Server-Side Swift from Scratch

- Brandon Williams
- @mbrandonw
- mbw234@gmail.com

Server-Side Swift from Scratch

In collaboration with:

- Stephen Celis
- @stephencelis
- stephencelis@gmail.com

POINT•FREE

POINT•FREE

www.pointfree.co

www.github.com/pointfreeco

The layers of a web server framework

Low-level layer

- Socket connections
- HTTP message parsing
- SSL
- Goal is to produce a `URLRequest`
  - URL, e.g. `https://www.pointfree.co/episodes/ep1-hello-world`
  - Method, e.g. GET
  - Post body of type `Data?`
  - Headers, e.g. `Accept-Language: en-US`

High-level layer

- Interprets the `URLRequest`
- Fetches needed data
- Renders a view
- Goal is to produce a `HTTPURLResponse`
  - Status code, e.g. `200 OK`, `302 FOUND`, `404 NOT FOUND`
  - Response headers, e.g. `Content-Type`, `Content-Length`, `Set-Cookie`
  - Response body of type `Data?`

```
(URLRequest) -> URLResponse
```

`(URLRequest) -> URLResponse`

Components

— Middleware
— Routing
— Data fetching
— View rendering

Middleware

- Naive: `(URLRequest) -> HTTPURLResponse`
- Better: `(Conn) -> Conn`
- Even better: `(Conn<A>) -> Conn<B>`
- Great: `(Conn<I, A>) -> Conn<J, B>`

Middleware

- Naive: `(URLRequest) -> HTTPURLResponse`
- Better: `(Conn) -> Conn`
- Even better: `(Conn<A>) -> Conn<B>`
- Great: `(Conn<I, A>) -> Conn<J, B>`

where

```
struct Conn<I, A> {
  let data: A
  let response: HTTPURLResponse
  let request: URLRequest
}
```

Middleware states

```
enum StatusOpen {}

enum HeadersOpen {}

enum BodyOpen {}

enum ResponseEnded {}
```

Middleware states

Status open

```
func writeStatus<A>(_ status: Int)
  -> (Conn<StatusOpen, A>)
  -> Conn<HeadersOpen, A> {
    ...
}
```

Middleware states

Headers open

```
func writeHeader<A>(_ name: String, _ value: String)
  -> (Conn<HeadersOpen, A>)
  -> Conn<HeadersOpen, A> {
    ...
}

func closeHeaders<A>(conn: Conn<HeadersOpen, A>)
  -> Conn<BodyOpen, A> {
    ...
}
```

Middleware states

Body open

```
func send(_ data: Data)
  -> (Conn<BodyOpen, Data>)
  -> Conn<BodyOpen, Data> {
    ...
}

func end<A>(conn: Conn<HeadersOpen, A>)
  -> Conn<ResponseEnded, Data> {
    ...
}
```

Middleware

```
end(
  send(Data("<html>Hello world!</html>".utf8))(
    closeHeaders(
      writeHeader("Content-Type", "text/html")(
        writeHeader("Set-Cookie", "foo=bar")(
          writeStatus(200)(conn)
        )
      )
    )
  )
)
```

```
infix operator >>>

func >>> <A, B, C>(f: (A) -> B, g: (B) -> C) -> (A) -> C {
    return { g(f($0)) }
}
```

```
let siteMiddleware =
  writeStatus(200)
    >>> writeHeader("Set-Cookie", "foo=bar")
    >>> writeHeader("Content-Type", "text/html")
    >>> closeHeaders
    >>> send(Data("<html>Hello world!</html>".utf8))
    >>> end

siteMiddleware(conn)
```

```
Status 200 OK
Content-Type: text/html
Set-Cookie: foo=bar

<html>Hello world!</html>
```

`(URLRequest) -> URLResponse`

Components

- ✓ Middleware
- Routing
- Data fetching
- View rendering

Routing

Routing Goal #1: Type-safety

```
(URLRequest) -> A?
```

Routing Goal #1: Type-safety

A construction is said to be more "type-safe" than some other construction if it can catch errors at compile-time that the other one can only catch at runtime.

Routing Goal #1: Type-safety

Approaches

```
router.get("/episodes/:id") { req in
    let id = req.parameters["id"] ?? ""
    // do something with `id` to produce a response
}
```

Routing Goal #1: Type-safety

Approaches

```
router.get("/episodes/:id") { (request, id: Int) in
  // do something with `id` to produce a response
}
```

Routing Goal #2

Invertible

- `(A) -> URLRequest`
- Useful for linking to parts of the site

Routing Goal #2

Invertible

- `(A) -> URLRequest`
- Useful for linking to parts of the site

```
episode_path(@episode)
# => /episodes/intro-to-functions

episode_path(@episode, ref: "twitter")
# => /episodes/intro-to-functions?ref=twitter
```

Routing Goal #3

Self-documenting

— Given an A, produce documentation

Routing Goal #3

Self-documenting

- Given an A, produce documentation
- `rake routes`

```
GET  /
GET  /episodes
GET  /episodes/:id
GET  /account
POST /account/settings
...
```

Routing: `(URLRequest) -> A?`

Demo

Routing: `(URLRequest) -> A?`

Demo

```
enum Routes {
  // e.g. /
  case root

  // e.g. /episodes?order=asc
  case episodes(order: Order?)

  // e.g. /episodes/intro-to-functions?ref=twitter
  case episode(param: Either<String, Int>, ref: String?)
}

enum Order {
  case asc
  case desc
}
```

Routing: (URLRequest) -> A?

