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

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

- Welcome to my talk with a very long title
- The Power of Functional Programming and Static Type Systems in Server-Side Web Applications

Elegant Weapons for a More Civilized Page

Elegant Weapons for a More Civilized Page

- This was my original title, but I did not dare apply with it

Me

- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, mostly in Clojure

Me

- I live and work in Malmö, Sweden

Me

- I live and work in Malmö, Sweden

- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, mostly in Clojure

Me

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

- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, 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

Me

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

- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, mostly in Clojure

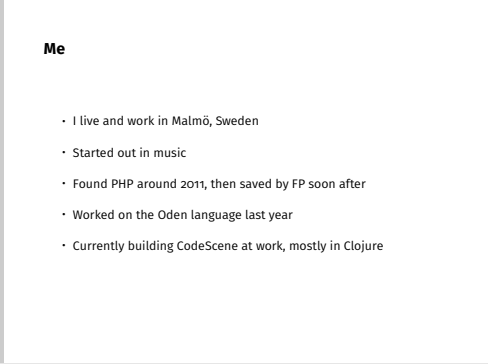
Me

- 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

- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, 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



- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, 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

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

- Let me introduce myself
- I live and work in Malmo in Sweden
- I started out in music
- Wrote my first band site in PHP
- Luckily I was saved by functional programming not long after
- Worked on Oden
- I am currently working on CodeScene, mostly in Clojure

Agenda

Agenda

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

Agenda

- Overview: Functional Programming and Web Applications

Agenda

- Overview: Functional Programming and Web Applications

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

Agenda

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

Agenda

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
    - Design
    - Type-Level Routing
    - XHR Clients
    - Future Work

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

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

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

- Let me go through what I'll cover today
- First: general overview of ...
- Then look closer at server-side rendering
- and some examples of static typing for server-side web, next level
- then we will dive into Hyper
  - Design
  - Type-Level Routing
  - XHR Clients
  - Future Work

Overview: Functional Programming and Web Applications

- Let’s get going

Functional Programming Influence

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

Functional Programming Influence

- FP influences Javascript

Functional Programming Influence

- FP influences Javascript

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

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

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

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

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

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 programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

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)

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

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

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

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

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

- Functional programming influences Javascript
- ECMAScript
- Underscore, Rambda, Fantasy Land
- React is functional ...
- FRP has made its way into ...
- Still, main focus of efforts...

Single-Page Applications

- Single page apps work more like traditional desktop apps
- Lot of JS frameworks, many just minor flavours of previous ones
- Some of the big ones are...
- And usually, without Javascript execution, you get nothing
  - Does this matter? Yes.
  - Disabled, flaky network, low-budget devices
- SPAs tends to reinvent things already present in web browers, like routing and forms

Single-Page Applications

- Work more like desktop applications

Single-Page Applications

- Work more like desktop applications

- Single page apps work more like traditional desktop apps
- Lot of JS frameworks, many just minor flavours of previous ones
- Some of the big ones are...
- And usually, without Javascript execution, you get nothing
  - Does this matter? Yes.
  - Disabled, flaky network, low-budget devices
- SPAs tends to reinvent things already present in web browers, like routing and forms

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

- Single page apps work more like traditional desktop apps
- Lot of JS frameworks, many just minor flavours of previous ones
- Some of the big ones are...
- And usually, without Javascript execution, you get nothing
  - Does this matter? Yes.
  - Disabled, flaky network, low-budget devices
- SPAs tends to reinvent things already present in web browers, like routing and forms

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)

- Single page apps work more like traditional desktop apps
- Lot of JS frameworks, many just minor flavours of previous ones
- Some of the big ones are...
- And usually, without Javascript execution, you get nothing
  - Does this matter? Yes.
  - Disabled, flaky network, low-budget devices
- SPAs tends to reinvent things already present in web browers, like routing and forms

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

- Single page apps work more like traditional desktop apps
- Lot of JS frameworks, many just minor flavours of previous ones
- Some of the big ones are...
- And usually, without Javascript execution, you get nothing
  - Does this matter? Yes.
  - Disabled, flaky network, low-budget devices
- SPAs tends to reinvent things already present in web browers, like routing and forms

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

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

- Single page apps work more like traditional desktop apps
- Lot of JS frameworks, many just minor flavours of previous ones
- Some of the big ones are...
- And usually, without Javascript execution, you get nothing
  - Does this matter? Yes.
  - Disabled, flaky network, low-budget devices
- SPAs tends to reinvent things already present in web browers, like routing and forms

What about Server-Side Rendering?

