Making the Site Interactive

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren – xpirit.com/gill

Module overview

Searching using JavaScript and an ASP.NET Core API

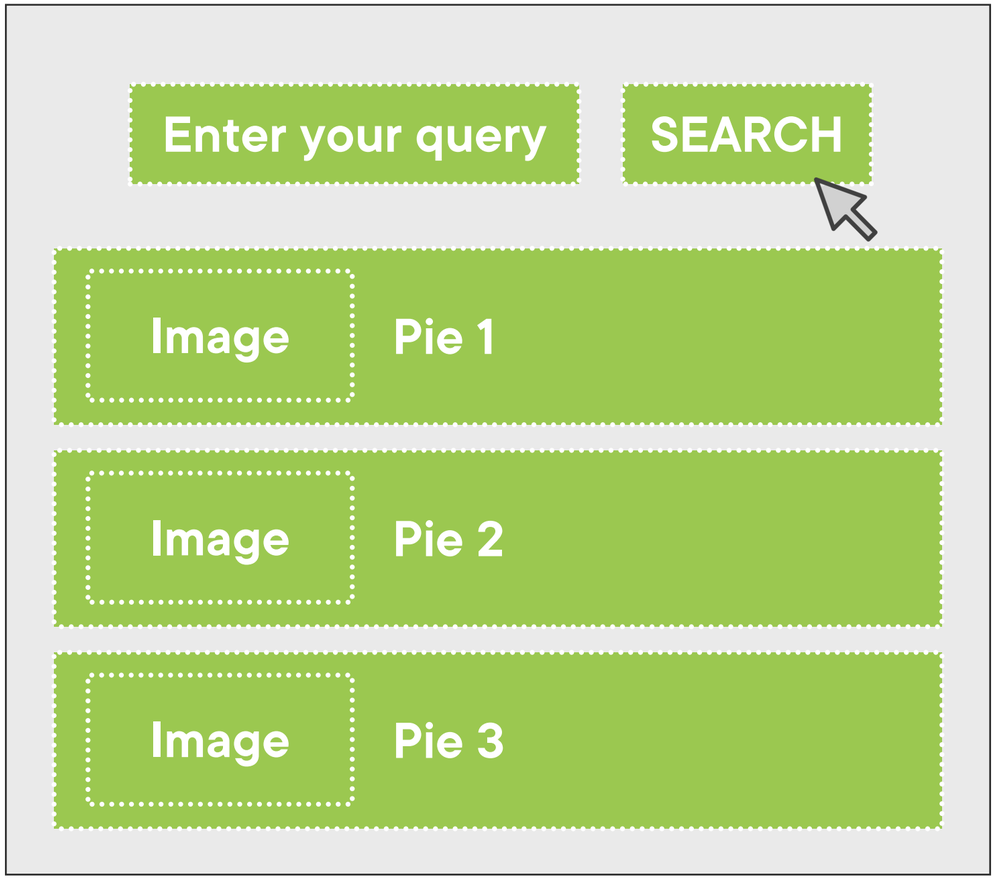
- Creating an ASP.NET Core API
- Adding jQuery

Introducing ASP.NET Core Blazor

Creating the Search Page with Blazor

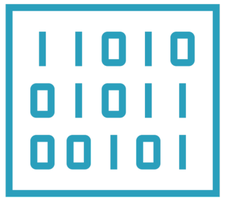
Searching Using JavaScript and an ASP.NET Core API

