The Power of Functional Programming and Static Type Systems in Server-Side Web Applications

Oskar Wickström

https://wickstrom.tech

Kats Conf 2, Dublin, Feb 2017

Elegant Weapons for a More Civilized Page

Me

Me

- I live and work in Malmö, Sweden

Me

- I live and work in Malmö, Sweden
- Started out in music

Me

- I live and work in Malmö, Sweden
- Started out in music
- Found PHP around 2011, then saved by FP soon after

Me

- I live and work in Malmö, Sweden
- Started out in music
- Found PHP around 2011, then saved by FP soon after
- Worked on the Oden language last year

Me

- I live and work in Malmö, Sweden
- Started out in music
- Found PHP around 2011, then saved by FP soon after
- Worked on the Oden language last year
- Currently building CodeScene at work, mostly in Clojure

Me

- I live and work in Malmö, Sweden
- Started out in music
- Found PHP around 2011, then saved by FP soon after
- Worked on the Oden language last year
- Currently building CodeScene at work, mostly in Clojure
- Building Hyper in free time

Agenda

Agenda

- Overview: Functional Programming and Web Applications

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?
- Static Typing for Server-Side Web

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?
- Static Typing for Server-Side Web
- Hyper

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?
- Static Typing for Server-Side Web
- Hyper
    - Design

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?
- Static Typing for Server-Side Web
- Hyper
    - Design
    - Type-Level Routing

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?
- Static Typing for Server-Side Web
- Hyper
    - Design
    - Type-Level Routing
    - XHR Clients

Agenda

- Overview: Functional Programming and Web Applications
- What about Server-Side Rendering?
- Static Typing for Server-Side Web
- Hyper
    - Design
    - Type-Level Routing
    - XHR Clients
    - Future Work

Overview: Functional Programming and Web Applications

Functional Programming Influence

Functional Programming Influence

- FP influences Javascript

Functional Programming Influence

- FP influences Javascript
- ECMAScript 6 has higher-order functions, arrow function syntax

Functional Programming Influence

- FP influences Javascript
- ECMAScript 6 has higher-order functions, arrow function syntax
- Libraries like Underscore, Rambda, Fantasy Land

Functional Programming Influence

- FP influences Javascript
- ECMAScript 6 has higher-order functions, arrow function syntax
- Libraries like Underscore, Rambda, Fantasy Land
- React is functional at its core

Functional Programming Influence

- FP influences Javascript
- ECMAScript 6 has higher-order functions, arrow function syntax
- Libraries like Underscore, Rambda, Fantasy Land
- React is functional at its core
- Functional Reactive Programming (FRP)

Functional Programming Influence

- FP influences Javascript
- ECMAScript 6 has higher-order functions, arrow function syntax
- Libraries like Underscore, Rambda, Fantasy Land
- React is functional at its core
- Functional Reactive Programming (FRP)
- Javascript as a compile target for FP languages

Functional Programming Influence

- FP influences Javascript
- ECMAScript 6 has higher-order functions, arrow function syntax
- Libraries like Underscore, Rambda, Fantasy Land
- React is functional at its core
- Functional Reactive Programming (FRP)
- Javascript as a compile target for FP languages
- Still, main focus is single-page apps

Single-Page Applications

Single-Page Applications

- Work more like desktop applications

Single-Page Applications

- Work more like desktop applications
- There are **a lot** of these frameworks in JS

Single-Page Applications

- Work more like desktop applications
- There are **a lot** of these frameworks in JS
- Angular, Ember, Meteor, React (with friends)

Single-Page Applications

- Work more like desktop applications
- There are **a lot** of these frameworks in JS
- Angular, Ember, Meteor, React (with friends)
- Without Javascript, you get nothing

Single-Page Applications

- Work more like desktop applications
- There are **a lot** of these frameworks in JS
- Angular, Ember, Meteor, React (with friends)
- Without Javascript, you get nothing
- Reinventing the browser

What about Server-Side Rendering?

Progressive Enhancement

80/20

“Isomorphic” Web Applications

“Isomorphic” Web Applications

- Goal: one framework that runs on both client and server

“Isomorphic” Web Applications

- Goal: one framework that runs on both client and server
- “Free progressive enhancements”

“Isomorphic” Web Applications

- Goal: one framework that runs on both client and server
- “Free progressive enhancements”
- The client and server state machines

“Isomorphic” Web Applications

- Goal: one framework that runs on both client and server
- “Free progressive enhancements”
- The client and server state machines
- Are we really talking about initial rendering?

