SUSTAINABLE WEB DEVELOPMENT WITH RUST

Miika Alikirri

QUICK COUPLE OF QUESTIONS

- How many of you have heard about rust?
- How many of you have written any rust?
- How many of you are rust experts?
- Are there any non-programers?

TODAY'S TOPICS

1. Define sustainability
2. Short introduction to Rust
3. Sustainable backend
4. Sustainable frontend
5. Environmental impact of code

WHAT IS SUSTAINABILITY

1. Efficient development
2. Efficient maintenance
3. Being environmentally friendly

WHAT'S RUST?

- Relased in 2010 by Mozilla
- Fully open source
- Compiles to machine code
- Backwards compatible
- Focus on safety and correctness
- Loved by developers!

SHORT INTRODUCTION TO RUST

EXAMPLE TYPESCRIPT CODE

```typescript
interface Data {
    foo: number
}

function important_function(data: Data) {
    data.foo += 1;
    console.log("Result", data.foo);
    console.log("Under 44?", data.foo < 44);
}

function main() {
    const api_data = get_data(); // '{"foo": 42}'
    const foo_data = JSON.parse(api_data) as Data;
    important_function(foo_data);
    // a lot of code in between
    important_function(foo_data);
}
```

COMPILING AND RUNNING TYPESCRIPT

Normal input

```
const api_data = "{\"foo\": 42}";
```

Normal output

```
Result 43
Under 44? true
Result 44
Under 44? false
```

Undefined behavior

```
const api_data = "{}";
```

```
Result NaN
Under 44? false
Result NaN
Under 44? false
```

```
console.log(typeof(foo_data.foo)) // number
```

Strings and numbers

```
const api_data = "{\"foo\": \"42\"}";
```

```
Result 142
Under 44? false
Result 1421
Under 44? false
```

```
console.log(typeof(foo_data.foo)) // string
```

WHAT ABOUT RUST?

```rust
struct Data {
    foo: i64
}

fn important_function(data: Data) {
    data.foo += 1;
    println!("Result {}", data.foo);
    println!("Under 44? {}", data.foo < 44);
}

fn main() {
    let api_data = get_data(); // '{"foo": 42}'
    let foo_data = from_str(api_data);
    important_function(foo_data);
    // a lot of code in between
    important_function(foo_data);
}
```

LETS RUN IT!

```
error[E0308]: mismatched types
  --> json/examples/main_js.rs:27:24
   |
27 |     important_function(foo_data);
   |     ------------------ ^^^^^^^^ expected struct `Data`, found enum `Result`
   |     |
   |     arguments to this function are incorrect
   |
   = expected struct `Data`
        found enum `Result<_, serde_json::Error>`
```

It doesn't compile!

```
expected struct `Data`, found enum `Result`
```

RESULT ENUM

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

OPTION ENUM

```
enum Option<T> {
    None,
    Some(T),
}
```

TYPING SAFELY

```
 1 fn main() {
 2   // let foo_data = from_str(api_data); // err
 3   let foo_data: Value = match from_str(api_data) {
 4     Ok(data) => data,
 5     Err(err) => panic!("{:?}", err)
 6   };
 7
 8   let foo_value = match foo_data.get("foo") {
 9     Some(value) => value,
10     None => panic!("JSON data didn't contain foo")
11   };
12
13   let foo_num = match foo_value.as_i64() {
14     Some(num) => num,
15     None => panic!("Foo wasn't i64")
16   };
17
18   let new_data = Data {foo: foo_num};
19   important_function(new_data);
20   // a lot of code in between
21   important_function(new_data);
22 }
```

RESULTS ARE IN

```
error[E0594]: cannot assign to `data.foo`, as `data` is not declared as mutable
  --> json/examples/02_fix_types.rs:18:5
   |
16 | fn important_function(data: Data) {
   |                       ---- help: consider changing this to be mutable: `mut data`
17 |     println!("Result {}", 1 + data.foo);
18 |     data.foo += 1;
   |     ^^^^^^^^^^^^^ cannot assign
```

It still doesn't compile!

FINAL FIXES

```
// fn important_function(data: Data) {/* code */} // err
fn important_function(data: &mut Data) {/* code */}
```

```
fn main() {
  // -- snip --
  // let new_data = Data {foo: foo_num}; // err
  let mut new_data = Data {foo: foo_num};

  important_function(&mut new_data);
  // a lot of code in between
  important_function(&mut new_data);
}
```

IT'S ALIVE!

Normal input

```
let api_data = "{\"foo\": 42}";
```

Normal output

```
Result 43
Under 44? true
Result 44
Under 44? false
```

WHAT ABOUT ERRORS

With no data

```
let api_data = "{}";
```

```
panicked at 'JSON data didn't contain foo', src/main.rs:36:17
```

Strings and numbers

```
let api_data = "{\"foo\": \"42\"}";
```

```
panicked at 'Foo wasn't i64', src/main.rs:43:17
```

Now we actually have run time errors!

QUICK RECAP

- Is the string json?
- Does it contain foo?
- Is it actually a number?

