



IDE-level coding in web apps

A POC PROJECT BUILD WITH RUST/AXUM/REACT

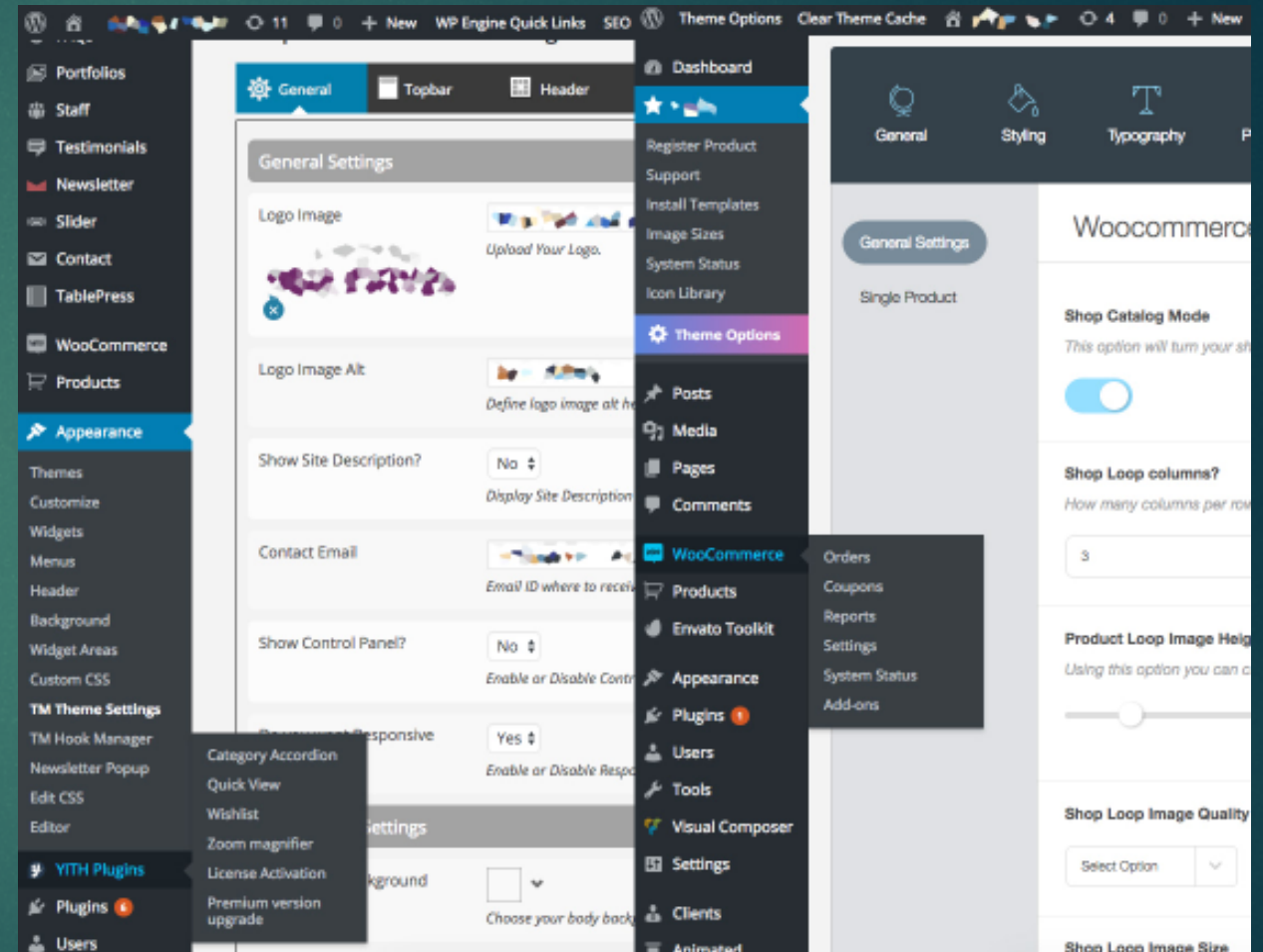
ROLAND BRAND, RUST ZURICH 2023-07-03

Use case

- ▶ Configurable systems
- ▶ Tools
 - ▶ UI (configurators, builders, wizards)
 - ▶ DSL "snippets"

