Why you should consider Web Assembly in your next frontend project

Håkan Silfvernagel

Miles

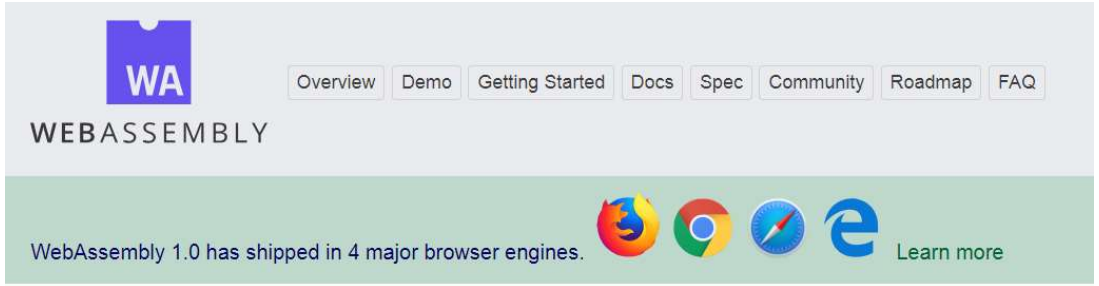
Who Am I?

- Background in software development for process automation and robotics
- Worked on the .NET platform since its release 2001-2002
- Consultant since 2010
- 19 years' experience of software development in various positions such as developer, tester, architect, project manager, scrum master, practice manager and team lead.
- MIT Artificial Intelligence: Implications for business strategy
- MIT Internet of Things: Business implications and opportunities

What is Web Assembly?

Web Assembly

- Group formed in 2015 (all major browsers)
- 2017, Nov, Web Assembly Released
- 2018, Feb, W3C public draft published



WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

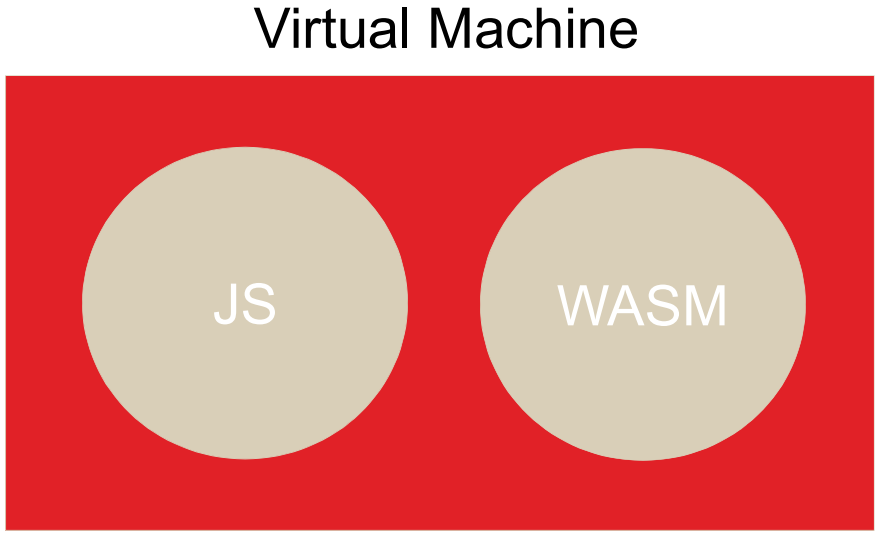
Developer reference documentation for Wasm can be found on MDN's WebAssembly pages. The open standards for WebAssembly are developed in a W3C Community Group (that includes representatives from all major browsers) as well as a W3C Working Group.

Source: https://webassembly.org/

Web platform as virtual machine

- JavaScript
  - High level language
  - Flexible and expressive
  - Dynamically typed
  - No compilation
  - Huge ecosystem

- WebAssembly (WASM)
  - Low level assembly like language
  - Compact binary format
  - Near native performance
  - C, C++, Rust, Go…



Wasm

- A new browser standard
- Based on asm.js
- Intermediate language
  - Java (bytecode), C# (msil)
- Strongly typed
  - i32, i64, f32, f64

Hello WASM

WASM example (1)

add.wat

```
(module
(func $addTwo (param i32 i32) (result i32)
get_local 0
get_local 1
i32.add)
(export "addTwo" (func $addTwo)))
```

WASM example (2)

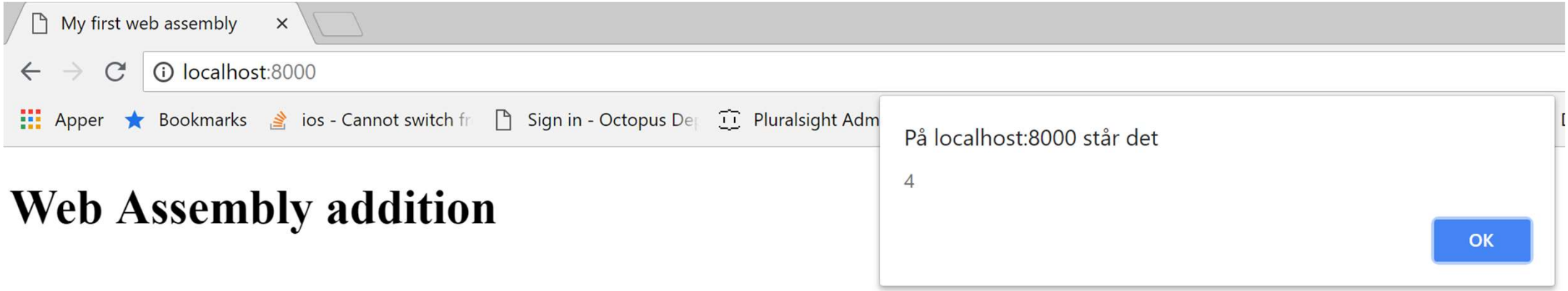
wat2wasm add.wat -o add.wasm
wat2wasm add.wat -v

```
0000000: 0061 736d                 ; WASM_BINARY_MAGIC
0000004: 0100 0000                 ; WASM_BINARY_VERSION
; section "Type" (1)
0000008: 01                        ; section code
0000009: 00                        ; section size (guess)
000000a: 01                        ; num types
; type 0
000000b: 60                        ; func
000000c: 02                        ; num params
000000d: 7f                        ; i32
000000e: 7f                        ; i32
000000f: 01                        ; num results
0000010: 7f                        ; i32
0000009: 07                        ; FIXUP section size
; section "Function" (3)
0000011: 03                        ; section code
0000012: 00                        ; section size (guess)
0000013: 01                        ; num functions
0000014: 00                        ; function 0 signature index
0000012: 02                        ; FIXUP section size
; section "Export" (7)
0000015: 07                        ; section code
0000016: 00                        ; section size (guess)
0000017: 01                        ; num exports
0000018: 06                        ; string length
0000019: 6164 6454 776f                 addTwo  ; export name
000001f: 00                        ; export kind
0000020: 00                        ; export func index
0000016: 0a                        ; FIXUP section size
; section "Code" (10)
0000021: 0a                        ; section code
0000022: 00                        ; section size (guess)
0000023: 01                        ; num functions
; function body 0
0000024: 00                        ; func body size (guess)
0000025: 00                        ; local decl count
0000026: 20                        ; get_local
0000027: 00                        ; local index
0000028: 20                        ; get_local
0000029: 01                        ; local index
000002a: 6a                        ; i32.add
000002b: 0b                        ; end
0000024: 07                        ; FIXUP func body size
0000022: 09                        ; FIXUP section size
```

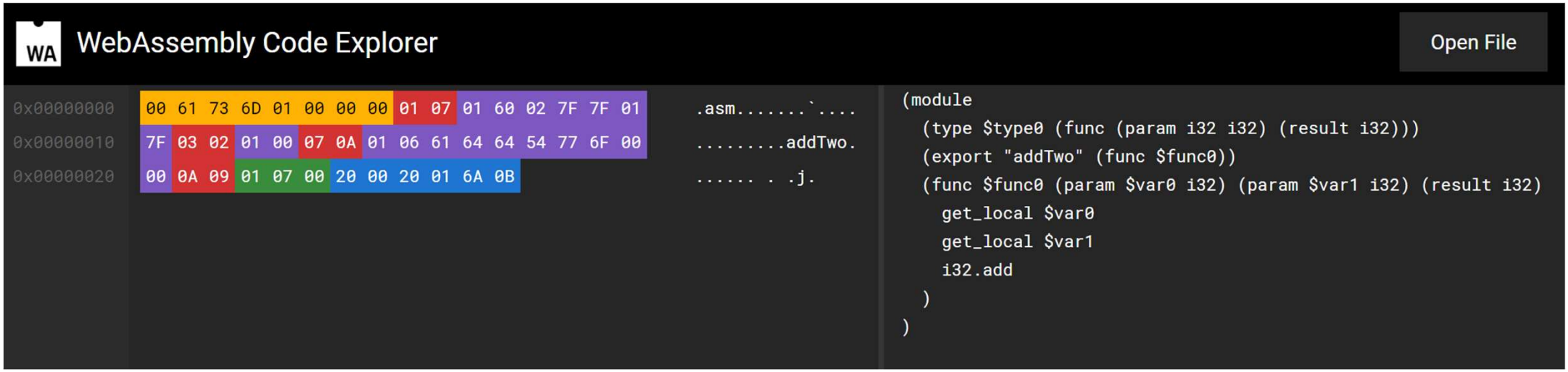
WASM example (3)

