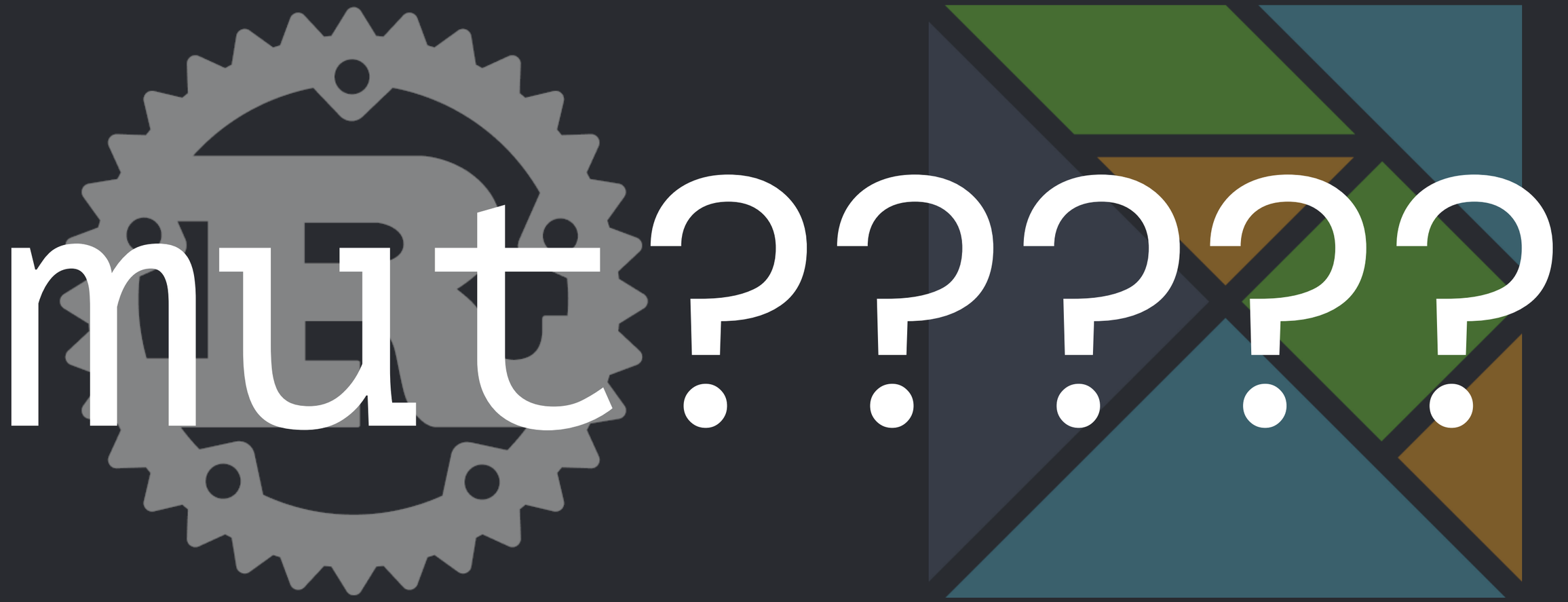
How I used

Rust

to become

Extremely Offline

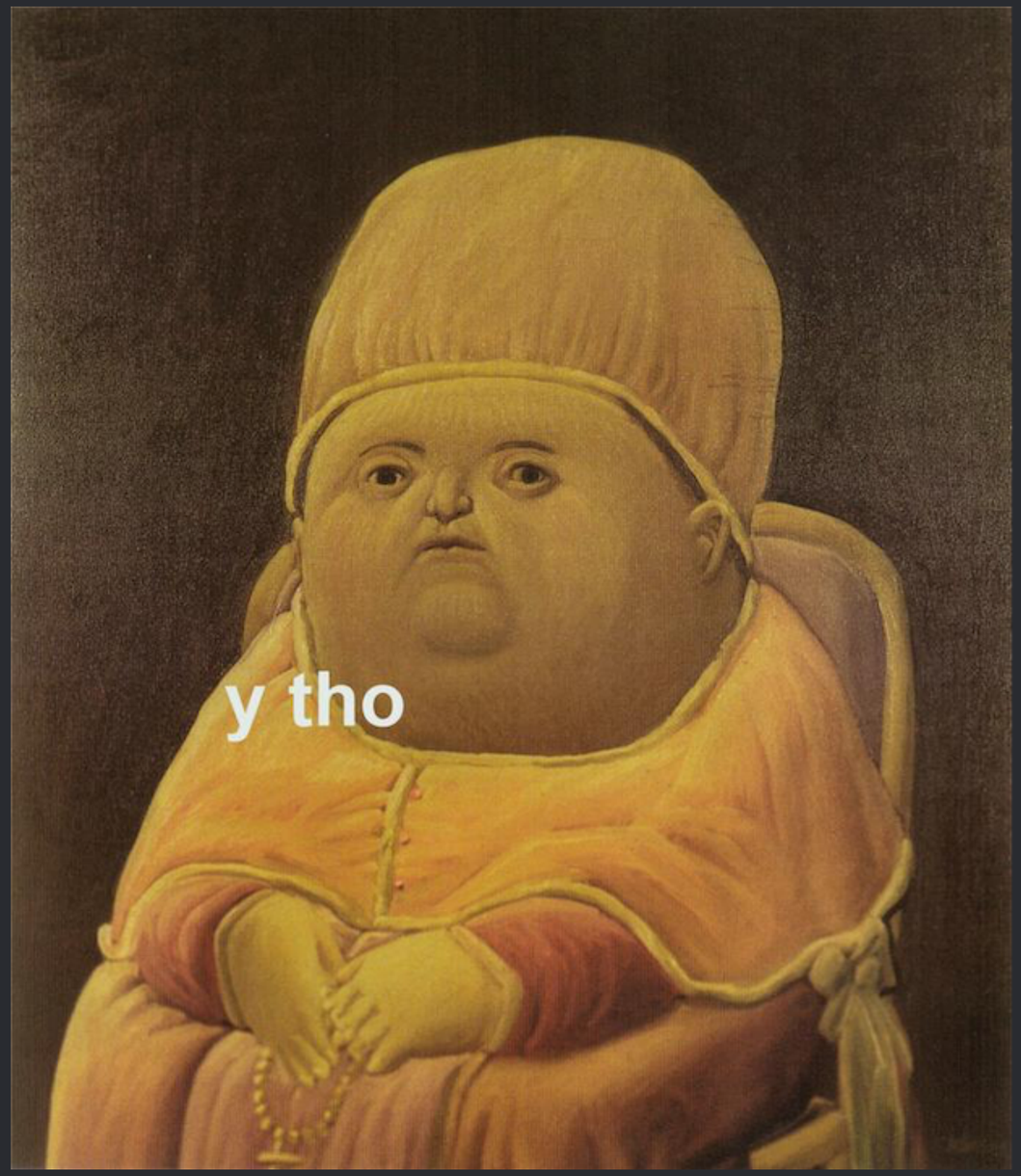
i'm Luke


mut?????

