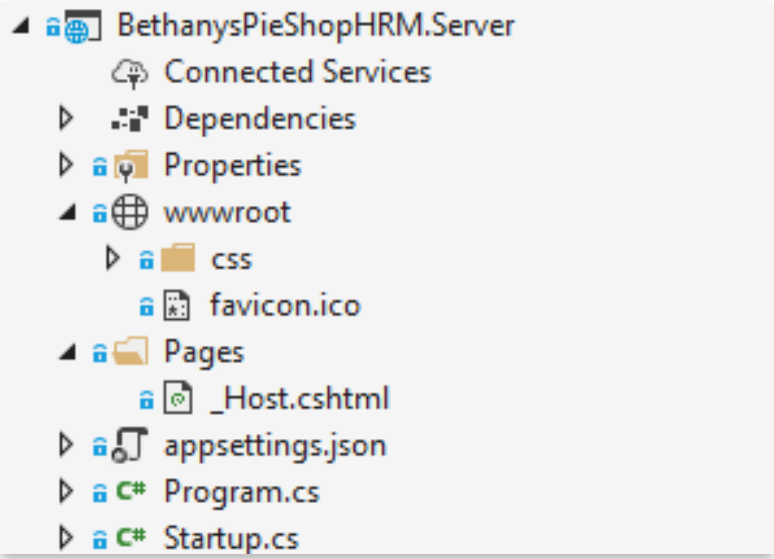
REST APIs

HttpClient usage

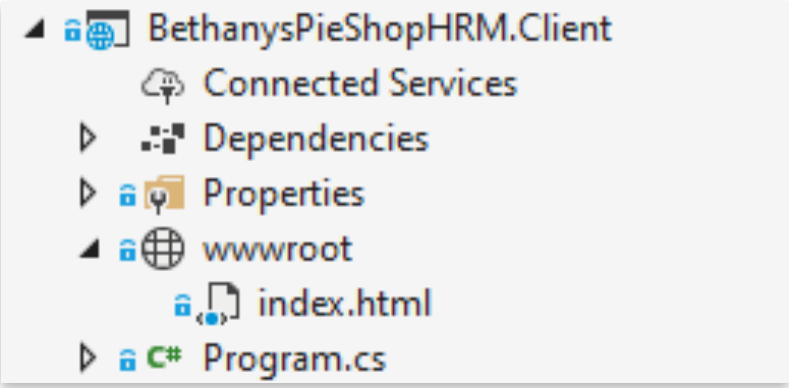
Proposed Approach

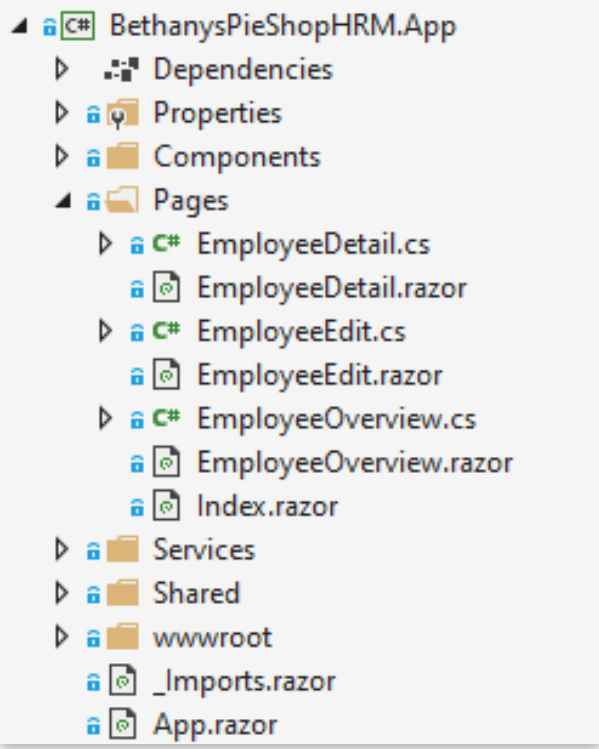


Server-side Project



Client-side Project





App Project

- index.html is removed
- Project is now a Razor Class Library
- All code can be fully reused

