Blazor를 사용한 풀스택 웹개발

.NET

박용준(RedPlus)
Microsoft MVP 2019-2020
www.devlec.com
https://www.dotnetkorea.com

- 블레이저는 풀스택 웹 프레임워크입니다.

블레이저는 웹 + SPA를 위한

하나의 완성된 풀스택 웹 프레임워크입니다.

앞으로 가장 인기있는? 웹 프레임워크가 될 것입니다.

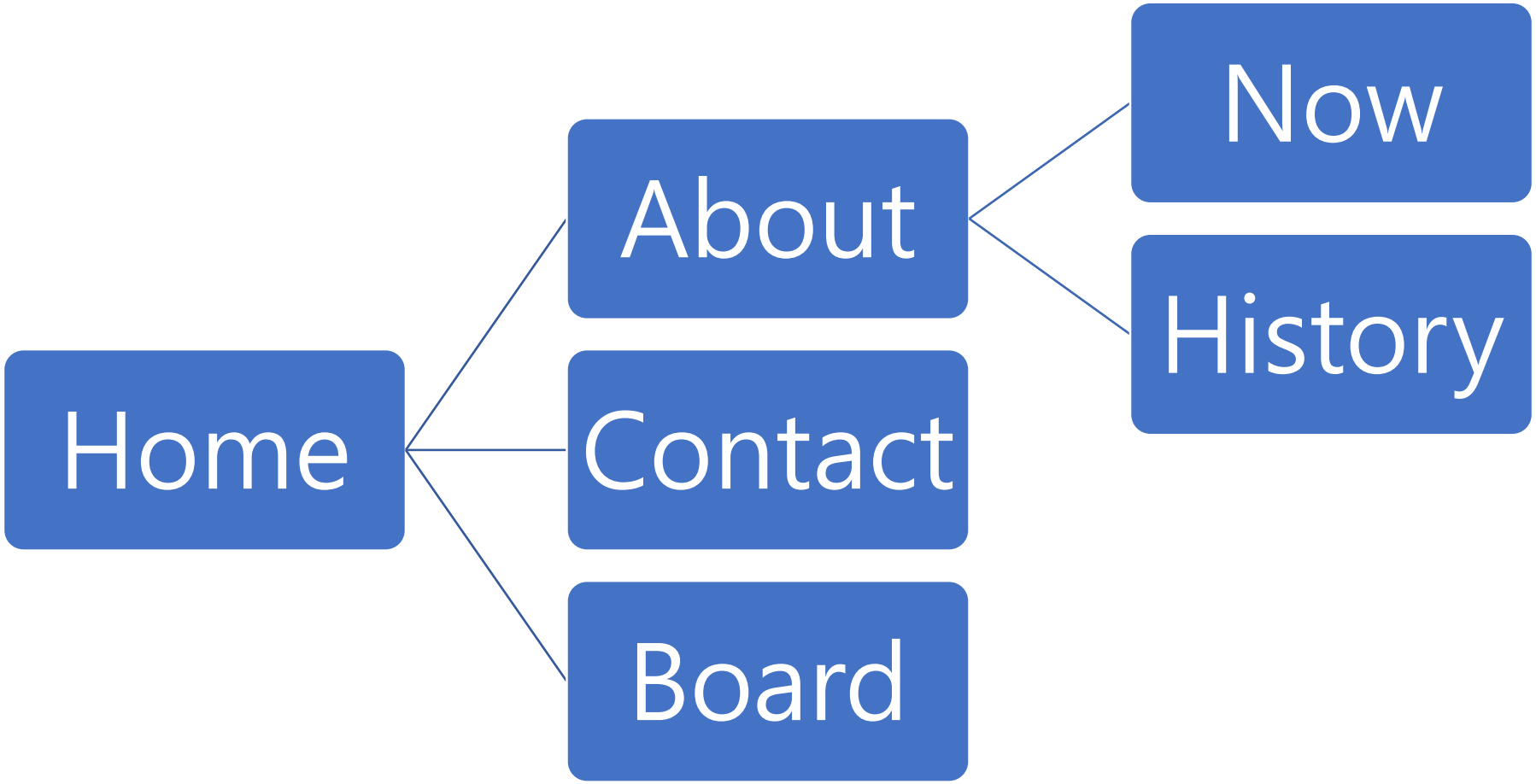
블레이저는

서버측 기능과 클라이언트측 프레임워크 기능을 모두 가집니다.

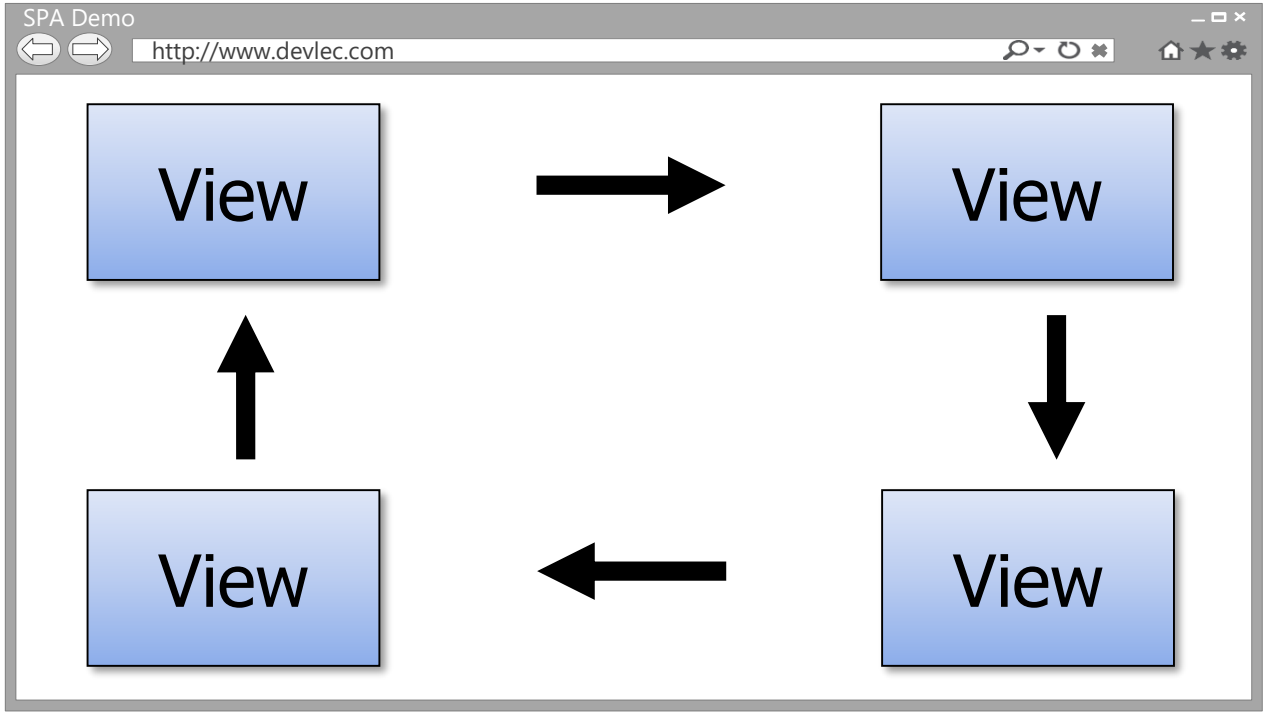
Angular, React, Vuejs, jQuery 같은 자바스크립트가 아닙니다.

C#을 사용한 풀스택 웹개발 프레임워크입니다.

Multi Page Application에서 SPA로 이동

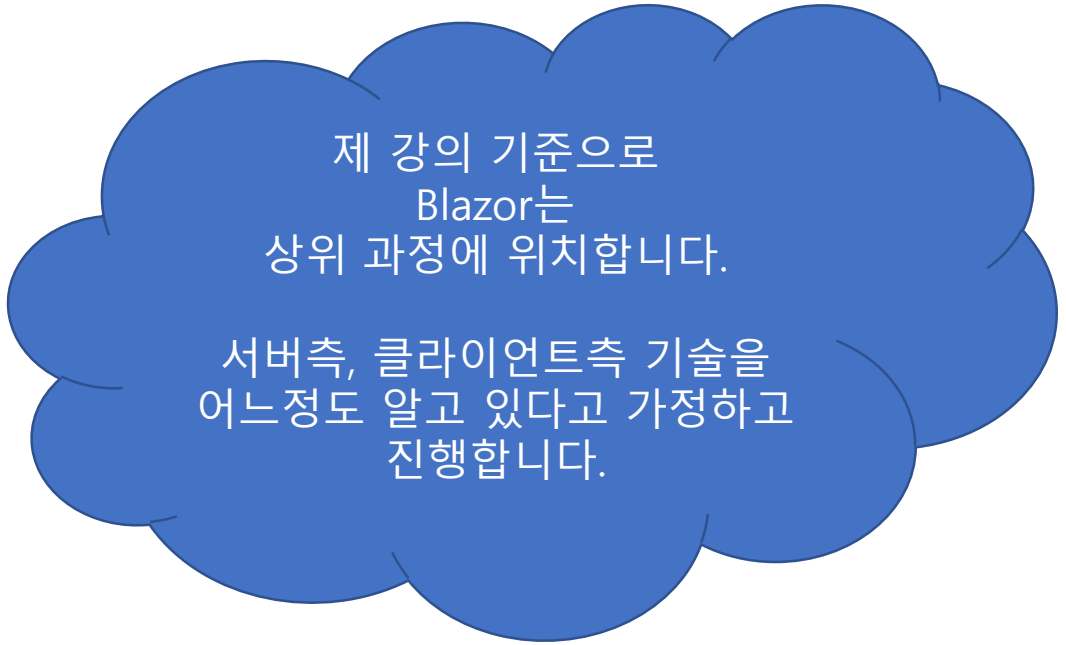


Single Page Application (SPA)



선수 학습(박용준 강사 책/강의 기준)

- 필수
  - C#과 Web 기술
- 선택
  - Web
    - HTML5
    - CSS3
    - JavaScript
    - Bootstrap
    - jQuery
  - C#
    - C# 기초 입문
  - SQL Server 기초
    - Entity Framework Core
  - ASP.NET 기초 입문
    - ASP.NET & Core를 다루는 기술



이 강의는 친절한 버전의 강의가 아닙니다. 이미 위와 같은 많은 기술을 알고 있다고 가정하고 진행합니다.

이 과정을 통해서 우리가 배울 내용

- 블레이저 기본
  - 컴포넌트
  - 속성 바인딩과 이벤트 바인딩
  - 서비스와 의존성 주입
  - 폼
  - 라우팅
  - HTTP
  - 하나의 완성된 웹앱을 만드는 모든 기술

클라이언트측과 서버측

Blazor

(Angular, React, Vuejs, ...)

ASP.NET

(Spring Boot, PHP, Node, Python...)

블레이저와 어울리는 서버 측 기술들

ASP.NET Core

SQL Server

Node.js

MongoDB

ASP.NET Core Web API

- 최고의 RESTful 서비스 구축
- JSON
  - XML

Blazor(블레이저) 개발 환경 구축

- Visual Studio 2019 설치
  - Git 설치
  - Visual Studio Code 설치

강의 소스

- https://github.com/VisualAcademy

[참고] 표시 강좌

- [참고] 표시된 강좌는 반드시 볼 필요 없습니다.
  - 참고용으로 보시면 됩니다.

- [구강좌]
  - 최근 새롭게 방법이 변경된 부분 이전에 촬영된 강좌입니다.

주요 슬라이드 출처

- 대부분 직접 작성
- Channel9의 주요 이벤트에서 참고
  - BUILD
  - Ignite
  - 기타

- .NET Foundation
- Microsoft GitHub

강의 시작합니다.

ASP.NET Core 3.0의 특징 3가지

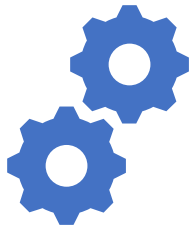
박용준:
Microsoft MVP
데브렉(www.devlec.com)

ASP.NET Core 3.0의 새로운 특징 3가지

SPA + Identity

Angular/React/Blazor에

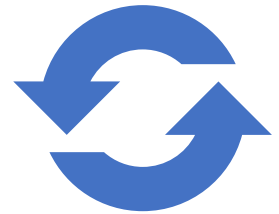
인증 기능 포함



Worker Service

장시간 실행되는

백그라운드 서비스 구현



gRPC

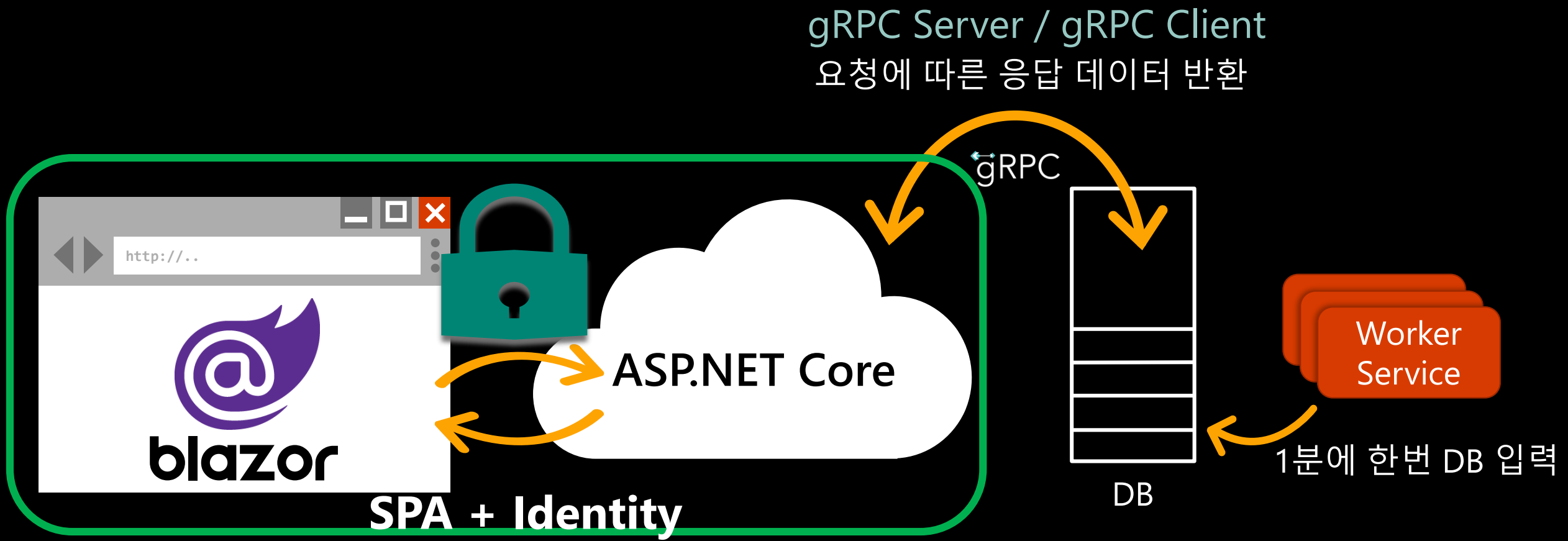
고성능의 계약 기반

RPC 프레임워크

인증(Authentication)이 기본으로 포함된 SPA 템플릿

- IdentityServer를 사용한 인증 기능 포함
  - Angular
  - Blazor
  - React

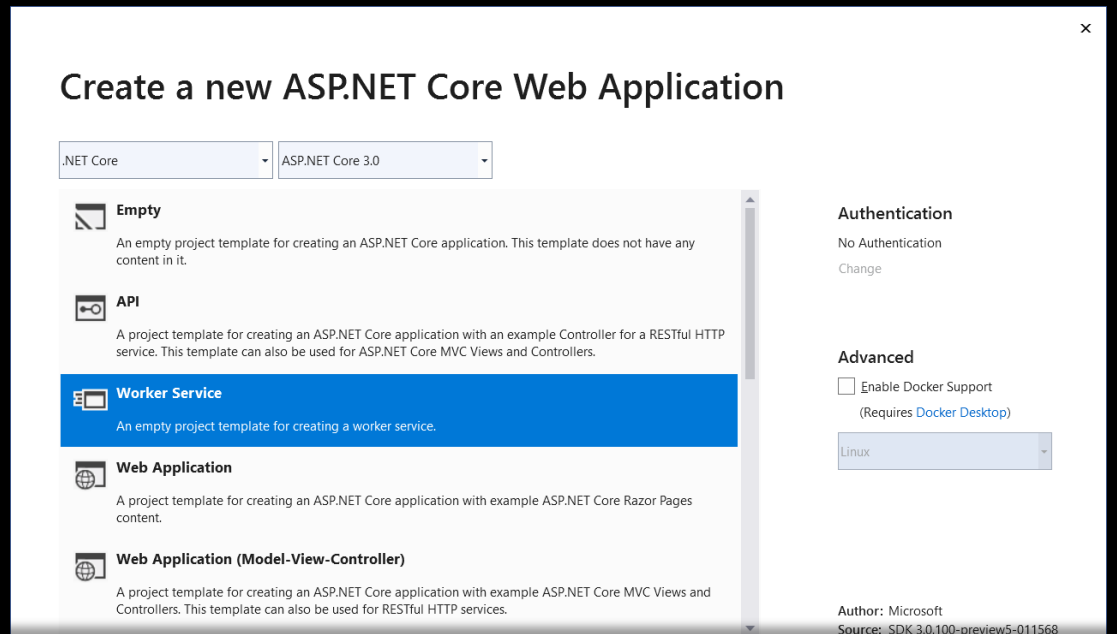
ASP.NET Core 주요 특징을 사용한 리스트 앱 아키텍처



데모 소스: https://github.com/VisualAcademy/Features

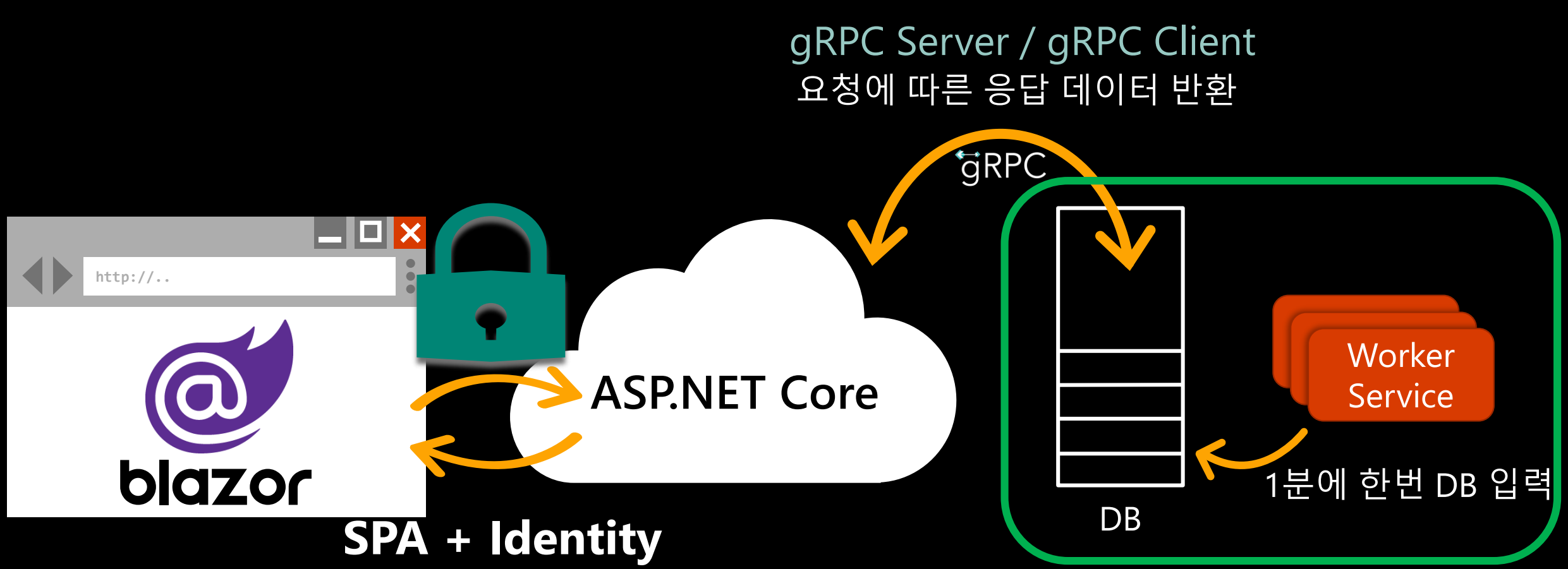
워커 서비스(Worker Service)

