


OXIDAR.org



Agenda :

“OxidAR.org” ~ Hernán Gonzalez

“WebAssembly” ~ Tomas Kenda

<https://oxidar.org> - 12/06/2025

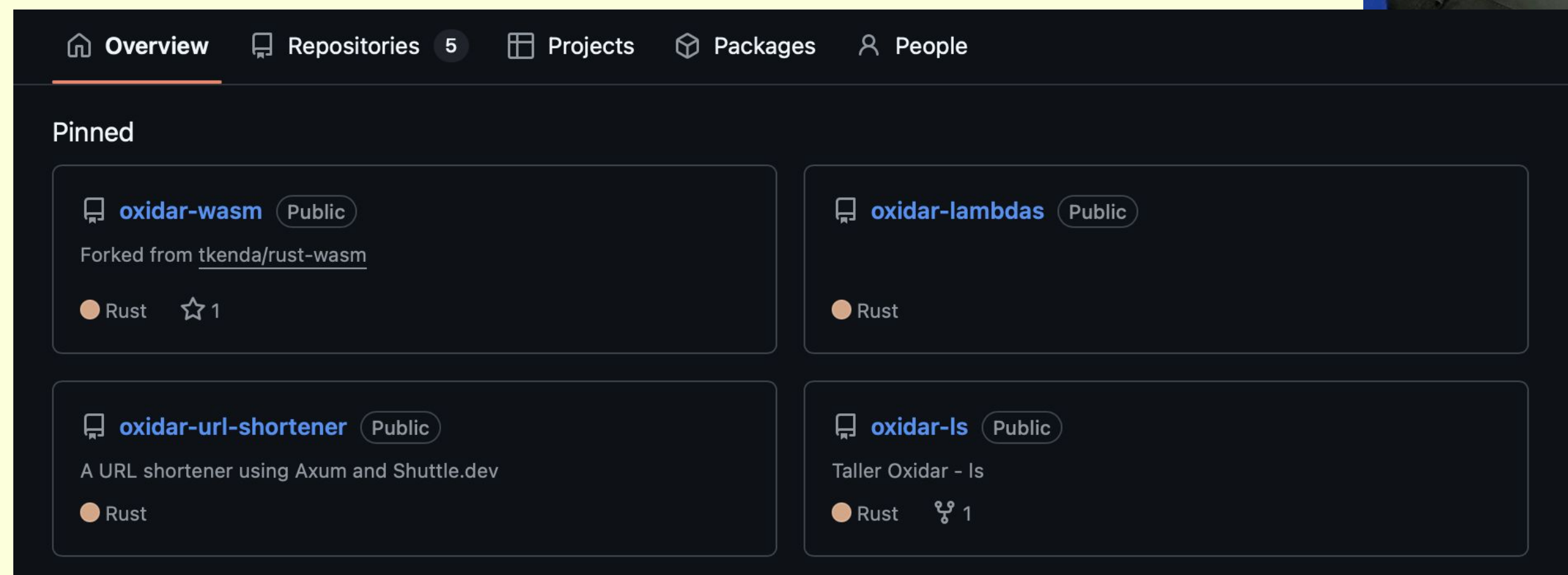
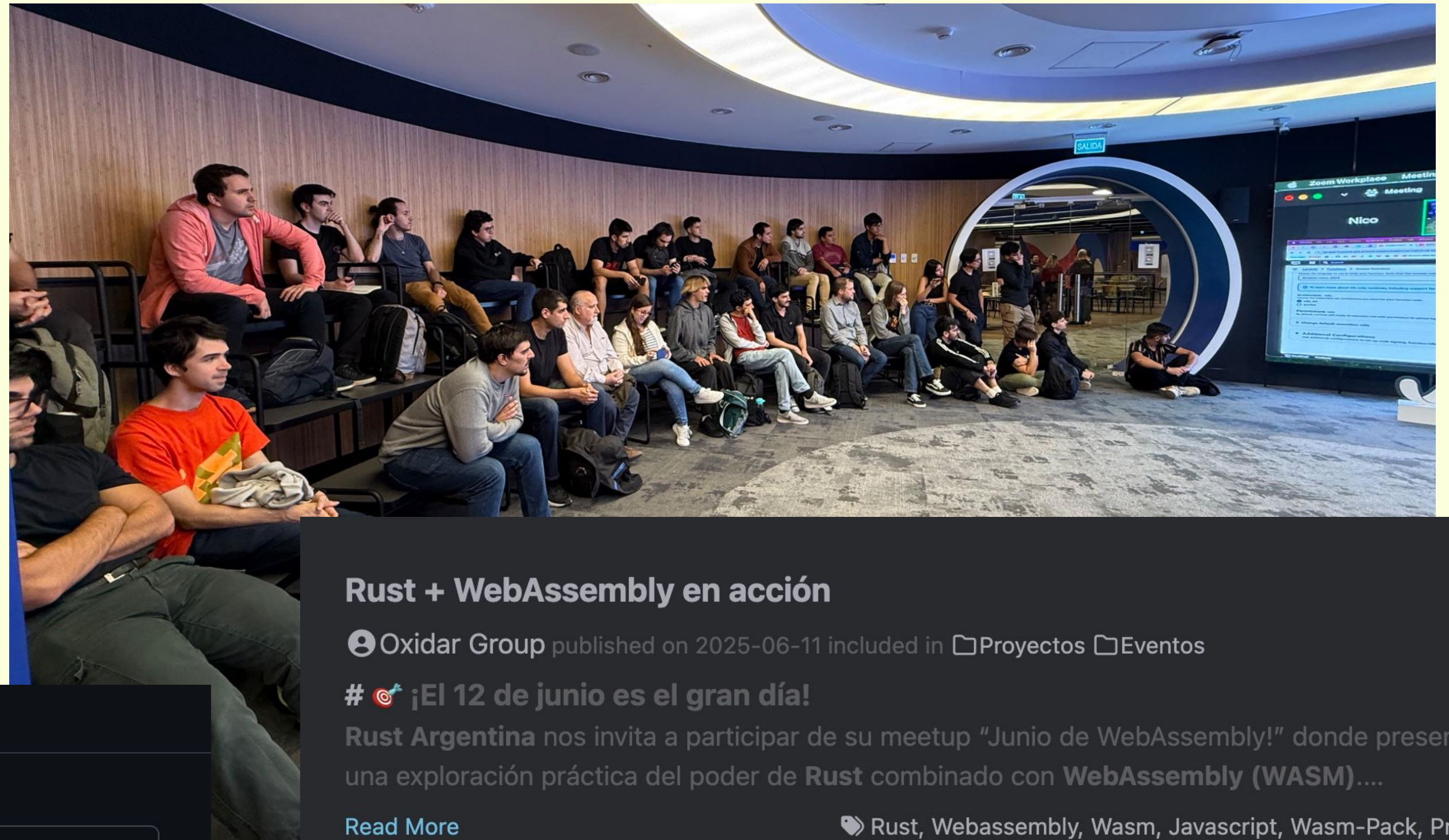


