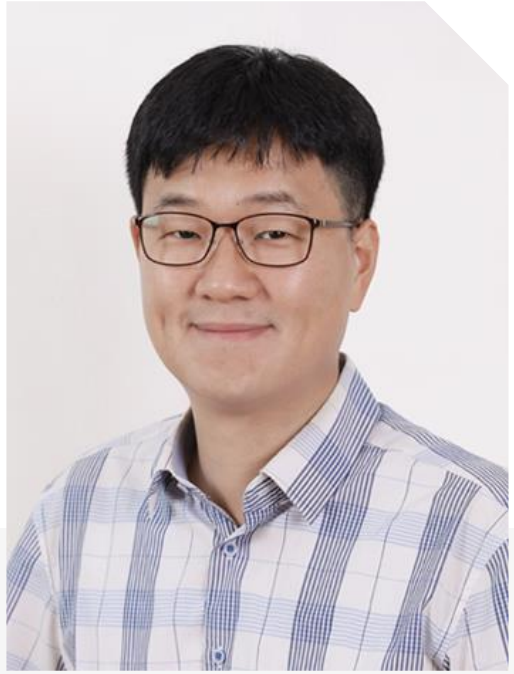
2023.08

BLAZOR 밋업

Wi-Fi : Blueport_Edu
PWD : BLUEport#$56

Sessions

- Follow-up
- WebAssembly를 만들어보자



Speaker

- 김 진석
- **(주)케이에이치시스템즈** 대표이사
- 한국블레이저개발자모임 / Blazor Korea
- facebook.com/iamjinseok.kr

Follow-up

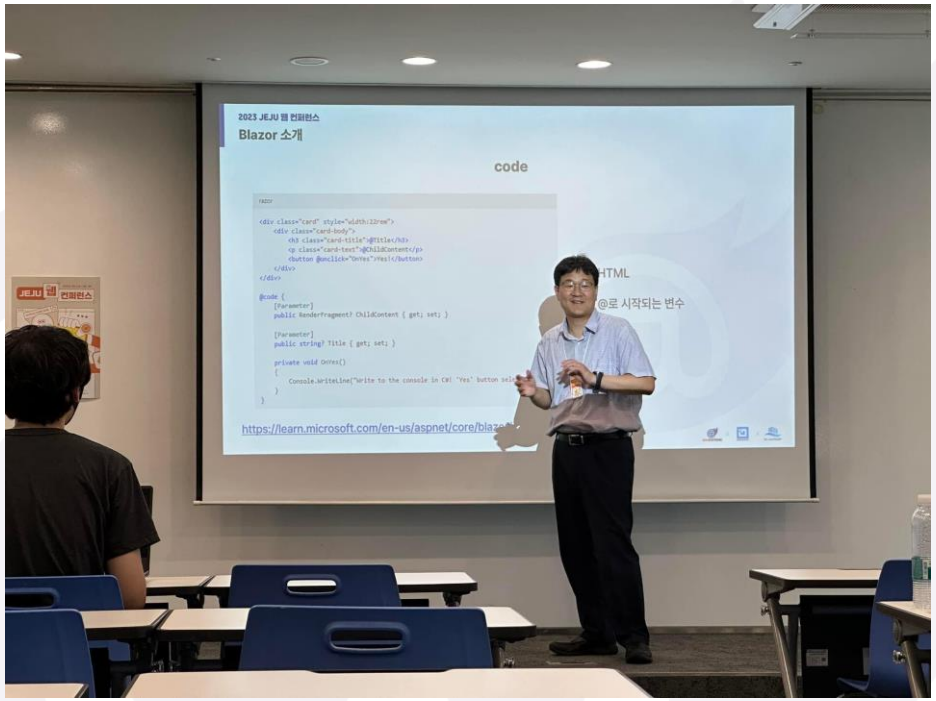
2023.08 Blazor 밋업

Blazor Korea 이벤트 기획

- 8/4-5 제주 웹 컨퍼런스 참여

2023.08 Blazor 밋업

Blazor Korea 이벤트 기획



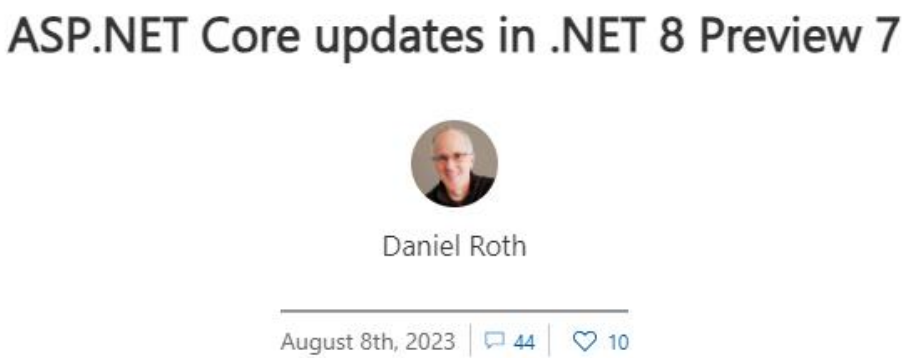
Blazor Korea 이벤트 기획

- 9월 23일(토)
- 한국마이크로소프트 13층
- 워크샵
- 경품 행사
- 온오프믹스에서 신청 (점심식사 제공)

https://github.com/blazorstudy/blazor-workshop

.NET 8 Preview 7

.NET 8 preview 7

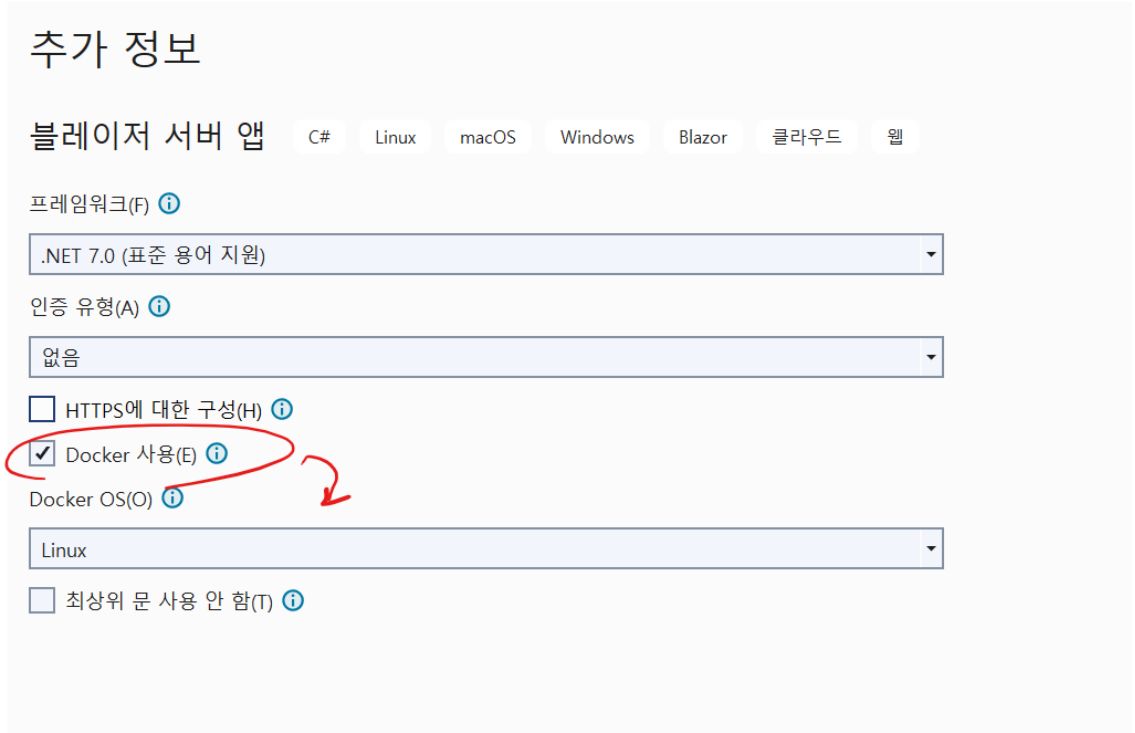


- Antiforgery integration
- Server-side form handling improvements
- Auto render mode
- Register root-level cascading values
- Improved integration of interactive components with server-side rendering
- New EmptyContent parameter for Virtualize

https://devblogs.microsoft.com/dotnet/asp-net-core-updates-in-dotnet-8-preview-7/#blazor