Demo

```
let router = [

  Routes.iso.root
    <¢> get,

  Routes.iso.episodes
    <¢> get %> lit("episodes")
    %> queryParam("order", opt(.order)),

  Routes.iso.episode
    <¢> get %> lit("episodes") %> pathParam(.intOrString)
    <%> queryParam("ref", opt(.string))

  ]
  .reduce(.empty, <|>)
```

Routing: `(URLRequest) -> A?`

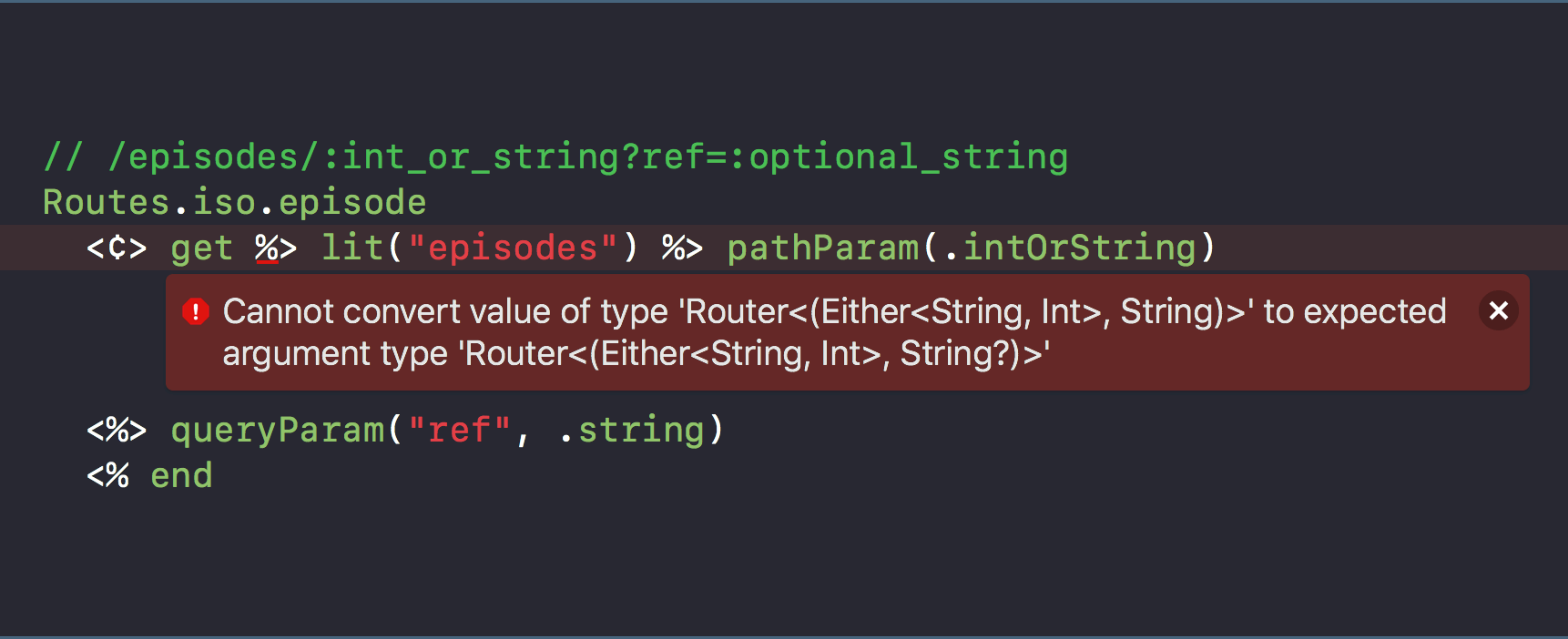
```
switch router.match(request) {
case .some(.root):
  // Homepage

case let .some(.episodes(order)):
  // Episodes page

case let .some(.episode(param, ref)):
  // Episode page

case .none:
  // 404
}
```