PJAX

PJAX

- Hooks in on link and form events

PJAX

- Hooks in on link and form events
- Requests pages over XHR with special header

PJAX

- Hooks in on link and form events
- Requests pages over XHR with special header
- Server responds with only inner content

PJAX

- Hooks in on link and form events
- Requests pages over XHR with special header
- Server responds with only inner content
- PJAX swaps the inner content on the client

If server-side web has tooling problems, let’s build nice tools!

Static Typing for Server-Side Web

Static Typing for Server-Side Web

Static Typing for Server-Side Web

- Mainstream languages in web server programming

Static Typing for Server-Side Web

- Mainstream languages in web server programming
- Compile-time guarantees

Static Typing for Server-Side Web

- Mainstream languages in web server programming
- Compile-time guarantees
- Safely abstract and compose

Static Typing for Server-Side Web

- Mainstream languages in web server programming
- Compile-time guarantees
- Safely abstract and compose
- Maintainable code

Static Typing for Server-Side Web

- Mainstream languages in web server programming
- Compile-time guarantees
- Safely abstract and compose
- Maintainable code
- Developer experience

Things I’ve Found

Things I’ve Found

- Haskell:

Things I’ve Found

- Haskell:
    - Yesod

Things I’ve Found

- Haskell:
    - Yesod
    - Servant

Things I’ve Found

- Haskell:
    - Yesod
    - Servant
- Scala:

Things I’ve Found

- Haskell:
    - Yesod
    - Servant
- Scala:
    - Play

Things I've Found

- Haskell:
    - Yesod
    - Servant
- Scala:
    - Play
    - Rho

Things I've Found

- Haskell:
    - Yesod
    - Servant
- Scala:
    - Play
    - Rho
- PureScript:

Things I've Found

- Haskell:
    - Yesod
    - Servant
- Scala:
    - Play
    - Rho
- PureScript:
    - purescript-express

Things I’ve Found

- Haskell:
    - Yesod
    - Servant
- Scala:
    - Play
    - Rho
- PureScript:
    - purescript-express
    - purescript-rest

Statically Typed Middleware

Statically Typed Middleware

- Middleware is a common abstraction

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
    - Incorrect ordering

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
    - Incorrect ordering
    - Corrupt or incomplete responses

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
    - Incorrect ordering
    - Corrupt or incomplete responses
    - Conflicting writes

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
    - Incorrect ordering
    - Corrupt or incomplete responses
    - Conflicting writes
    - Incorrect error handling

Statically Typed Middleware

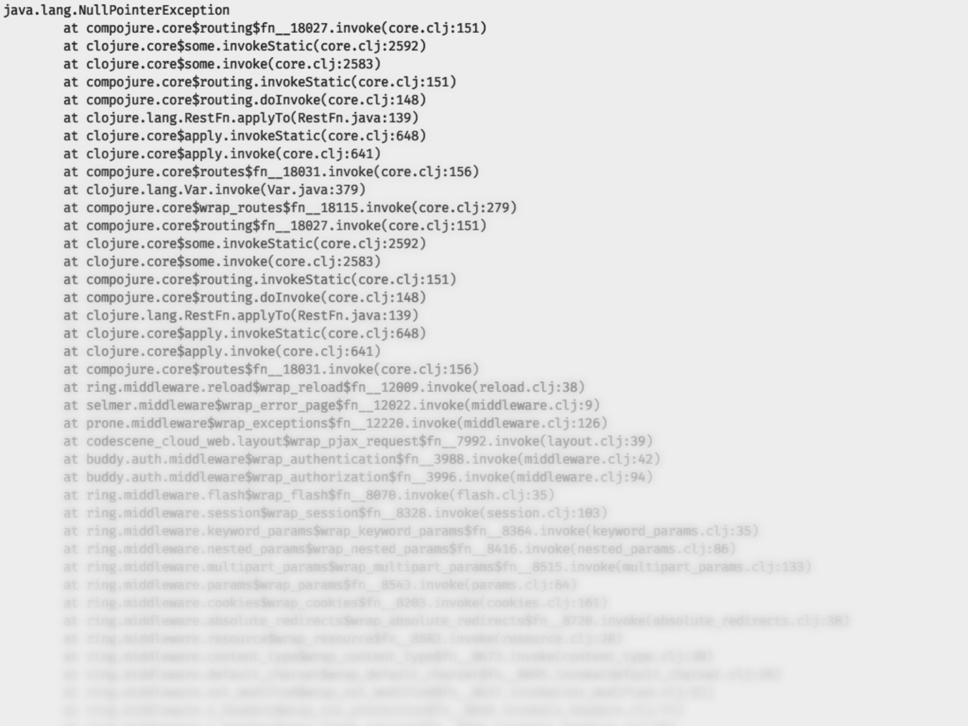
- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
    - Incorrect ordering
    - Corrupt or incomplete responses
    - Conflicting writes
    - Incorrect error handling
    - Consuming non-parsed, or badly parsed, request body