docker in Visual Studio

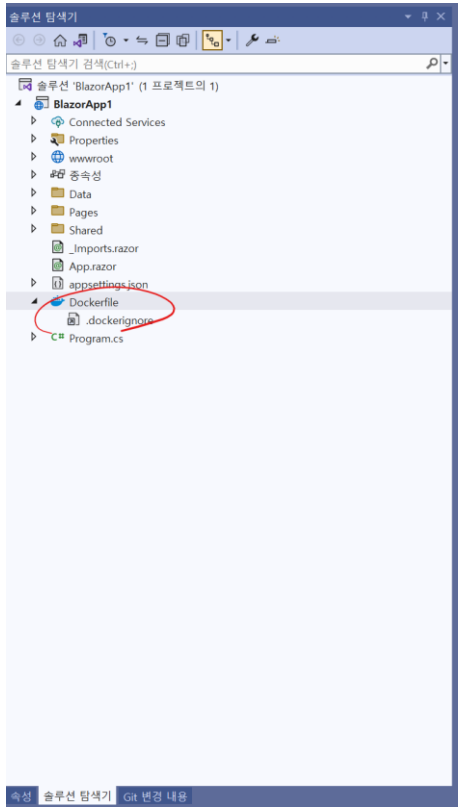
docker in Visual Studio



- 프로젝트 생성중 Docker 사용 선택
- Docker가 설치되어 있어야 함

docker in Visual Studio

docker in Visual Studio




```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["BlazorApp1/BlazorApp1.csproj", "BlazorApp1/"]
RUN dotnet restore "BlazorApp1/BlazorApp1.csproj"
COPY . .
WORKDIR "/src/BlazorApp1"
RUN dotnet build "BlazorApp1.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "BlazorApp1.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "BlazorApp1.dll"]
```

DevContainer Settings

DevContainer Settings

```jsonc
// For format details, see https://aka.ms/devcontainer.json. For config options, see the
// README at: https://github.com/devcontainers/templates/tree/main/src/dotnet
{
  "name": "C# (.NET)",
  // Or use a Dockerfile or Docker Compose file. More info: https://containers.dev/guide/dockerfile
  "image": "mcr.microsoft.com/devcontainers/dotnet:0-7.0",
  "features": {
    "ghcr.io/devcontainers/features/dotnet:1": {}
  }

  // Features to add to the dev container. More info: https://containers.dev/features.
  // "features": {},

  // Use 'forwardPorts' to make a list of ports inside the container available locally.
  // "forwardPorts": [5000, 5001],
  // "portsAttributes": {
  //    "5001": {
  //      "protocol": "https"
  //    }
  // }

  // Use 'postCreateCommand' to run commands after the container is created.
  // "postCreateCommand": "dotnet restore",

  // Configure tool-specific properties.
  // "customizations": {},

  // Uncomment to connect as root instead. More info: https://aka.ms/dev-containers-non-root.
  // "remoteUser": "root"
}
```

- name : UI에서 표시 되는 이름
- image : predefined image
- features : devcontainer에 추가하는 기능
- DockerFile
  - build > dockerfile 추가
- dockercompose
  - dockerComposeFile
  - service
  - workspaceFolder

https://containers.dev/implementors/json_reference/

Blazor 소개

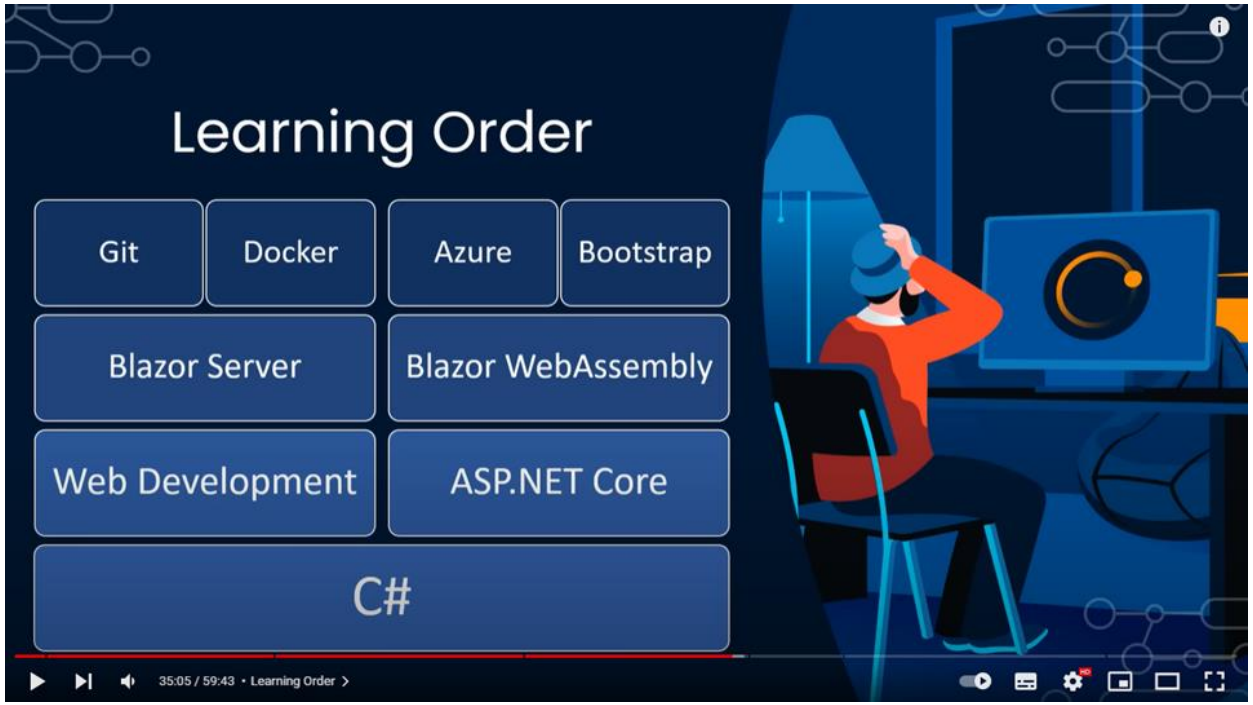
Blazor 특징

- .NET 및 C#의 기능을 사용하여
- JavaScript를 작성하지 않고
- 전체 스택 웹 앱(Full Stack Web App)을 빌드

https://dotnet.microsoft.com/ko-kr/apps/aspnet/web-apps/blazor

Blazor 소개

Blazor 기술 배경



- C# and .NET
- Single Page Application
- Minimal API
- Signal R
- WebAssembly
- BootStrap

https://www.youtube.com/watch?v=GIcAADkInRo

Blazor 소개

C# and .NET

C#

- 객체지향 프로그래밍 언어
- 문법적인 특성이 JAVA와 유사
- C# 11

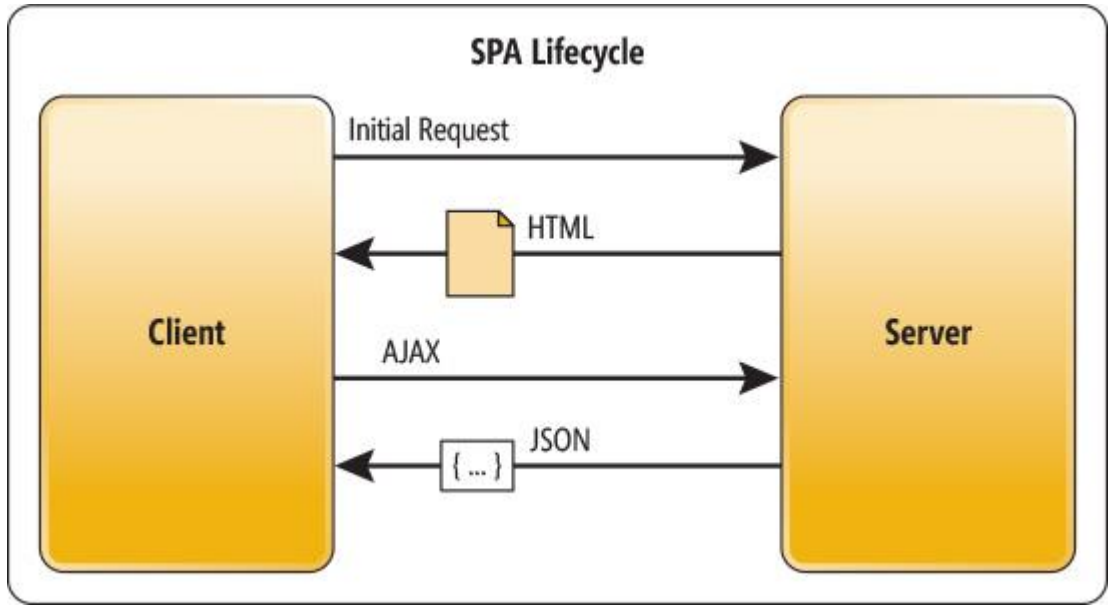
.NET

- 오픈소스 개발자 플랫폼
- 웹, 모바일, 데스크톱, 게임, IoT 등을 빌드
- 크로스플랫폼(일부)

