

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

The screenshot shows the official website for WebAssembly. At the top, there's a navigation bar with links for Overview, Demo, Getting Started, Docs, Spec, Community, Roadmap, and FAQ. Below the navigation is a purple logo consisting of a white 'W' and 'A' stacked vertically. The word 'WEBASSEMBLY' is written in a sans-serif font below the logo. A green banner at the bottom of the main content area states 'WebAssembly 1.0 has shipped in 4 major browser engines.' followed by icons for Firefox, Chrome, Safari, and Edge, and a 'Learn more' link. The main content area contains a paragraph about WebAssembly being a binary instruction format for a stack-based virtual machine, designed for portable deployment across client and server applications. A callout box in the bottom right corner provides developer documentation details.

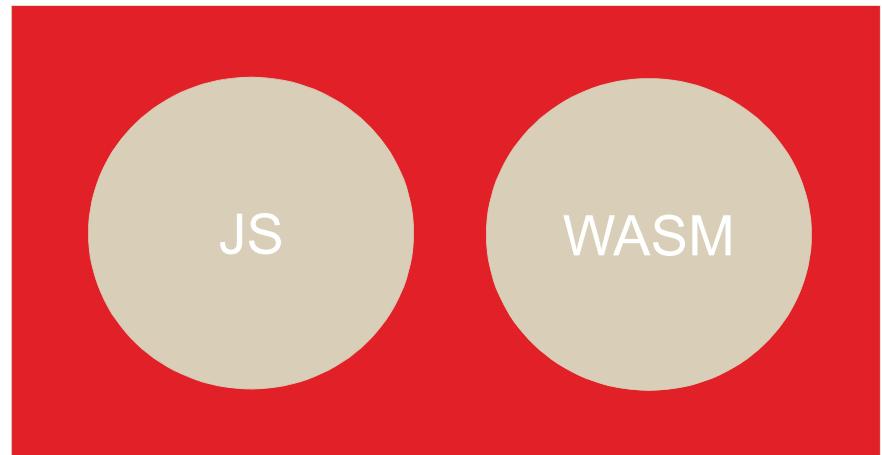
Developer reference documentation for Wasm can be found on [MDN's WebAssembly pages](#). The open standards for WebAssembly are developed in a [W3C Community Group](#) (that includes representatives from all major browsers) as well as a [W3C Working Group](#).

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...

Virtual Machine



Wasm

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

Hello WASM

Miles

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)

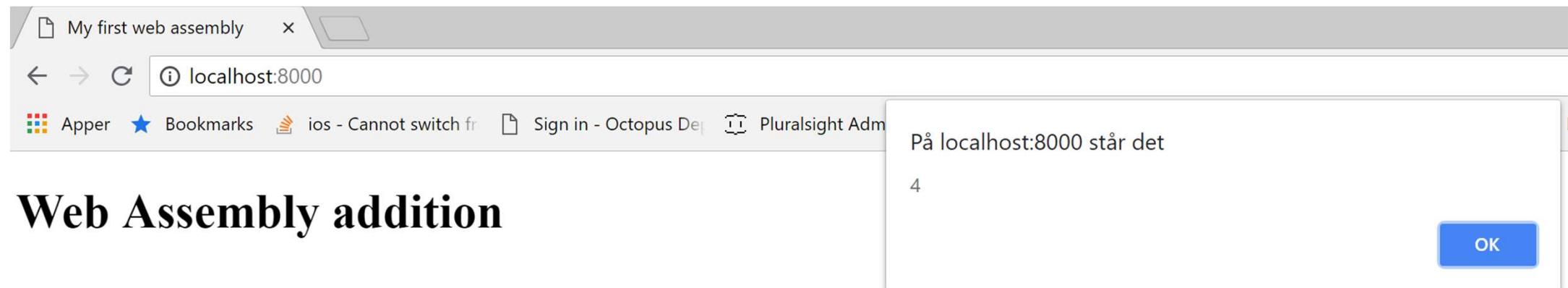
```
wat2wasm add.wat -o add.wasm  
wat2wasm add.wat -v
```

```
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)

The screenshot shows the WebAssembly Code Explorer interface. On the left, there is a memory dump table with three rows of hex values:

	0x00000000	0x00000010	0x00000020
	00 61 73 6D 01 00 00 00 01 07 01 60 02 7F 7F 01	7F 03 02 01 00 07 0A 01 06 61 64 64 54 77 6F 00	.asm.....`....
		addTwo.

On the right, the assembly code is displayed:

