

Rust Front-end with Yew



Ciro Nunes 14 Sep 22

Agenda

!? What is Yew? Why Wasm & Rust?

⭐ Simple tutorial

🤩 My impressions

🗺 Where to go from here



What is Yew? Why Wasm & 🦀?



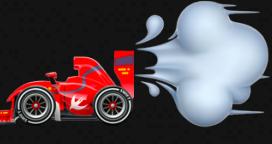
What is Yew?

Rust / Wasm client web app framework

Yew

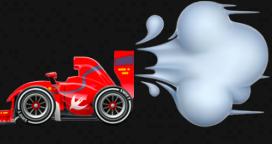
Yew

Multi-threaded front-end web apps using WebAssembly



Yew

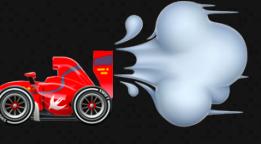
Multi-threaded front-end web apps using WebAssembly



- Component-based

Yew

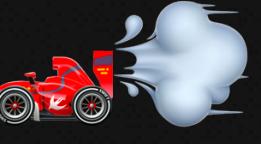
Multi-threaded front-end web apps using WebAssembly



- Component-based
- Great performance

Yew

Multi-threaded front-end web apps using WebAssembly



- Component-based
- Great performance
- JavaScript interoperability

The background features a dark blue gradient with three distinct wavy layers. The top layer is a lighter shade of blue, the middle layer is a medium shade, and the bottom layer is a darker shade. These waves create a sense of depth and motion.

Why Wasm?

Low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages like Rust with a compilation target that can run in modern web browsers.

WebAssembly

WebAssembly

Not a silver bullet for improving performance of web apps 😜

WebAssembly

Not a silver bullet for improving performance of web apps 😜

- DOM APIs from Wasm are still slower than directly from JS

WebAssembly

Not a silver bullet for improving performance of web apps 😜

- DOM APIs from Wasm are still slower than directly from JS
- Perfect for heavy computation applications