Extremely Offline

not being

Extremely Online


y tho

Why Log Off?

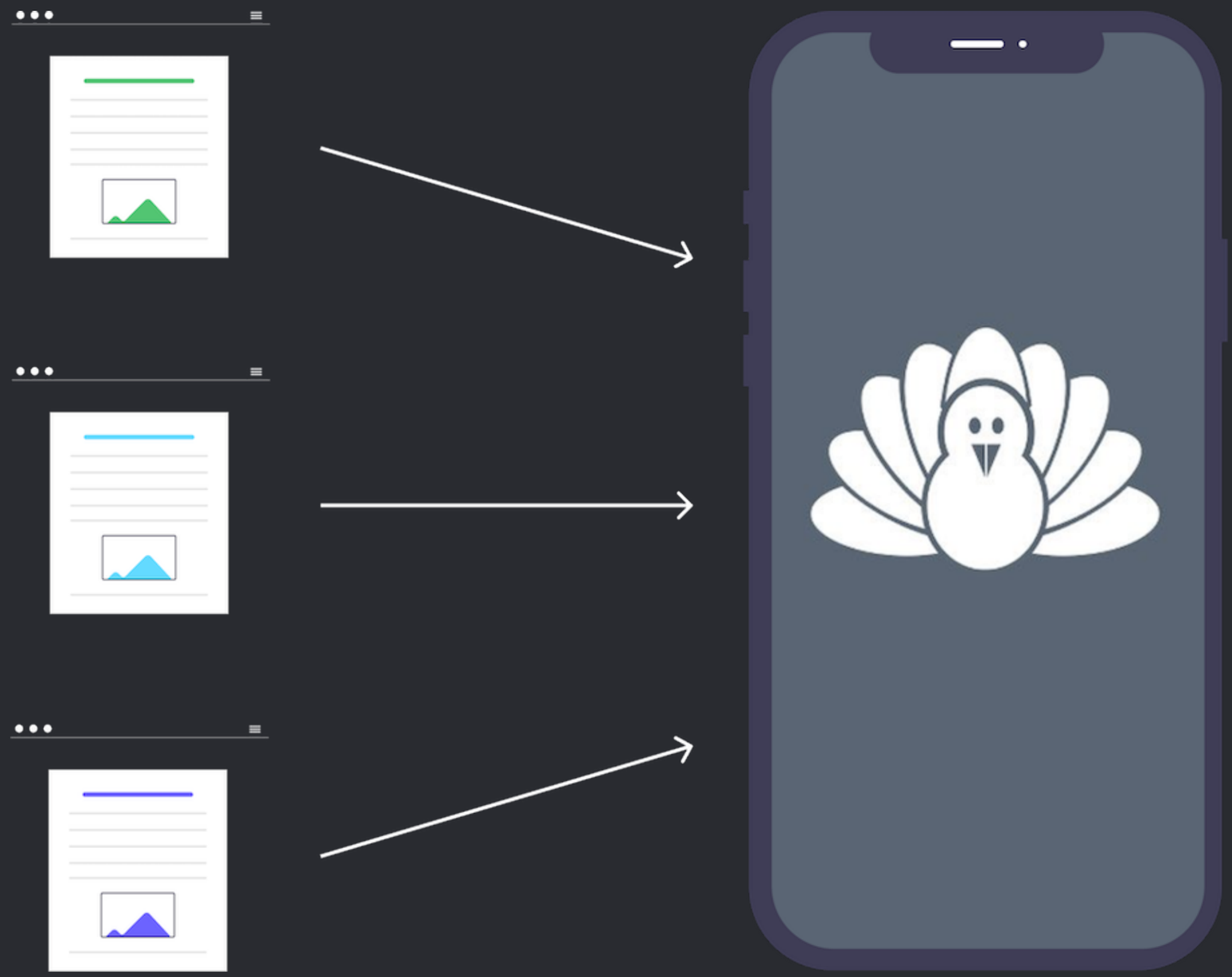
Doom•scroll•ing