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>My first web assembly</title>
<script>
      fetch("add.wasm").then(response =>
            response.arrayBuffer()
      ).then(bytes =>
            WebAssembly.instantiate(bytes, {})
      ).then(results => {
            let addTwo = results.instance.exports.addTwo;
            alert(addTwo(2,2));
      });
</script>
</head>
<body>
<h1>Web Assembly addition</h1>
</body>
```

WASM example (4)



WASM example (5)

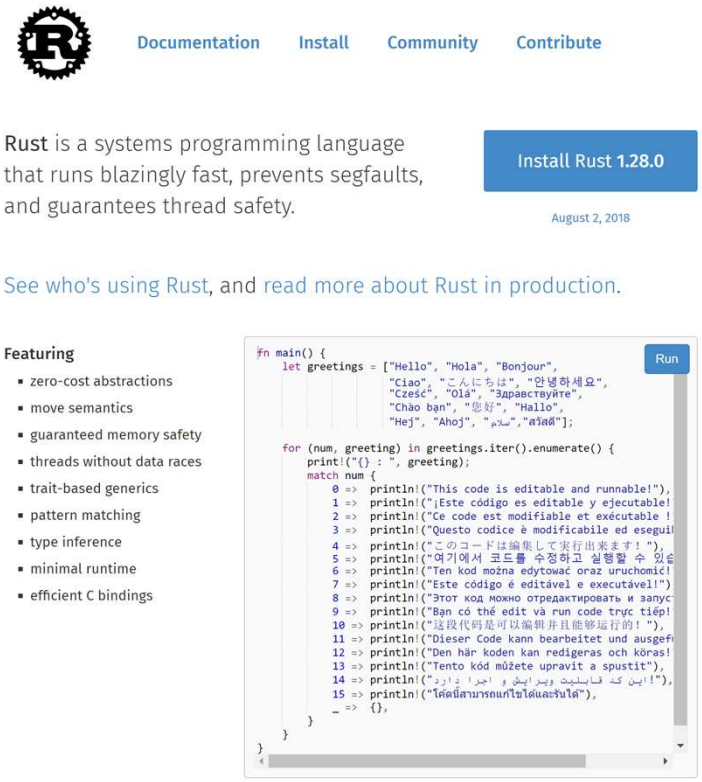


https://wasdk.github.io/wasmcodeexplorer/

An introduction to Rust

What is Rust?

- From Mozilla
- System programming language
- Syntax similar to C/C++
- Strong typed
- Compile to native code (no VM)
- Memory safe guarantee (no GC)



Rust toolchain rustup, cargo & rustc

rustup is an installer for

the systems programming language **Rust**

Cargo is the Rust *package manager*. Cargo downloads your Rust project's dependencies, compiles your project, makes packages, and upload them to crates.io, the Rust community's *package registry*.

```
fn main() { println!("Hello, world!"); }
```

```
> rustc main.rs
> .\main.exe Hello, world!
```

Using cargo

```
> cargo new hello_world
     Created binary (application) `hello_world` project
```

main.rs

```
fn main() {
println!("Hello, world!");
}
```

Cargo.toml

```
[package]
name = "hello_world"
version = "0.1.0"
authors = ["Hakan Silfvernagel <abbhakan@gmail.com>"]

[dependencies]
```

```
> cargo run
   Compiling hello_world v0.1.0 (file:///C:/source/webassembly/hello_world)
    Finished dev [unoptimized + debuginfo] target(s) in 1.93s
     Running `target\debug\hello_world.exe`
Hello, world!
```

Types in Rust

- Primitive Types
  - Boolean bool (true/false)
  - Integer/float
  - Textual type char/str
  - Never type !
- Primitive constructs
  - Tuples
  - Arrays
  - Slices
  - Function pointers
  - References
  - Pointers

Rust -> WASM and publish to NPM

How to compile rust programs to wasm

1. Install Rust toolchain (rustup, cargo & rustc)
2. Add the wasm32-unknown-unknown target to compile to web assembly
3. Install wasm-bindgen for bindings to/from JS for Rust and wasm

source: https://rustwasm.github.io/wasm-bindgen/

Rust example (1)

Cargo.toml

```
[package]
name = "hello-wasm-silver"
version = "0.1.0"
authors = ["Hakan Silfvernagel <abbhakan@gmail.com>"]
description = "Code used to demonstrate how to use wasm-pack"
license = "MIT/Apache-2.0"
repository = "https://github.com/abbhakan/hello-wasm-silver"

[lib]
crate-type = ["cdylib"]

[dependencies]
wasm-bindgen = "0.2.13"
```

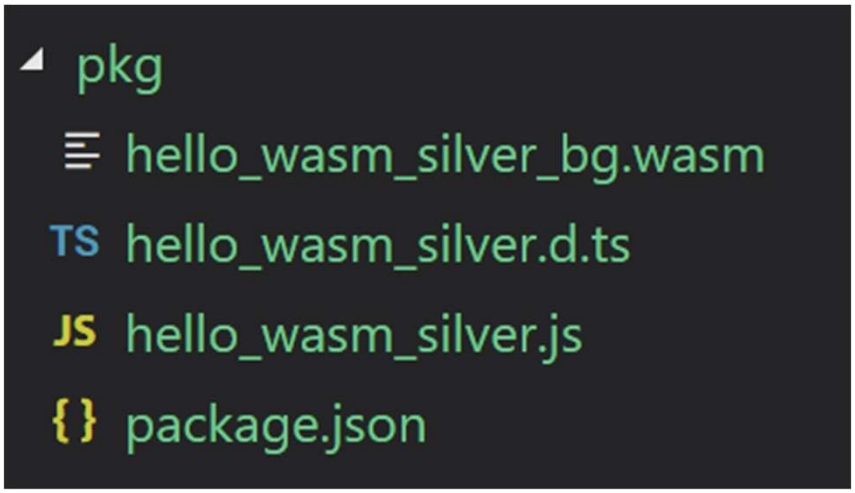
Rust example (2)

src/lib.rs

```
#![feature(proc_macro)]

extern crate wasm_bindgen;

use wasm_bindgen::prelude::*;

#[wasm_bindgen]
extern {
        fn alert(s: &str);
}

#[wasm_bindgen]
pub fn greet(name: &str) {
        alert(&format!("Hello, {}!", name));
}
```

Rust example (3)

wasm-pack init



cd pkg
npm publish package

Rust example (4)

hello-wasm-silver.d.ts

```
/* tslint:disable */
export function greet(arg0: string): void;
```

hello-wasm-silver.js

```
/**
* @param {string} arg0
* @returns {void}
*/
export function greet(arg0) {
const [ptr0, len0] = passStringToWasm(arg0);
try {
return wasm.greet(ptr0, len0);
} finally {
wasm.__wbindgen_free(ptr0, len0 * 1);
}
}
```

package.json

```
{
"name": "hello-wasm-silver",
"collaborators": [
"Hakan Silfvernagel <abbhakan@gmail.com>"
],
"description": "Code used to demonstrate how to use wasm-pack",
"version": "0.1.0",
"license": "MIT/Apache-2.0",
"repository": {
"type": "git",
"url": "https://github.com/abbhakan/hello-wasm-silver"
},
"files": [
"hello_wasm_silver_bg.wasm",
"hello_wasm_silver.d.ts"
],
"main": "hello_wasm_silver.js",
"types": "hello_wasm_silver.d.ts"
}
```

Rust example (5): Use the package

webpack.config.js

```
const path = require('path');
module.exports = {
      entry: "./index.js",
      output: {
      path: path.resolve(__dirname, "dist"),
      filename: "index.js",
      },
      mode: "development"
};
```

package.json

