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

Oskar Wickström

https://wickstrom.tech

flatMap(Oslo), May 2, 2017

https://xkcd.com/297/

Me

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

Agenda

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

Overview: Functional Programming and Web Applications

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

- 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/Universal Web Applications

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

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

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

Things I've Found

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

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

```
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__8695.invoke(default_charset.clj:30)
        at ring.middleware.not_modified$wrap_not_modified$fn__8717.invoke(not_modified.clj:61)
        at ring.middleware.x_headers$wrap_x_headers$fn__8804.invoke(x_headers.clj:73)
```

Idea: PureScript + HTTP server middleware

!!YPER

Type-safe, statically checked composition of HTTP servers

Goals

- Provide a common API for middleware
- Make the effects of applying middleware explicit in types
- Safe, composable, middleware and application components
- Backend-agnostic middleware where possible
- Interoperability with NodeJS and other backends (purerl, purescript-native)
- No magic

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

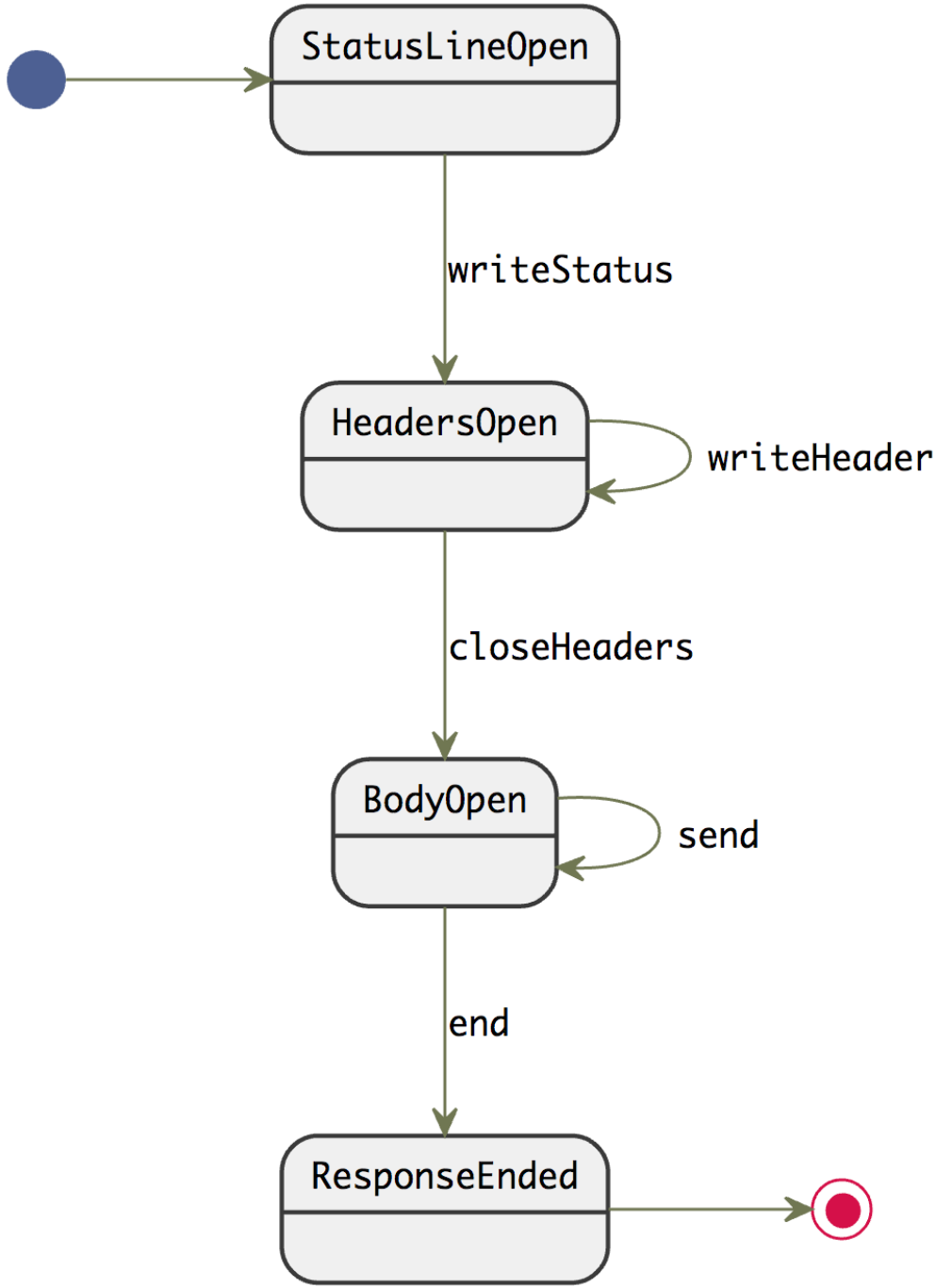
```
badMiddleware conn = do
  _ <- respond "First response" conn
  respond "Second response, crash!" conn
```