Demo

Adding a server-side project

Adding a new client-side project

Converting our original project

Summary

Client-side Blazor projects can easily be converted to server-side by following some patterns

All code is by default possible to share

- It's essentially the same product

Up next:

Deploying the application

Deploying Your Application

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

Overview

Server requirements

Performing a manual publish of the app

Server Requirements

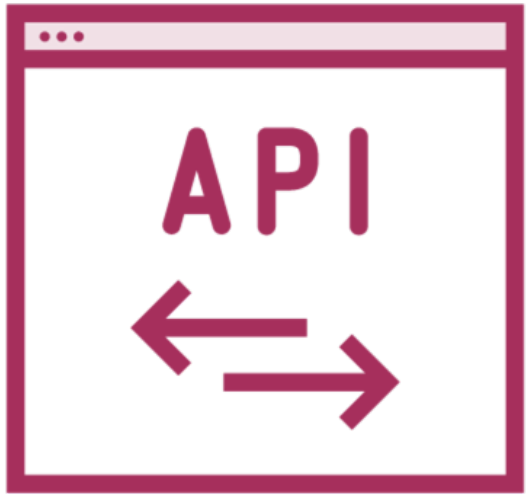
API

- ASP.NET Core
- SQL Server

Blazor app

- No specific server requirements
- Static files
- Hosting HTML file
- Connect with API
  - CORS
- Compression

Used Azure Services



API

Azure App Service (Web Apps)

Azure SQL Database

Blazor app

Azure App Service (Web Apps)

Azure Storage

Blazor server-side app

- ASP.NET Core application
  - Azure App Service
- SignalR connection
  - Azure SignalR Service

Demo

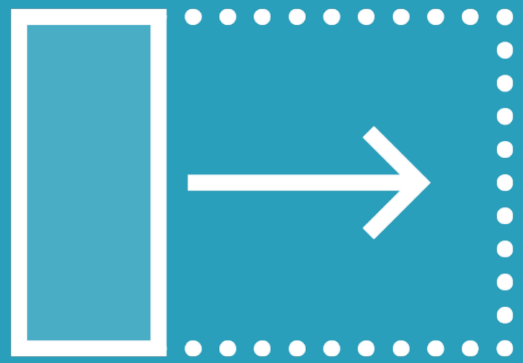
Looking at the deployed API

Demo

Deploying the Blazor application

Summary

Azure is a good fit to publish a Blazor app



Up next:

Updating to .NET 5

Enhancing the Application with .NET 5 Features

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

Overview

Introducing .NET 5 for Blazor

Migrating the application

Adding new features

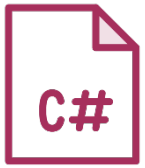
Improvements in the developer experience

Introducing .NET 5 for Blazor

Introducing .NET 5

Introduced late 2020

Part of unification of .NET



C# 9.0

New features for technologies

Blazor Is Part of .NET 5

Cross-platform web UI technology

Several new features added

All existing code and functionality remains the same

Migrating the Application

Demo

Migrating the application to .NET 5

Adding New Features

New Components in Blazor

InputFile

InputRadio

InputRadioGroup

InputFile

- Uploading files from Blazor
- Single or multiple files
- OnChange event

```
<InputFile OnChange="OnInputFileChange" multiple />


private async Task OnInputFileChange(InputFileChangeEventArgs e)
{    }
```

Using the InputFile

Demo

Uploading an image for new employees

Storing the image on the server from the API

Radio button and radio button group

Single selection from range

Support for data binding

Replace InputSelect or InputCheckbox

```
<InputRadioGroup Name="jobType" @bind-Value="@Employee.JobType">
    <InputRadio Name="jobType" Value="JobType.Finance" />
    <InputRadio Name="jobType" Value="JobType.Management" />
</InputRadioGroup>
```

Using the InputRadio and InputRadioGroup

Demo

Adding the InputRadio and InputRadioGroup

Demo

Setting the focus in the Edit Form

List Performance

```
<table class="table">
    @foreach (var employee in Employees)
    {
        <tr>
            <td><img src="@employee.ImageName" width="75"  /></td>
            <td>@employee.EmployeeId</td>
            <td>@employee.FirstName</td>
            <td>@employee.LastName</td>
        </tr>
    }
</table>
```

