

# Prerequisites

1. Install the wasm32 Rust target

```
rustup install wasm32-unknown-unknown
```

2. Install the Tauri CLI

```
cargo install tauri-cli
```

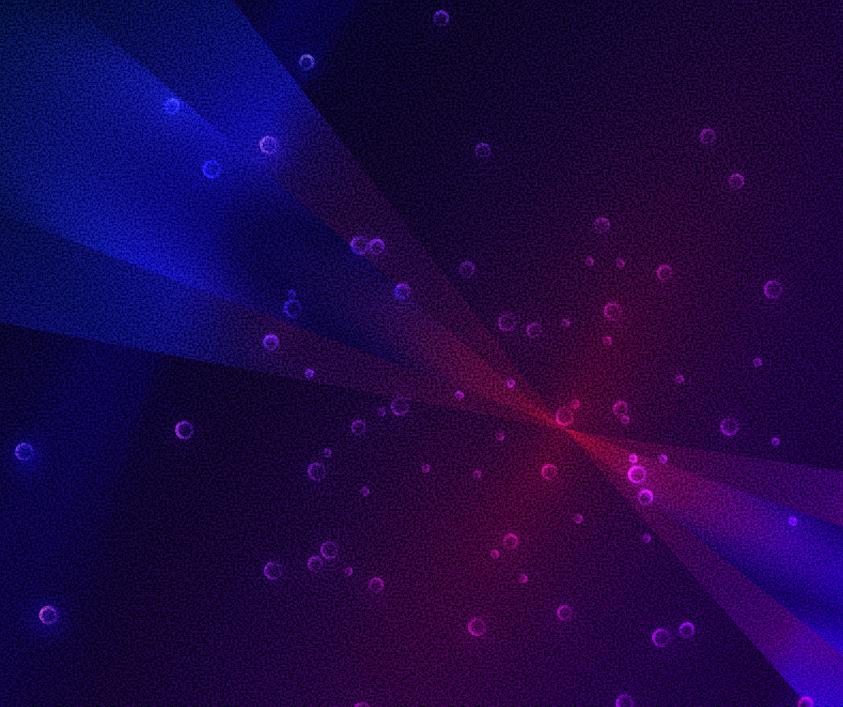
3. Install Trunk

```
cargo install trunk
```

4. Check out the Code

```
git clone https://github.com/crabnebula-dev/rustnation23-workshop
```

# Hands-on with Tauri & Yew



# Intro Me



# ToC

- Prerequisites
- About Tauri

# Prerequisites

1. Install the wasm32 Rust target

```
rustup install wasm32-unknown-unknown
```

2. Install the Tauri CLI

```
cargo install tauri-cli
```

3. Install Trunk

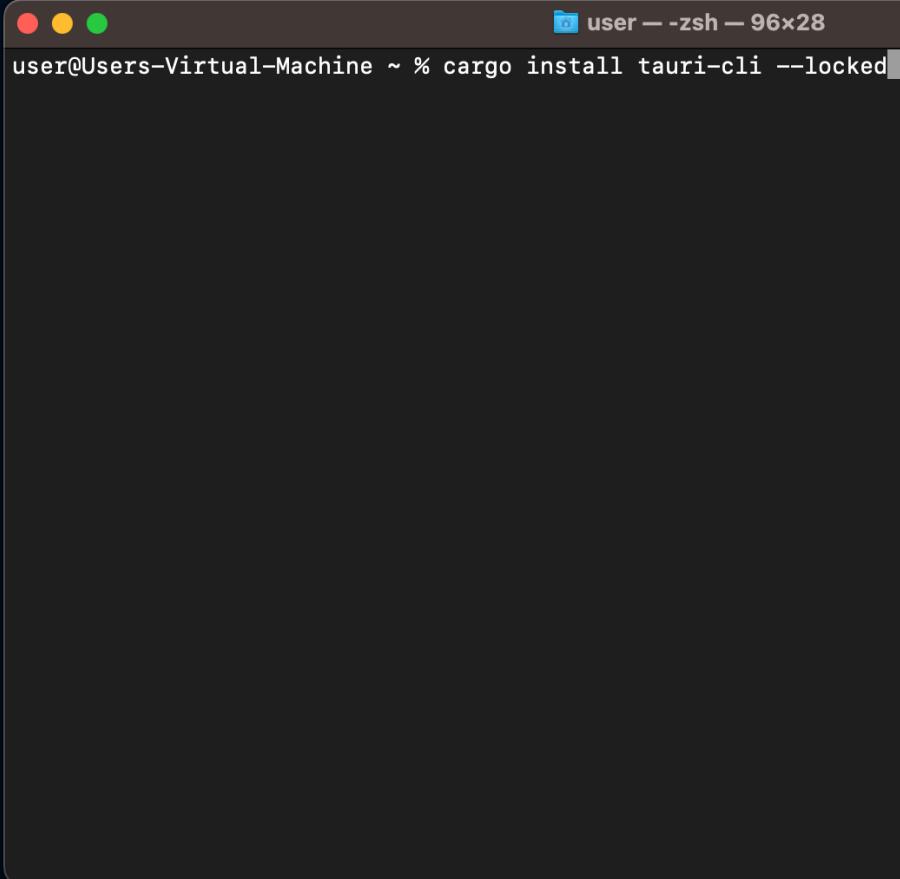
```
cargo install trunk
```

4. Check out the Code

```
git clone https://github.com/crabnebula-dev/rustnation23-workshop
```

# Tauri CLI

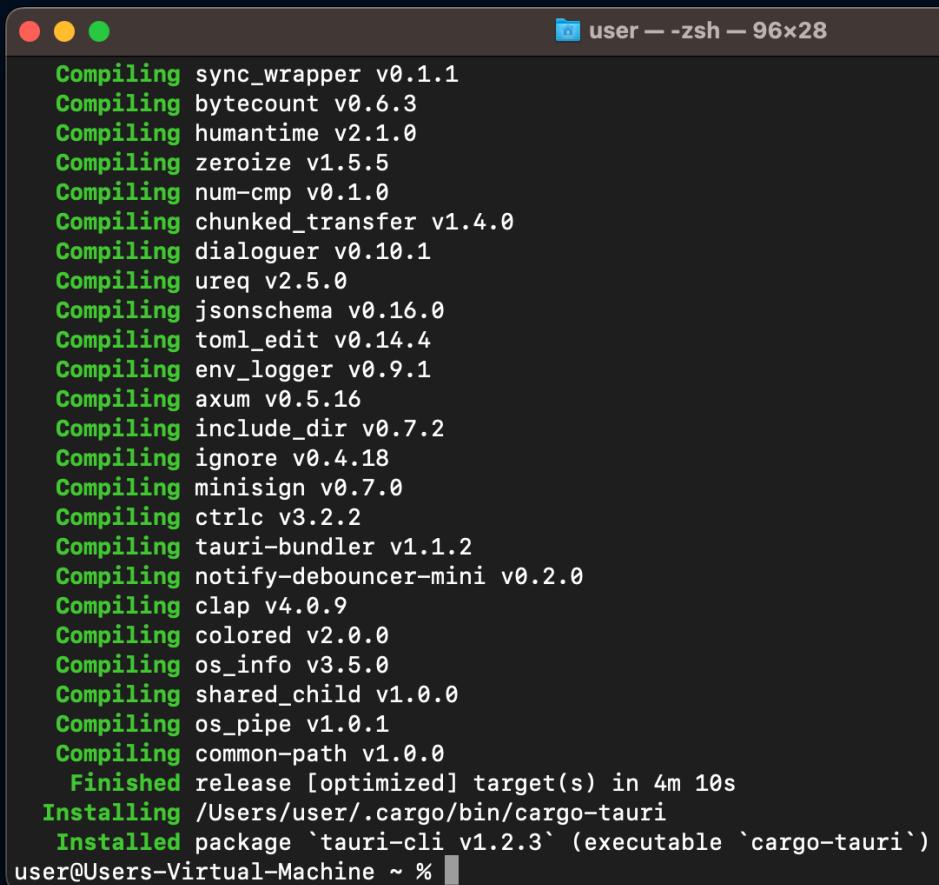
- Development Runner
- Build-tool and Packager
- Generate Icons
- Generate Signing Keys



A screenshot of a dark-themed terminal window. The title bar shows three colored window control buttons (red, yellow, green) and the text "user — -zsh — 96x28". The main area of the terminal shows the command "user@Users-Virtual-Machine ~ % cargo install tauri-cli --locked" entered at the prompt.

# Tauri CLI

- Development Runner
- Build-tool and Packager
- Generate Icons
- Generate Signing Keys

