Fast and Fearless Evolution of Server-Side Web Applications

Oskar Wickström

@owickstrom

Outline

- Introduction
- Web Applications
- Writing Web Applications with Haskell
  - Scotty
  - Yesod
  - Airship
- Client-Side Technologies

Introduction

Evolving Software

- New features
- Bug fixes
- Refactoring
- External factors
  - New regulation
  - Deprecation of software and services
- Recruiting

Risks

- Evolving software, we face risks:
  - Delay
  - Error
  - Burnout
- These risks make it scary to evolve freely

Tooling

- We can reduce fear with better tools
  - Stronger correctness guarantees
  - Robustness
  - Faster feedback
  - Communicates intent
- Functional programming
- Type systems
- Error handling

Web Applications

Web Applications

- Many of us work with the web somehow
- Single-page apps (SPAs) are in vogue
  - More like desktop apps
  - Reinventing parts of the browser
  - No Javascript, no application
- Universal webapps (aka “isomorphic”)

Reminder

$$\text{Newer} \not\Longrightarrow \text{Better}$$

Server-Side Web Applications

- Do not dismiss server-side web applications
- Progressive enhancement
- 80/20 rule
- Use client-side code where you need it!
- PJAX

Static Typing for Server-Side Web

- Compile-time checking
  - Run-time robustness with defined behaviour
  - Use types for correct-by-construction
  - Machine-verified living documentation, communicates intent
- Safely evolve our codebase
  - Reduce fear of change throughout the codebase
  - Modify core domain, follow the errors
  - Not split by an API
- Focus tests on our domain
  - No need to write tests for type errors
  - Domain code free of side effects

Functional Statically Typed Web

- Many languages, many frameworks!
- Look for the patterns and safety
- Less power is more power
- Today's focus is Haskell

Writing Web Applications with Haskell

Underpinnings

- Web Application Interface (WAI)
  - Common interface betwen web applications and web servers
  - Mix frameworks in one application
  - Comparable with Java Servlet API
- Warp
  - WAI web server
  - Uses GHC's lightweight threads

Frameworks

- Scotty
- Spock
- Yesod
- Happstack
- Snap
- Airship
- Servant
- MFlow

Scotty

Scotty

- Inspired by Ruby's Sinatra
- Features
  - Routing and parameters
  - Web server setup
  - Extensible
- "Build your own framework"

Scotty Routing

```
app :: ScottyM ()
app = do

  get "/" $
    html "Welcome!"

  get "/greet/:who" $ do
    who <- param "who"
    html ("Hello, " <> who <> "!")
```

Scotty Server

```haskell
main :: IO ()
main = scotty 8080 app
```

HTML Templates

```
get "/greet-with-template/:who" $ do
  who <- param "who"
  html $
       "<!DOCTYPE html>\
       \<html lang=\"en\">\
       \<head>\
       \  <meta charset=\"UTF-8\">\
       \  <title>My Page</title>\
       \  <link rel=\"stylesheet\"\
       \        href=" <> bootstrapCss <> "\">\
       \</head>\
       \<body>\
       \  <div class=\"jumbotron\">\
       \     <h1>Hello, " <> who <> "!</h1>\
       \  </div>\
       \</body>\
       \</html>"
```

HTML Template Error!

```
get "/greet-with-template/:who" $ do
  who <- param "who"
  html $
       "<!DOCTYPE html>\
       \<html lang=\"en\">\
       \<head>\
       \  <meta charset=\"UTF-8\">\
       \  <title>My Page</title>\
       \  <link rel=\"stylesheet\"\
       \       href=" <> bootstrapCss <> "\">\
       \</head>\
       \<body>\
       \  <div class=\"jumbotron\">\
       \     <h1>Hello, " <> who <> "!</h1>\
       \  </div>\
       \</body>\
       \</html>"
```