What about Server-Side Rendering?

- What about ...
- Do we need to make a binary choice?
- No. One approach is Progressive Enhancement...

Progressive Enhancement

- Progressive enhancement is about defining the lower bound of support
- Then gradually make things nicer for more capable clients
- Everyone above that threshold gets something, even if it’s rough

80/20

80/20

- Let’s say all of your code and functionality is not equally important
- A smaller part provides the core value of your business
- You have lots of supporting code as well
- Login forms, settings pages, payment, integrations, documentation
- Do they really needs Javascript to be able to work?
- I would claim most would work fine with links and forms
- Let’s write client-side code where it provides value
- Unless you enjoy reinventing the web browser over and over

“Isomorphic” Web Applications

- I want to mention the hype around *isomorphic* web apps
- Goal:...
- One executes in the browser locally
- The other transfers state over the network through links and forms
- Are we really talking about initial rendering ...?

“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

- I want to mention the hype around *isomorphic* web apps
- Goal:...
- One executes in the browser locally
- The other transfers state over the network through links and forms
- Are we really talking about initial rendering ...?

“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”

- I want to mention the hype around *isomorphic* web apps
- Goal:...
- One executes in the browser locally
- The other transfers state over the network through links and forms
- Are we really talking about initial rendering ...?

“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

- I want to mention the hype around *isomorphic* web apps
- Goal:...
- One executes in the browser locally
- The other transfers state over the network through links and forms
- Are we really talking about initial rendering ...?

“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?

“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?

- I want to mention the hype around *isomorphic* web apps
- Goal:...
- One executes in the browser locally
- The other transfers state over the network through links and forms
- Are we really talking about initial rendering ...?

PJAX

- A nice approach that we use for CodeScene is PJAX...
- It progressively enhances a server-side app with low effort
- *explain steps...*
- Maybe this is a bit rough...

PJAX

- Hooks in on link and form events

PJAX

- Hooks in on link and form events

- A nice approach that we use for CodeScene is PJAX...
- It progressively enhances a server-side app with low effort
- *explain steps...*
- Maybe this is a bit rough...

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

- A nice approach that we use for CodeScene is PJAX...
- It progressively enhances a server-side app with low effort
- *explain steps...*
- Maybe this is a bit rough...

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

- A nice approach that we use for CodeScene is PJAX...
- It progressively enhances a server-side app with low effort
- *explain steps...*
- Maybe this is a bit rough...

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

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

- A nice approach that we use for CodeScene is PJAX...
- It progressively enhances a server-side app with low effort
- *explain steps...*
- Maybe this is a bit rough...

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

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

- I do not know if this is the case...
- Let’s use good languages and make use of ... (change slide)

Static Typing for Server-Side Web

- Static typing for server-side web

Static Typing for Server-Side Web

- We have seen a lot of dynamically typed languages in web development, PHP, Ruby...
- Also the mainstream object-oriented like Java and C#, perhaps with MVC frameworks
- When programs grow big, and teams grow big, static typing is our friend
- We should be able to safely build abstractions and compose things
- Be confident that the pieces work together
- We should be able to maintain code, refactor it, change it, delete it
- Again, I think we need a good developer experience
- Not only making stuff shiny, but by using solid theory and research

