Rust Web Frameworks

And what we've learnt

Rust Web Frameworks (RWFs)

- WASM (Optional)
- Rust front end development

Types of frameworks

- ‘Game engine’ type
- ‘normal’ type

‘game engine RWF’: EGUI

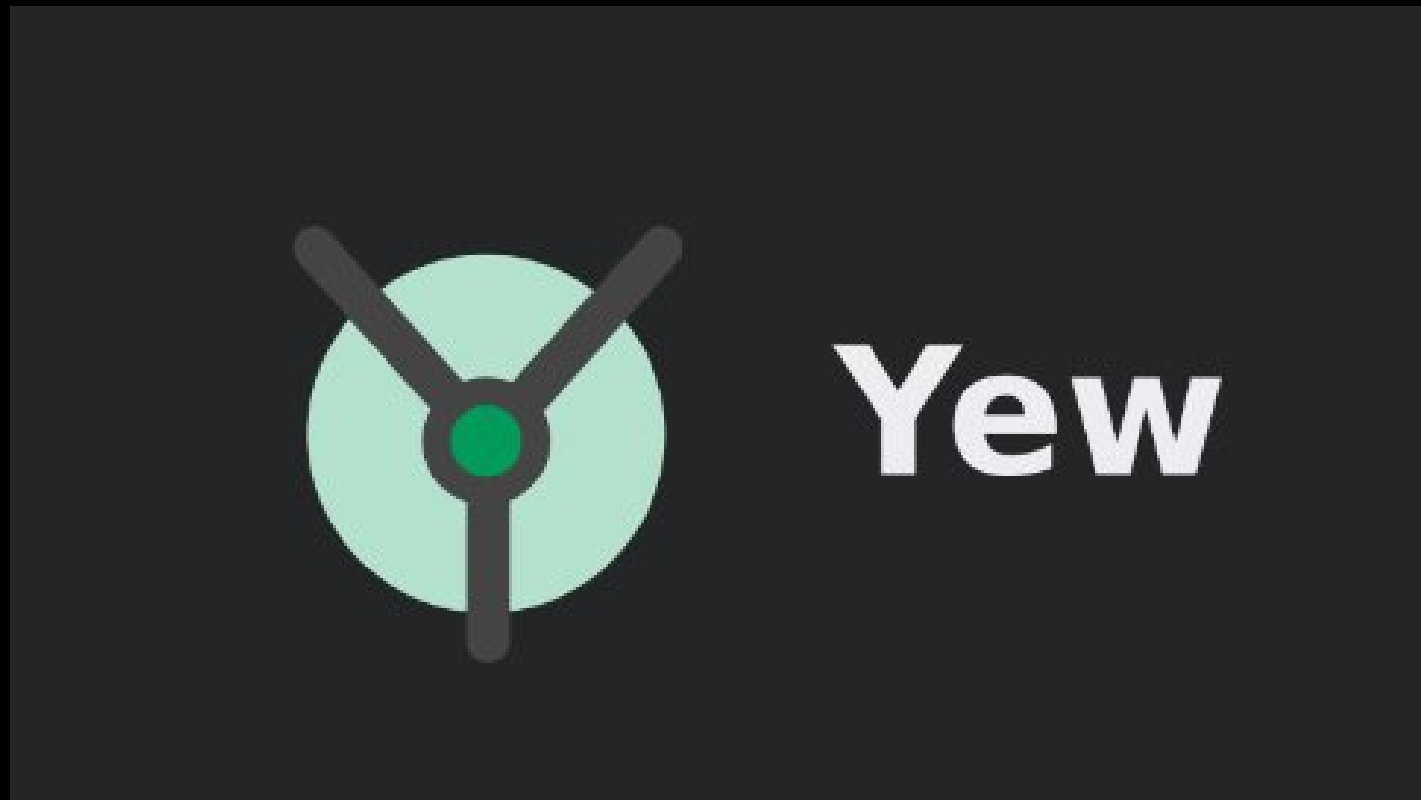
- Rendering: Canvas (WebGL or WGPU)
- Design: widgets
- Mode: Immediate mode GUI