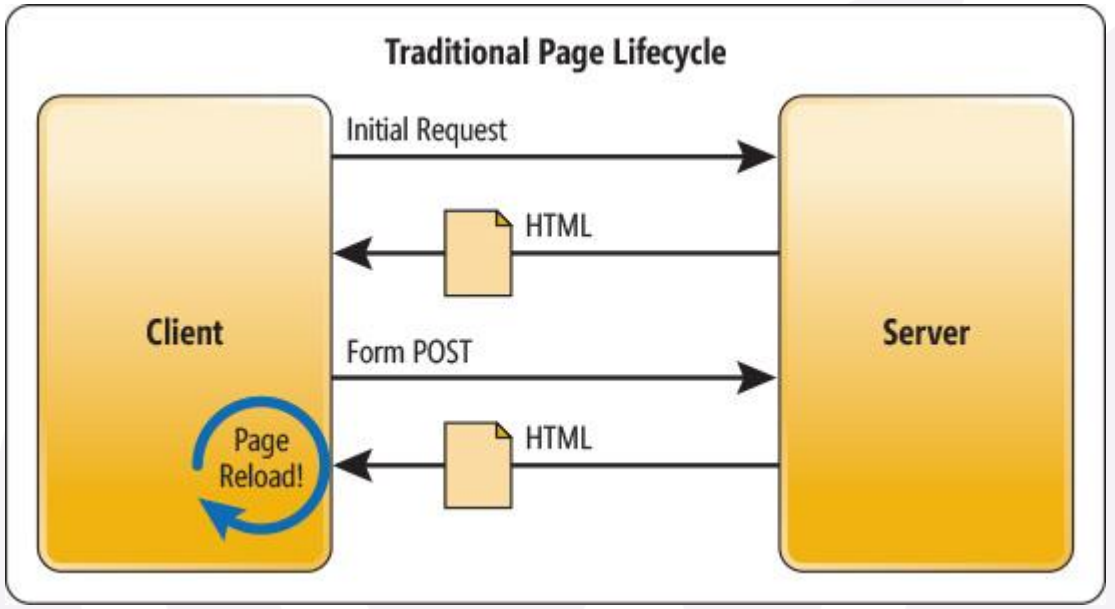
https://learn.microsoft.com/ko-kr/dotnet/
https://learn.microsoft.com/ko-kr/dotnet/csharp/

Blazor 소개

Single Page Application



SPA

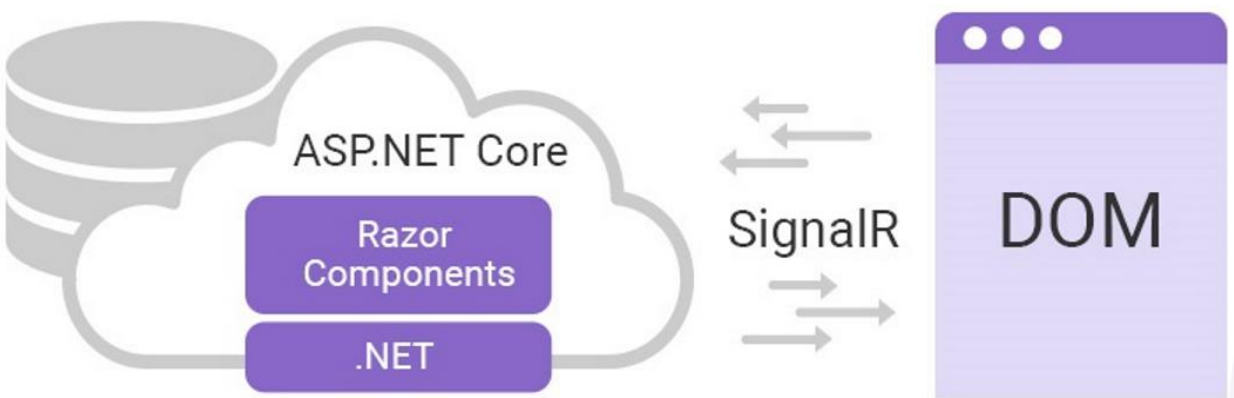


MPA

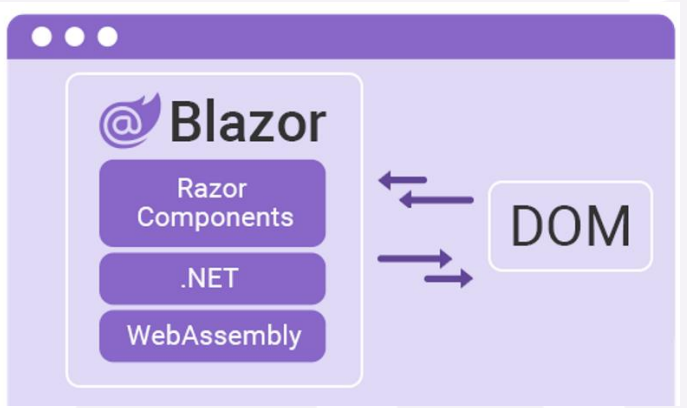
https://velog.io/@gwanuuoo/

Blazor 소개

Single Page Application



Blazor Server

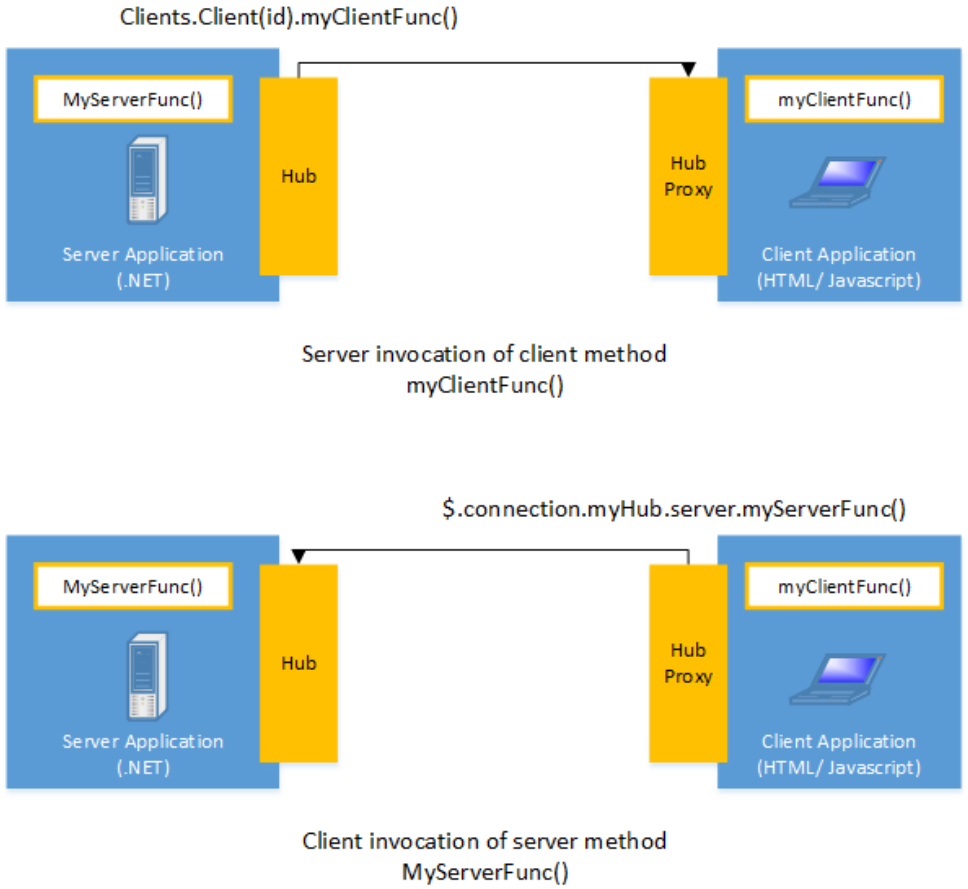


Blazor WebAssembly

https://www.infragistics.com/community/blogs/b/jason_beres/posts/blazor-vs-react-comparison