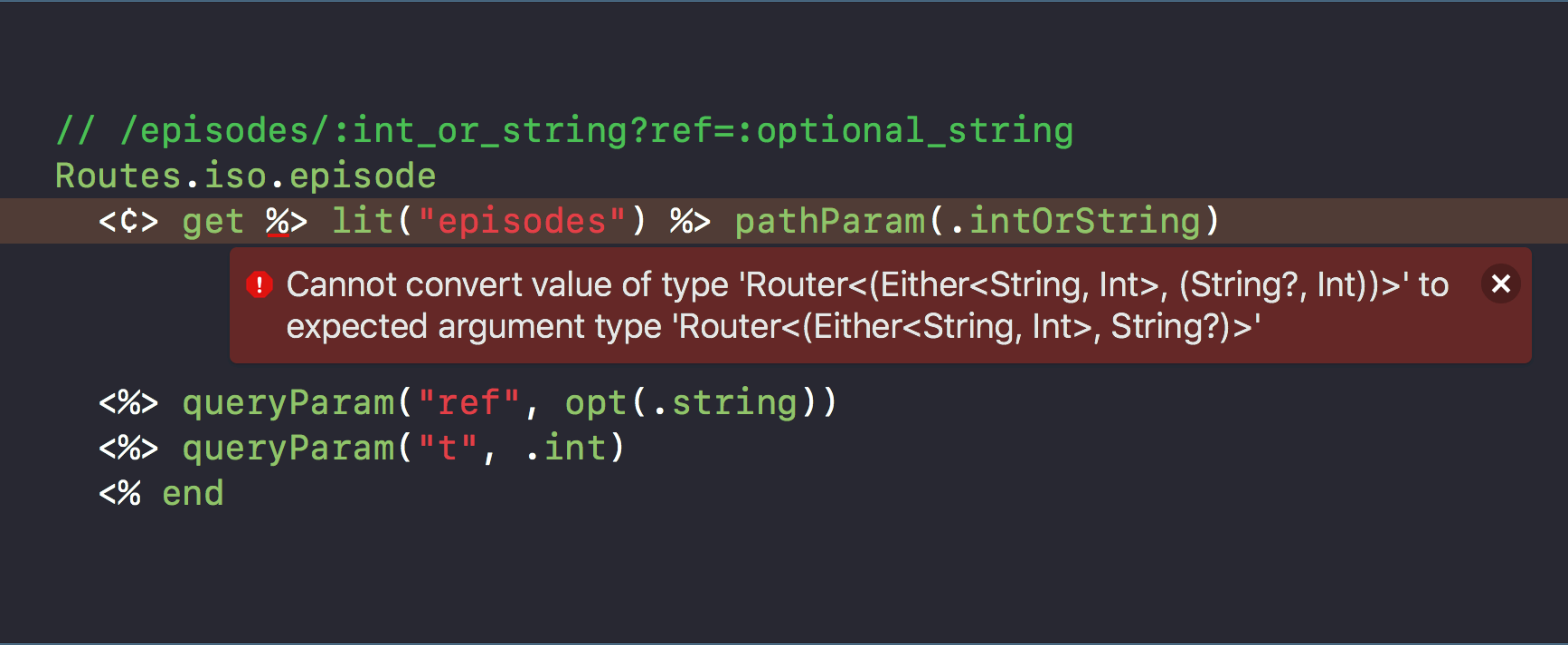
```
// /episodes/:int_or_string?ref=:optional_string
Routes.iso.episode
  <¢> get %> lit("episodes") %> pathParam(.intOrString)
  <%> queryParam("ref", opt(.string))
  <% end
```



```
// /episodes/:int_or_string?ref=:optional_string
Routes.iso.episode
  <¢> get %> lit("episodes") %> pathParam(.intOrString)
```

Cannot convert value of type 'Router<(Either<String, Int>, String)>' to expected argument type 'Router<(Either<String, Int>, String?)>'

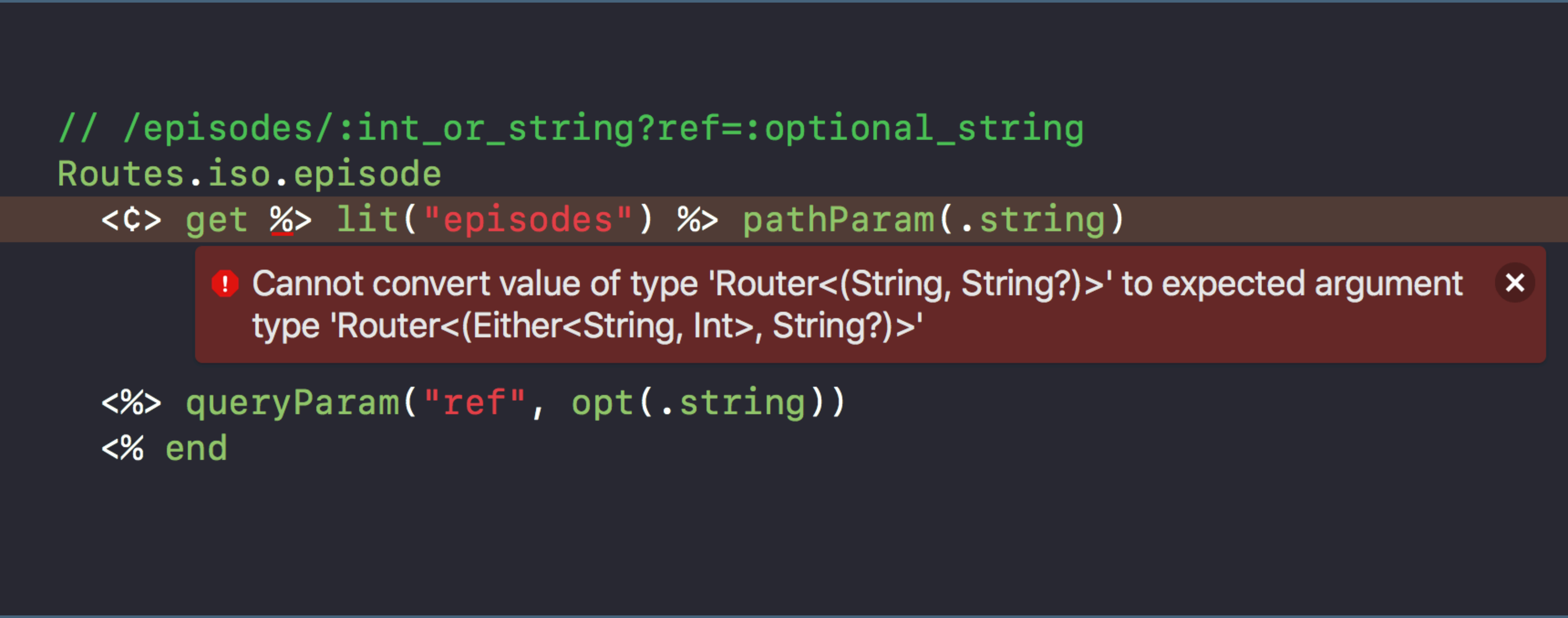
```
  <%> queryParam("ref", .string)
  <% end
```



```
// /episodes/:int_or_string?ref=:optional_string
Routes.iso.episode
  <¢> get %> lit("episodes") %> pathParam(.intOrString)
```