Example

```
ui.heading("My egui Application");
ui.horizontal(|ui| {
    ui.label("Your name: ");
    ui.text_edit_singleline(&mut name);
});
ui.add(egui::Slider::new(&mut age, 0..=120).text("age"));
if ui.button("Increment").clicked() {
    age += 1;
}
ui.label(format!("Hello '{name}', age {age}"));
ui.image(egui::include_image!("ferris.png"));
```

‘'Normal RWF’: Yew.rs

- Rendering: HTML
- Design: HTML Macro
- Mode: retained mode GUI



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

#[function_component]
fn App() -> Html {
    let counter = use_state(|| 0);
    let onclick = {
        let counter = counter.clone();
        move |_| {
            let value = *counter + 1;
            counter.set(value);
        }
    };

    html! {
        <div>
            <button {onclick}>{ "+1" }</button>
            <p>{ *counter }</p>
        </div>
    }
}
```

RWFS:

- What issues do they face?
- How did Dioxus fix them?

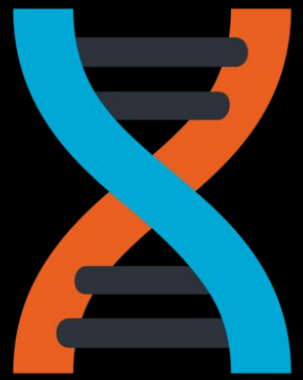
One codebase, every platform.

Dioxus is *the* Rust framework for building fullstack web, desktop, and mobile apps. Iterate with live hotreloading, add server functions, and deploy in record time.

Macros...

- useless errors
- cargo fmt
- Html { } not <HTML/>

```
html! {
    <main>
        <h1> {"Hello!"} </h1>
        <div>
            <p> {"lorem ipsum or something"}  </p>
        </div>
    </main>
}
```



```
rsx! {
    div {
        h1 { "Hi!" }
        div {
            p { "lorem ipsum or something" }
        }
    }
}
```

```
error: this closing tag has no corresponding opening tag
  --> src/app.rs:10:12
   |
10 |             </div>
   |             ^^^^^^

error: could not compile `yew-rust-conf` (bin "yew-rust-conf") due to 1 previous error
2025-09-29T11:44:07.3245477Z ERROR ❌ error
error from build pipeline
```

```
    html! {
        <main>
            <h1> {"Hello!"} </h1>
                <div>
                    <p> {"lorem ipsum or something"} // </p>
                </div>
        </main>
    }
```

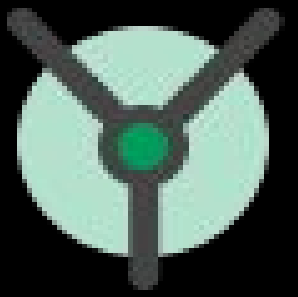
```
  --> src/main.rs:97:3
   |
55 | pub fn Hello() -> Element {
   |                           - unclosed delimiter
...
60 |                 p { "lorem ipsum or something" // }
   |                   - this delimiter might not be properly closed...
61 |             }
   |             - ...as it matches this but it has different indentation
...
97 | }
   |  ^
```

```
rsx! {
    div {
        h1 { "Hi!" }
        div {
            p { "lorem ipsum or something" // }
        }
    }
}
```

Cargo fmt is the best thing ever

