Full Stack Rust with Leptos

2024-04-02 03:00:00 UTC

A bartender at a British pub with mood lighting

I was thrilled to have my talk accepted by Rust Nation UK, and give the first Rust conference talk about Leptos and using Rust for the full web stack. I've reproduced it below, in a fairly similar form. Enjoy!

In a lot of ways, I feel we've reached an inflection point for Leptos as a web framework, where the features and API has mostly solidified, and the benefits are becoming clear. Only time will tell if it spreads its wings and soars.

Common Criticisms I've Heard About Webassembly

1. The bundle size is too big
2. It's limited by a lack of direct DOM access for Webassembly
3. The startup time is too slow
4. Compiling takes too long, making iteration painfully slow.

I want you to keep these in mind as we move through the article, and if you have any others, feel free to let me know.

Performance

Let's start out with the Krausest JS Framework Benchmark, which many of you who've spent time thinking about web framework performance have probably already seen.

For anyone who's not familiar, the JS Framework Benchmark compares a large number of frontend frameworks on a variety of DOM manipulation operations: creating large numbers of rows in a table, deleting rows, swapping rows, and making fine-grained updates to individual rows. It measures the overhead of each framework relative to a baseline 'vanilla JS' implementation that is designed to represent the actual browser rendering time.

Leptos has always done well in this framework, handily beating React and Vue in most metrics, while not being able to catch vanillajs and solidjs. For some of the Leptos users I talked to, this benchmark was the initial hook that tempted them to try it. Let's talk about the numbers a bit.

Duration

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

Name Duration for...	vanillajs	svelte-v5.0.0-next.28	solid-v1.8.0	leptos-0.7-sledge-hammer-v0.7.0	leptos-0.7-v0.7.0	leptos-v0.6.9	vue-v3.4.3	angular-ngfor-v17.0.2	react-hooks-v18.2.0	alpine-v3.12.0
Implementation notes	772			1139	1139	1139				1139
Implementation link	code	code	code	code	code	code	code	code	code	code
create rows creating 1,000 rows (5 warmup runs).	33.9 ± 0.2 (1.00)	35.3 ± 0.2 (1.04)	35.4 ± 0.1 (1.04)	36.8 ± 0.1 (1.09)	39.8 ± 0.3 (1.17)	42.7 ± 0.1 (1.26)	41.3 ± 0.5 (1.22)	42.6 ± 0.2 (1.26)	43.1 ± 0.4 (1.27)	100.8 ± 0.3 (2.97)
replace all rows updating all 1,000 rows (5 warmup runs).	37.6 ± 0.2 (1.00)	40.0 ± 0.2 (1.06)	40.1 ± 0.4 (1.07)	41.0 ± 0.1 (1.09)	45.8 ± 0.1 (1.22)	47.9 ± 0.3 (1.27)	47.7 ± 0.6 (1.27)	50.4 ± 0.1 (1.34)	50.8 ± 0.3 (1.35)	122.5 ± 0.5 (3.26)