Cannot convert value of type 'Router<(Either<String, Int>, (String?, Int))>' to expected argument type 'Router<(Either<String, Int>, String?)>'

```
  <%> queryParam("ref", opt(.string))
  <%> queryParam("t", .int)
  <% end
```



```
// /episodes/:int_or_string?ref=:optional_string
Routes.iso.episode
  <¢> get %> lit("episodes") %> pathParam(.string)
```

Cannot convert value of type 'Router<(String, String?)>' to expected argument type 'Router<(Either<String, Int>, String?)>'

```
  <%> queryParam("ref", opt(.string))
  <% end
```

```
// /episodes/:int_or_string?ref=:optional_string
Routes.iso.episode
  <¢> get %> lit("episodes") %> pathParam(.intOrString)
  <%> queryParam("ref", opt(.string))
  <% end
```

Routing: `(URLRequest) -> A?`

Linking URL’s for free

```
path(to: .episodes(order: .some(.asc)))
// => "/episodes?order=asc"

path(to: .episode(param: .left("intro-to-functions"), ref: "twitter"))
// => "/episodes/intro-to-functions?ref=twitter"

url(to: .episode(param: .right(42), ref: nil))
// => "https://www.pointfree.co/episodes/42"
```

Routing: `(URLRequest) -> A?`

Template URL's for free

```
template(for: .root)
// => "GET /"

template(for: .episodes(order: nil))
// => "GET /episodes?order=:optional_order"

template(for: .episode(param: .left(""), ref: nil))
// => "GET /episodes/:string_or_int?ref=optional_string"
```

Applicative Parsing

— Namespaces and nesting
`/v1/`

— CRUD Resources
`(POST GET PUT DELETE) /episodes/:id`

— Responsive Route
```
/episodes/1.json
/episodes/1.xml
...
```

— And more...

`(URLRequest) -> URLResponse`

Components

- ✓ Middleware
- ✓ Routing
- Data fetching
- View rendering

Data fetching

Data fetching

@stephencelis
stephencelis@gmail.com

`(URLRequest) -> URLResponse`

Components

- ✓ Middleware
- ✓ Routing
- ✓ Data fetching
- View rendering

View rendering

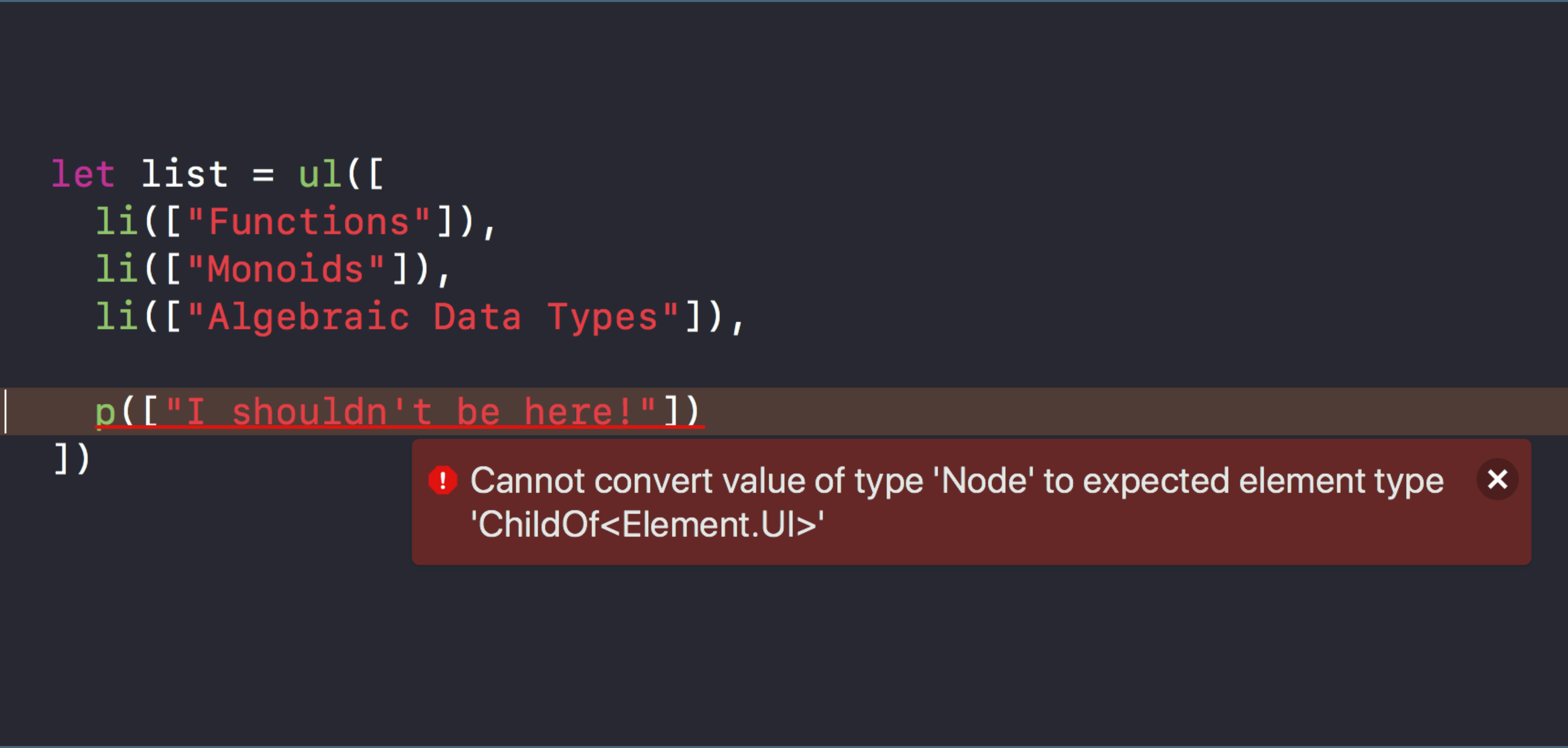
View rendering

```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