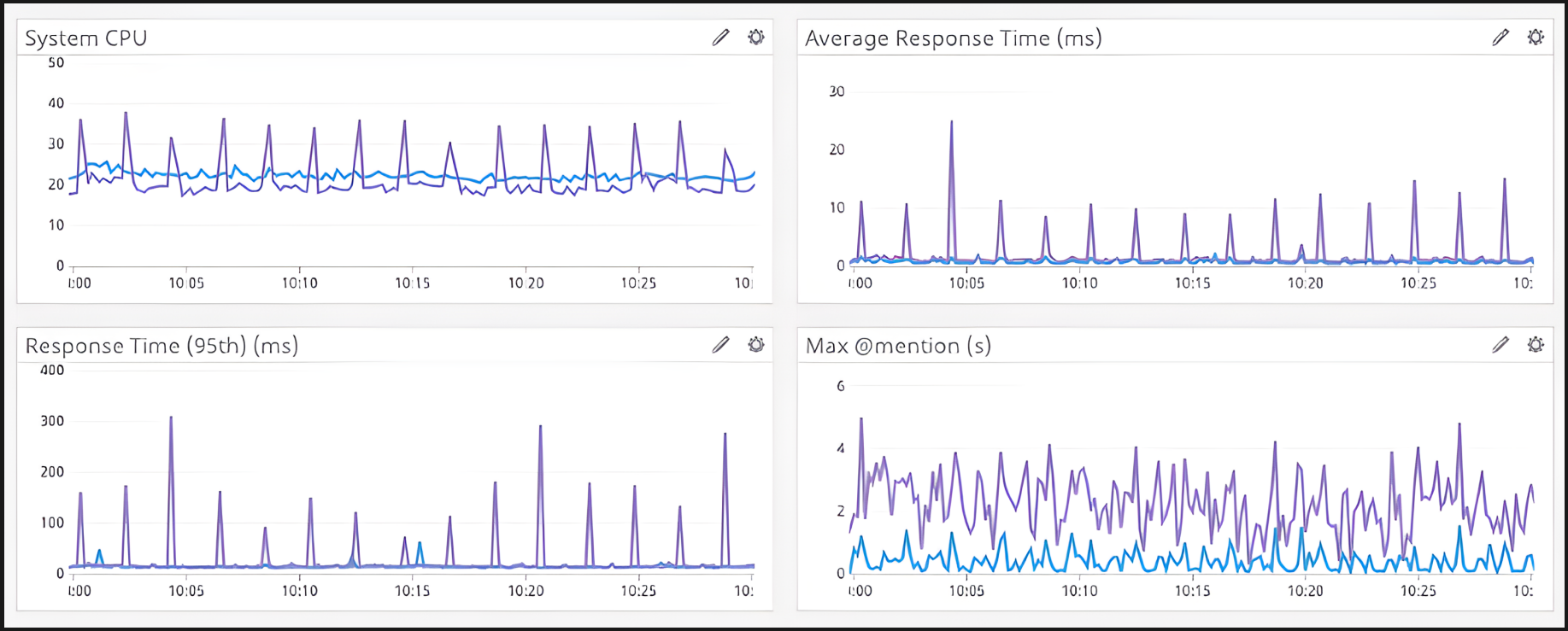
WHERE RUST IS SAFE

RUST IN REAL WORLD

RUST IN BACKEND

USECASE DISCORD

- Service written in Go
- Garbage collector got in the way
- Rewrite in rust
- Instant performance benefits



Blue: Rust, Purple: Go

Src: https://discord.com/blog/why-discord-is-switching-from-go-to-rust

RUST AND AWS

AWS LAMDA

```
# Rust
Init Duration: 33.60 ms
Billed Duration: 393 ms Max Memory Used: 31 MB
Billed Duration: 51 ms  Max Memory Used: 31 MB
```

```
# Node.js
Init Duration: 236.67 ms
Billed Duration: 916 ms Max Memory Used: 81 MB
Billed Duration: 268 ms Max Memory Used: 81 MB
```

@aviillouz

SAFETY FIRST!

SQL QUERYS

```rust
struct Duck {
  quacks: bool
}

fn get_duck(id: i64) -> Result<Duck> {
  return sqlx::query_as!(
    Duck,
    "SELECT * FROM ducks WHERE id = ?", id
  ).fetch_one();
}
```

- Is id in ducks table?
- Is it valid type?
- Can we create Duck from ducks table
- All of this during compile time

RUST IN FRONTEND

YEW.RS

```rust
#[function_component(HelloComponent)]
fn Hello(props: &Props) -> Html {
  html! {
    <p>{ &props.text }</p>
  }
}
```

WEBGL

ENVIRONMENTAL IMPACT OF RUST

Total

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97

Energy (J)	value	Time (ms)	value
C	1.00	C	1.00
Rust	1.03	Rust	1.04
Java	1.98	Java	1.89
C#	3.14	C#	3.14
Go	3.23	Go	2.83
JavaScript	4.45	JavaScript	6.52
PHP	29.30	PHP	27.64
Python	75.88	Python	71.90

THE END