OXIDAR.org

Espacio Latinoamericano para la divulgación del lenguaje de programación Rust. 🦀

Rustaceans

- Talleres auto-guiados
- Materiales de divulgación
- Repositorios públicos (MIT)
- Grupos de Colaboración



Rust + WebAssembly en acción

👤 Oxidar Group published on 2025-06-11 included in [Proyectos](#) [Eventos](#)

🦀 ¡El 12 de junio es el gran día!

Rust Argentina nos invita a participar de su meetup "Junio de WebAssembly!" donde presentaremos una exploración práctica del poder de **Rust** combinado con **WebAssembly (WASM)**....

[Read More](#)

🔖 Rust, Webassembly, Wasm, Javascript, Wasm-Pack, Presentacion

Rust en AWS Lambda con Cargo Lambda

👤 Oxidar Group published on 2025-04-03 included in [Proyectos](#) [Tutoriales](#)

🚀 Una nueva colaboración de la comunidad

En **OxidAR** creemos en el poder del aprendizaje colaborativo. Por eso, nos complace compartir uno de nuestros proyectos más recientes: **oxidar-lambdas**, una exploración práctica sobre cómo desplegar...

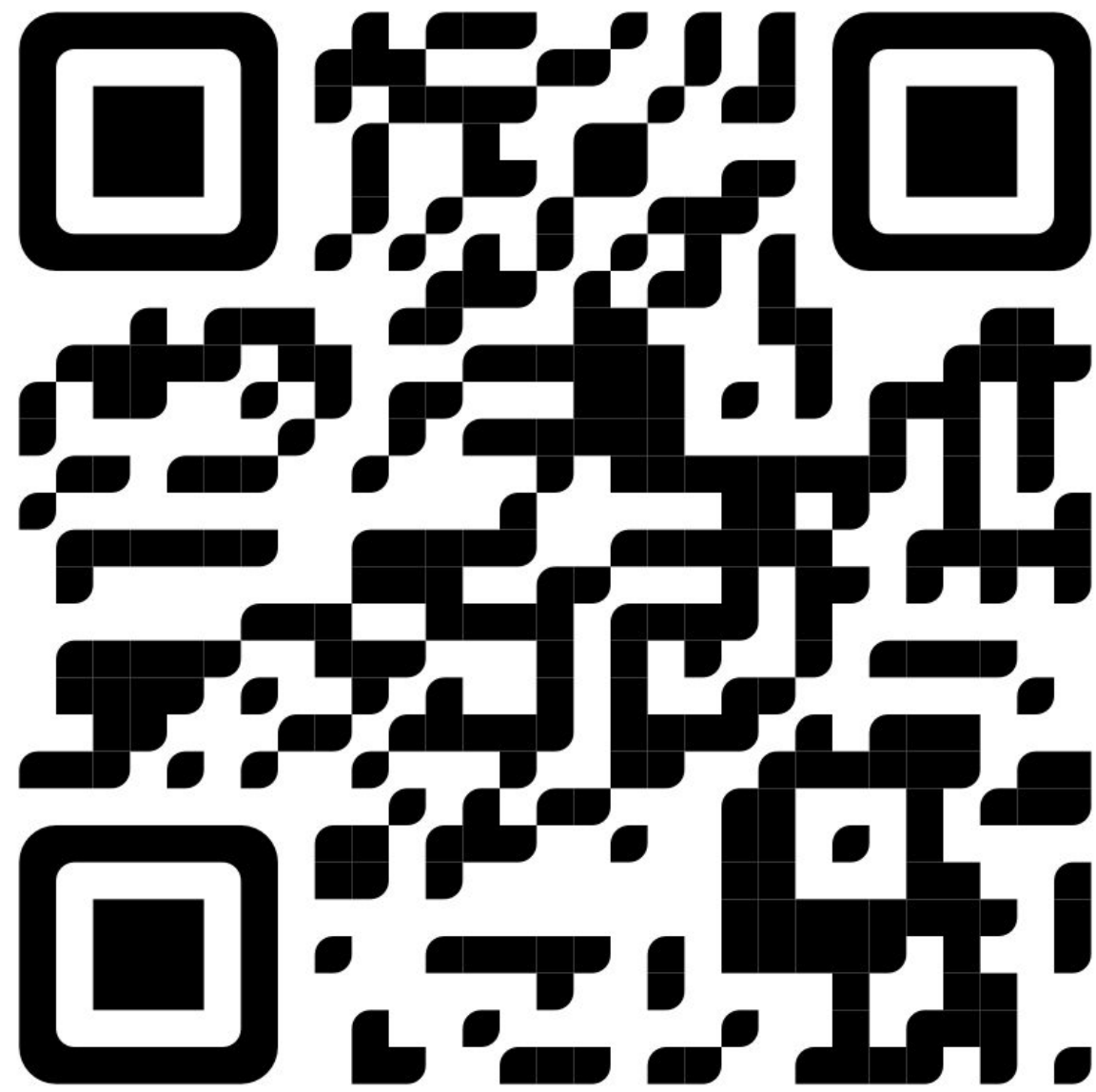
[Read More](#)

🔖 Rust, Aws, Lambda, Cargo-Lambda, Serverless, Colaboracion

OXIDAR.org

*Espacio Latinoamericano para la divulgación del lenguaje de
programación Rust. 🦀*

Sumate a la comunidad!