Blazor 소개

SignalR

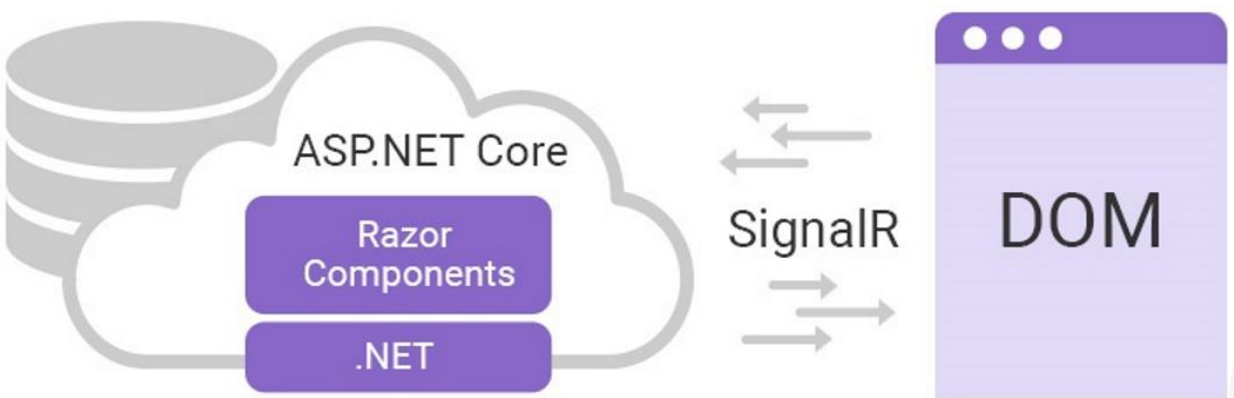


- 실시간 웹기능을 추가하기 위한 라이브러리
- 서버에서 연결된 클라이언트에 컨텐츠를 푸시
- 채팅, 페이지 새로고침, 긴 폴링
- 대시보드, 모니터링, 공동 작업 어플리케이션

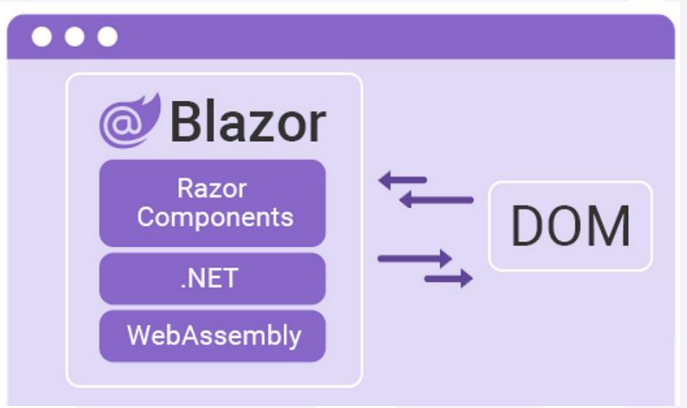
https://learn.microsoft.com/ko-kr/aspnet/signalr/

Blazor 소개

Single Page Application



Blazor Server

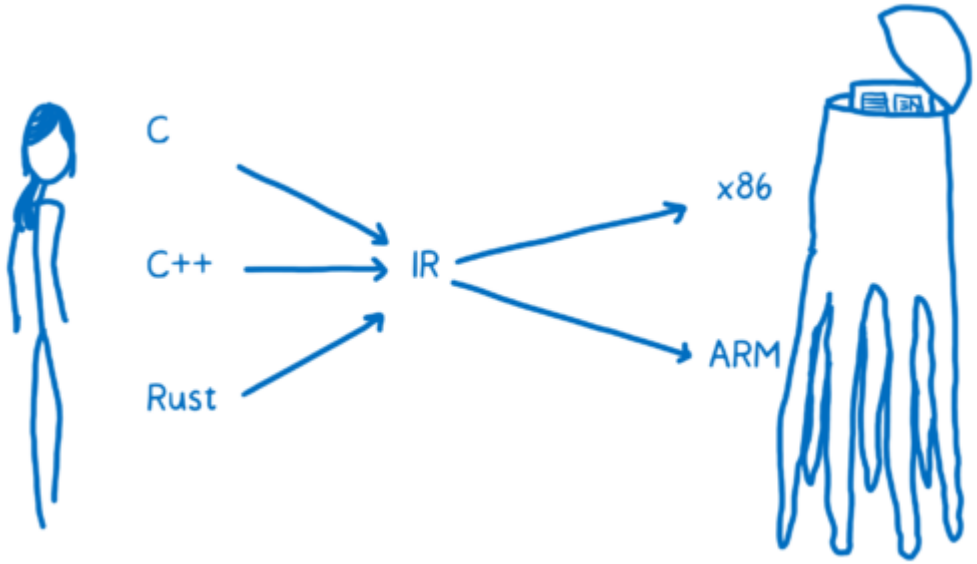


Blazor WebAssembly

https://www.infragistics.com/community/blogs/b/jason_beres/posts/blazor-vs-react-comparison

Blazor 소개

WebAssembly



- 2017년 처음 발표
- 브라우저에서 실행되는 바이트 코드
- 최신 브라우저 지원

https://dongwoo.blog/

Blazor 소개

code

razor

```
<div class="card" style="width:22rem">
    <div class="card-body">
        <h3 class="card-title">@Title</h3>
        <p class="card-text">@ChildContent</p>
        <button @onclick="OnYes">Yes!</button>
    </div>
</div>

@code {
    [Parameter]
    public RenderFragment? ChildContent { get; set; }

    [Parameter]
    public string? Title { get; set; }

    private void OnYes()
    {
        Console.WriteLine("Write to the console in C#! 'Yes' button selected.");
    }
}
```

- HTML
- @로 시작되는 변수
- C# 코드

https://learn.microsoft.com/en-us/aspnet/core/blazor/

Blazor 소개

BootStrap

Examples

Bootstrap includes several predefined button styles, each serving its own semantic purpose, with a few extras thrown in for more control.

Primary Secondary Success Danger Warning Info Light Dark Link

```
<Button Color="ButtonColor.Primary"> Primary </Button>
<Button Color="ButtonColor.Secondary"> Secondary </Button>
<Button Color="ButtonColor.Success"> Success </Button>
<Button Color="ButtonColor.Danger"> Danger </Button>
<Button Color="ButtonColor.Warning"> Warning </Button>
<Button Color="ButtonColor.Info"> Info </Button>
<Button Color="ButtonColor.Light"> Light </Button>
<Button Color="ButtonColor.Dark"> Dark </Button>
<Button Color="ButtonColor.Link"> Link </Button>
```

- CSS 프레임워크
- 하나의 코드로 다양한 기기에서 작동
- 최신 브라우저 지원
- 반드시 써야 하는 것은 아니에요!

https://getblazorbootstrap.com/

Blazor 소개

Minimal APIs

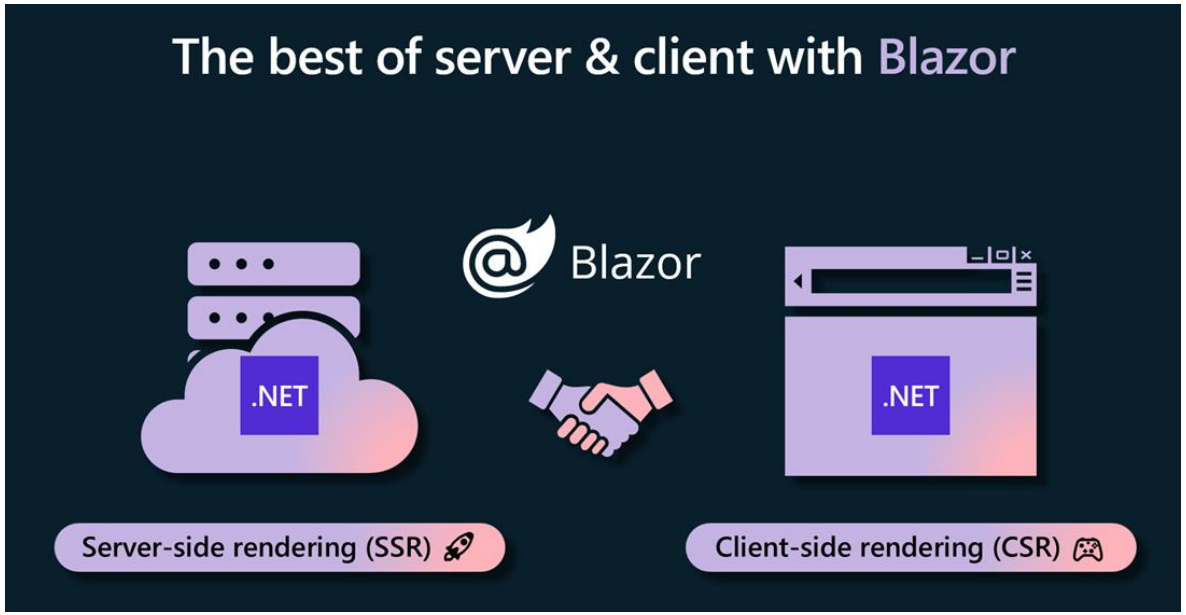
```C#
var app = WebApplication.Create(args);

app.MapGet("/", () => "Hello World!");

app.Run();
```

- RESTful API처럼 구현하고자 할 때
- ASP.NET Core의 기능

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/overview

Blazor 소개

.NET 8 이후



- Blazor Web App으로 통합

https://build.microsoft.com/en-US/sessions/da223f08-abb2-4f38-87de-0856ffa22317?source=sessions

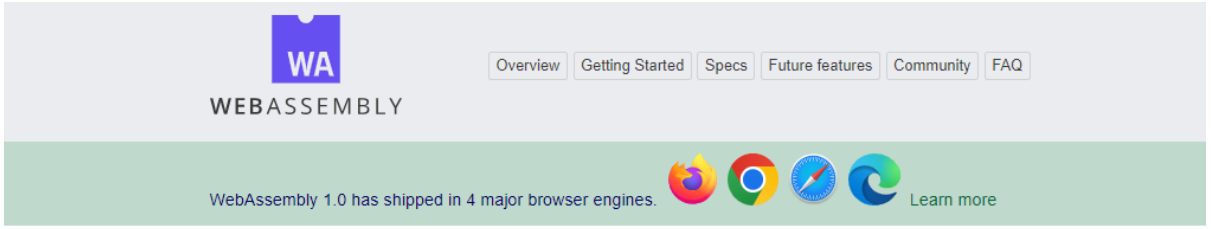
break
time

오늘의 도전!