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
    - Incorrect ordering
    - Corrupt or incomplete responses
    - Conflicting writes
    - Incorrect error handling
    - Consuming non-parsed, or badly parsed, request body
    - Missing authentication and/or authorization

Statically Typed Middleware

- Middleware is a common abstraction
- Very easy to mess up if dynamically typed
  - Incorrect ordering
  - Corrupt or incomplete responses
  - Conflicting writes
  - Incorrect error handling
  - Consuming non-parsed, or badly parsed, request body
  - Missing authentication and/or authorization
- Idea: use extensible records in PureScript!



```
java.lang.NullPointerException
        at compojure.core$routing$fn__18027.invoke(core.clj:151)
        at clojure.core$some.invokeStatic(core.clj:2592)
        at clojure.core$some.invoke(core.clj:2583)
        at compojure.core$routing.invokeStatic(core.clj:151)
        at compojure.core$routing.doInvoke(core.clj:148)
        at clojure.lang.RestFn.applyTo(RestFn.java:139)
        at clojure.core$apply.invokeStatic(core.clj:648)
        at clojure.core$apply.invoke(core.clj:641)
        at compojure.core$routes$fn__18031.invoke(core.clj:156)
        at clojure.lang.Var.invoke(Var.java:379)
        at compojure.core$wrap_routes$fn__18115.invoke(core.clj:279)
        at compojure.core$routing$fn__18027.invoke(core.clj:151)
        at clojure.core$some.invokeStatic(core.clj:2592)
        at clojure.core$some.invoke(core.clj:2583)
        at compojure.core$routing.invokeStatic(core.clj:151)
        at compojure.core$routing.doInvoke(core.clj:148)
        at clojure.lang.RestFn.applyTo(RestFn.java:139)
        at clojure.core$apply.invokeStatic(core.clj:648)
        at clojure.core$apply.invoke(core.clj:641)
        at compojure.core$routes$fn__18031.invoke(core.clj:156)
        at ring.middleware.reload$wrap_reload$fn__12009.invoke(reload.clj:38)
        at selmer.middleware$wrap_error_page$fn__12022.invoke(middleware.clj:9)
        at prone.middleware$wrap_exceptions$fn__12220.invoke(middleware.clj:126)
        at codescene_cloud_web.layout$wrap_pjax_request$fn__7992.invoke(layout.clj:39)
        at buddy.auth.middleware$wrap_authentication$fn__3988.invoke(middleware.clj:42)
        at buddy.auth.middleware$wrap_authorization$fn__3996.invoke(middleware.clj:94)
        at ring.middleware.flash$wrap_flash$fn__8070.invoke(flash.clj:35)
        at ring.middleware.session$wrap_session$fn__8328.invoke(session.clj:103)
        at ring.middleware.keyword_params$wrap_keyword_params$fn__8364.invoke(keyword_params.clj:35)
        at ring.middleware.nested_params$wrap_nested_params$fn__8416.invoke(nested_params.clj:86)
        at ring.middleware.multipart_params$wrap_multipart_params$fn__8515.invoke(multipart_params.clj:133)
        at ring.middleware.params$wrap_params$fn__8543.invoke(params.clj:64)
        at ring.middleware.cookies$wrap_cookies$fn__8203.invoke(cookies.clj:161)
        at ring.middleware.absolute_redirects$wrap_absolute_redirects$fn__8720.invoke(absolute_redirects.clj:38)
        at ring.middleware.resource$wrap_resource$fn__8583.invoke(resource.clj:28)
        at ring.middleware.content_type$wrap_content_type$fn__8673.invoke(content_type.clj:30)
        at ring.middleware.default_charset$wrap_default_charset$fn__[illegible].invoke(default_charset.clj:[illegible])
        [illegible]
        [illegible]
```

Let’s use extensible records in PureScript!