Oxidar.org






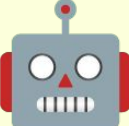
Telegram



GitHub

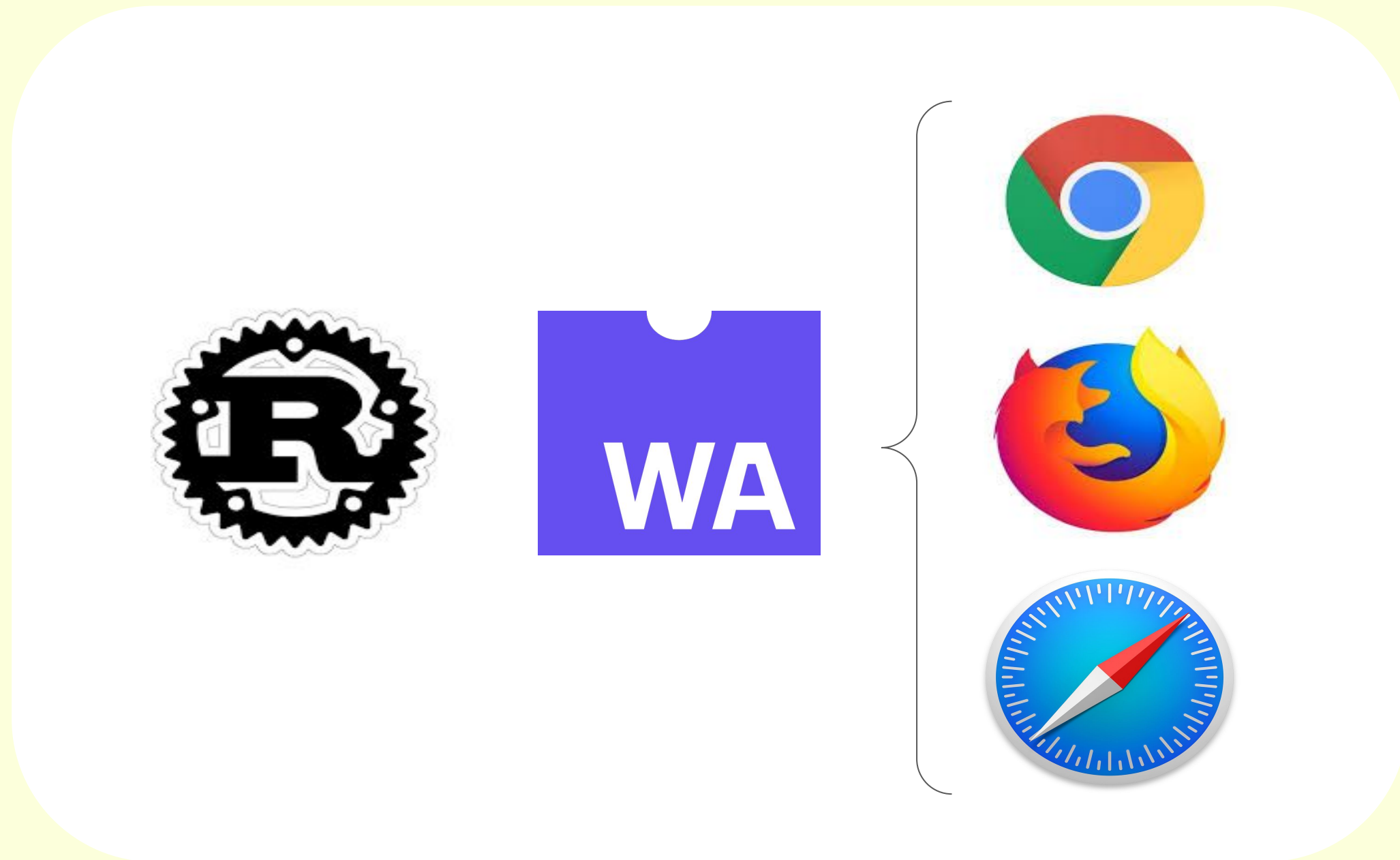
Tomas G. Kenda

WebAssembly con Rust

-  Background en electrónica
-  Healthcare Interoperability PACS, DICOM & HL7
-  Desarrollador de software, trabajando actualmente en startup de healthcare.
-  Rust, C/C++, JS