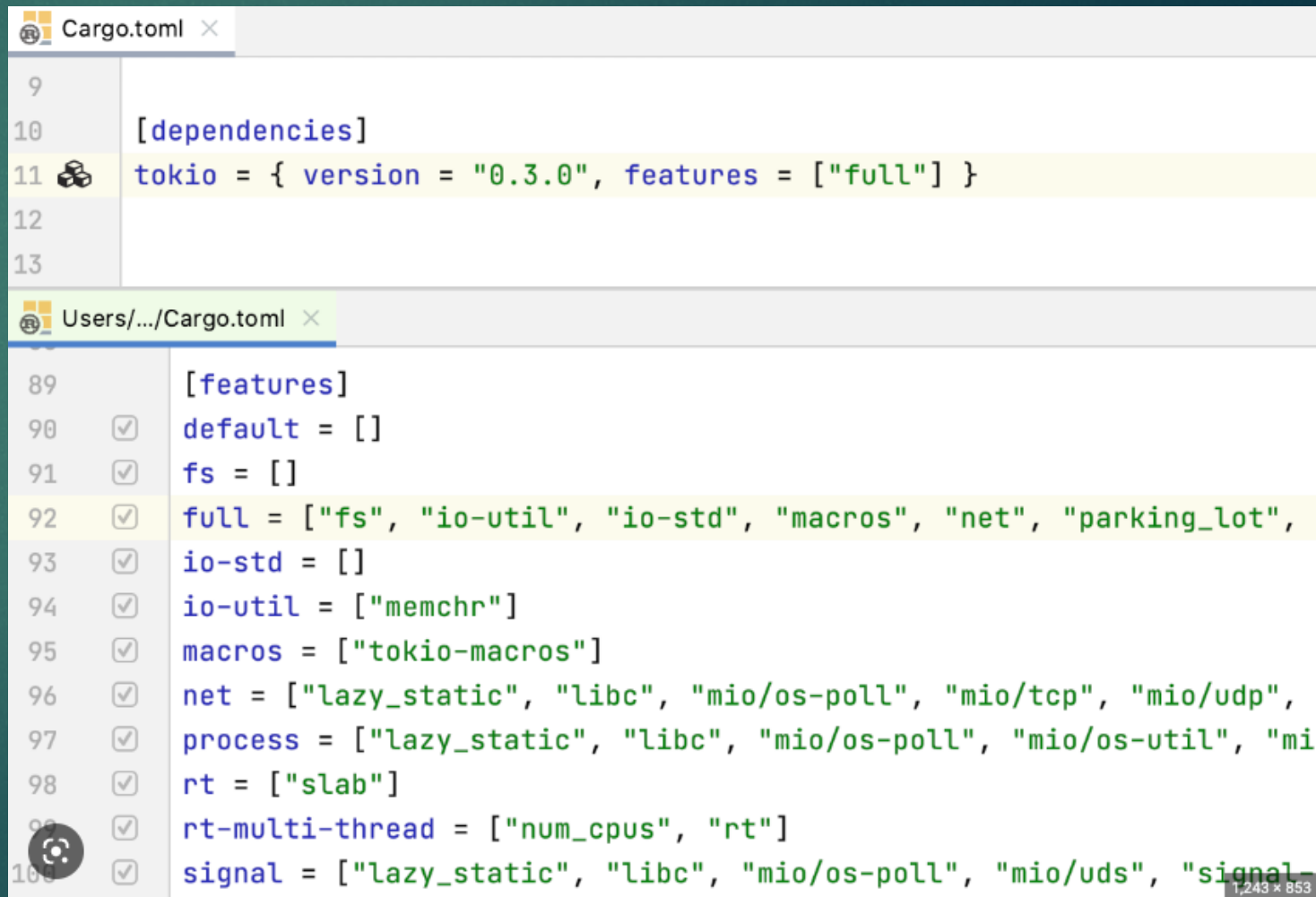
```
Constants:
3  #foo=xyzite/foohe/xy type=boolean label=Global no_cache
4  config.no_cache = 1
5
6  bgColor = red
7  topimg.width = 200
8  topimg.file.purl = filesdmain/logo2.gif
9  file.toplogo = filesdmain/logo.gif
10
11 #constantEditor_MySite (
12   header = Clearing the cache
13   description = This explains the constant, although it is really just a silly example
14   image = EXT:doc/tut/templating/Resources/Public/Images/Distribution.png
15   bulletList = Please note/What the numbered item/Is missing in the image
16   1 = config.no_cache
17 )
sys_template > constants

Setups:
1  page = PAGE
2  page.typekey = 0
3
4  page.bodytag = <body bgcolor="{{bgColor}}">
5  page.id = IMAGE
6  page.id.file = {{file.toplogo}}
7
8
9
10
11
12
13
14
15
sys_template > config
```