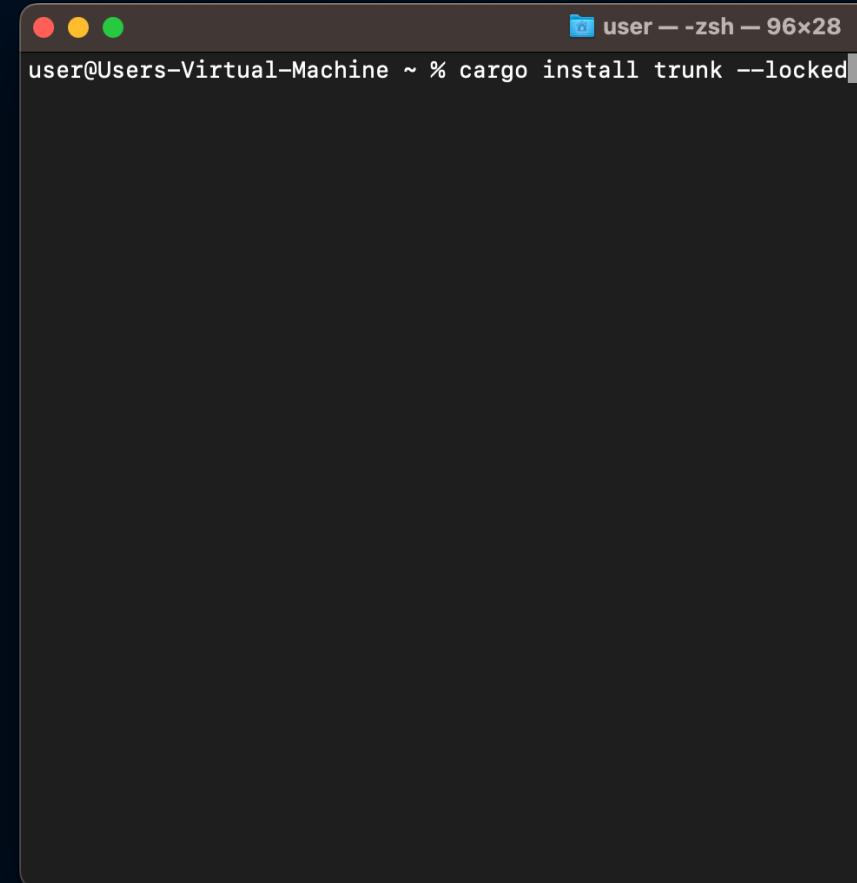


The screenshot shows a macOS terminal window with three colored window controls (red, yellow, green) at the top. The title bar reads "user -- -zsh -- 96x28". The main area of the terminal displays the following text:

```
Compiling sync_wrapper v0.1.1
Compiling bytecount v0.6.3
Compiling humantime v2.1.0
Compiling zeroize v1.5.5
Compiling num-cmp v0.1.0
Compiling chunked_transfer v1.4.0
Compiling dialoguer v0.10.1
Compiling ureq v2.5.0
Compiling jsonschema v0.16.0
Compiling toml_edit v0.14.4
Compiling env_logger v0.9.1
Compiling axum v0.5.16
Compiling include_dir v0.7.2
Compiling ignore v0.4.18
Compiling minisign v0.7.0
Compiling ctrlc v3.2.2
Compiling tauri-bundler v1.1.2
Compiling notify-debouncer-mini v0.2.0
Compiling clap v4.0.9
Compiling colored v2.0.0
Compiling os_info v3.5.0
Compiling shared_child v1.0.0
Compiling os_pipe v1.0.1
Compiling common-path v1.0.0
Finished release [optimized] target(s) in 4m 10s
Installing /Users/user/.cargo/bin/cargo-tauri
Installed package `tauri-cli v1.2.3` (executable `cargo-tauri`)
user@Users-Virtual-Machine ~ %
```

# Trunk

- WASM web app bundler



A screenshot of a dark-themed terminal window. At the top, there are three colored window control buttons (red, yellow, green). To the right of them, the window title is "user -- -zsh -- 96x28". Below the title, the terminal prompt shows the command: "user@Users-Virtual-Machine ~ % cargo install trunk --locked". The rest of the window is blank, indicating the command has not yet been executed or is still running.

# Trunk

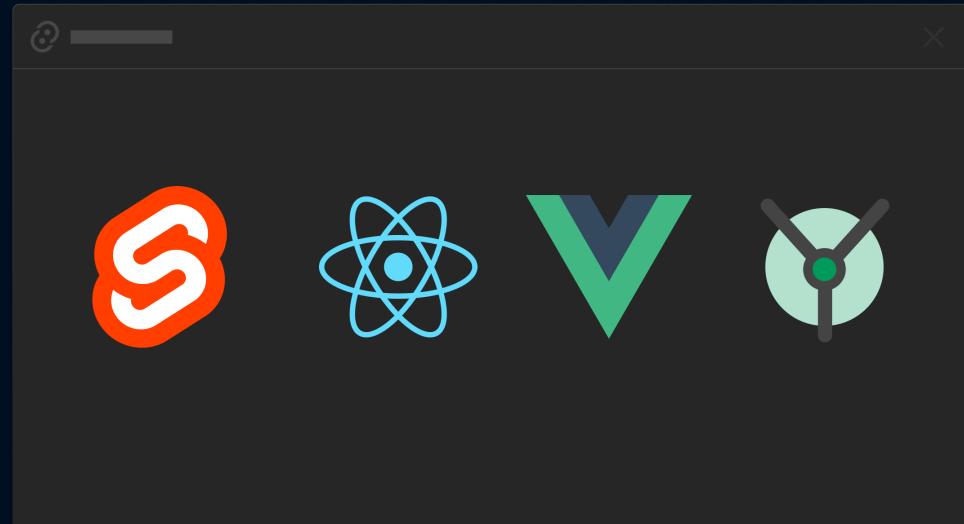
- WASM web app bundler

```
Compiling unicode-width v0.1.9
Compiling open v2.1.3
Compiling console v0.15.0
Compiling which v4.2.5
Compiling axum v0.5.10
Compiling tracing-subscriber v0.3.12
Compiling clap v3.2.8
Compiling directories v4.0.1
Compiling cargo-lock v7.1.0
Compiling tar v0.4.38
Compiling cargo_metadata v0.14.2
Compiling reqwest v0.11.11
Compiling nipper v0.1.9
Compiling notify v4.0.17
Compiling tokio-stream v0.1.9
Compiling envy v0.4.2
Compiling remove_dir_all v0.7.0
Compiling dunce v1.0.2
Compiling seahash v4.1.0
Compiling fs_extra v1.2.0
Compiling bzip2 v0.4.3
Compiling zstd v0.10.2+zstd.1.5.2
Compiling zip v0.6.2
Compiling trunk v0.16.0
Finished release [optimized] target(s) in 3m 21s
Installing /Users/user/.cargo/bin/trunk
Installed package `trunk v0.16.0` (executable `trunk`)
user@Users-Virtual-Machine ~ %
```

**That Tauri thing;  
How does it work?**

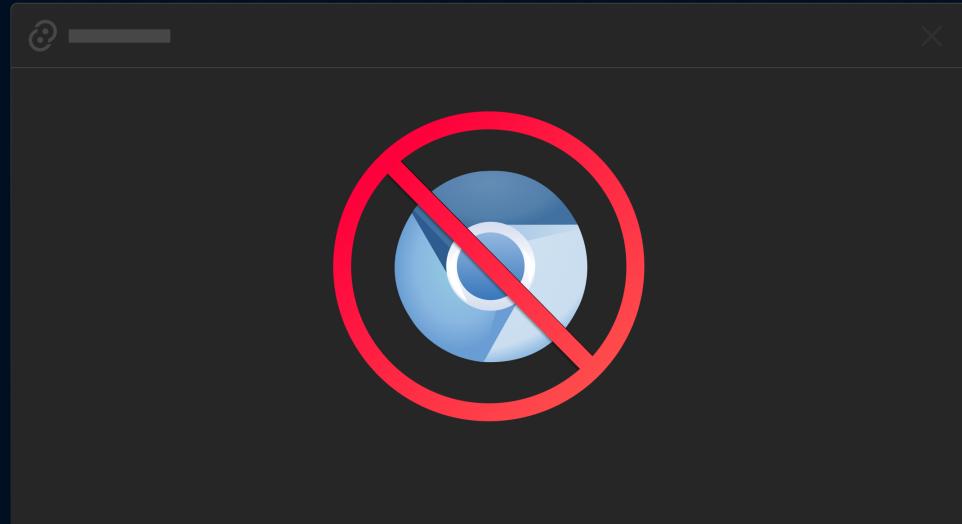
# How does Tauri work?

- Renders HTML/CSS/JS Frontend



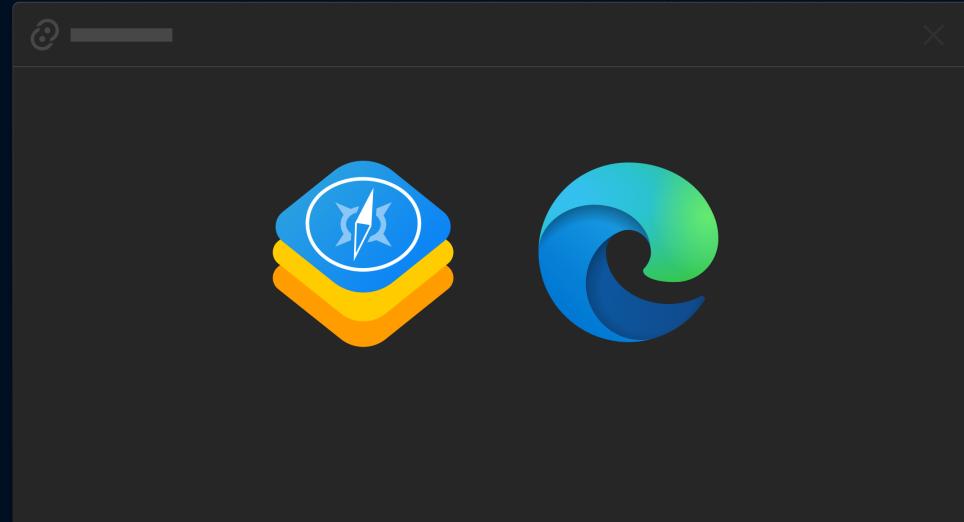
# How does Tauri work?

- Renders HTML/CSS/JS Frontend
- No embedded browser



# How does Tauri work?

- Renders HTML/CSS/JS Frontend
- No embedded browser
- Uses the OS WebView instead



# How does Tauri work?

- WebView is Sandboxed
- No Access to Operating System
- CORS applies



# How does Tauri work?



# How does Tauri work?



# How does Tauri work?



IPC





Window Creation Library

- Windows
- Menus
- System Trays





Webview Rendering Library

- Bindings to the OS Webview
- Windows: Edge WebView2
- macOS: WKWebView
- Linux: WebKitGTK





# TAURI

- Integrates Frontend Framework
- Additional APIs



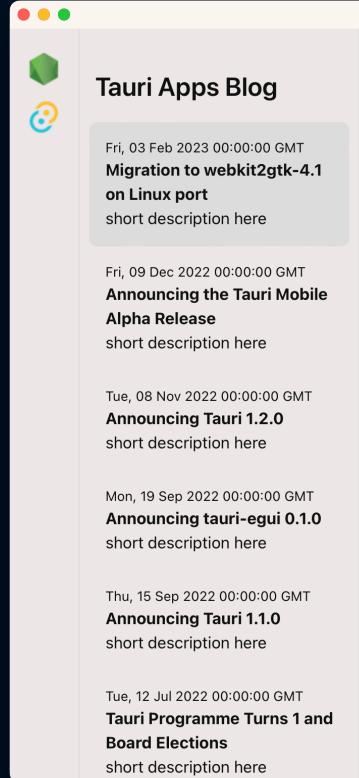


- Integrates Frontend Framework
- Additional APIs
- Bundling & Packaging



# What are we building?

Yet another RSS reader!



