Zellij - Developing WASM Rust Plugins

https://zellij.dev

by Aram Drevekenin (@imsnif)

The Future is Terminal

> Develop in the terminal? Use the terminal for development? You are a terminal developer

> Terminals are everywhere

> The terminal is a long-standing robust and underutilized platform, let's do amazing things with it

Terminal Developer

Terminals are for Humans

> Stop catering to "l33t h4x0rs"

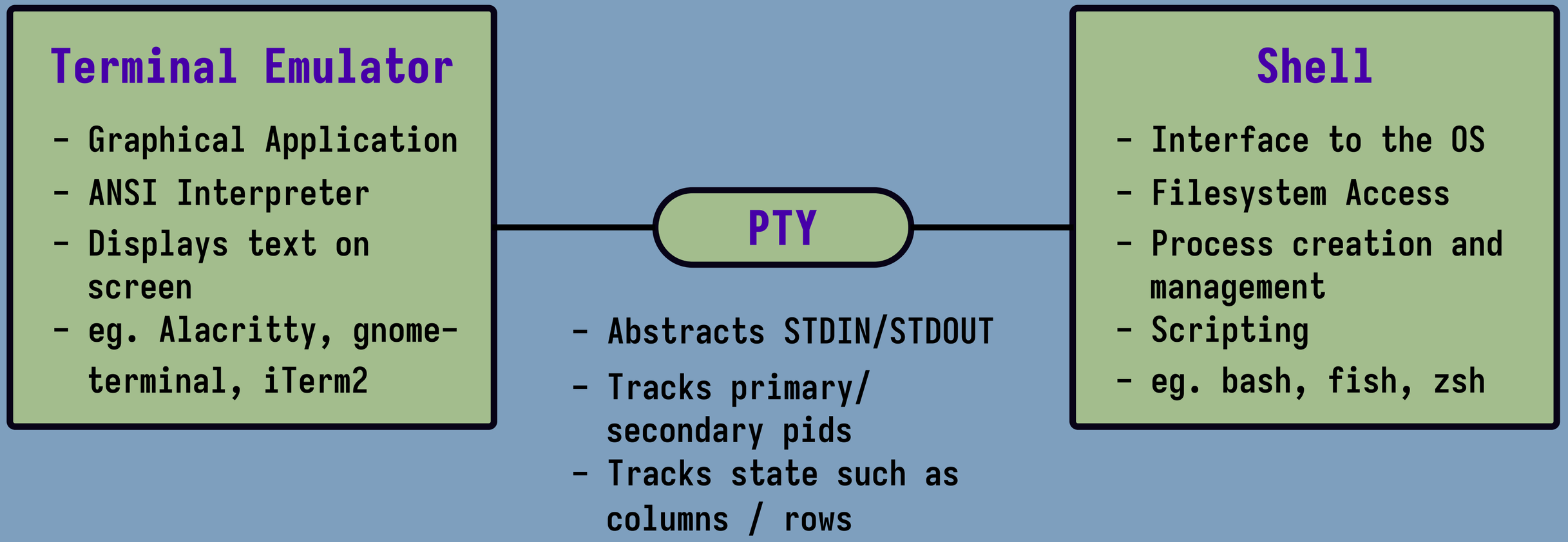
> Give equal weight to product and tech

> Terminals as a Development Environment - a user friendly dashboard

Terminal Crash Course

(Simplified...)