WebAssembly를 만들어 보자 (알아보자)

WebAssembly를 만들어보자

WEBASSEMBLY



WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

Developer reference documentation for Wasm can be found on MDN's WebAssembly pages. The open standards for WebAssembly are developed in a W3C Community Group (that includes representatives from all major browsers) as well as a W3C Working Group.

Efficient and fast

The Wasm stack machine is designed to be encoded in a size- and load-time-efficient binary format. WebAssembly aims to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms.

Safe

WebAssembly describes a memory-safe, sandboxed execution environment that may even be implemented inside existing JavaScript virtual machines. When embedded in the web, WebAssembly will enforce the same-origin and permissions security policies of the browser.

Open and debuggable

WebAssembly is designed to be pretty-printed in a textual format for debugging, testing, experimenting, optimizing, learning, teaching, and writing programs by hand. The textual format will be used when viewing the source of Wasm modules on the web.

Part of the open web platform

WebAssembly is designed to maintain the versionless, feature-tested, and backwards-compatible nature of the web. WebAssembly modules will be able to call into and out of the JavaScript context and access browser functionality through the same Web APIs accessible from JavaScript. WebAssembly also supports non-web embeddings.

- Efficient and fast
- Safe
- Open and debuggable
- Part of the open web platform

https://webassembly.org/

WebAssembly를 만들어보자

웹어셈블리의 목표

- 빠르고, 효과적이고, 이식성이 좋을 것 : 네이티브에 가까운 속도
- 읽기 쉽고 디버깅이 가능할 것 : 사람이 읽을 수 있는 텍스트 포멧(ing...)
- 안전함을 유지할 것 : 샌드박싱된 실행환경에서 안전하게... same-origin
- 웹을 망가뜨리지 않을 것 : 다른 웹기술과 문제없이 하위호환

https://developer.mozilla.org/ko/docs/WebAssembly/Concepts

WebAssembly를 만들어보자

웹어셈블리가 뭔가요?

- 최신 웹 브라우저에서 실행할 수 있는 새로운 유형의 코드
- C, C++, RUST 등의 저레벨 소스 언어를 효과적으로 컴파일하도록 고안
- 웹에서 여러 언어로 작성된 코드를 네이티브에 가까운 속도로 실행하는 길을 제공

https://developer.mozilla.org/ko/docs/WebAssembly/Concepts

WebAssembly를 만들어보자

이 그림 저작권???

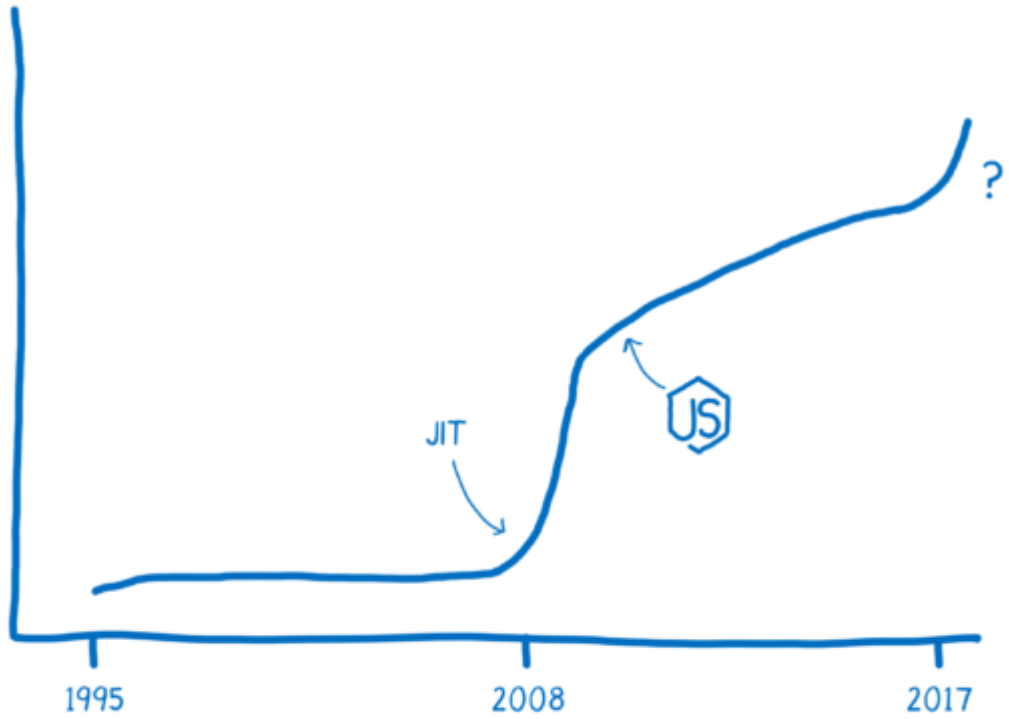
WebAssembly를 만들어보자

- 카툰으로 소개하는 웹어셈블리
- 저스트-인-타임(JIT)컴파일러 집중코스
- 어셈블리 집중코스
- 웹어셈블리 모듈의 생성과 동작
- 웹어셈블리는 왜 빠를까?
- 웹어셈블리의 현재 위치와 미래

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

일단 알아야 될 히스토리



- 1995년 JavaScript 탄생
- 2008년 저스트 인 타임(Just In Time)

  컴파일러 장착
- JIT Compiler 전후로 10배 이상의 속도차이

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

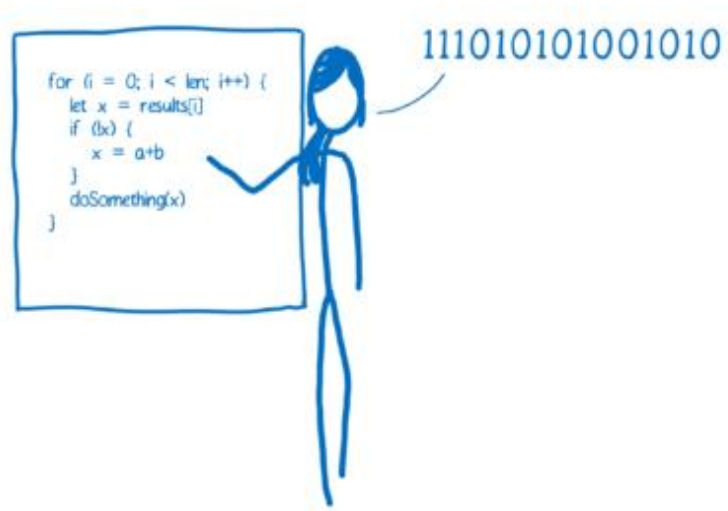
저스트-인-타임(JIT) 컴파일러에 집중코스

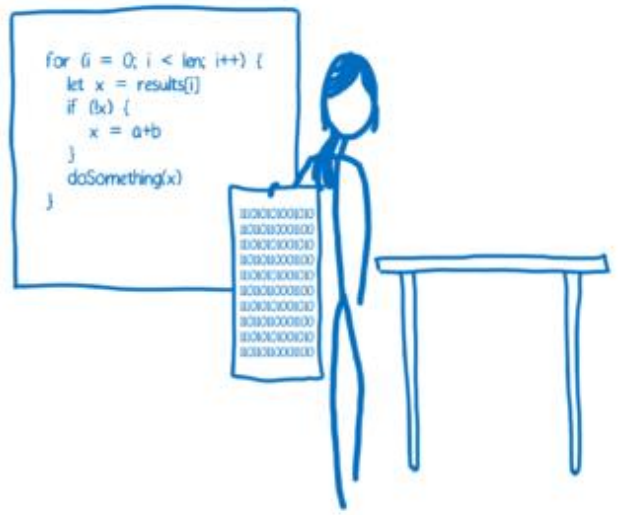
- 컴퓨터 기계어(바이너리 코드) 사용
- 인간은 JavaScript
- JavaScript 엔진이 JavaScript를 기계어로 번역