```
(module
  (type $type0 (func (param i32 i32) (result i32)))
  (export "addTwo" (func $func0))
  (func $func0 (param $var0 i32) (param $var1 i32) (result i32)
    get_local $var0
    get_local $var1
    i32.add
  )
)
```

<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)



[Documentation](#) [Install](#) [Community](#) [Contribute](#)

[Install Rust 1.28.0](#)

August 2, 2018

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

See who's using Rust, and [read more about Rust in production](#).

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                    "Ciao", "こんには", "안녕하세요",
                    "Czešć", "Olá", "Здравствуйте",
                    "Chào bạn", "您好", "Hello",
                    "Hej", "Ahoj", "ஆলু", "ສະບັບ"];
    for (num, greeting) in greetings.iter().enumerate() {
        match num {
            0 => println!("This code is editable and runnable!"),
            1 => println!("Este código es editable y ejecutable!"),
            2 => println!("Ce code est modifiable et exécutable !"),
            3 => println!("Questo codice è modificabile ed eseguibile!"),
            4 => println!("このコードは編集して実行出来ます！"),
            5 => println!("여기에서 코드를 수정하고 실행할 수 있습니다"),
            6 => println!("Ten kod można edytować oraz uruchomić!"),
            7 => println!("Este código é editável e executável!"),
            8 => println!("你可以编辑并运行此代码！"),
            9 => println!("Bạn có thể edit và run code trực tiếp!"),
            10 => println!("这段代码是可以编辑并且能够运行的！"),
            11 => println!("Dieser Code kann bearbeitet und ausgeführt werden!"),
            12 => println!("Den här koden kan redigeras och köras!"),
            13 => println!("Tento kód môžete upraviť a spustiť!"),
            14 => println!("(رچ) ویرایش و پیش از اجرا!"),
            15 => println!("ເພີ້ມແຕ່ລາຍການໃຫຍ່ແລ້ວເອົາໃຫຍ່!"),
            _ => {}
        }
    }
}
```

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 <abhbakan@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 <abkhakan@gmail.com>"]
description = "Code used to demonstrate how to use wasm-pack"
license = "MIT/Apache-2.0"
repository = "https://github.com/abkhakan/hello-wasm-silver"

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

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

Rust example (2)

src/lib.rs

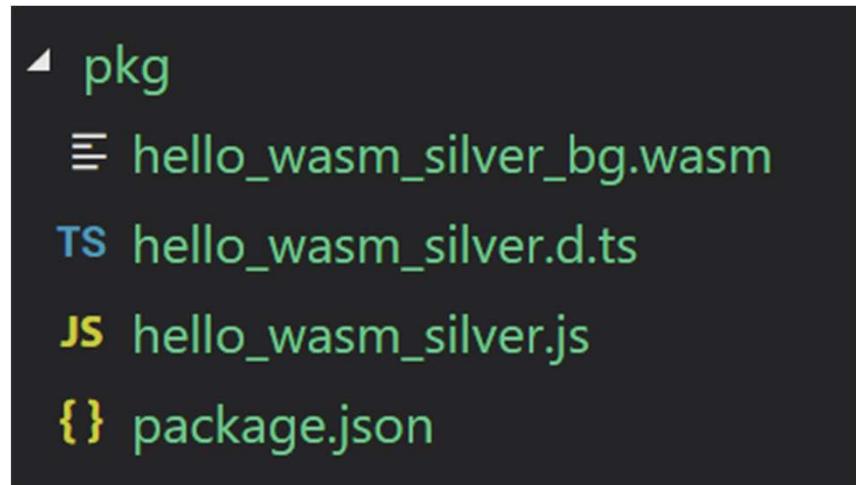
```
#![feature(proc_macro)]  
  
extern crate wasm_bindgen;  
  
use wasm_bindgen::prelude::*;

#[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 <abhbakan@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/abhbakan/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)

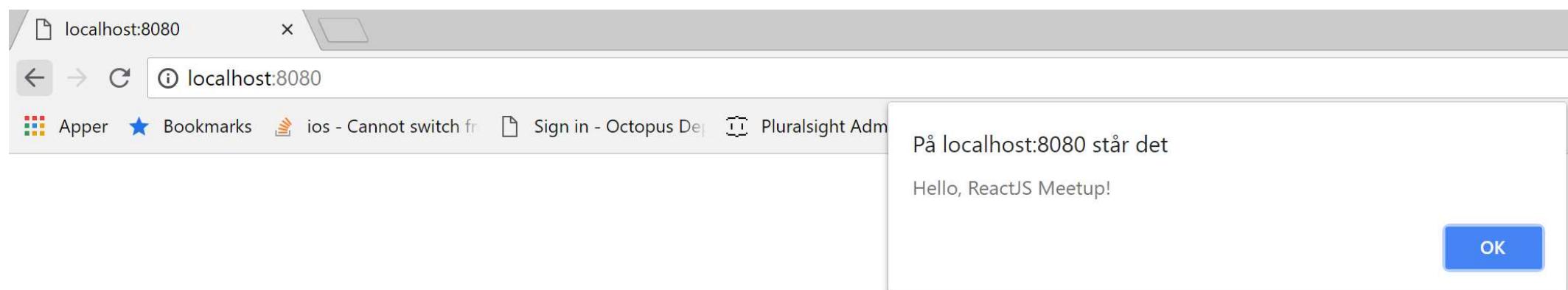
index.html