Our Ecosystem

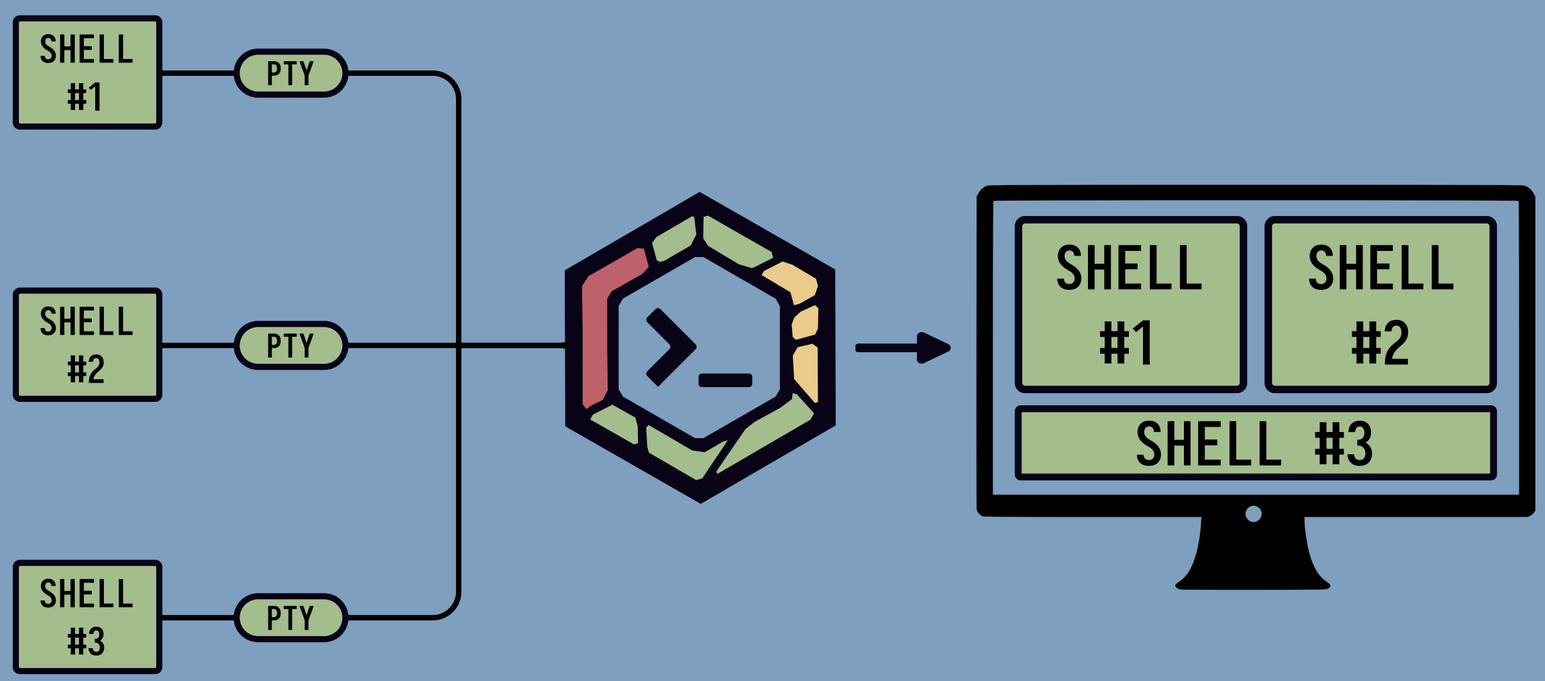


ANSI Escape Codes

Demo

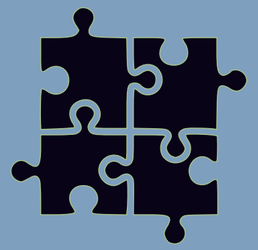
What's a terminal multiplexer?

- Manages pty connections
- Keeps session alive
- Arranges panes on screen
- Tracks their viewport

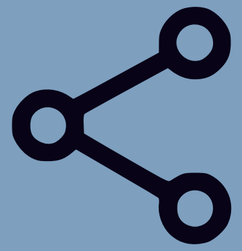


Zellij - a Terminal Workspace

> First class command, edit and plugin panes



> Plugins in any language that compiles
> to wasm/wasi



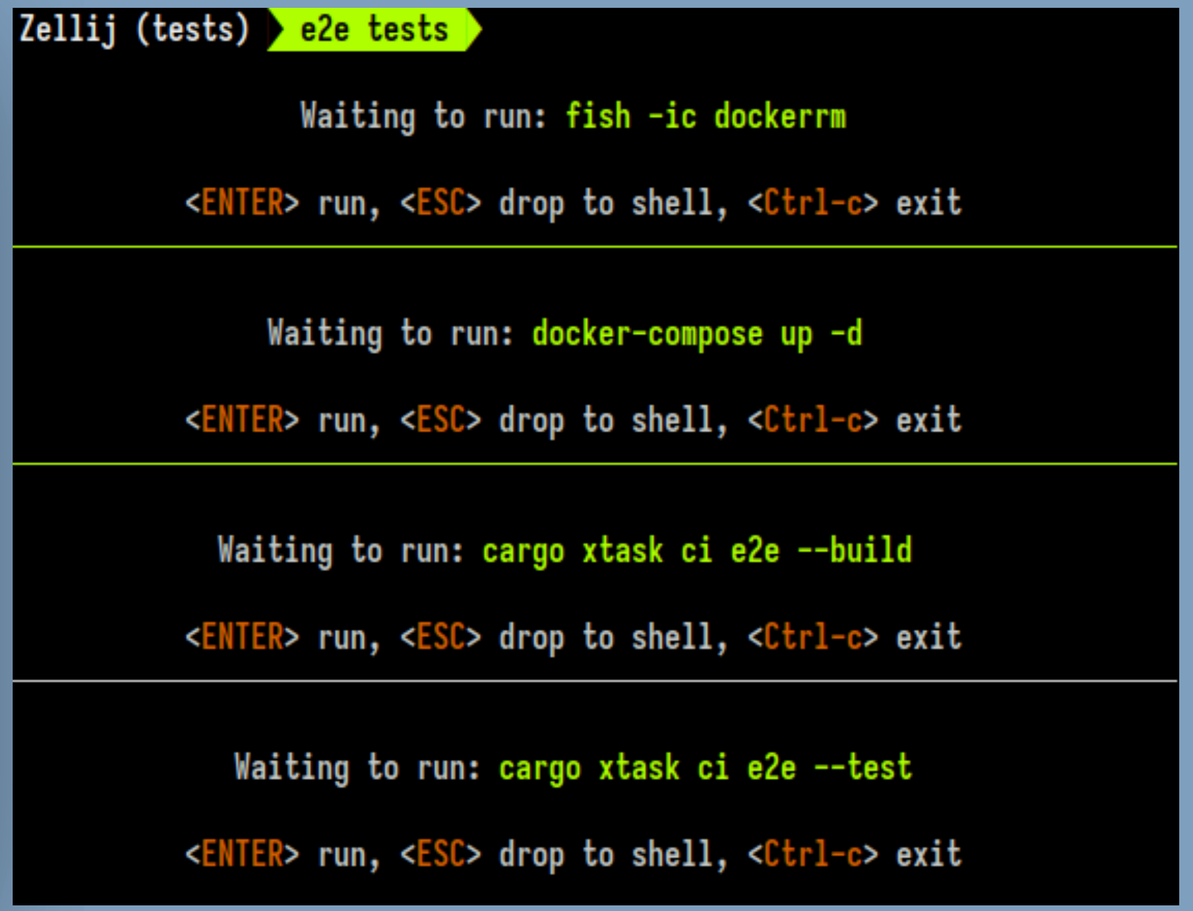
> Serializable, resurrectable,
> shareable sessions