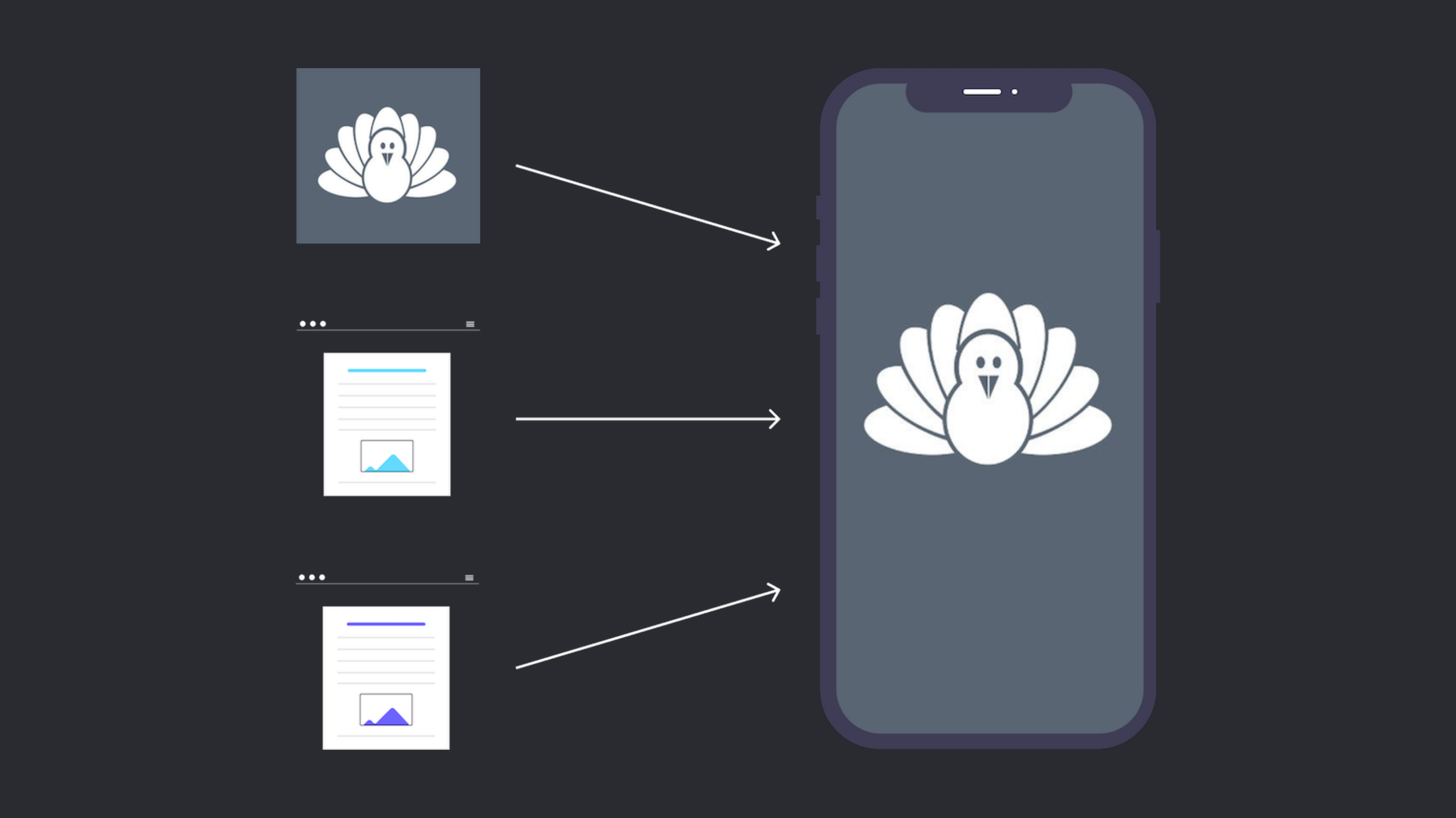
/ˈdumˌskroʊ lɪŋ/

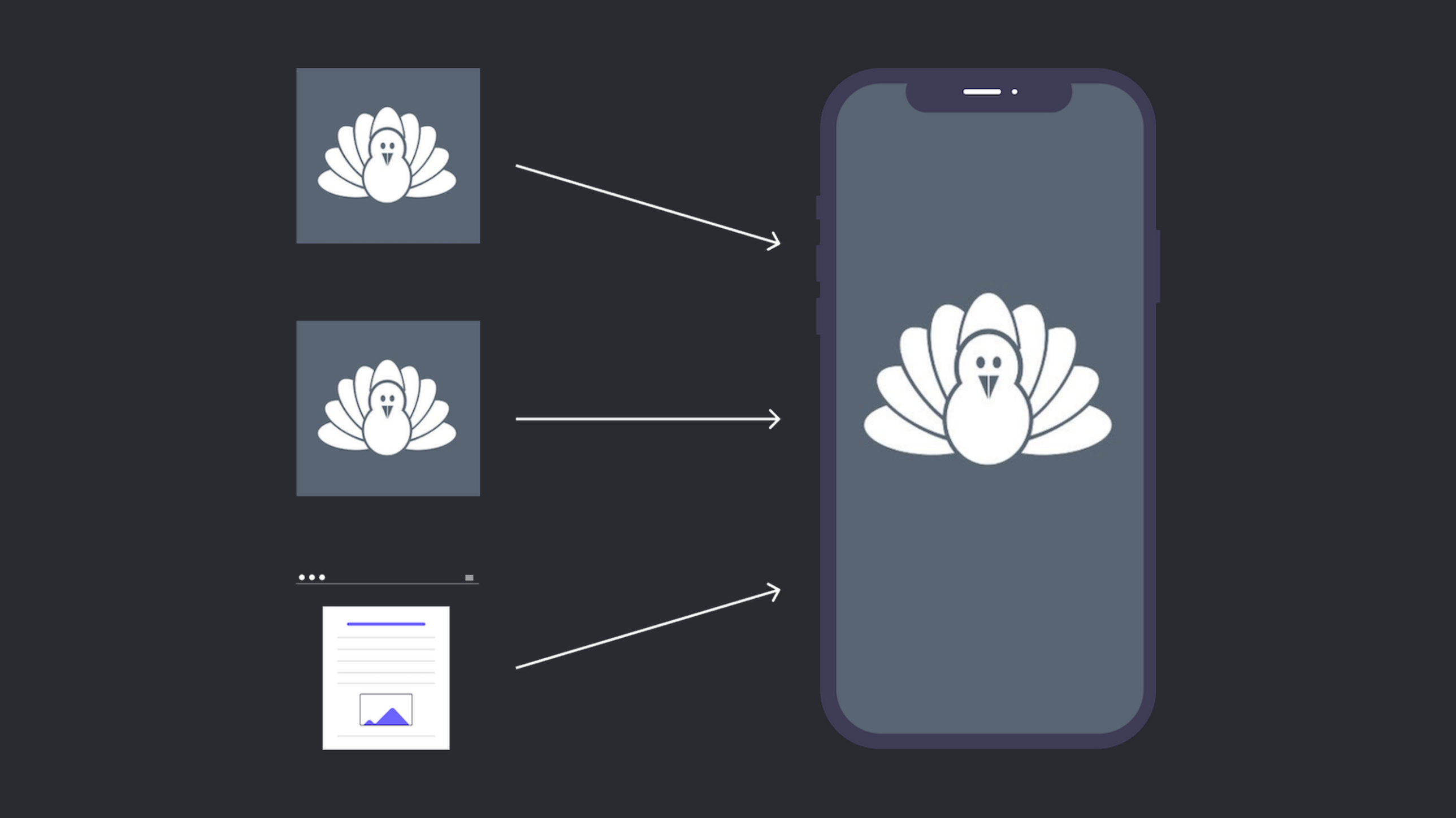
It means scrolling through doom!