```
<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

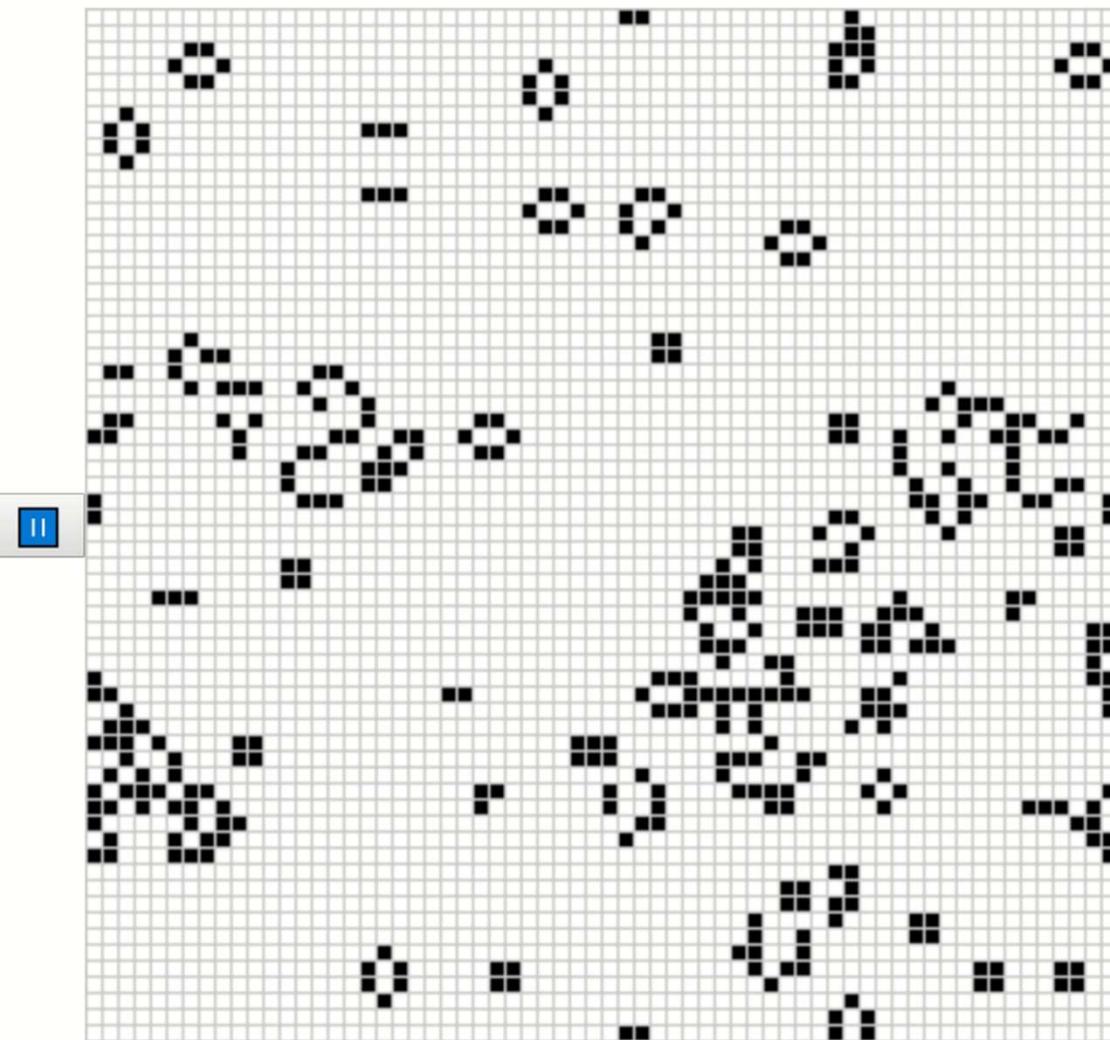
Miles

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 <abhbakan@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>
<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";

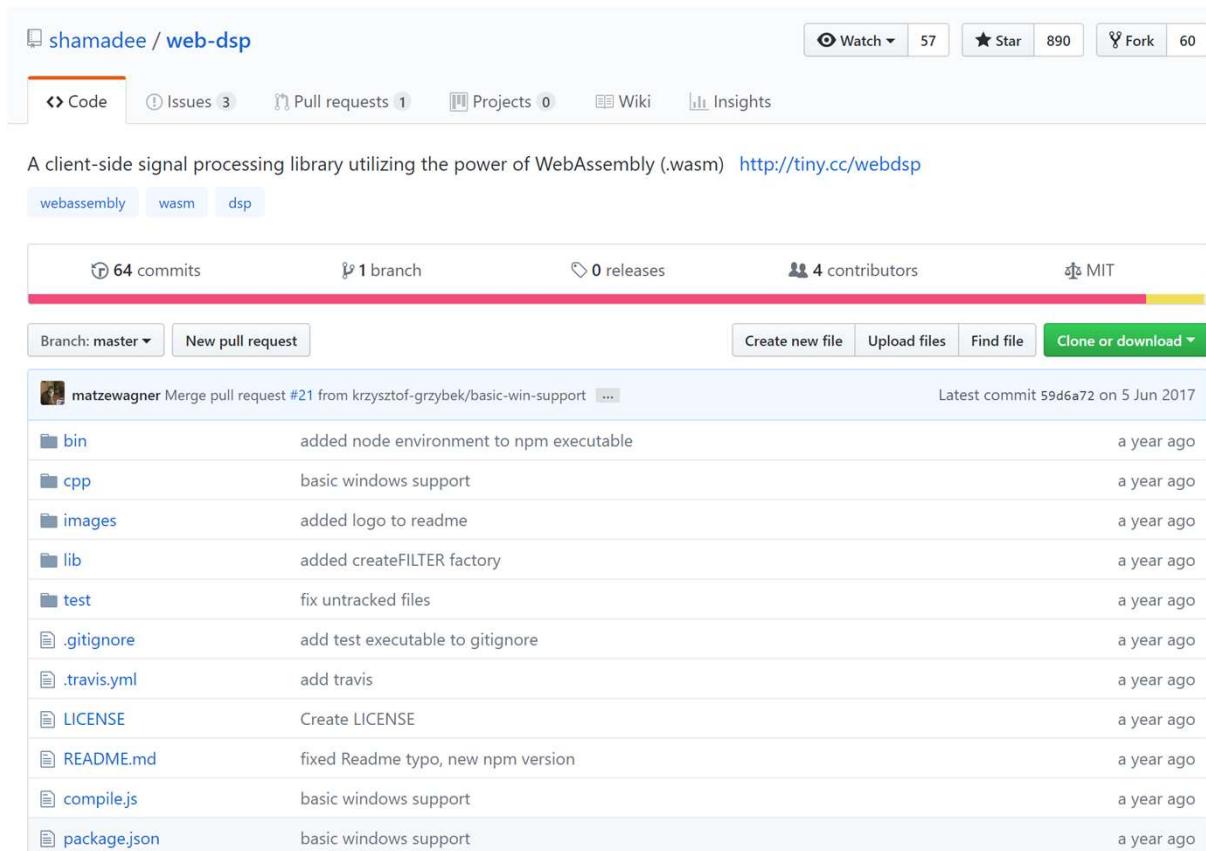
canvas.addEventListener("click", event => {
  const boundingRect = canvas.getBoundingClientRect();
  const scaleX = canvas.width / boundingRect.width;
  const scaleY = canvas.height / boundingRect.height;
  const canvasLeft = (event.clientX -
    boundingRect.left) * scaleX;
  const canvasTop = (event.clientY - 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