partial update updating every 10th row for 1,000 rows (3 warmup runs). 4 x CPU slowdown.	16.1 ±0.3 (1.00)	16.0 ±0.1 (1.00)	16.1 ±0.1 (1.01)	16.3 ±0.4 (1.02)	16.8 ±0.3 (1.05)	16.6 ±0.2 (1.04)	19.5 ±0.2 (1.22)	16.7 ±0.2 (1.05)	20.4 ±0.5 (1.28)	21.4 ±0.2 (1.34)
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	5.4 ±0.9 (1.04)	5.7 ±0.9 (1.09)	6.7 ±1.1 (1.28)	6.8 ±1.1 (1.31)	7.1 ±1.5 (1.36)	6.2 ±1.3 (1.19)	5.4 ±1.0 (1.02)	5.2 ±1.0 (1.00)	5.8 ±0.6 (1.12)	33.5 ±0.8 (6.41)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	18.6 ±0.3 (1.00)	19.9 ±0.5 (1.07)	19.7 ±0.3 (1.06)	19.4 ±0.2 (1.04)	19.1 ±0.2 (1.03)	19.3 ±0.2 (1.04)	21.0 ±0.4 (1.13)	165.2 ±1.4 (8.89)	159.4 ±0.8 (8.58)	33.8 ±0.4 (1.82)
remove row removing one row. (5 warmup runs). 2 x CPU slowdown.	15.5 ±0.1 (1.00)	16.0 ±0.1 (1.03)	15.9 ±0.1 (1.03)	16.2 ±0.2 (1.04)	16.0 ±0.2 (1.03)	16.1 ±0.1 (1.04)	19.3 ±0.1 (1.25)	16.6 ±0.2 (1.07)	18.0 ±0.1 (1.16)	25.2 ±0.1 (1.63)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	364.0 ±1.0 (1.00)	373.2 ±1.8 (1.03)	376.2 ±1.8 (1.03)	381.1 ±3.6 (1.05)	422.9 ±2.0 (1.16)	454.0 ±3.2 (1.25)	432.4 ±2.2 (1.19)	441.3 ±2.7 (1.21)	587.0 ±4.8 (1.61)	906.1 ±3.6 (2.49)
append rows to large table appending 1,000 to a table of 1,000 rows.	39.2 ±0.3 (1.00)	41.6 ±0.2 (1.06)	41.3 ±0.2 (1.05)	41.6 ±0.3 (1.06)	46.5 ±0.3 (1.19)	48.2 ±0.5 (1.23)	47.2 ±0.2 (1.20)	48.0 ±0.4 (1.22)	49.8 ±0.1 (1.27)	117.6 ±0.5 (3.00)
clear rows clearing a table with 1,000 rows. 4 x CPU slowdown. (5 warmup runs).	12.3 ±0.3 (1.00)	13.5 ±0.3 (1.10)	14.1 ±0.2 (1.15)	16.0 ±0.2 (1.31)	16.6 ±0.2 (1.35)	15.9 ±0.2 (1.30)	15.5 ±0.3 (1.26)	26.7 ±0.2 (2.17)	25.3 ±0.3 (2.06)	47.0 ±0.9 (3.83)
weighted geometric mean of all factors in the table	1.00	1.05	1.06	1.09	1.16	1.19	1.21	1.34	1.45	2.56
compare: Green means significantly faster, red significantly slower	compare	compare	compare	compare	compare	compare	compare	compare	compare	compare

Krauset JS Framework March 29, 2024

On this chart we have three Leptos versions benchmarked, the current one(Leptos 0.6), Leptos 0.7(alpha), and Leptos 0.7(sledgehammer), as well as a variety of JS options like React, Alpine, Vue, Svelte, Solid and more.

Leptos does quite well here, as mentioned previously, edging out Vue and handily beating Angular, React, and Alpine. Due to some changes in the benchmark, our overall weighted mean dropped compared to previous versions.

The leptos 0.7-sledgehammer version is faster, but as we'll see later on, we don't believe the benefits outweigh the drawbacks.

This alone handily debunks concerns 1, 2, and 3, but we'll talk about bundle size and startup time a bit more in the next two sections.

Memory

Memory allocation in MBs ± 95% confidence interval

Name	leptos-0.7-sledge-hammer-	leptos-0.7-v0.7.0	leptos-v0.6.9

	v0.7.0		
ready memory Memory usage after page load.	1.7 (3.80)	1.8 (3.93)	1.7 (3.75)
run memory Memory usage after adding 1,000 rows.	5.0 (2.76)	4.8 (2.64)	5.6 (3.06)
update every 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	5.1 (3.02)	4.9 (2.87)	5.5 (3.27)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	4.4 (7.26)	3.9 (6.49)	5.5 (9.18)
run memory 10k Memory usage after adding 10,000 rows.	35.0 (2.88)	31.9 (2.63)	40.7 (3.35)
geometric mean of all factors in the table	3.67	3.48	4.10

We benchmarked memory for the different versions, and 0.7 seems to have a nice reduction here as well. We don't compare to the JS frameworks because Webassembly tends to allocate in blocks, making the comparison meaningless.