```
{
      "scripts": {
            "serve": "webpack-dev-server"
      },
      "dependencies": {
            "hello-wasm-silver": "^0.1.0"
      },
      "devDependencies": {
            "copy-webpack-plugin": "^4.5.2",
            "webpack": "^4.16.1",
            "webpack-cli": "^3.1.0",
            "webpack-dev-server": "^3.1.4"
      }
}
```

Rust example (6)

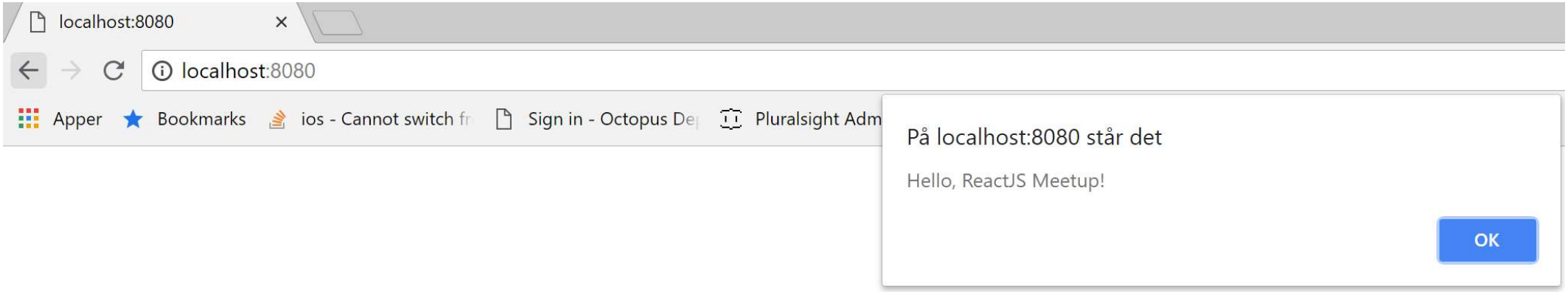
index.html

```
<html>
<head>
<meta content="text/html;charset=utf-8" http-
equiv="Content-Type"/>
</head>
<body>
<script src='./index.js'></script>
</body>
</html>
```

index.js

```
const js = import("hello-wasm-silver");
js.then(js => {
js.greet("ReactJS Meetup");
});
```

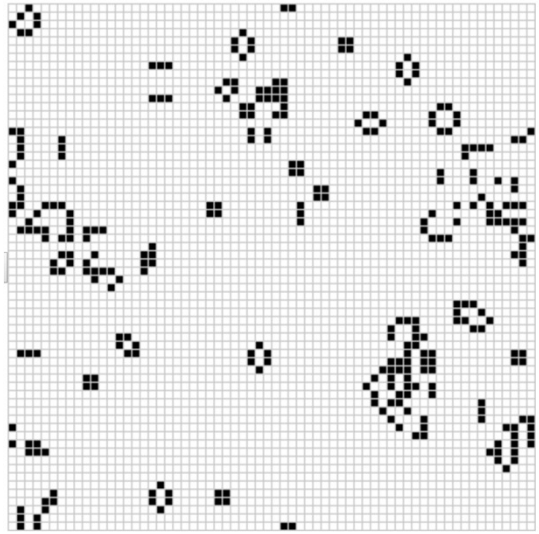
Rust example (7)



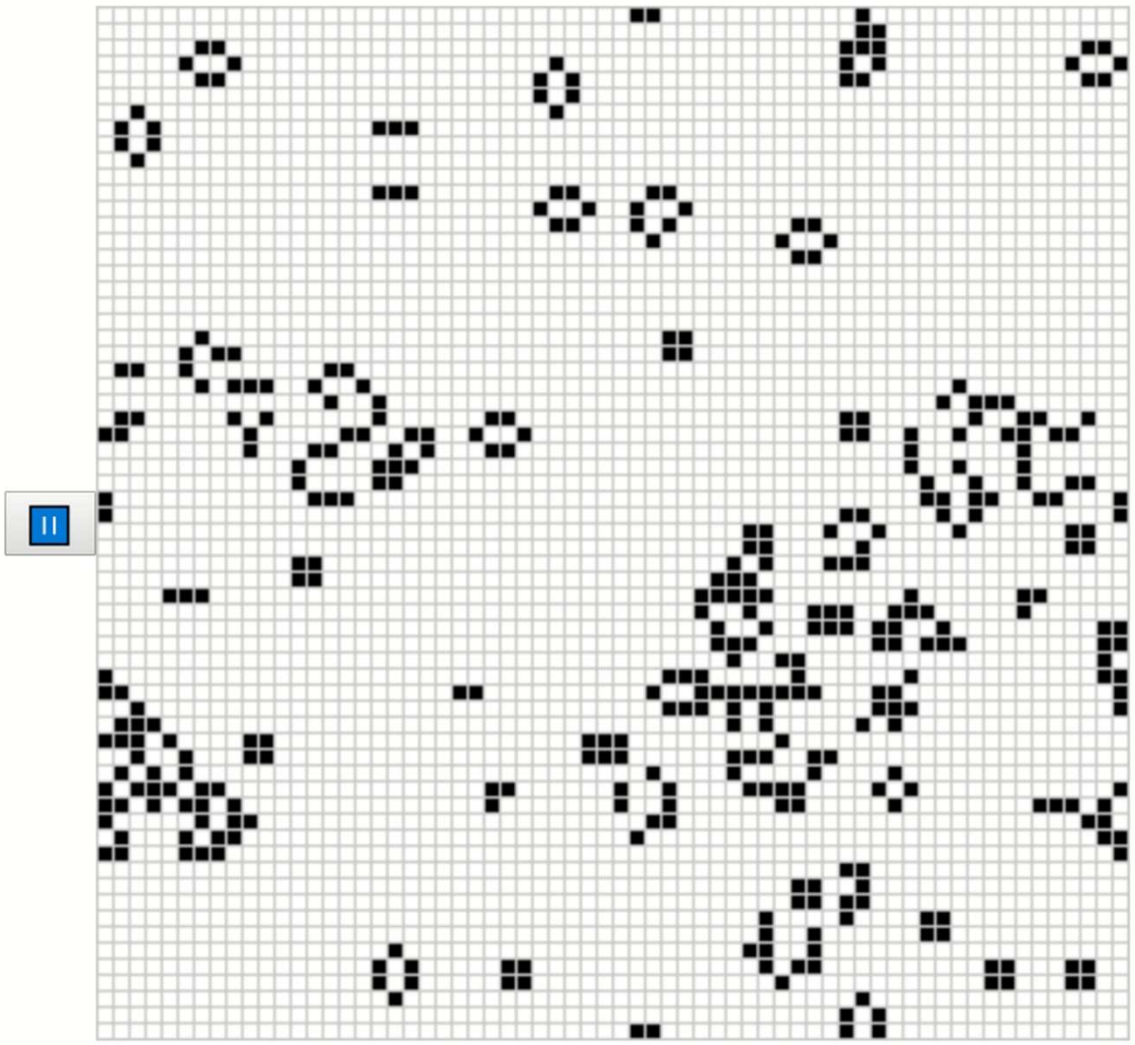
Game of life

Game of life

1.Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
2.Any live cell with two or three live neighbours lives on to the next generation.
3.Any live cell with more than three live neighbours dies, as if by overpopulation.
4.Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.



Source: https://rustwasm.github.io/book/game-of-life/introduction.html



Code overview: cargo.toml

```
[package]
name = "wasm_game_of_life"
version = "0.1.0"
authors = ["Hakan Silfvernagel <abbhakan@gmail.com>"]

[lib]
crate-type = ["cdylib"]

[dependencies]
wasm-bindgen = "0.2.13"

[profile.release]
# Include function names in the `.wasm` for better debugging and
# profiling. Comment this out if you're trying to create the smallest `.wasm`
# binaries you can.
debug = true
```

Code overview: lib.rs

```
pub fn new() -> Universe {
      let width = 64;
      let height = 64;

      let cells = (0..width * height)
      .map(|i| {
      if i % 2 == 0 || i % 7 == 0 {
            Cell::Alive
      } else {
            Cell::Dead
      }
})
.collect();

Universe {
      width,
      height,
      cells,
      }
}
```

```
pub fn width(&self) -> u32 {
self.width
}

pub fn height(&self) -> u32 {
self.height
}

pub fn cells(&self) -> *const Cell {
self.cells.as_ptr()
}

pub fn toggle_cell(&mut self, row: u32, column: u32)
{
let idx = self.get_index(row, column);
self.cells[idx].toggle();
}
```

Code overview: wasm_game_of_life.d.ts

```
/* tslint:disable */
export class Universe {
free(): void;
tick(): void;

static new(): Universe;

render(): string;

width(): number;

height(): number;

cells(): number;

toggle_cell(arg0: number, arg1: number): void;