> True Multiplayer collaboration experience

Zellij Layouts

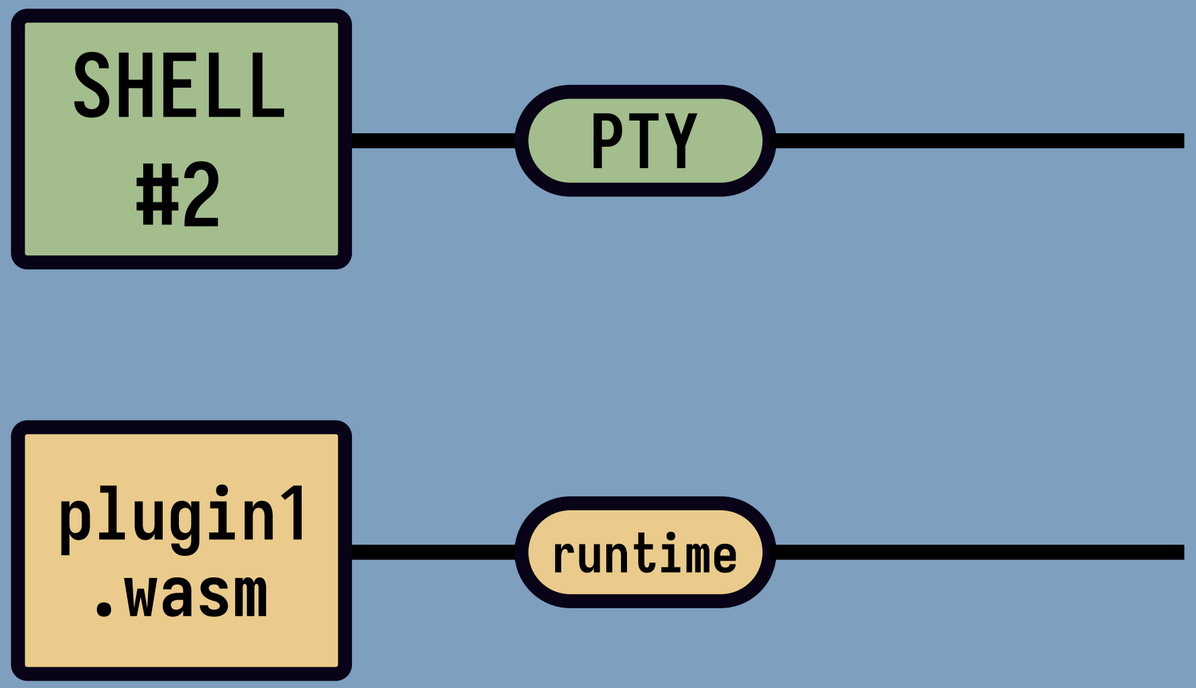
```
// e2e-tests.kdl
layout {
    pane name="Reset" start_suspended=true {
        command "fish"
        args "-ic" "dockerrm"
    }
    pane name="Start Environment" start_suspended=true {
        command "docker-compose"
        args "up" "-d"
    }
    pane name="Build E2E Executable" start_suspended=true {
        command "cargo"
        args "xtask" "ci" "e2e" "--build"
    }
    pane name="E2E Tests" start_suspended=true {
        command "cargo"
        args "xtask" "ci" "e2e" "--test"
    }
}
```