- 장시간 실행되는 워커 프로세스 개발
  - 시뮬레이션, 민감도 분석, ...
- Windows Service, systemd, WebJobs 등에 호스팅
  - Windows Service
    - Windows 이벤트 뷰어
  - Linux Daemon
- 설정, 로깅, 종속성 주입(DI) 통합
  - 기존 경험 그대로 서비스 프로그램 생성




```
public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;

    public Worker(ILogger<Worker> logger)
    {
        _logger = logger;
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

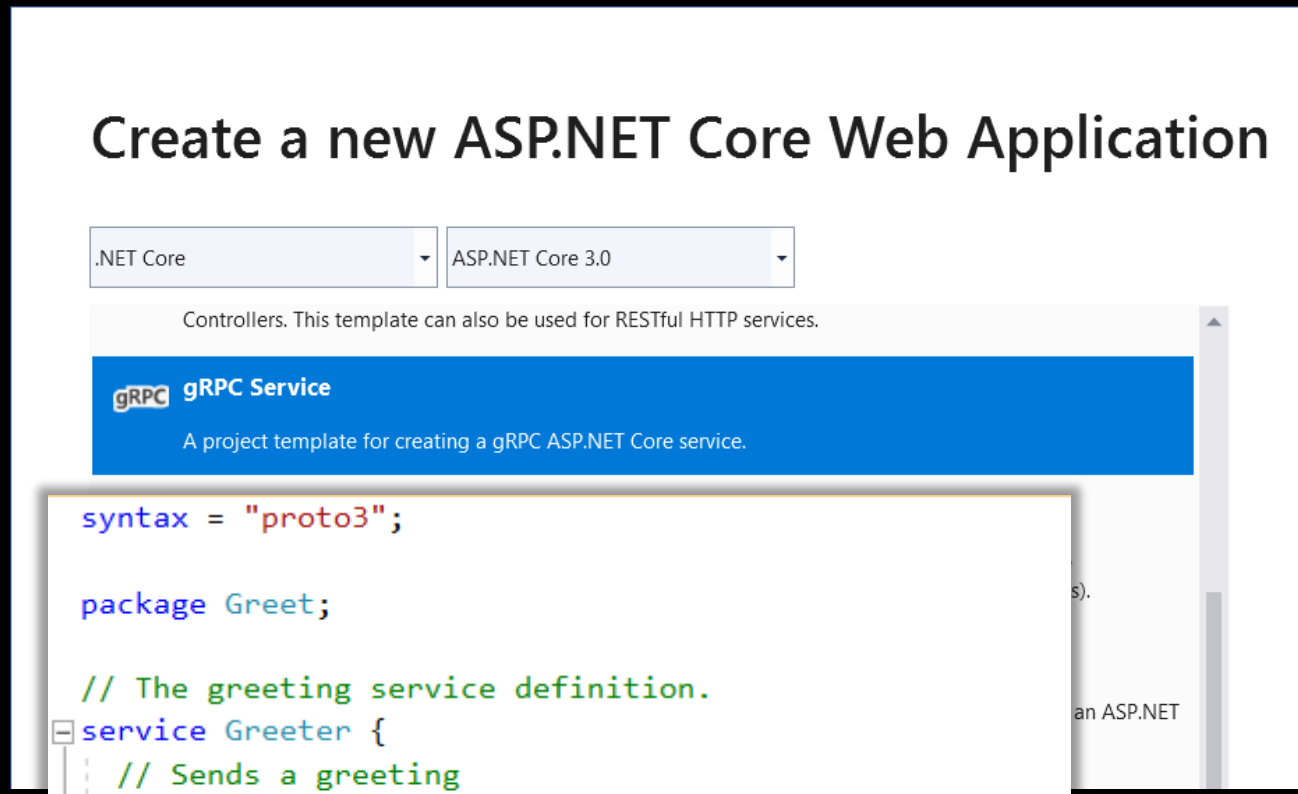
ASP.NET Core 주요 특징을 사용한 리스트 앱 아키텍처



데모 소스: https://github.com/VisualAcademy/Features

gRPC

- 언어 제약이 없는 오픈소스 고성능/경량 계약(Contract) 기반 RPC(원격 프로시저 호출) 프레임워크
  - gRPC(gRPC Remote Procedure Calls)
  - C#, Go, Python, Java, ...
  - HTTP/2
  - 프로토콜 버퍼(Protocol Buffers)
  - https://grpc.io/docs/

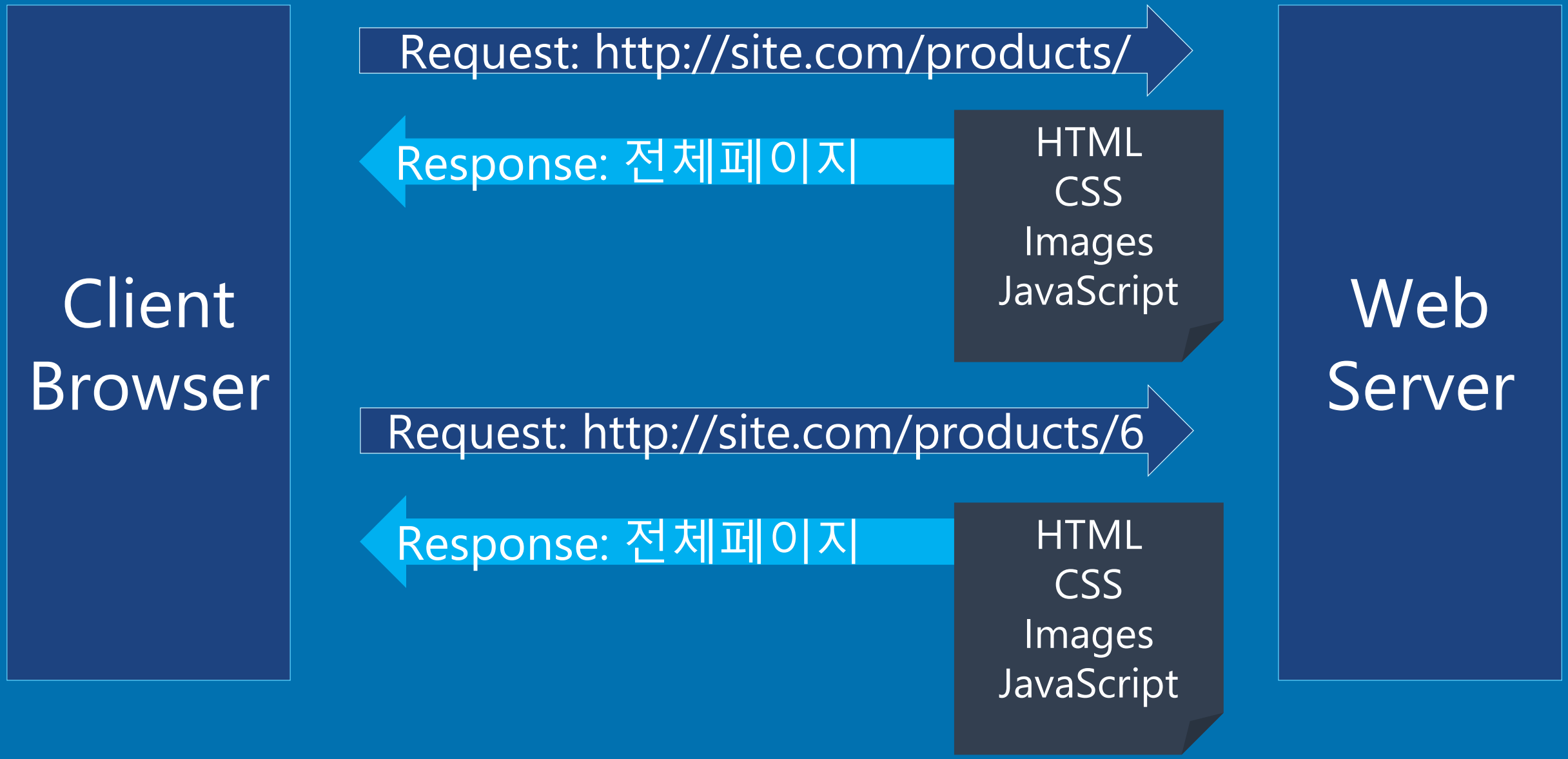



```
syntax = "proto3";

package Greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
```

```
public class GreeterService : Greeter.GreeterBase
{
    3 references
    public override Task<HelloReply> SayHello(HelloRequest request,
      ServerCallContext context)
    {
        return Task.FromResult(new HelloReply
        {
            Message = "Hello " + request.Name
        });
    }
}
```

단일 페이지 응용프로그램 (Single Page Applications) 역사 살펴보기

비교: Multiple Page Application



비교: Single Page Application

Single Page Applications

Frameworks

```
<!doctype html>
 <html ng-app="app">
 <body>
     <tabs>
         <pane title="Localization">
             Date: {{ '2013-04-01' | date:'fullDate' }} <br>
             Currency: {{ 123456 | currency }} <br>
             Number: {{ 98765.4321 | number }} <br>
         </pane>
         <pane title="Pluralization">
             <div ng-controller="BeerCounter">
                 <div ng-repeat="beerCount in beers">
                     <ng-pluralize
                          count="beerCount“
                          when="beerForms">
                     </ng-pluralize>
                 </div>
             </div>
         </pane>
     </tabs>
 </body>
</html>
```

AngularJS

- Custom Tag Attributes
- Powerful Directives
- Declarative Style
- Strict Separation

```
var HelloMessage = React.createClass({
    render: function() {
        return <div>Hello {this.props.name}</div>;
    }
});

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

React

React

Not MVC – Just V (UI)

Virtual DOM

One-way data flow

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

Vue.js

Approachable

Versatile

Performant

Knockout.js

Declarative Bindings

Automatic UI refresh

Dependency Tracking

Templating

```
Choose a ticket class:
<select data-bind="options: tickets,
                   optionsCaption: 'Choose...',
                   optionsText: 'name',
                   value: chosenTicket"></select>

<button data-bind="enable: chosenTicket,
                   click: resetTicket">Clear</button>

<p data-bind="with: chosenTicket">
    You have chosen <b data-bind="text: name"></b>
    ($<span data-bind="text: price"></span>)
</p>
```

Binding attributes declaratively link DOM elements with model properties

```
<script>
    function TicketsViewModel() {
        this.tickets = [
            { name: "Economy", price: 199.95 },
            { name: "Business", price: 449.22 },
            { name: "First Class", price: 1199.99 }
        ];
        this.chosenTicket = ko.observable();
        this.resetTicket = function() { this.chosenTicket(null) }
    }
    ko.applyBindings(new TicketsViewModel());
</script>
```

Your ***view model*** holds the UI's underlying data and behaviors

Activates Knockout

Ember.js

Ember is a framework

Convention over config

URLs are important

```
<script type="text/x-handlebars" id="posts">
  <div class="container-fluid">
    <div class="row-fluid">
      <div class="span3">
        <table class='table'>
          <thead>
            <tr><th>Recent Posts</th></tr>
          </thead>
          {{#each}}
          <tr><td>
            {{#link-to 'post' this}}
              {{title}} <small class='muted'>by {{author.name}}</small>
            {{/link-to}}
          </td></tr>
          {{/each}}
        </table>
      </div>
      <div class="span9">
        {{outlet}}
      </div>
    </div>
  </div>
</script>
```

JavaScript Tools

Grunt

JavaScript Task Runner

Automates minification, unit testing, compilation, etc.

Configuration over code

Common tasks already available as Grunt Plugins

```
module.exports = function(grunt) {

  grunt.initConfig({
    jshint: {
      files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
      options: {
        globals: {
          jQuery: true
        }
      }
    },
    watch: {
      files: ['<%= jshint.files %>'],
      tasks: ['jshint']
    }
  });

  grunt.loadNpmTasks('grunt-contrib-jshi
  grunt.loadNpmTasks('grunt-contrib-watch'

  grunt.registerTask('default', ['jshint

};
```

```
var gulp = require('gulp');
var coffee = require('gulp-coffee');
var concat = require('gulp-concat');
var uglify = require('gulp-uglify');
var sourcemaps = require('gulp-sourcemaps');

var paths = {
  scripts: ['client/js/**/*.coffee', '!client/external/**/*.coffee'],
  images: 'client/img/**/*'
};

gulp.task('scripts', ['clean'], function() {
  // Minify and copy all JavaScript (except vendor scripts)
  // with sourcemaps all the way down
  return gulp.src(paths.scripts)
    .pipe(sourcemaps.init())
      .pipe(coffee())
      .pipe(uglify())
      .pipe(concat('all.min.js'))
    .pipe(sourcemaps.write())
    .pipe(gulp.dest('build/js'));
});

// Rerun the task when a file changes
```

Gulp

Another JS Task Runner

Code over config

Streaming build system

Tasks are executed in maximum concurrency

Visual Studio Support for Grunt / Gulp

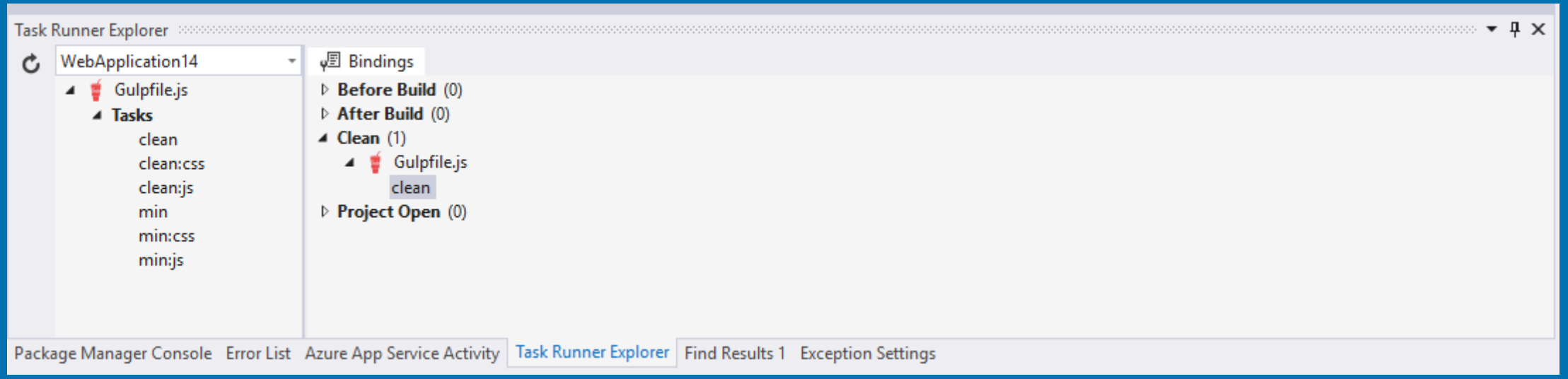
gulpfile.js | package.json

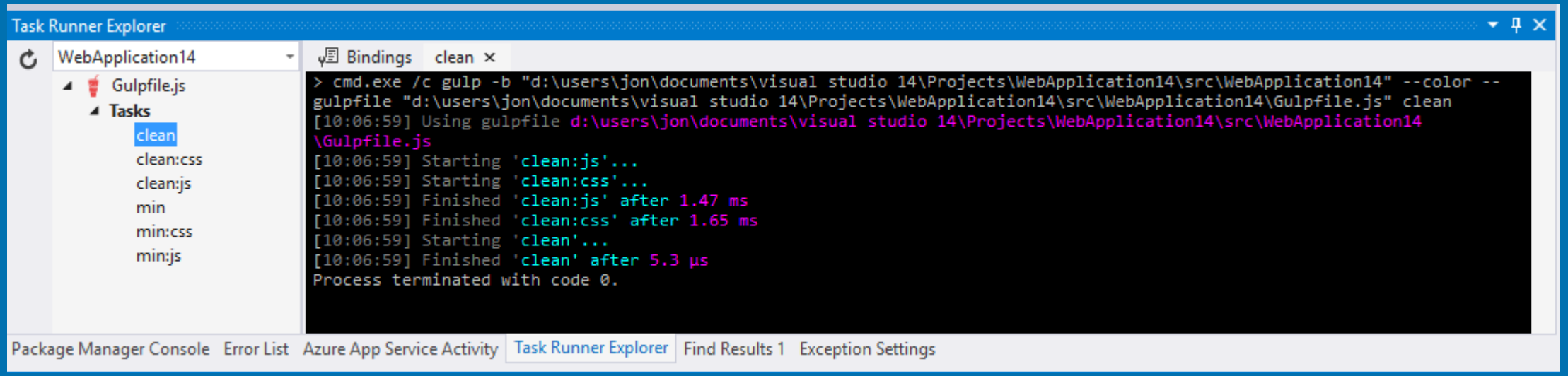
<global> | gulp

```
paths.css = paths.webroot + "css/**/*.css";
paths.minCss = paths.webroot + "css/**/*.min.css";
paths.concatJsDest = paths.webroot + "js/site.min.js";
paths.concatCssDest = paths.webroot + "css/site.min.css";

gulp.task("clean:js", function (cb) {
    rimraf(paths.concatJsDest, cb);
});

gulp.task("clean:css", function (cb) {
    rimraf(paths.concatCssDest, cb);
});

gulp.task("clean", ["clean:js", "clean:css"]);
```