!!YPER

Type-safe, statically checked composition of HTTP servers

Goals

Goals

- A safe HTTP middleware architecture

Goals

- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types

Goals

- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition of middleware and application components

Goals

- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition of middleware and application components
- Interoperability with NodeJS and other backends (purerl, purescript-native)

Goals

- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition of middleware and application components
- Interoperability with NodeJS and other backends (purerl, purescript-native)
- No magic

How?

How?

- Track middleware effects in type system, pure transformations and side effects

How?

- Track middleware effects in type system, pure transformations and side effects
- Leverage extensible records in PureScript

How?

- Track middleware effects in type system, pure transformations and side effects
- Leverage extensible records in PureScript
- Provide a common API for middleware

How?

- Track middleware effects in type system, pure transformations and side effects
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware where possible

How?

- Track middleware effects in type system, pure transformations and side effects
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware where possible
- Integrate with existing NodeJS libraries

Design

Conn

```
type Conn req res components =
  { request :: req
  , response :: res
  , components :: components
  }
```

Middleware (Old Design)

```
type Middleware m c c' = c -> m c'
```

Middleware (Old Design)

```
authenticateUser >=> parseForm >=> saveTodo
```

Whoops, Not Safe

```
badMiddleware conn = do
  _ <- respond "First response" conn
  respond "Second response, crash!" conn
```

Middleware (Revised)

```
newtype Middleware m i o a =
  Middleware (i -> m (Tuple a o))
```

Response State Transitions

Response State Transitions

- Hyper tracks the state of response writing

Response State Transitions

- Hyper tracks the state of response writing
- Guarantees correctness in response side effects

Response State Transitions

- Hyper tracks the state of response writing
- Guarantees correctness in response side effects
- Abstractions can be built on top safely

Response State Transitions

- Hyper tracks the state of response writing
- Guarantees correctness in response side effects
- Abstractions can be built on top safely
- Response-writing middleware can be backend-agnostic

ResponseStateTransition

```
type ResponseStateTransition m rw from to =
```

ResponseStateTransition

```
type ResponseStateTransition m rw from to =
  forall req res c.
```

ResponseStateTransition

```
type ResponseStateTransition m rw from to =
  forall req res c.
  Middleware
  m
```

ResponseStateTransition

```
type ResponseStateTransition m rw from to =
  forall req res c.
  Middleware
  m
  (Conn req {writer :: rw from | res} c)
```

ResponseStateTransition

```
type ResponseStateTransition m rw from to =
  forall req res c.
  Middleware
  m
  (Conn req {writer :: rw from | res} c)
  (Conn req {writer :: rw to | res} c)
```

ResponseStateTransition

```
type ResponseStateTransition m rw from to =
  forall req res c.
  Middleware
  m
  (Conn req {writer :: rw from | res} c)
  (Conn req {writer :: rw to | res} c)
  Unit
```

ResponseWriter

```haskell
class ResponseWriter rw m b | rw -> b where
```

ResponseWriter

```
class ResponseWriter rw m b | rw -> b where
  writeStatus
    :: Status
    -> ResponseStateTransition m rw StatusLineOpen HeadersOpen
```

ResponseWriter

```
class ResponseWriter rw m b | rw -> b where
  writeStatus
    :: Status
    -> ResponseStateTransition m rw StatusLineOpen HeadersOpen

  writeHeader
    :: Header
    -> ResponseStateTransition m rw HeadersOpen HeadersOpen
```

ResponseWriter

```
class ResponseWriter rw m b | rw -> b where
  writeStatus
    :: Status
    -> ResponseStateTransition m rw StatusLineOpen HeadersOpen

  writeHeader
    :: Header
    -> ResponseStateTransition m rw HeadersOpen HeadersOpen

  closeHeaders
    :: ResponseStateTransition m rw HeadersOpen BodyOpen
```

ResponseWriter

```
class ResponseWriter rw m b | rw -> b where
  writeStatus
    :: Status
    -> ResponseStateTransition m rw StatusLineOpen HeadersOpen

  writeHeader
    :: Header
    -> ResponseStateTransition m rw HeadersOpen HeadersOpen

  closeHeaders
    :: ResponseStateTransition m rw HeadersOpen BodyOpen

  send
    :: b
    -> ResponseStateTransition m rw BodyOpen BodyOpen
```

ResponseWriter

```
class ResponseWriter rw m b | rw -> b where
  writeStatus
    :: Status
    -> ResponseStateTransition m rw StatusLineOpen HeadersOpen

  writeHeader
    :: Header
    -> ResponseStateTransition m rw HeadersOpen HeadersOpen

  closeHeaders
    :: ResponseStateTransition m rw HeadersOpen BodyOpen

  send
    :: b
    -> ResponseStateTransition m rw BodyOpen BodyOpen

  end
    :: ResponseStateTransition m rw BodyOpen ResponseEnded
```