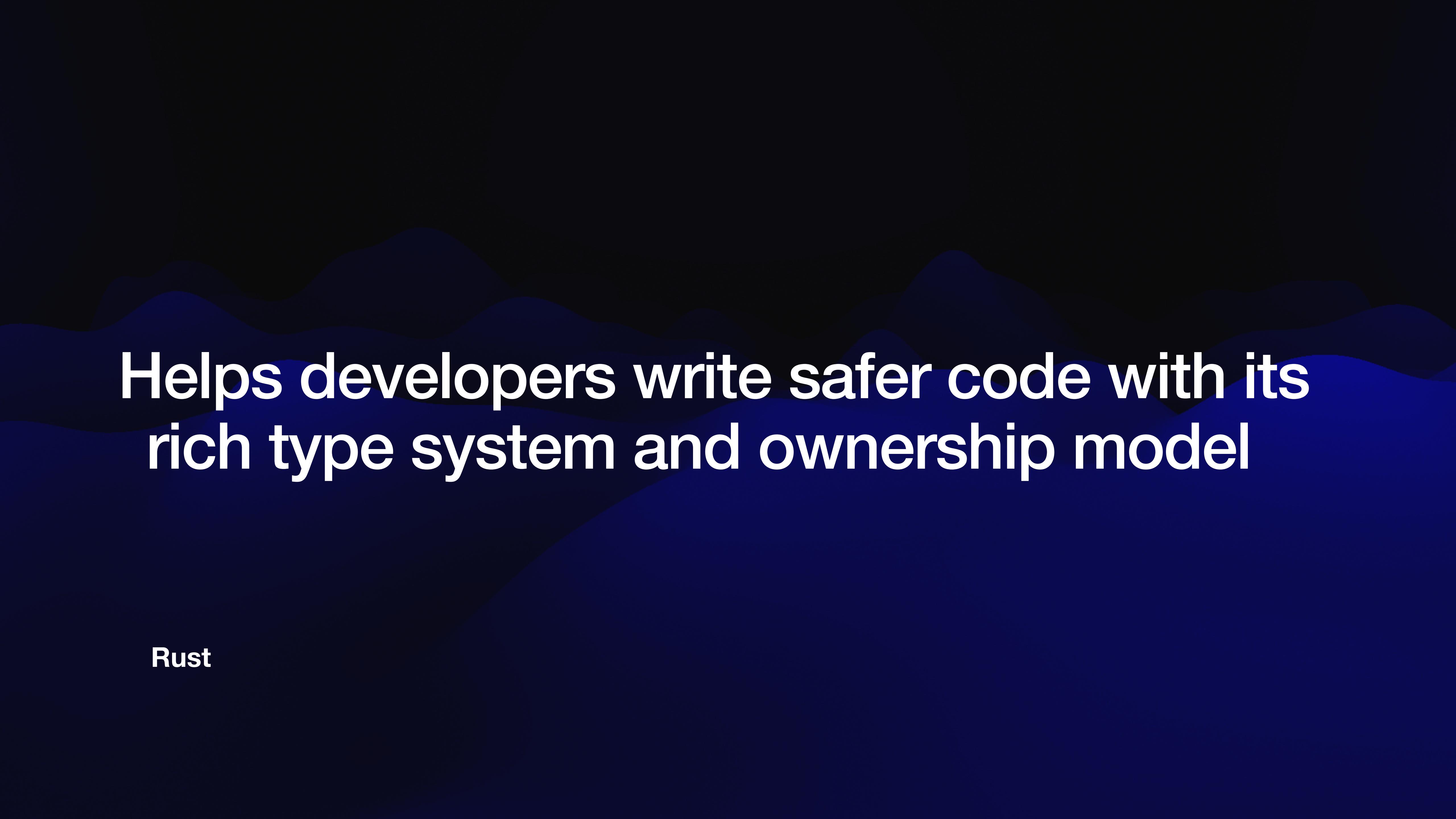
WebAssembly

Not a silver bullet for improving performance of web apps 😜

- DOM APIs from Wasm are still slower than directly from JS
- Perfect for heavy computation applications
- Multi-threading via Web Workers



Why Rust?