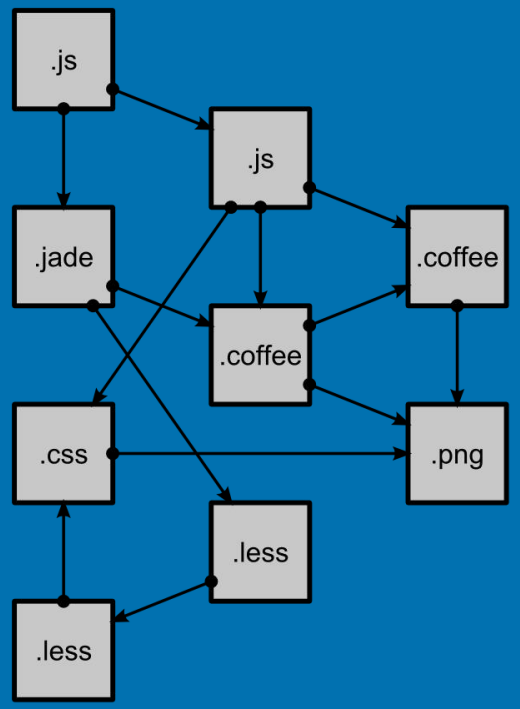

117 %

Visual Studio Support for Grunt / Gulp



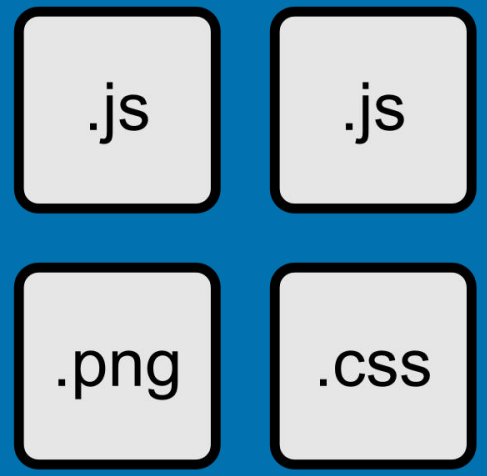


Web Pack



modules
with dependencies

webpack
MODULE BUNDLER



static
assets

Bower

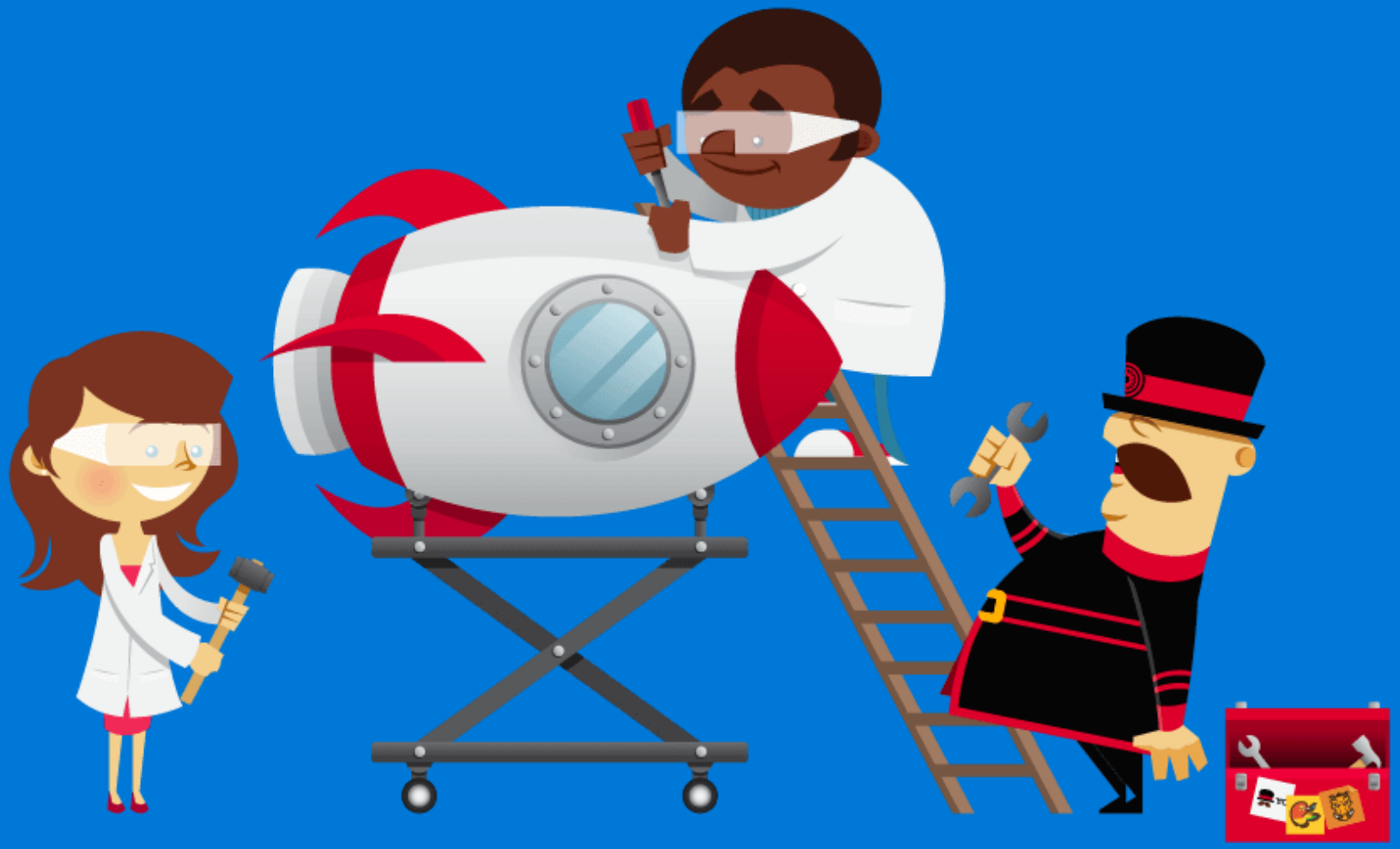
The package manager for the Web

50k existing users

>25k packages

```
# registered package
$ bower install jquery

# GitHub shorthand
$ bower install angular/angular.js

# Git endpoint
$ bower install git://github.com/twbs/bootstrap.git

# URL
$ bower install http://example.com/script.js
```

Yeoman

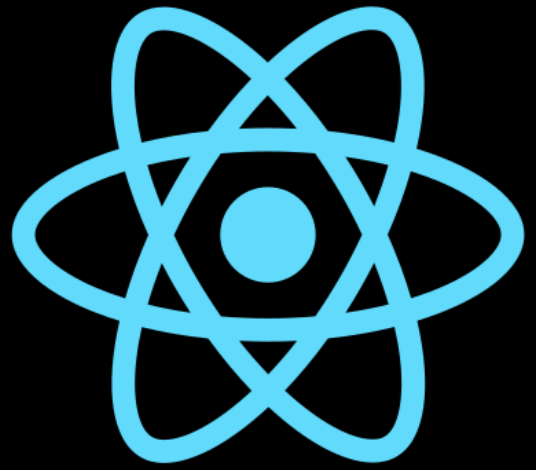


모던 - 단일 페이지 응용 프로그램(SPA)

- 살아남은(?) 단일 페이지 응용 프로그램(Single Page Application)
  - Angular
  - Blazor
  - React
  - Vue

Angular

Blazor



React

Vue

ASP.NET Core SPA 템플릿

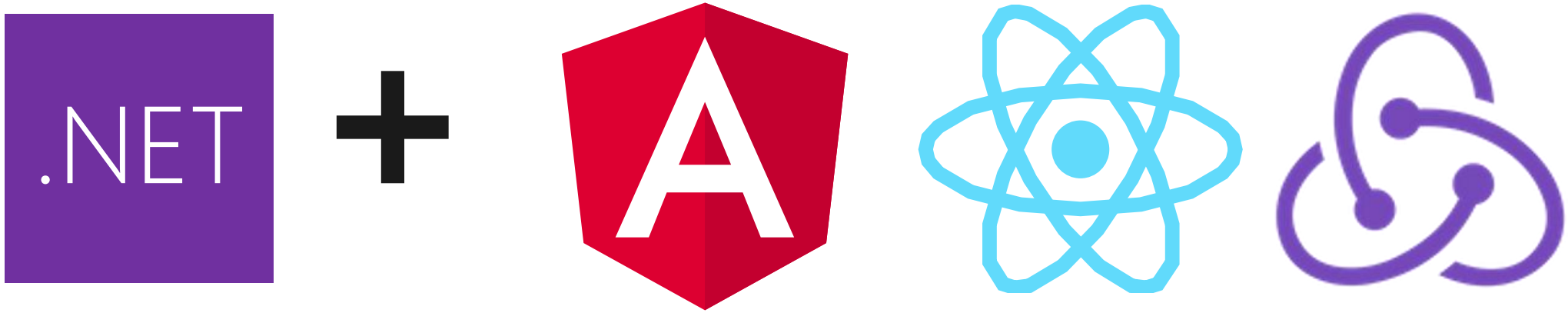
ASP.NET Core 프로젝트

Blazor/Angular/React/React+Redux

주요 기능 제공

hot module reload, source maps, unit testing, …

Angular CLI와 create-react-app

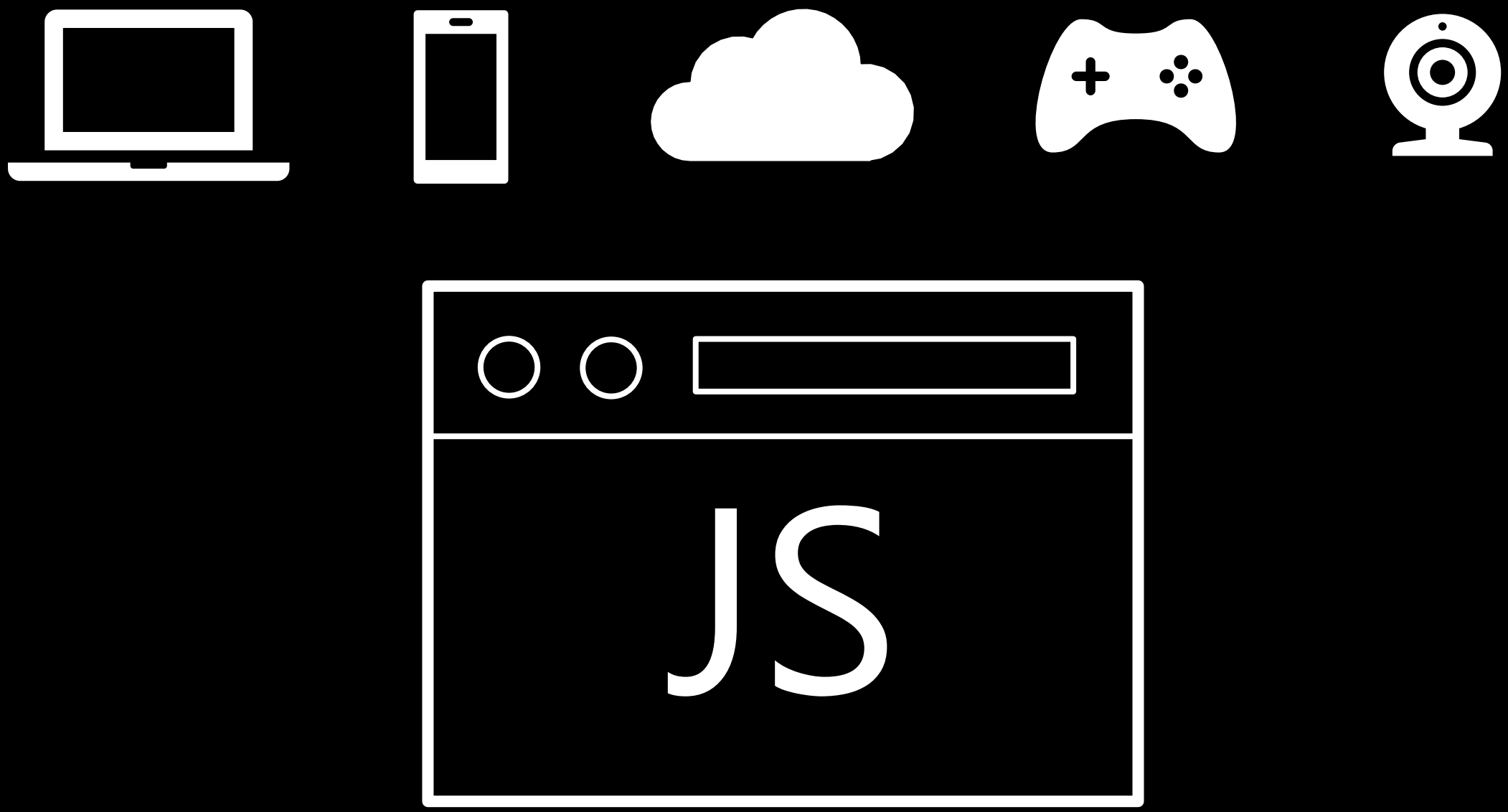


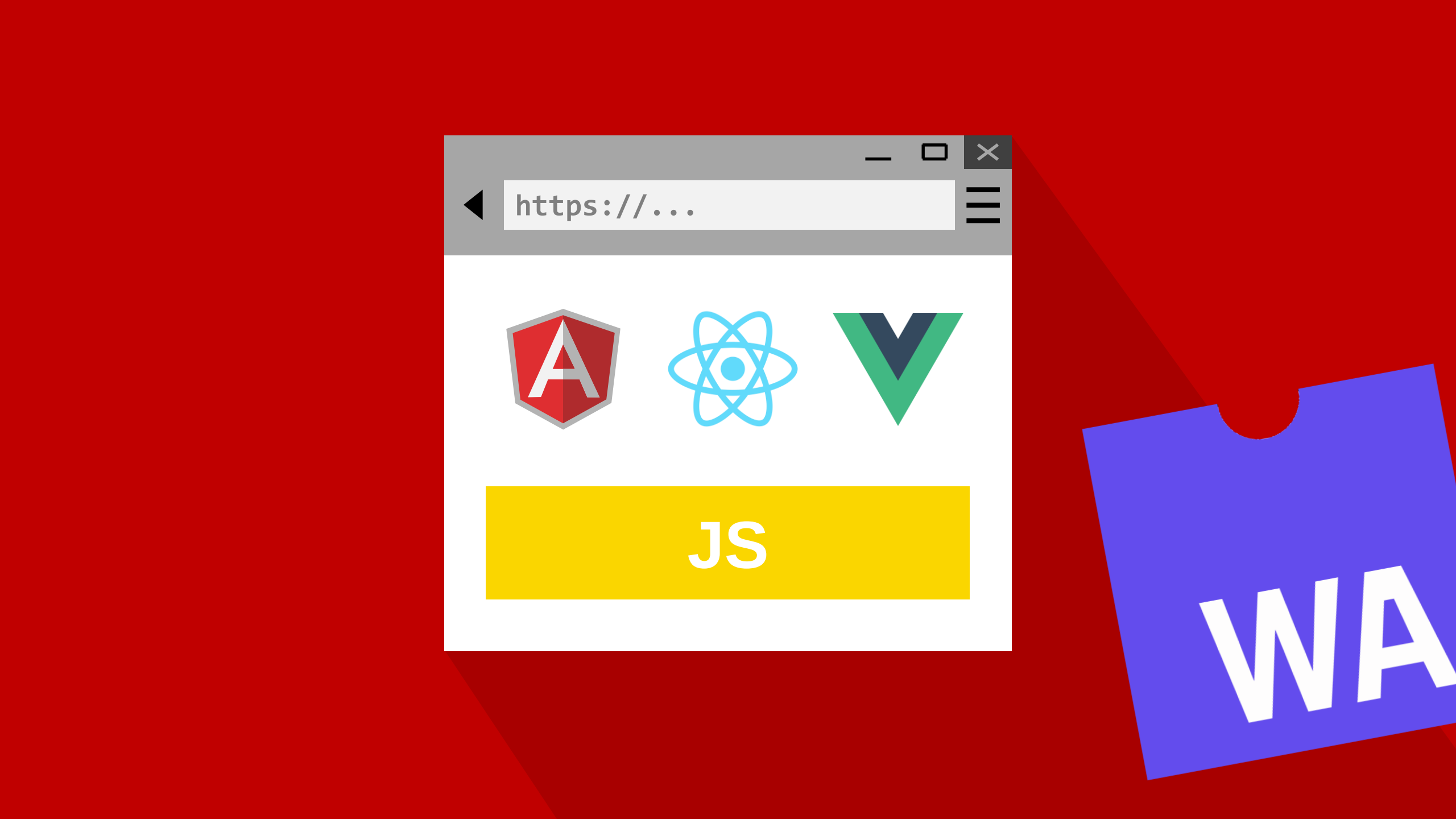
Blazor 소개

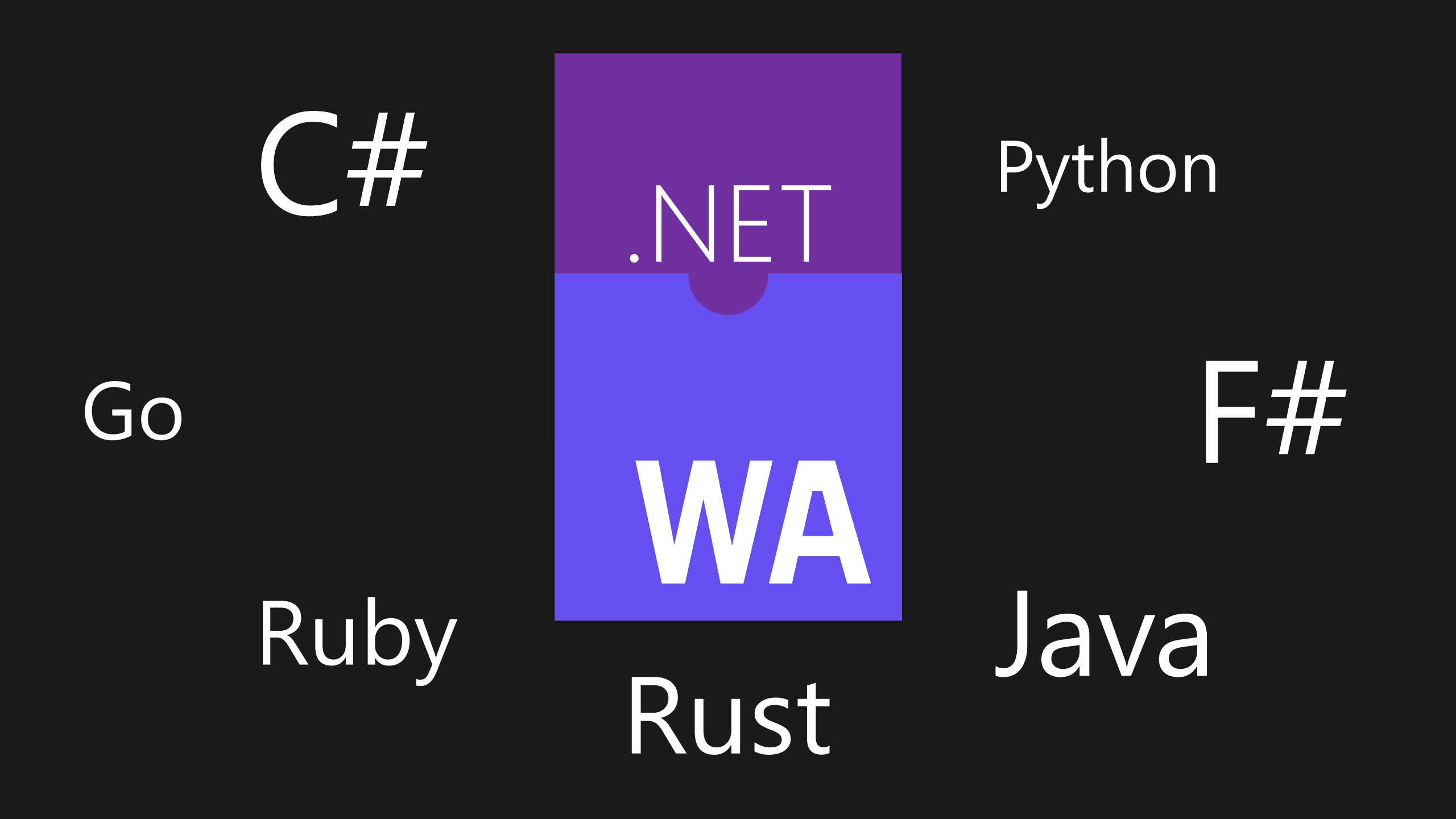
박용준(RedPlus)
Microsoft MVP 2019-2020
www.devlec.com

