Day 08

Blitzen's Debug Blitz

The Story

The North Pole's DevOps room was unusually quiet—well, as quiet as it could be with Blitzen strutting around like he owned the place. The reindeer had declared himself Tech Lead™ (again), and with Santa still off on his "vision quest", nobody had dared challenge him. Every five minutes, Blitzen would casually remind the room, "I'm the Tech Lead, in case anyone forgot."

The Story

The elves, however, had not forgotten. Frostbyte, the team's fastest typist, was already regretting showing up for work today.

The Discovery of Errors

It all started when Blitzen stumbled upon a string of unusual log entries:

```
ERROR: Toy Tracker 3000 overheating.
ERROR: SleighOS failed to authenticate.
ERROR: Reindeer AI unable to locate Prancer.
```

He squinted at the terminal, his antlers practically buzzing with excitement. "This... this is big," Blitzen declared dramatically. "These errors need to be isolated, analyzed, and stored in their own file. This could save Christmas!"

The Discovery of Errors

Prancer, barely looking up from their desk, muttered, "Couldn’t we just pipe the logs through `grep ERROR` and append it to a file with `>> file.log`?"

Blitzen whipped around, scandalized. "Prancer, what part of 'I'm the Tech Lead' didn't you understand? We don't use grep here. We build solutions. Innovative solutions. In Rust!"

The Discovery of Errors

"Here's the plan," Blitzen said, pacing like a founder pitching to VCs. "We'll extend our `LogQuery` tool to not just search logs, but also export specific entries to a file. A Rustacean-grade solution, not some bash script hack job."

An elf raised their hand timidly. "But why?"

Blitzen grinned. "Because I'm the Tech Lead."

Your Mission

You must come to the elves rescue! Implement the `export_to_file(&self, keyword: &str, file_path: &str)` method in `LogQuery` that:

1. Use the `search` method we created earlier to get logs matching the `keyword`.
2. Loop over the returned logs and write them to a file, the location is given by the parameter `path`.
3. **Handles Errors:** If the file can't be created or written to, the function should return an appropriate error instead of crashing.

Hints

If you're stuck or need a starting point, here are some hints to help you along the way!

- Make sure you import the necessary modules. e.g., `use std::{fs::File, io::Write};` .
- Get the logs using the `search` method we created in the previous challenge. e.g. `let logs = self.search(keyword);` .
- Create a mutable file using `File::create` . e.g. `let mut file = File::create(path)?;`

Hints

- Properly handle errors with `Result` and `?` to propagate them.
- Loop over the logs and use the `writeln!` macro to write to the file. e.g.

```
for log in logs {
    writeln!(file, "{}", log)?;
}
```

- Return `Ok(())` if everything goes well.