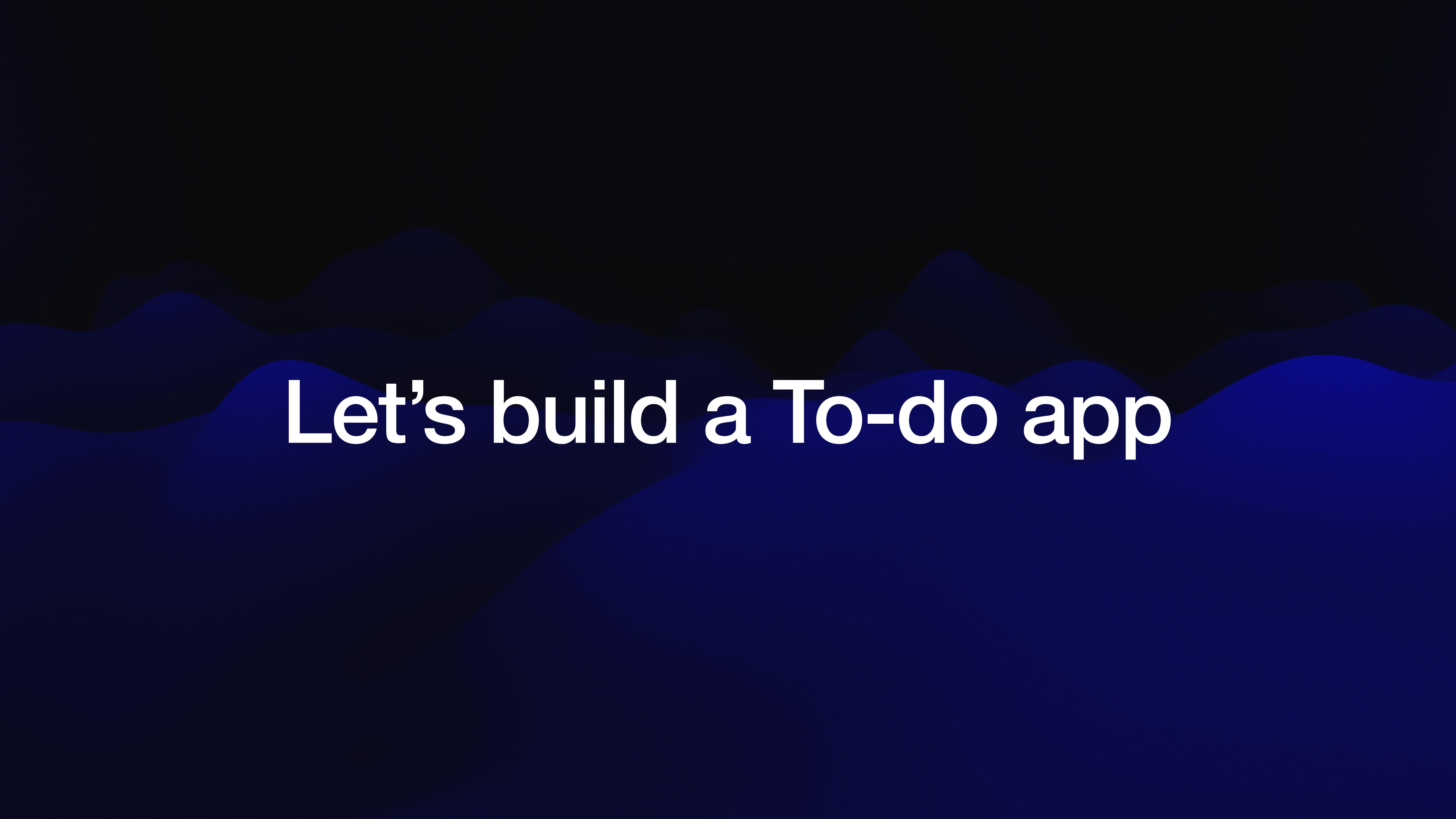


Helps developers write safer code with its rich type system and ownership model

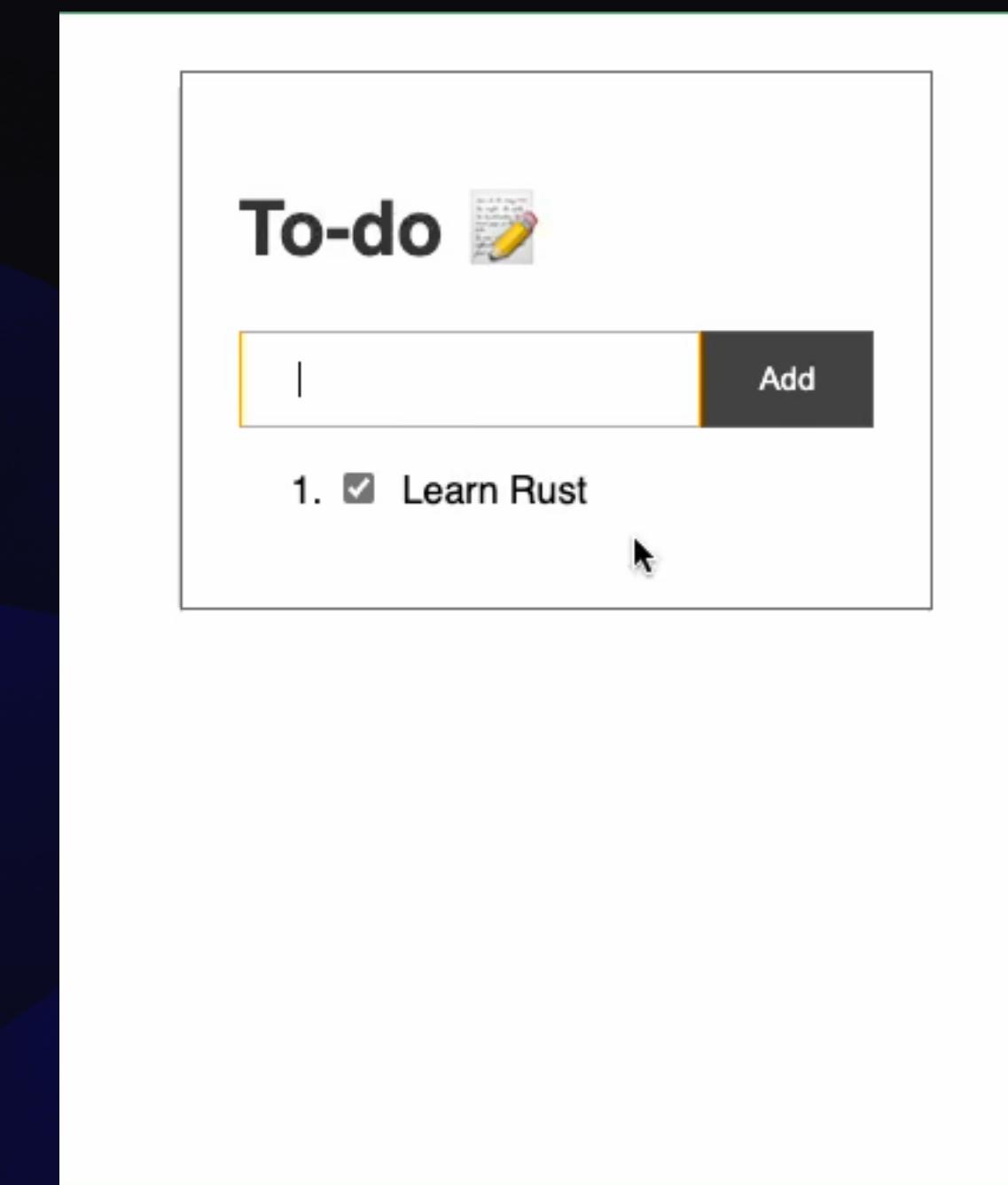
Rust

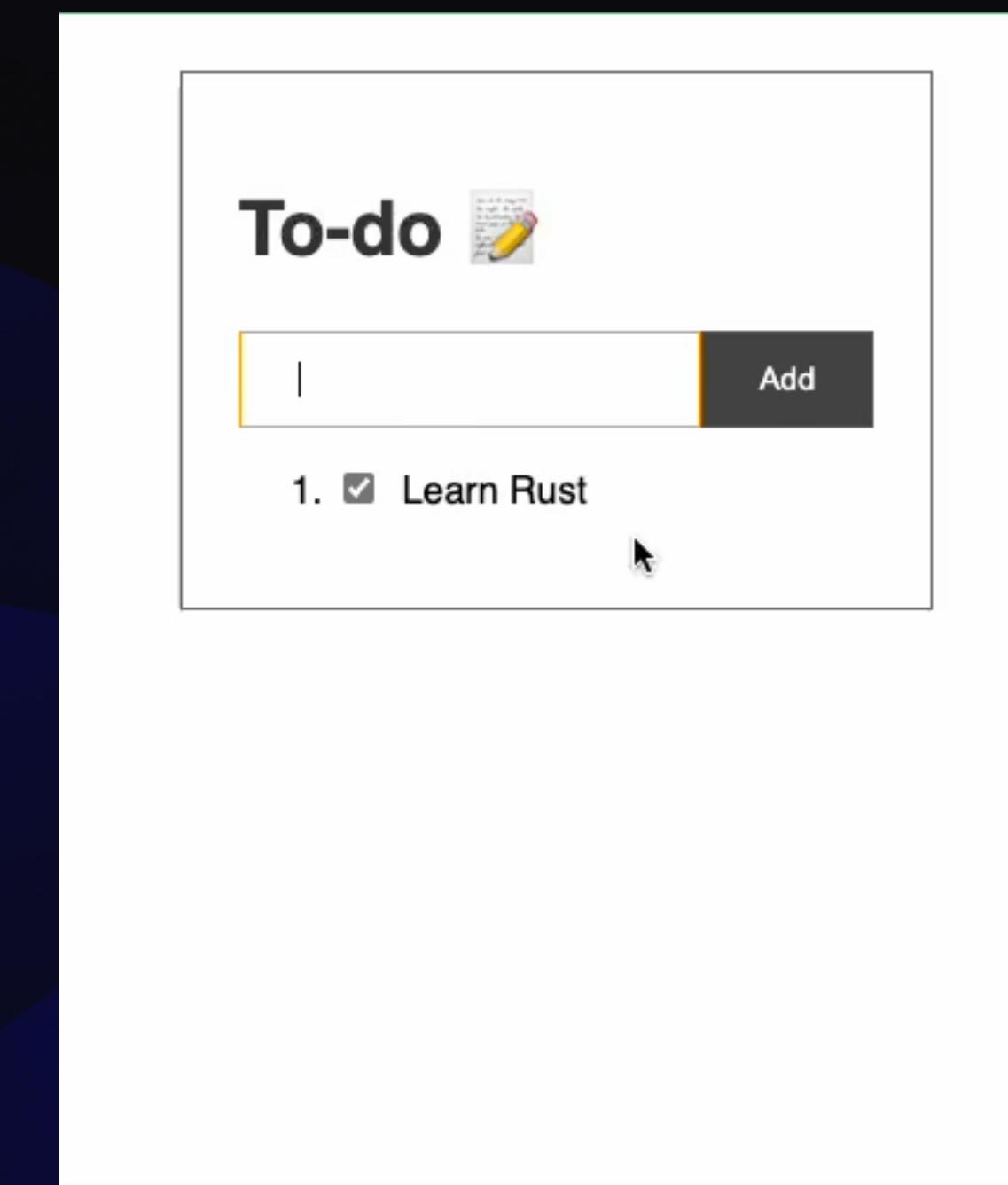
Simple tutorial



The background features a dark blue gradient with three prominent, wavy horizontal bands. The top band is a lighter shade of blue, while the middle and bottom bands are a darker shade. The wavy lines create a sense of depth and movement.