KDL
https://kdl.dev



WASM/WASI plugin system

- Easier distribution
- Sandboxing
- Easier development
  - *Composability*
  - *UI components*
  - *Output buffer*
- Stronger capabilities
  - *Affect Zellij*
  - *Know whole application state*
  - *Permission system for sensitive access*



Developing (Rust) plugins

> Lifecycle Methods

> Subscribe to Events

> Run plugin commands

src/main.rs

```rust
#[derive(Default)]
struct State {}

register_plugin!(State)

impl ZellijPlugin for State {
    fn load(Configuration)
    fn update(Event)
    fn render(rows, cols)
}
```

How does it work?

Zellij

STDIN / STDOUT

App state through protobuffers

```
message TabInfo {
    uint32 position = 1;
    string name = 2;
    bool active = 3;
    // ...
}
```

Rendering with ANSI

^[[36mHi there!\r\n

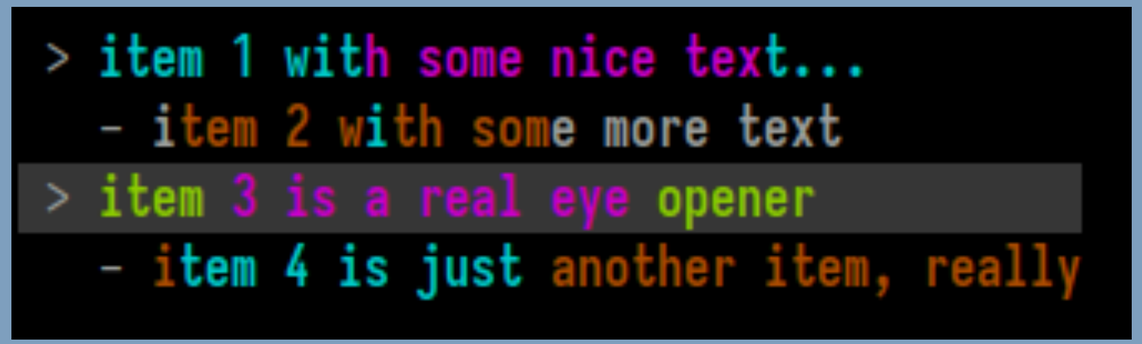
plugin1.wasm

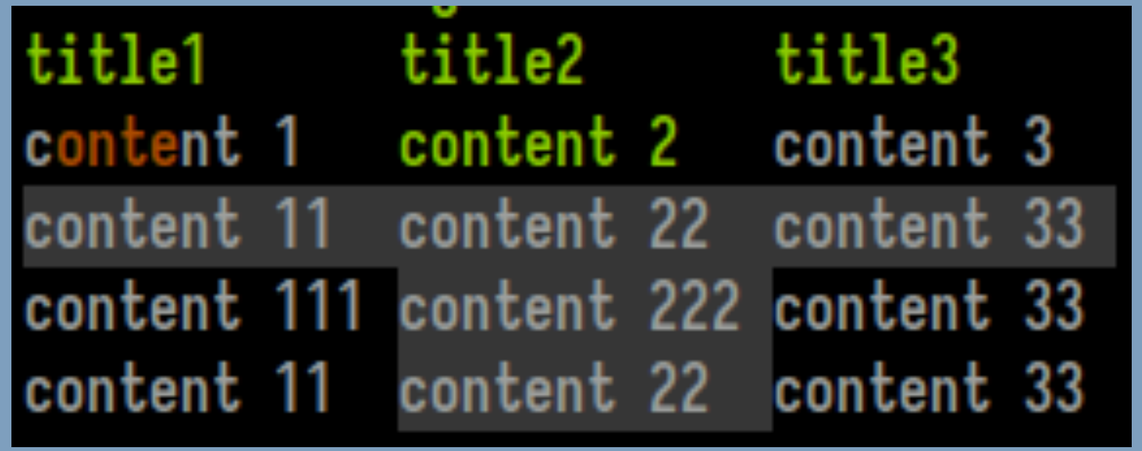
UI Elements for plugins

> Cross language

> Easy to get off the ground

> Allow for a consistent experience across the app (eg. fits the user's theme)