Fri, 03 Feb 2023 00:00:00 GMT

## Migration to webkit2gtk-4.1 on Linux

Hello everybody! We just released [Tauri v2.0.0-alpha3](#) recently. While it does huge impacts on Linux port. We will use [WebKit2GTK-4.1](#) in 2.0 from now on.

### What does this mean?

If you are using Tauri version 1.x, there's nothing to worry about. Everything is the same. Tauri version 2.0 alpha version starting from [alpha.3](#), you will need to install WebKit2GTK-4.1. We will update the prerequisites in our site soon. But if you want to know more instructions from [wry](#):

```
# On Arch Linux / Manjaro: sudo pacman -S webkit2gtk-4.1# On Debian/Ubuntu: sudo apt-get install webkit2gtk-4.1
```

### Will this bring breaking changes to my code?

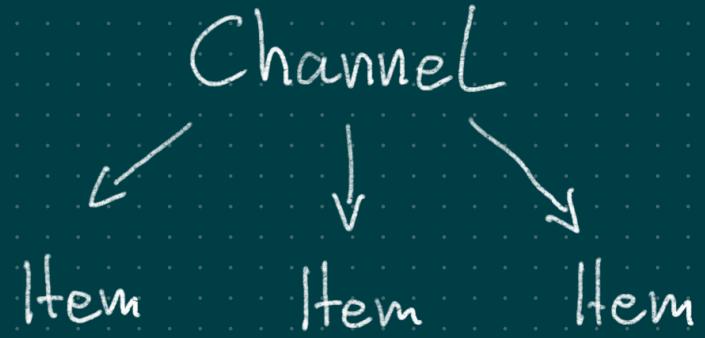
The main difference between version 4.0 and 4.1 are the soup library. WebKit2GTK-4.1 uses soup3. So if you didn't use any soup2-specific APIs, your application should work as usual.

The reason behind this change is because we aim to add flatpak support, also some subtle bugs like [this](#) that only happen in soup2 and they can be fixed.

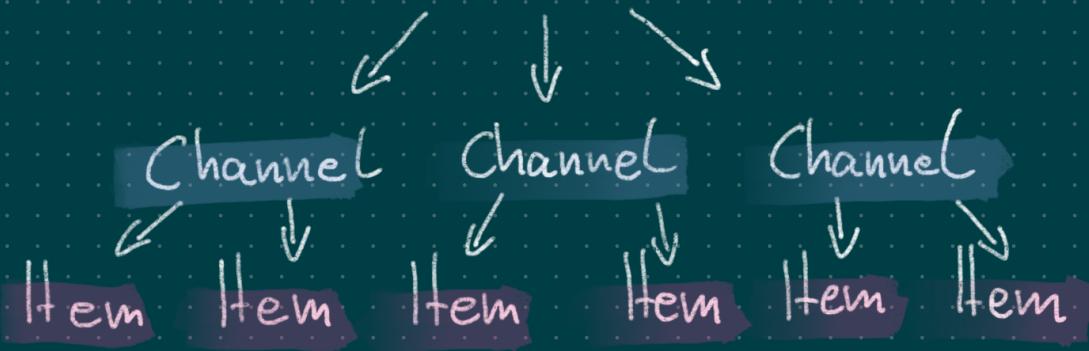
### What other breaking changes we are going to experience?