Creating the Search Page

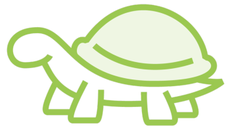


Disadvantages of this Approach

Full page needs to refresh



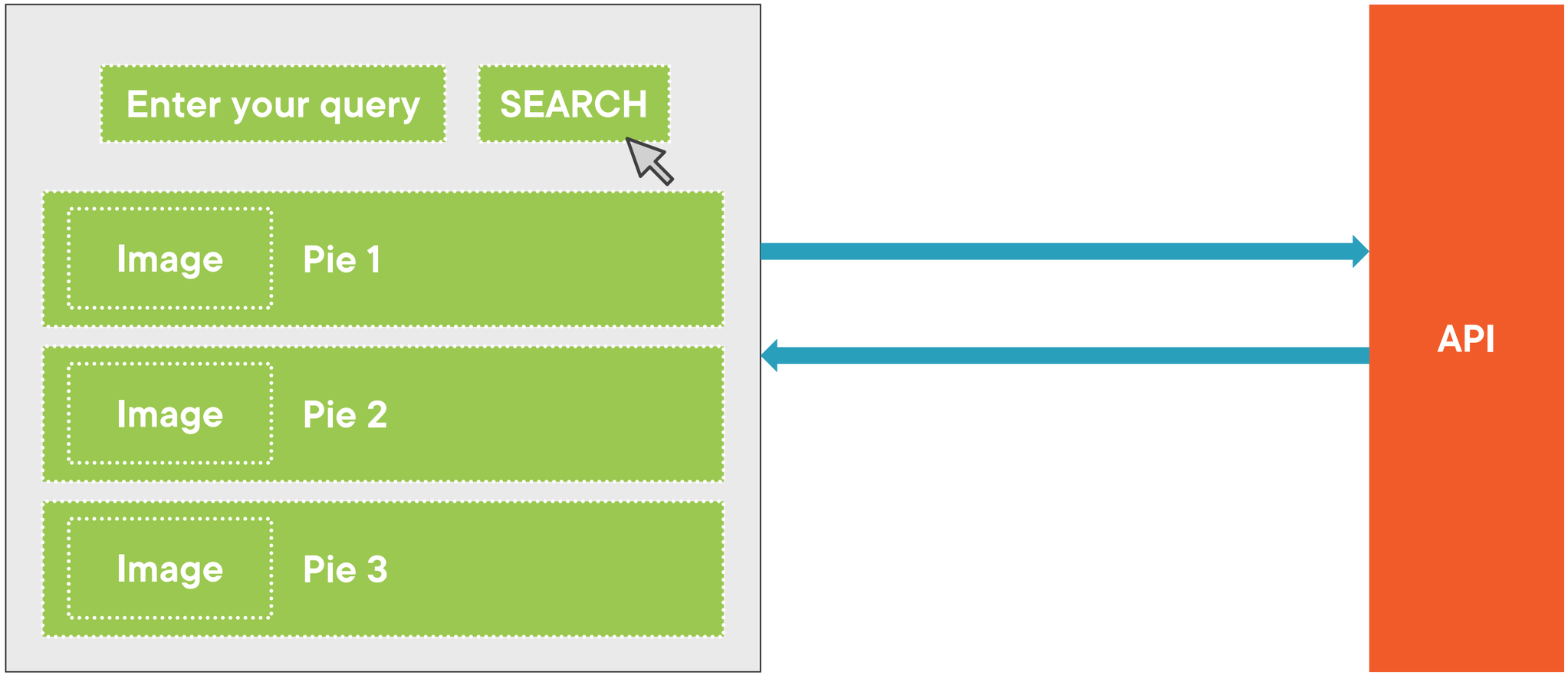
More data over the wire



Slower

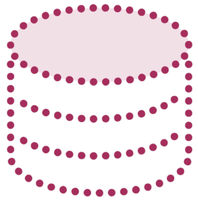
“Data” is not accessible for third-party