Static Typing for Server-Side Web

- Mainstream languages in web server programming

Static Typing for Server-Side Web

- Mainstream languages in web server programming

- We have seen a lot of dynamically typed languages in web development, PHP, Ruby...
- Also the mainstream object-oriented like Java and C#, perhaps with MVC frameworks
- When programs grow big, and teams grow big, static typing is our friend
- We should be able to safely build abstractions and compose things
- Be confident that the pieces work together
- We should be able to maintain code, refactor it, change it, delete it
- Again, I think we need a good developer experience
- Not only making stuff shiny, but by using solid theory and research

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

- We have seen a lot of dynamically typed languages in web development, PHP, Ruby...
- Also the mainstream object-oriented like Java and C#, perhaps with MVC frameworks
- When programs grow big, and teams grow big, static typing is our friend
- We should be able to safely build abstractions and compose things
- Be confident that the pieces work together
- We should be able to maintain code, refactor it, change it, delete it
- Again, I think we need a good developer experience
- Not only making stuff shiny, but by using solid theory and research

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

- We have seen a lot of dynamically typed languages in web development, PHP, Ruby...
- Also the mainstream object-oriented like Java and C#, perhaps with MVC frameworks
- When programs grow big, and teams grow big, static typing is our friend
- We should be able to safely build abstractions and compose things
- Be confident that the pieces work together
- We should be able to maintain code, refactor it, change it, delete it
- Again, I think we need a good developer experience
- Not only making stuff shiny, but by using solid theory and research

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

- We have seen a lot of dynamically typed languages in web development, PHP, Ruby...
- Also the mainstream object-oriented like Java and C#, perhaps with MVC frameworks
- When programs grow big, and teams grow big, static typing is our friend
- We should be able to safely build abstractions and compose things
- Be confident that the pieces work together
- We should be able to maintain code, refactor it, change it, delete it
- Again, I think we need a good developer experience
- Not only making stuff shiny, but by using solid theory and research

Static Typing for Server-Side Web

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

Static Typing for Server-Side Web

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

- We have seen a lot of dynamically typed languages in web development, PHP, Ruby...
- Also the mainstream object-oriented like Java and C#, perhaps with MVC frameworks
- When programs grow big, and teams grow big, static typing is our friend
- We should be able to safely build abstractions and compose things
- Be confident that the pieces work together
- We should be able to maintain code, refactor it, change it, delete it
- Again, I think we need a good developer experience
- Not only making stuff shiny, but by using solid theory and research

Things I’ve Found

Things I’ve Found

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I’ve Found

- Haskell:

Things I’ve Found

- Haskell:

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I've Found

- Haskell:
  - Yesod

Things I've Found

- Haskell:
  - Yesod

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I’ve Found

- Haskell:
  - Yesod
  - Servant

Things I’ve Found

- Haskell:
  - Yesod
  - Servant

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I've Found

- Haskell:
  - Yesod
  - Servant
- Scala:

Things I've Found

- Haskell:
  - Yesod
  - Servant
- Scala:

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I've Found

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

Things I've Found

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

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I've Found

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

Things I've Found

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

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I've Found

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

Things I've Found

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

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

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

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Things I've Found

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

Things I've Found

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

- There are some things out there
- For Haskell, two alternatives are ...
- Not that I know much about Scala, but I found these...
- And, as we are going to talk about PureScript, I want to mention...
- (10 MINUTES?)

Statically Typed Middleware

Statically Typed Middleware

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

Statically Typed Middleware

- Middleware is a common abstraction

Statically Typed Middleware

- Middleware is a common abstraction

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

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!

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!

- Looking at other frameworks, like connect and Plug...
- In Java Servlets it's called filters
- It is very easy to mess up. You can get... (up to missing auth)
- And, if you're working with Clojure, you often get this ...

```
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__8565.invoke(resource.clj:28)
        at ring.middleware.content_type$wrap_content_type$fn__8631.invoke(content_type.clj:38)
        at ring.middleware.default_charset$wrap_default_charset$fn__8666.invoke(default_charset.clj:28)
        at ring.middleware.not_modified$wrap_not_modified$fn__8687.invoke(not_modified.clj:61)
        at ring.middleware.x_headers$wrap_x_headers$fn__8904.invoke(x_headers.clj:51)
```

Let’s use extensible records in PureScript!