A client-side signal processing library utilizing the power of WebAssembly (.wasm) <http://tiny.cc/webdsp>

Code Issues 3 Pull requests 1 Projects 0 Wiki Insights

64 commits 1 branch 0 releases 4 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download ▾

matzwagner	Merge pull request #21 from krzysztof-grzybek/basic-win-support	...	Latest commit 59d6a72 on 5 Jun 2017
bin	added node environment to npm executable	a year ago	
cpp	basic windows support	a year ago	
images	added logo to readme	a year ago	
lib	added createFILTER factory	a year ago	
test	fix untracked files	a year ago	
.gitignore	add test executable to gitignore	a year ago	
.travis.yml	add travis	a year ago	
LICENSE	Create LICENSE	a year ago	
README.md	fixed Readme typo, new npm version	a year ago	
compile.js	basic windows support	a year ago	
package.json	basic windows support	a year ago	

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

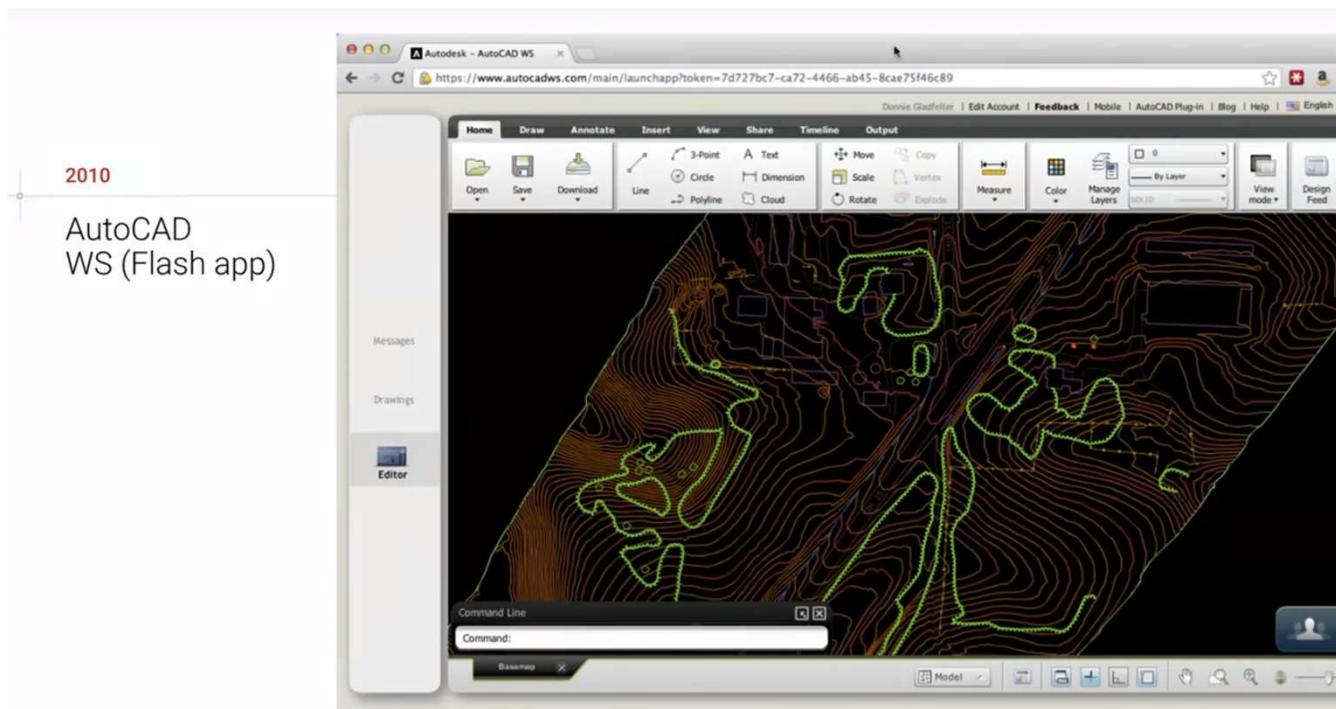


00 : 00 : 00 :