Developers

- ▶ We also configure a lot
 - ▶ Systems
 - ▶ Components
 - ▶ Libraries
- ▶ Our tools
 - ▶ IDE
 - ▶ Type systems
 - ▶ Syntax highlighting,
 - ▶ Linting
 - ▶ refactoring tools

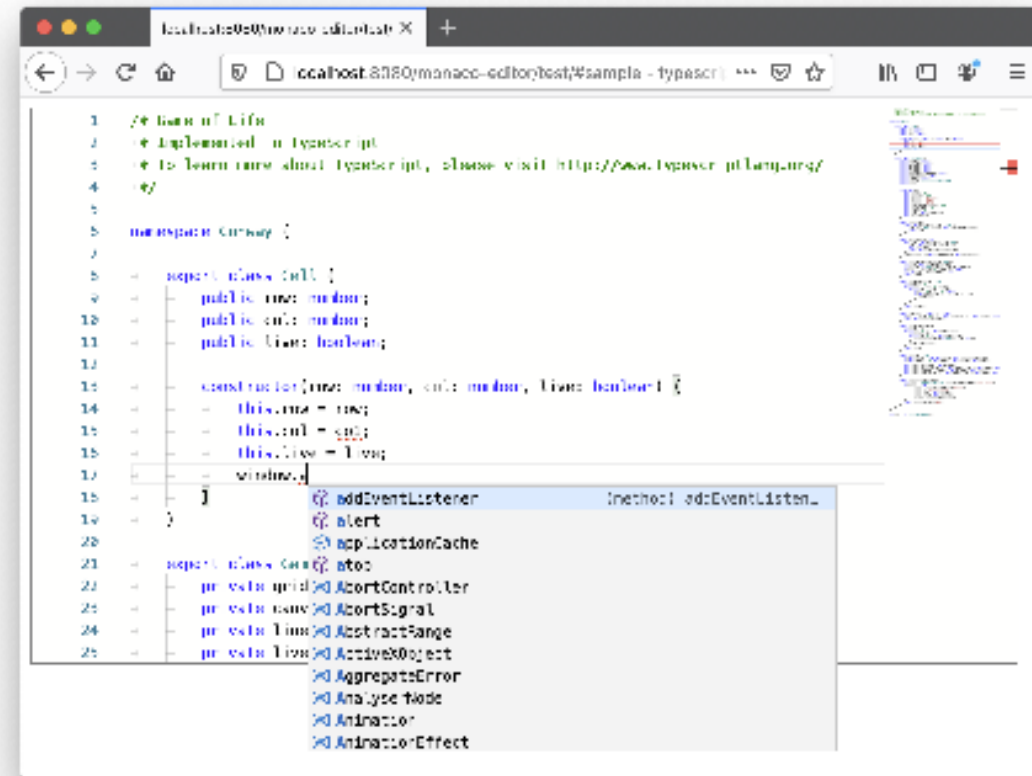


```
Cargo.toml x
9
10 [dependencies]
11 tokio = { version = "0.3.0", features = ["full"] }
12
13

Users/.../Cargo.toml x
89 [features]
90 default = []
91 fs = []
92 full = ["fs", "io-util", "io-std", "macros", "net", "parking_lot",
93 io-std = []
94 io-util = ["memchr"]
95 macros = ["tokio-macros"]
96 net = ["lazy_static", "libc", "mio/os-poll", "mio/tcp", "mio/udp",
97 process = ["lazy_static", "libc", "mio/os-poll", "mio/os-util", "mi
98 rt = ["slab"]
99 rt-multi-thread = ["num_cpus", "rt"]
100 signal = ["lazy_static", "libc", "mio/os-poll", "mio/uds", "signal
```

Why not give this to configurators?