Let’s use extensible records in PureScript!

- After a talk by Edwin Brady on Idris and session types, I got this idea...
- How about using extensible records ...

!!YPER

Type-safe, statically checked composition of HTTP servers

!!YPER

Type-safe, statically checked composition of HTTP servers

- And that's where Hyper started
- (20 MINUTES?)

Goals

Goals

- The goals are...
- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition
- Interoperability with NodeJS and other backends
- Different web servers, Erlang backend, purescript-native
- Also: no magic. No preprocessing or code generation.

Goals

- A safe HTTP middleware architecture

Goals

- A safe HTTP middleware architecture

- The goals are...
- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition
- Interoperability with NodeJS and other backends
- Different web servers, Erlang backend, purescript-native
- Also: no magic. No preprocessing or code generation.

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

- The goals are...
- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition
- Interoperability with NodeJS and other backends
- Different web servers, Erlang backend, purescript-native
- Also: no magic. No preprocessing or code generation.

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

- The goals are...
- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition
- Interoperability with NodeJS and other backends
- Different web servers, Erlang backend, purescript-native
- Also: no magic. No preprocessing or code generation.

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)

- The goals are...
- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition
- Interoperability with NodeJS and other backends
- Different web servers, Erlang backend, purescript-native
- Also: no magic. No preprocessing or code generation.

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

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

- The goals are...
- A safe HTTP middleware architecture
- Make the effects of applying middleware explicit in types
- Ensure correct composition
- Interoperability with NodeJS and other backends
- Different web servers, Erlang backend, purescript-native
- Also: no magic. No preprocessing or code generation.

How?

How?

- Track middleware effects in type system...
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware
- Integrate with existing NodeJS libraries

How?

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

How?

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

- Track middleware effects in type system...
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware
- Integrate with existing NodeJS libraries

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

- Track middleware effects in type system...
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware
- Integrate with existing NodeJS libraries

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

- Track middleware effects in type system...
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware
- Integrate with existing NodeJS libraries

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

- Track middleware effects in type system...
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware
- Integrate with existing NodeJS libraries

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

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

- Track middleware effects in type system...
- Leverage extensible records in PureScript
- Provide a common API for middleware
- Write backend-agnostic middleware
- Integrate with existing NodeJS libraries

Design

- I will go through the core design first

Conn

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


Conn

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

- Conn, short for Connection
- Models the entirety of an HTTP connection
- Design adapted from Plug in Elixir
- It has a request and response, maybe not surprising
- But also components, which is an extension point for middleware
- I'm not so sure about components yet, it may go away

Middleware (Old Design)

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


Middleware (Old Design)

type Middleware m c c’ = c -> m c’

- A function transformed a Conn to another Conn, in some type *m*
- The *m* was usually an Applicative or a Monad
- A type synonym for a regular function
- This signature might remind you of Bind

Middleware (Old Design)

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


Middleware (Old Design)

authenticateUser >=> parseForm >=> saveTodo

- Middleware could therefore be composed using regular Kleisli composition
- Here we see three middleware, composed from left to right
- The connection goes into the first one, it gives back a new one, goes into the second, and so on
- All good? Not really.

Whoops, Not Safe

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


Whoops, Not Safe

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

- We define a middleware called *badMiddleware*
- It takes a conn and passes it to the first respond middleware
- It ignores that resulting conn, passes the argument again
- Had we passed the result, or used Kleisli, we would have a type error
- The problem: Our safety is in the Conn values, and we can discard them
- The solution: The monad has to carry the Conn type

Middleware (Revised)

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


Middleware (Revised)


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


- The new middleware is an indexed monad
- Works as a state monad, with state type changes in its parameters
- You cannot run an effectful middleware without the types matching
- Let's see how this can solve the double-respond problem from before
- We will begin by looking at Response State Transitions