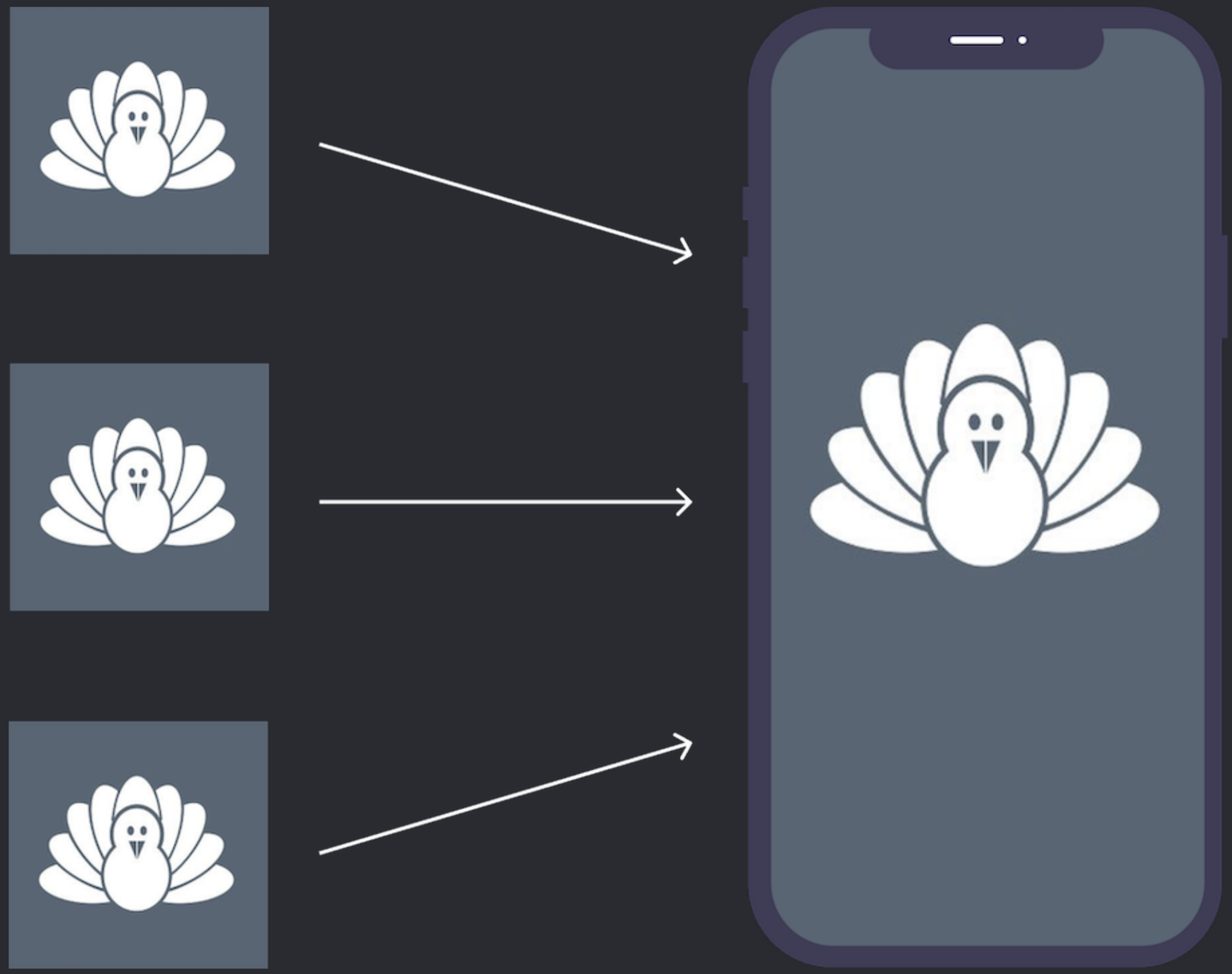
Cold Turkey

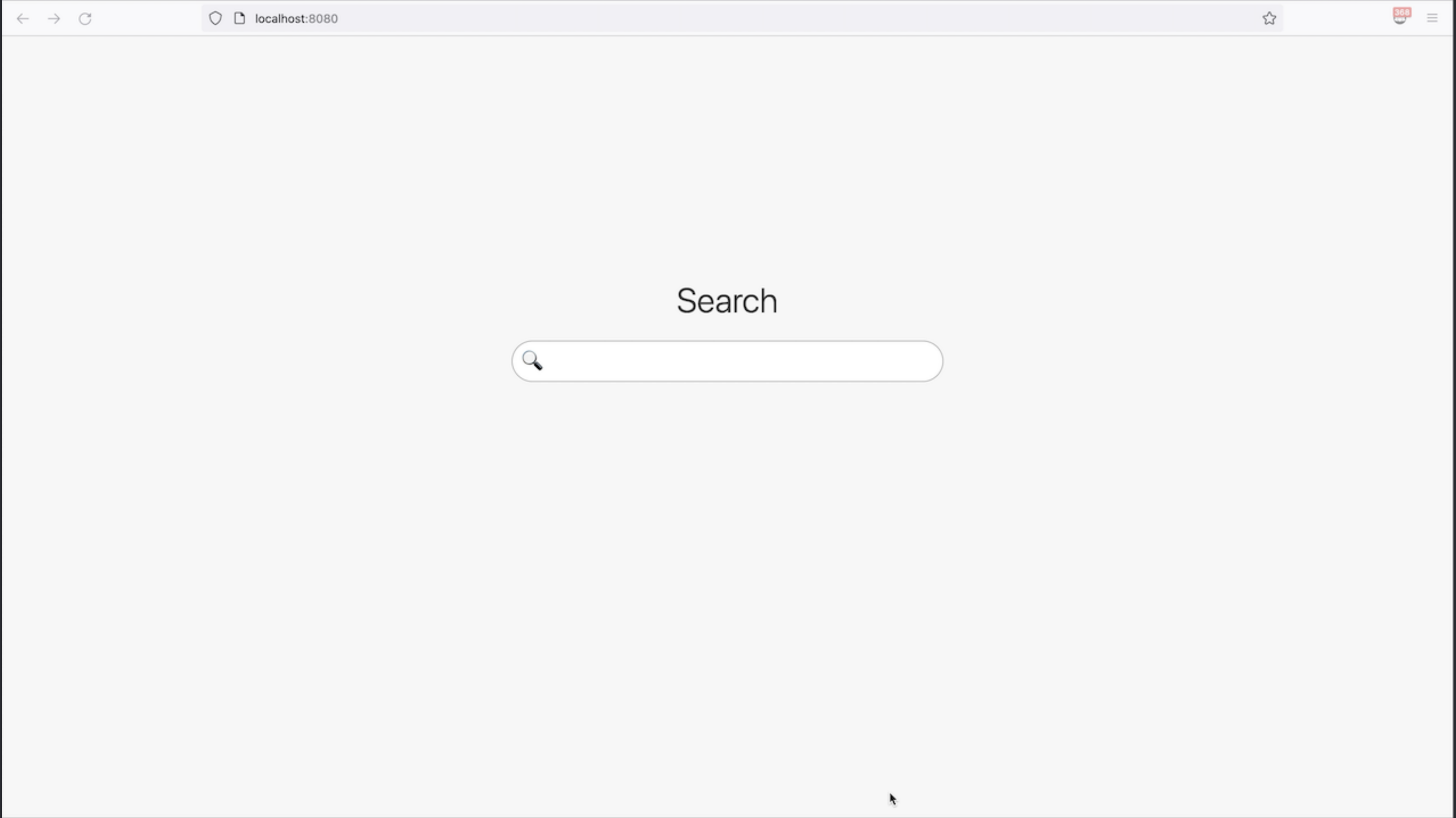
getcoldturkey.com

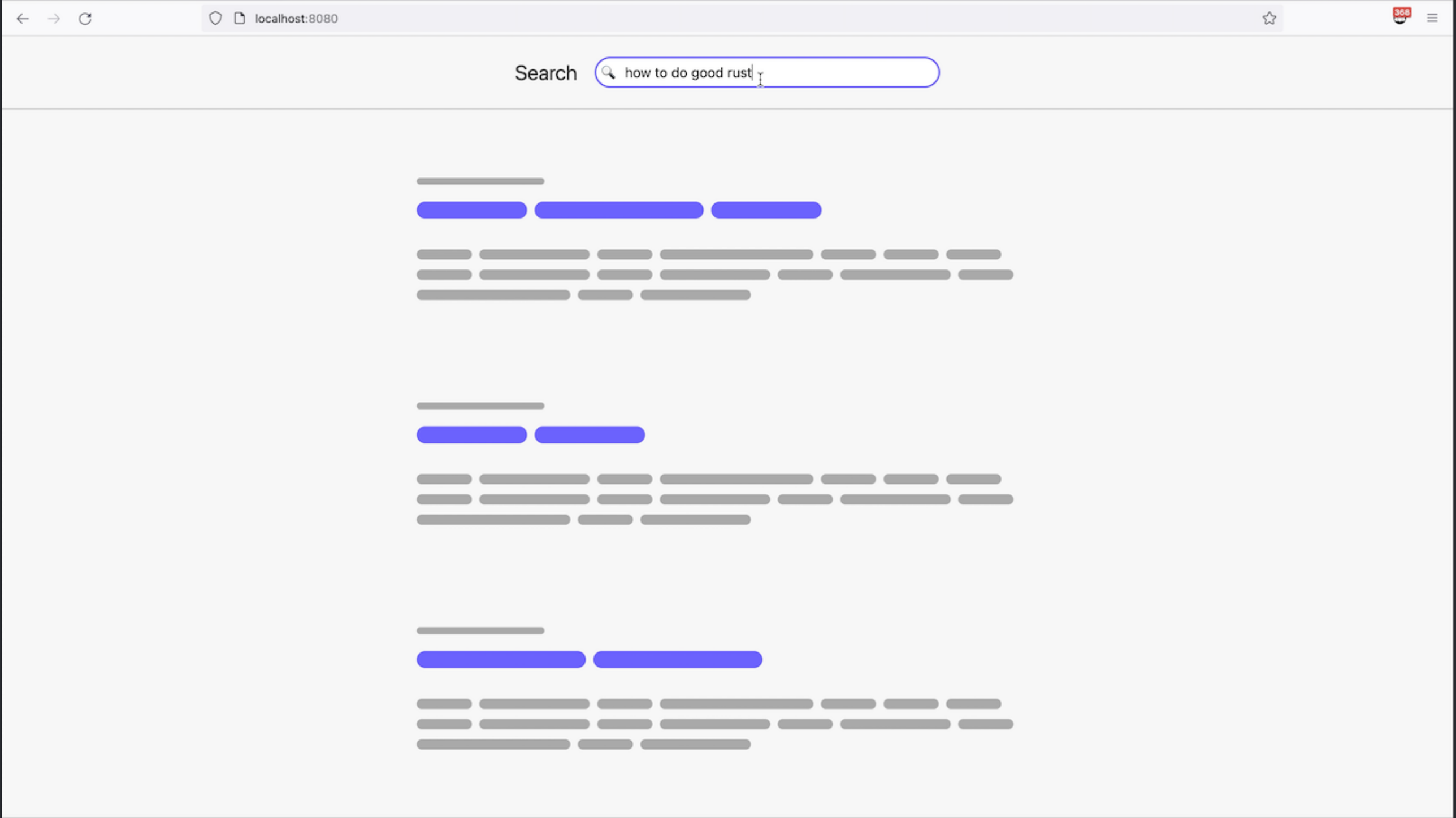









localhost:8080
Search

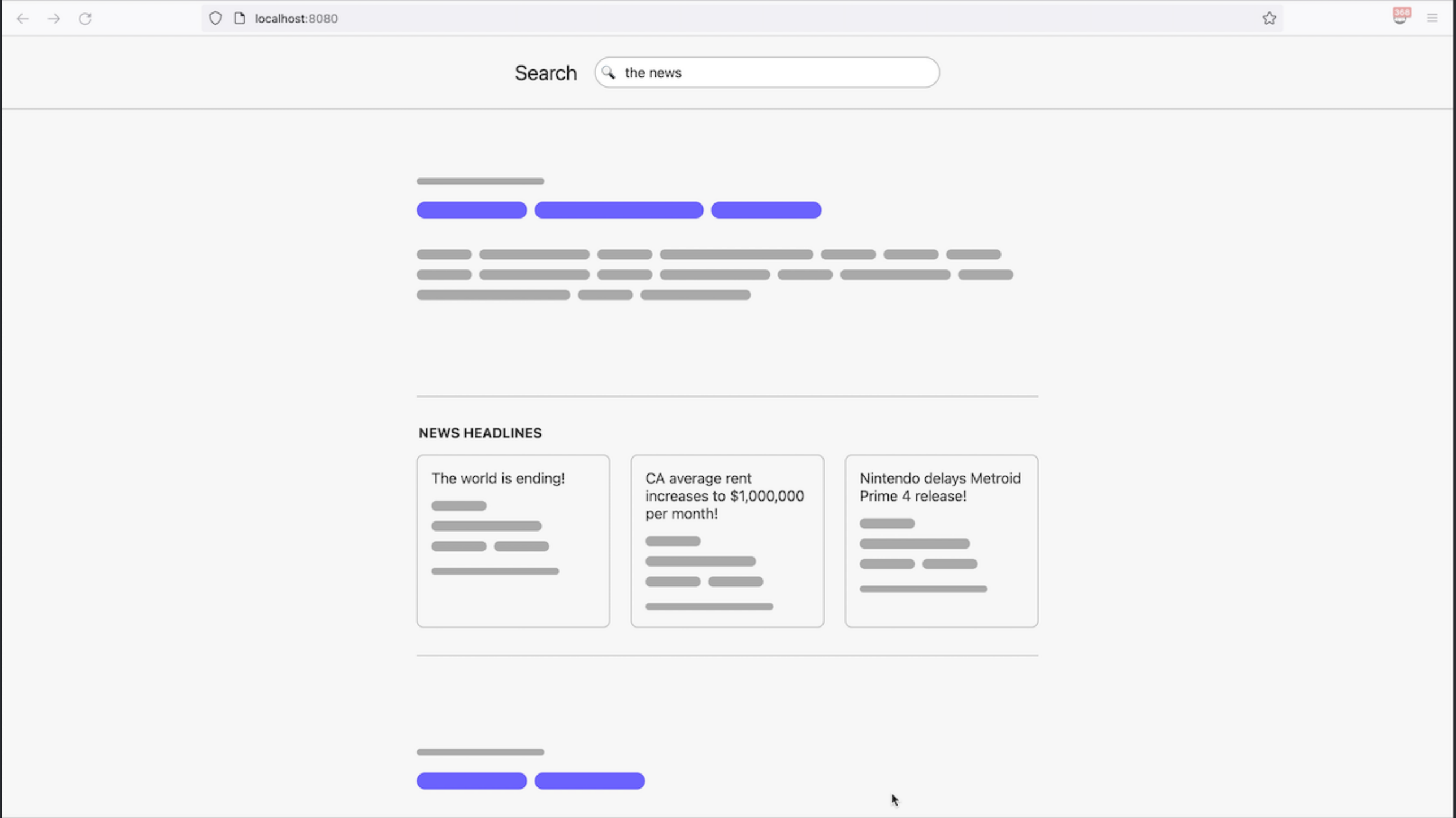

localhost:8080
Search
how to do good rust

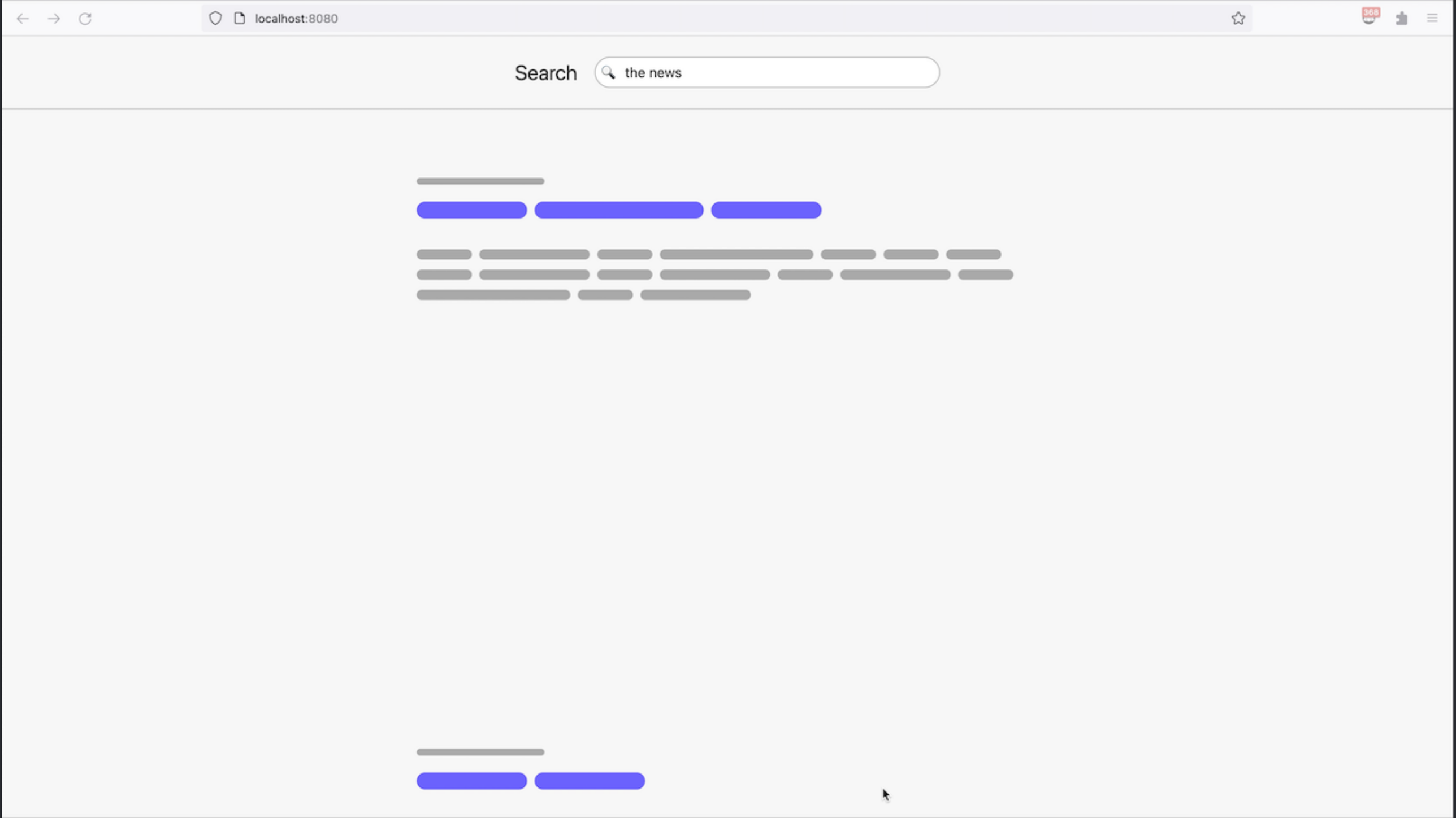
```
const removeContent = () => {
  document
    .querySelectorAll('.distracting-content')
    .forEach((element) => {
      element.style.display = 'none'
    })
}

const observer = new MutationObserver(removeContent)
observer.observe(document.documentElement, config)
```

```
const removeContent = () => {
  document
    .querySelectorAll('.distracting-content')
    .forEach((element) => {
      element.style.display = 'none'
    })
}

const observer = new MutationObserver(removeContent)
observer.observe(document.documentElement, config)
```

```
const removeContent = () => {
  document
    .querySelectorAll('.distracting-content')
    .forEach((element) => {
      element.style.display = 'none'
    })
}

const observer = new MutationObserver(removeContent)
observer.observe(document.documentElement, config)
```

```
const removeContent = () => {
  document
    .querySelectorAll('.distracting-content')
    .forEach((element) => {
      element.style.display = 'none'
    })
}

const observer = new MutationObserver(removeContent)
observer.observe(document.documentElement, config)
```


localhost:8080
Search
the news
NEWS HEADLINES
The world is ending!
CA average rent increases to $1,000,000 per month!
Nintendo delays Metroid Prime 4 release!