Example: AutoCAD

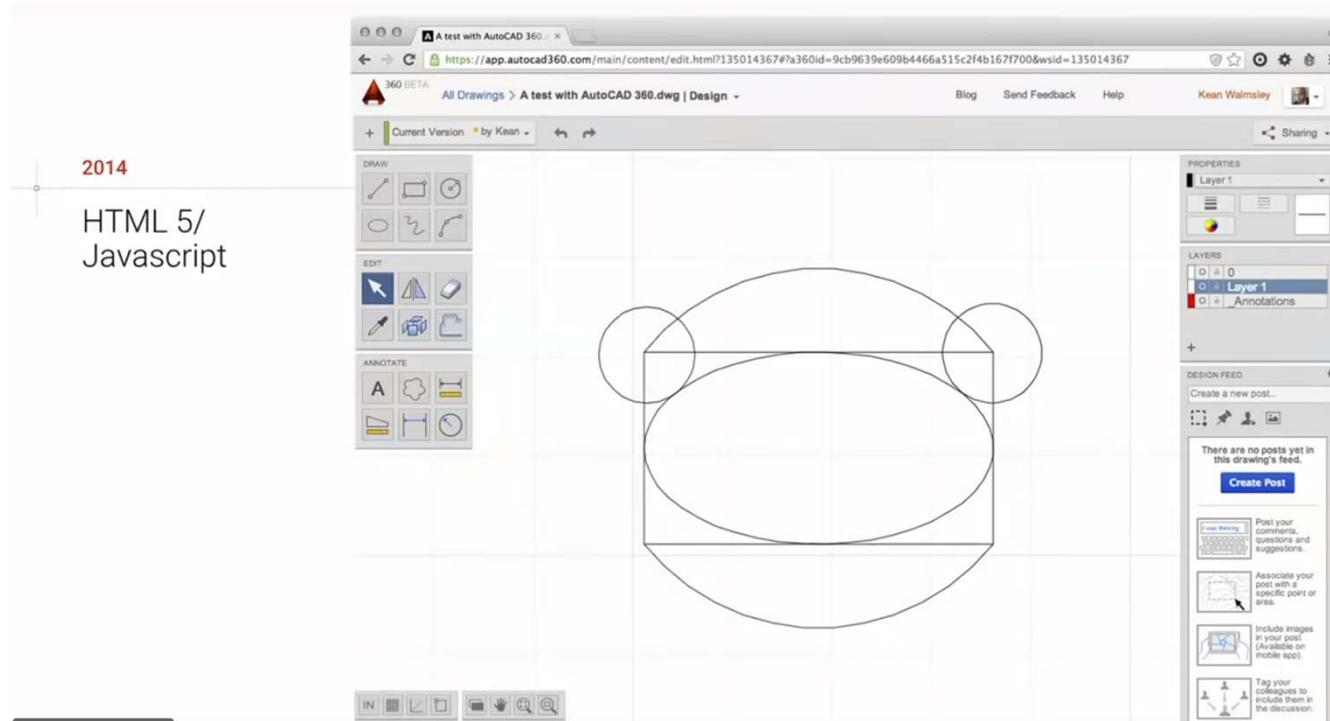
Examples: AutoCAD



Source: <https://youtu.be/BnYq7JapeDA>

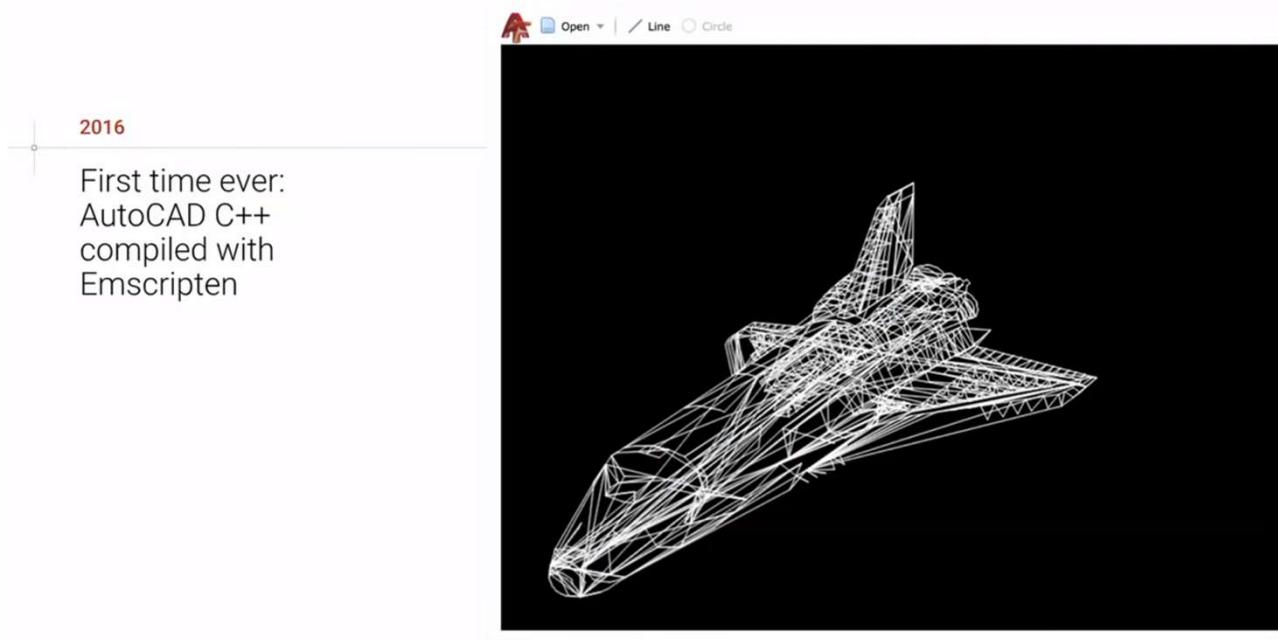
Miles

Examples: AutoCAD



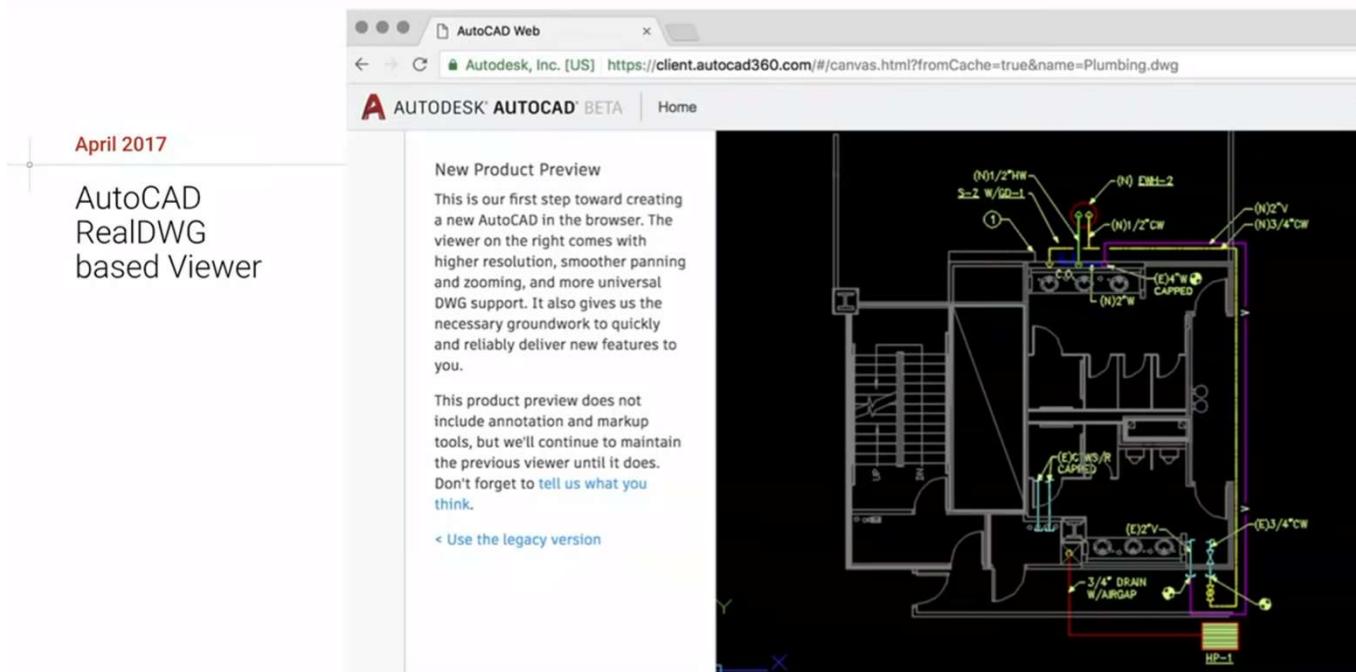