You might notice that the sledgehammer version uses more memory in most of these cases, which may or may not be a signifigant concern for your usecase.

Startup Times

Startup metrics (lighthouse with mobile simulation)

Name	vanillajs	svelte-v5.0.0-next.28	solid-v1.8.0	leptos-0.7-sledge-hammer-v0.7.0	leptos-0.7-v0.7.0	leptos-v0.6.9	vue-v3.4.3	angular-ngfor-v17.0.2	react-hooks-v18.2.0	alpine-v3.12.0
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	1,877.2 (1.00)	1,876.4 (1.00)	1,877.3 (1.00)	2,032.2 (1.08)	1,894.8 (1.01)	1,895.2 (1.01)	2,176.9 (1.16)	2,646.2 (1.41)	2,565.2 (1.37)	2,036.7 (1.09)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	149.7 (1.00)	156.5 (1.05)	150.0 (1.00)	323.2 (2.16)	306.0 (2.04)	313.4 (2.09)	198.4 (1.33)	285.7 (1.91)	280.5 (1.87)	182.0 (1.22)
geometric mean of all factors in the table	1.00	1.02	1.00	1.53	1.44	1.45	1.24	1.64	1.60	1.15

One of the things I hear alot about is startup time, and that webassembly based frameworks have a consistently lower time to interactive than JS. From this chart, we can see that is not true. While the total kilobyte weight of the bundle is roughly double the size, the TTI is roughly equivalent to vanillajs.

How can that be?

As it turns out, it's a lot more efficient to load Webassembly into a browser, a binary format, than it is to parse JS through the JIT and load it. As it turns out we can load almost 2x as much Webassembly as JS for the same TTI. Neat huh?

For more details on that, check out this handy article from the Firefox team.

Again, the Leptos sledgehammer version suffers due to an increased bundle size, causing a fairly significant regression in TTI. Personally I'm of the opinion that the slight increase in performance is not worth the increased memory and bundle size tradeoffs, but it's a very interesting idea nonetheless.

Leptos has a variety of tricks to reduce bundle size, most of which aren't employed here, and 0.7 reduces this further as well. Well within the ballpark for a web framework.

Ben's Blog Test

The JS Framework Benchmark is all well and good, but I wanted to do a bit more real world test with server side rendering. For that, I built different versions of this blog, and measured how they performed under load. Test candidates are Leptos and Remix, both with identical HTML/CSS and as identical as possible page logic.