Blazor

https://blazor.net

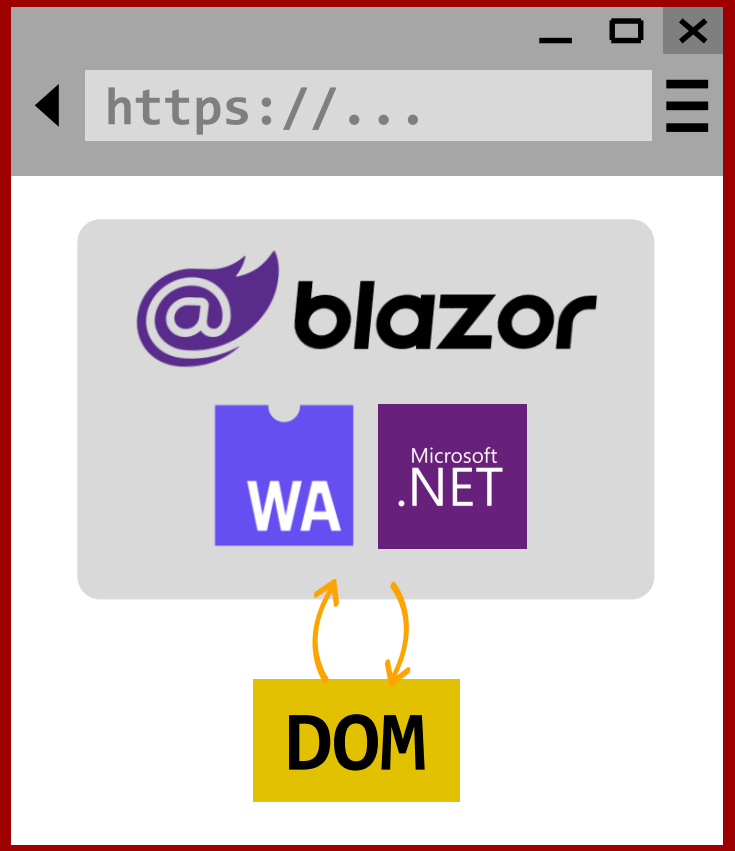

JS


https://...
JS
WA

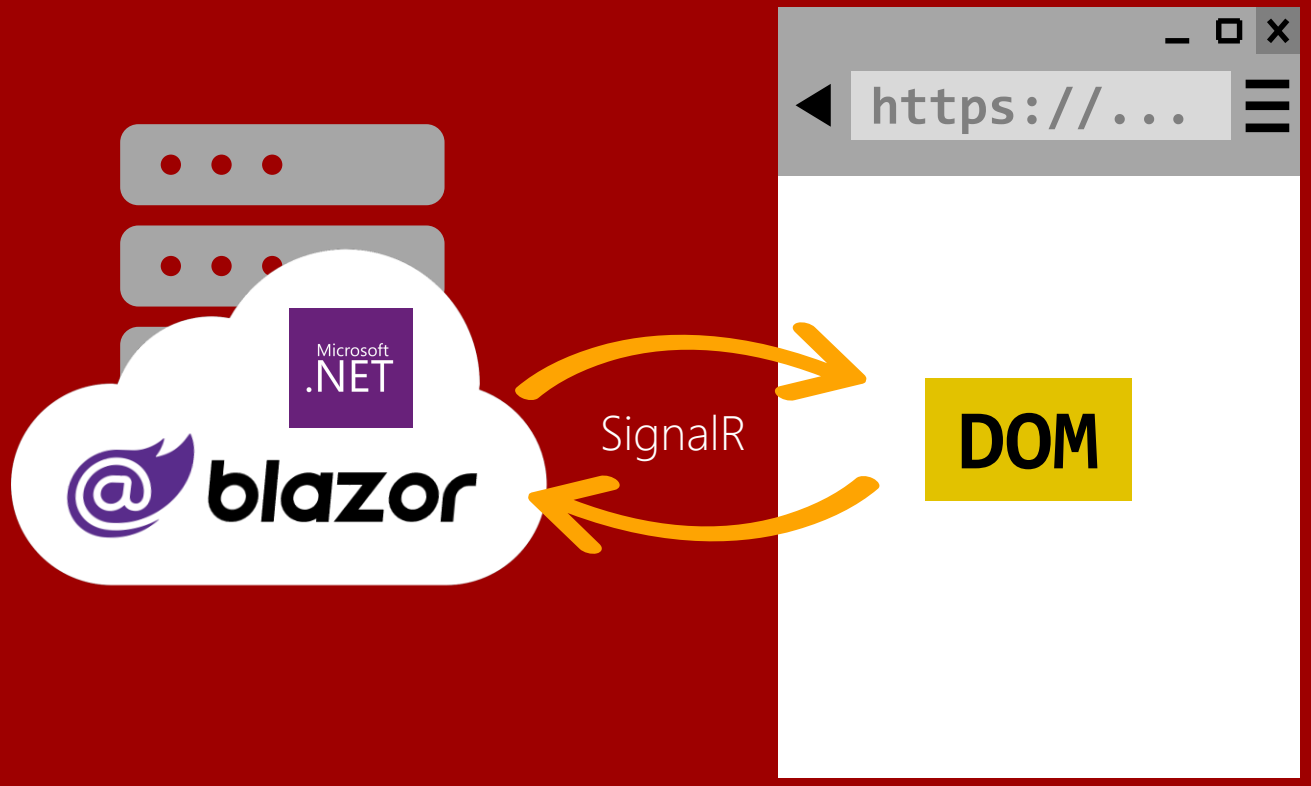

C#
.NET
Python
Go
F#
WA
Ruby
Java
Rust

Client-side

(Blazor WebAssembly)



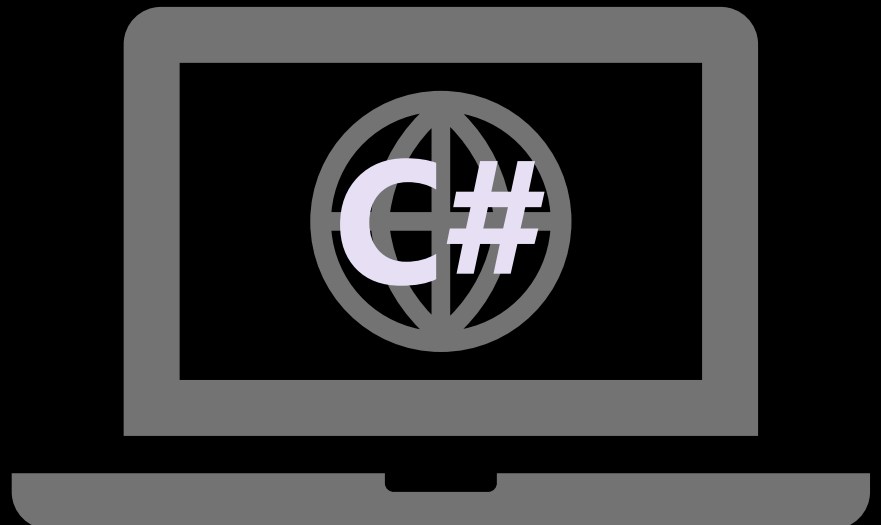
Client-side
(Blazor WebAssembly)



Server-side
(Blazor Server)

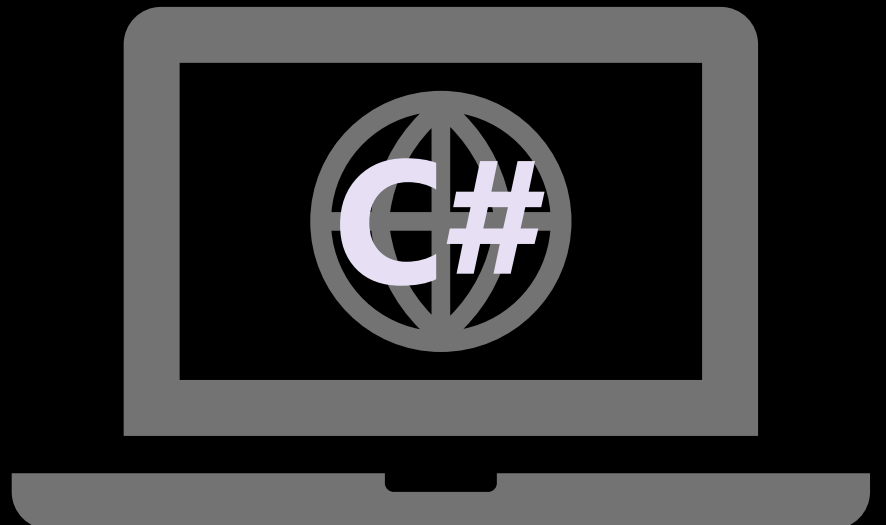
Blazor

- 닷넷을 사용하여 SPA(Single Page Application) 웹 앱을 작성하는 프레임워크
- Browser + Razor = Blazor!
- 풀스택 웹 개발 with (닷넷) 또는 (닷넷 + 웹어셈블리)
- 플러그인 및 코드 변환 필요없음
- 모바일 브라우저를 포함한 최근 웹브라우저에서 실행
  - 서버 쪽 Blazor를 사용하여 웹어셈블리를 기본 지원하지 않는 IE에서도 실행 가능



Blazor

- 자바스크립트를 대신하여 클라이언트 쪽 웹 UI 작성
- C#과 Razor를 사용하여 재 사용 가능한 UI 컴포넌트 작성
- 하나의 닷넷 코드를 클라이언트와 서버에서 공유해서 사용
- 필요하다면 자바스크립트 라이브러리 및 브라우저 API 호출 가능

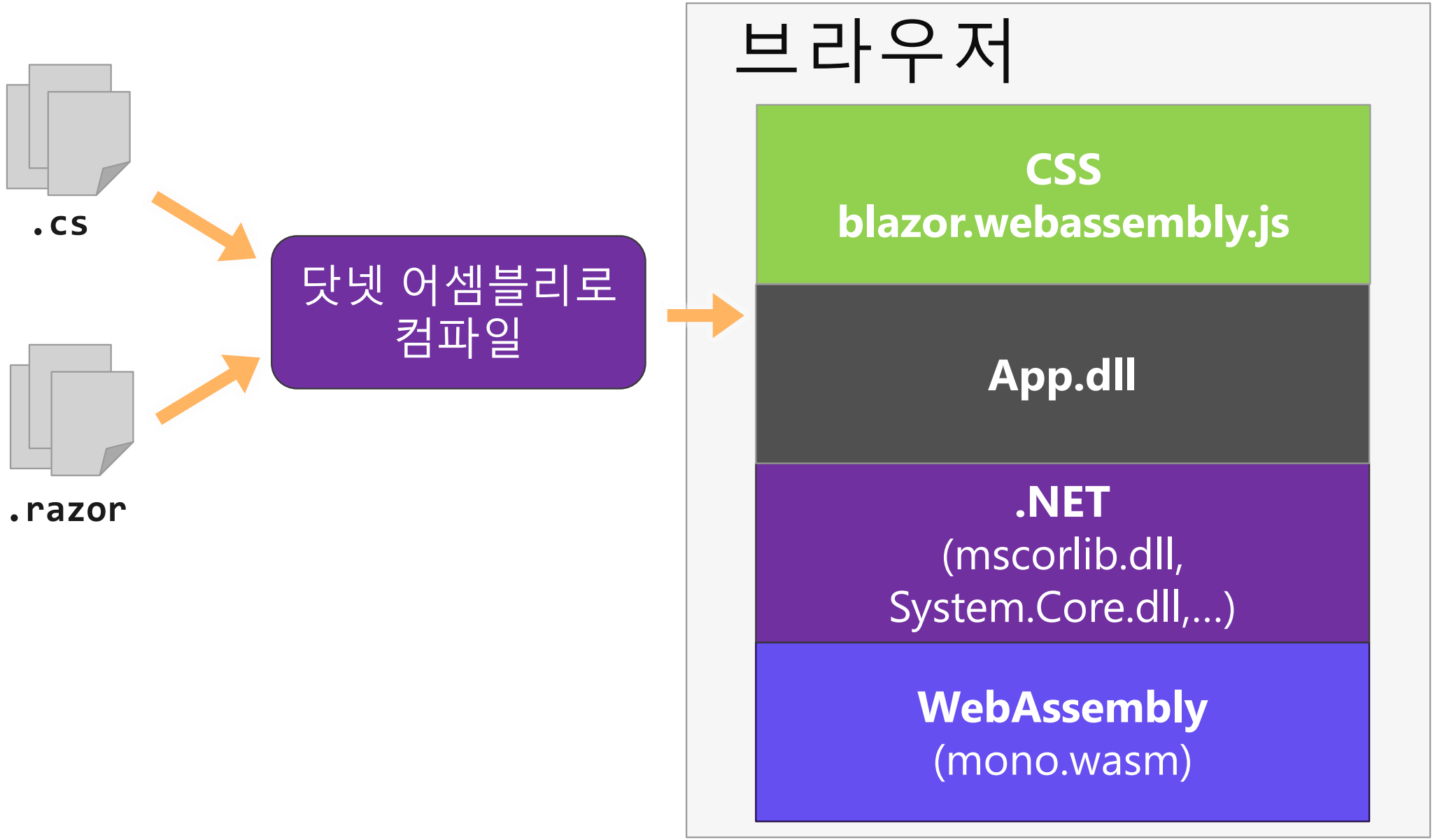


브라우저 앱을 닷넷 기술로 만드는 이유?

- **안정성, 성숙, 생산성**: .NET Standard, MSBuild
- **빠름, 확장성, 믿음성**: .NET Core로 백엔드 서비스구현
- **최신 언어**: C#, Razor의 신기술 적용
- **최고의 개발 도구**: Visual Studio, IntelliSense

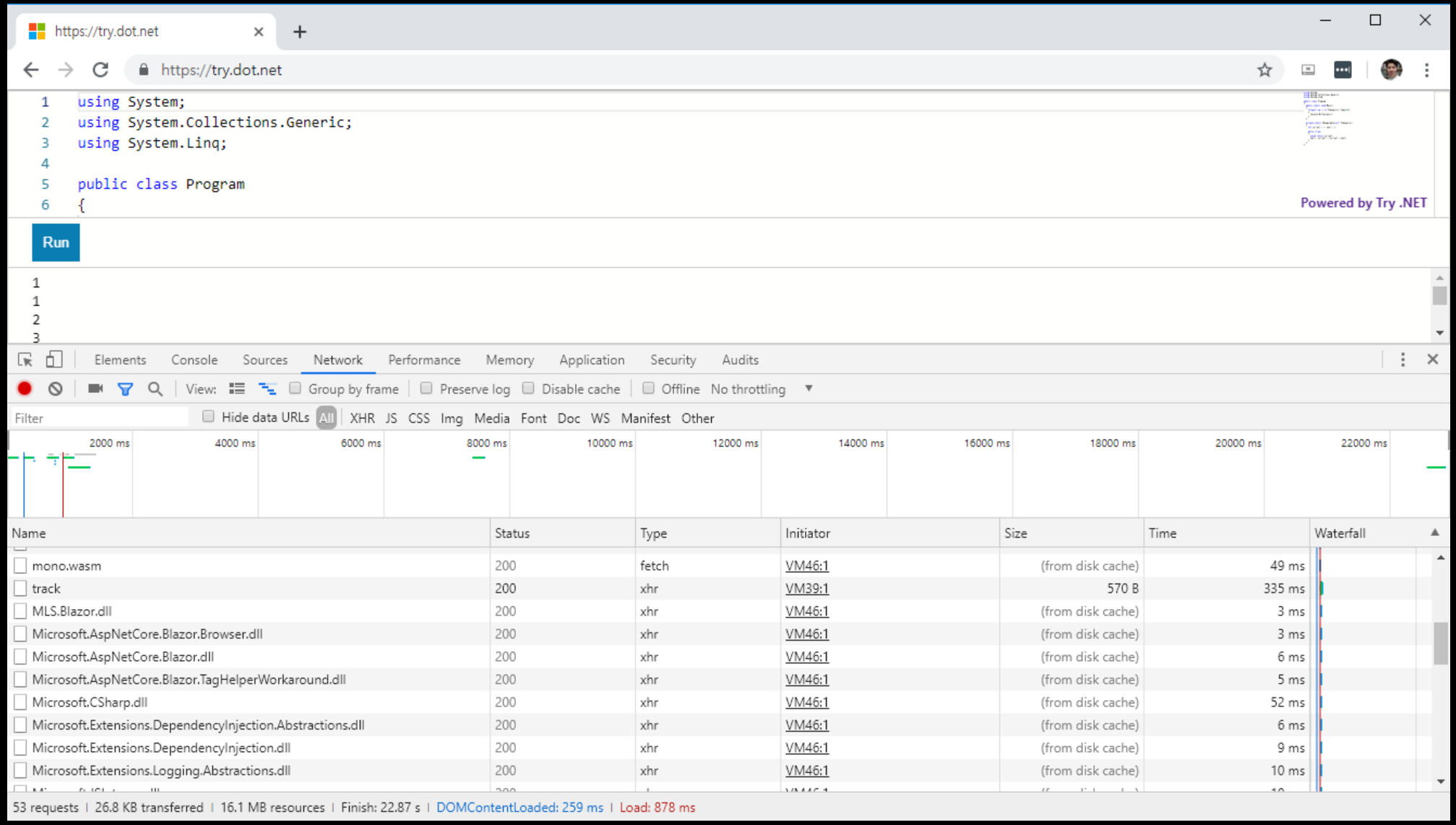
Blazor 특징

- 컴포넌트 모델
- 라우팅
- 레이아웃
- DI(Dependency Injection)
- JavaScript Interop
- 자동 빌드
- 디버깅
- 게시
- Visual Studio의 인텔리센스


.cs
.razor
닷넷 어셈블리로
컴파일
브라우저
CSS
blazor.webassembly.js
App.dll
.NET
(mscorlib.dll,
System.Core.dll,...)
WebAssembly
(mono.wasm)

Blazor 사용 예

- Try.NET



2020년 01월 버전의 Blazor 개발 환경 구축

박용준:
Microsoft MVP
데브렉(www.devlec.com)

2020년 01월 버전의 Blazor 개발 환경 구축

- Visual Studio 2019 최신 버전 설치에 포함
- Visual Studio Code + .NET Core 3.1 SDK

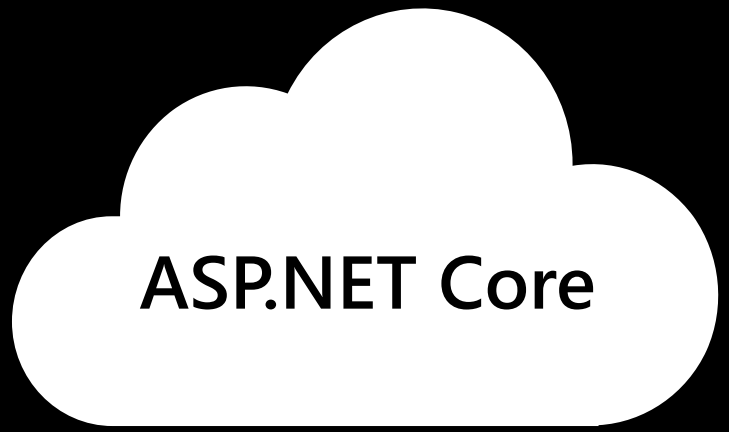
- 참고
  - Blazor 설치
    - > explorer https://www.visualstudio.com/preview
    - > explorer https://dot.net
    - > explorer https://blazor.net
  - dotnet 명령어로 Blazor 템플릿 설치
    - > dotnet new –i Microsoft.AspNetCore.Blazor.Templates

.NET Core 3.X: 풀스택 솔루션

Client

- Blazor
- Components
- SPA (JavaScript)

Frontend



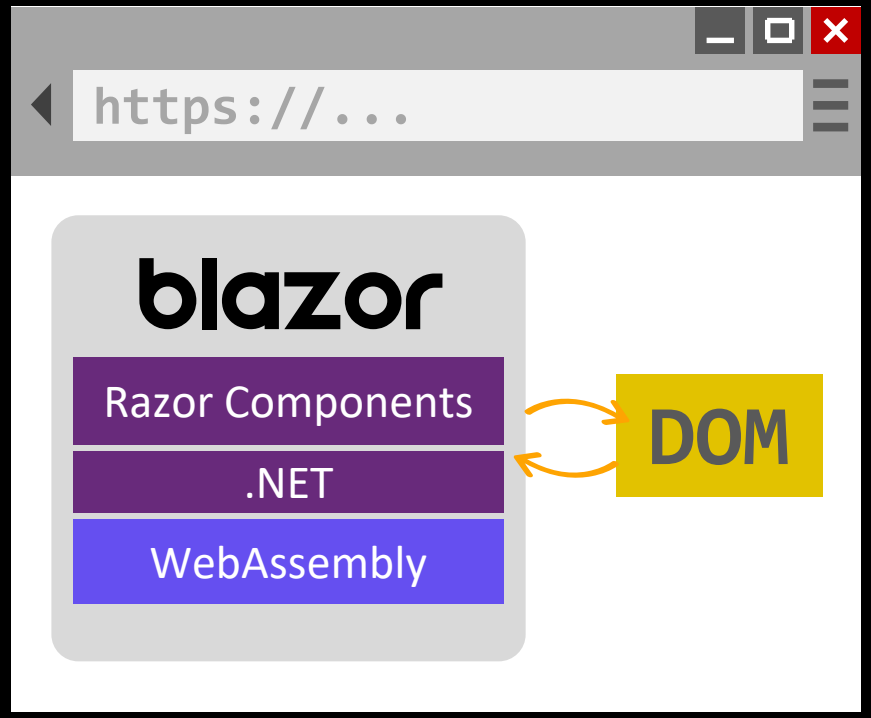
- MVC / Razor Pages
- Web APIs
- SignalR
- Security & identity

