

Rust for Backend and Frontend Development
International Acceleration with Rust

■ 1. Introduction

Framework comparisons

Foreword

It is important to note that the technologies being demonstrated here are all self-declared **not** production ready.

Therefore, expect bugs and inconsistencies, and apply the knowledge acquired here with caution.

With that being said, the APIs of the used frameworks have mostly stayed the same throughout the last few years, and, while breaking changes are bound to happen, it's safe to assume most of what is being presented today will remain relevant once they stabilize.

What we will be learning today

- How to write a backend Rust application from scratch
 - REST API / JSON
 - Database connection
 - ORM
 - Many-To-Many relationships
- How to write a frontend Rust application from scratch
 - SSR vs CSR
 - Hydration
 - Backend resource fetching
 - Reactivity

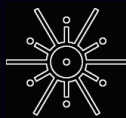
What we won't be learning today

- How to style components
 - That's beyond Rust and Leptos: that's CSS
- Authorization/Authentication
 - There's not enough time to cover both backend and frontend while also dealing with this
 - Besides, there are multiple ways of doing these, and they all depend on the context of the application
 - Take a look at <https://loco.rs/> for an integrated built-in solution

Backend Framework Comparison

Currently, Rust features 3 major backend frameworks:

Axum



- Newest/Freshest ✓
- Fastest practical web framework ✓
- Almost no macro usage ✓
 - leads to faster compile times
- Core of <https://loco.rs/> ✓
- No official guide yet ✗

Rocket



- Oldest web framework ✓
- Almost 1:1 with Actix Web's API
- **Fantastic** official guide ✓
 - Beginner friendlier
- Most built-in integrations ✓
 - `diesel`, `sqlx`, `rustqlite`, `memcache`, `tera`, ...
- Maintenance problems ✗

We'll be using Rocket today!

Actix Web



- Richest Ecosystem ✓
 - 876 results on <https://crates.io/>
- Built-in test client ✓
- Very detailed official guide ✓

ORM Comparison

■ Diesel



- SQL-first approach ✗
 - Rust types based off SQL's
- Features an official `rocket` integration ✓
- Few concepts to learn before using it ✓
 - All core traits are `derive`able

■ SeaORM



- Rust-first approach ✓
 - Migrations written in Rust
- Async-first ✓
- Lots of concepts and traits to learn before using ✗
 - Non-`derive`able traits
 - Naming-sensitive `structs` and `enums`

SeaORM is far too complex for the time we have, so **Diesel** is gonna be our ORM of choice today.

Frontend Comparison

Rust has a huge frontend ecosystem, with crates such as Leptos, Yew, Dioxus, Sycamore and Slint UI.

Leptos and Yew are web-centered and feature syntaxes heavily inspired by React and JSX/TSX.

Dioxus and Sycamore use syntax similar to HTML, and Dioxus specifically can be used to build web, desktop and even terminal interfaces.

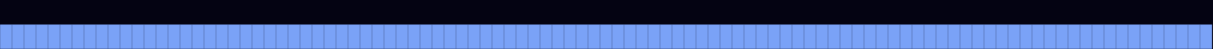
Slint UI brings own a whole new file format (.slint) with its own styling and interface syntaxes. Its web-export is still beta, however.



■ Leptos

For the purposes of today's demo, we are gonna be using Leptos, which brings a more familiar syntax, while having less boilerplate than Yew.

```
1 #[component]
2 fn Doubler() -> impl IntoView {
3     let (value, set_value) = create_signal(1);
4     let on_click = move |_| {
5         set_value.update(|val| *val *= 2);
6     };
7
8     view! {
9         <button on_click>Double it</button>
10        <p>Value: {value}</p>
11    }
12 }
```



2. Rust Refresher

Refreshing some concepts that will be used by the frameworks of our choice.

2.1. Types of macros

Function-macros

```
1 // Leptos example
2 view! {
3     <button on_click=|_| logging::debug_warn!("Hi!")>
4         "Check your console!"
5     </button>
6 }
```

- Used like functions, end with an exclamation mark `!`
- Don't necessarily use round parentheses `()`, can use any of the 3
- Manipulate the tokens received to change the whole expression

Attribute-macros

- Annotates items such as structs (and fields), enums (and variants), functions, etc.
- Also replaces the whole expression with zero or more outputs

```
1 // Rocket Example
2 #[get("/")]
3 async fn hello_world() -> &'static str {
4     "Hello, World!"
5 }
```

Derive-macros

```
1 // Serde example
2 #[derive(Serialize, Deserialize)]
3 struct LotteryResponseJson {
4     winning_numbers: [u8; 6],
5     your_numbers: Vec<u8>,
6     win: bool,
7 }
```

- Annotates structs and enums
- Used to automatically implement traits with logic that depends on reflection
 - Ex: Debug, Clone, serde's Serialize and Deserialize

2.2. Rust's Memory Model and Multi-Threading

In Rust, memory from the stack is often shared mutably or immutably via regular references `&mut` and `&`. These references are valid for as long as the scope in which the variables were created is valid.

However, when dealing with multiple threads, no data from the stack can safely be shared. Therefore, it's common to `.clone()` or move the data before sending it out to another thread.

Closures

Closures are anonymous functions that capture the context around them. This means that closures can use data from the current scope, without needing to clone or own it (by borrowing).

Unsurprisingly, if a closure is sent to another thread, it may outlive the data it borrows from. Safe Rust, of course, wouldn't allow it, resulting in a compilation error.

```
fn main() {
    let y = {
        // Simulating the scope of another thread
        let x = String::from("X value");
        || x.as_str()
    };

    print_result(y);
}

fn print_result<'a, F: Fn() -> &'a str>(f: F) {
    println!("Res: {}", f());
}
```

```
--> t.rs:4:9
|
4 |         || x.as_str()
|         ^^ - `x` is borrowed here
|         |
|         may outlive borrowed value `x`
note: block requires argument type to outlive `'1`
--> t.rs:2:9
2 |         let y = {
|         ^
help: to force the closure to take ownership
      [...] use the `move` keyword
4 |         move || x.as_str()
|         +++++
```

On such cases, the `move` keyword can be used to force the closure to take ownership of the data it'd borrow from.

3. Project Structure