}
```

Code overview: wasm_game_of_life.js

```
/**
* @returns {number}
*/
width() {
if (this.ptr === 0) {
throw new Error('Attempt to use a moved value');
}
return wasm.universe_width(this.ptr);
}
/**
* @returns {number}
*/
height() {
if (this.ptr === 0) {
throw new Error('Attempt to use a moved value');
}
return wasm.universe_height(this.ptr);
}
```

```
/**
* @param {number} arg0
* @param {number} arg1
* @returns {void}
*/
toggle_cell(arg0, arg1) {
if (this.ptr === 0) {
throw new Error('Attempt to use a moved value');
}
return wasm.universe_toggle_cell(this.ptr, arg0,
arg1);
}
```

Code overview: index.html

Index.html

```
<style>
body {
width: 100%;
height: 100%;
display: flex;
align-items: center;
justify-content: center;
}
</style>
</head>
<body>
<body>
<button id="play-pause"></button>
<canvas id="game-of-life-canvas"></canvas>
<script src='./bootstrap.js'></script>
</body>
```

bootstrap.js

```
const index = import("./index");
index.then(() => {
console.log("Loaded...");
});
```

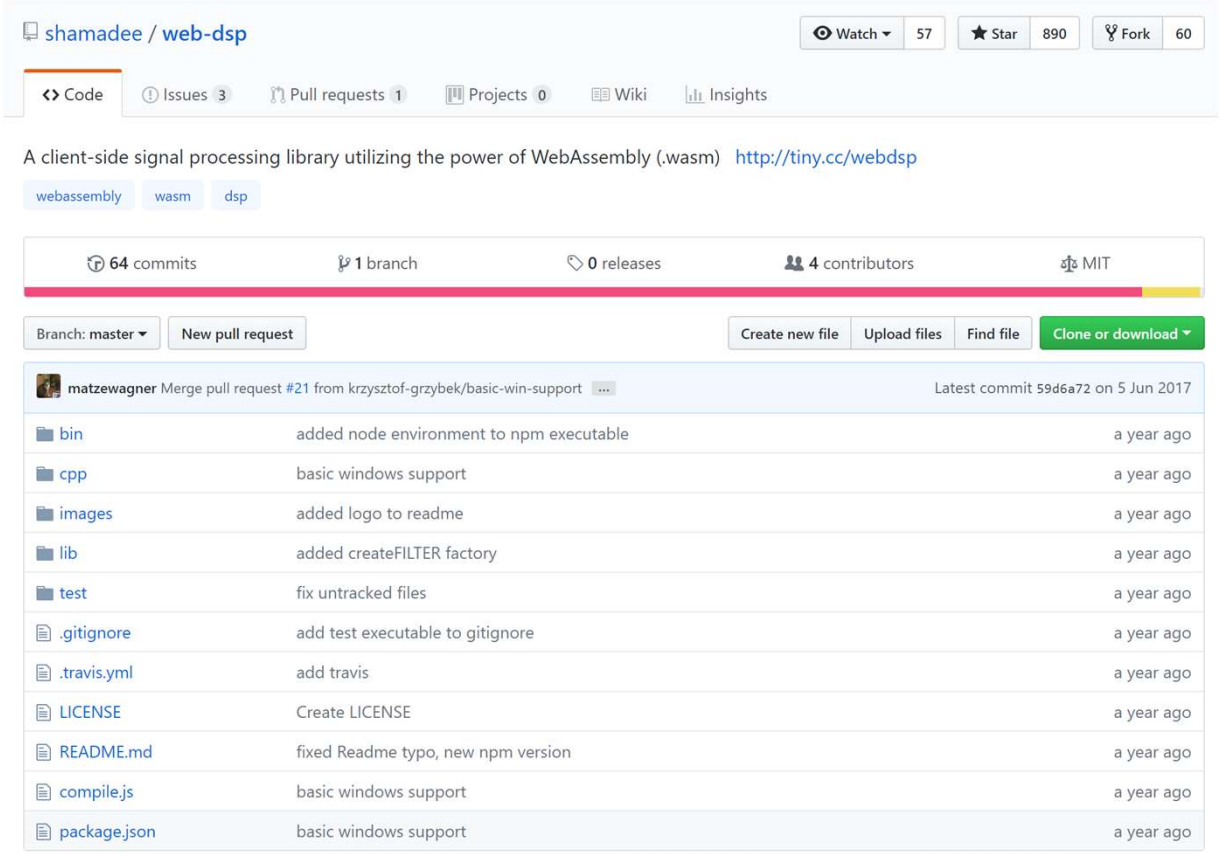
Code overview: index.js

```
import { Universe } from "./wasm_game_of_life";