Backend

- Worker services
- gRPC

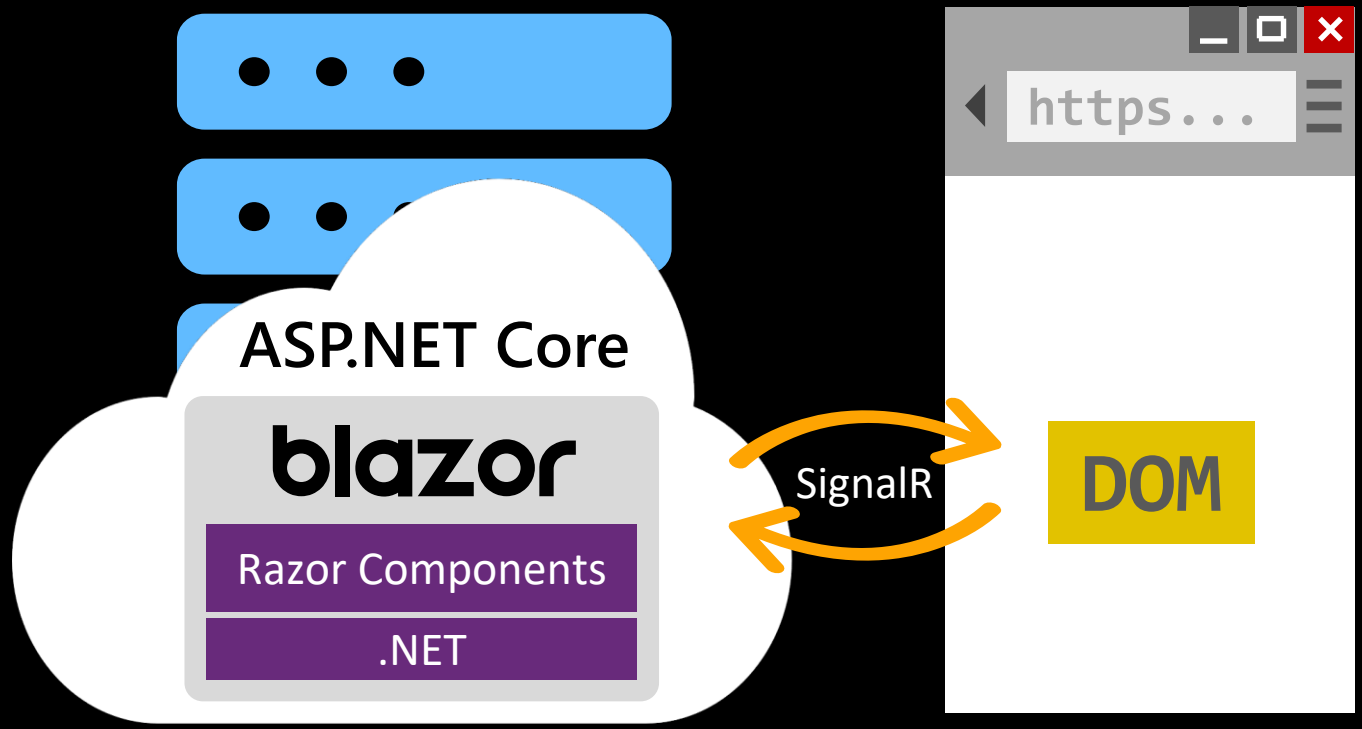
클라이언트 쪽 Blazor와 서버 쪽 Blazor

Client-side



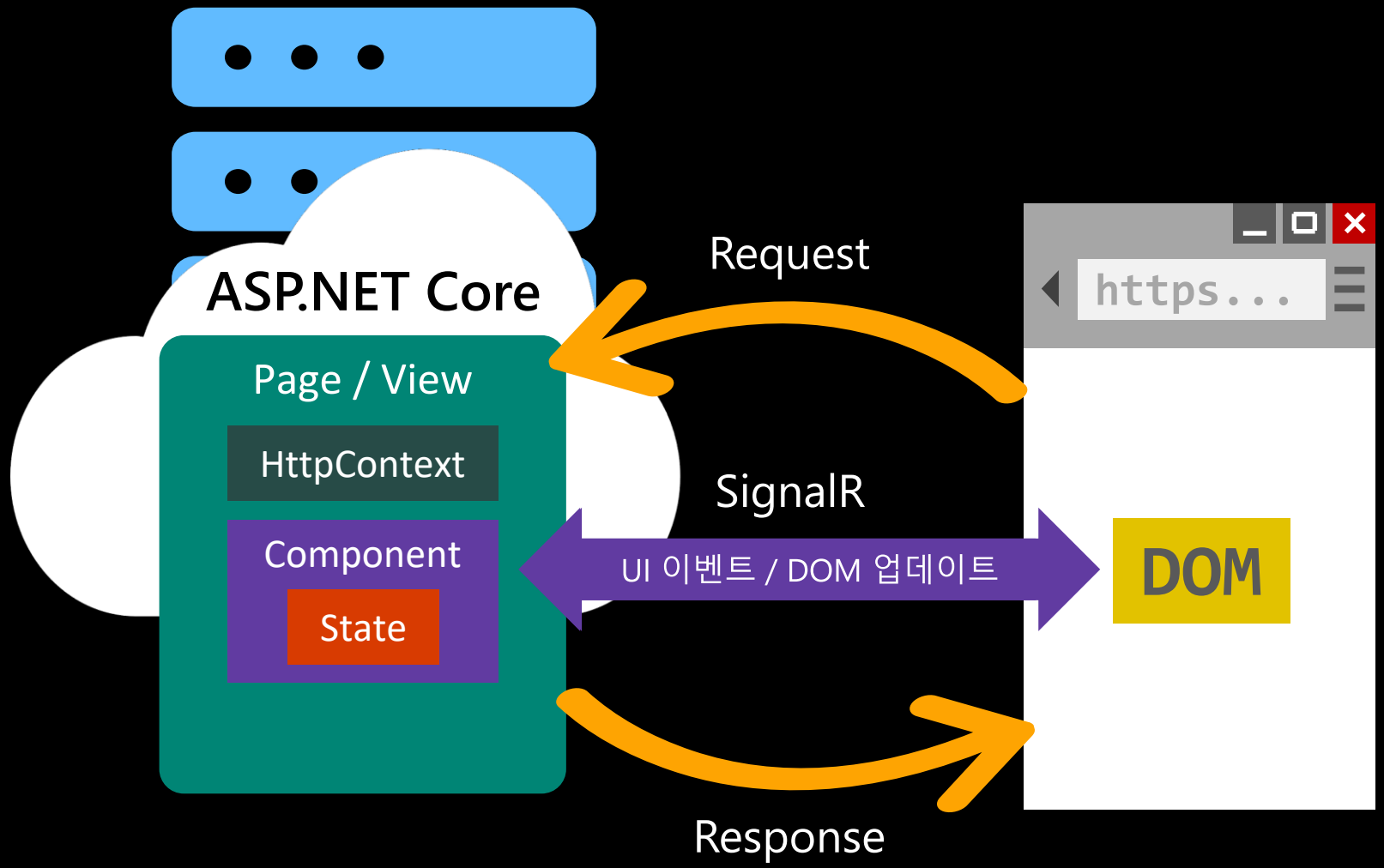
.NET Core 3.1 출시 이후

Server-side

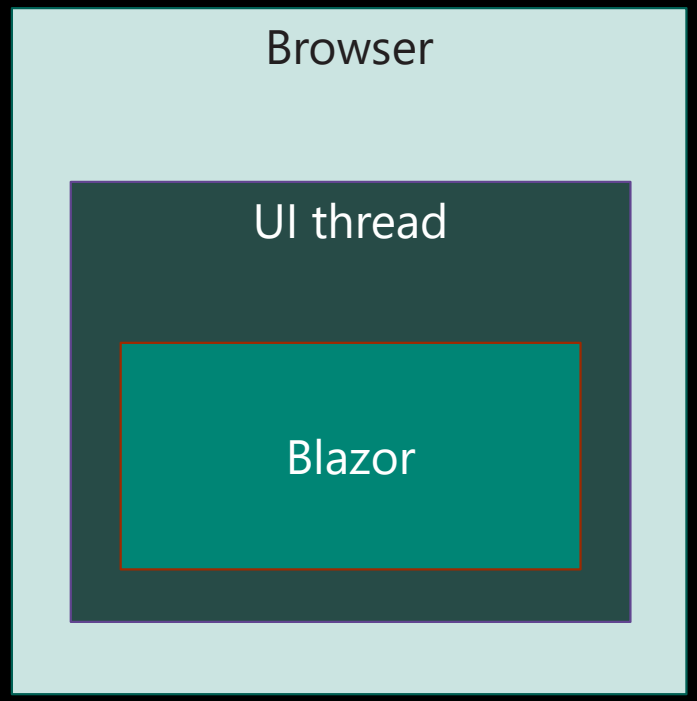


.NET Core 3.0과 함께

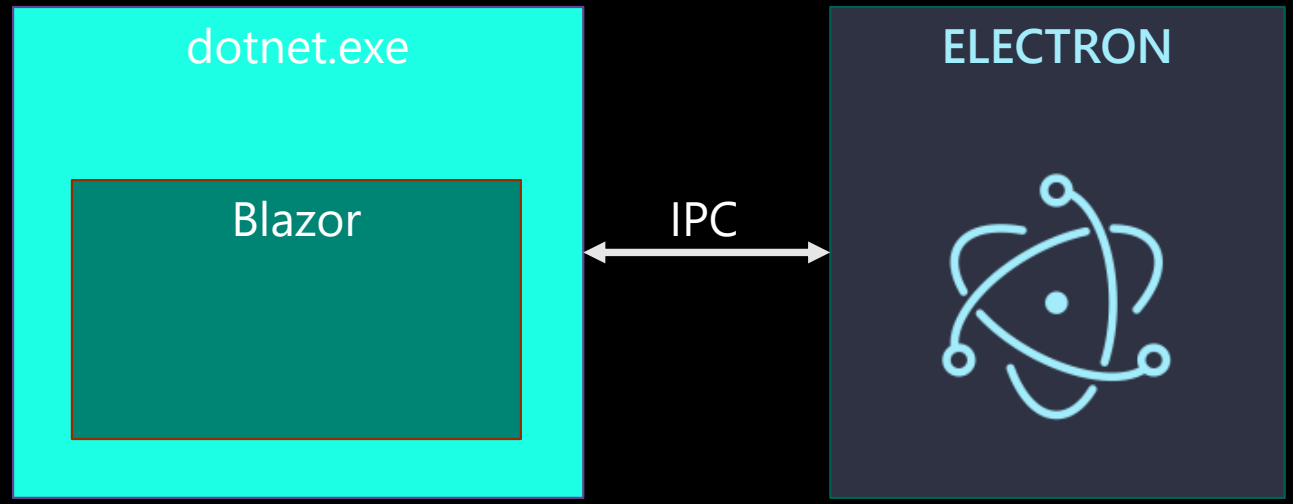
컴포넌트 기반 페이지와 뷰



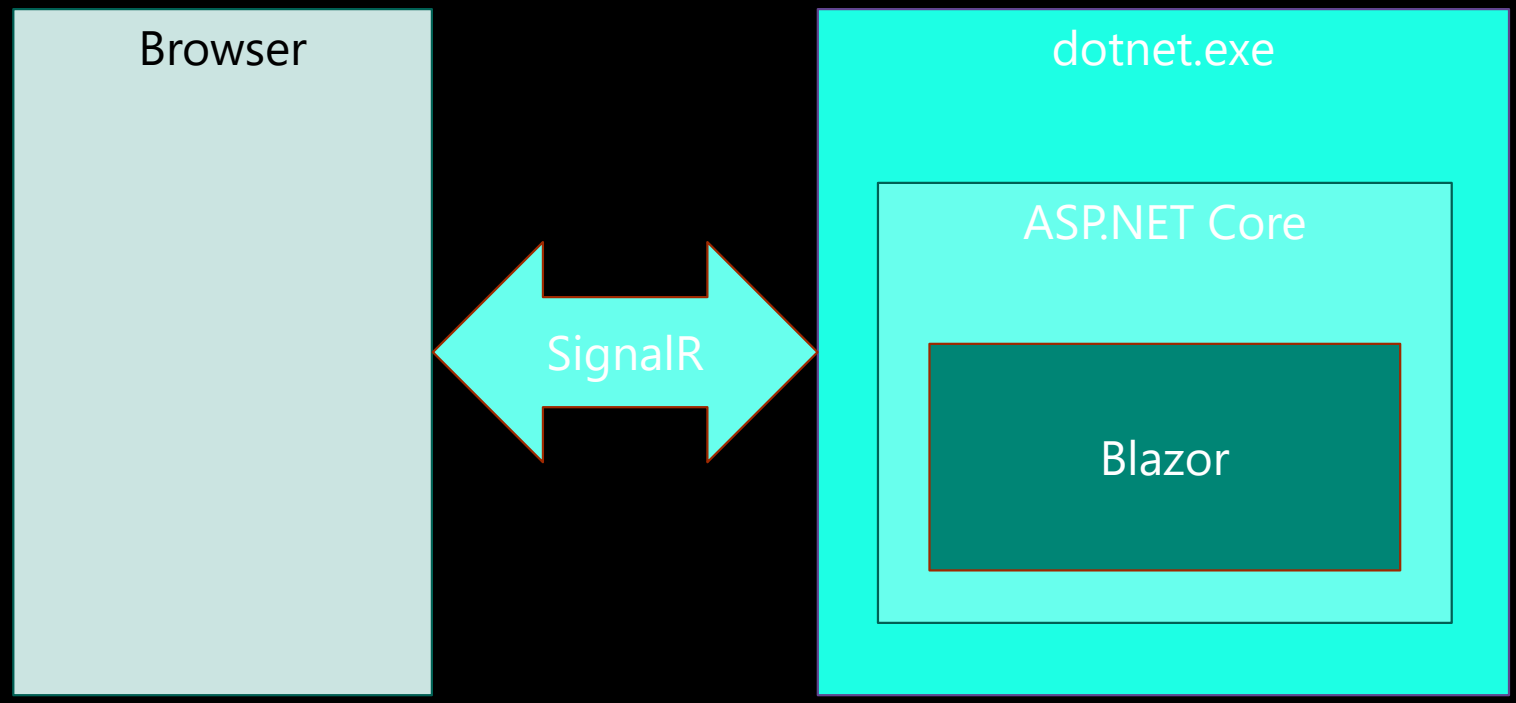
Client-side Blazor



Blazor + Electron



Server-side Blazor

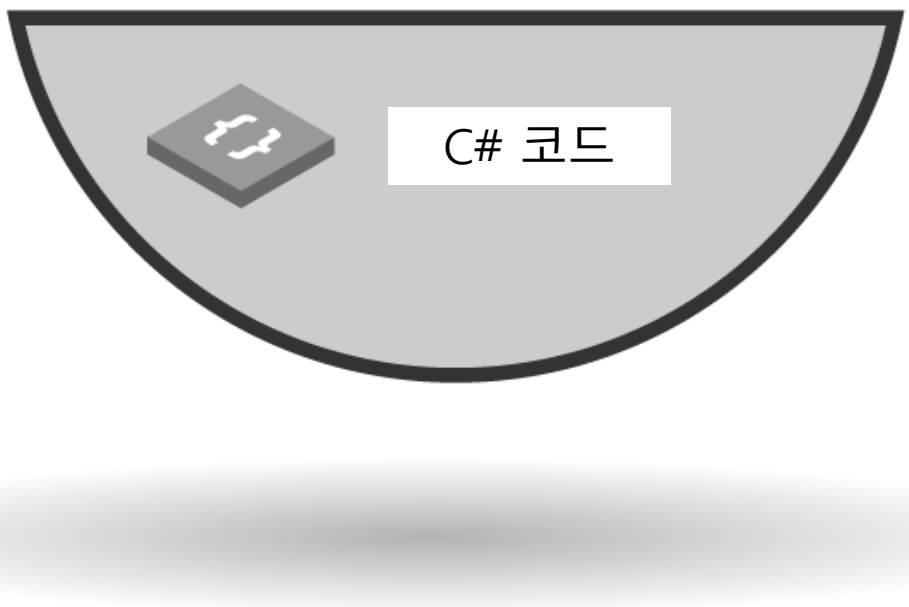


.NET Core 3.1 / ASP.NET Core / Blazor 참고 링크

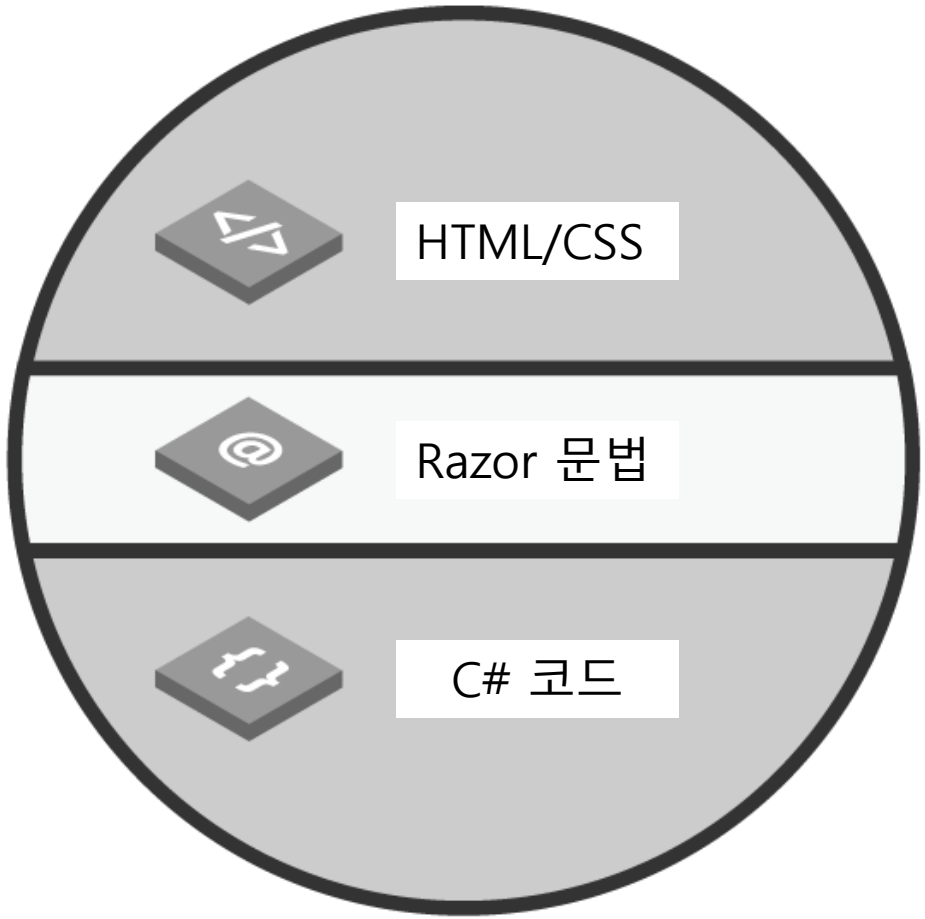
다운로드: https://dot.net/get-core3
Visual Studio: https://visualstudio.com/preview
Docs: https://docs.asp.net
Blazor: https://blazor.net
Workshop: https://aka.ms/blazorworkshop

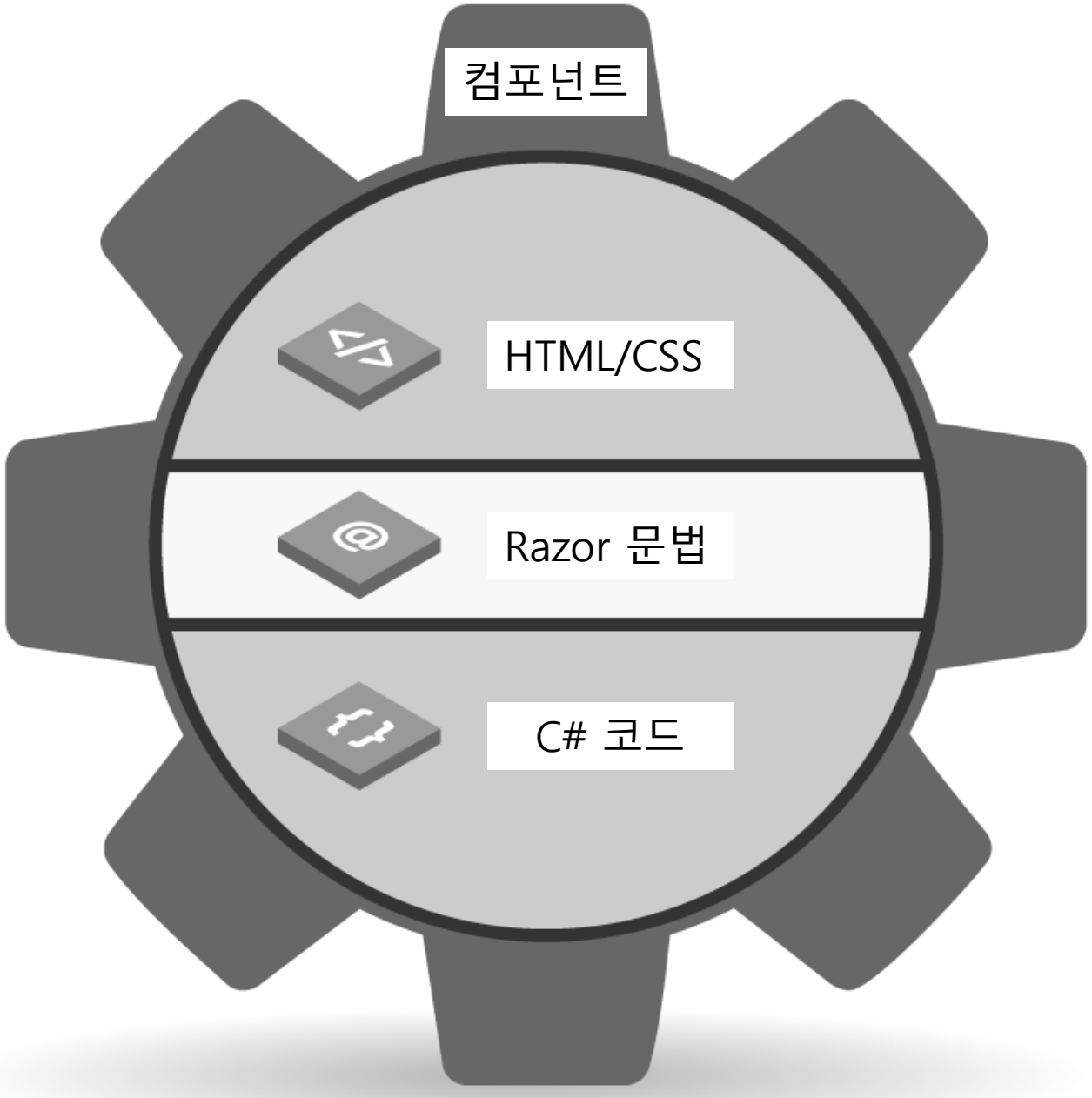
Blazor 소개는 여기까지...