Updating Parts of the Page



Creating an ASP.NET Core RESTful API

Creating an API



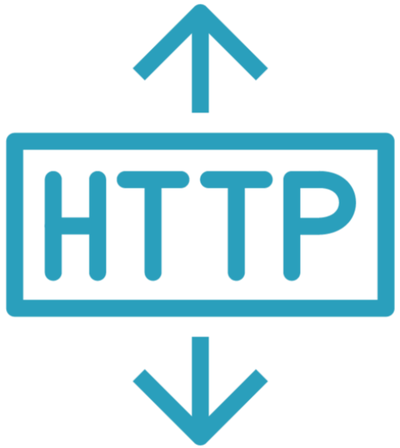
Uses “just” the data

JSON or XML

Open for many types of clients

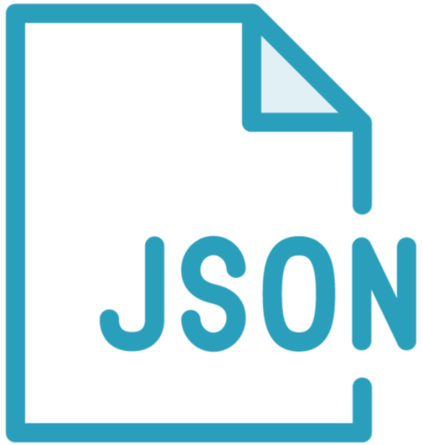
Can also be built using ASP.NET Core and MVC approach

Creating a RESTful API



HTTP request GET, POST, PUT...

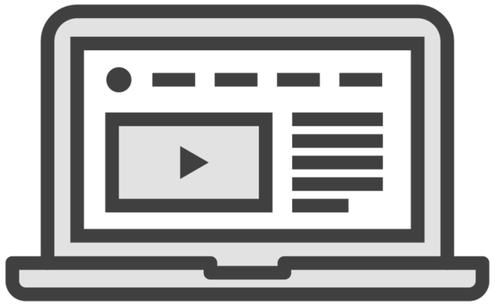
Resources with URLs



Responses in JSON

Status codes 200, 404...

Creating a RESTful API



GET <url-to-resource>

200 {json}

API

HTTP Verbs

GET

POST

PUT

DELETE

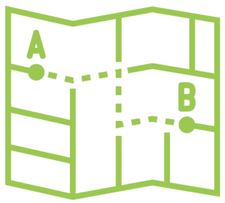
JSON Response


```
[
    {
        "pieId": 1,
        "name": "Apple Pie",
        "shortDescription": "Our famous apple pies!",
        "longDescription": "",
        "allergyInformation": "",
        "price": 12.95,
        "imageUrl": "files/applepie.jpg",
        "imageThumbnailUrl": "files/applepiesmall.jpg",
        "isPieOfTheWeek": true,
        "inStock": true,
        "categoryId": 1,
        "category": {
            "categoryId": 1,
            "categoryName": "Fruit pies",
            "description": null,
            "pies": [
                null
            ]
        }
    },
    {
        "pieId": 2,
        "name": "Blueberry Cheese Cake",
        "shortDescription": "You'll love it!",
        "longDescription": "Icing",
        "allergyInformation": "",
        "price": 18.95,
        "imageUrl": "files/blueberrycheesecake.jpg",
        "imageThumbnailUrl": "files/blueberrycheesecakesmall.jpg",
        "isPieOfTheWeek": false,
        "inStock": true,
        "categoryId": 2,
        "category": {
            "categoryId": 2,
            "categoryName": "Cheese cakes",
            "description": null,
            "pies": [
                null
            ]
        }
    },
```

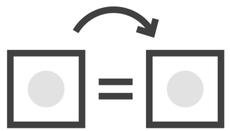
Creating an API with ASP.NET Core

Controller that returns data

JsonResult



Attribute-based routing



Other concepts are identical

```
builder.Services.AddControllers();

app.MapControllers();
```

Configuring the Application

Program.cs
Not needed separately if already done for regular MVC

```
public class PieController : ControllerBase
{

}
```

Creating a Controller

ControllerBase adds support for access to HttpContext, Request...