canvas.addEventListener("click", event => {
const boundingRect = canvas.getBoundingClientRect();
const scaleX = canvas.width / boundingRect.width;
const scaleY = canvas.height / boundingRect.height;
const canvasLeft = (event.clientX -
boundingRect.left) * scaleX;
const canvasTop = (event.clientY - boundingRect.top)
* scaleY;
const row = Math.min(Math.floor(canvasTop /
(CELL_SIZE + 1)), height - 1);
const col = Math.min(Math.floor(canvasLeft /
(CELL_SIZE + 1)), width - 1);
universe.toggle_cell(row, col);
drawCells();
drawGrid();
});
```

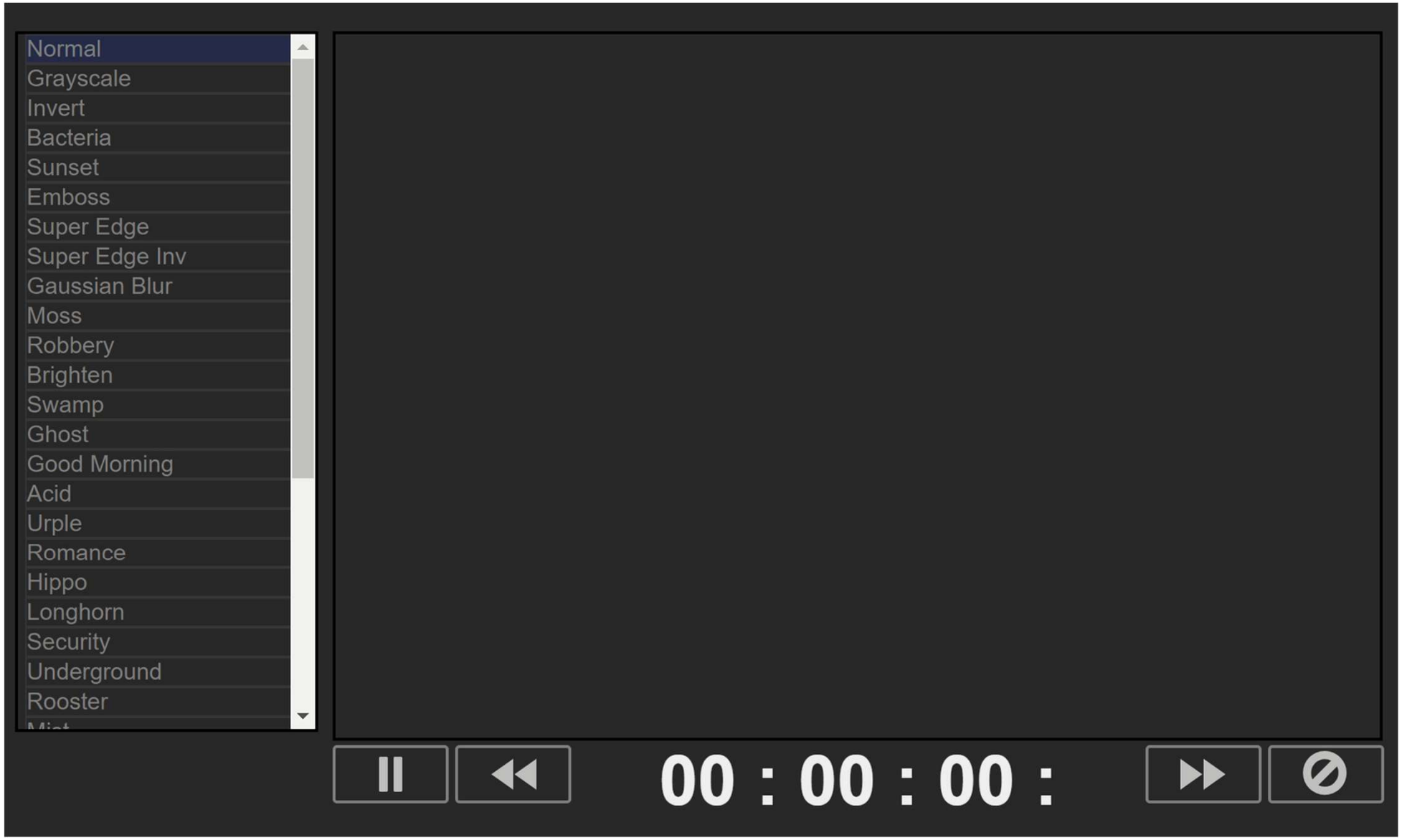

THE
REAL
WORLD

Example: DSP library

Examples: web-dsp



Source: https://github.com/shamadee/web-dsp

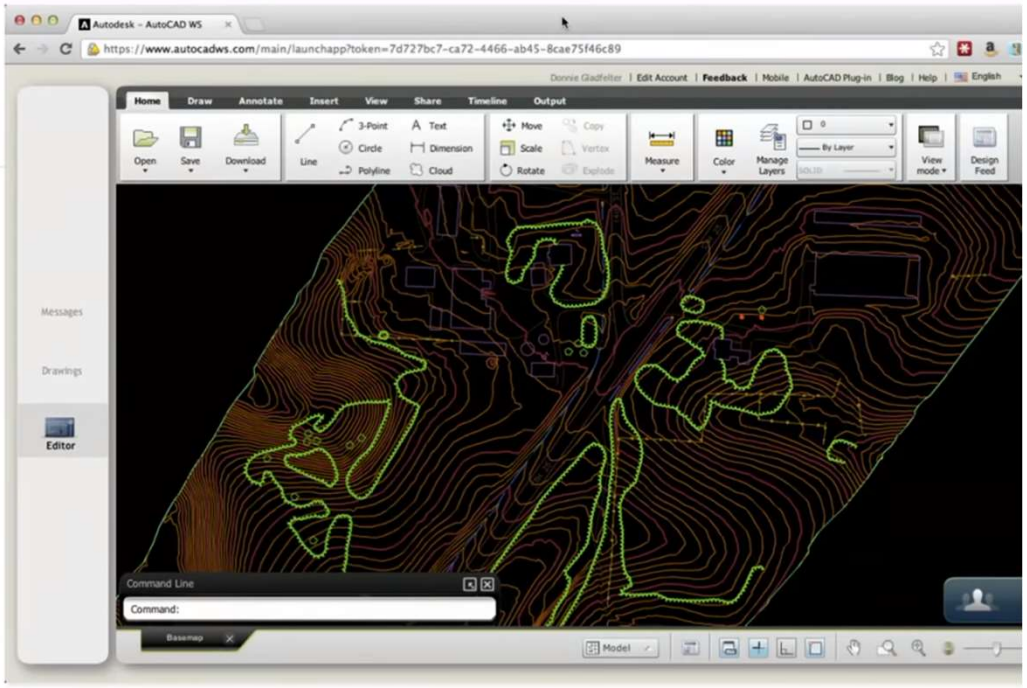

Normal
Grayscale
Invert
Bacteria
Sunset
Emboss
Super Edge
Super Edge Inv
Gaussian Blur
Moss
Robbery
Brighten
Swamp
Ghost
Good Morning
Acid
Urple
Romance
Hippo
Longhorn