UI Element Example (Rust SDK)

```rust
let table = Table::new()
    .add_row(vec!["title1", "title2", "title3"])
    .add_styled_row(vec![
        Text::new("content 1").color_range(0, 1..5),
        Text::new("content 2").color_range(2, ..),
        Text::new("content 3")
    ])
    .add_styled_row(vec![
        Text::new("content 11").selected(),
        Text::new("content 22").selected(),
        Text::new("content 33").selected()
    ])
    .add_styled_row(vec![
        Text::new("content 111"),
        Text::new("content 222").selected(),
        Text::new("content 33")
    ])
    .add_styled_row(vec![
        Text::new("content 11"),
        Text::new("content 22").selected(),
        Text::new("content 33")
    ]);
print_table(table);
```

title1	title2	title3
content 1	content 2	content 3
content 11	content 22	content 33
content 111	content 222	content 33
content 11	content 22	content 33

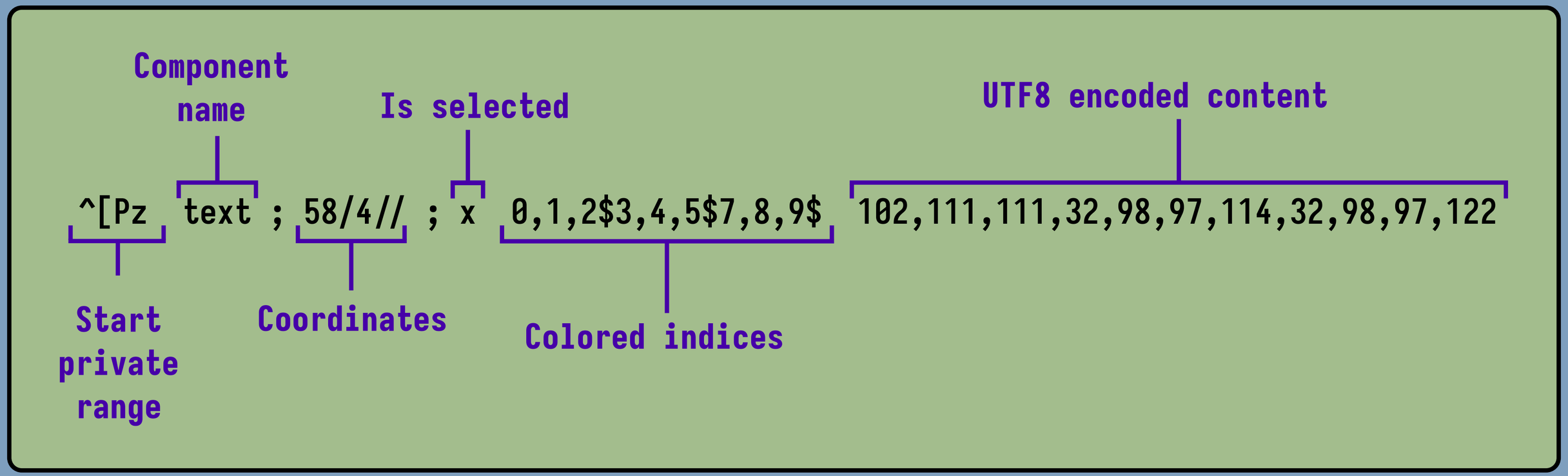
^[Pztable5;7;116,105,116,108,101,49;116,105,116,108,101,50;116,105,116,108,101,51;116,105,116,108,101,52;116,105,116,108,101,53;49,108,107,115,100,106,102,108,107,115,100,106,102;108,107,106,100,115,102,108,107,115,106,100,102;108,107,106,115,100,108,107,102,106,115,102,100;108,107,106,115,100,108,107,106...

How do UI Elements work?