WebAssembly con Rust



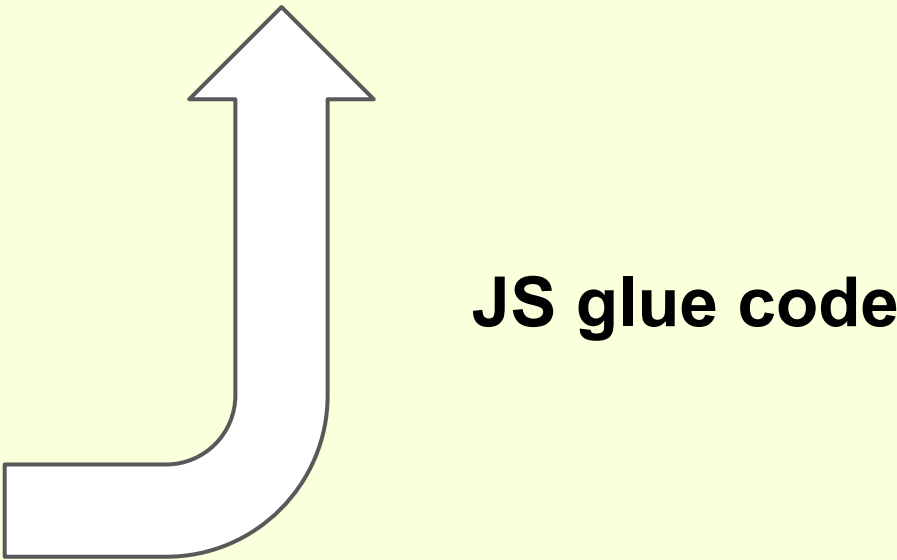
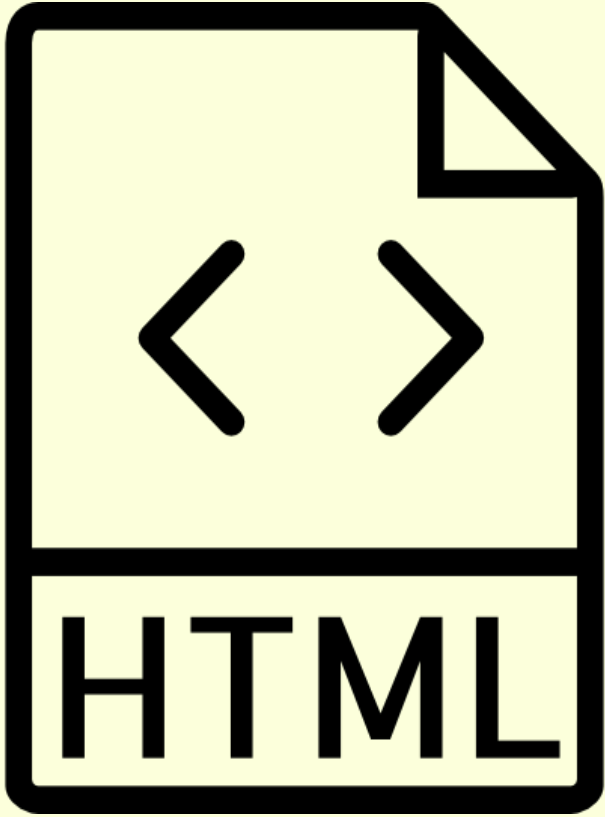
WASM es un formato de código binario diseñado para ejecutarse en navegadores web con alto rendimiento y cercano al nativo.



¿Qué es WebAssembly?

```
#[unsafe(no_mangle)]
pub fn add(a: i32, b: i32) -> i32
{
    a + b
}
```