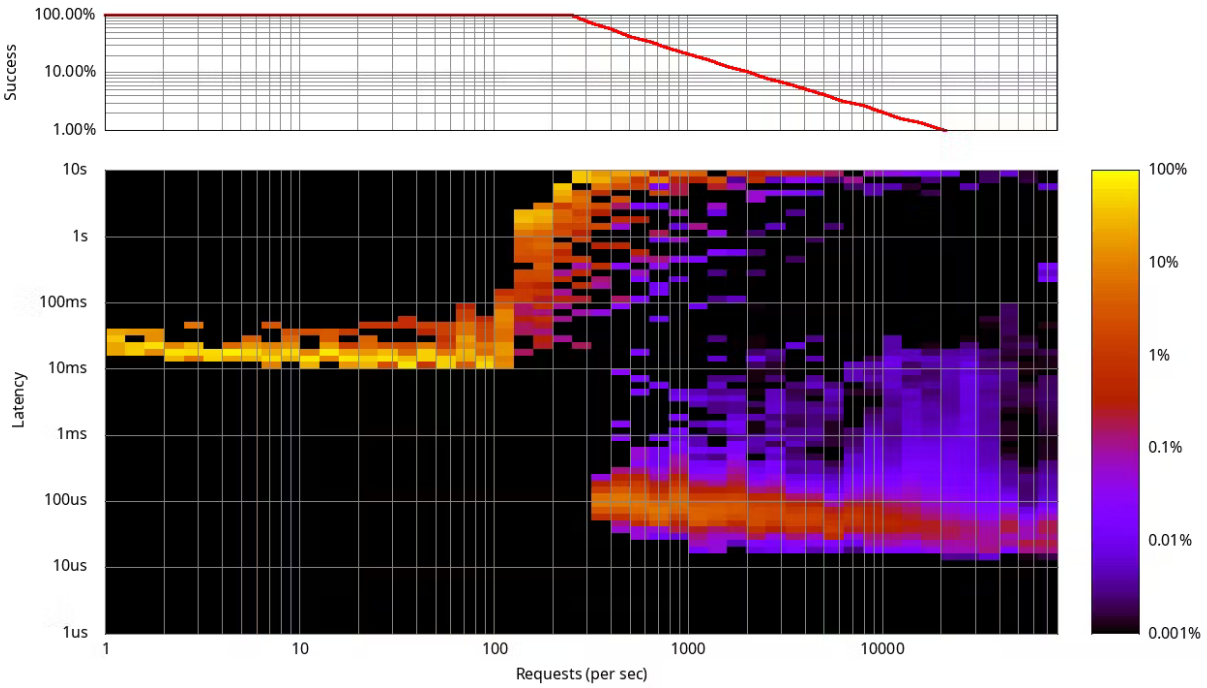
I tested the time to load the home page, which fetches the three most recent posts from the SQLite database, as well as reads cookies and does basic calcs to determine whether the user is logged in and what colorscheme they prefer.

For test hardware, I ran the server and the test client on a DigitalOcean Dedicated General Purpose Droplet with the following specs:

- Two dedicated AMD "vcpus"
- 4GB RAM
- 2Gbps bandwidth

Each server was benchmarked by vegeta, a Go based load tester, under varying levels of server load.

First up, Remix running on Express.js.



Remix and Express.js tested at different load levels

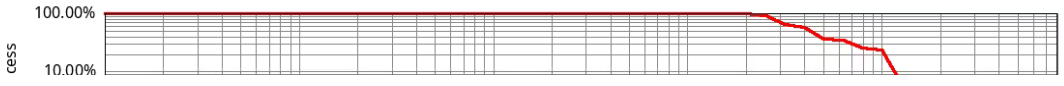
I love this graph, it's beautiful and informative. On the top we have the success rate of the request at different requests per second. The lines on that line up with the graph below.