Let’s hide the Conn

```
badMiddleware = do
  respond "First response"
  respond "Second response, not ok!"
```

Middleware (Revised)

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


StatusLineOpen
writeStatus
HeadersOpen
writeHeader
closeHeaders
BodyOpen
send
end
ResponseEnded

Response State Transitions

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

ResponseStateTransition

```
type ResponseStateTransition m res from to =
  forall req c.
  Middleware
  m
  (Conn req (res from) c)
  (Conn req (res to) c)
  Unit
```

Response

```
class Response (res :: * -> *) m b | res -> b where
  writeStatus
    :: Status
    -> ResponseStateTransition m res StatusLineOpen HeadersOpen

  writeHeader
    :: Header
    -> ResponseStateTransition m res HeadersOpen HeadersOpen

  closeHeaders
    :: ResponseStateTransition m res HeadersOpen BodyOpen

  send
    :: b
    -> ResponseStateTransition m res BodyOpen BodyOpen

  end
    :: ResponseStateTransition m res BodyOpen ResponseEnded
```

What if we do it wrong?

```
do
  writeStatus statusOK
  respond "Hello, Hyper!"
  closeHeaders


  Could not match type

    BodyOpen

  with type

    HeadersOpen
```

Writing Headers

```
headers
  :: forall f m req res b c
   . ( Foldable f
     , Monad m
     , Response res m b
     )
  => f Header
  -> Middleware
     m
     (Conn req (res HeadersOpen) c)
     (Conn req (res BodyOpen) c)
     Unit
headers hs =
  traverse_ writeHeader hs
  :*> closeHeaders
```

Hyperdrive

Hyperdrive

```
type Application m request response =
  request -> m response
```

Hyperdrive

```
app _ =
  response "Hello Hyperdrive!"
  # status statusOK
  # header (Tuple "X-Hello" "Hyperdrive")
  # pure
```

Hyperdrive

```
app _ =
  pure
  (header (Tuple "X-Hello" "Hyperdrive")
   (status statusOK
    (response "Hello Hyperdrive!")))
```

Type-Level Routing

A Routing Type

```
data Home = Home

type Site1 = Resource (Get Home) HTML
```

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!")
```

How do we pass routing types to functions?

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

```
site1Router = router site1 home onRoutingError

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

Main Entrypoint

```
main = runServer defaultOptions {} site1Router
```

More Resources!

Desired Routing

```
GET /                   Home
GET /users              [User]
GET /users/:user-id     User
```

Multiple Resources with Captures

```
data Home = Home

data AllUsers = AllUsers (Array User)

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

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

Multiple Handlers

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

allUsers :: forall m. Monad m => ExceptT RoutingError m AllUsers

getUser :: forall m. Monad m => Int -> ExceptT RoutingError m User
```

Type-Safe Links

```
instance encodeHTMLAllUsers :: EncodeHTML AllUsers where
  encodeHTML (AllUsers users) =
    div do
      h1 (text "Users")
      ul (traverse_ 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

```
type TasksResource =
  Resource (Get (Array Task)) JSON

type TaskResource =
  Resource (Get Task) JSON

type Site =
  "tasks" :/ (TasksResource
              :<|> Capture "id" TaskId :> TaskResource)

site :: Proxy Site
site = Proxy
```

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

- Remove the ordering in type-level routing
- Type-safe forms
- PJAX, but with JSON data and client-side templates
- Mocked servers and clients using `Arbitrary` instances
- Other backends

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

flatMap(Oslo), May 2, 2017