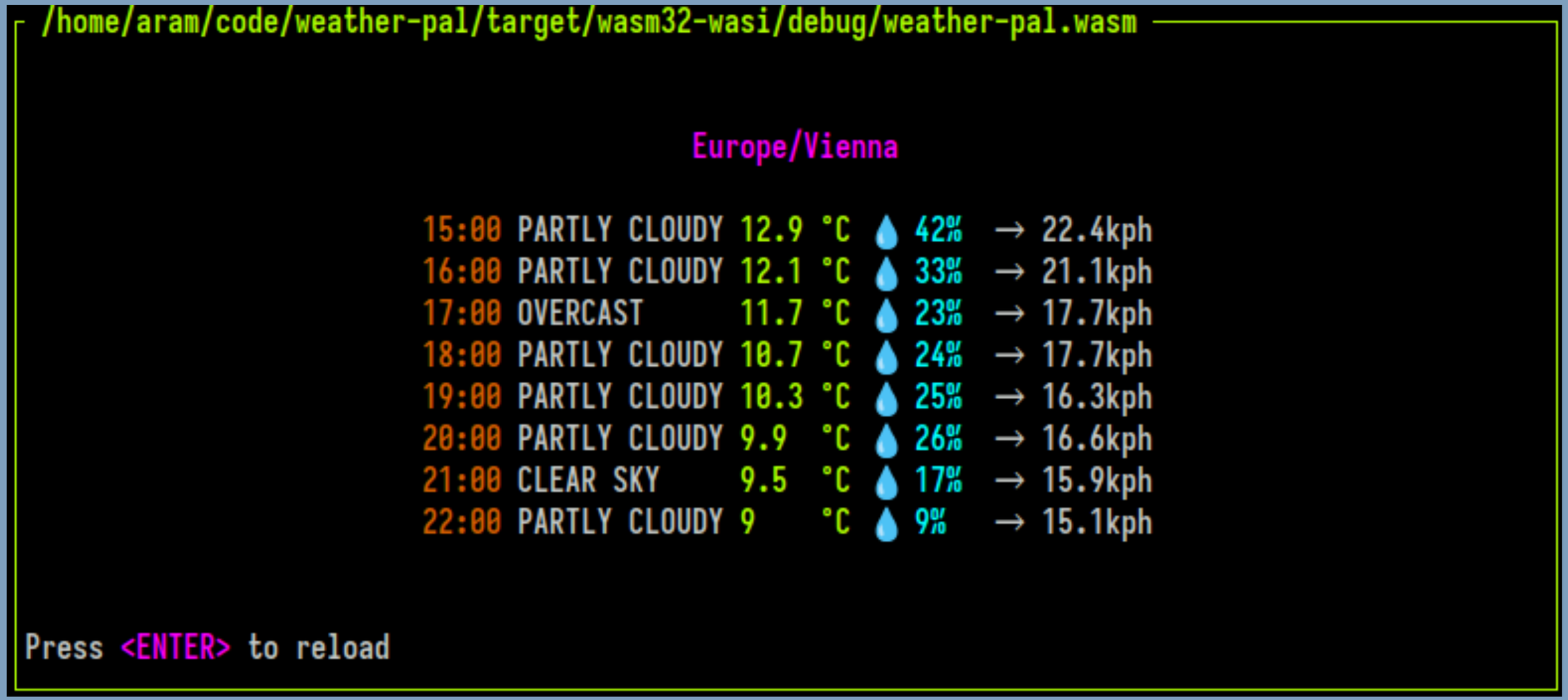
> Interpreted as a private ANSI instruction