Routing Options in ASP.NET Core

Convention-based routing

Attribute-based routing

```
[Route("api/[controller]")]
public class PieController : ControllerBase
{

}
```

Using the Route Attribute

Accessible via /api/search

Reaching the Action Methods

```
[Route("api/[controller]")]
public class PieController : ControllerBase
{
    private readonly IPieRepository _pieRepository;

    [HttpGet]
    public IActionResult GetAll()
    {
        ...
    }
}
```

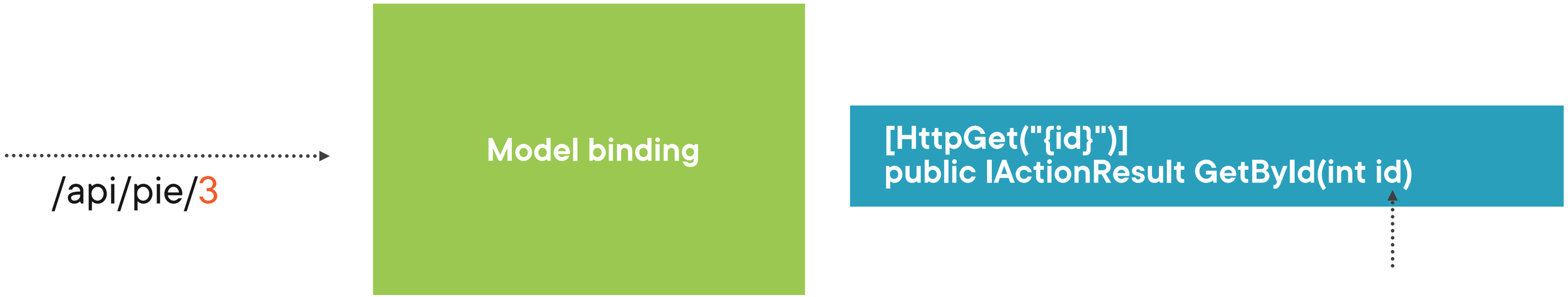
```
[Route("api/[controller]")]
public class PieController : ControllerBase
{
    private readonly IPieRepository _pieRepository;

    [HttpGet("{id}")]
    public IActionResult GetById(int id)
    {
        ...
    }
}
```

Passing a Parameter

Uses model binding again
Can work with complex types too

Routing to an API Action Method



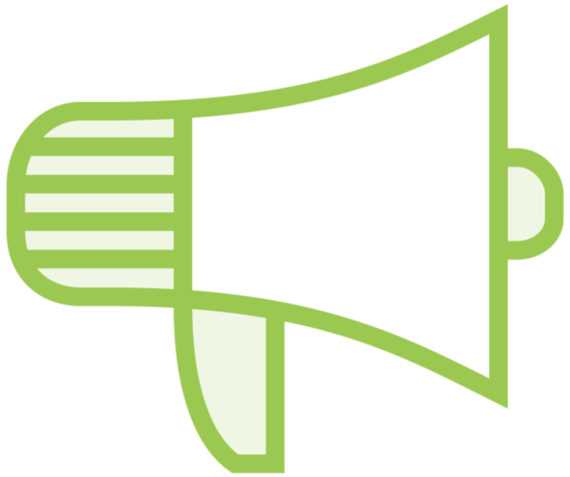
Demo

Creating an API for searching pies

The API Response

Data (often JSON)

Status code



Returning data

- Single instance or list
- Will be serialized into JSON

Helper methods defined on ControllerBase

Action Result Methods on ControllerBase

Ok()

BadRequest()

NotFound()

NoContent()

Returning a 200 Response

```
[Route("api/[controller]")]
public class PieController : ControllerBase
{
    private readonly IPieRepository _pieRepository;

    [HttpGet]
    public IActionResult GetAll()
    {
        return Ok(_pieRepository.AllPies);
    }
}
```