Introducing Virtualization

Render just the components that are in view

Mainly useful for long lists

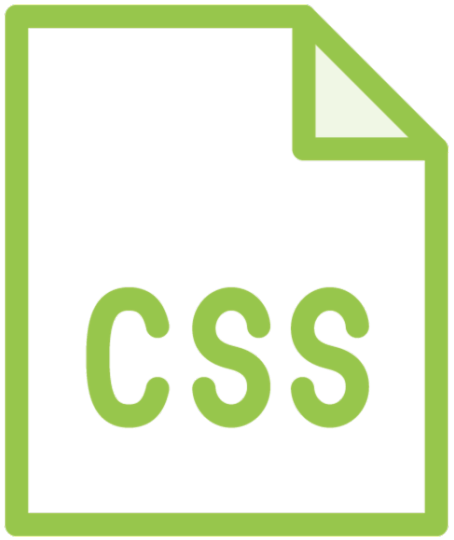
Using Virtualize

```
<table class="table">
    <Virtualize Items="Employees" ItemSize="itemHeight">
        <tr @key="context.GetHashCode()" style="@ItemStyle">
            <td>@context.EmployeeId</td>
            <td>@context.FirstName</td>
            <td>@context.LastName</td>
        </tr>
    </Virtualize>
</table>
```

Demo

Using Virtualize to visualize a long list of employees

Using the item provider delegate



CSS Isolation

- Limit styles to single component
- Less use of global styles
- Avoid conflicts with other CSS libraries

ComponentName.razor.css

Demo

Adding CSS isolation

Moving styles from the global CSS file

Lazy Loading

- Defer loading assemblies until required
- Downloaded based on navigation
- Assemblies are cached
- Speed improvement in start of the application

```
<ItemGroup>
    <BlazorWebAssemblyLazyLoad Include="BethanysPieShopHRM.ComponentsLibrary.dll" />
</ItemGroup>
```

Project File Change

Demo

Extending the project file

Changing the Router

Other Improvements

- Overall performance improvements
- Catch-all
- JS isolation
- More pre-rendering options
- Protected Storage (server-side Blazor only)

Improvements in the Developer Experience

Developer Experience Enhancements

File grouping

Compat analyzer

Improved debugging experience

Demo

Using dotnet watch for improved debugging experience

Using file grouping in the Solution Explorer

Using compat analyzer in Visual Studio

“Thank you for this great app!”

Bethany

Other Blazor Courses

Blazor: The Big Picture (Barry Luijbregts)

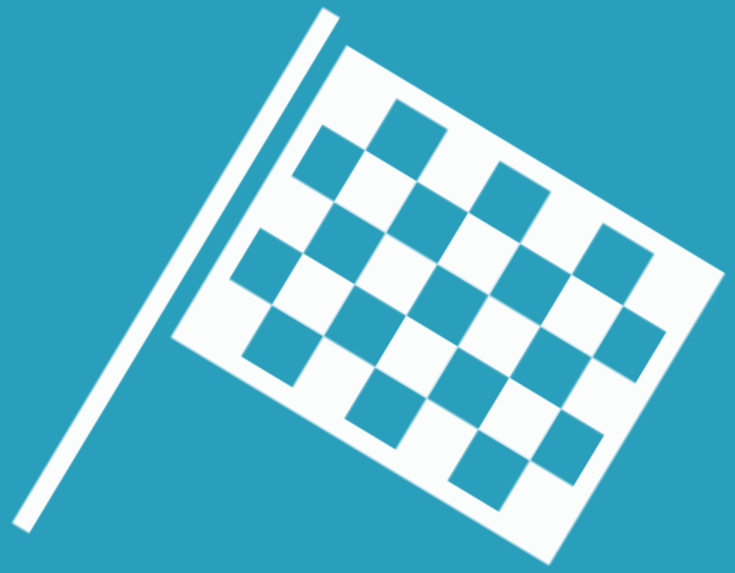
Blazor: Getting Started (this course)

Enterprise apps with Blazor (Alex Wolf)

Creating Blazor Components (Roland Guijt)

Authentication with Blazor (Kevin Dockx)

JavaScript interop in Blazor Applications (Thomas Claudius Huber)



Congratulations
on finishing this course!