Channel

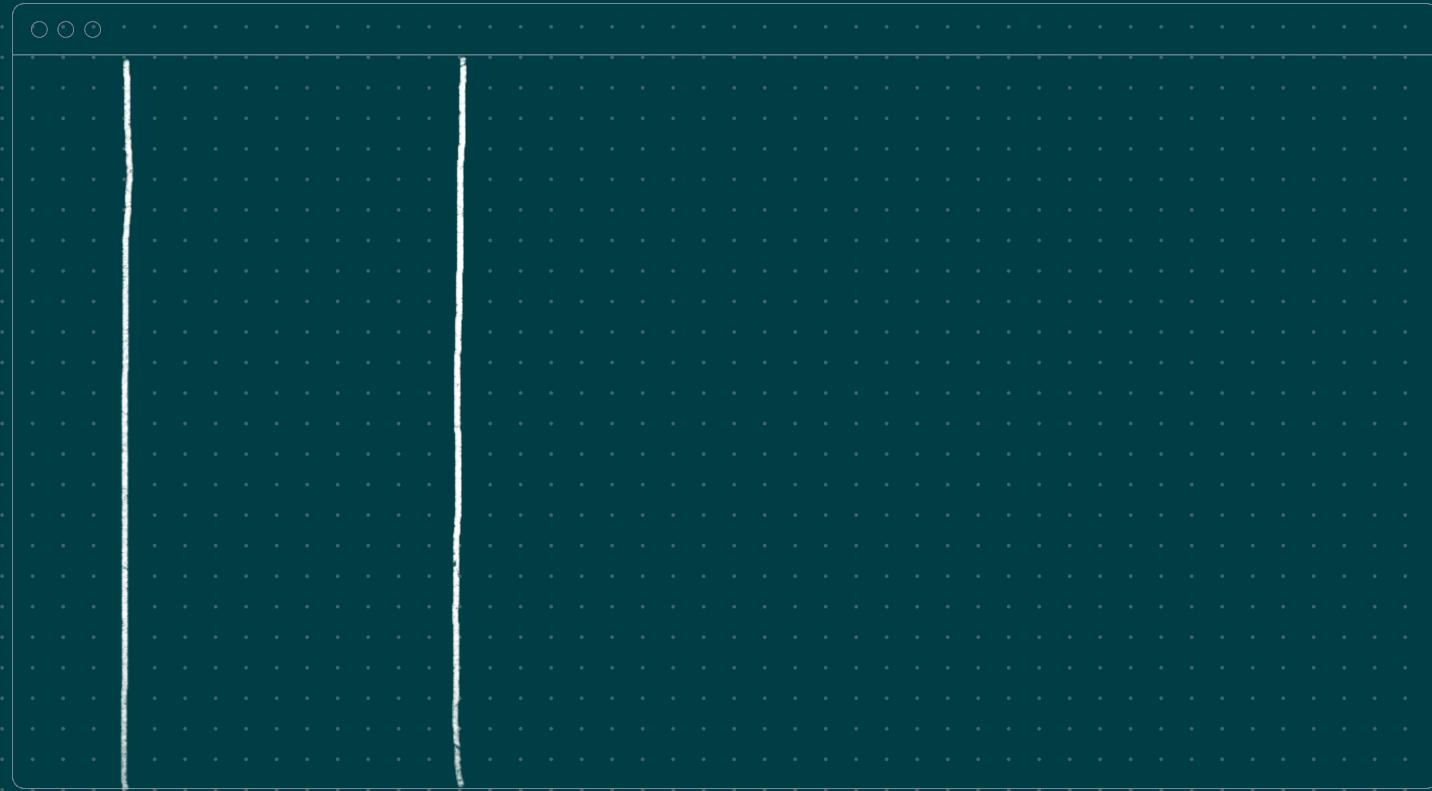
Channel



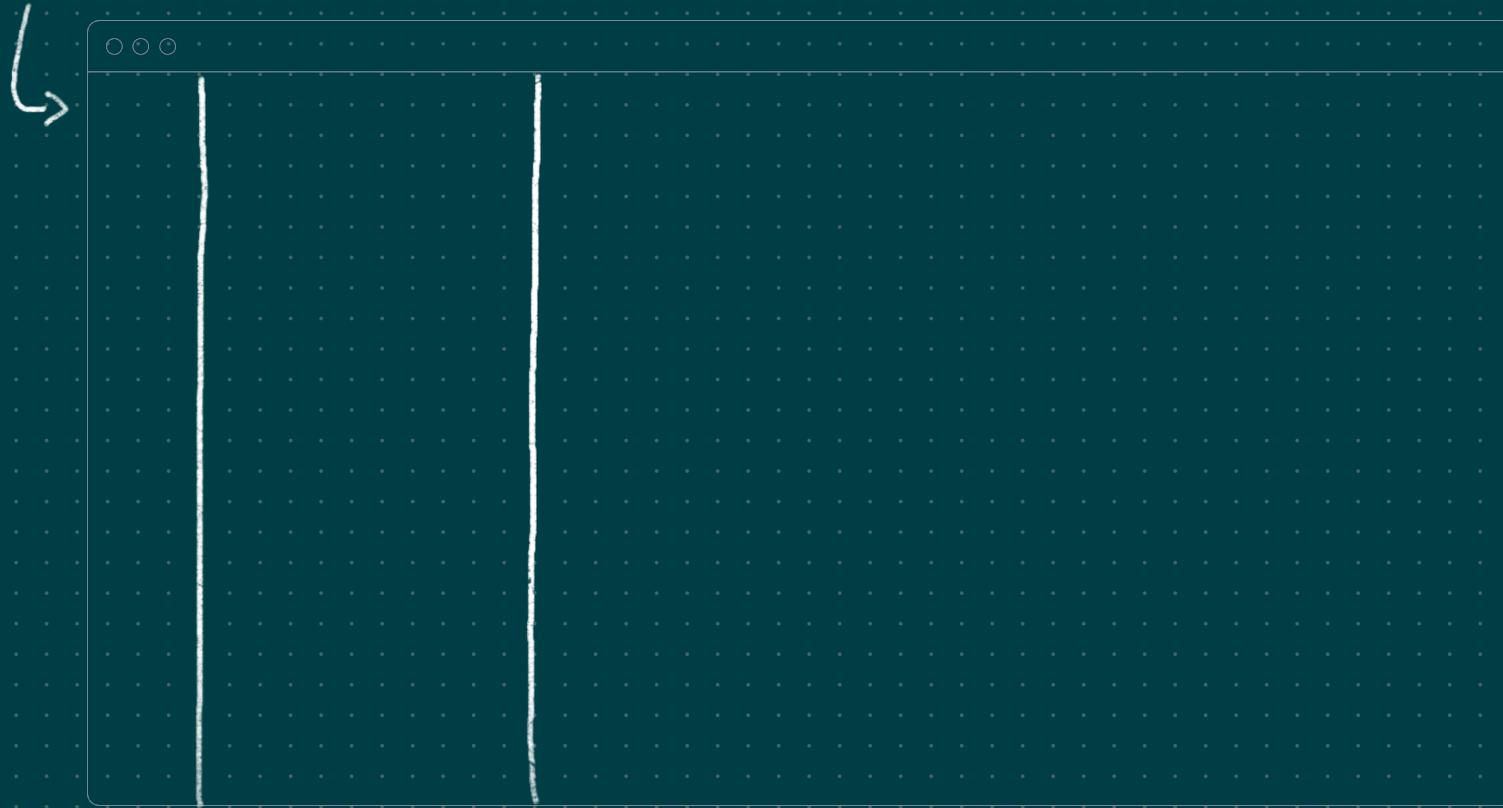
# Channels



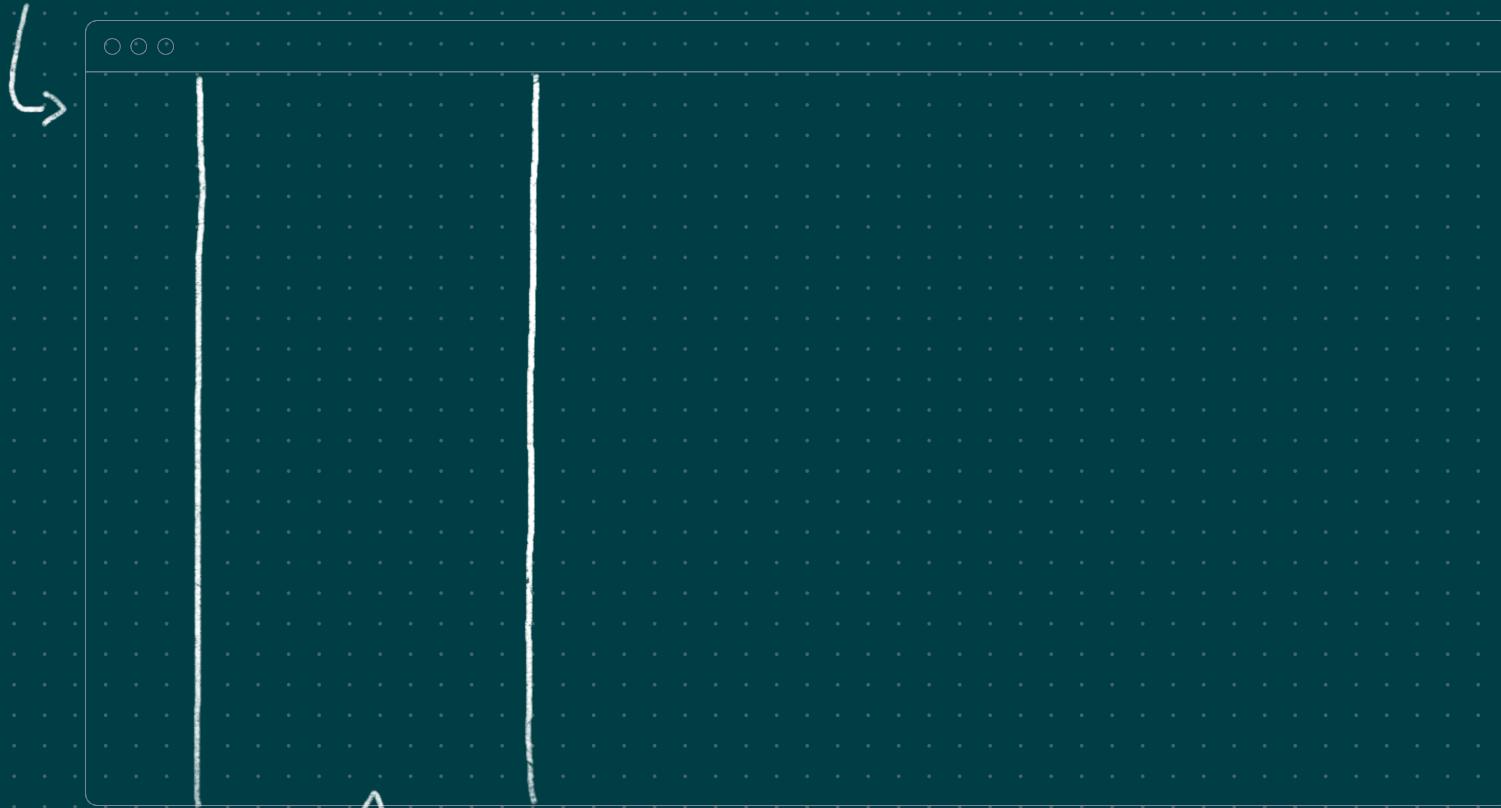




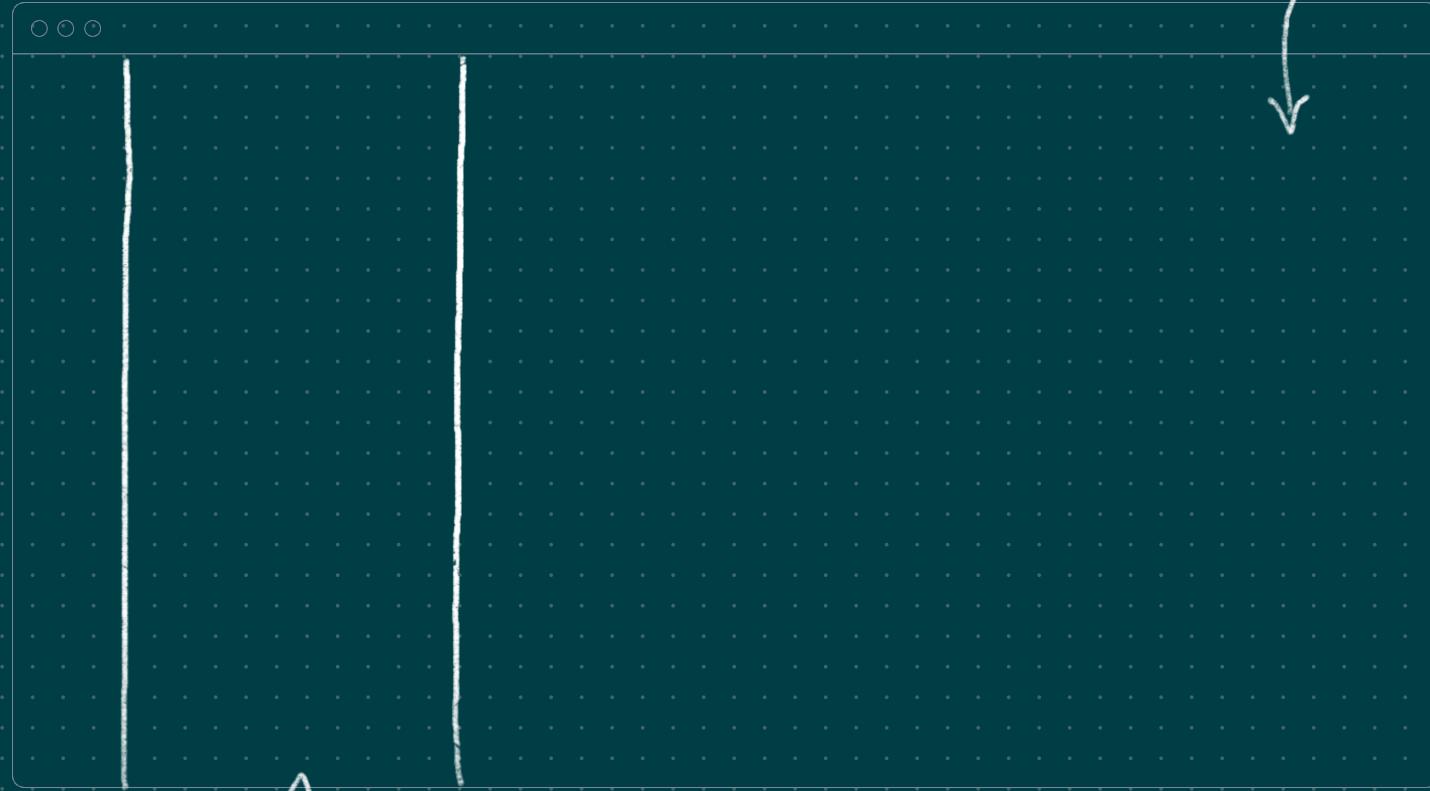
# Channels



# Channels



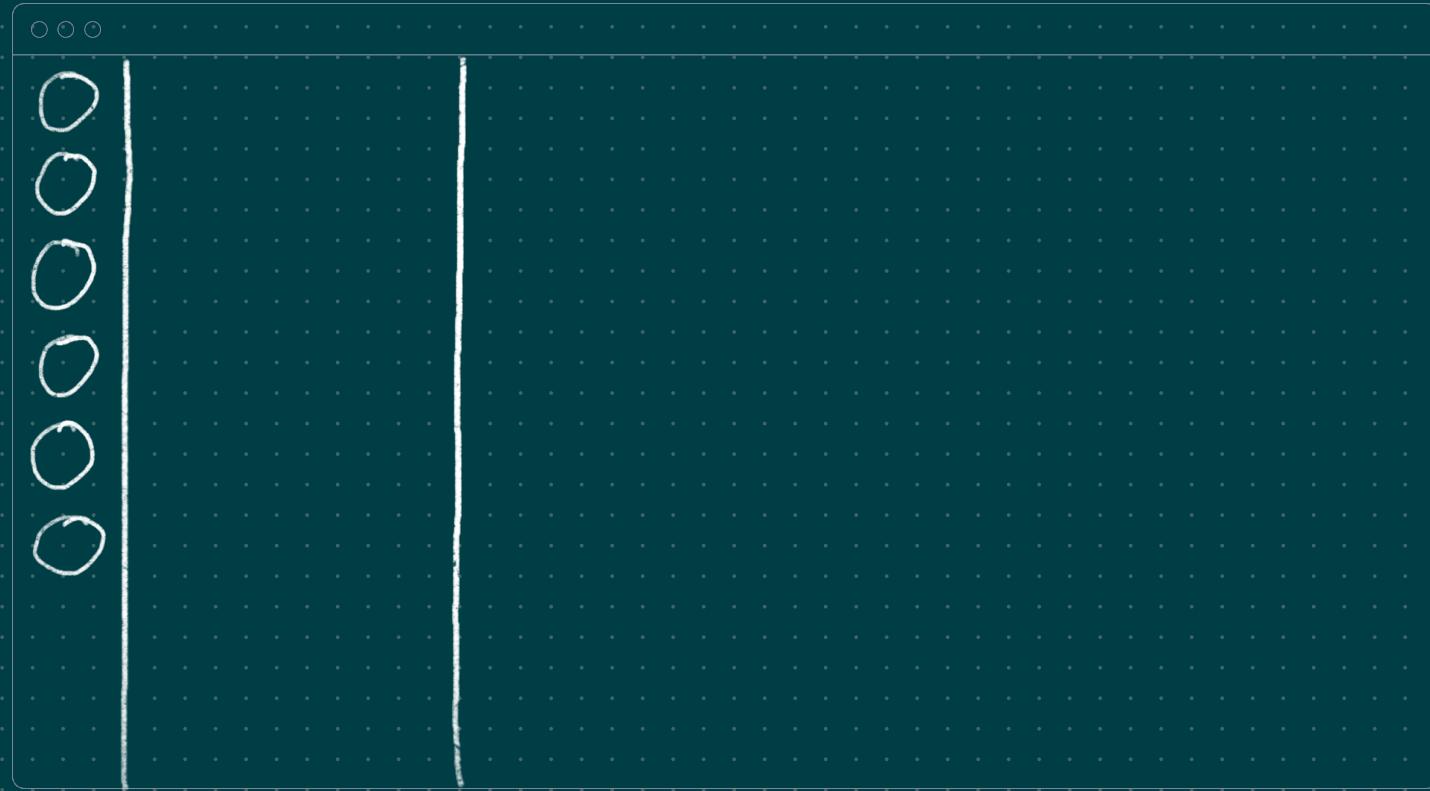
Channels

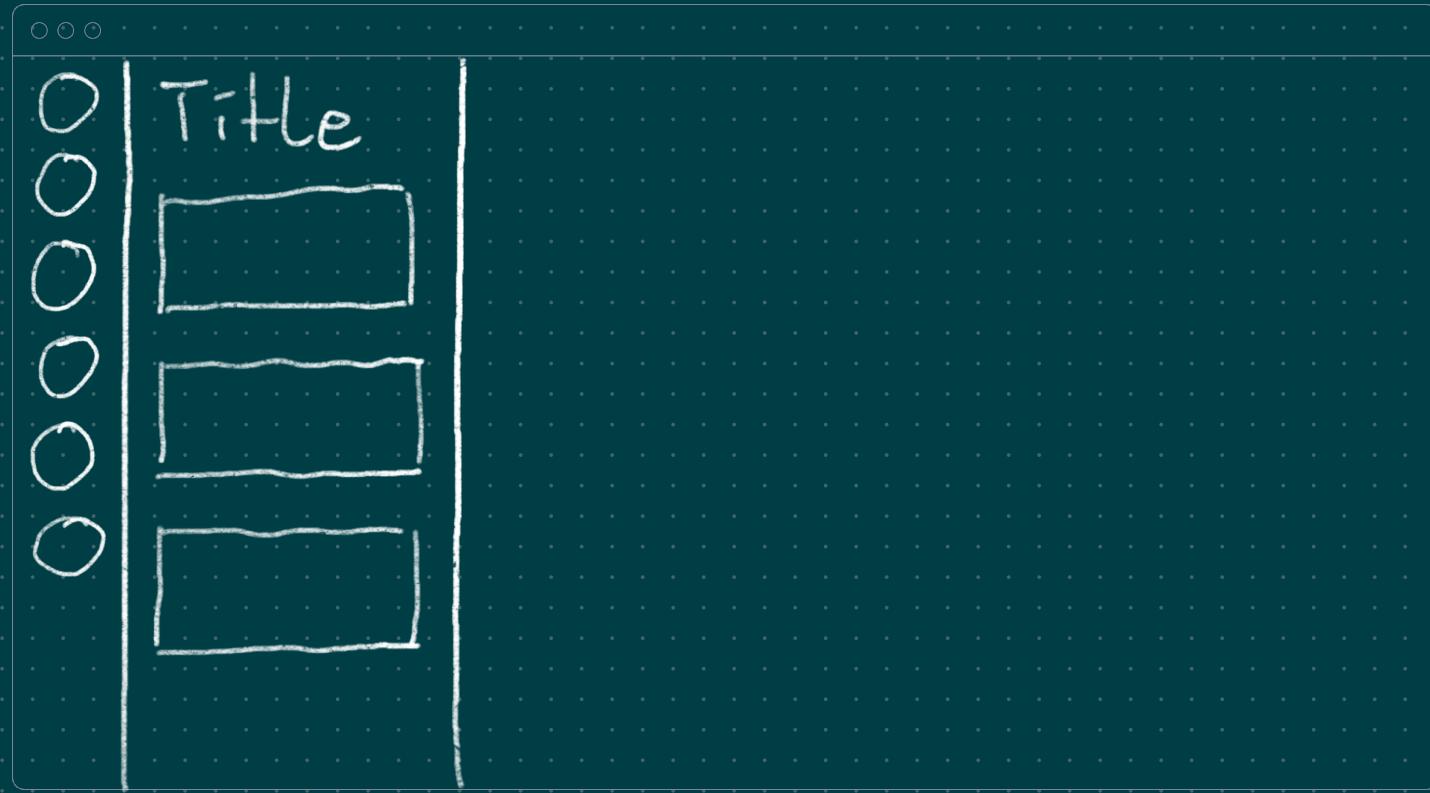


↑ Channel