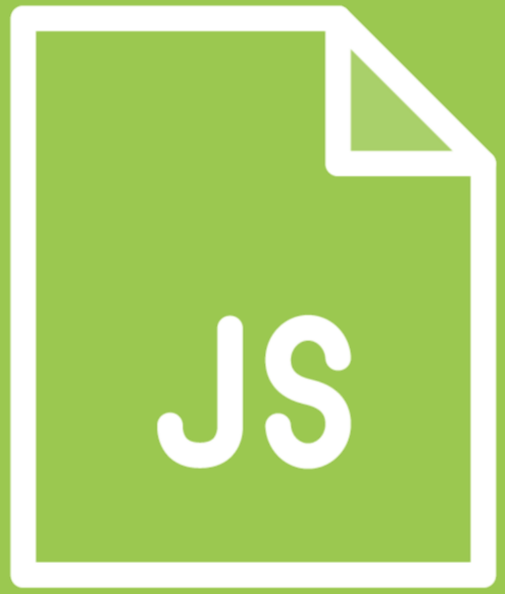
Returning a NotFound Response

```
[HttpGet("{id}")]
public IActionResult GetById(int id)
{
    if(!_pieRepository.AllPies.Any(p =>p.PieId == id))
        return NotFound();
    ...
}
```

Demo

Completing the API

Adding jQuery and Ajax

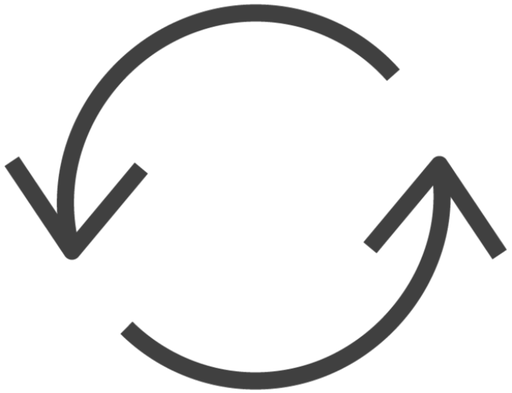


This is not a JavaScript course!

We'll look at a basic use case of invoking our API using client-side script code.

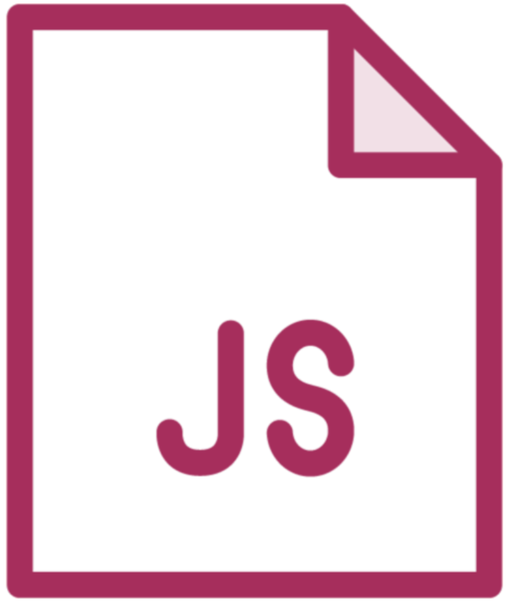
Using Ajax

Partial page



Load and send data separately

Background



Using jQuery

- Commonly used JavaScript library
- Simplifies JavaScript development
- Easy to find elements, handle events and perform Ajax calls
- Open-source

```
$(document).ready(function() {
    console.log("Welcome to Bethany");
});
```