```
document([
  html([
    head([
      title("Point-Free")
    ]),
    body([
      h1(["Welcome to Point-Free!"]),
      h2(["Episodes"]),
      ul([
        li(["Pure Functions"]),
        li(["Monoids"]),
        li(["Algebraic Data Types"])
      ])
    ])
  ])
])
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>
      Point-Free
    </title>
  </head>
  <body>
    <h1>Welcome to Point-Free!</h1>
    <h2>Episodes</h2>
    <ul>
      <li>Pure Functions</li>
      <li>Monoids</li>
      <li>Algebraic Data Types</li>
    </ul>
  </body>
</html>
```

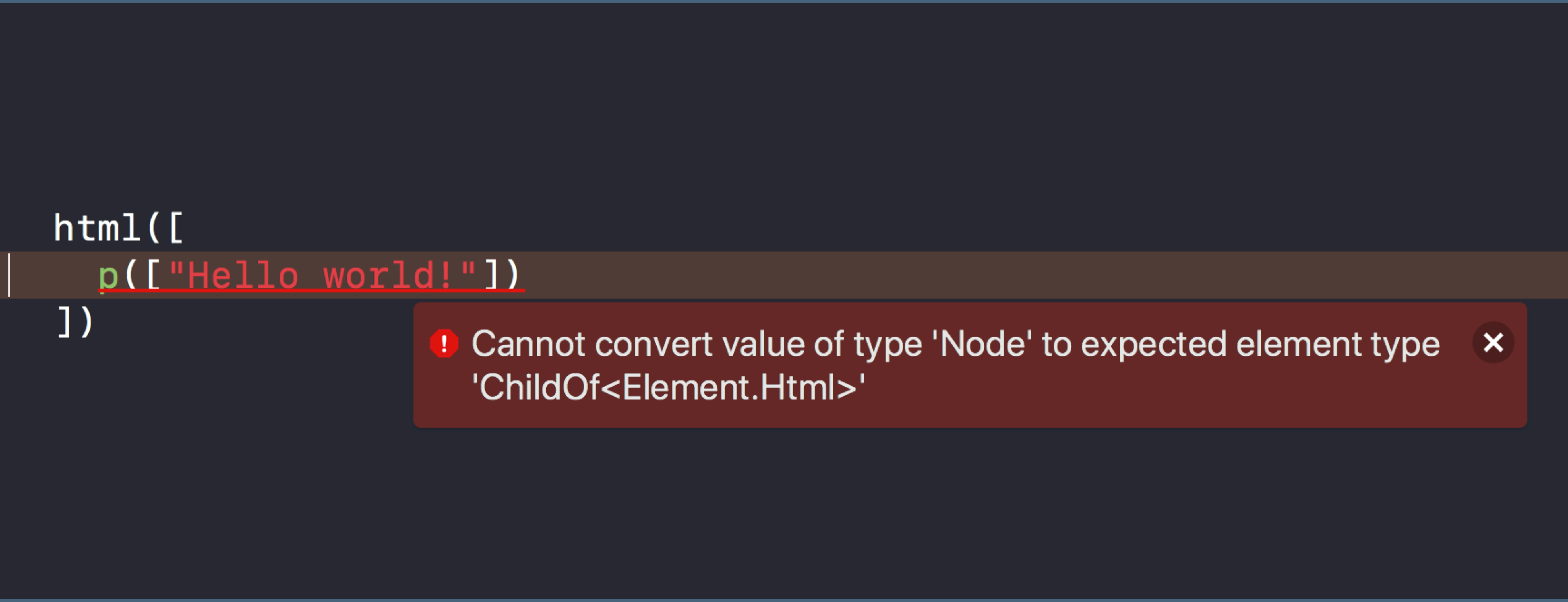
```
let list = ul([
  li(["Functions"]),
  li(["Monoids"]),
  li(["Algebraic Data Types"]),
])
```



```
let list = ul([
  li(["Functions"]),
  li(["Monoids"]),
  li(["Algebraic Data Types"]),

  p(["I shouldn't be here!"])
])
```

Cannot convert value of type 'Node' to expected element type 'ChildOf<Element.Ul>'