Security
Underground
Rooster
00 : 00 : 00 :

Example: AutoCAD

Examples: AutoCAD

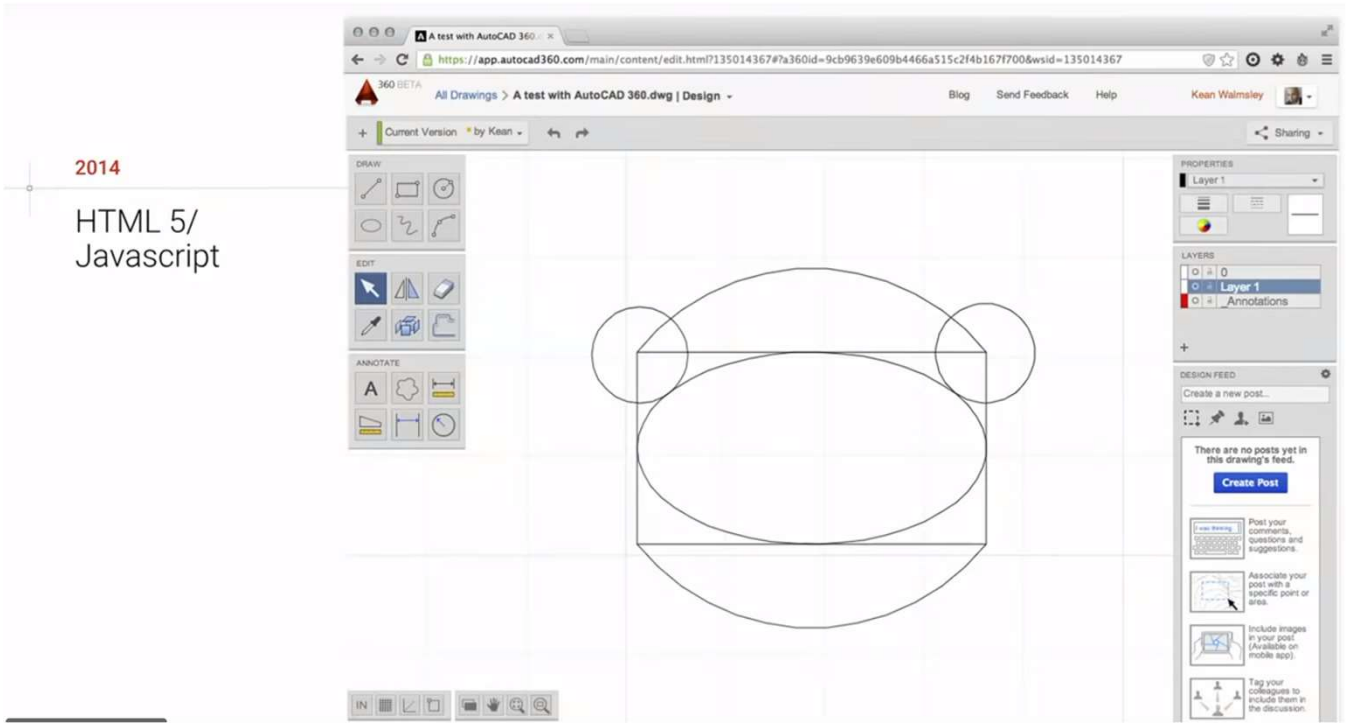
2010

AutoCAD
WS (Flash app)



Source: https://youtu.be/BnYq7JapeDA

Examples: AutoCAD

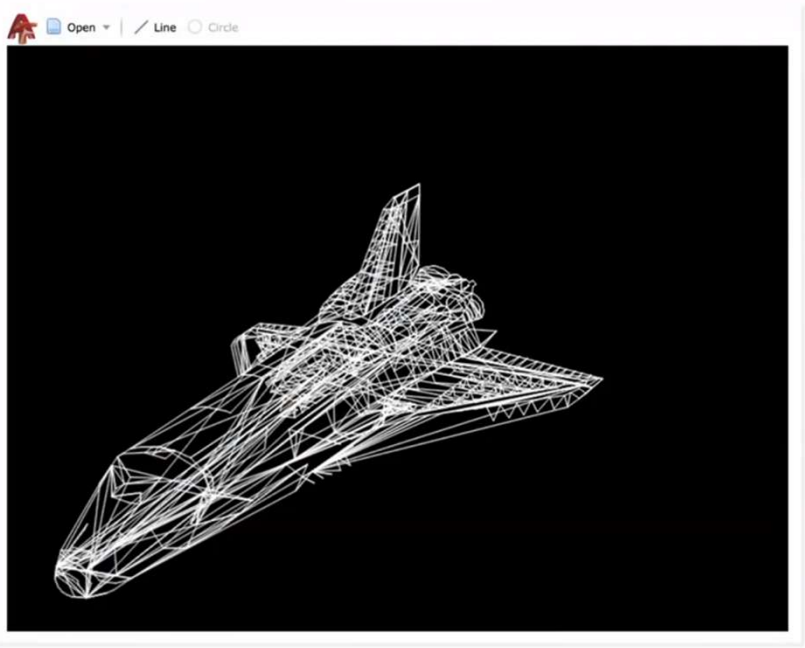


Source: https://youtu.be/BnYq7JapeDA

Examples: AutoCAD

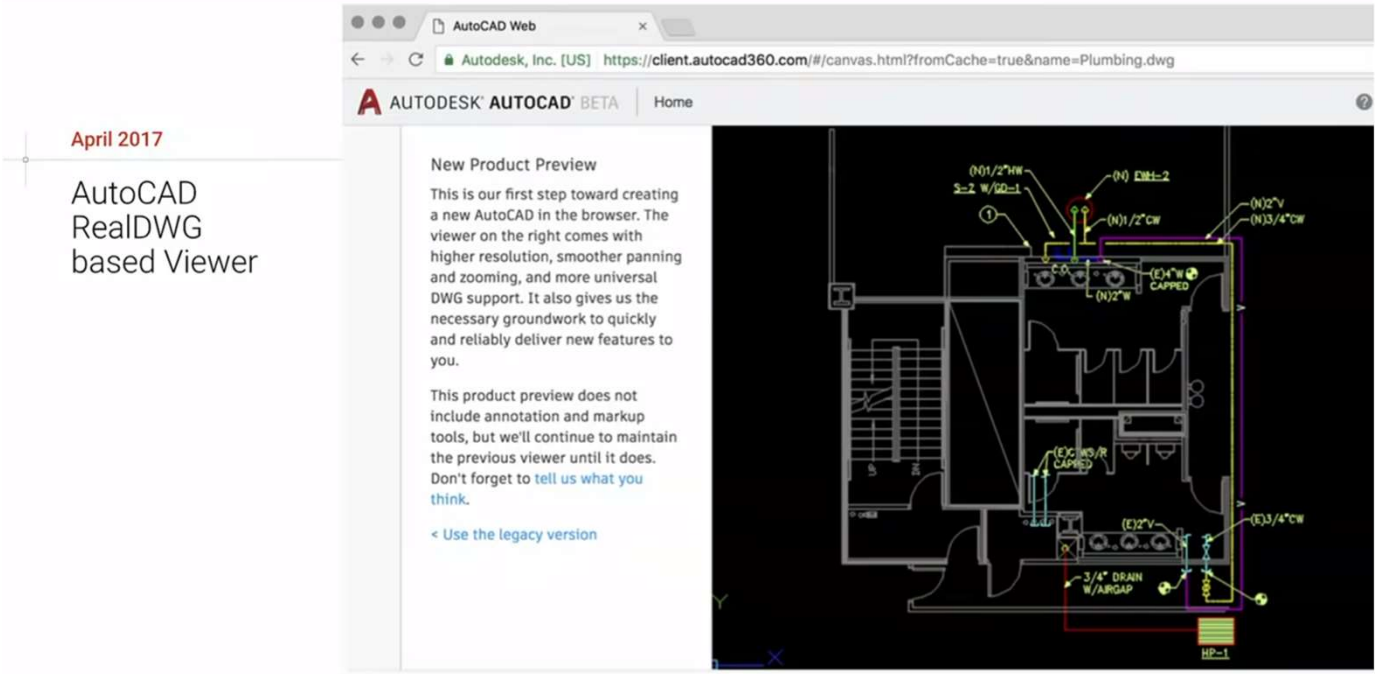
2016

First time ever:
AutoCAD C++
compiled with
Emscripten



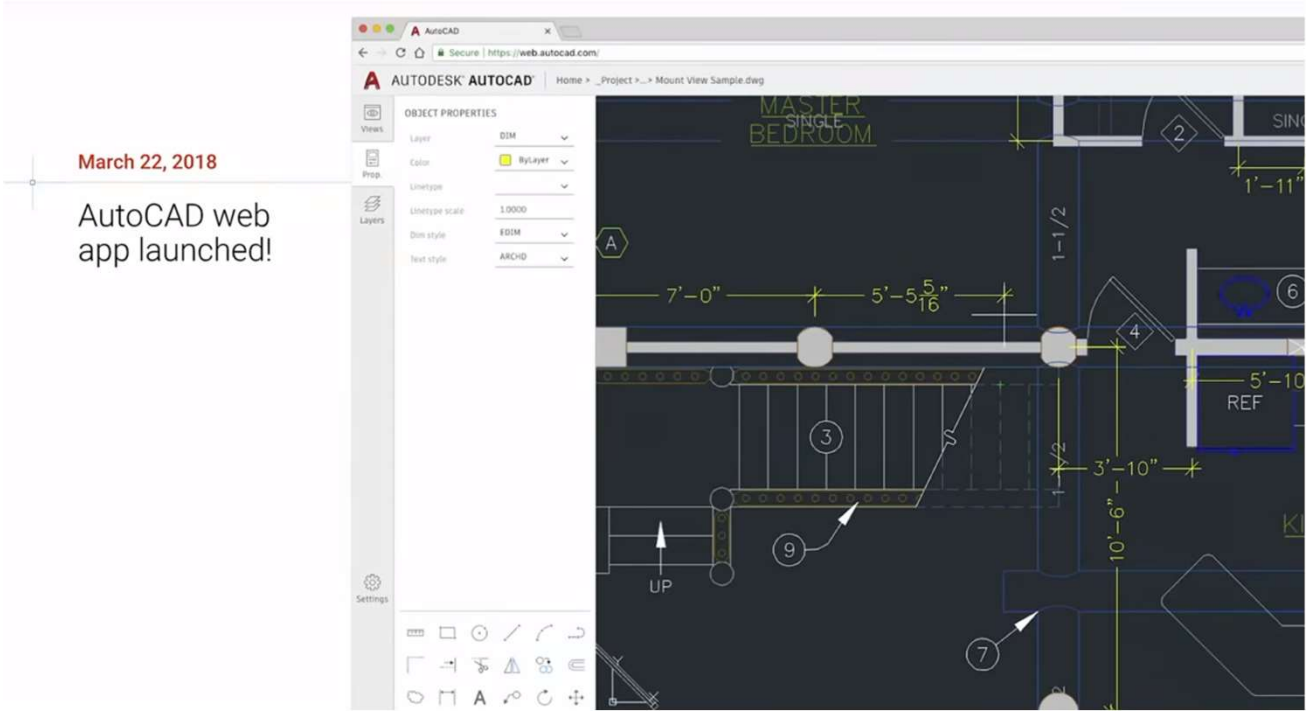
Source: https://youtu.be/BnYq7JapeDA

Examples: AutoCAD



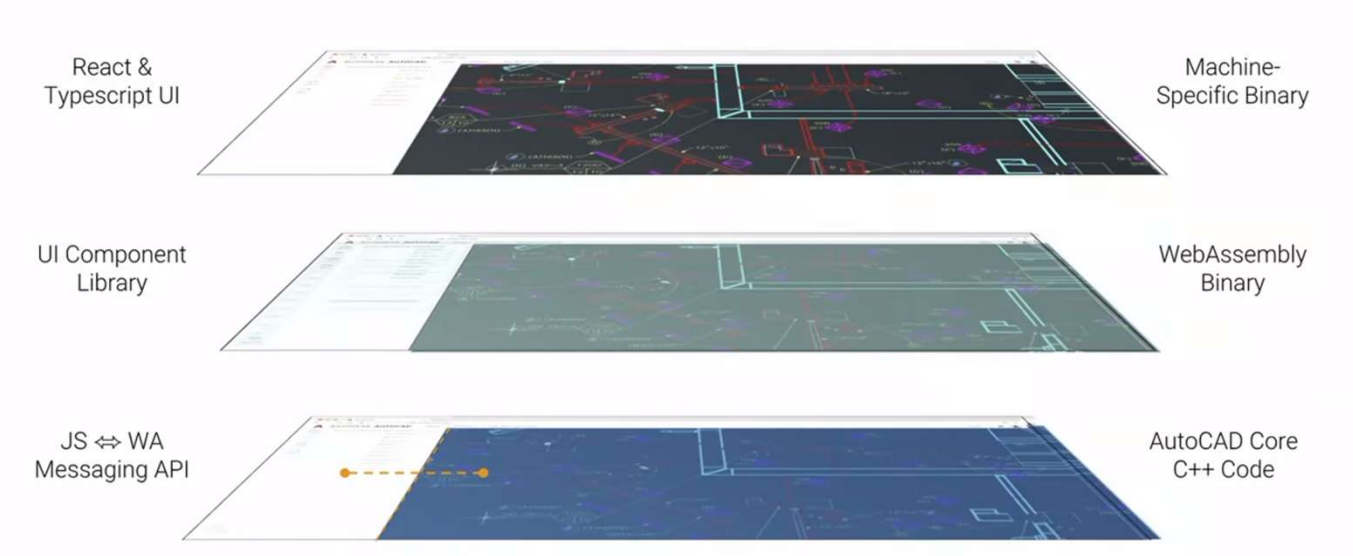
Source: https://youtu.be/BnYq7JapeDA

Examples: AutoCAD



Source: https://youtu.be/BnYq7JapeDA

Examples: AutoCAD



Source: https://youtu.be/BnYq7JapeDA

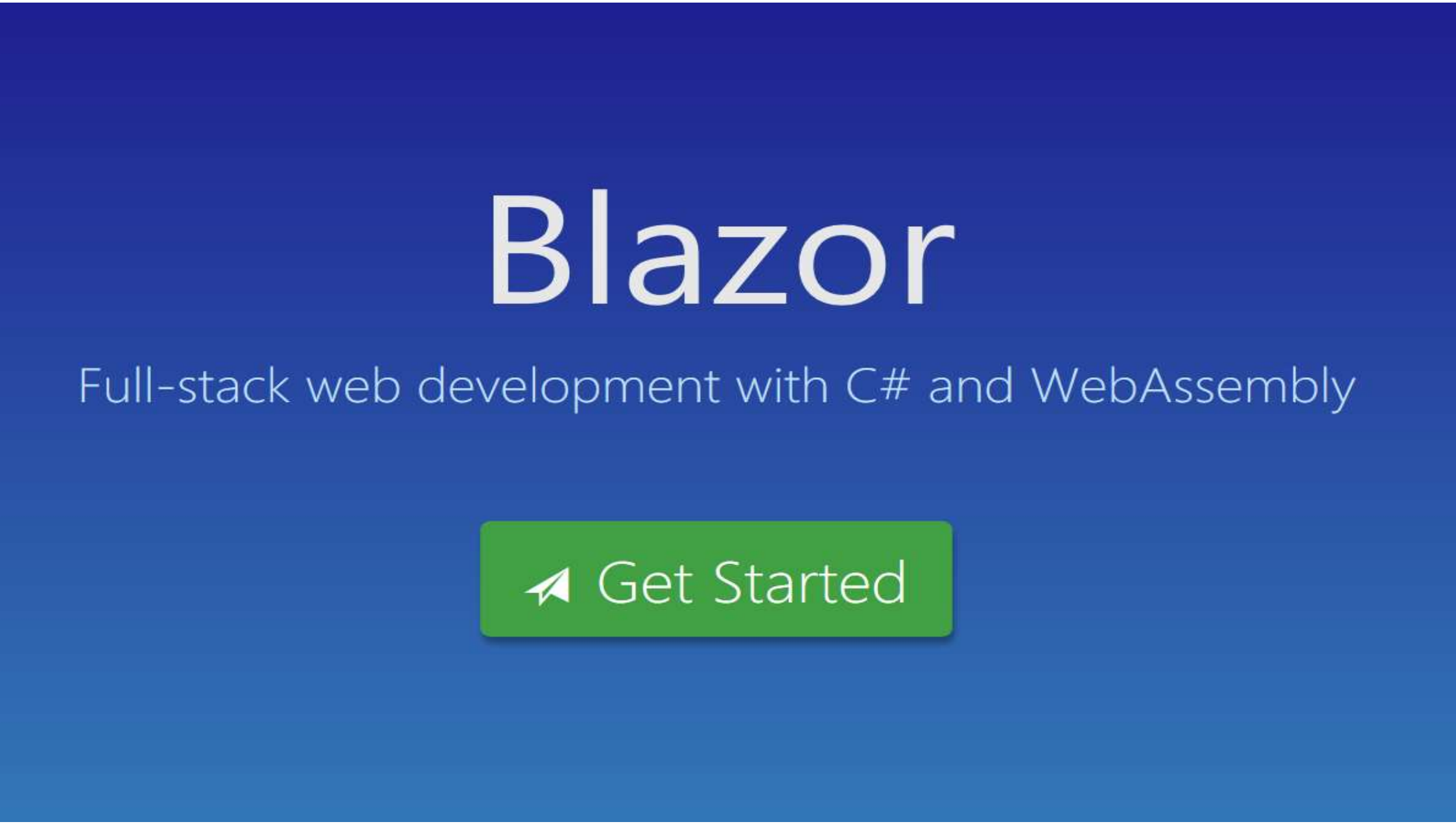
Future plans for WASM

What's next?

- Wasm call directly to DOM/Web Api
- Direct import of wasm in JavaScript
- Support more languages