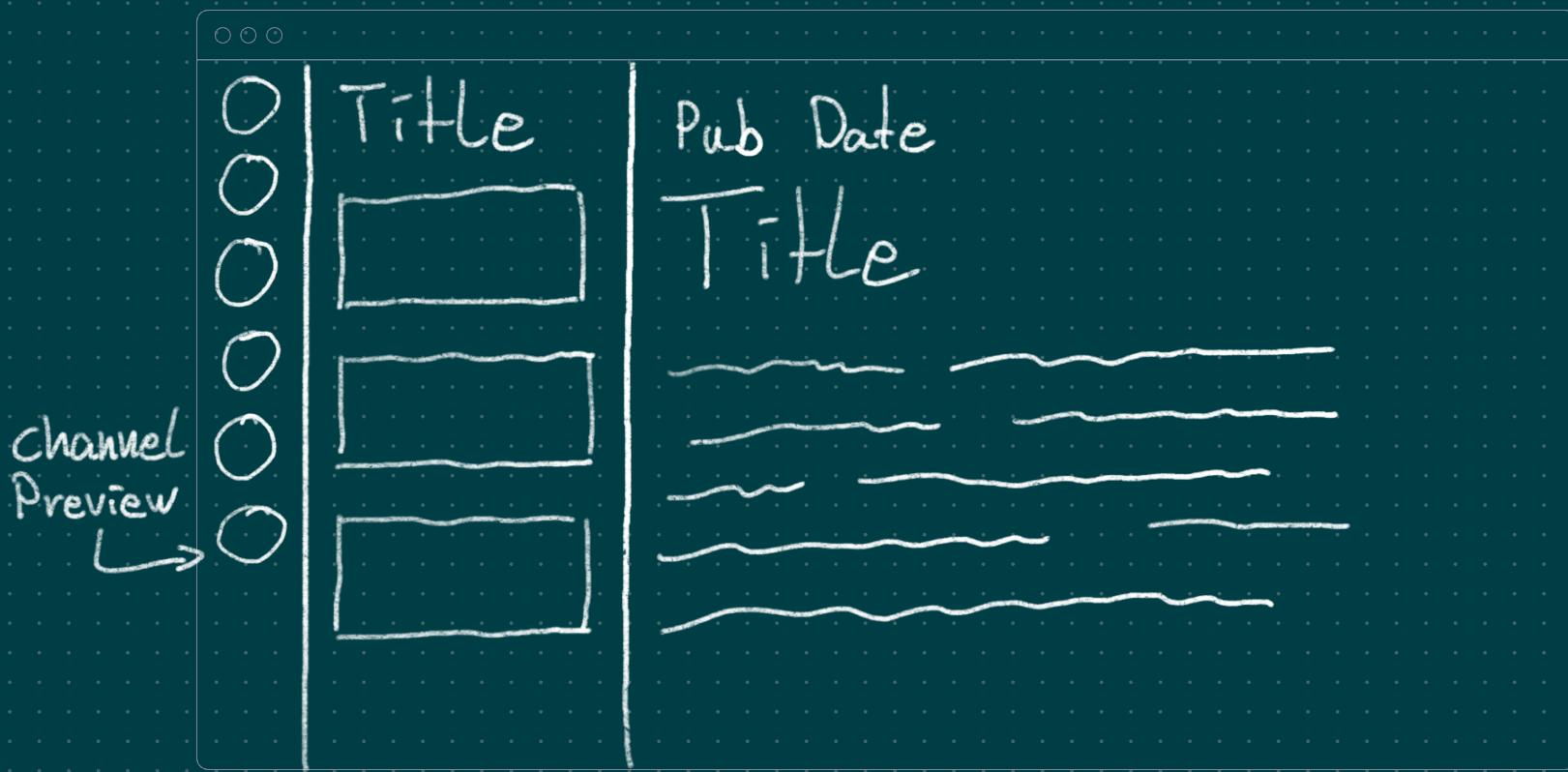
Item



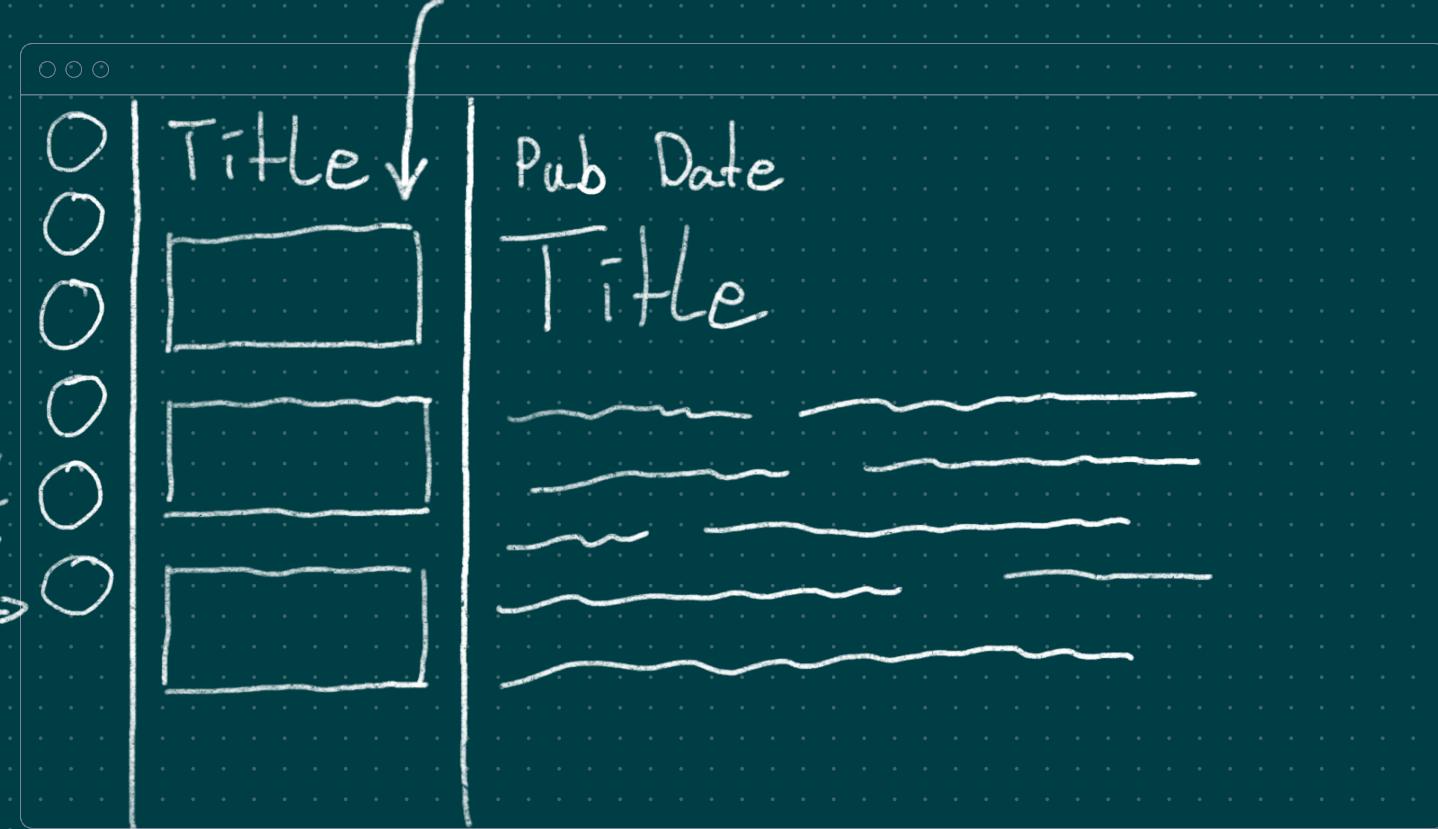




	Title	Pub Date	Title
O			
O			
O			
O			
O			
O			



# Item Preview



# Channels

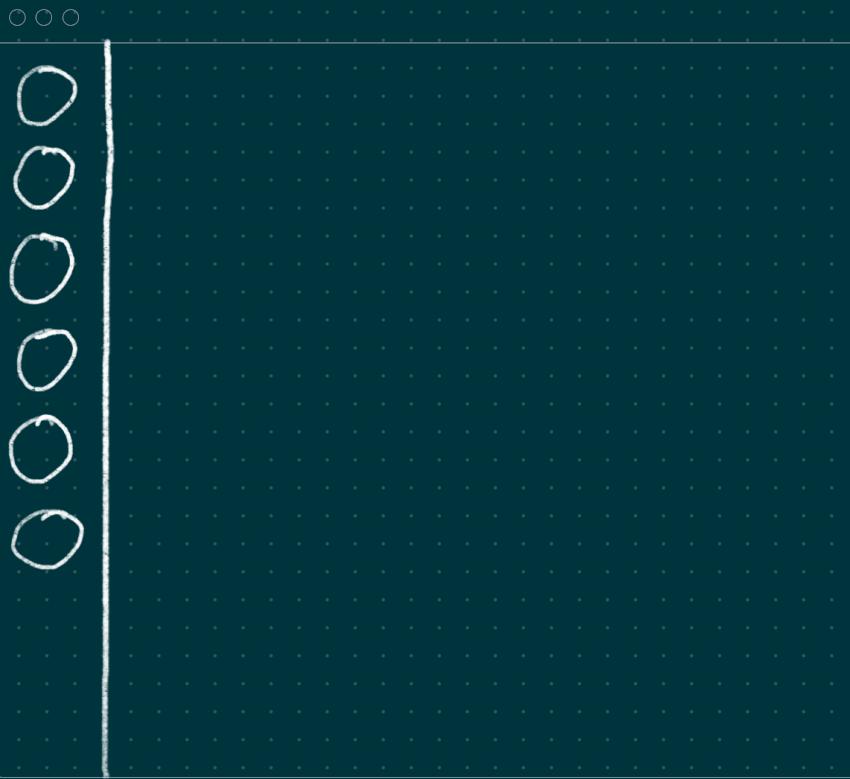
○ ○ ○



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

#[function_component]
pub fn Channels() -> Html {
    html! {
        <aside>
            <ul class="channels">
                {"channels"}
            </ul>
        </aside>
    }
}
```

# ChannelPreview



```
#[function_component]
pub fn ChannelPreview() -> Html {
    html! {
        <li class="channel-preview">
            {"channel icon here"}
        </li>
    }
}
```

# Channel

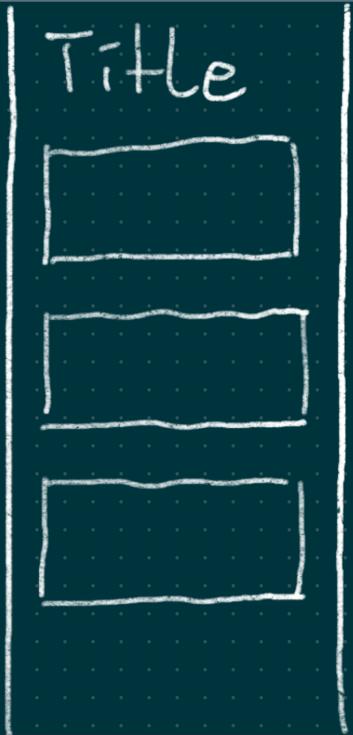
○ ○ ○

Title