.rs



```
00000000 00 61 73 6D 01 00 00 01 07 01 60 02 7F 7F 01 7F 03 02 .asm.....'.....
00000013 01 00 04 05 01 70 01 01 01 05 03 01 00 10 06 19 03 7F 01 .....p.....
00000026 41 80 80 C0 00 0B 7F 00 41 80 80 C0 00 0B 7F 00 41 80 80 A.....A.....A..
00000039 C0 00 0B 07 2B 04 06 6D 65 6D 6F 72 79 02 00 03 61 64 64 ....+..memory...add
0000004c 00 00 0A 5F 5F 64 61 74 61 5F 65 6E 64 03 01 0B 5F 5F 68 ..._data_end..._h
0000005f 65 61 70 5F 62 61 73 65 03 02 0A 09 01 07 00 20 01 20 00 eap_base..... .
00000072 6A 0B 00 2E 04 6E 61 6D 65 00 0B 0A 62 61 73 69 63 2E 77 j....name...basic.w
00000085 61 73 6D 01 06 01 00 03 61 64 64 07 12 01 00 0F 5F 5F 73 asm.....add.....s
00000098 74 61 63 6B 5F 70 6F 69 6E 74 65 72 00 4D 09 70 72 6F 64 tack_pointer.M.prod
000000ab 75 63 65 72 73 02 08 6C 61 6E 67 75 61 67 65 01 04 52 75 ucers..language..Ru
000000be 73 74 00 0C 70 72 6F 63 65 73 73 65 64 2D 62 79 01 05 72 st..processed-by..r
000000d1 75 73 74 63 1D 31 2E 38 36 2E 30 20 28 30 35 66 39 38 34 ustc.1.86.0 (05f984
000000e4 36 66 38 20 32 30 32 35 2D 30 33 2D 33 31 29 00 49 0F 74 6F8 2025-03-31).I.t
000000f7 61 72 67 65 74 5F 66 65 61 74 75 72 65 73 04 2B 0A 6D 75 arget_features+.mu
0000010a 6C 74 69 76 61 6C 75 65 2B 0F 6D 75 74 61 62 6C 65 2D 67 ltivalue+.mutable-g
0000011d 6C 6F 62 61 6C 73 2B 0F 72 65 66 65 72 65 6E 63 65 2D 74 lobals+.reference-t
00000130 79 70 65 73 2B 08 73 69 67 6E 2D 65 78 74 ypes+.sign-ext
```

.wasm

```
import init, { add } from
'./pkg/ejemplo.js';

await init();
console.log(add(2, 3));
```

.js



¿Qué es WebAssembly?

⚡ **Rápido**

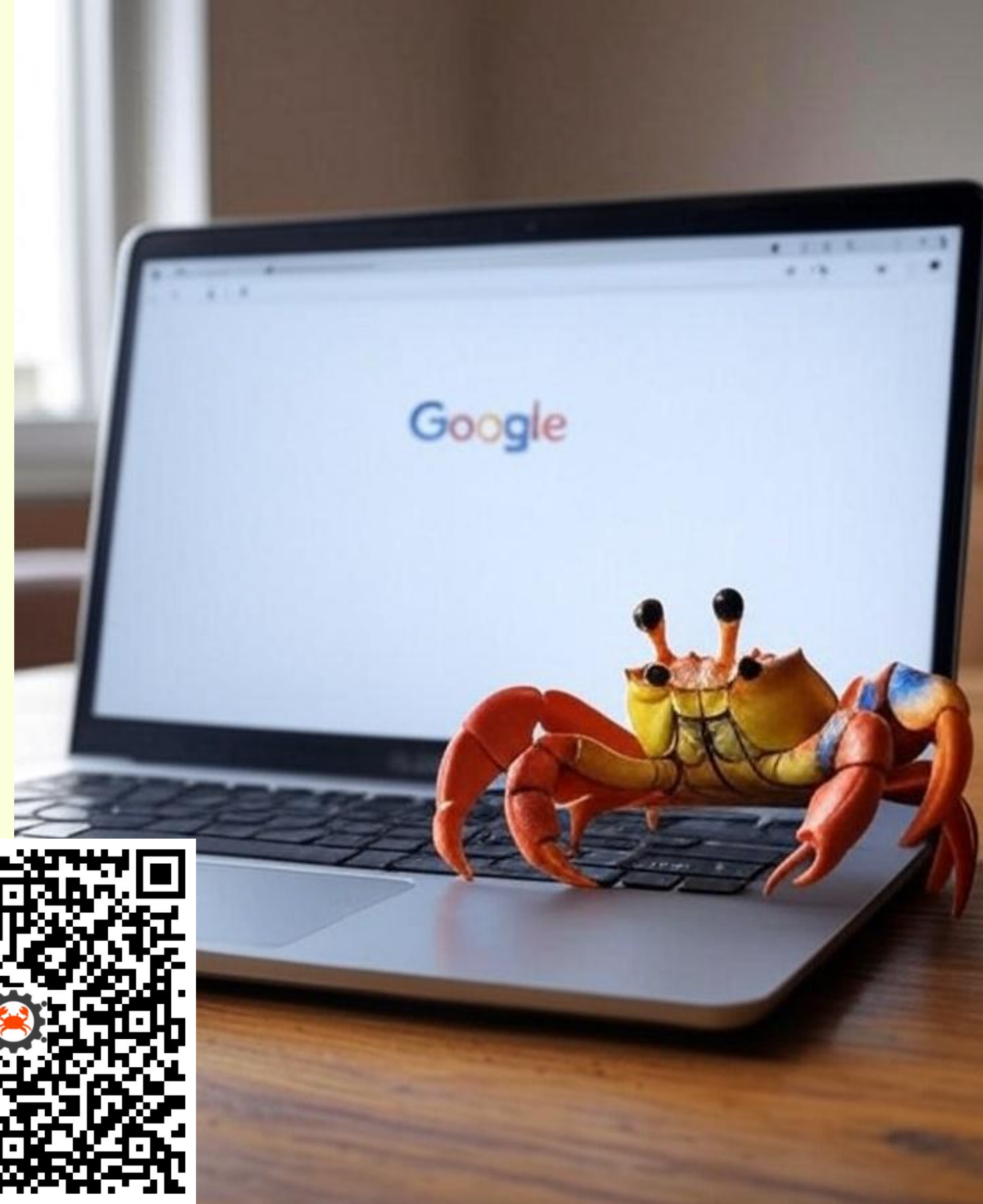
Velocidades cercanas al código nativo.