DSLs for HTML

- Instead of HTML in strings, we use DSLs
- Embedded:
  - Blaze
  - Lucid
- External:
  - Heist
  - Hamlet
- Type safety
- Composable

Lucid HTML Template

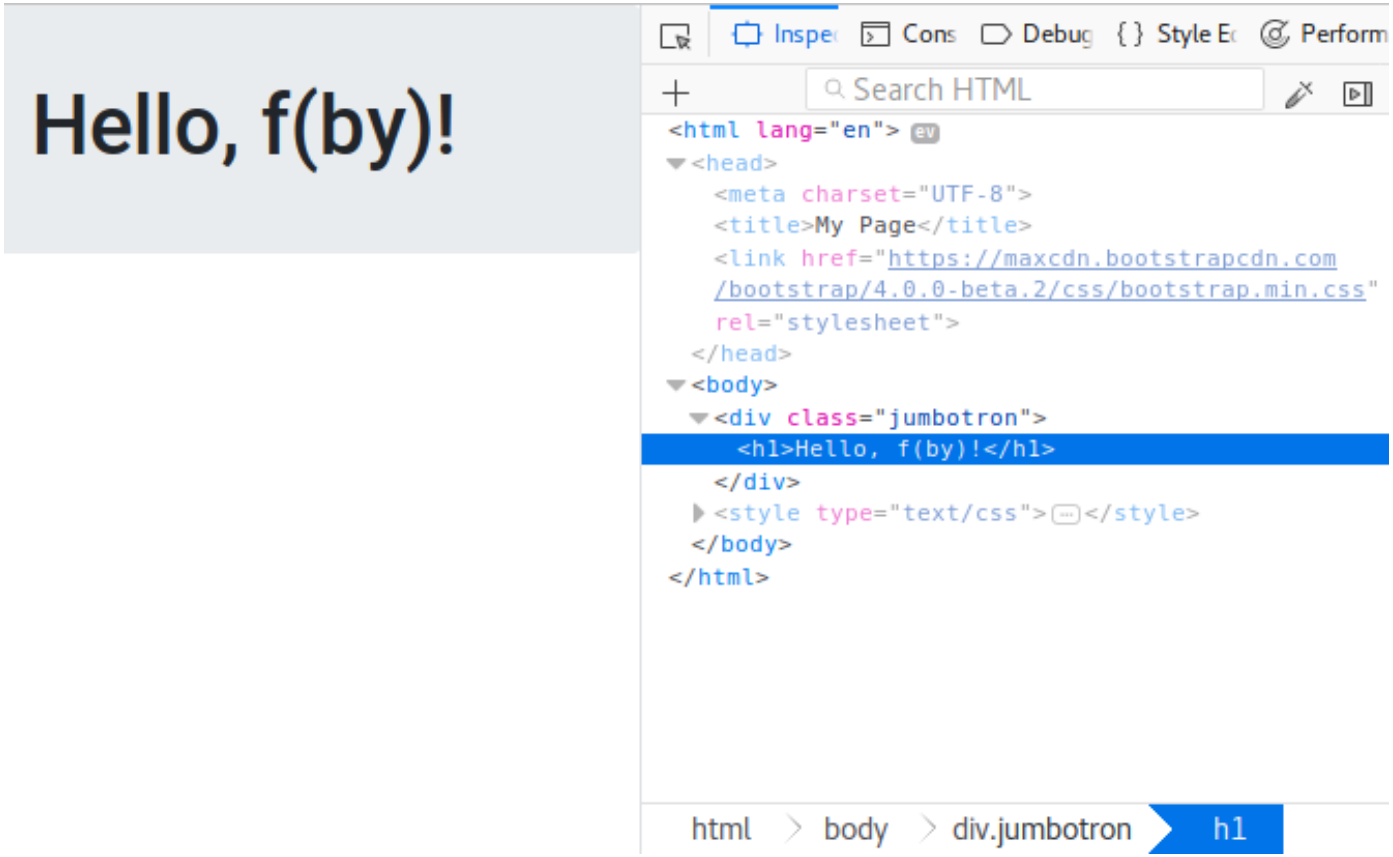
```haskell
homeView :: Text -> Html ()
homeView who =
  html_ [lang_ "en"] $ do
    head_ $ do
      meta_ [charset_ "UTF-8"]
      title_ "My Page"
      link_ [rel_ "stylesheet", href_ bootstrapCss]

    body_ $
      div_ [class_ "jumbotron"] $
        h1_ ("Hello, " <> toHtml who <> "!")
```