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

저스트-인-타임(JIT) 컴파일러에 집중코스





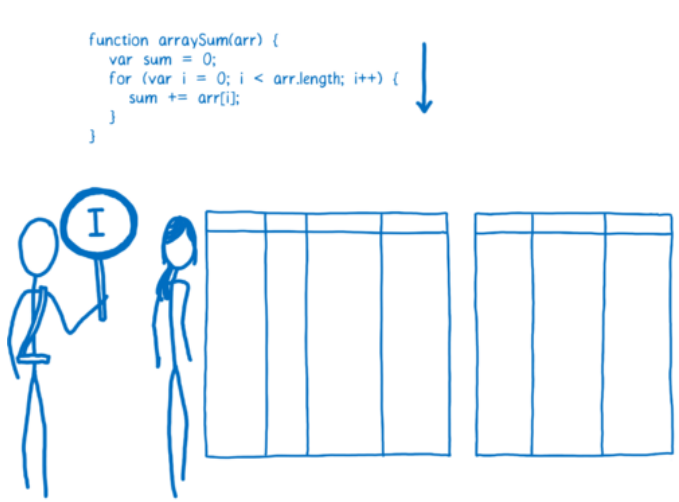
- 인터프린터
- 한줄 한줄 번역, 바로 시작
- 동일한 코드를 계속 실행하면 손해
- 초기 브라우저들이 사용

- 컴파일러
- 전체코드를 번역, 최적화
- 브라우저가 바로 보여줄 수 없음

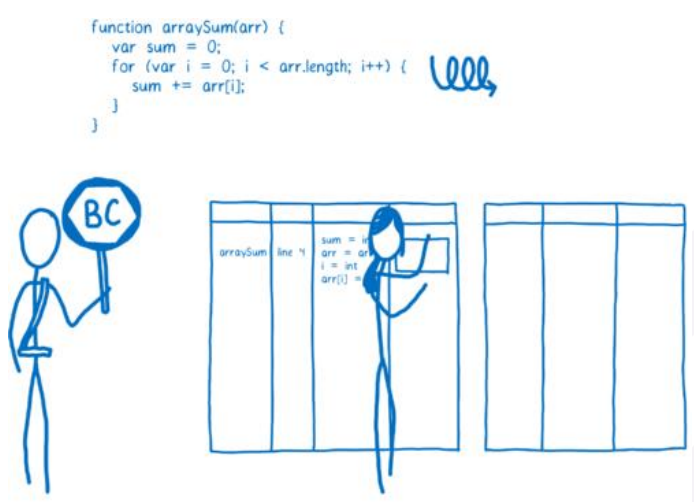
https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

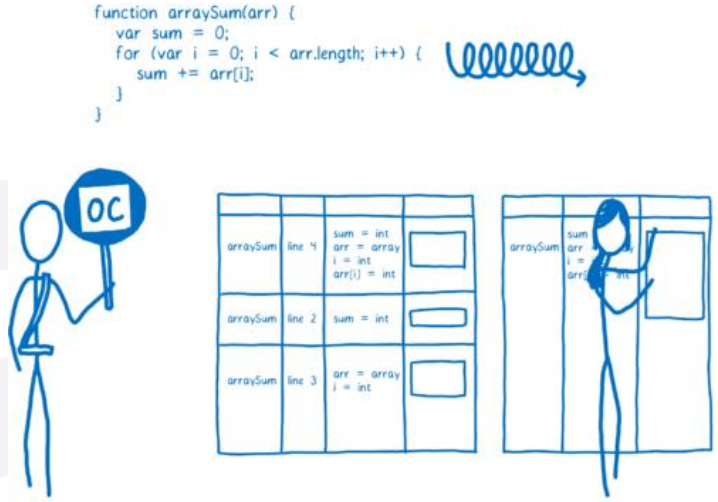
저스트-인-타임(JIT) 컴파일러에 집중코스



- 인터프린터(I)
- 일단 번역



- 기본 컴파일러(BC)
- 자주 쓰는 코드
- 컴파일하여 보관
- 재사용

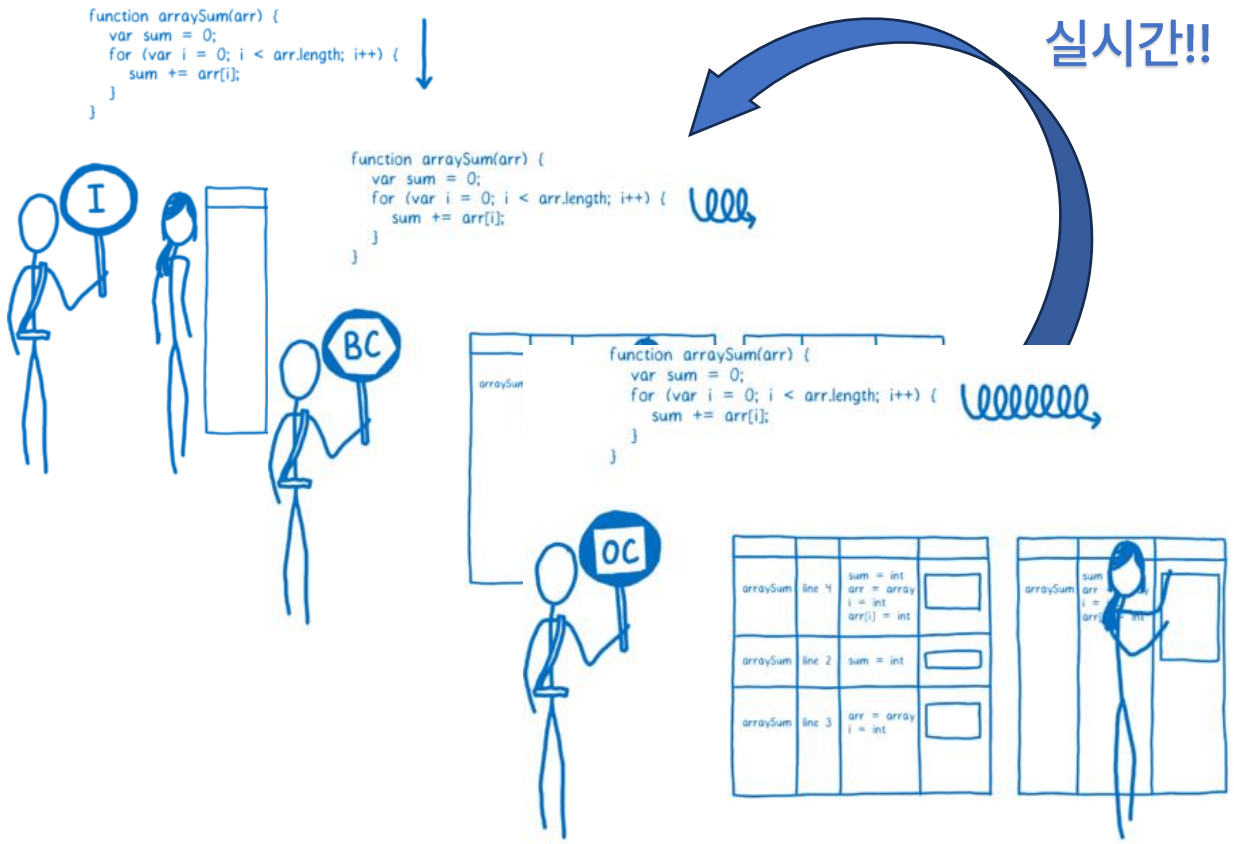


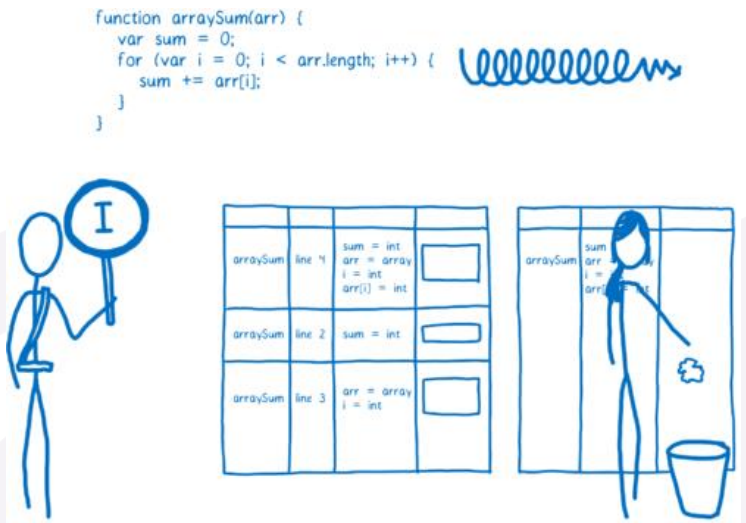
- 최적화 컴파일러(OC)
- 완전 자주 쓰는 코드
- 최적화 컴파일
- 재사용