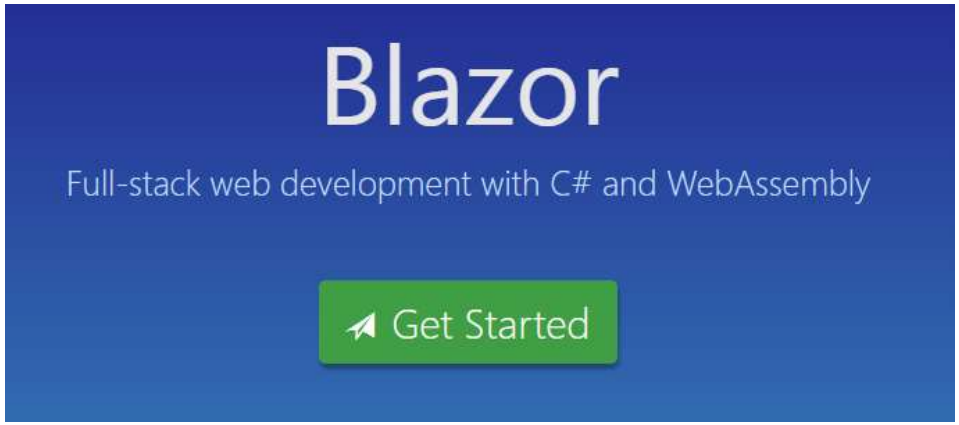
Source: https://youtu.be/BnYq7JapeDA

Other frameworks using WASM


Blazor
Full-stack web development with C# and WebAssembly
Get Started

Blazor

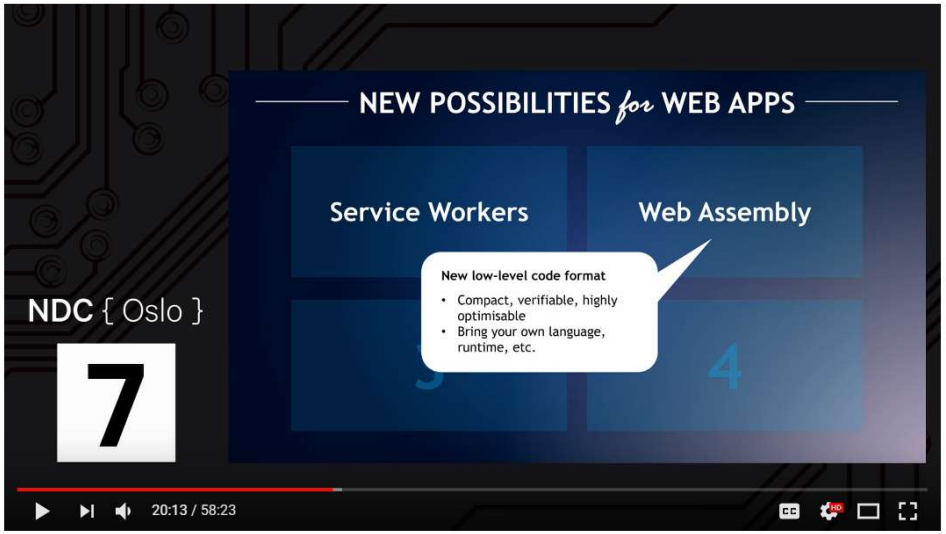
- Experimental .NET web framework
- C# and HTML running in the browser
- Runs on Web Assembly
- Native performance
- Trusted security sandbox



Source: https://blazor.net/

Origin

- Steve Sanderson – NDC 2017
- Mono team: Mono as compile target for WASM
- ASP.NET Team: Blazor

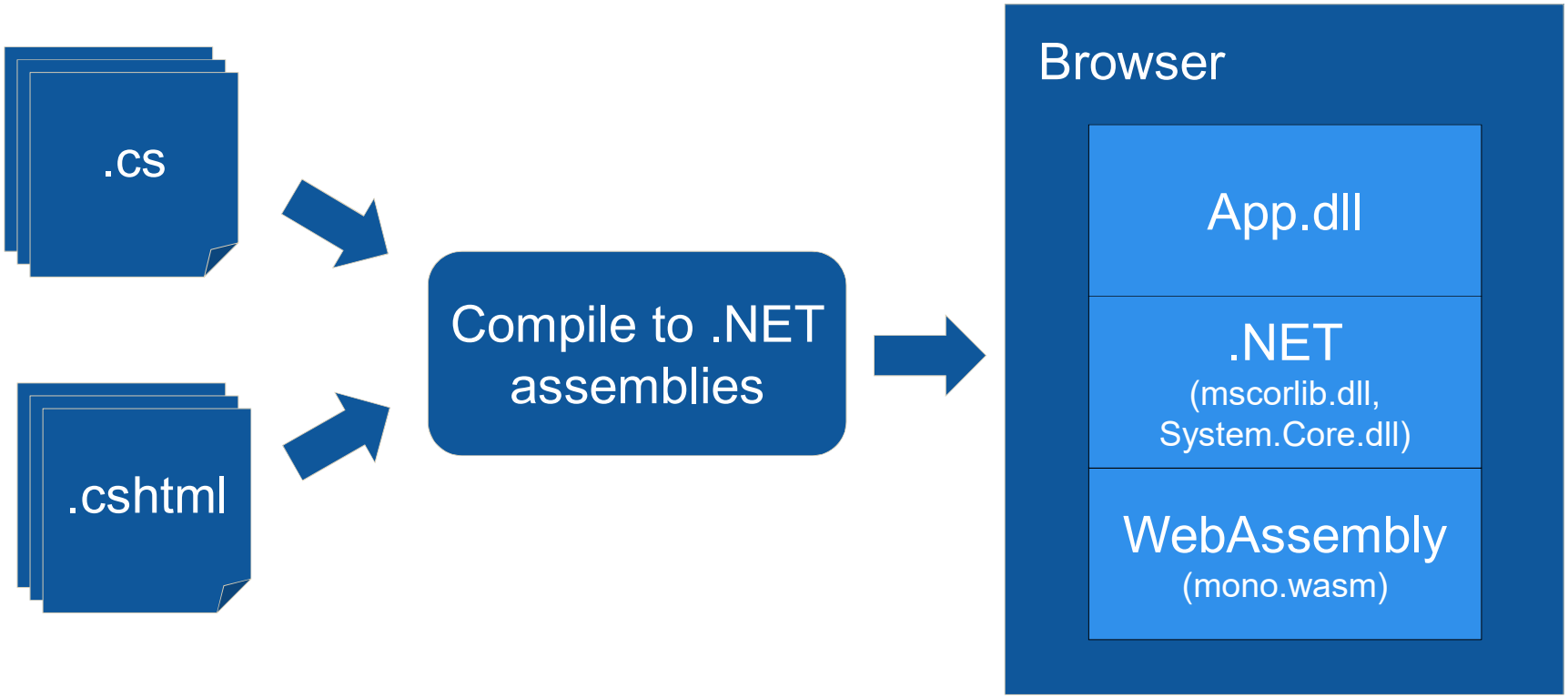


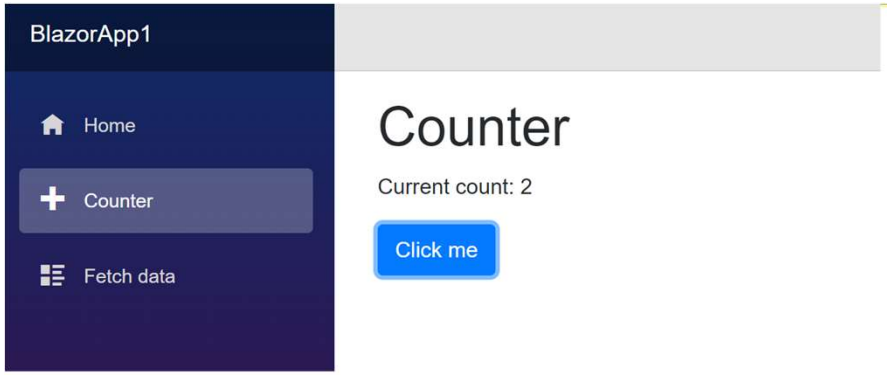
Web Apps can't really do *that*, can they? - Steve Sanderson

124,156 views 2K 65 SHARE

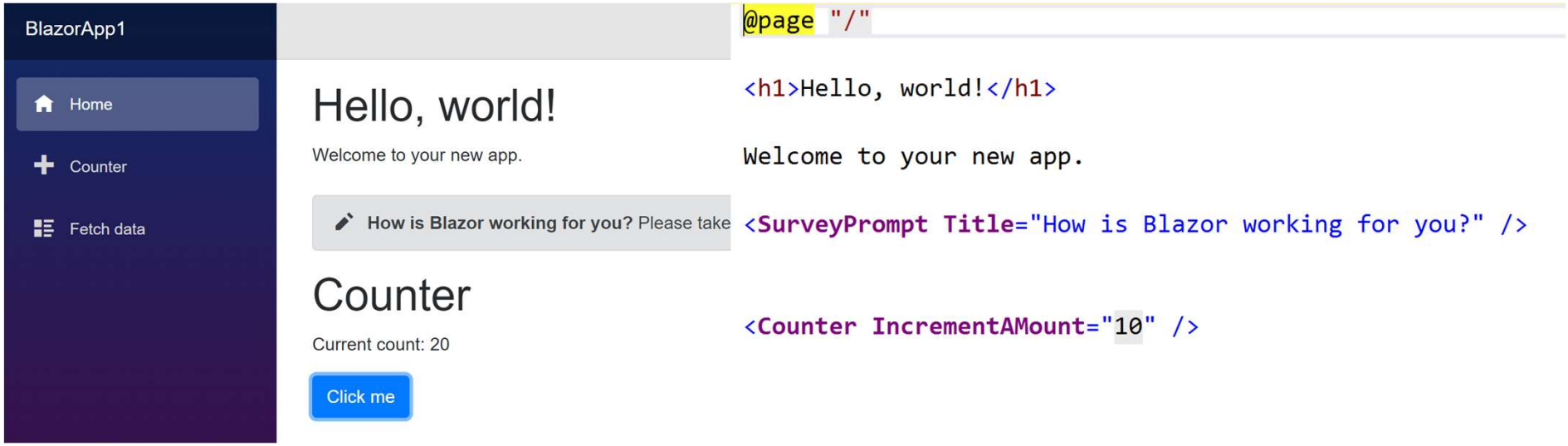
Source: https://www.youtube.com/watch?v=MiLAE6HMr10&t=1268s/

Overview


.cs
.cshtml
Compile to .NET assemblies
Browser
App.dll
.NET
(mscorlib.dll, System.Core.dll)
WebAssembly
(mono.wasm)