What if we do it wrong?

```
          v
20        writeStatus statusOK
21        :*> respond "Hello, Hyper!"
22        :*> closeHeaders
                         ^

Could not match type

  BodyOpen

with type

  HeadersOpen
```

Writing Headers

```
headers
```

Writing Headers

```
headers
  :: forall t m req res rw b c.
     (Traversable t, Monad m, ResponseWriter rw m b) =>
     t Header
```

Writing Headers

```
headers
  :: forall t m req res rw b c.
     (Traversable t, Monad m, ResponseWriter rw m b) =>
     t Header
  -> Middleware
     m
     (Conn req { writer :: rw HeadersOpen | res } c)
     (Conn req { writer :: rw BodyOpen | res } c)
     Unit
```

Writing Headers

```
headers
  :: forall t m req res rw b c.
     (Traversable t, Monad m, ResponseWriter rw m b) =>
     t Header
  -> Middleware
     m
     (Conn req { writer :: rw HeadersOpen | res } c)
     (Conn req { writer :: rw BodyOpen | res } c)
     Unit
headers hs =
  traverse_ writeHeader hs
  :*> closeHeaders
```

Type-Level Routing

A Routing Type

```
data Home = Home

type Site1 = Get HTML Home
```

Handler

```
home :: forall m. Monad m
     => ExceptT RoutingError m Home
home = pure Home
```

HTMLEncode

```
instance encodeHTMLHome :: EncodeHTML Home where
  encodeHTML Home =
    p [] [ text "Welcome to my site!" ]
```

Proxy

The Proxy type and values are for situations where type information is required for an input to determine the type of an output, but where it is not possible or convenient to provide a value for the input.

Site Proxy

```
site1 :: Proxy Site1
site1 = Proxy
```

Site Router

```
onRoutingError status msg =
  writeStatus status
  :*> contentType textHTML
  :*> closeHeaders
  :*> respond (maybe "" id msg)

site1Router = router site1 home onRoutingError
```

Main Entrypoint

```
main =
  runServer defaultOptions
            onListening
            onRequestError
            {}
            site1Router
  where
    onListening (Port port) =
      log ("Listening on http://localhost:" <> show port)

    onRequestError err =
      log ("Request failed: " <> show err)
```

More Routes!

Multiple Endpoints with Captures

```
data Home = Home

data AllUsers = AllUsers (Array User)

newtype User = User { id :: Int, name :: String }

type Site2 =
  Get HTML Home
  :<|> "users" :/ Get HTML AllUsers
  :<|> "users" :/ Capture "user-id" Int :> Get HTML User
```

Multiple Handlers

```
home :: forall m. Monad m => ExceptT RoutingError m Home
home = pure Home
```

Multiple Handlers

```
home :: forall m. Monad m => ExceptT RoutingError m Home
home = pure Home

allUsers :: forall m. Monad m => ExceptT RoutingError m AllUsers
allUsers = AllUsers <$> getUsers
```

Multiple Handlers

```
home :: forall m. Monad m => ExceptT RoutingError m Home
home = pure Home

allUsers :: forall m. Monad m => ExceptT RoutingError m AllUsers
allUsers = AllUsers <$> getUsers

getUser :: forall m. Monad m => Int -> ExceptT RoutingError m User
getUser id’ =
  find userWithId <$> getUsers >>=
  case _ of
    Just user -> pure user
    Nothing ->
      throwError (HTTPError { status: statusNotFound
                            , message: Just "User not found."
                            })
  where
    userWithId (User u) = u.id == id’
```

Type-Safe Links

```
instance encodeHTMLAllUsers :: EncodeHTML AllUsers where
  encodeHTML (AllUsers users) =
```

Type-Safe Links

```
instance encodeHTMLAllUsers :: EncodeHTML AllUsers where
  encodeHTML (AllUsers users) =
    element_ "div" [ h1 [] [ text "Users" ]
                   , ul [] (map linkToUser users)
                   ]
```

Type-Safe Links

```
instance encodeHTMLAllUsers :: EncodeHTML AllUsers where
  encodeHTML (AllUsers users) =
    element_ "div" [ h1 [] [ text "Users" ]
                   , ul [] (map linkToUser users)
                   ]
    where
      linkToUser (User u) =
        case linksTo site2 of
          _ :<|> _ :<|> getUser’ ->
            li [] [ linkTo (getUser’ u.id) [ text u.name ] ]
```