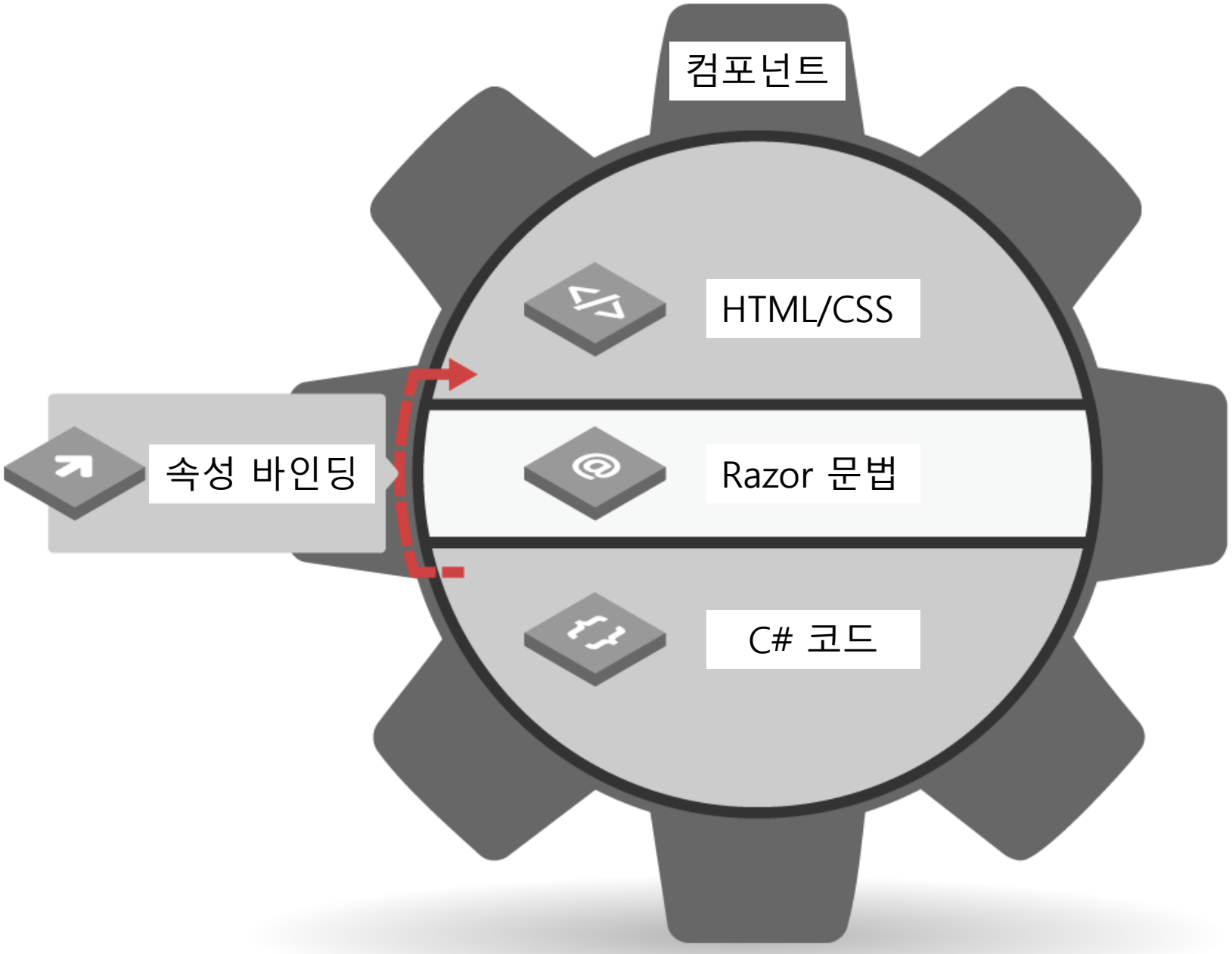
블레이저에서 알아야 할 8가지 핵심 개념

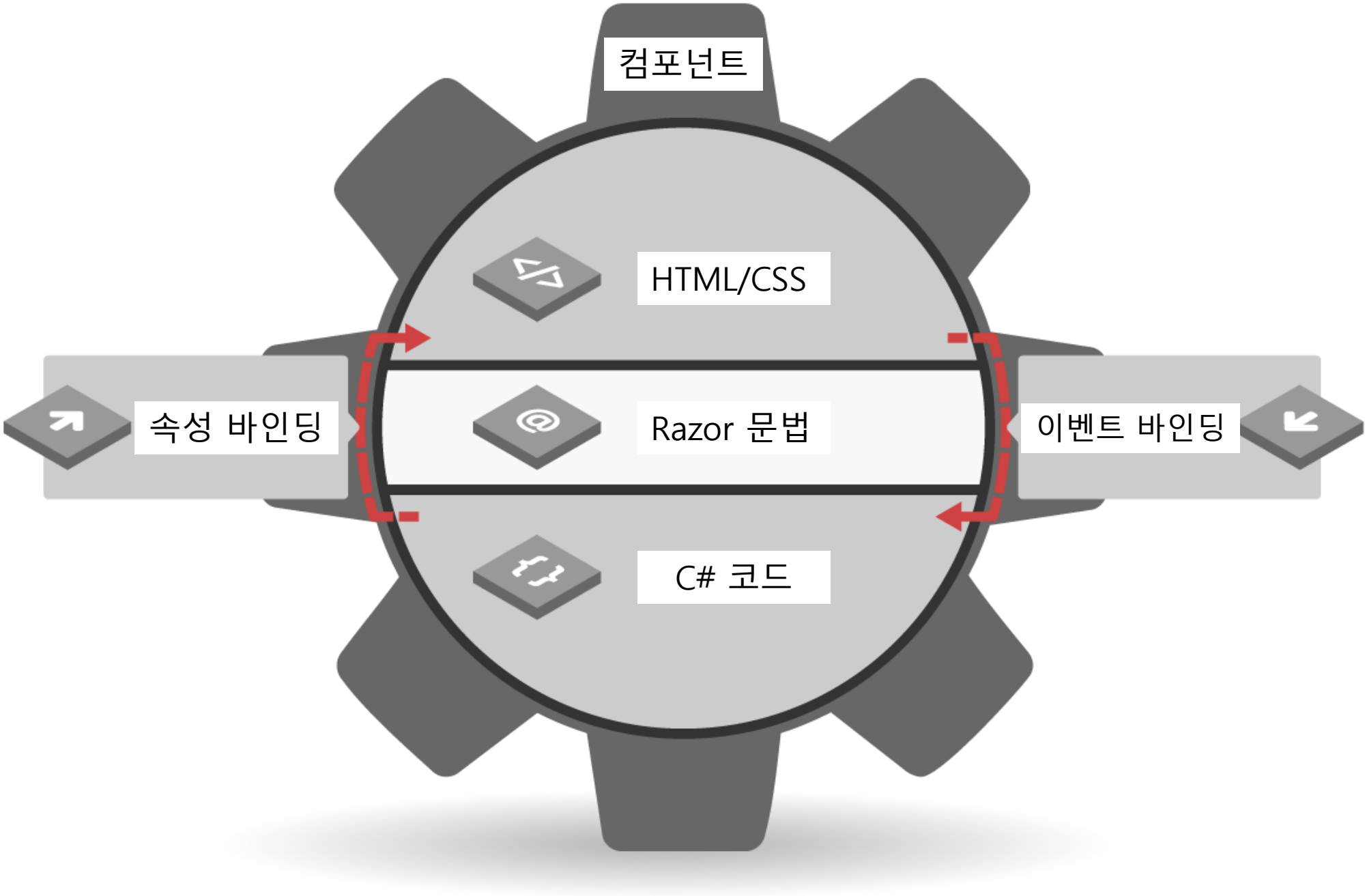

C# 코드

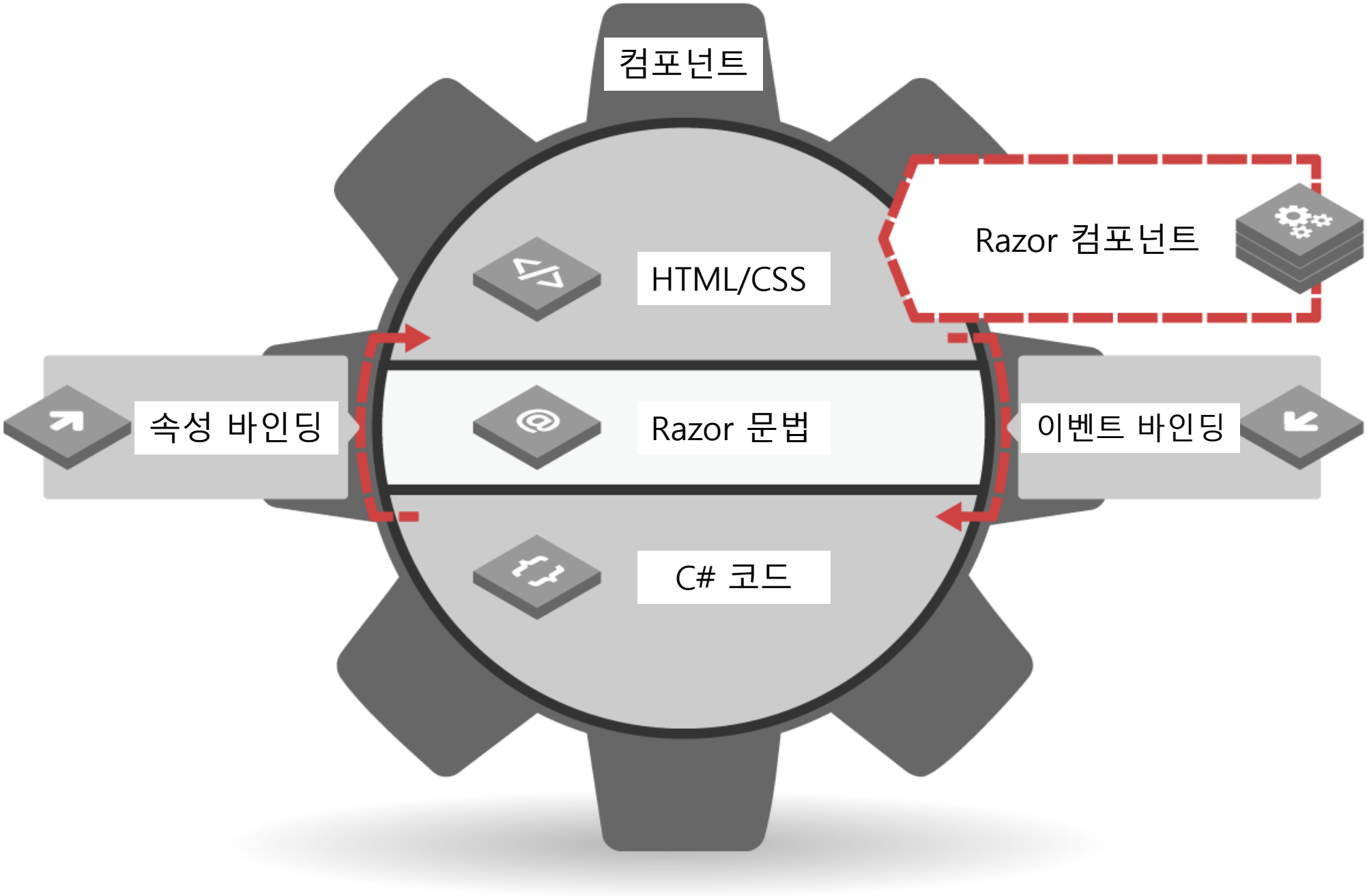

Razor 문법
C# 코드

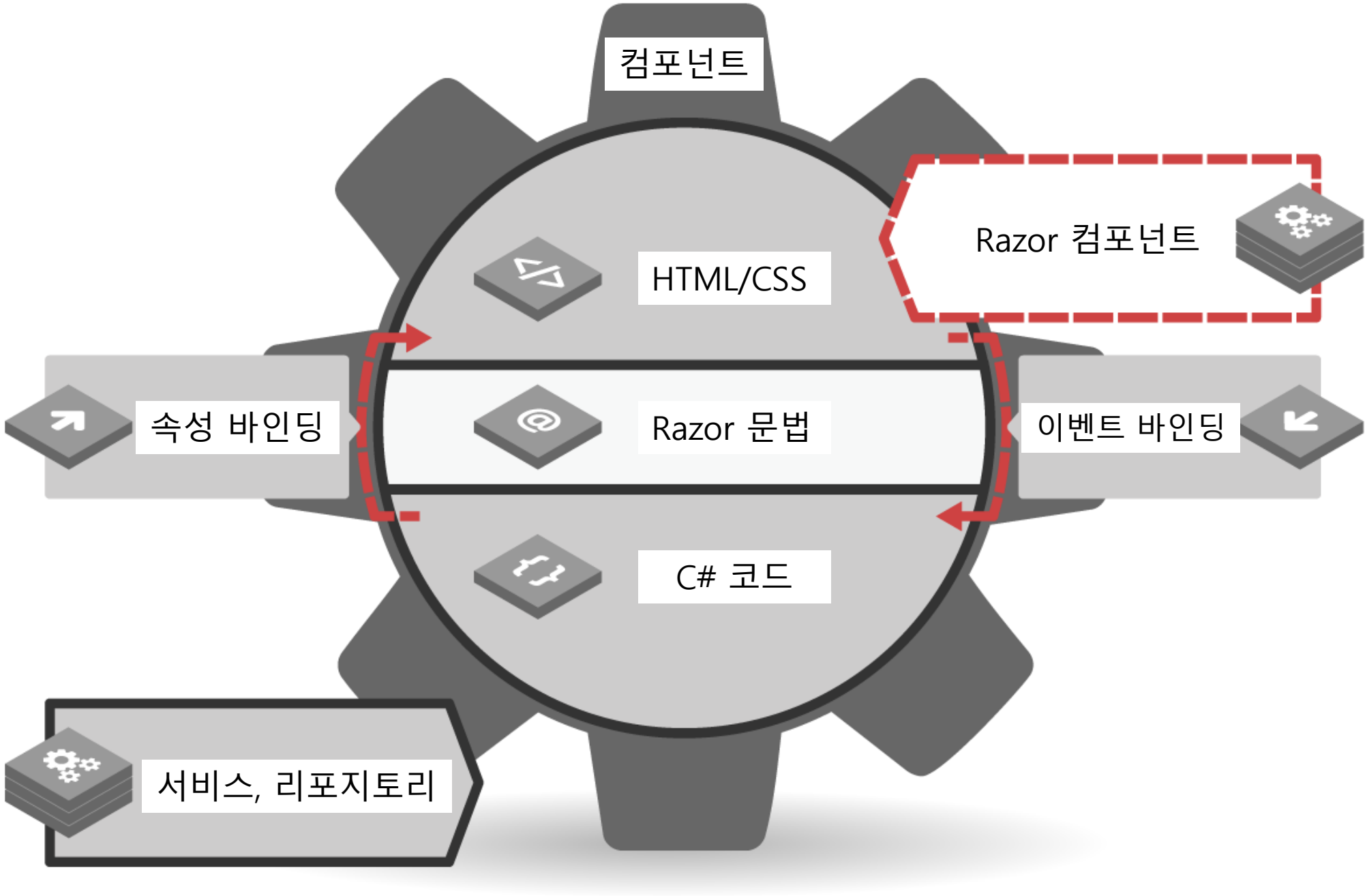

HTML/CSS
Razor 문법
C# 코드

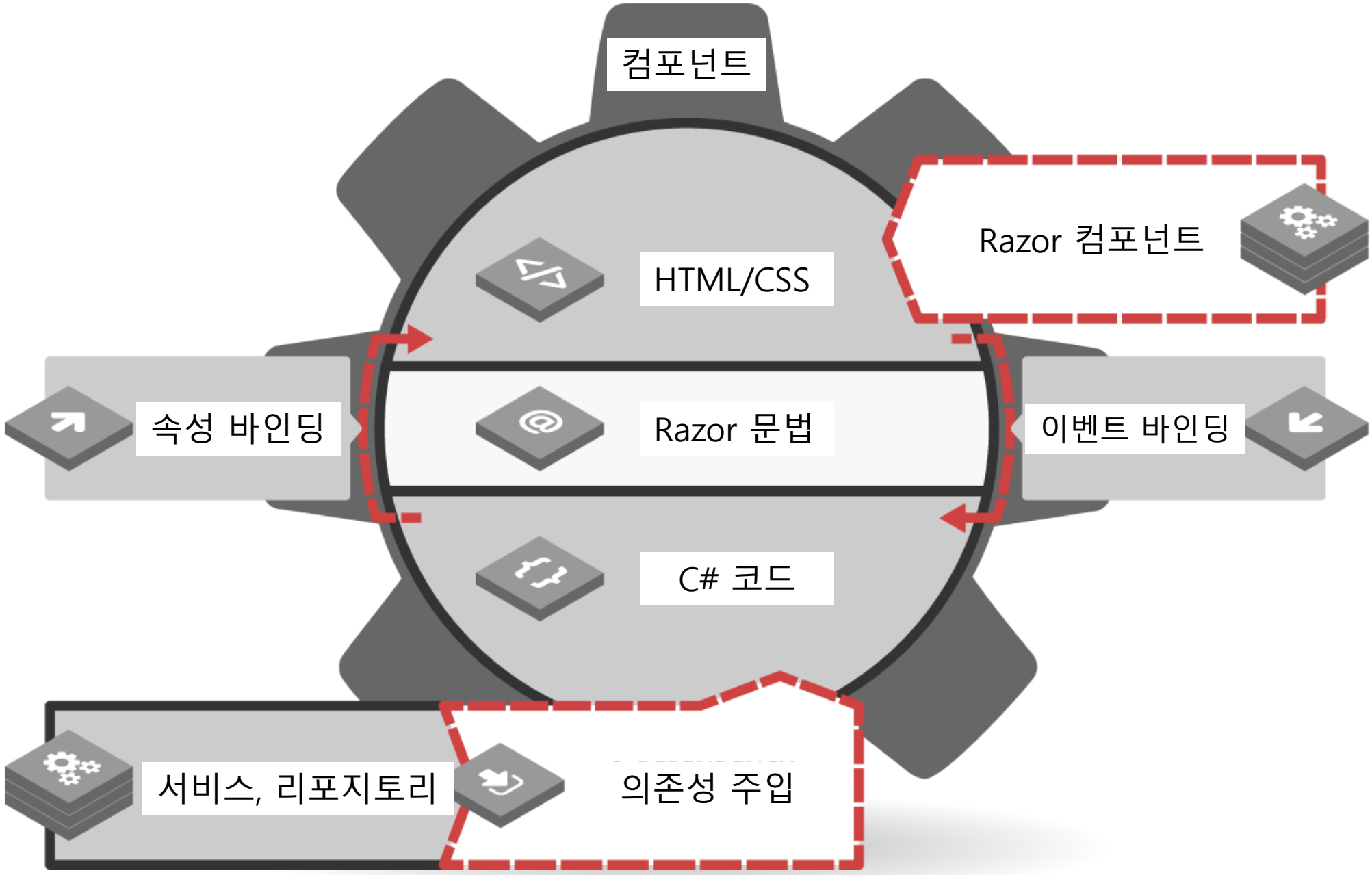

컴포넌트
HTML/CSS
Razor 문법
C# 코드


컴포넌트
HTML/CSS
속성 바인딩
Razor 문법
C# 코드


컴포넌트
HTML/CSS
속성 바인딩
Razor 문법
이벤트 바인딩
C# 코드

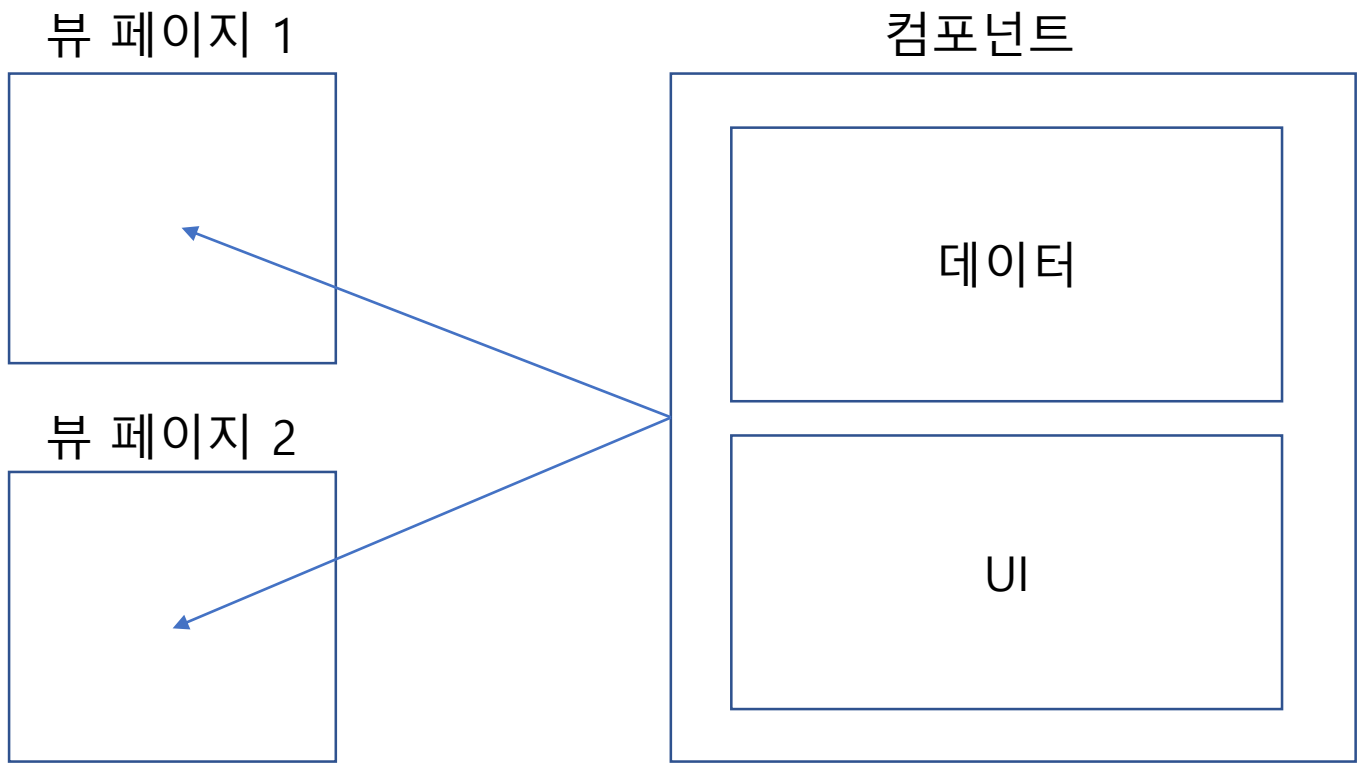

컴포넌트
HTML/CSS
Razor 컴포넌트
속성 바인딩
Razor 문법
이벤트 바인딩
C# 코드


컴포넌트
HTML/CSS
Razor 컴포넌트
속성 바인딩
Razor 문법
이벤트 바인딩
C# 코드
서비스, 리포지토리

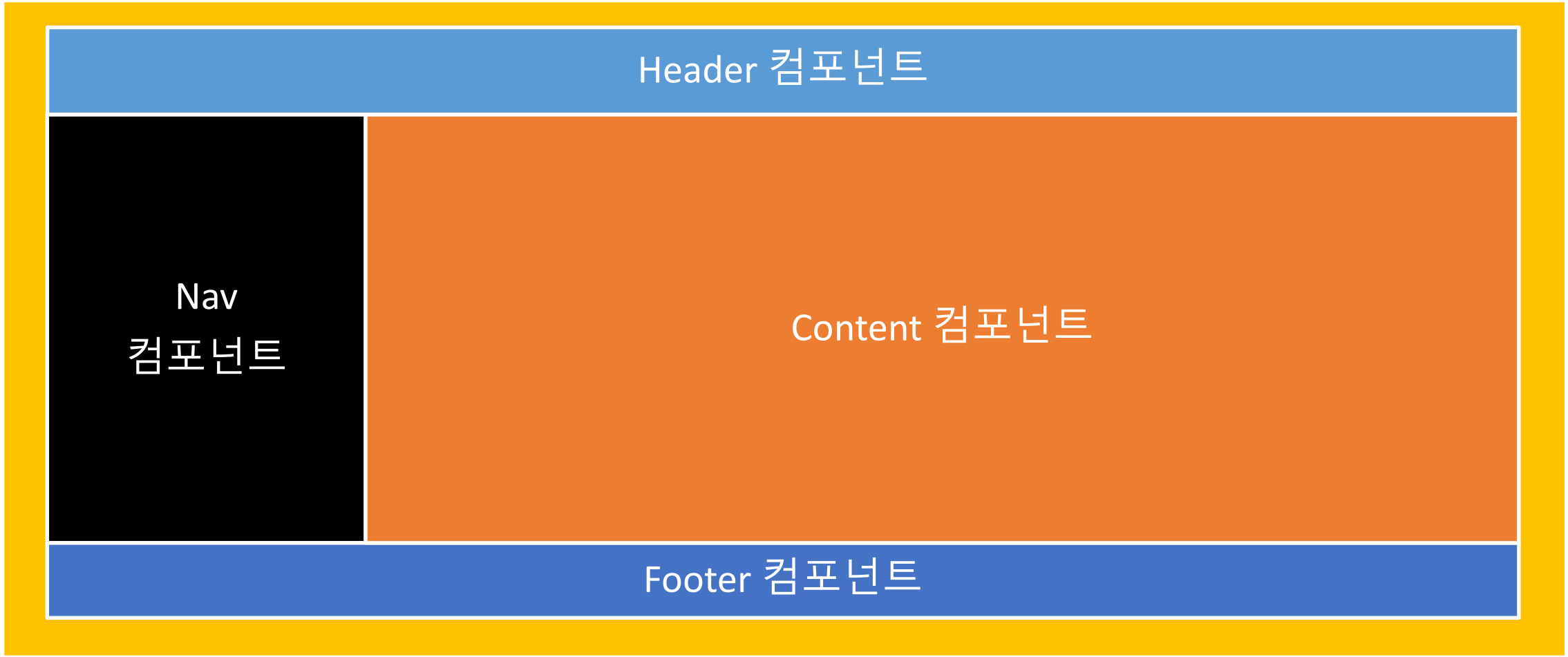

컴포넌트
HTML/CSS
Razor 컴포넌트
속성 바인딩
Razor 문법
이벤트 바인딩
C# 코드
서비스, 리포지토리
의존성 주입

컴포넌트(Component)

컴포넌트(Component): 뷰 컴포넌트



컴포넌트를 사용하여 앱 조립



블레이저 컴포넌트

- App 컴포넌트
- Child 컴포넌트
- Service 컴포넌트

컴포넌트(Component)

- 컴포넌트 = 템플릿 + 클래스 + 메타데이터
  - 템플릿
    - 레이아웃 뷰
    - HTML으로 생성