localhost:8080
Search
the news


Browser Extension
JS
stdout
stdin
Native Messaging Host


Daemon
IPC
Native Messaging Host

```
enum Outcome {
    CloseBrowser,
    LeaveBrowserOpen,
}

if let Some(browser_pid) = find_browser_pid() {
    let (tx, rx) = mpsc::channel();
```

```
enum Outcome {
    CloseBrowser,
    LeaveBrowserOpen,
}

if let Some(browser_pid) = find_browser_pid() {
    let (tx, rx) = mpsc::channel();

    let communication_tx = tx.clone()
```

```
    CloseBrowser,
    LeaveBrowserOpen,
}

if let Some(browser_pid) = find_browser_pid() {
    let (tx, rx) = mpsc::channel();

    let communication_tx = tx.clone()
    thread::spawn(move || {
```

```
    LeaveBrowserOpen,
}

if let Some(browser_pid) = find_browser_pid() {
    let (tx, rx) = mpsc::channel();

    let communication_tx = tx.clone();
    thread::spawn(move || {
        let mut conn = connect_ipc();
```

```
}

if let Some(browser_pid) = find_browser_pid() {
    let (tx, rx) = mpsc::channel();

    let communication_tx = tx.clone();
    thread::spawn(move || {
        let mut conn = connect_ipc();

        conn.write("ping\n");
```

```rust
if let Some(browser_pid) = find_browser_pid() {
    let (tx, rx) = mpsc::channel();

    let communication_tx = tx.clone();
    thread::spawn(move || {
        let mut conn = connect_ipc();

        conn.write("ping\n");

        let response = conn.read_line();
```

```
let (tx, rx) = mpsc::channel();

let communication_tx = tx.clone();
thread::spawn(move || {
    let mut conn = connect_ipc();

    conn.write("ping\n");

    let response = conn.read_line();

    if response == "pong\n" {
```

```
let communication_tx = tx.clone();
thread::spawn(move || {
    let mut conn = connect_ipc();

    conn.write("ping\n");

    let response = conn.read_line();

    if response == "pong\n" {
        communication_tx.send(LeaveBrowserOpen);
```

```rust
thread::spawn(move || {
    let mut conn = connect_ipc();

    conn.write("ping\n");

    let response = conn.read_line();

    if response == "pong\n" {
        communication_tx.send(LeaveBrowserOpen);
    } else {
```

```
let mut conn = connect_ipc();

conn.write("ping\n");

let response = conn.read_line();

if response == "pong\n" {
    communication_tx.send(LeaveBrowserOpen);
} else {
    communication_tx.send(CloseBrowser);
}
```

```
        conn.write("ping\n");

        let response = conn.read_line();

        if response == "pong\n" {
            communication_tx.send(LeaveBrowserOpen);
        } else {
            communication_tx.send(CloseBrowser);
        }
    });
```

```
        let response = conn.read_line();

        if response == "pong\n" {
            communication_tx.send(LeaveBrowserOpen);
        } else {
            communication_tx.send(CloseBrowser);
        }
    });

    let timeout_tx = tx.clone();
    thread::spawn(move || {
```

```
        if response == "pong\n" {
            communication_tx.send(LeaveBrowserOpen);
        } else {
            communication_tx.send(CloseBrowser);
        }
    });

    let timeout_tx = tx.clone();
    thread::spawn(move || {
        thread::sleep(Duration::from_secs(20));
```

```
            communication_tx.send(LeaveBrowserOpen);
        } else {
            communication_tx.send(CloseBrowser);
        }
    });

    let timeout_tx = tx.clone();
    thread::spawn(move || {
        thread::sleep(Duration::from_secs(20));
```

```
        } else {
            communication_tx.send(CloseBrowser);
        }
    });

    let timeout_tx = tx.clone();
    thread::spawn(move || {
        thread::sleep(Duration::from_secs(20));

        timeout_tx.send(CloseBrowser);
    });
```

```
            communication_tx.send(CloseBrowser);
        }
    });

    let timeout_tx = tx.clone();
    thread::spawn(move || {
        thread::sleep(Duration::from_secs(20));

        timeout_tx.send(CloseBrowser);
    });
```

```
    }
});

let timeout_tx = tx.clone();
thread::spawn(move || {
    thread::sleep(Duration::from_secs(20));

    timeout_tx.send(CloseBrowser);
});

for outcome in rx.iter() {
```

```
});

let timeout_tx = tx.clone();
thread::spawn(move || {
    thread::sleep(Duration::from_secs(20));

    timeout_tx.send(CloseBrowser);
});

for outcome in rx.iter() {
    if let CloseBrowser = should_close {
```

```rust
let timeout_tx = tx.clone();
thread::spawn(move || {
    thread::sleep(Duration::from_secs(20));

    timeout_tx.send(CloseBrowser);
});

for outcome in rx.iter() {
    if let CloseBrowser = should_close {
        quit_process(browser_pid);
    }
```

```rust
thread::spawn(move || {
    thread::sleep(Duration::from_secs(20));

    timeout_tx.send(CloseBrowser);
});

for outcome in rx.iter() {
    if let CloseBrowser = should_close {
        quit_process(browser_pid);
    }
    process::exit(0);
```

```
    thread::sleep(Duration::from_secs(20));

    timeout_tx.send(CloseBrowser);
});

for outcome in rx.iter() {
    if let CloseBrowser = should_close {
        quit_process(browser_pid);
    }
    process::exit(0);
}
```

```
        timeout_tx.send(CloseBrowser);
    });

    for outcome in rx.iter() {
        if let CloseBrowser = should_close {
            quit_process(browser_pid);
        }
        process::exit(0);
    }
}
```

```
    });

    for outcome in rx.iter() {
        if let CloseBrowser = should_close {
            quit_process(browser_pid);
        }
        process::exit(0);
    }
}
```

```
    for outcome in rx.iter() {
        if let CloseBrowser = should_close {
            quit_process(browser_pid);
        }
        process::exit(0);
    }
}
```

```
    for outcome in rx.iter() {
        if let CloseBrowser = should_close {
            quit_process(browser_pid);
        }
        process::exit(0);
    }
}
```


Daemon
IPC
Native Messaging Host

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```

```
let stream = stream_ipc();

for mut conn in stream {
    let request = conn.read_line();

    if request == "ping\n" {
        send_ping_to_browser();

        if check_response_from_browser() {
            conn.write("pong\n");
        }
    }
}
```


Browser Extension
JS
stdout
stdin
Native Messaging Host

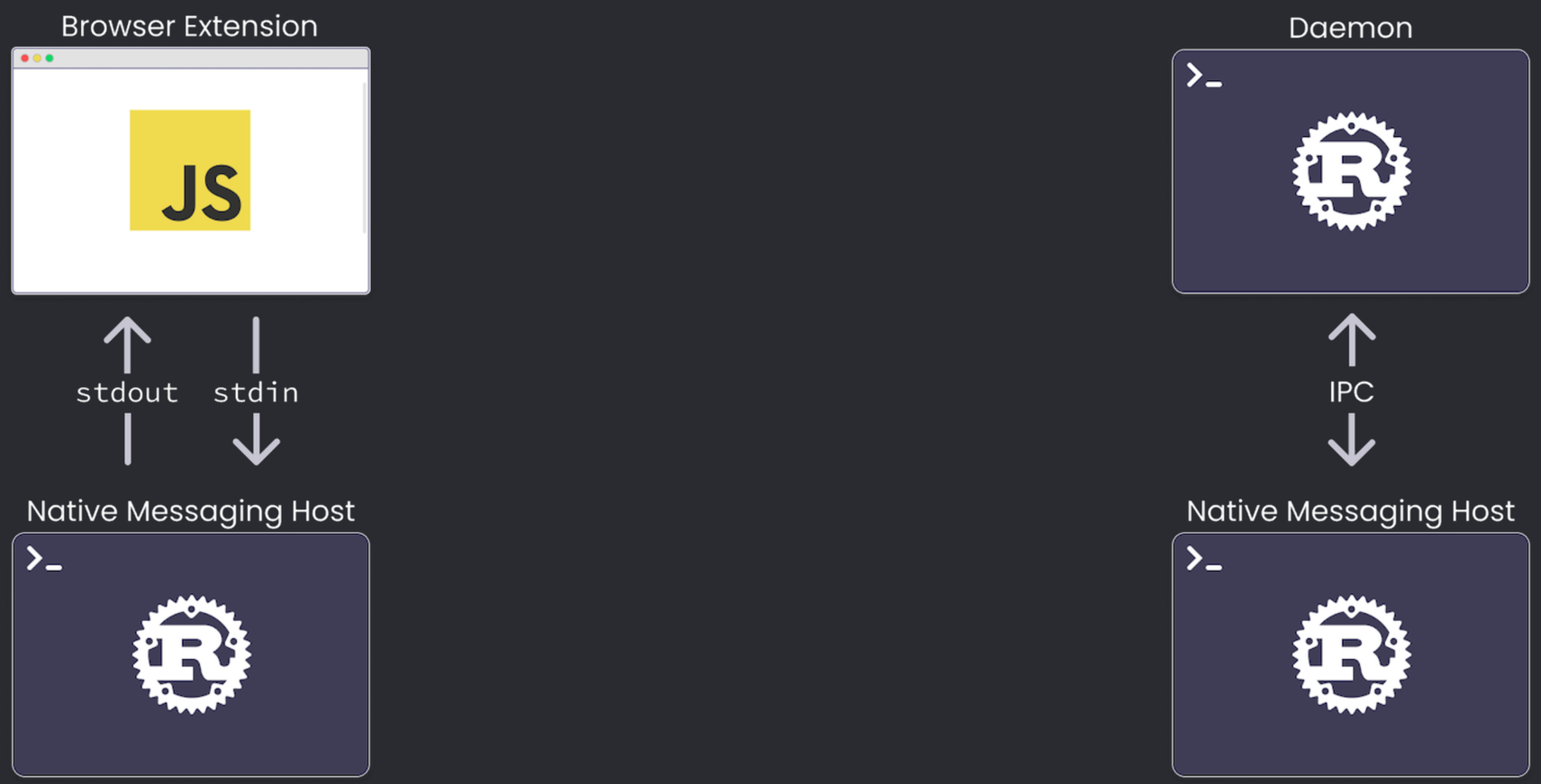
```
const port = connectToHost()

port.onMessage.addListener(async (message, port) => {
  const allowed = await isAllowedInPrivate()

  if (message === 'ping' && allowed) {
      port.postMessage('pong')
  }
})
```

