DME

A delightful Markdown experience ?

That’s possible !

Markdown experience outside editing

Current experience

- Preview not always pleasant
- Code highlighting too much basic
- PDF export very hard and broken
- Single file preview

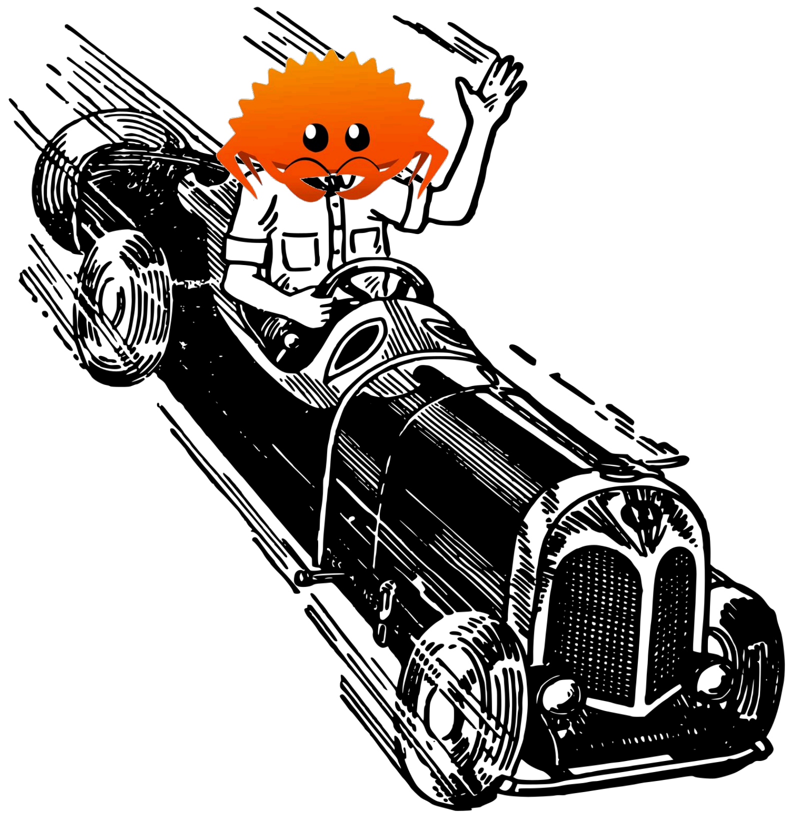
Dream experience

- Jumping easily through any Markdown file on disk
- Full text search on Markdown content
- Fast preview load and refresh, even for very big documents
- Full code highlighting with Tree-Sitter
- Easy export in PDF

Challenges

Speed

- Markdown files research
- Markdown content indexing
- Full text search
- HTML & Code preview generation
- PDF generation



Ownership and lifetimes

Applied to concurrent programming

Memory safety in concurrency

C++ vs Rust

```
void task(int *counter) {
    while (*counter < 10000000)
        (*counter)++;
}

int main(void) {
    int counter = 0;
    PcoThread *threads[30];
    for (int i = 0; i < 30; i++)
       threads[i] = new PcoThread(task,
&counter);
    // [...] joining threads
    cout << "counter " << counter << endl;
}
```

```
fn task(counter: &mut u32) {
    while *counter < 10000000 { *counter +=
1; }
}

fn main() {
    let mut counter = 0;
    let mut handles = Vec::new();
    for _ in 1..30 {
        handles.push(
            thread::spawn(|| task(&mut
counter))
        );
    }
    // [...] joining threads
    println!("counter {counter}");
}
```

Memory safety in concurrency

Results

```
> # BUILD OK
> # running once
counter 10000000

> # running 10 times
Results
 6 times: counter 10000000
 2 times: counter 10000001
 2 times: counter 10000002
```

`error:` **closure may outlive the current function, but it borrows `counter`, which is owned by the current function**

```
handles.push(thread::spawn(|| task(&mut counter)));
        `counter` is borrowed here
        may outlive borrowed value `counter`
```

`error:` **cannot borrow `counter` as mutable more than once at a time**

```
handles.push(thread::spawn(|| task(&mut counter)));
            `counter` was mutably borrowed here in the previous
            iteration of the loop
```

`error:` **cannot borrow `counter` as immutable because it is also borrowed as mutable**

```
handles.push(thread::spawn(|| task(&mut counter)));
                              mutable borrow occurs here
println!("Counter {counter}");
                  ^^^^^^^^^ immutable borrow occurs here
```

Why Rust ?

Memory safety and speed

- Concurrent access checked at compile time
- Strong typing system, smart types like `Mutex`
- No garbage collector and no manual memory management