    - 바인딩 포함
  - 클래스(C# 클래스): 대문자로 시작
  - 메타데이터(C#의 특성(Attribute), Java의 어노테이션)
    - @using, @inject 등
      - 데코레이터로 정의
      - 외부 기능(데이터) 제공

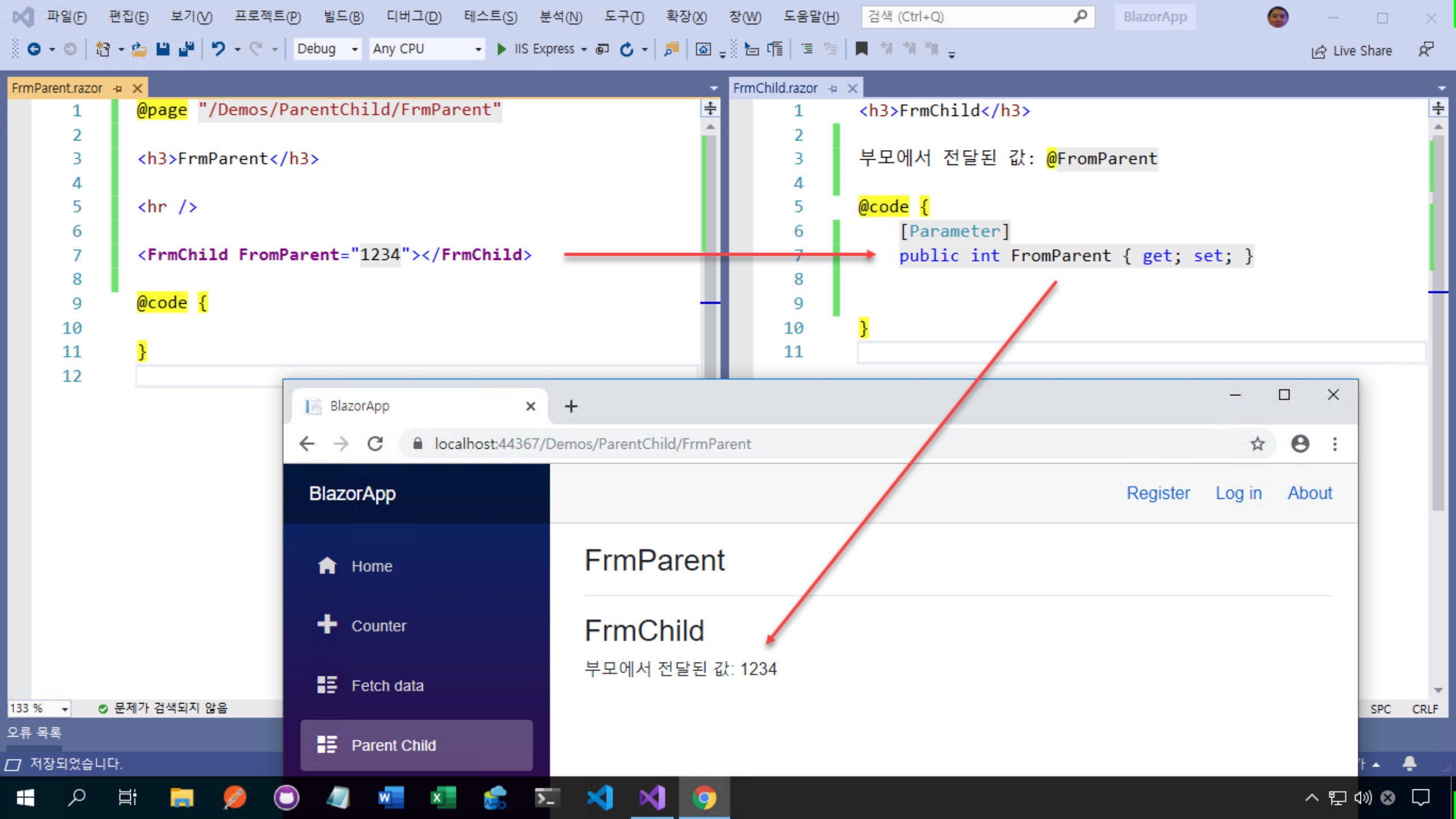
@using -필요한 외부 모듈 포함하기

- 외부 함수 또는 클래스를 현재 페이지에서 사용하기 위한 절차
- 앵귤러는 모든 기능이 모듈화 됨

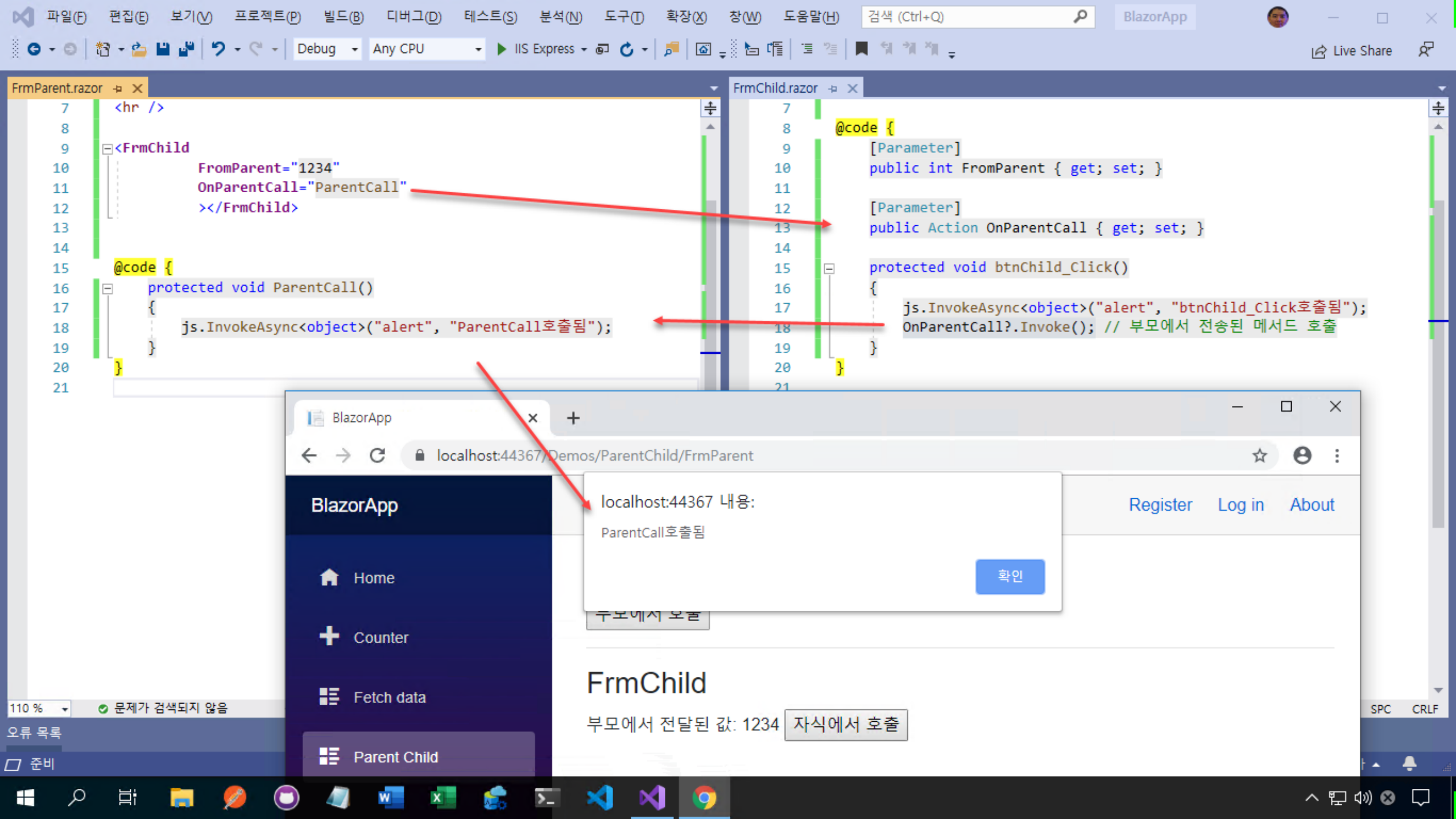
@inject 구문


inject 키워드
인터페이스 이름
사용할 클래스 이름
@inject IProjectRepository ProjectRepository;

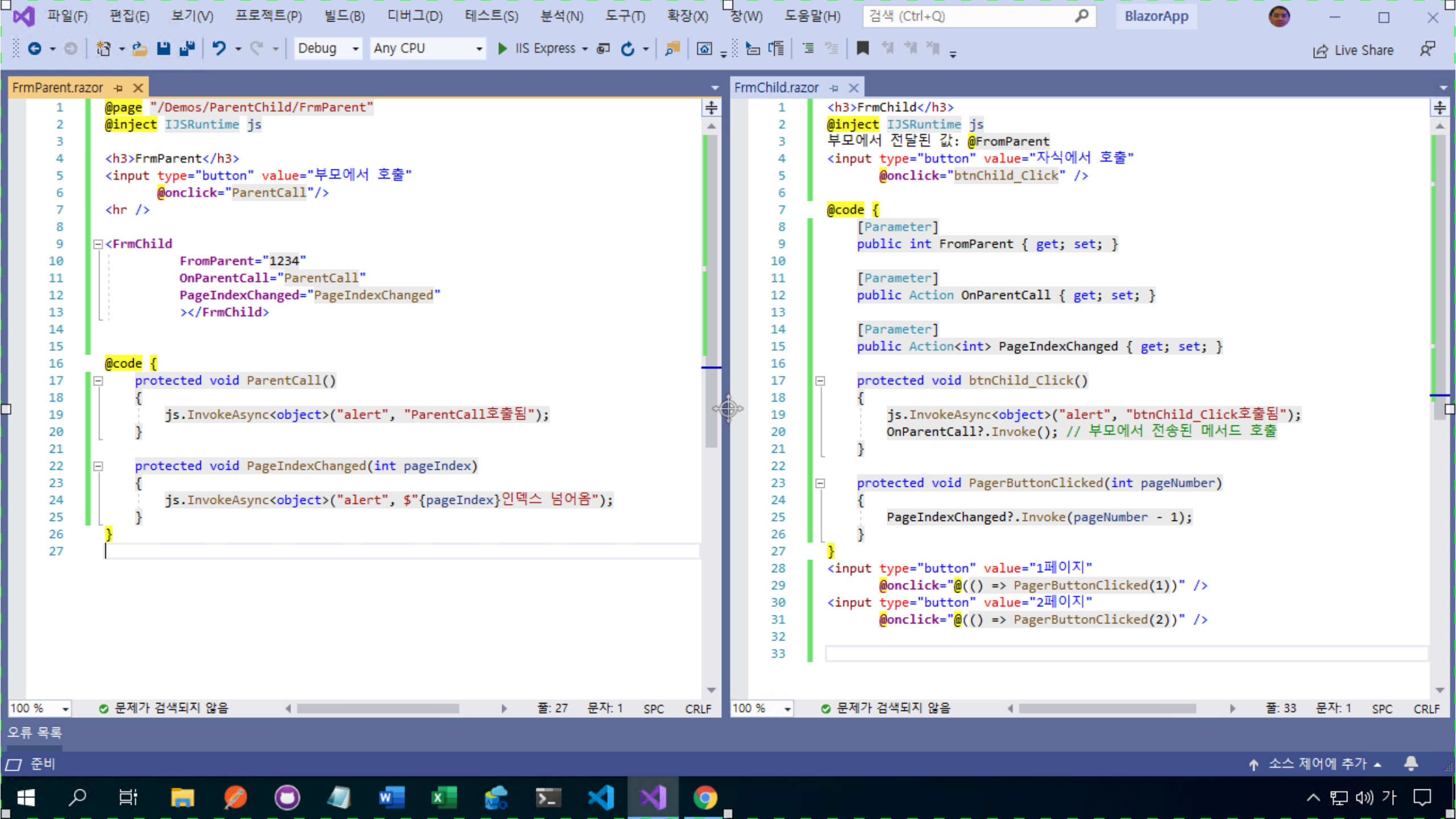
부모 컴포넌트에서 자식 컴포넌트로


FrmParent.razor
@page "/Demos/ParentChild/FrmParent"
<h3>FrmParent</h3>
<hr />
<FrmChild FromParent="1234"></FrmChild>
@code {
}
FrmChild.razor
<h3>FrmChild</h3>
부모에서 전달된 값: @FromParent
@code {
[Parameter]
public int FromParent { get; set; }
}
BlazorApp
localhost:44367/Demos/ParentChild/FrmParent
BlazorApp
Register
Log in
About
Home
Counter
Fetch data
Parent Child
FrmParent
FrmChild
부모에서 전달된 값: 1234