Multiple Endpoint Router

```
site2Router =
  router site2 (home :<|> allUsers :<|> getUser) onRoutingError
```

Automatically Derived XHR Clients

Shared Routing Type

```
type TaskId = Int

data Task = Task TaskId String
```

Shared Routing Type

```
derive instance genericTask :: Generic Task

instance showTask :: Show Task where
  show = gShow

instance encodeJsonTask :: EncodeJson Task where
  encodeJson = gEncodeJson

instance decodeJsonTask :: DecodeJson Task where
  decodeJson = gDecodeJson
```

Shared Routing Type

```
type Site =
  "tasks" :/ (Get Json (Array Task)
              :<|> Capture "id" TaskId :> Get Json Task)
```

Shared Routing Type

```
type Site =
  "tasks" :/ (Get Json (Array Task)
             :<|> Capture "id" TaskId :> Get Json Task)

site :: Proxy Site
site = Proxy
```

Shared Routing Type

```
update :: Action
       -> State
       -> EffModel State Action (ajax :: AJAX)
update RequestTasks state =
  case asClients site of
```

Shared Routing Type

```
update :: Action
       -> State
       -> EffModel State Action (ajax :: AJAX)
update RequestTasks state =
  case asClients site of
    allTasks :<|> _ ->
```

Shared Routing Type

```
update :: Action
       -> State
       -> EffModel State Action (ajax :: AJAX)
update RequestTasks state =
  case asClients site of
    allTasks :<|> _ ->
      { state: state { status = "Fetching tasks..." }
      , effects: [ ReceiveTasks <$> allTasks ]
      }
```

Other Possibilites/Future Work

Other Possibilites/Future Work

- Type-safe forms

Other Possibilites/Future Work

- Type-safe forms
- PJAX, but with JSON data and client-side templates

Other Possibilites/Future Work

- Type-safe forms
- PJAX, but with JSON data and client-side templates
- Mocked servers and clients using `Arbitrary` instances

Other Possibilites/Future Work

- Type-safe forms
- PJAX, but with JSON data and client-side templates
- Mocked servers and clients using `Arbitrary` instances
- Ring-like response map abstraction

Other Possibilites/Future Work

- Type-safe forms
- PJAX, but with JSON data and client-side templates
- Mocked servers and clients using `Arbitrary` instances
- Ring-like response map abstraction
- Other backends

Other Possibilites/Future Work

- Type-safe forms
- PJAX, but with JSON data and client-side templates
- Mocked servers and clients using `Arbitrary` instances
- Ring-like response map abstraction
- Other backends
- Continue to Quest For Type-Safe Web!

Summary

Thank You!

Useful References I

- Gustaf Nilsson Kotte. *6 Reasons Isomorphic Web Apps is not the Silver Bullet You're Looking For.* URL: https://blog.jayway.com/2016/05/23/6-reasons-isomorphic-web-apps-not-silver-bullet-youre-looking/.
- *PJAX = pushState + ajax.* URL: https://github.com/defunkt/jquery-pjax.
- *Yesod: Web Framework for Haskell.* URL: http://www.yesodweb.com/.
- *Servant: A Type-Level Web DSL.* URL: https://haskell-servant.github.io/.

Useful References II

- *Play: The High Velocity Web Framework For Java and Scala.* URL: https://www.playframework.com/.
- *Rho: A DSL for building HTTP services with http4s.* URL: https://github.com/http4s/rho.
- *purescript-express: Purescript wrapper around Node.js Express web-framework.* URL: https://github.com/dancingrobot84/purescript-express.
- *purescript-rest: A toolkit for creating REST services with Node and PureScript.* URL: https://github.com/dicomgrid/purescript-rest.

Useful References III

- *Hyper: Type-safe, statically checked composition of HTTP servers.*

  URL: https://owickstrom.github.io/hyper/.

- *purescript-proxy: Value proxy for type inputs.* URL:

  https://pursuit.purescript.org/packages/purescript-proxy/1.0.0.

- *Ring: Clojure HTTP server abstraction.* URL:

  https://github.com/ring-clojure.

- *Automatically derived XHR clients for Hyper routing types.* URL:

  https://github.com/owickstrom/purescript-hyper-routing-xhr.

The Power of Functional Programming and Static Type Systems in Server-Side Web Applications

Oskar Wickström

https://wickstrom.tech

Kats Conf 2, Dublin, Feb 2017