```
#[function_component]
pub fn Channel() -> Html {
    html! {
        <aside class="channel">
            <strong class="channel-title">
                {"title"}
            </strong>
            <ul>
                {"items"}
            </ul>
        </aside>
    }
}
```

# Channel

○ ○ ○



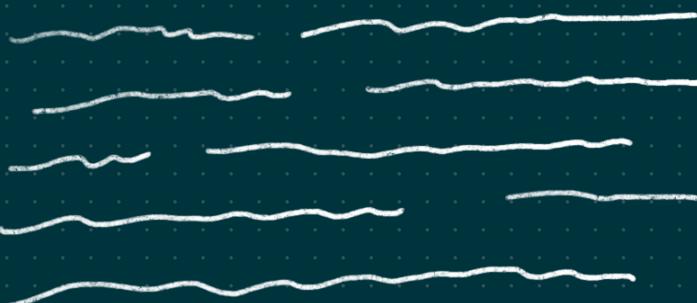
```
#[function_component]
pub fn Channel() -> Html {
    html! {
        <aside class="channel">
            <strong class="channel-title">
                {"title"}
            </strong>
            <ul>
                {"items"}
            </ul>
        </aside>
    }
}
```

# ItemPreview

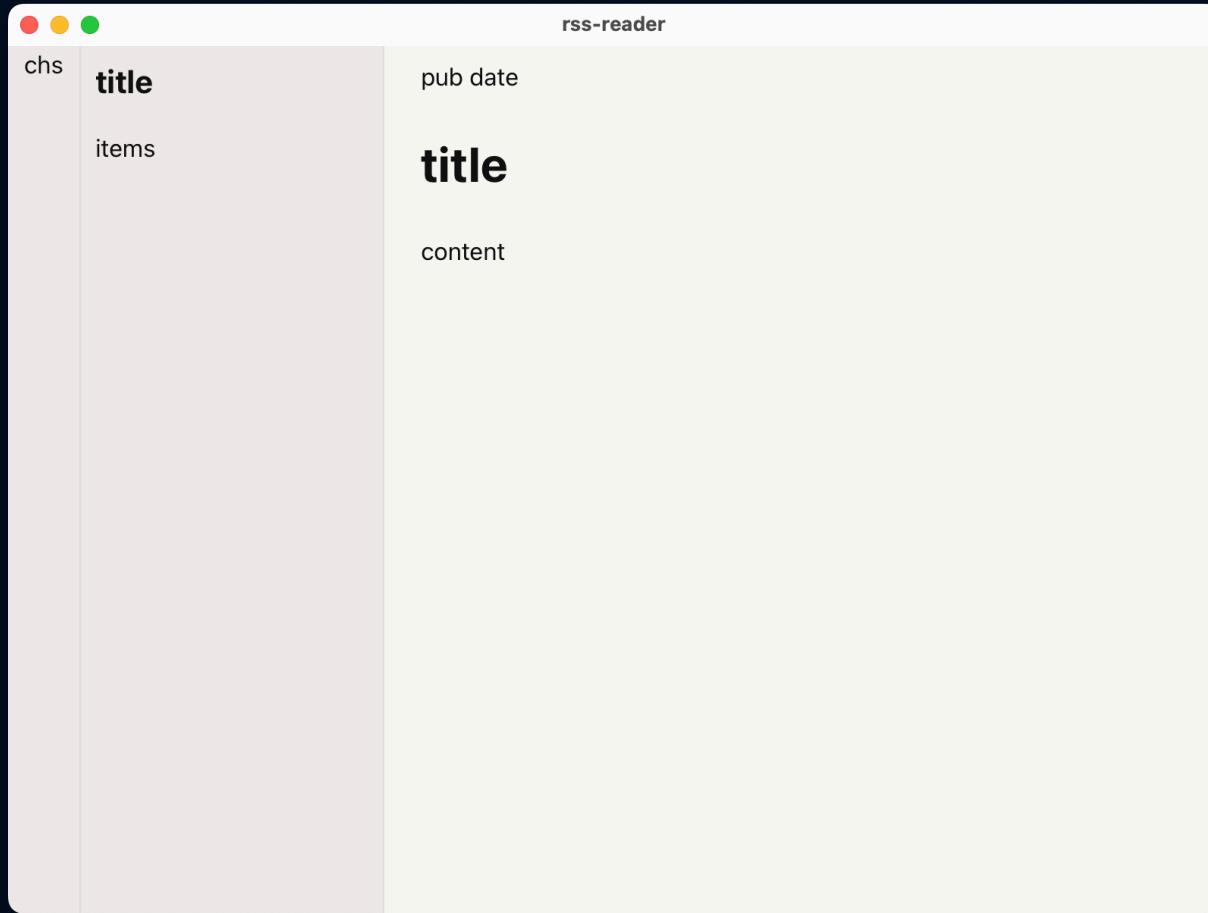
```
#[function_component]
pub fn ItemPreview() -> Html {
    html! {
        <li class="item-preview">
            {"pub date"}
            <strong>
                {"title"}
            </strong>
            <span class="item-preview-
description">
                {"short description"}
            </span>
        </li>
    }
}
```

# Item

Pub Date  
Title

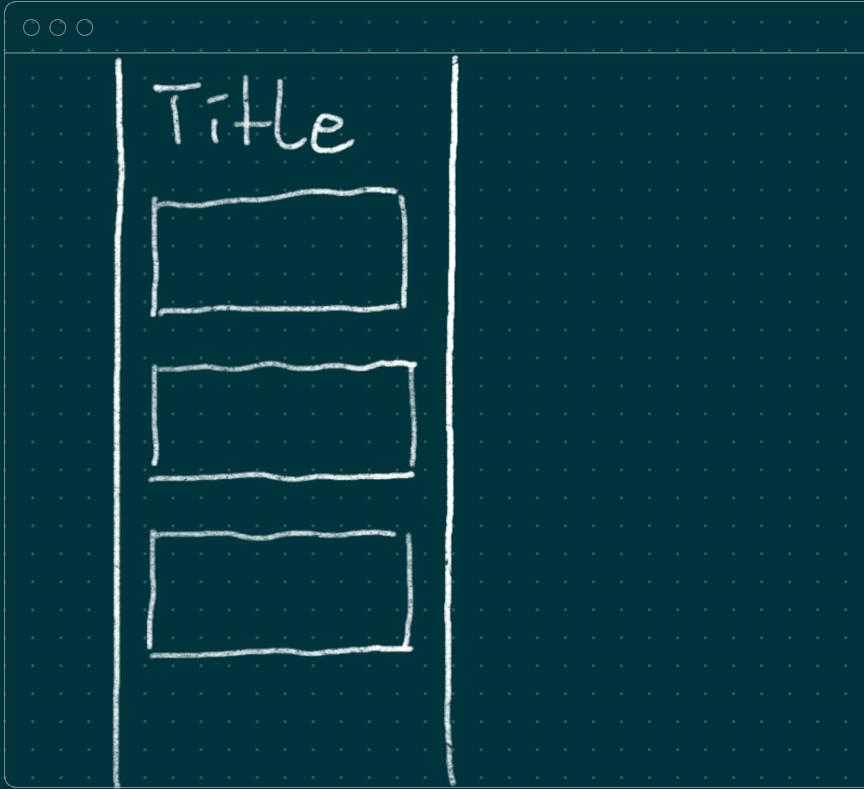


```
# [function_component]
pub fn Item() -> Html {
    html! {
        <main>
            <header>
                {"pub date"}
                <h1>{"title"}</h1>
            </header>
            <article>
                {"content"}
            </article>
        </main>
    }
}
```



# Application Logic

# Channel



- Need to fetch the URL
- Parse the RSS Channel
- Render the title
- Render ItemPreviews

# Commands

```
async fn fetch_channel() {  
}
```

# Commands

```
#[tauri::command]
async fn fetch_channel() {
}
```

# Commands

```
use url::Url;  
  
#[tauri::command]  
async fn fetch_channel(url: Url) {  
}  
}
```

# Commands

```
use url::Url;
use rss::Channel;

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
}
```

# Commands

```
use url::Url;
use rss::Channel;
use tauri::api::http::{HttpRequestBuilder, ResponseType};

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);
}
```

# Commands

```
use url::Url;
use rss::Channel;
use tauri::api::http::{ClientBuilder, HttpRequestBuilder, ResponseType};

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);

    let client = ClientBuilder::new().build()?;
    let content = client.send(request).await?.bytes().await?;
}
```

# Commands

```
use url::Url;
use rss::Channel;
use tauri::api::http::{ClientBuilder, HttpRequestBuilder, ResponseType};

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);

    let client = ClientBuilder::new().build()?;
    let content = client.send(request).await?.bytes().await?;

    let channel = Channel::read_from(&content.data[...])?;
}
```

# Commands

```
use url::Url;
use rss::Channel;
use tauri::api::http::{ClientBuilder, HttpRequestBuilder, ResponseType};

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);

    let client = ClientBuilder::new().build()?;
    let content = client.send(request).await?.bytes().await?;

    let channel = Channel::read_from(&content.data[...])?;
    Ok(channel)
}
```

# Commands

```
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);

    let client = ClientBuilder::new().build()?;
    let content = client.send(request).await?.bytes().await?;

    let channel = Channel::read_from(&content.data[..])?;
    Ok(channel)
}

fn main() {
    tauri::Builder::default()
        .run(tauri::generate_context!())
        .expect("error while running tauri application");
}
```

# Commands

```
let request = HttpRequestBuilder::new("GET", url)?
    .response_type(ResponseType::Binary);

let client = ClientBuilder::new().build()?;
let content = client.send(request).await?.bytes().await?;

let channel = Channel::read_from(&content.data[...])?;

Ok(channel)
}

fn main() {
    tauri::Builder::default()
        .invoke_handler(tauri::generate_handler![fetch_channel])
        .run(tauri::generate_context!())
        .expect("error while running tauri application");
}
```

```
error[E0599]: the method `async_kind` exists for reference `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>`, but its
trait bounds were not satisfied
--> src-tauri/src/main.rs:42:1
|
42 | #[tauri::command]
| ^^^^^^^^^^^^^^ method cannot be called on `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>` due to
unsatisfied trait bounds
...
60 |         .invoke_handler(tauri::generate_handler![fetch_channel])
|                                     ----- in this macro invocation
|
= note: the following trait bounds were not satisfied:
`impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: serde::ser::Serialize`
which is required by `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: tauri::command::private::SerializeKind`  

`<&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: Future>::Output = Result<_, _>`  

which is required by `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: tauri::command::private::ResultFutureKind`  

`&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: Future`  

which is required by `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: tauri::command::private::ResultFutureKind`  

= note: this error originates in the macro `__cmd__fetch_channel` which comes from the expansion of the macro `tauri::generate_handler` (in Nightly builds,
run with -Z macro-backtrace for more info)

For more information about this error, try `rustc --explain E0599`.
error: could not compile `rss-reader` due to previous error
```



```
error[E0599]: the method `async_kind` exists for reference `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>`, but its
trait bounds were not satisfied
--> src-tauri/src/main.rs:42:1
|
42 | #[tauri::command]
| ^^^^^^^^^^^^^^ method cannot be called on `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>` due to
unsatisfied trait bounds
...
60 |     .invoke_handler(tauri::generate_handler![fetch_channel])
|           ----- in this macro invocation
|
= note: the following trait bounds were not satisfied:
`impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: serde::ser::Serialize`
which is required by `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: tauri::command::private::SerializeKind`  

`<&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: Future>::Output = Result<_, _>`  

which is required by `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: tauri::command::private::ResultFutureKind`  

`&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: Future`  

which is required by `&impl Future<Output = Result<Channel, Box<(dyn std::error::Error + 'static)>>>: tauri::command::private::ResultFutureKind`  

= note: this error originates in the macro `__cmd__fetch_channel` which comes from the expansion of the macro `tauri::generate_handler` (in Nightly builds,
run with -Z macro-backtrace for more info)

For more information about this error, try `rustc --explain E0599`.
error: could not compile `rss-reader` due to previous error
```



# Commands

```
use url::Url;
use rss::Channel;
use tauri::api::http::{ClientBuilder, HttpRequestBuilder, ResponseType};