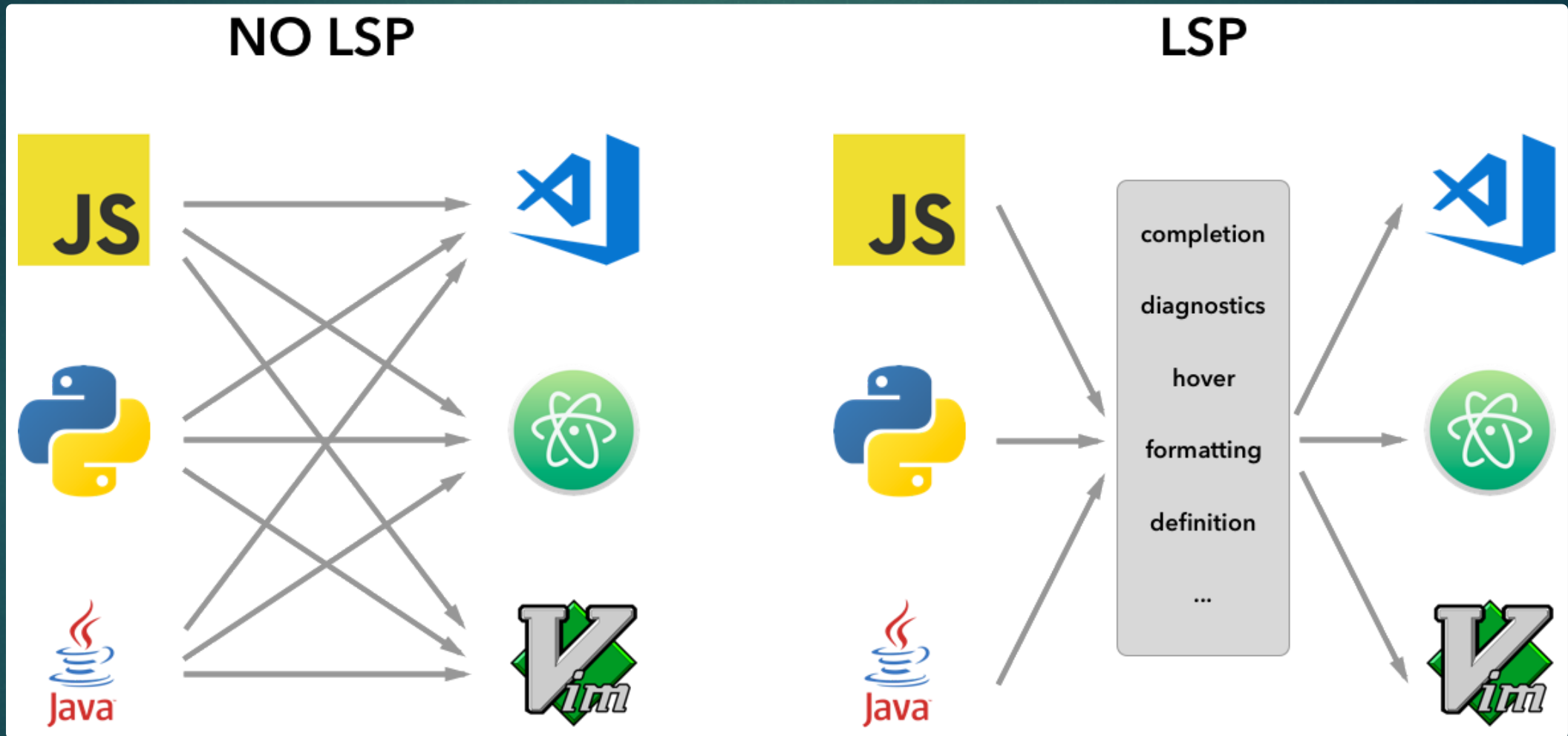
- ▶ Non-developers won't use an IDE because of setup and context
- ▶ Possible low-code Solution:
 - ▶ IDE within a context
 - ▶ Coding experience instead of clumsy wizards