Source: <https://youtu.be/BnYq7JapeDA>

Examples: AutoCAD



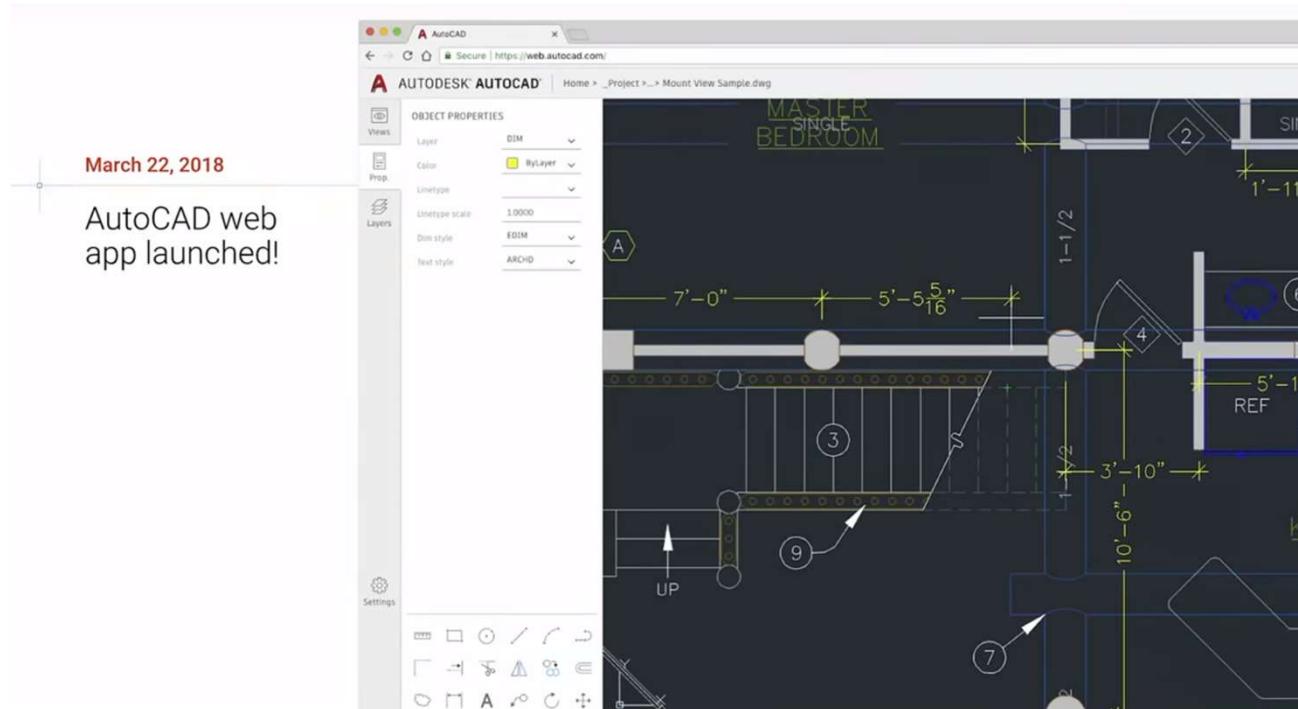
Source: <https://youtu.be/BnYq7JapeDA>

Examples: AutoCAD



Source: <https://youtu.be/BnYq7JapeDA>

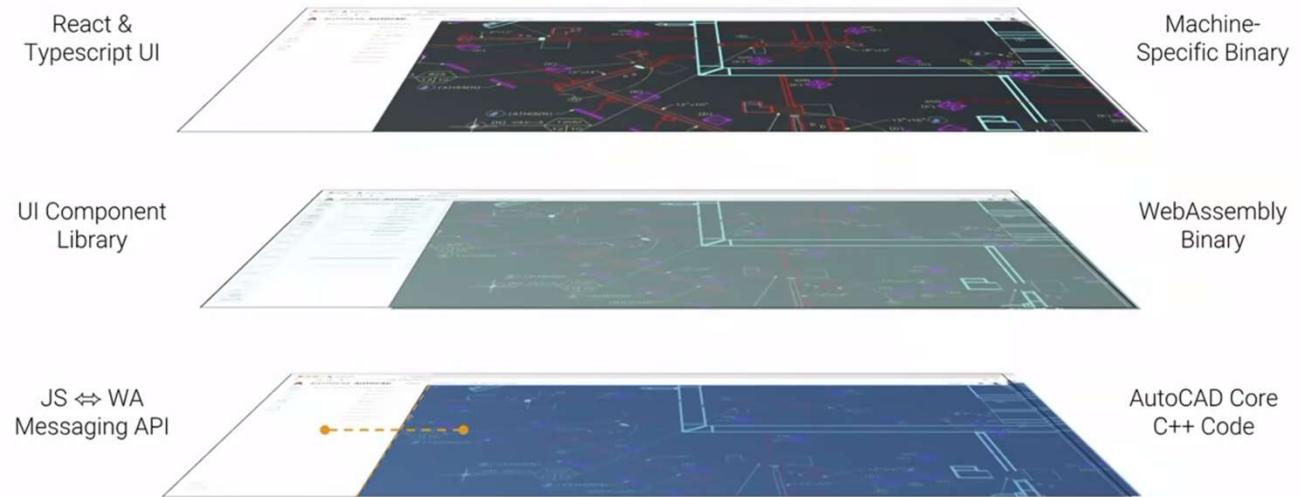