Response State Transitions

Response State Transitions

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

Response State Transitions

- Hyper tracks the state of response writing

Response State Transitions

- Hyper tracks the state of response writing

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

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

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

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

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

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

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

- Hyper tracks the state of response writing
- Guarantees correctness in response side effects, preventing common programming errors
- 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 =
```

- This is a helper type alias
- A transition is a Middleware
- Takes Conn with Response writer in the **from** state
- Transforms it to a Conn with Response writer in the **to** state

ResponseStateTransition

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


ResponseStateTransition

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

- This is a helper type alias
- A transition is a Middleware
- Takes Conn with Response writer in the **from** state
- Transforms it to a Conn with Response writer in the **to** state

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
```

- This is a helper type alias
- A transition is a Middleware
- Takes Conn with Response writer in the **from** state
- Transforms it to a Conn with Response writer in the **to** state

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)
```

- This is a helper type alias
- A transition is a Middleware
- Takes Conn with Response writer in the **from** state
- Transforms it to a Conn with Response writer in the **to** state

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)
```

- This is a helper type alias
- A transition is a Middleware
- Takes Conn with Response writer in the **from** state
- Transforms it to a Conn with Response writer in the **to** state

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
```


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
```

- This is a helper type alias
- A transition is a Middleware
- Takes Conn with Response writer in the **from** state
- Transforms it to a Conn with Response writer in the **to** state

ResponseWriter

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


ResponseWriter

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

- ResponseWriter encodes the possible state transitions
- Go through them one by one

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
```

- ResponseWriter encodes the possible state transitions
- Go through them one by one

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

- ResponseWriter encodes the possible state transitions
- Go through them one by one

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
```

- ResponseWriter encodes the possible state transitions
- Go through them one by one

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
```

- ResponseWriter encodes the possible state transitions
- Go through them one by one

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
```


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
```

- ResponseWriter encodes the possible state transitions
- Go through them one by one

What if we do it wrong?

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

Could not match type

  BodyOpen

with type

  HeadersOpen
```


What if we do it wrong?

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

Could not match type

  BodyOpen

with type

  HeadersOpen
```

- And if we write responses incorrectly, we get a compile-time error
- With this safety, we can without fear build new things on top

Writing Headers

```
headers
```


Writing Headers

```
headers
```

- The *headers* function is a minimal example of abstraction
- It writes all specified headers, then closes headers
- The implementation? Pretty simple.
- Now for a much higher-level abstraction on top of Hyper, namely *type-level routing*
- (30 MINUTES?)

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
```

- The *headers* function is a minimal example of abstraction
- It writes all specified headers, then closes headers
- The implementation? Pretty simple.
- Now for a much higher-level abstraction on top of Hyper, namely *type-level routing*
- (30 MINUTES?)

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
```

- The *headers* function is a minimal example of abstraction
- It writes all specified headers, then closes headers
- The implementation? Pretty simple.
- Now for a much higher-level abstraction on top of Hyper, namely *type-level routing*
- (30 MINUTES?)

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
```


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
```

- The *headers* function is a minimal example of abstraction
- It writes all specified headers, then closes headers
- The implementation? Pretty simple.
- Now for a much higher-level abstraction on top of Hyper, namely *type-level routing*
- (30 MINUTES?)

Type-Level Routing

- This is very much inspired by Servant, for Haskell
- I can really recommend Servant
- This version for Hyper, being written in PureScript, has some benefits
- I'll get to that later

A Routing Type

```
data Home = Home

type Site1 = Get HTML Home
```


A Routing Type

```
data Home = Home

type Site1 = Get HTML Home
```

- We first define a resource data type
- Then we define a routing type
- Note that the routing type has no value-level representation defined

Handler

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


Handler

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

- We also need a handler
- The handler is a function applied when a route is matched
- It returns a response for a request
- Argument types and return type can be our custom data types
- But how does one respond with HTML, or JSON

HTMLEncode

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


HTMLEncode


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


- We need an instance of MimeEncoder
- By defining an instance for EncodeHTML, we get the MimeEncoder automatically
- Hyper includes a small HTML DSL for now
- It will probably use an existing one called *smolder* instead
- Remember how our routing type had no value-level representation?
- We need to pass it to the router, but how?

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.

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.

- We use the PureScript Proxy type

Site Proxy

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


Site Proxy

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

- Here we create a proxy for our routing type

Site Router

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

site1Router = router site1 home onRoutingError
```