```
html([
  p(["Hello world!"])
])
```

Cannot convert value of type 'Node' to expected element type 'ChildOf<Element.Html>'

CSS

CSS

```
let baseFontStyle = fontFamily([
  "-apple-system", "Helvetica Neue", "sans-serif"
])

let baseHeadingStyle =
  baseFontStyle
    <> lineHeight(1.4)
    <> fontSize(.px(22))
```

```swift
let h1Style = h1 % (
  baseHeadingStyle
    <> color(.white(0, 1))
    <> padding(bottom: .px(16))
)

let h2Style = h2 % (
  baseHeadingStyle
    <> color(.white(0.6, 1))
    <> padding(bottom: .px(12))
)
```

```
render(css: h1Style)

h1 {
  font-family    : -apple-system,Helvetica Neue,sans-serif;
  line-height    : 1.4;
  font-size      : 22px;
  color          : #000000;
  padding-bottom : 16px;
}
```

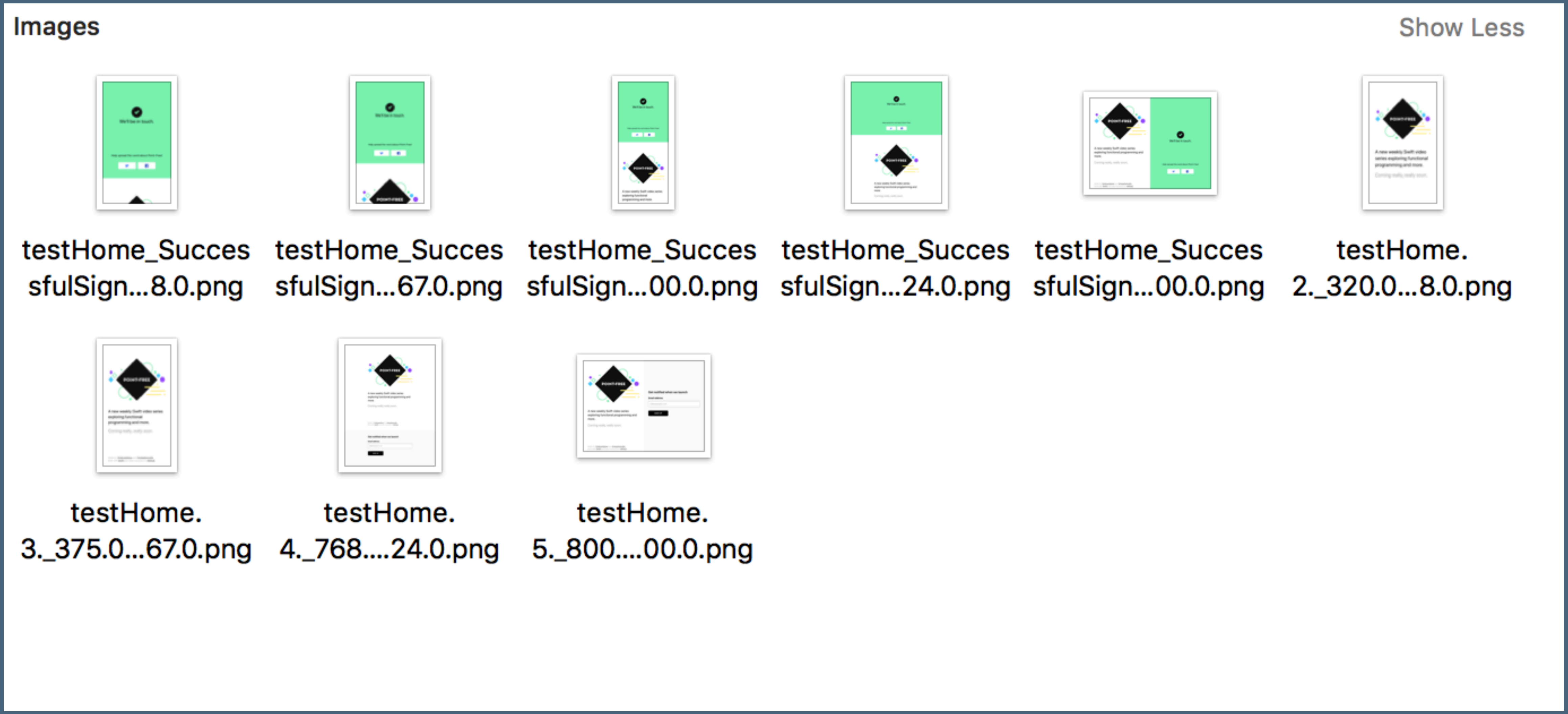
Testing

Snapshot testing

“A snapshot test is a test case that uses reference data—typically a file on disk—to assert the correctness of some code.”

– Stephen Celis
stephencelis.com/2017/09/snapshot-testing-in-swift