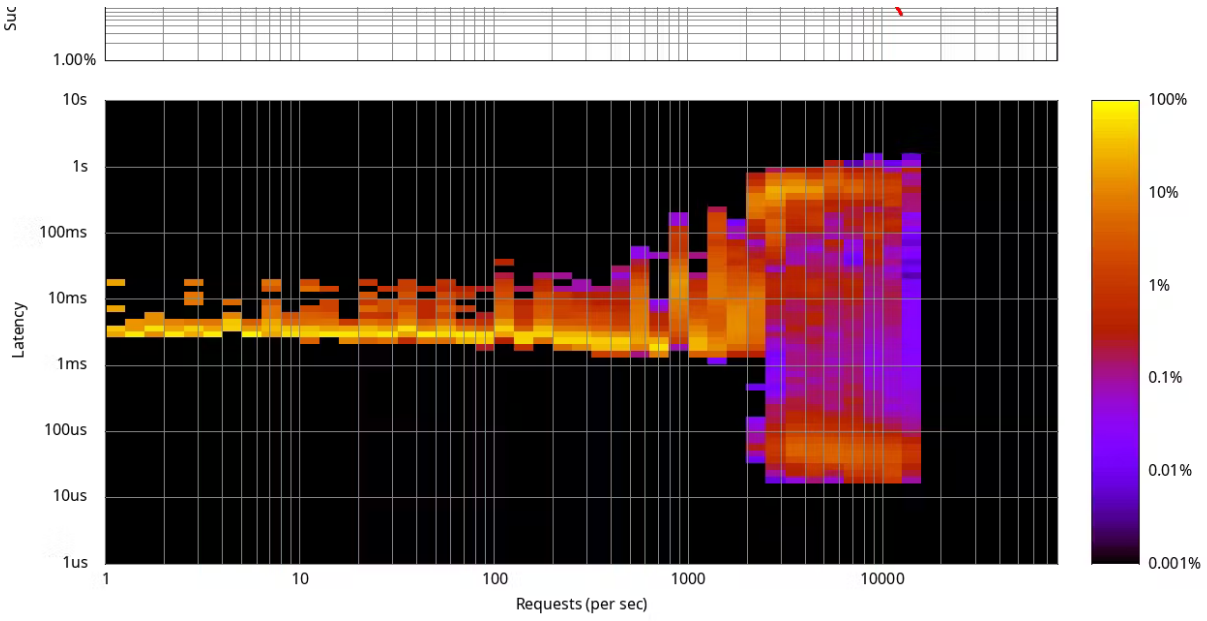
For the larger graph below, the Y axis denotes latency in milliseconds and the x axis requests per second on a logarithmic scale. The frequency of each latency is given a color to determine how common that latency was at the request level. Fun huh?

From this graph, we can see typical latency at no/low load starts at about **20ms**, and reduces to **10ms**, before fairly rapidly spiking to **8-10s** around **100rps**. Beyond that point, the latency drops to nothing, which I suspect means its basically fallen over and the data from there is invalid.

I believe the reason the JS version here is reducing latency as load increases is the JIT optimizer being able to optimize more and more as the number of requests increases, but I can't be sure.

Next, we have Leptos running on Axum:





Would you look at that! Average latency remains around **3-5ms** all the way up to **2000rps**, with a slightly widening variability as load increases. Unlike the Remix version, latency just barely exceeds one second instead of 10 seconds.

All in all, this means that for the same hardware, I can handle about **10x**, or **10,000%**, more requests per second than the JS one, which is massive. For commercial applications, that is a substantial decrease in infra costs and potentially increased ability to handle load spikes.

A good portion of this comes from how performant Axum is as a backend web server, but Leptos is no slouch here as well. Also worth noting that I haven't done any optimizations on either of these. The limit for this one appears to be the number of simultaneous file reads of the OS, as that was the error that started to appear. Perhaps if I stuck a cache in front of sqlite it could keep going here, but then I'd have to retest the JS one, so I opted not to do that for this post.

Type Reuse and Seamless API Endpoints

I'm going to pivot a bit here and talk about one of my favorite Leptos features, Server Functions. Server Functions are a tool Leptos provides to call Rust functions on the server as if they were local to the client.

This means you don't have to write a serializer or deserializer, you don't have

to write a validator, or add a route and a handler. All you do is write a Rust function, annotate it with the server macro, and then call it from the client.

```
Rust code

// Server Function
#[server]
pub async fn update_count(increment_by: i64) -> Result<
   let new_count= fetch_count() + increment_by;
   println!("Count is {new_count}");
    Ok(new_count)

// Call from Component
#[component]
pub fn Counter() -> impl IntoView{
    let update_count = create_server_action::<UpdateCou

    view!{
            <ActionForm action=update_count>
                <span>Increment By:</span>
                <input type="number" name="increment_by
                <button type="submit">Update</button>
            </ActionForm>
    }
}
```

This example is quite shortened, but serves the purpose. The server macro above will create a client side and server side version of the function, handling route generation, serialization and deserialization, and validation.

On the client, we can use an ActionForm, which is just a form with the action parameter filled with the autogenerated api endpoint url. Since the server function takes an argument `increment_by` we add an input field with that name.