> Pre-processed by the Zellij ANSI interpreter

foo bar baz



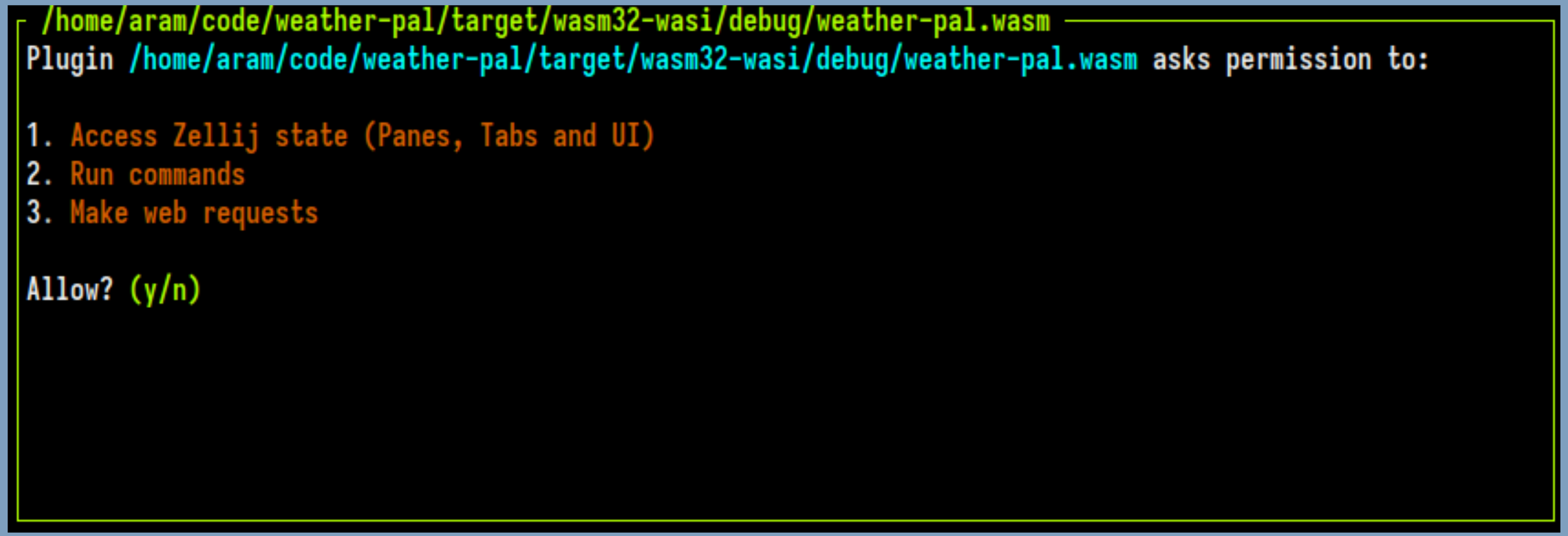
Plugin Demo: WeatherPal



Plugin Demo: WeatherPal

```rust
fn load(&mut self, _configuration: BTreeMap<String, String>) {
    request_permission(&[
        PermissionType::ReadApplicationState,
        PermissionType::RunCommands,
        PermissionType::WebAccess
    ]);
    subscribe(&[
        EventType::Key,
        EventType::WebRequestResult,
        EventType::RunCommandResult
    ]);
}
```

Plugin Demo: WeatherPal



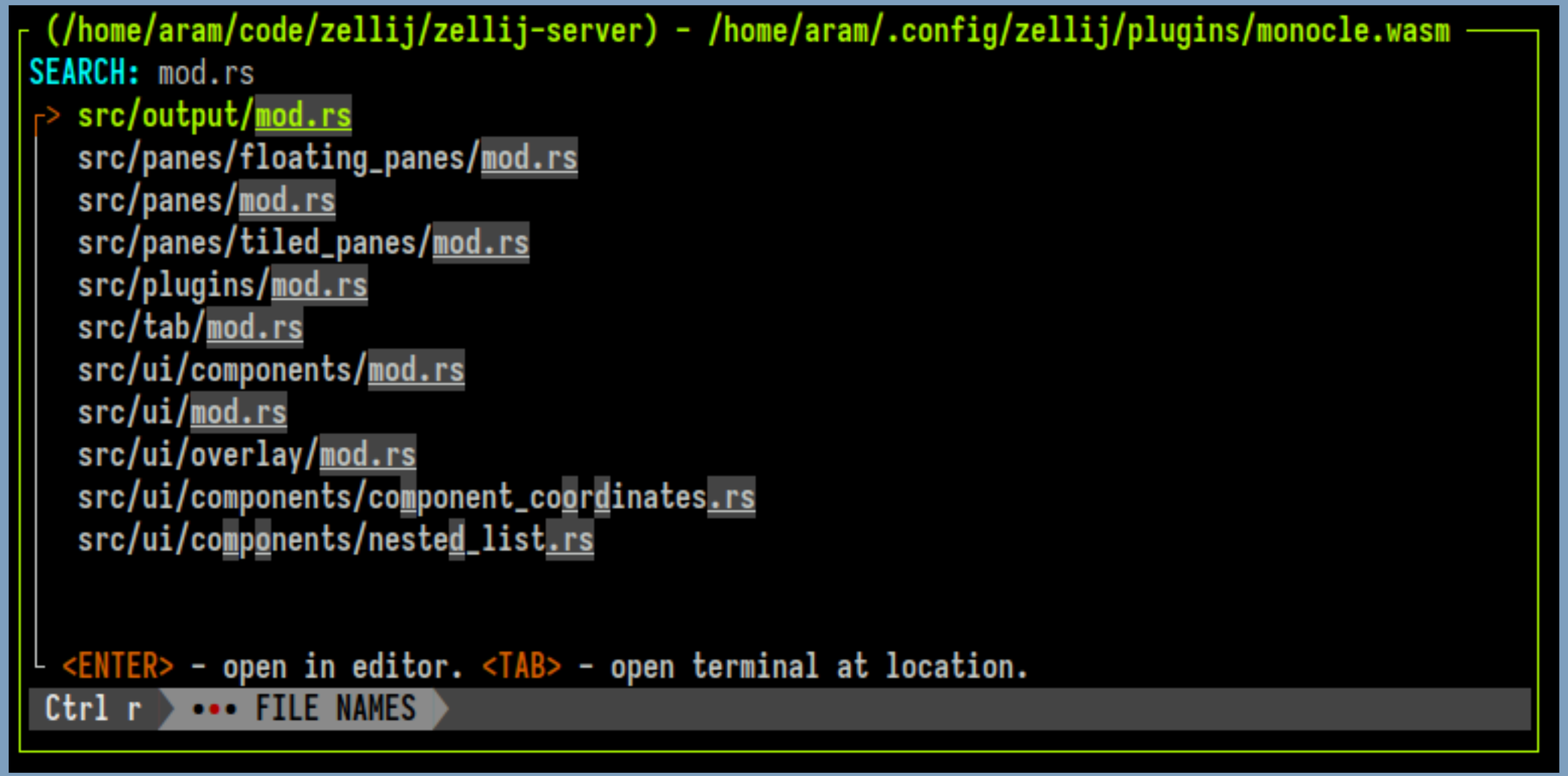
Plugin Demo: WeatherPal

```rust
fn update(&mut self, event: Event) -> bool {
    // ...
    let mut should_render = false;
    match event {
        // ..
        Event::Key(key) => {
            if let Key::Char('\n') = key {
                self.fetching_data = true;
                let get_timezone = "timedatectl | grep \"Time zone\" | awk \'{print $3}\'";
                let mut c = BTreeMap::new();
                c.insert("id".to_owned(), "TIMEZONE_COMMAND_ID".to_owned());
                run_command(&vec!["bash", "-c", get_timezone], c);
                should_render = true;
            }
        }
        Event::RunCommandResult(exit_code, stdout, stderr, context) => {
            // ..
            self.timezone = String::from_utf8(stdout).ok().map(|s| s.trim().to_owned());
            make_geocode_request(&self.timezone);
        }
    }
    should_render
}
```