자식 컴포넌트에서 부모 컴포넌트로


파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 테스트(S) 분석(N) 도구(T) 확장(X) 창(W) 도움말(H)
검색 (Ctrl+Q)
BlazorApp
Debug
Any CPU
IIS Express
Live Share
FrmParent.razor
<hr />
<FrmChild
FromParent="1234"
OnParentCall="ParentCall"
></FrmChild>
@code {
protected void ParentCall()
{
js.InvokeAsync<object>("alert", "ParentCall호출됨");
}
}
FrmChild.razor
@code {
[Parameter]
public int FromParent { get; set; }
[Parameter]
public Action OnParentCall { get; set; }
protected void btnChild_Click()
{
js.InvokeAsync<object>("alert", "btnChild_Click호출됨");
OnParentCall?.Invoke(); // 부모에서 전송된 메서드 호출
}
}
BlazorApp
localhost:44367/Demos/ParentChild/FrmParent
localhost:44367 내용:
ParentCall호출됨
확인
BlazorApp
Register
Log in
About
Home
Counter
Fetch data
Parent Child
FrmChild
부모에서 전달된 값: 1234
자식에서 호출
110 %
문제가 검색되지 않음
오류 목록
준비
SPC
CRLF

자식 컴포넌트에서 부모로 값 전달


파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 테스트(S) 분석(N) 도구(T) 확장(X) 창(W) 도움말(H)
검색 (Ctrl+Q)
BlazorApp
Debug
Any CPU
IIS Express
Live Share
FrmParent.razor
@page "/Demos/ParentChild/FrmParent"
@inject IJSRuntime js
<h3>FrmParent</h3>
<input type="button" value="부모에서 호출"
@onclick="ParentCall"/>
<hr />
<FrmChild
FromParent="1234"
OnParentCall="ParentCall"
PageIndexChanged="PageIndexChanged"
></FrmChild>
@code {
protected void ParentCall()
{
js.InvokeAsync<object>("alert", "ParentCall호출됨");
}
protected void PageIndexChanged(int pageIndex)
{
js.InvokeAsync<object>("alert", $"{pageIndex}인덱스 넘어옴");
}
}
100 %
문제가 검색되지 않음
줄: 27
문자: 1
SPC
CRLF
FrmChild.razor
<h3>FrmChild</h3>
@inject IJSRuntime js
부모에서 전달된 값: @FromParent
<input type="button" value="자식에서 호출"
@onclick="btnChild_Click" />
@code {
[Parameter]
public int FromParent { get; set; }
[Parameter]
public Action OnParentCall { get; set; }
[Parameter]
public Action<int> PageIndexChanged { get; set; }
protected void btnChild_Click()
{
js.InvokeAsync<object>("alert", "btnChild_Click호출됨");
OnParentCall?.Invoke(); // 부모에서 전송된 메서드 호출
}
protected void PagerButtonClicked(int pageNumber)
{
PageIndexChanged?.Invoke(pageNumber - 1);
}
}
<input type="button" value="1페이지"
@onclick="@(() => PagerButtonClicked(1))" />
<input type="button" value="2페이지"
@onclick="@(() => PagerButtonClicked(2))" />
100 %
문제가 검색되지 않음
줄: 33
문자: 1
SPC
CRLF
오류 목록
준비
소스 제어에 추가

중첩 컴포넌트(부모와 자식)

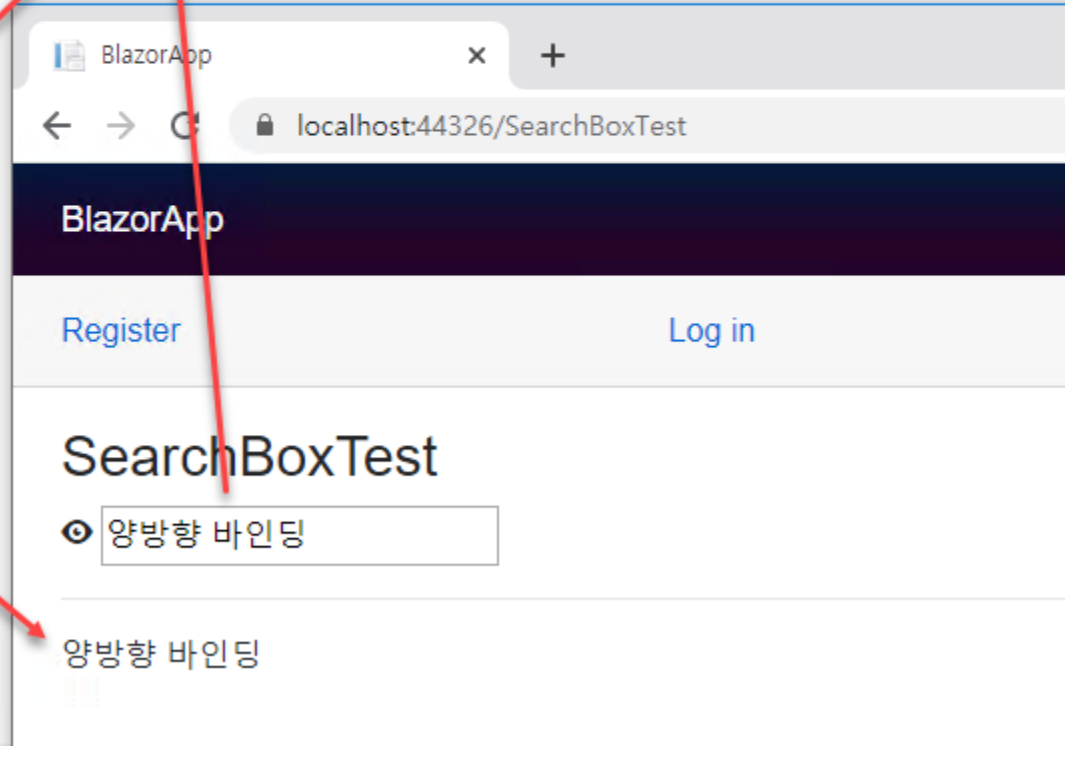
부모 컴포넌트와 자식 컴포넌트

- 모든 컴포넌트는 자식 컴포넌트 포함
  - 손자 컴포넌트(?)
  - 중첩(Nested) 컴포넌트

Parameter로 부모에서 자식으로 값 전달

양방향 바인딩

```
<div class="search">
    <i class="oi oi-eye"></i>
    <input placeholder="Search..." @bind="SearchQuery" @bind:event="oninput" />
</div>
<hr />
@SearchQuery

@code {
    private string searchQuery;

    public string SearchQuery
    {
        get => searchQuery;
        set { searchQuery = value; }
    }
}
```



SearchBox.razor

```razor
@using System.Timers
@implements IDisposable

<div class="search">
    <i class="oi oi-eye"></i>
    <input placeholder="Search..."
           @attributes="AdditionalAttributes" @bind="SearchQuery" @bind:event="oninput" />
    <input type="button" value="Search" @onclick="Search" />
</div>
<hr />
자식: @SearchQuery

@code {
    private string searchQuery;
    private Timer debounceTimer;

    public string SearchQuery
    {
        get => searchQuery;
        set
        {
            searchQuery = value;
            debounceTimer.Stop();
            debounceTimer.Start();
        }
    }

    [Parameter(CaptureUnmatchedValues = true)]
    public IDictionary<string, object> AdditionalAttributes { get; set; }

    // 자식 컴포넌트에서 발생한 정보를 부모 컴포넌트에게 전달
    [Parameter]
    public EventCallback<string> SearchQueryChanged { get; set; }

    [Parameter]
    public int Debounce { get; set; } = 300;

    protected override void OnInitialized()
    {
        debounceTimer = new Timer();
        debounceTimer.Interval = Debounce;
        debounceTimer.AutoReset = false;
        debounceTimer.Elapsed += SearchHandler;
    }

    protected void Search()
    {
        SearchQueryChanged.InvokeAsync(SearchQuery); // 부모의 메서드에 검색어 전달
    }

    protected async void SearchHandler(object source, ElapsedEventArgs e)
    {
        await InvokeAsync(() => SearchQueryChanged.InvokeAsync(SearchQuery)); // 부모의 메서드에 검색어 전달
    }

    public void Dispose()
    {
        debounceTimer.Dispose();
    }
}

```

SearchBoxTest.razor

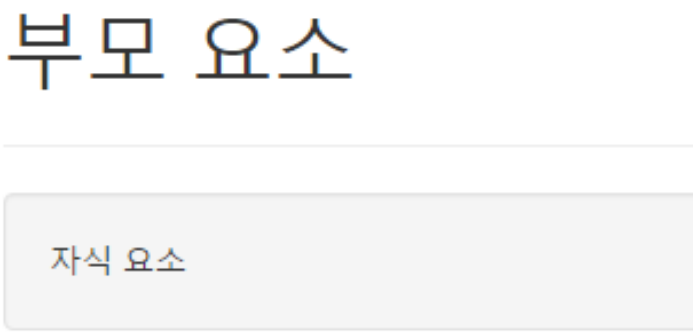
```razor
@page "/SearchBoxTest"
@using BlazorApp.Components


<h3>SearchBoxTest</h3>

<SearchBox></SearchBox>
<SearchBox placeholder="Search Query..." SearchQueryChanged="SearchQueryChanged"></SearchBox>

<hr />
부모: @searchQuery

@code {
    private string searchQuery;
    protected void SearchQueryChanged(string searchQuery)
    {
        this.searchQuery = searchQuery;
    }
}

```

컴포넌트

- 재사용 가능한 UI와 로직
- 컴포넌트도 하나의 클래스
  - ComponentBase 클래스 상속

Input 데코레이터와 Output 데코레이터

- Input Parameter
  - 부모에서 자식으로
  - 컴포넌트에게 속성을 사용하여 데이터 전달
- Output Parameter
  - 자식에서 부모로
  - Action
    - 자식의 값을 부모에서 사용하도록
    - 자식에서 부모의 메서드를 호출
  - EventCallback<T>
    - 제네릭으로 number, string 형 정해서 전달



자식요소의 @ref를 사용하여 접근

- @ref
  - 템플릿 참조 변수
- <부모템플릿>
- <자식요소 @ref="id"></자식요소>
- @id.자식요소의속성또는메서드
- </부모템플릿>

[Inject] 특성

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

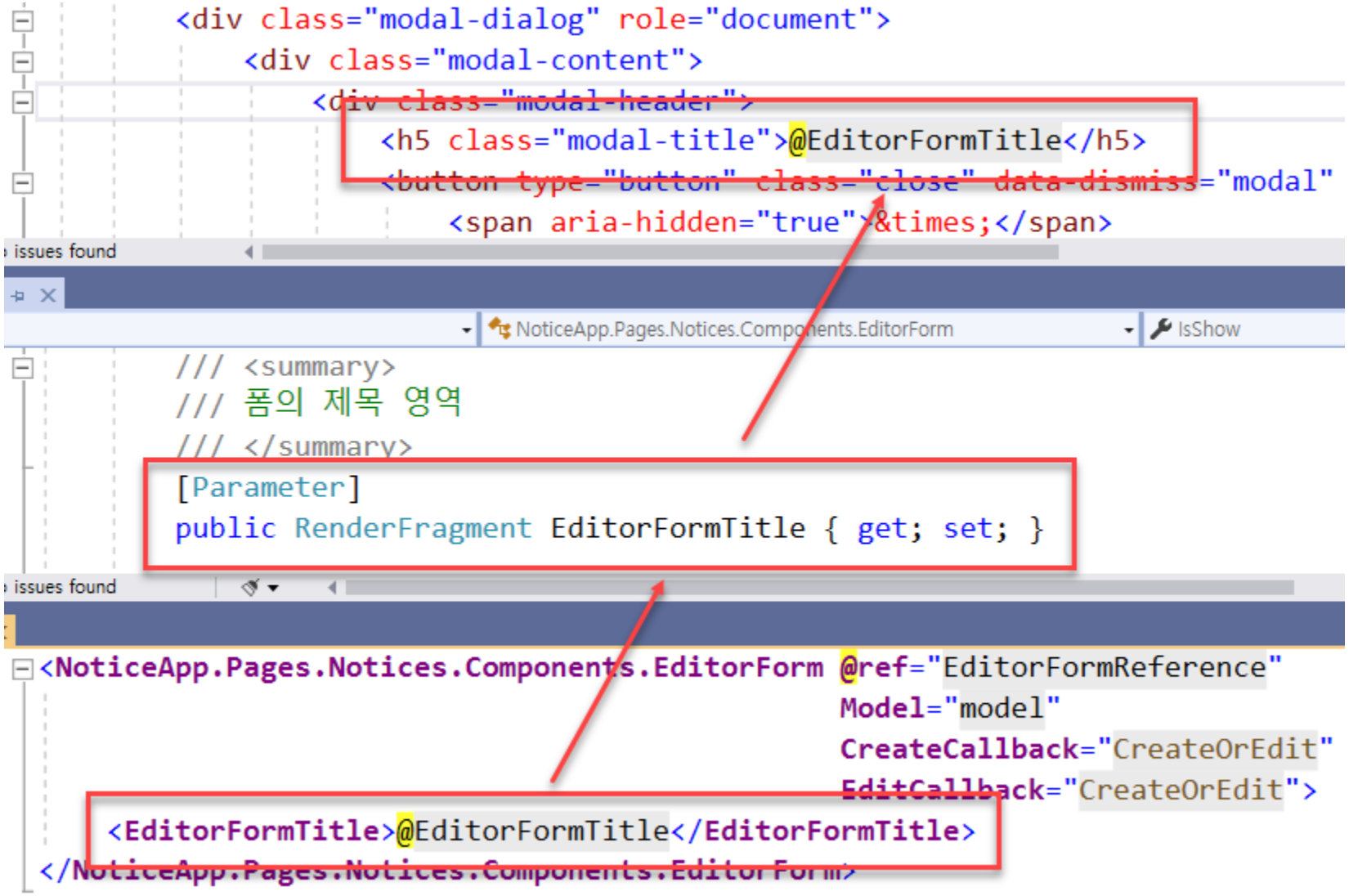
```
private void btnProductName_Click(int ProductId)
{
    NavigationManager.NavigateTo($"/Products/Details/{ProductId}");
}
```

MarkupString

```
<div class="form-group row">
    <div class="col-sm-12">
        @((MarkupString)content)
    </div>
</div>
```

```
protected string content = "";

protected override async Task OnInitializedAsync()
{
    model = await NoticeRepositoryAsyncReference.GetByIdAsync(Id);
    content = Dul.HtmlUtility.EncodeWithTabAndSpace(model.Content);
}
```

RenderFragment: 자식 컴포넌트 HTML



NavigationManager

- Uri
  - 절대 URI
- BaseUri
  - 기본 URI
- NavigateTo()
  - 지정된 URL로 이동
- LocationChanged
  - URL이 변경될 때 발생되는 이벤트
- ToAbsoluteUri()
  - 상대 URL을 절대 URL로 변경
- ToBaseRelativePath()
  - 절대 URI를 상대 URI로 변경

참고 강의: 강의 소스 이동: BlazorApp

- BlazorApp 솔루션을 VisualAcademy 솔루션으로
  - https://github.com/VisualAcademy/BlazorApp

참고 강의: 강의 소스 이동: IdeaAppCore

- IdeaAppCore 솔루션을 VisualAcademy 솔루션으로
  - https://github.com/VisualAcademy/IdeaAppCore

참고: 강의 소스 이동: VideoAppCore

- VideoAppCore 솔루션을 VisualAcademy 솔루션으로
  - https://github.com/VisualAcademy/VideoAppCore

참고: 강의 소스 이동: ManufacturerApp

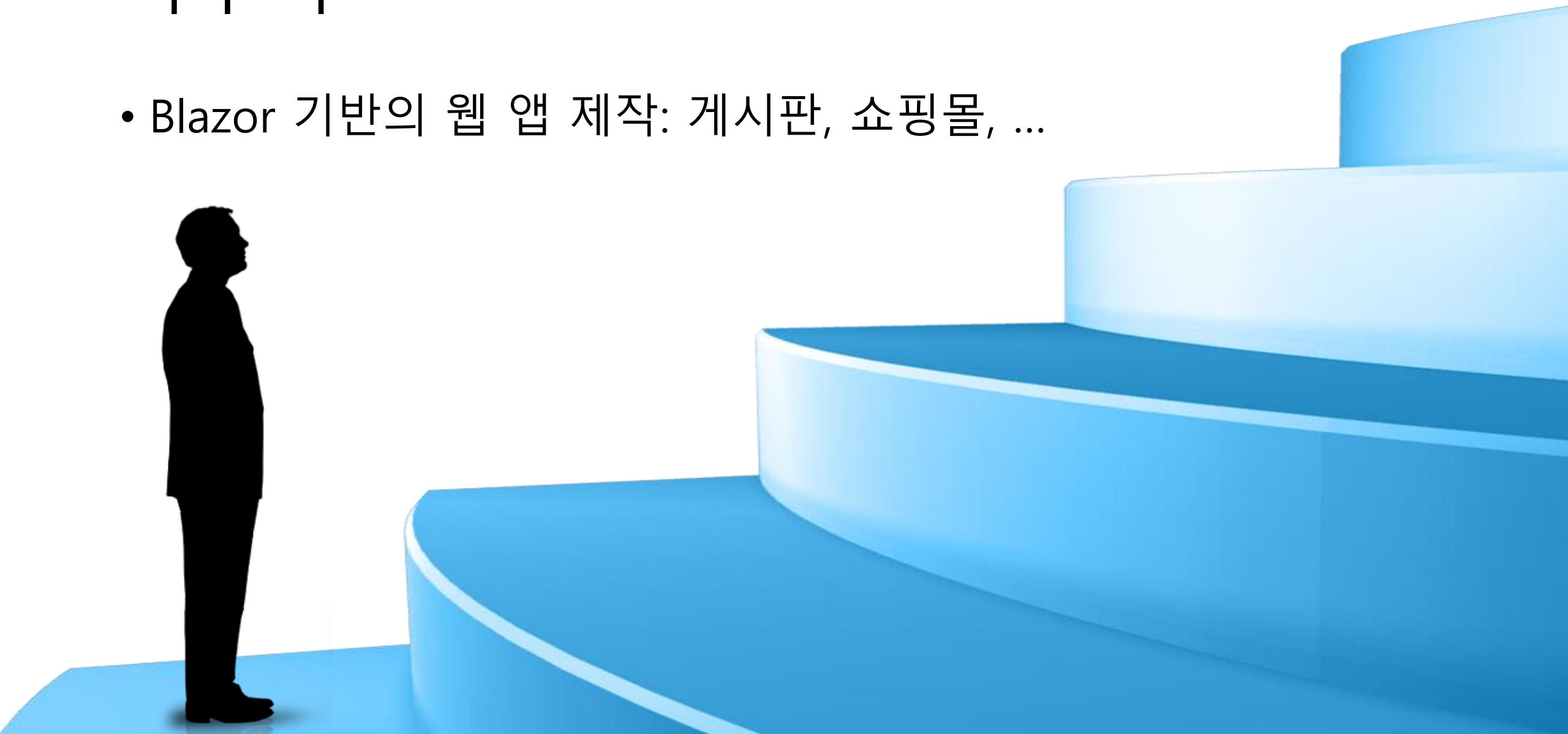
- ManufacturerApp 솔루션을 VisualAcademy 솔루션으로
  - https://github.com/VisualAcademy/ManufacturerApp

참고자료

- 블레이저 공식 사이트
- 채널9
  - Microsoft Ignite

마무리

- Blazor 기반의 웹 앱 제작: 게시판, 쇼핑몰, ...



감사합니다

감사합니다.
박용준

닷넷코리아(https://www.dotnetkorea.com) : 쉽게 배우는 Blazor