The Document Ready

Multiple ways exist to hook into this

Performing an Ajax Call

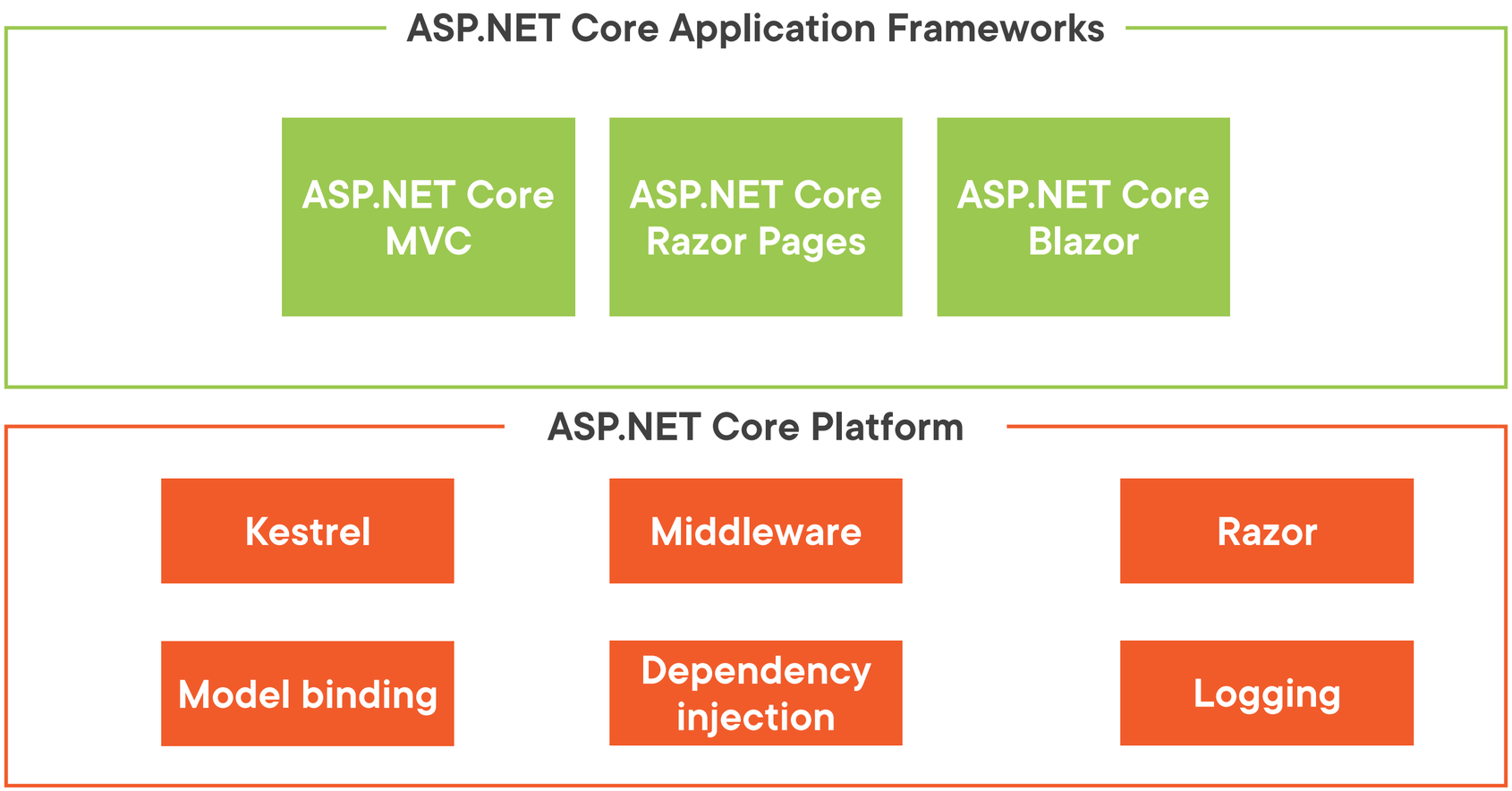
```
$.ajax('/url/to/api',
{
    dataType: 'json',
    success: function (data) {
        ...
    },
    error: function (jqXhr, status, error) {
        ...
    }
});
```

Demo

Creating the search page with Ajax and the API

Introducing ASP.NET Core Blazor

ASP.NET Core Application Frameworks



Blazor is a framework
to build interactive web UIs
using C# and HTML.
It's part of ASP.NET Core.