Plugin Demo: WeatherPal

```rust
fn render(&mut self, rows: usize, cols: usize) {
    // ...
    let mut weather_table = Table::new();
    for hourly_data in &self.weather_data {
        weather_table = weather_table.add_styled_row(vec![
            Text::new(hourly_data.hour_text).color_range(0, ..),
            Text::new(hourly_data.weather_description),
            Text::new(hourly_data.degrees_text).color_range(2, ..),
            Text::new(hourly_data.degrees_symbol).color_range(2, ..),
            Text::new(hourly_data.precipitation_text).color_range(1, ..),
            Text::new(hourly_data.wind_direction_text),
        ]);
    }
    print_table(weather_table);
}
```

Plugin Demo: Monocle



Plugin Demo: Monocle

```rust
fn load(&mut self, config: BTreeMap<String, String>) {
    // ..
    post_message_to(PluginMessage::new_to_worker(
        "file_name_search",
        &serde_json::to_string(&MessageToSearch::ScanFolder).unwrap(),
        "",
    ));
    post_message_to(PluginMessage::new_to_worker(
        "file_contents_search",
        &serde_json::to_string(&MessageToSearch::ScanFolder).unwrap(),
        "",
    ));
    self.loading = true;
    set_timeout(0.5); // for displaying loading animation
}
```

Plugin Demo: Monocle

```
impl<'de> ZellijWorker<'de> for FileNameWorker {
    fn on_message(&mut self, message: String, payload: String) {
        match serde_json::from_str::<MessageToSearch>(&message) {
            Ok(MessageToSearch::ScanFolder) => {
                self.scan_hd(); // blocking long call
                post_message_to_plugin(PluginMessage::new_to_plugin(
                    &serde_json::to_string(&MessageToPlugin::DoneScanningFolder).unwrap(),
                    "",
                ));
            }
            // ...
        }
    }
}
```

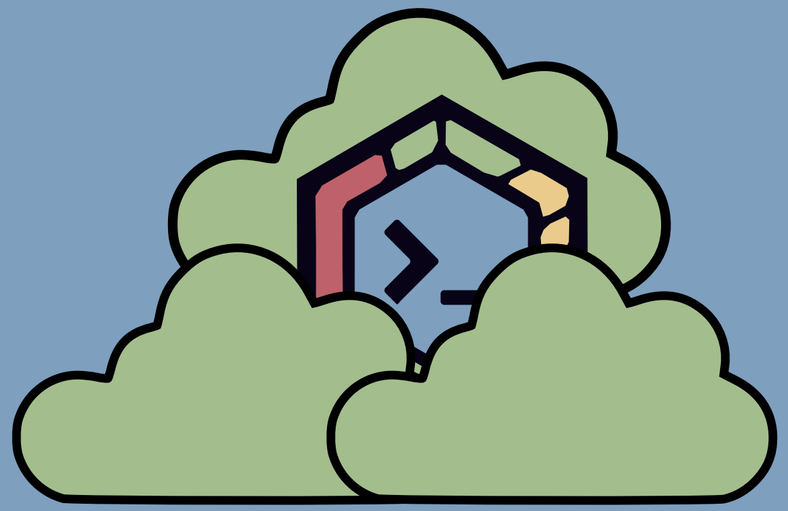
Future Plans

> Windows version

> Web client

> Share / Resurrect sessions remotely as a service

> Plugin marketplace



Would you like to help me sustain my work?

https://github.com/sponsors/imsnif

Questions?

Thank you!

Check out Zellij:

https://github.com/zellij-org/zellij
https://zellij.dev

Learn more about Zellij plugins:

https://zellij.dev/documentation/plugins

Zellij screencasts and tutorials:

https://zellij.dev/screencasts

Join us on:

Discord
Matrix

Follow us on:

Mastodon
Twitter