Examples: AutoCAD



Source: <https://youtu.be/BnYq7JapeDA>

Miles

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

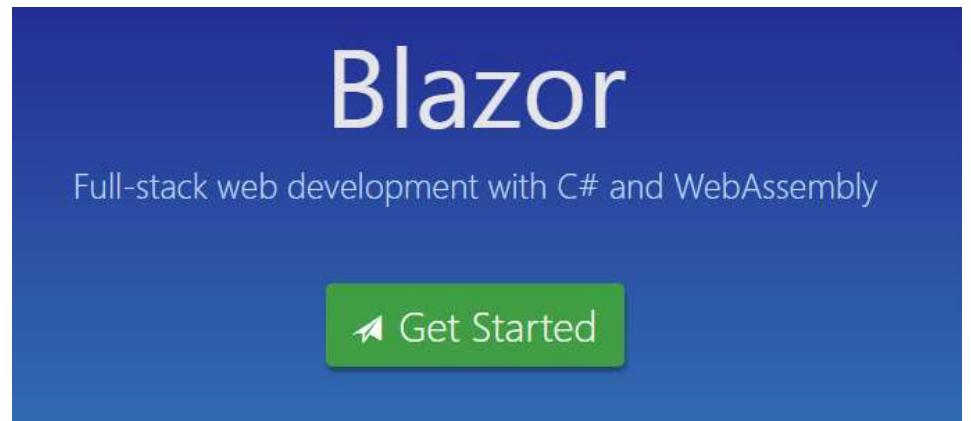
Full-stack web development with C# and WebAssembly