Site Router

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

site1Router = router site1 home onRoutingError
```

- We define an HTTP error handler
- And our router, by passing the proxy and our home handler

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)
```


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)
```

- Lastly, we define our main function that starts the server

More Routes!

- Having only one route seems lame. Let’s add some more

Multiple Endpoints with Captures

```haskell
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 Endpoints with Captures


```haskell
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
```


- (explain code)

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
```

- Our home handler looks like before
- allUsers is simply wrapping the array of user values
- getUser performs a lookup, possibly throws Not Found error
- Now that we have multiple routes, how do we link between them?

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
```

- Our home handler looks like before
- allUsers is simply wrapping the array of user values
- getUser performs a lookup, possibly throws Not Found error
- Now that we have multiple routes, how do we link between them?

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'
```


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'
```

- Our home handler looks like before
- allUsers is simply wrapping the array of user values
- getUser performs a lookup, possibly throws Not Found error
- Now that we have multiple routes, how do we link between them?

Type-Safe Links

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


Type-Safe Links

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

- In our HTML encoder, we want to produce anchor tags
- As we have routing types, we get type-safe links for free

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)
                   ]
```


- In our HTML encoder, we want to produce anchor tags
- As we have routing types, we get type-safe links for free

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 ] ]
```


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 ] ]
```

- In our HTML encoder, we want to produce anchor tags
- As we have routing types, we get type-safe links for free

Multiple Endpoint Router

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


Multiple Endpoint Router

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

- We combine our handlers with a structure matching the routing type
- Matching the type incorrectly would result in a type error
- OK, so that is nice.
- But I talked about progressive enhancement before? How about that?

Automatically Derived XHR Clients

- Instead of sprinkling Javascript or jQuery scripts all over the place
- with no type safety or guarantees they are in sync with routes
- Let’s automatically dervice safe XHR clients

Shared Routing Type

```
type TaskId = Int

data Task = Task TaskId String
```


Shared Routing Type

```
type TaskId = Int

data Task = Task TaskId String
```

- We define a type Task

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

```
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
```

- Provide some encoding and decoding instances for JSON

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)
```

- Define the routing type
- and our Proxy value
- We write our handlers and a server just like before
- Finally, we use it in a client-side application

Shared Routing Type

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

site :: Proxy Site
site = Proxy
```


Shared Routing Type

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

site :: Proxy Site
site = Proxy
```

- Define the routing type
- and our Proxy value
- We write our handlers and a server just like before
- Finally, we use it in a client-side application

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
```

- This is a snippet from a PureScript Pux app
- The code in bold is where we generate AJAX clients for our site
- We pattern match on the structure to get a specific client
- I think this is a very strong case for Hyper

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 :<|> _ ->
```

- This is a snippet from a PureScript Pux app
- The code in bold is where we generate AJAX clients for our site
- We pattern match on the structure to get a specific client
- I think this is a very strong case for Hyper

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 ]
      }
```


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 ]
      }
```

- This is a snippet from a PureScript Pux app
- The code in bold is where we generate AJAX clients for our site
- We pattern match on the structure to get a specific client
- I think this is a very strong case for Hyper

Other Possibilites/Future Work

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

Other Possibilites/Future Work

- Type-safe forms

Other Possibilites/Future Work

- Type-safe forms

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

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

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

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

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

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

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

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

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

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!

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!

- So, what more can we do here?
- I'd like type safe form posts
- PJAX, the hyper way
- Mocked servers with Arbitrary
- Ring-like abstraction for response maps
- Other backends
- And in general, continue the quest for type-safe web!

Summary

- Server-side or SPA is not a binary choice
- We can make the middleground much nicer, building better tools
- Good programming languages and type systems are our allies

Thank You!

Thank You!

- Thank you
- Here are some links to things I've mentioned...

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 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 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.

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

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

- Questions?