Introducing ASP.NET Core Blazor

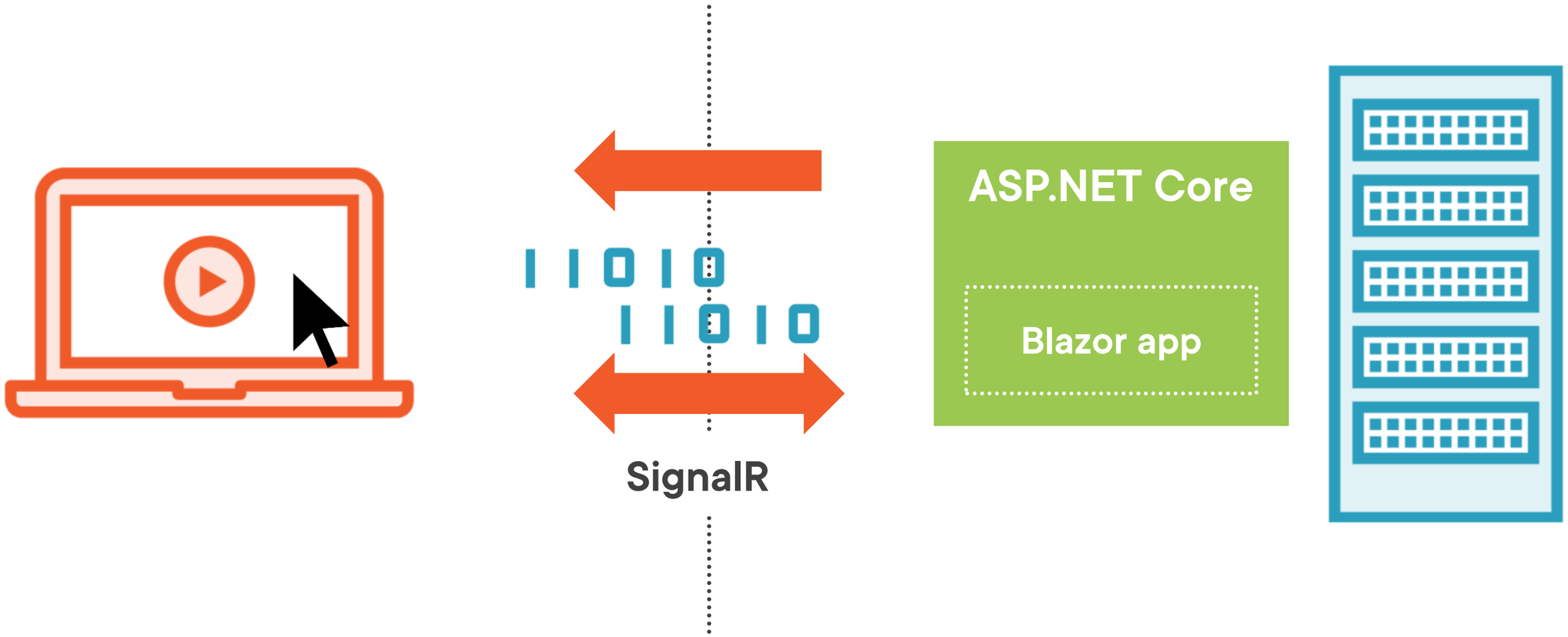
Based on WebAssembly or run on server

No plugin, based on web standards

Integrate with JavaScript

Benefits of Visual Studio and .NET including performance and libraries

How Blazor Server Works



```
builder.Services.AddServerSideBlazor();

app.MapBlazorHub();
app.MapFallbackToPage("/_Host");
```

Adding Blazor to Our Application

Blazor can co-exist with other ASP.NET Core technologies in the same project

Blazor Is Component-based

A 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

Using Code

Mixed approach using @code

“Code behind” using partial

```
public partial class PieOverview
{

}
```

Using Partial Classes

Demo

Exploring a Blazor project

Creating a first Blazor component

Creating the Search Page with Blazor

Demo

Creating the search page using Blazor

Summary

ASP.NET Core can be used to build RESTful APIs

Using Blazor, ASP.NET Core can also be used to include interactivity in our pages

Up next:
Adding authentication and authorization to the site