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

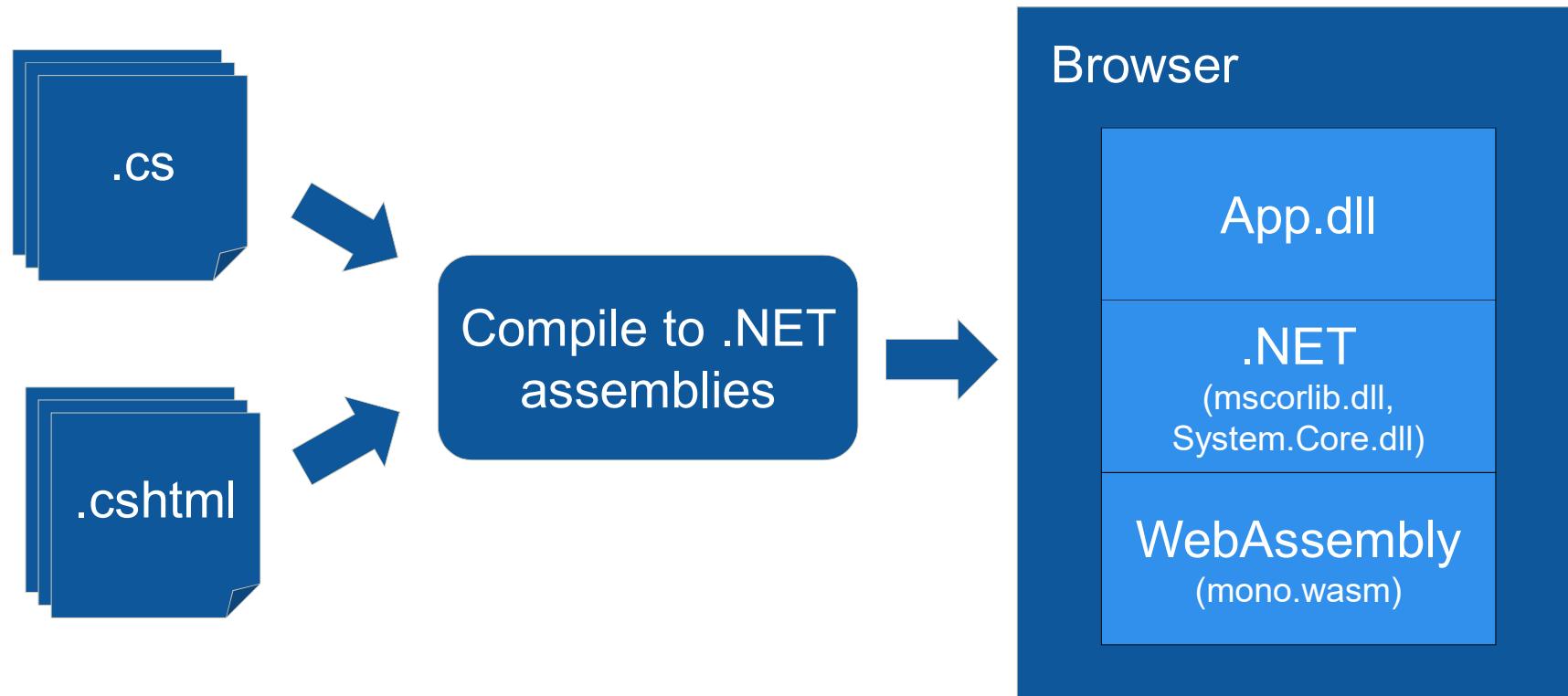
124,156 views

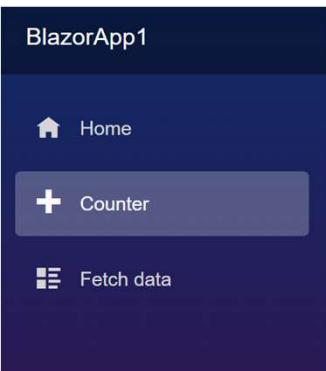
2K 65 SHARE ...

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

Miles

Overview





Counter

Current count: 2

Click me

```
@page "/counter"

<h1>Counter</h1>

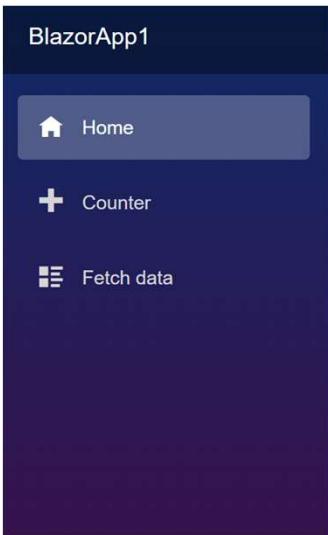
<p>Current count: @currentCount</p>

<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;
    }
}
```



```
@page "/"
```

Hello, world!

Welcome to your new app.

✍ How is Blazor working for you? Please take <SurveyPrompt Title="How is Blazor working for you?" />

Counter

Current count: 20

Click me



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

Miles

Newold?

Miles

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