Let's build a To-do app





```
#[derive(Clone)]
struct Todo {
    text: String,
    completed: bool,
}
```

```
struct TodoComponent {  
    todos: Vec<Todo>,  
    new_todo_txt: String,  
}
```

```
enum Msg {  
    AddTodo,  
    UpdateNewTodo(String),  
}
```

```
impl Component for TodoComponent {  
    type Message = Msg;  
    type Properties = ();  
  
    fn create(_ctx: &Context<Self>) → Self {}  
  
    fn update(&mut self, _ctx, msg: Self::Message) → bool {}  
  
    fn view(&self, ctx: &Context<Self>) → Html {}  
}
```

```
fn create(_ctx: &Context<Self>) → Self {  
    Self {  
        todos: vec![Todo {  
            text: "Learn Rust".to_string(),  
            completed: true,  
        }],  
        new_todo_txt: "".to_string(),  
    }  
}
```

```
fn view(&self, ctx: &Context<Self>) → Html {
    let link = ctx.link();
    let on_cautious_change = link.batch_callback(|e: Event| {
        let input = e.target_dyn_into::<HtmlInputElement>();

        input.map(|input| Msg::UpdateNewTodo(input.value()))
    });

    html! {
        <div>
            <ol>
                { self.todos.iter().map(|todo| html! {
                    <li>
                        <input type="checkbox" checked={todo.completed} />
                        <span>{format!("{}", todo.text)}</span>
                    </li>
                }).collect::<Html>())
            </ol>

            <div>
                <input type="text" value={self.new_todo_txt.to_string()} onchange={on_cautious_change} />
                <button onclick={link.callback(|_| Msg::AddTodo)}>{ "Add" }</button>
            </div>
        </div>
    }
}
```

```
fn update(&mut self, _ctx: &Context<Self>, msg: Self::Message) → bool {  
    match msg {  
        Msg::AddTodo ⇒ {  
            let new_todo = Todo {  
                text: self.new_todo_txt.to_string(),  
                completed: false,  
            };  
            self.new_todo_txt = "".to_string();  
            self.todos.push(new_todo);  
            true  
        }  
        Msg::UpdateNewTodo(new_txt) ⇒ {  
            self.new_todo_txt = new_txt;  
            true  
        }  
    }  
}
```

```
use yew::prelude::*;

fn main() {
    yew::start_app::<TodoComponent>();
}
```



My impressions

My impressions

Yew

My impressions

Yew

- Very similar to React, but it's Rust

My impressions

Yew

- Very similar to React, but it's Rust
- Easy to setup and learn the basics

My impressions

Yew

- Very similar to React, but it's Rust
- Easy to setup and learn the basics
- Good documentation

My impressions

Yew

- Very similar to React, but it's Rust
- Easy to setup and learn the basics
- Good documentation
- Looks promising

A black and white photograph of a rugged mountain range. The foreground shows a steep, rocky slope with sparse vegetation. In the background, several sharp, rocky peaks rise against a sky filled with soft, diffused clouds. The lighting creates strong shadows and highlights on the mountain faces.

Where to go from here

Where to go from here

Yew

Where to go from here

Yew

- Check out the docs [yew.rs](#)

Where to go from here

Yew

- Check out the docs [yew.rs](#)
- Learn about Properties, Function Components, Hooks, Agents and the Router

Where to go from here

Yew

- Check out the docs [yew.rs](#)
- Learn about Properties, Function Components, Hooks, Agents and the Router
- Checkout [stylist](#) for CSS-in-Rust

Where to go from here

Yew

- Check out the docs [yew.rs](#)
- Learn about Properties, Function Components, Hooks, Agents and the Router
- Checkout [stylist](#) for CSS-in-Rust
- Take advantage of Rust parallelism in Wasm for heavy computations

Thanks for listening 🤝