🔒 **Seguro (sandboxed)**

Todo acceso a recursos externos debe pasar por el entorno del navegador o APIs controladas.

🌐 **Portable**

Se puede ejecutar en cualquier navegador.



¿Por qué si se llama **WebAssembly** también corre en servidores?



Creado para correr código de alto rendimiento en el navegador, como alternativa a JavaScript.



Runtimes para ejecutar .wasm en backend:

Wasmtime

<https://wasmtime.dev>

 **wasmer**





<https://wasmer.io>



¿Porque Rust?

Rust compila a WASM sin dolores de cabeza.

Tooling sólido

-  **cargo** (compila)
-  **wasm-pack** (empaqueta, JS bindings)
-  **wasm-bindgen** (conecta WASM + JS)
-  **wasm-opt** (optimiza .wasm)

Sin Garbage Collector



El JS sí tiene GC, pero ese GC solo afecta al código y objetos gestionados por el motor de JS (V8, SpiderMonkey, etc.).



¿Por dónde comenzar?

- **Instalar Rustup**

<https://www.rust-lang.org/tools/install>

Herramienta oficial para instalar y gestionar Rust.

- **Instalar wasm-pack**

<https://rustwasm.github.io/wasm-pack/installer>

Empaquetador para compilar código Rust a WebAssembly.

- **Instalar npm**

<https://nodejs.org/es>

[Node.js](https://nodejs.org/es) & npm necesario para gestionar deps JS.



¿Por dónde comenzar?

1. Crear una APP con Vite.

```
$ mkdir rust-wasm  
$ cd rust-wasm  
$ npm create vite@latest js -- --template react  
$ cd js  
$ npm install
```

```
1 rust-wasm/  
2 └─ js/  
3   └─ index.html  
4   └─ src/  
5       └─ main.tsx / main.jsx  
6       └─ App.tsx / App.jsx  
7   └─ package.json  
8   └─ tsconfig.json  
9   └─ vite.config.ts / js
```

Raíz del proyecto
Todo el código relacionado a JS

Código fuente React



¿Por dónde comenzar?

2. Crear proyecto con Rust.

```
$ mkdir wasm
```

```
$ cd wasm
```

```
$ cargo init --lib
```

```
1 rust-wasm/
2 |   js/                                     # Todo el código relacionado al JS
3 |   |   index.html
4 |   |   src/
5 |   |   |   main.tsx / main.jsx
6 |   |   |   App.tsx / App.jsx
7 |   |   package.json
8 |   |   tsconfig.json
9 |   |   vite.config.ts / js
10 |
11 |   wasm/                                  # Código fuente WASM
12 |   |   src/
13 |   |   |   lib.rs
14 |   Cargo.toml
```



Dibujando a Ferris the Crab con WASM



Podes encontrar el código en:
oxidar-org/oxidar-wasm



Y por último un ejemplo!



MUCHAS GRACIAS!

Déjanos tus PRs y
comentarios.

github.com/oxidar-org