POC goals

- ▶ Developer experience in textfield
- ▶ e2e System setup
- ▶ Showcase for a configurator

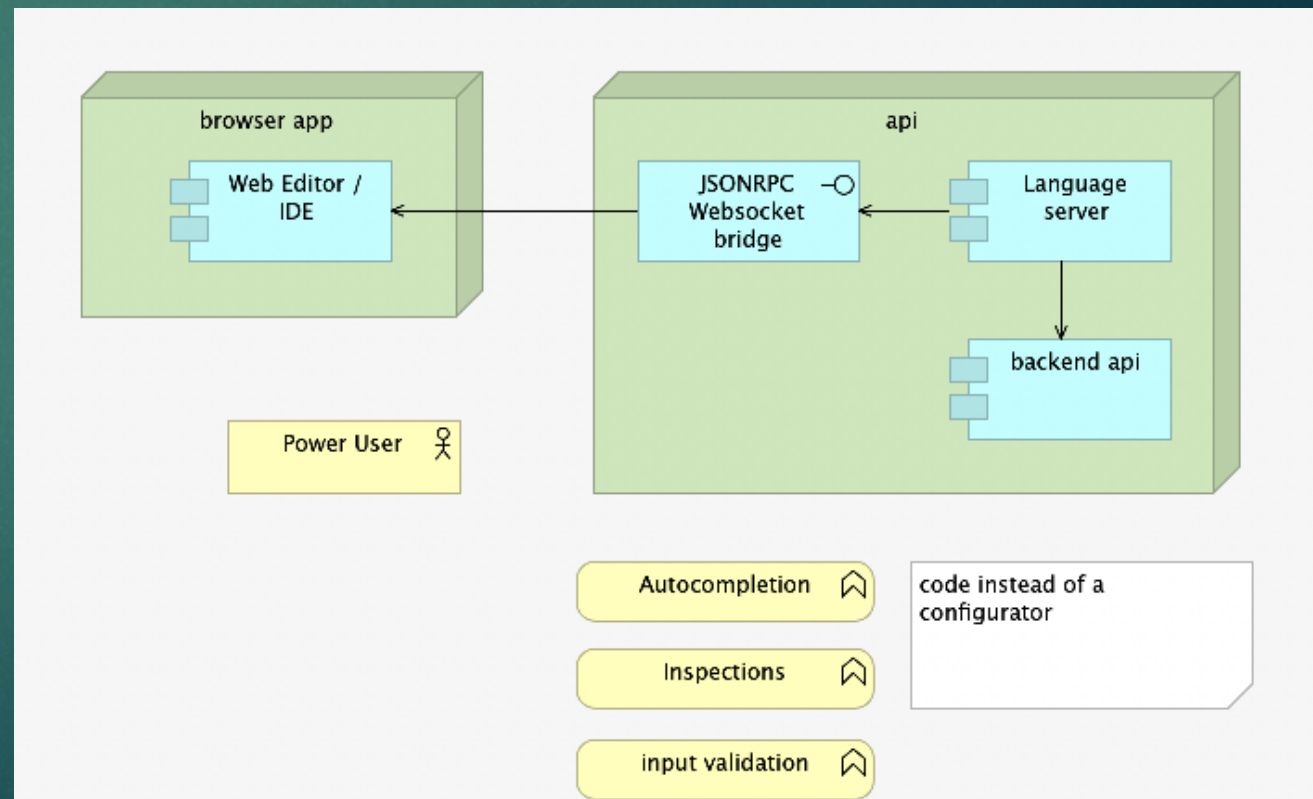
Language server protocol



Img src: <https://code.visualstudio.com/>

System design

- ▶ Rust based server API (that can be enhanced to a product backend)
- ▶ Communication:
 - ▶ JSONRPC over websockets
 - ▶ (inspectable)
- ▶ Front End: React