```html
<!DOCTYPE html>
<html>
  <head>
    <title>
      Point-Free — A weekly video series on Swift and functional programming.
    </title>
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
  </head>
  <body>
    <header class="hero">
      <div class="container">
        <a href="/">
          <img src="logo.png" alt="Point Free" class="logo">
        </a>
        <h1>A new weekly Swift video series exploring functional programming and more.</h1>
        <h2>Coming really, really soon.</h2>
        <footer>
          <p>
            Made by
            <a href="https://twitter.com/mbrandonw" target="_blank">@mbrandonw</a>
             and
            <a href="https://twitter.com/stephencelis" target="_blank">@stephencelis</a>
            .
          </p>
          <p>
            Built with
            <a href="https://swift.org" target="_blank">Swift</a>
             and open-sourced on
            <a href="https://github.com/pointfreeco/pointfreeco" target="_blank">GitHub</a>
          </p>
        </footer>
      </div>
    </header>
    <section class="signup">
      <form class="container" action="/launch-signup" method="POST">
        <h3>Get notified when we launch</h3>
        <label for="email">Email address</label>
        <input type="email" placeholder="hi@example.com" name="email" id="email">
        <input type="submit" value="Sign up">
      </form>
    </section>
  </body>
</html>
```


Images

Show Less

testHome_SuccessfulSign...8.0.png

testHome_SuccessfulSign...67.0.png

testHome_SuccessfulSign...00.0.png

testHome_SuccessfulSign...24.0.png

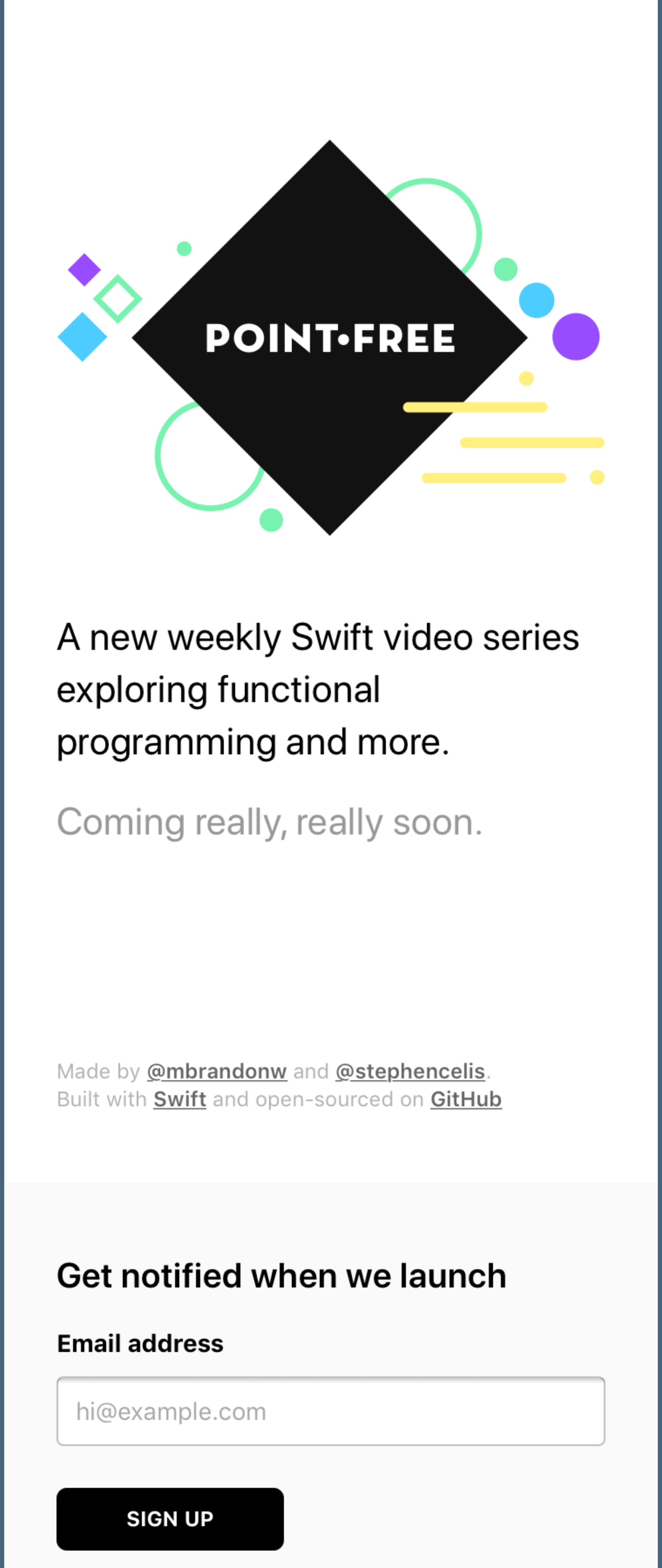
testHome_SuccessfulSign...00.0.png

testHome.2._320.0...8.0.png

testHome.3._375.0...67.0.png

testHome.4._768....24.0.png

testHome.5._800....00.0.png



A new weekly Swift video series exploring functional programming and more.

Coming really, really soon.

Made by @mbrandonw and @stephencelis.
Built with Swift and open-sourced on GitHub

Get notified when we launch

Email address

hi@example.com

SIGN UP

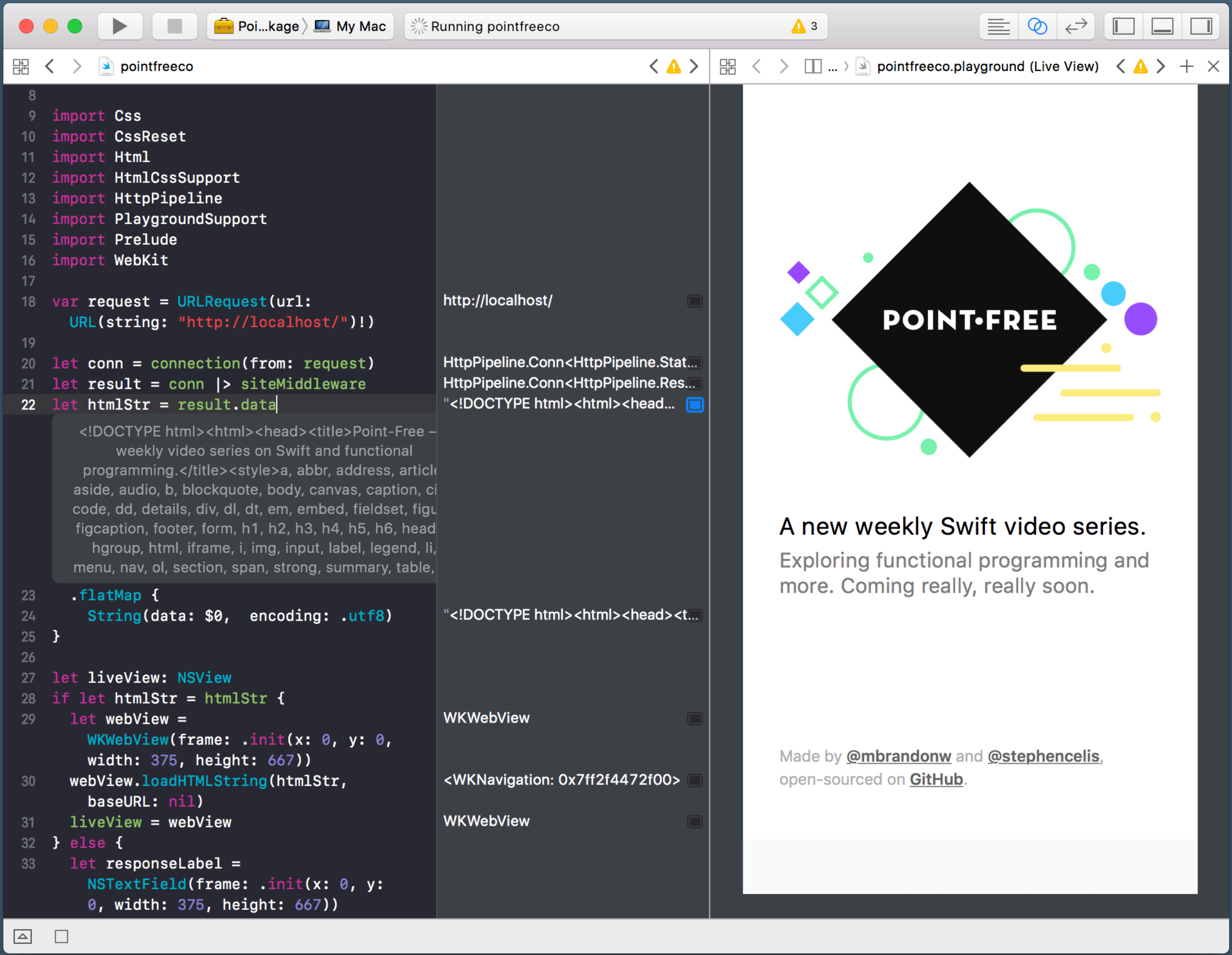
A new weekly Swift video series exploring functional programming and more.

Coming really, really soon.

We'll be in touch.

Help spread the word about Point-Free!

Made by @mbrandonw and @stephencelis.
Built with Swift and open-sourced on GitHub