```
cargo fmt
```

:/



```
    html! {
    <main>
                <h1> {"Hello!"} </h1>

                                <div>
<p> {"lorem ipsum or something"}  </p>
            </div>
        </main>
    }
```

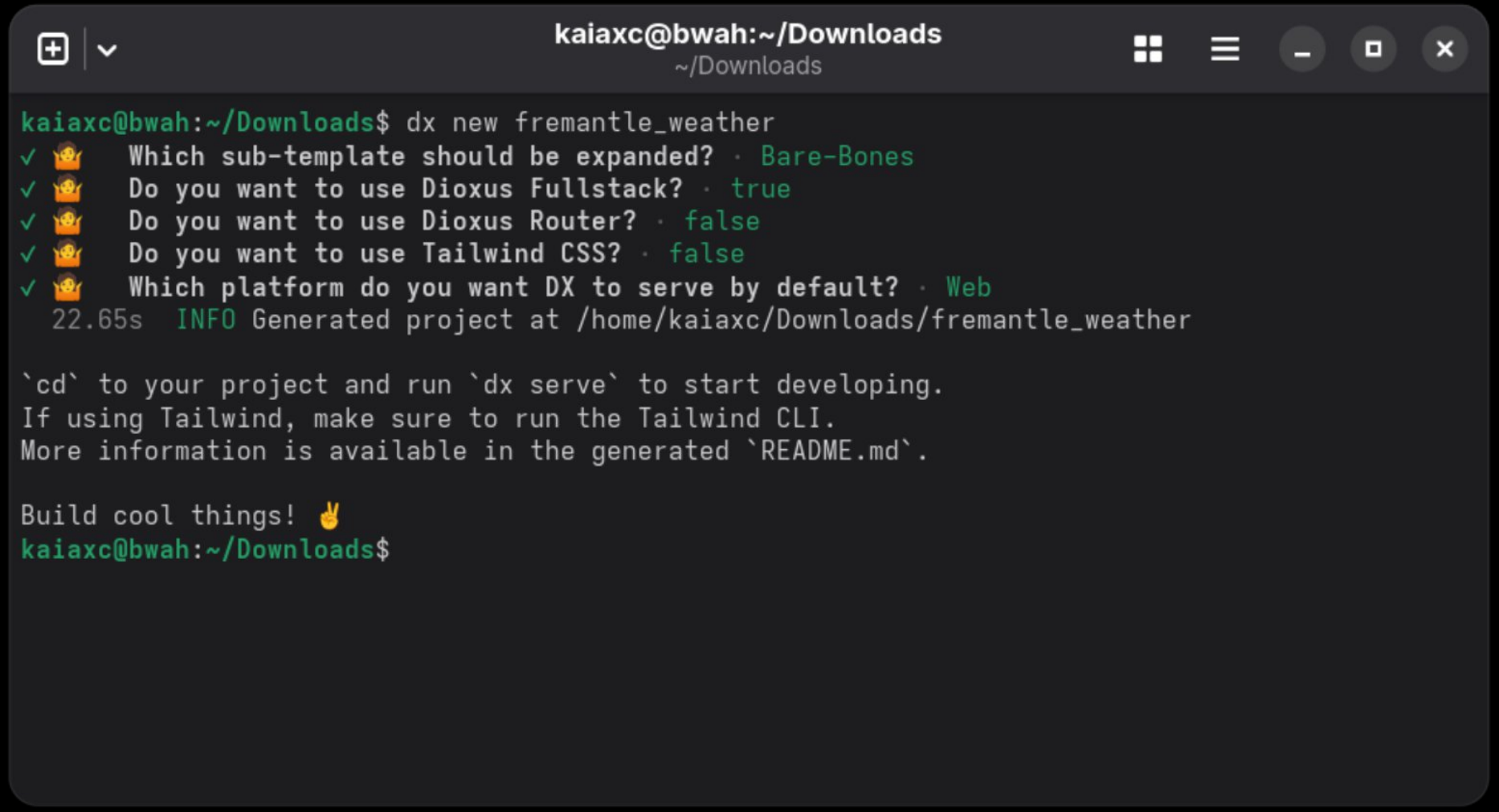
Dioxus: dx fmt

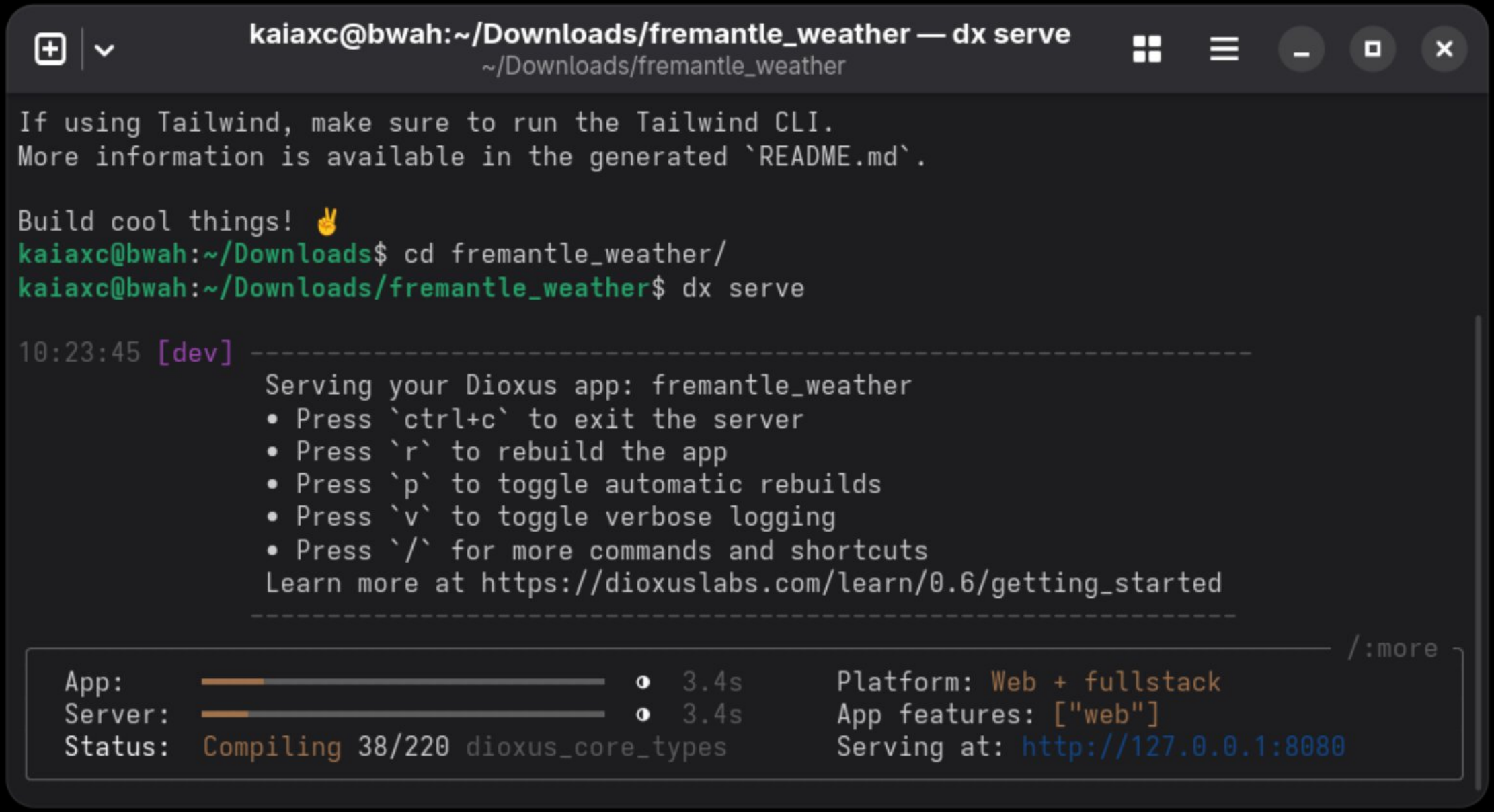
```
Automatically format RSX