That's it! ActionForm will even work with JS and WASM disabled, since the default server encoding is regular formdata via a POST request.

Tooling

I think there's an argument to be made that all of the work Rust has done on helpful error messages, cargo, and the surrounding ecosystem have produced top notch tools.

Odds are that if I come back to a Rust project, it'll still be working 6/12/18/24 months later. I wish I could say the same for the Node/Python projects I've worked on.

In the end its subjective, but I'd much prefer to use cargo, rust-analyzer, rustfmt, leptosfmt, and cargo-leptos than tsc/NPM/Vite/Prettier/Eslint.

I'm fascinated by the number of the above mentioned tools being rebuilt with Rust or Go for speed reasons. There's probably at least two different competing attempts too!

Commercial Feedback

Being a new web framework, there's definitely some of the chicken and egg problem. Companies (and some developers) have asked "Are there any companies using it?", and that's a fair question. I'm glad to say that there are numerous commercial users, some of which I talked to and polled for the next few sections. Below are links to their sites, all of which see some fairly heavy use.

Patr: Web Hosting Platform

> The rust part of it + reactivity brings amazing benefits to making sure that we spend (a little bit) more time building our application and almost no time debugging the version that's already running. So most of our time is spent on building new features and focusing on the product / user experience rather than fixing bugs and pushing patches. - Rakshith Ravi - VP Engineering, Patr

This is one of my favorite quotes, and lines up nicely with my own experiences.

The idea that spending a bit more time to make the site correctly is worth the savings in debugging and maintenance.

Further it meshes with one of my core beliefs, that rapid iteration time is not the be all end all of web development or startups. Often time it is not the the startup that's first to market that suceeds, it's the one that delivers a good experience. According to Business Insider, first movers are more than six times as likely to fail as fast followers.

Houski: World's Largest Open Source Property Database

> Leptos, and Rust, have allowed us to build a complicated site with a very small team and very good performance. This project would probably not be possible with a traditional stack - Alex, Houski

Upon reviewing these, I'm beginning to notice a new trend, that Rust and Leptos has allowed them to do more with less. It makes some sense, as the more work the tooling and compiler does, the less the developer needs to do and remember.

I'm sure we've all heard or been the subject of a story about a developer forgetting about an invariant, or making an obvious mistake. We're human after all. I'm not here to say that Leptos will eliminate mistakes, only that it will probably reduce them.

Rust Adventure: Chris Biscardi's Rust Web Training Platform

> Having a language built with a type system from the beginning combined with a framework that is competitive with modern JS frameworks and all of the use cases that implies means that I can build comparable sites to what I've done my entire career with far less cognitive overhead. - Chris Biscardi, Rust Adventure

More supporting evidence for the above, from the lovely Chris Biscardi. Check him out if you're interested in doing some guided Rust workshops.

CBVA: California Beach Volleyball Association

> Leptos is essentially taking all the benefits of Rust and marrying them to all the benefits of Signals & SSR... I have done truly nothing to optimize yet and I already have top notch time to paint and time to reactive. Even on poor LTE beaches. - Alex, CBVA

This one makes me chuckle, it just seems so random. The CBVA as an early adopter of a web stack. Still, they seem to be enjoying the performance, and we're glad to have them.

Compile Times

Ah, concern #4. Thought I forgot about that one didn't ya? This one is usually tied to concerns about iteration speed, because you can't go as fast if you're waiting for compilation all the time. I asked each of the aforementioned companies how long their incremental compile times were, and how they felt about it. Here's the numbers for each

- Patr: 5s
- CBVA: 5s
- Rust Adventure: 2s

Not terrible right? But frontend developers like to see thir changes instantly, which is a bit tricky with Leptos. Thankfully these times can be reduced further with a neat trick called Hot Reloading.