POC

- ▶ <https://github.com/bar9/lsp-expression-calculator>

```
async fn handle_socket(mut socket: WebSocket) {
    let (mut service : LspService<?> , _) = LspService::new(|_| Backend);

    while let Some(msg : Result<Message, Error> ) = socket.recv().await {
        let mut method : String = String::from( s: "");
        let res : Option<Response> = if let Ok(Text(msg : String )) = msg {
            let deserialized : Result<Request, Error> = serde_json::from_value::<Request>( value: msg.clone().parse().unwrap());
            if let Ok(deserialized : Request ) = deserialized {
                method = deserialized.method().to_string();
                if let Ok(res : Option<Response> ) = service.call( req: deserialized).await {
                    res
                } else {
                    return;
                }
            } else {
                return;
            }
        } else {
            return;
        }
    } else {
        return;
    };

    if let Some(res : Response ) = res {
        if let Some(result : &Value ) = res.result() {
            let result : String = result.to_string();
            let mut map : Map<String, Value> = Map::new();
            map.insert( k: String::from( s: "id"), v: Value::Number(res.id().to_string().parse().unwrap()));
            map.insert( k: String::from( s: "jsonrpc"), v: Value::String(String::from( s: "2.0")));
            map.insert( k: String::from( s: "result"), v: from_str( s: &result[..]).unwrap());
            let obj = Value::Object(map);
            if socket.send( msg: Message::Text(obj.to_string())).await.is_err() {
                println!("{}", "ERROR: send failed");
                return;
            }
        }
    }
}
```


POC demo

A configuration screen in a web app

We're developers.

Most of us don't like configurators, wizards, complicated UIs.

We prefer code and the little helpers inside our IDE.

Maybe our customers prefer this too, if we show them.

```
1 {  
2   "type": "SUM_ROW",  
3   "name": "CA",  
4   "title": "C CASH_FLOW_FROM_OPERATIONS user defined variable  
5   "unit": "MO ADDITIONS_TO_CASH  
6   "formula": SUBTRACTIONS_FROM_CASH  
7 }  
8
```

cd ~/PhpstormProjects/lsp-expression-calculator/ && cargo run & cd client && yarn dev && fg

Goals reached?

- ▶ Developer experience in textfield (✓)
- ▶ e2e System setup ✓
- ▶ Showcase for a configurator (-)



Is it useful?

- ▶ Coding with guard rails
- ▶ Need to provide language server
 - ▶ With tower-lsp: write your own (for DSLs)
 - ▶ Possibility to combine with access control or other application state, e.g. some variables only suggested to admins?
- ▶ Every keystroke sends a request to the API

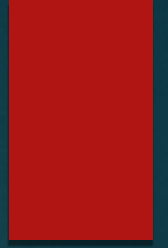


- ▶ Alternative architecture:
 - ▶ Run the language server client side in a web worker

Learnings

- ▶ **Axum framework**
 - ▶ Use the `#[debug_handler]` from `axum_macros`
 - ▶ Otherwise the error messages are not useful
- ▶ **If let** + debug printing is good for gradually unwrapping and debugging values of types you don't know well yet
- ▶ **Axum** never got in my way, but it lacks documentation
- ▶ Struggles:
 - ▶ Protocols (ws, LSP) -> you just need to know them well
- ▶ Initial expectation: I can “**just plug in the tower-lsp middleware**” ❌
 - ▶ Did not achieve this yet. Maybe possible but not obvious.
 - ▶ Instantiating `LspService` instead and calling it from handler
- ▶ The rust type system helps you a lot in “not getting things wrong”. They just won't build

What's next?



- ▶ Direction
 - ▶ Library?
 - ▶ Template App?
 - ▶ Specific Apps
 - ▶ CMS
 - ▶ Coding competitions?
 - ▶ Hackathon