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

저스트-인-타임(JIT) 컴파일러에 집중코스





- 최적화 가정이 틀리거나
- 자료형이 생각과 달랐거나
- 최적화가 아니라고 판단되면

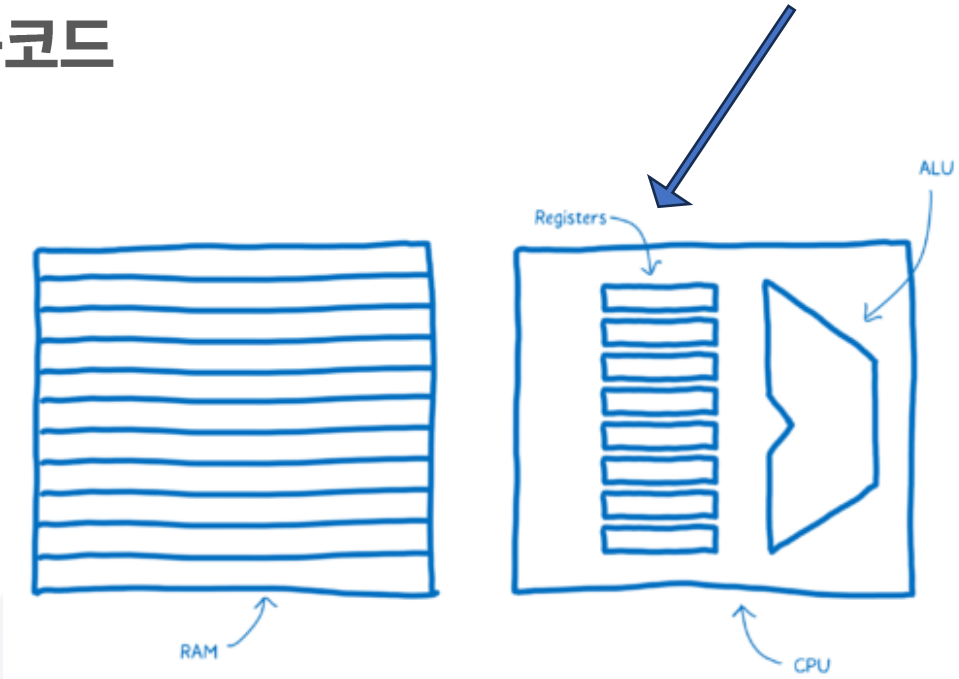
버린다

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

컴퓨터의 구조 수업시간이 생각남

어셈블리 집중코드



- 프로그래밍 언어랑 바이너리 코드랑 달라
- 왜 자꾸 이런 이야기???

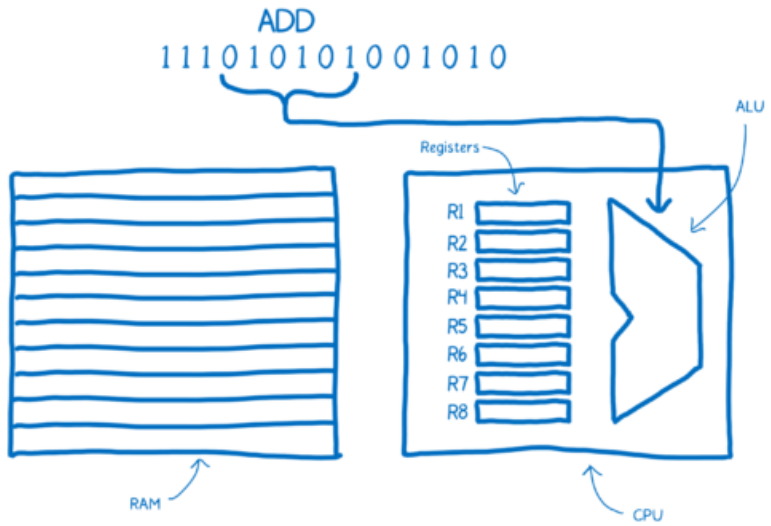
- 브러우저도 이렇게 동작
- ALU : 산술논리장치
- 레지스터
- Random Access Memory

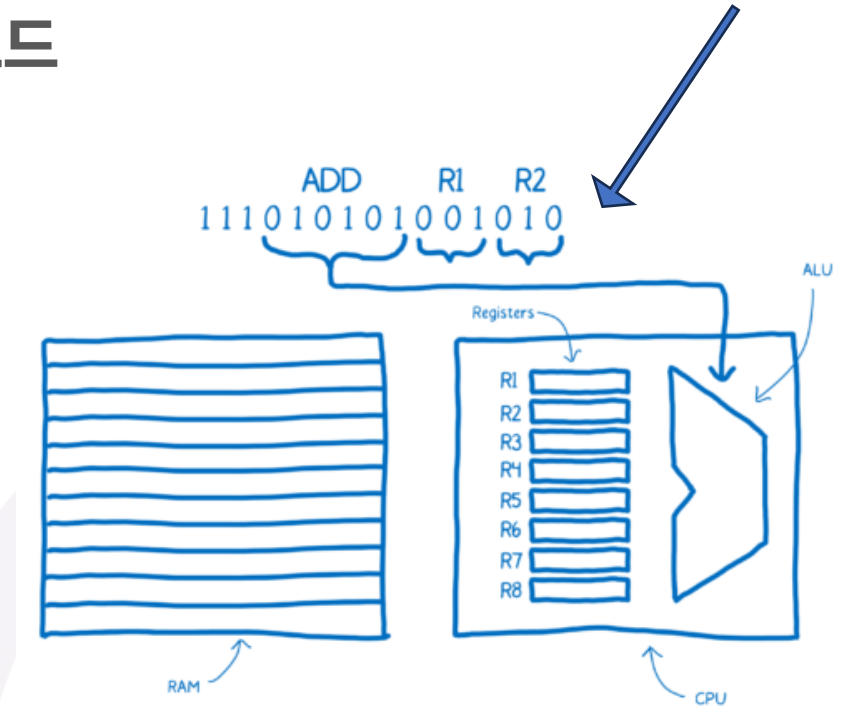
https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

컴퓨터의 구조 수업시간이 생각남

어셈블리 집중코드





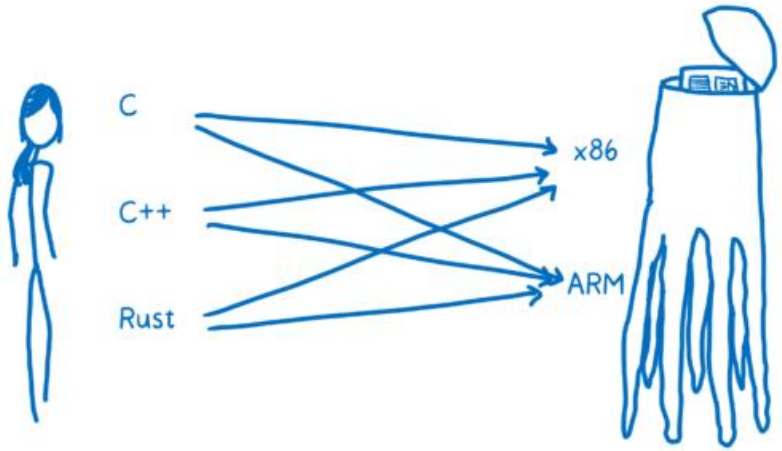
- ADD 명령어도 010101로 정해져 있음

- 더해야 하는 숫자 R1, R2의 바이트크기도 정해져있음
- 하드웨어에 종속적임(어셈블리니까)

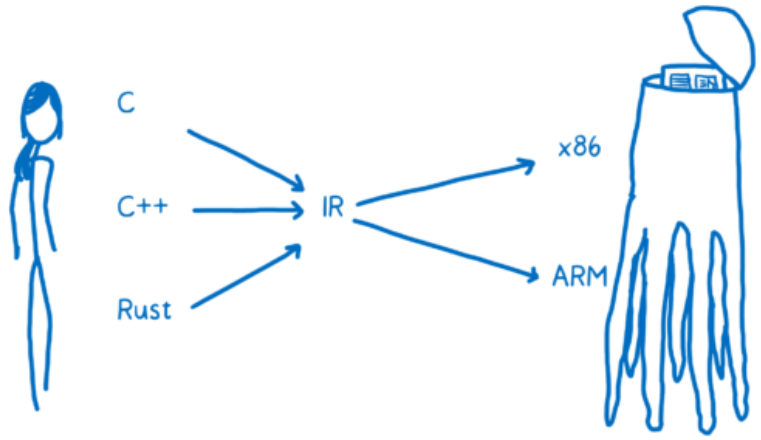
https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