```razor
@page "/counter"

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" onclick="@IncrementCount">Click me</button>

@functions {
    int currentCount = 0;


    [Parameter]
    protected int IncrementAmount { get; set; } = 1;


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


BlazorApp1
Home
Counter
Fetch data
Hello, world!
Welcome to your new app.
How is Blazor working for you? Please take
Counter
Current count: 20
Click me
@page "/"
<h1>Hello, world!</h1>
Welcome to your new app.
<SurveyPrompt Title="How is Blazor working for you?" />
<Counter IncrementAMount="10" />

Name	Status	Type	Initiator
counter	304	document	Other
blazor.webassembly.js	304	script	counter
open-iconic-bootstrap.min.css	304	stylesheet	counter
blazor.boot.json	304	fetch	blazor.webassembly.js:1
mono.js	304	script	blazor.webassembly.js:1
favicon.ico	304	text/html	Other
mono.wasm	304	fetch	mono.js:1
BlazorApp1.dll	304	xhr	blazor.webassembly.js:1
Microsoft.AspNetCore.Blazor.Browser.dll	304	xhr	blazor.webassembly.js:1
Microsoft.AspNetCore.Blazor.dll	304	xhr	blazor.webassembly.js:1
Microsoft.Extensions.DependencyInjection.Abstractions.dll	304	xhr	blazor.webassembly.js:1
Microsoft.Extensions.DependencyInjection.dll	304	xhr	blazor.webassembly.js:1
Microsoft.JSInterop.dll	304	xhr	blazor.webassembly.js:1
Mono.WebAssembly.Interop.dll	304	xhr	blazor.webassembly.js:1
mscorlib.dll	304	xhr	blazor.webassembly.js:1
netstandard.dll	304	xhr	blazor.webassembly.js:1

Newold?

Have we not seen this before?

References

References

- Web Assembly
  - http://webassembly.org/getting-started/developers-guide/
  - https://developers.google.com/web/updates/2018/03/emscripting-a-c-library
  - https://github.com/shamadee/web-dsp

- Blazor
  - https://github.com/aspnet/Blazor
  - https://github.com/lohithgn/blazor-tour-of-heroes
  - https://codepen.io/ddprrt/pen/EgyBAJ
  - https://www.youtube.com/watch?v=5HSKDGHijdI&t

Thank you!

Twitter: @agrevlis

Email: hakan.silfvernagel@miles.no

Miles

FAGLIG AUTORITET OG VARME