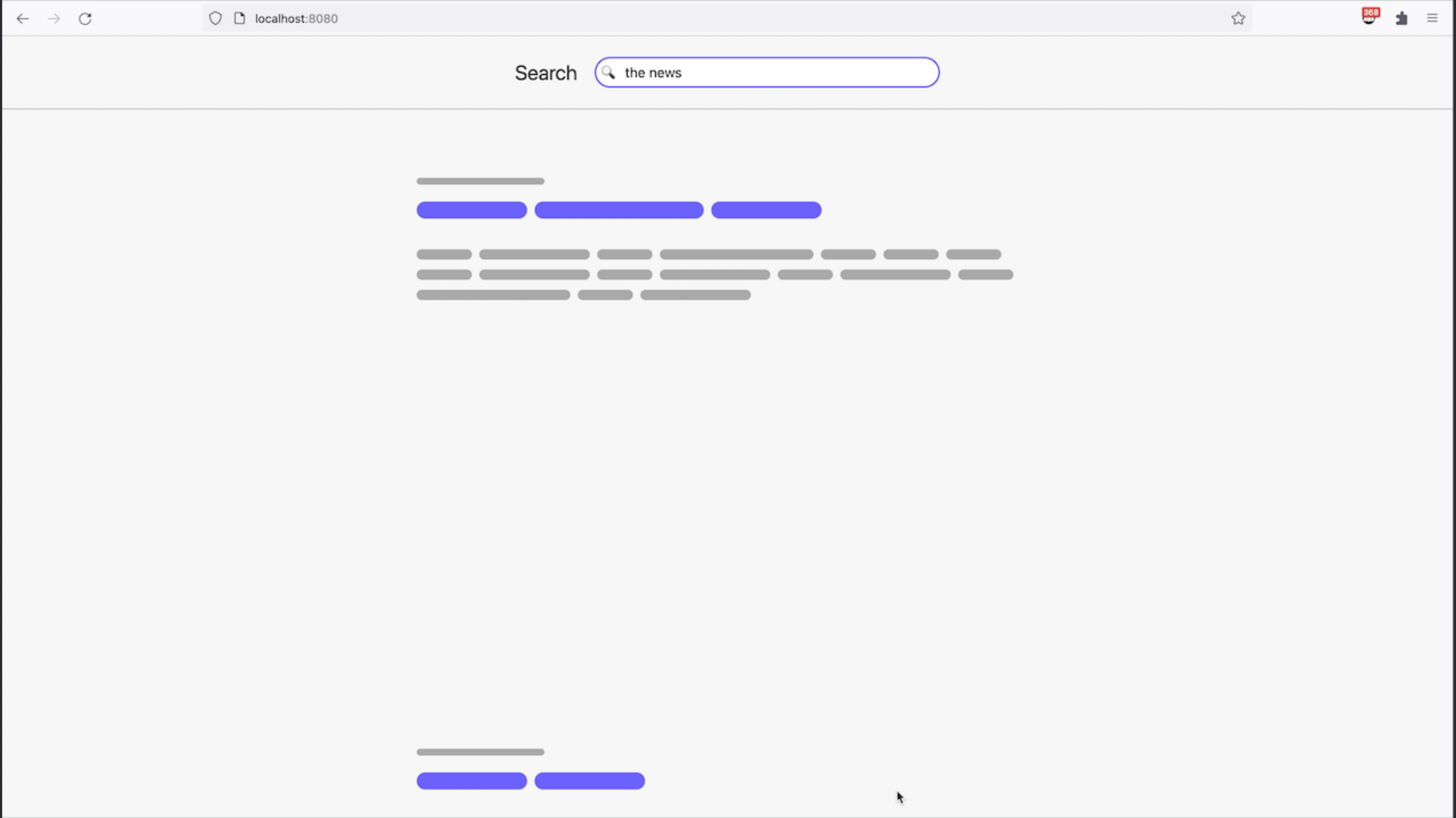
```
const port = connectToHost()

port.onMessage.addListener(async (message, port) => {
  const allowed = await isAllowedInPrivate()

  if (message === 'ping' && allowed) {
      port.postMessage('pong')
  }
})
```

```
const port = connectToHost()

port.onMessage.addListener(async (message, port) => {
  const allowed = await isAllowedInPrivate()

  if (message === 'ping' && allowed) {
      port.postMessage('pong')
  }
})
```

```
const port = connectToHost()

port.onMessage.addListener(async (message, port) => {
  const allowed = await isAllowedInPrivate()

  if (message === 'ping' && allowed) {
      port.postMessage('pong')
  }
})
```

```
const port = connectToHost()

port.onMessage.addListener(async (message, port) => {
  const allowed = await isAllowedInPrivate()

  if (message === 'ping' && allowed) {
      port.postMessage('pong')
  }
})
```


Browser Extension
JS
stdout
stdin
Native Messaging Host
Daemon
IPC
Native Messaging Host


localhost:8080
Search
the news

Crates used

- psutil
- interprocess
- libc
- serde
- byteorder

structionsite.com/careers

STRUCTIONSITE

I'm Luke

- lukewestby@protonmail.com
- github.com/lukewestby
- chess.com/member/lukewestby

Illustrations in this presentation courtesy of undraw.co