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);

    let client = ClientBuilder::new().build()?;
    let content = client.send(request).await?.bytes().await?;
    let channel = Channel::read_from(&content.data[..])?;
    Ok(channel)
}
```

# Commands

```
use url::Url;
use rss::Channel;
use tauri::api::http::{ClientBuilder, HttpRequestBuilder, ResponseType};

#[tauri::command]
async fn fetch_channel(url: Url) -> Result<Channel, Box<dyn std::error::Error>> {
    let request = HttpRequestBuilder::new("GET", url)?
        .response_type(ResponseType::Binary);

    let client = ClientBuilder::new().build()?;
    let content = client.send(request).await?.bytes().await?;
    let channel = Channel::read_from(&content.data[...])?;
    Ok(channel)
}
```

# Commands



IPC



All types MUST implement `serde::Serialize` !

# Commands

```
#[derive(Debug, thiserror::Error)]
enum Error {
    #[error(transparent)]
    Tauri(#[from] tauri::api::Error),
    #[error(transparent)]
    Rss(#[from] rss::Error)
}
```

# Commands

```
#[derive(Debug, thiserror::Error)]
enum Error {
    #[error(transparent)]
    Tauri(#[from] tauri::api::Error),
    #[error(transparent)]
    Rss(#[from] rss::Error)
}

impl serde::Serialize for Error {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::ser::Serializer,
    {
        serializer.serialize_str(self.to_string().as_ref())
    }
}
```

# Channel

```
#[wasm_bindgen]
extern "C" {
    #[wasm_bindgen(catch, js_namespace = ["window", "__TAURI__", "tauri"])]
    async fn invoke(cmd: &str, args: JsValue) -> Result<JsValue, JsValue>;
}
```

# Channel

```
#[wasm_bindgen]
extern "C" {
    #[wasm_bindgen(catch, js_namespace = ["window", "__TAURI__", "tauri"])]
    async fn invoke(cmd: &str, args: JsValue) -> Result<JsValue, JsValue>;
}

#[derive(Serialize)]
struct FetchChannelArgs<'a> {
    url: &'a Url,
}
```

# Channel

```
# [function_component]
pub fn Channel(props: &ChannelProps) -> HtmlResult {
```

# Channel

```
# [function_component]
pub fn Channel(props: &ChannelProps) -> HtmlResult {
    let result = use_future_with_deps(
        |url| async move {
            },
        props.url.clone(),
    )?;
```

# Channel

```
# [function_component]
pub fn Channel(props: &ChannelProps) -> HtmlResult {
    let result = use_future_with_deps(
        |url| async move {
            let raw_args = serde_wasm_bindgen::to_value(
                &FetchChannelArgs { url: &url }
            )?;
        },
        props.url.clone(),
    )?;
}
```

# Channel

```
#[function_component]
pub fn Channel(props: &ChannelProps) -> HtmlResult {
    let result = use_future_with_deps(
        |url| async move {
            let raw_args = serde_wasm_bindgen::to_value(
                &FetchChannelArgs { url: &url }
            )?;
            let raw_result = invoke("fetch_channel", raw_args).await?;
        },
        props.url.clone(),
    )?;
}
```

# Channel

```
# [function_component]
pub fn Channel(props: &ChannelProps) -> HtmlResult {
    let result = use_future_with_deps(
        |url| async move {
            let raw_args = serde_wasm_bindgen::to_value(
                &FetchChannelArgs { url: &url }
            )?;
            let raw_result = invoke("fetch_channel", raw_args).await?;
            serde_wasm_bindgen::from_value::<rss::Channel>(raw_result)
        },
        props.url.clone(),
    )?;
```

# Channel

```
pub fn Channel(props: &ChannelProps) -> HtmlResult {
    let result = use_future_with_deps(
        |url| async move {
            let raw_args = serde_wasm_bindgen::to_value(
                &FetchChannelArgs { url: &url }
            )?;
            let raw_result = invoke("fetch_channel", raw_args).await?;
            serde_wasm_bindgen::from_value::<rss::Channel>(raw_result)
        },
        props.url.clone(),
    )?;

    let Ok(channel) = &*result else {
        return Ok(html! { "Failed to fetch channel" });
    };
}
```

# Channel

```
    props.url.clone(),
)?;

let Ok(channel) = &*result else {
    return Ok(html! { "Failed to fetch channel" });
};

Ok(html! {
    <aside class="channel">
        <strong class="channel-title">{"title"}</strong>
        <ul>
            {"items"}
        </ul>
    </aside>
})
}
```

# Channel

```
    props.url.clone(),
)?;

let Ok(channel) = &*result else {
    return Ok(html! { "Failed to fetch channel" });
};

Ok(html! {
    <aside class="channel">
        <strong class="channel-title">{&channel.title}</strong>
        <ul>
            {"items"}
        </ul>
    </aside>
})
```



rss-reader

chs

## Tauri Apps Blog

items

pub date

# title

content

# ItemPreview

```
#[function_component]
pub fn ItemPreview(props: &ItemPreviewProps) -> Html {
    let safe_description = props
        .description
        .as_ref()
        .map(|input| AmmoniaBuilder::empty().clean(input).to_string())
        .unwrap_or_default();

    html! {
        <li key={props.id} class="item-preview">
            {"pub date"}
            if let Some(title) = &props.title {
                <strong>
                    {title}
                </strong>
            }
        </li>
    }
}
```

# Description

DOCS.RS

ammonia-3.3.0 ▾

\* Platform ▾

Feature flags

Rust ▾

Find crate



## Crate ammonia

Version 3.3.0

All Items

Re-exports

Structs

Enums

Traits

Functions

Crates

ammonia

Click or press 'S' to search, '?' for more options...

?



source · [-]

## Crate ammonia



Ammonia is a whitelist-based HTML sanitization library. It is designed to prevent cross-site scripting, layout breaking, and clickjacking caused by untrusted user-provided HTML being mixed into a larger web page.

Ammonia uses [html5ever](#) to parse and serialize document fragments the same way browsers do, so it is extremely resilient to syntactic obfuscation.

Ammonia parses its input exactly according to the HTML5 specification; it will not linkify bare URLs, insert line or paragraph breaks, or convert `(C)` into `©`. If you want that, use a markup processor before running the sanitizer, like [pulldown-cmark](#).

## Examples

```
let result = ammonia::clean(
    "<b><img src=' onerror=alert('hax')>I'm not trying to XSS you</b>"
);
assert_eq!(result, "<b><img src=\"\">I'm not trying to XSS you</b>");
```

## Re-exports

```
pub use url;
```

# ItemPreview

```
#[function_component]
pub fn ItemPreview(props: &ItemPreviewProps) -> Html {
    let safe_description = props
        .description
        .as_ref()
        .map(|input| AmmoniaBuilder::empty().clean(input).to_string())
        .unwrap_or_default();

    html! {
        <li key={props.id} class="item-preview">
            {"pub date"}
            if let Some(title) = &props.title {
                <strong>
                    {title}
                </strong>
            }
        </li>
    }
}
```

# ItemPreview

```
.unwrap_or_default();

html! {
    <li key={props.id} class="item-preview">
        {"pub date"}
        if let Some(title) = &props.title {
            <strong>
                {title}
            </strong>
        }
        <span class="item-preview-description">
            {"short description here"}
        </span>
    </li>
}
```

# ItemPreview

```
.unwrap_or_default();

html! {
    <li key={props.id} class="item-preview">
        {"pub date"}
        if let Some(title) = &props.title {
            <strong>
                {title}
            </strong>
        }
        <span class="item-preview-description">
            {&safe_description[0..100.min(safe_description.len())]}
        </span>
    </li>
}
```

# Channel

```
let Ok(channel) = &*result else {
    return Ok(html! { "Failed to fetch channel" });
};

Ok(html! {
    <aside class="channel">
        <strong class="channel-title">{&channel.title}</strong>
        <ul>
            {"items"}
        </ul>
    </aside>
})
```

# Channel

```
let items = channel
  .items
  .iter()
  .enumerate()
  .map(|(id, item)| {
});
```

```
Ok(html! {
    <aside class="channel">
        <strong class="channel-title">{&channel.title}</strong>
        <ul>
            {"items"}
        </ul>
    </aside>
})
```

# Channel

```
let items = channel
  .items
  .iter()
  .enumerate()
  .map(||(id, item)| {
    html! {
      <ItemPreview
        {id}
        channel_url={props.url.clone()}
        channel_title={channel.title.clone()}
        title={item.title.clone()}
        pub_date={item.pub_date.clone()}
        description={item.description.clone()}
      />
    }
  });
}
```

# Channel

```
        title={item.title.clone()}  
        pub_date={item.pub_date.clone()}  
        description={item.description.clone()}  
    />  
}  
});
```

```
Ok(html! {  
    <aside class="channel">  
        <strong class="channel-title">{&channel.title}</strong>  
        <ul  
            {"items"}  
        </ul>  
    </aside>  
})  
}
```

# Channel

```
        title={item.title.clone()}  
        pub_date={item.pub_date.clone()}  
        description={item.description.clone()}  
    />  
}  
});
```

```
Ok(html! {  
    <aside class="channel">  
        <strong class="channel-title">{&channel.title}</strong>  
        <ul>  
            {items.collect::<Html>()}  
        </ul>  
    </aside>  
})  
}
```



rss-reader

chs

## Tauri Apps Blog

pub date

### **Tauri Community Growth & Feedback**

Tauri Community Su...

pub date

### **Migration to webkit2gtk-4.1 on Linux port**

New v2.0 alpha rele...

pub date

### **Announcing the Tauri Mobile Alpha Release**

Tauri mobile is here!...

pub date

### **Announcing Tauri 1.2.0**

The Tauri team is ha...

pub date

## **title**

content

# Takeaways

- Two-process model
- Sharing types between processes is great
- All types crossing the IPC bridge are serialized
- wasm-bindgen let's us harness the power of JavaScript too
- Don't be afraid of new Rust features!

# Exercises for the reader



**EVERY BYTE  
MATTERS**