Hot Reloading

```bash
```

```
cargo leptos watch --hot-reload
```

In Leptos, if you use the above mentioned command, we'll send an HTML/CSS patch to the browser to instantly update the view while waiting for the rest to compile. It can make HTML/CSS changes feel downright speedy.

Leptos 0.7

Leptos 0.7 represents a fairly significant rewrite of the reactive core of Leptos, in a way that should be mostly transparent to users. We've figured out a better design that should yield numbers improvements, such as:

- **Send/Sync** support in reactive system
- Generic view-management layer that can be adapted to support multiple backend renderers (DOM via web-sys, DOM via sledgehammer, GTK, others)
- Ergonomic improvements for async data loading
- Smaller WASM binary sizes, lower browser memory use

The most exciting for me, as a Leptos developer, is Send/Sync support, which means we can remove some of the hackery around the Axum integration requiring all handlers to be Send/Sync.

For y'all this release makes it even easier to combine the Leptos reactive system with backend renderers, everything from GTK or Iced to one of the TUIs or alternative DOM renderers. I expect we'll see Leptos used in more projects moving forward.

Planned New Integrations

My friend and esteemed Rust engineer Luca Palmieri gave a talk about Pavex, his new Rust backend framework that focuses on user ergonomics, friendly error messages, and being batteries included a la Ruby on Rails. It's in closed beta right now, but you can sign up for the waitlist. I've begun integrating Leptos with Pavex, in the hopes that will lead to an even more user friendly full stack Rust web stack! Updates on that to come soon.

Besides that, I met the Fastly Edge Compute team, and we decided that it might be nice to integrate with their Webassembly Edge framework. Details

might be nice to integrate with their Webassembly Edge framework. Details TBD.

It was also nice to meet the Shuttle folks. who are also building a neat Webassembly module platform. Due to an issue, Leptos with SSR is not working on the platform(without a hack), but they assure me they'll be fixing it soon. Fingers crossed!

Tooling Improvements

RustRover

It was also nice to see the RustRover team, who seem fully engaged in building a nice and powerful Rust IDE. We discussed some of the issues surrounding cfg attrs, html in view macros, and other QOL improvements. They promised to look into it, and I hope they do, but we're totally dependent on them.

Neovim, VsCode and Treesitter

On the open source front, Leptos user @rayliwell has created a treesitter grammar for rstml, which nicely handles HTML embedded in Rust. I've been testing it out and it quite nice. We're also looking at enabling auto closing html tags in rs files inside view macros, although details of that are a bit hairier.

Formatting and More

In the meantime, @bram209 continues to improve leptosfmt with formatting html in our files and cargo-leptos has added some new features surrounding build targets. Check out those projects for more details.

We have new updates of so many things, they're almost too numerous to describe here, so check out the list at awesome leptos.

The Conference Itself

First off, huge props to Ernest Kissiedu and the team at Vitis Events, the conference went off amazingly. The food was delicious, the talks varied and interesting, and the venue was beautiful. As far as Rust conferences I've been to go(Rust Conf 22, Rust Conf 23, and Rust Nation UK), this one takes the

cake.

As a speaker I missed a good portion of the talks, but I'm hoping to catch up on them later. It's always lovely to see familiar faces, and make new friends. I met alot of European Rustaceans and finally put some faces to Discord and Github handles.

Not to mention doing touristy things in London. The food, the pubs, the sightseeing. It was my first European city, and it was different. I won't go too far into that here though, because that is not what this is about

Conclusion

I'm honestly of the belief now that using Rust on the web is now a **competitive advantage** for any startup, company, or individual who uses it. Doing more with less more reliably is a compelling case. I'll sum up the benefits discussed earlier for those who skipped to the end:

1. Reduced page load time
2. Reduced infra costs by a large margin or Increased response handling
3. Rust's type system, error messages, and tooling
4. Server functions
5. Reduced developer time vs app complexity

I'm excited for Leptos in the year ahead, and to attend Rust Nation UK next year. Depending on how things work out, you might find me at Rust NL, Euro Rust, and Rust Conf, feel free to come chat. If I've done something wrong, have a question/comment, or if you just want to tell me how wrong I am, I'm on Mastodon at `@benwis@hachyderm.io`.