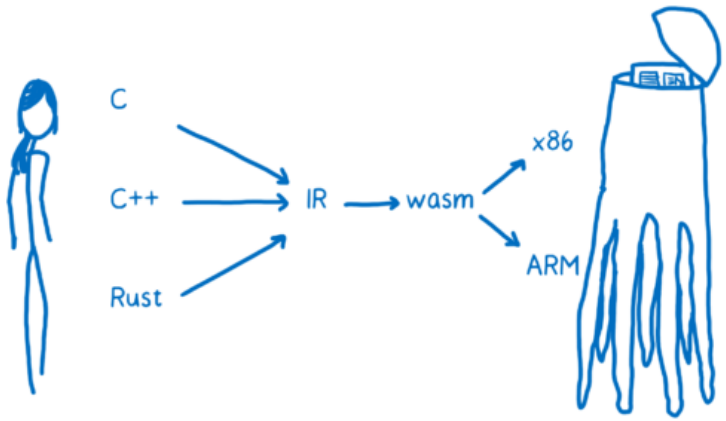
웹어셈블리 모듈의 생성과 동작



각 언어를 하드웨어별로 만들면 복잡



중간 언어를 만들면 유연

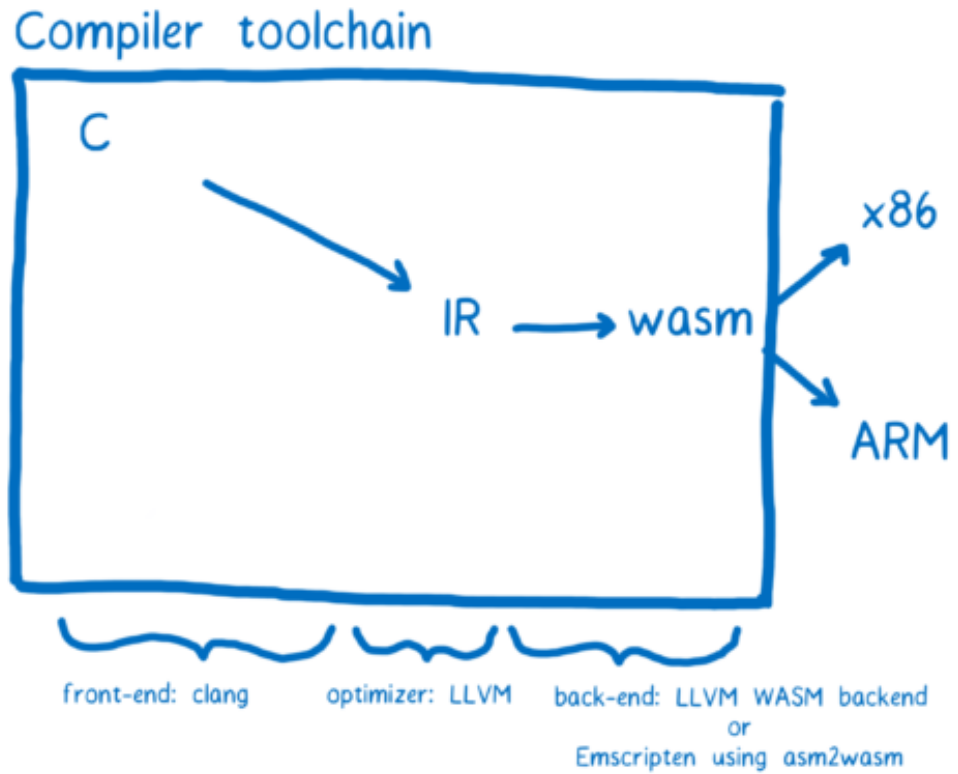


중간언어를 wasm 모듈로 생성

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

웹어셈블리 모듈의 생성과 동작



- 옆의 그림처럼 만드는 것
- http://mbebenita.github.io/WasmExplorer/
- 한눈에 분위기를 느껴보자

```
int add42(int a){  return a + 42;}
```

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

WebAssembly를 만들어보자

웹어셈블리는 왜 빠를까?

- 자바스크립트보다 더 간결하다.
- 해독(decode)하는 시간이 구문을 해석 (parse) 하는 시간보다 빠르다.
- 이미 한 번 최적화하여 준비된 코드이다.
- JIT 컴파일러가 컴파일-최적화하는 단계가 필요없다.
- 머신에 적합한 명령어셋으로 구성되어 있다.
- 웹어셈블리에서 메모리를 직접 관리하도록 되어 있다.

https://hacks.mozilla.org/2017/02/a-cartoon-intro-to-webassembly/
https://dongwoo.blog/2017/06/06/번역-카툰으로-소개하는-웹어셈블리/

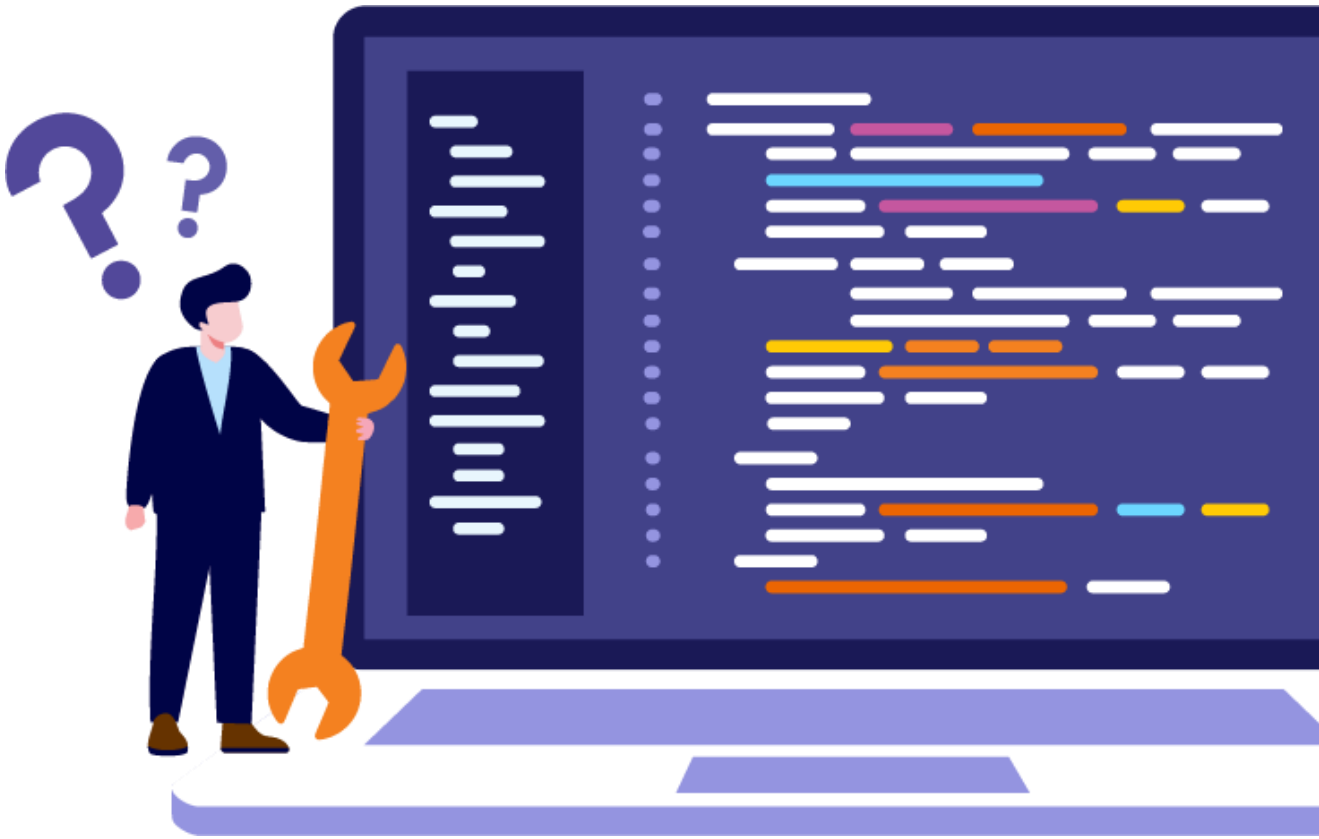
Q&A

Blazor Korea User Group

facebook.com/groups/blazor.korea

GitHub

https://github.com/blazorstudy



오늘 내용???

Code가 없었는데???

9월 일정

다음 달(9월) 일정

9월 23일(토) 워크샵!!!

THANK YOU

KH@SYSTEMS | INFRAGISTICS | BLUEPORT