Usage: dx fmt [OPTIONS]

Options:
      --all-code                  Format rust code before the formatting the rsx macros
  -c, --check                     Run in 'check' mode. Exits with 0 if input is formatted
                                  correctly. Exits with 1 and prints a diff if formatting is
                                  required
  -r, --raw <RAW>                 Input rsx (selection)
  -f, --file <FILE>               Input file
  -s, --split-line-attributes     Split attributes in lines or not
  -p, --package <PACKAGE>         The package to build
      --verbose                   Use verbose output [default: false]
      --trace                     Use trace output [default: false]
      --json-output               Output logs in JSON format
```

This section *was* much longer

Lets make an app!

Dioxus templating




kaiaxc@bwah:~/Downloads/fremantle_weather — dx serve
~/Downloads/fremantle_weather
If using Tailwind, make sure to run the Tailwind CLI.
More information is available in the generated `README.md`.
Build cool things! ✌️
kaiaxc@bwah:~/Downloads$ cd fremantle_weather/
kaiaxc@bwah:~/Downloads/fremantle_weather$ dx serve
10:23:45 [dev]
Serving your Dioxus app: fremantle_weather
• Press `ctrl+c` to exit the server
• Press `r` to rebuild the app
• Press `p` to toggle automatic rebuilds
• Press `v` to toggle verbose logging
• Press `/` for more commands and shortcuts
Learn more at https://dioxuslabs.com/learn/0.6/getting_started
/:more
App:
3.4s
Server:
3.4s
Status: Compiling 38/220 dioxus_core_types
Platform: Web + fullstack
App features: ["web"]
Serving at: http://127.0.0.1:8080

Weather api (too hard)

Australian Government

Bureau of Meteorology

Space Weather API

API specification

Space Weather API

API specification

Overview

This API provides access to near real-time Australian Space Weather Forecasting Centre data from the Australian Bureau of Meteorol

Use of the API requires registration. Upon successful registration, an API key will be provided. This key needs to be included in reques

The API request methods are as follows.

get-a-index	Requests the most recent A index for the Australian region, or historical values.
get-k-index	Requests the most recent K index from a specified location (Australian region or several observing sites),
get-dst-index	Requests the most recent Dst index for the Australian region, or historical values.
get-mag-alert	Requests details of any magnetic alert current for the Australian region.
get-mag-warning	Requests details of any geophysical warning currently active for the Australian region.
get-aurora-alert	Requests details of any aurora alert current for the Australian region.
get-aurora-watch	Requests details of any aurora watch current for the Australian region.
get-aurora-outlook	Requests details of any aurora outlook current for the Australian region.

Each API method URL includes the prefix:

```
https://sws-data.sws.bom.gov.au/api/v1/
```

Data can be obtained using an HTTP POST request to the appropriate URL. An API key and any relevant options must be included in
response use JSON. The character encoding is UTF-8.

backend

```rust
#[server]
0 implementations
pub async fn weather_get() -> Result<Root, ServerFnError> {
    let url = format!("https://api.openuv.io/api/v1/uv?lat=-32.056946&lng=115.743889");

    let client = reqwest::Client::new();
    let response = client
        .get(url)
        .header("x-access-token", TOKEN.to_string())
        .send()
        .await
        .unwrap()
        .json()
        .await
        .unwrap();
    Ok(response)
}
```

Lazy statics are cool

```
#[cfg(feature = "server")]
use lazy_static::lazy_static;
#[cfg(feature = "server")]
lazy_static! {
    static ref TOKEN: String = String::from_utf8(std::fs::read("token.txt").unwrap()).unwrap();
}
```

Make request to backend on startup

```
let Some(weather: Root) = use_resource(move || async move { weather_get().await.unwrap() })
    .read_unchecked()
    .clone()
else {
    return rsx! {
        h1 { "Server Error :/" }
    };
};
```

Error handling

```
#[component]
fn example() -> Element {
    rsx! {
        p { "hello world!" }
    }
}
```

Error condition

```rust
#[component]
fn example() -> Element {
    let result: Result<(), io::Error> = Result::Err(io::Error::other("oops!"));

    if result.is_err() {
        return rsx! {
            p { "oh no!!" }
        };
    }

    rsx! {
        p { "normal webpage (success)" }
    }
}
```

CSS is dead simple

```rust
const MAIN_CSS: Asset = asset!("/assets/ebook.css");
#[component]
fn App() -> Element {
    rsx! {
        document::Link { rel: "stylesheet", href: MAIN_CSS }
        Echo {}
    }
}
```

Buttons! :3

```rust
#[component]
0 implementations
fn button(is_loading: Signal<bool>, weather: Signal<Root>) -> Element {
    rsx! {

        div { class: "boxes",
            a {
                class: "box bx-line",
                onclick: move |_| async move {
                    is_loading.set(true);
                    let uv = weather_get().await.unwrap();
                    weather.set(uv);
                    is_loading.set(false);
                },
                "Refresh"
            }
        }
    }
}
```

conditional HTML

```
if !*is_loading.read() {

    h2 { class: "box bx-line", "Max: {max}" }
    h2 { class: "box bx-line", "Current: {current}" }

} else {
    h2 { class: "box bx-line", ". . . " }
    h2 { class: "box bx-line", ". . ." }
}
```

Dots dots indicate loading

fin

UV site can be found here:

- https://github.com/toastxc/fremantle_weather