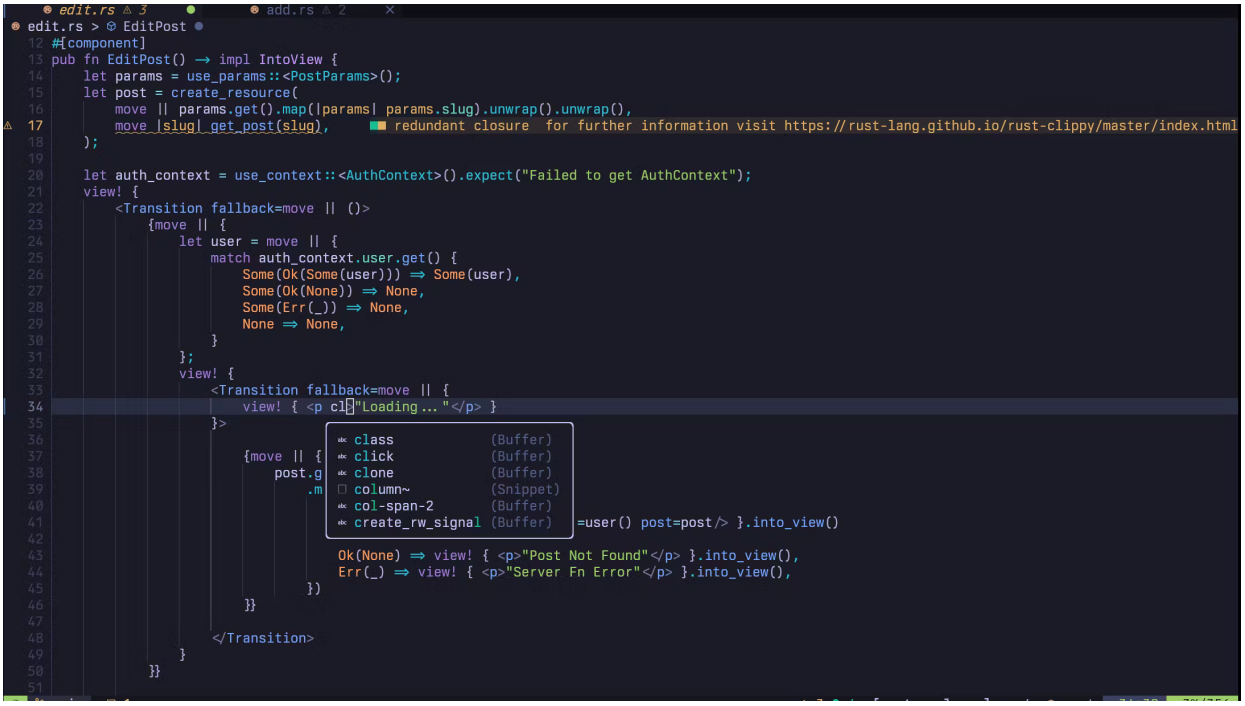
Previous

Compiling Rust to WASI

2024-03-18 01:00:00 UTC

Is WASI preview 2 ready for prime time? I explore using it for my Rust/C project to bring additional functionality to the browser!

Next



Easy Leptos Editor

2024-04-12 21:00:00 UTC

There's a subtle pleasure in having your editor setup just the way you'd like. When you've got good error messages, autocompletion, intellisense, and all the LSP goodies we as developers have come to expect. wanted to know if I could get the same experience writing Leptos components that I do writing React components. And the answer is yes! Setting it all up will require a little bit of work, but in the end we should have a Leptos IDE rivaling that of Javascript in Visual Studio Code!

Built with Leptos v3

Design by Underscorefunk Design

Signup

Login