```swift
import Css
import CssReset
import Html
import HtmlCssSupport
import HttpPipeline
import PlaygroundSupport
import Prelude
import WebKit

var request = URLRequest(url:
  URL(string: "http://localhost/")!)

let conn = connection(from: request)
let result = conn |> siteMiddleware
let htmlStr = result.data
  .flatMap {
    String(data: $0,  encoding: .utf8)
}

let liveView: NSView
if let htmlStr = htmlStr {
  let webView =
    WKWebView(frame: .init(x: 0, y: 0,
    width: 375, height: 667))
  webView.loadHTMLString(htmlStr,
    baseURL: nil)
  liveView = webView
} else {
  let responseLabel =
    NSTextField(frame: .init(x: 0, y:
    0, width: 375, height: 667))
```

```
<!DOCTYPE html><html><head><title>Point-Free –
weekly video series on Swift and functional
programming.</title><style>a, abbr, address, article
aside, audio, b, blockquote, body, canvas, caption, ci
code, dd, details, div, dl, dt, em, embed, fieldset, figu
figcaption, footer, form, h1, h2, h3, h4, h5, h6, head
hgroup, html, iframe, i, img, input, label, legend, li,
menu, nav, ol, section, span, strong, summary, table,
```

```
http://localhost/
HttpPipeline.Conn<HttpPipeline.Stat...
HttpPipeline.Conn<HttpPipeline.Res...
"<!DOCTYPE html><html><head...
"<!DOCTYPE html><html><head><t...
WKWebView
<WKNavigation: 0x7ff2f4472f00>
WKWebView
```

A new weekly Swift video series.

Exploring functional programming and more. Coming really, really soon.

Made by @mbrandonw and @stephencelis, open-sourced on GitHub.

Conclusion

- Take good ideas from existing frameworks, but nothing more
- Leverage Swift's type-system
- Keep as much in Swift as possible
- Look to functional programming
- Focus on small, composable pieces

Thank you

@mbrandonw

POINT•FREE

www.pointfree.co

www.github.com/pointfreeco