Rendering Lucid with Scotty

```
get "/greet-with-lucid/:who" $ do
  who <- param "who"
  html (renderText (homeView who))
```

Result


Hello, f(by)!
Inspe
Cons
Debug
Style E
Perform
Search HTML
<html lang="en">
<head>
<meta charset="UTF-8">
<title>My Page</title>
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>

<h1>Hello, f(by)!</h1>

<style type="text/css">...</style>
</body>
</html>
html
body
div.jumbotron
h1

Side Effects in Scotty

- We need more than sending HTML responses
- We want to do IO:
  - Database queries
  - Logging
  - External service calls
- IO in Scotty handlers using `liftIO`

IO in Scotty

- Given these definitions:

```
type ArticleId = Text

addNewComment :: ArticleId -> Text -> IO ()
```

- We can *lift* the IO action into a handler:

```
post "/articles/:article-id/comments" $ do
  articleId <- param "article-id"
  -- accepts a form or query parameter "comment"
  comment <- param "comment"
  liftIO (addNewComment articleId comment)
  redirect ("/articles/" <> articleId)
```

Starting with Scotty

- Easy to get started, learn the basics
- What you don't get:
  - Templating
  - Sessions
  - Authentication and Authorization
  - Logging
  - Persistence
- Have a look at Spock[1] for more features

[1] https://www.spock.li

Yesod

Yesod

- “One-stop shop” for Haskell web development
  - A framework
  - Batteries included
  - Still very modular
- Also runs on WAI

Batteries Included with Yesod

- Type-safe routing
- External templates for:
  - HTML
  - CSS
  - Javascript (and TypeScript)
- Widgets
- Forms
- Sessions
- Integration with Persistent
- Authentication and Authorization
- Internationalization
- Logging
- Configuration
- Auto-reloading web server

Getting Started

- Use a template (see `stack templates`)
- There will be things you don't understand at first
- Start out exploring:
  - Routing
  - Templates (HTML, CSS, Javascript)
  - The "Foundation" type
  - Getting something done!
- Over time, you'll understand the scaffolding
- Use the auto-reloading web server
  - Install `yesod-bin`, run `yesod devel`

Routes Configuration

```
/ HomeR GET

/articles/#ArticleId ArticleR GET
```

A Simple Handler

```
getHomeR :: Handler Html
getHomeR = do
  articles <- allArticles
  defaultLayout $ do
    setTitle "My Blog"
    $(widgetFile "homepage")
```

Hamlet Template

```
<h1>My Blog
<ul>
  $forall article <- articles
    <li>
      <a href=@{ArticleR (articleId article)}>#{articleTitle arti
```

Routing with Path Pieces

```
getArticleR :: ArticleId -> Handler Html
getArticleR id' = do
  article <- getArticle id'
  comments <- getArticleComments id'
  defaultLayout $ do
    setTitle (Html.text (articleTitle article))
    $(widgetFile "article")
```

Article Hamlet Template

```
<h1>#{articleTitle article}
<p>#{articleContent article}

<div .comments>
  <h2>Comments
  <ul>
    $forall comment <- comments
      <li .comment>
        <span .author>#{commentAuthor comment}
        <span .content>#{commentContents comment}
```

Widgets

- Reusable components of HTML, CSS, and Javascript
- We used widgets in handlers:

```
$(widgetFile "article")
```

- Yesod tries to find matching widget files:

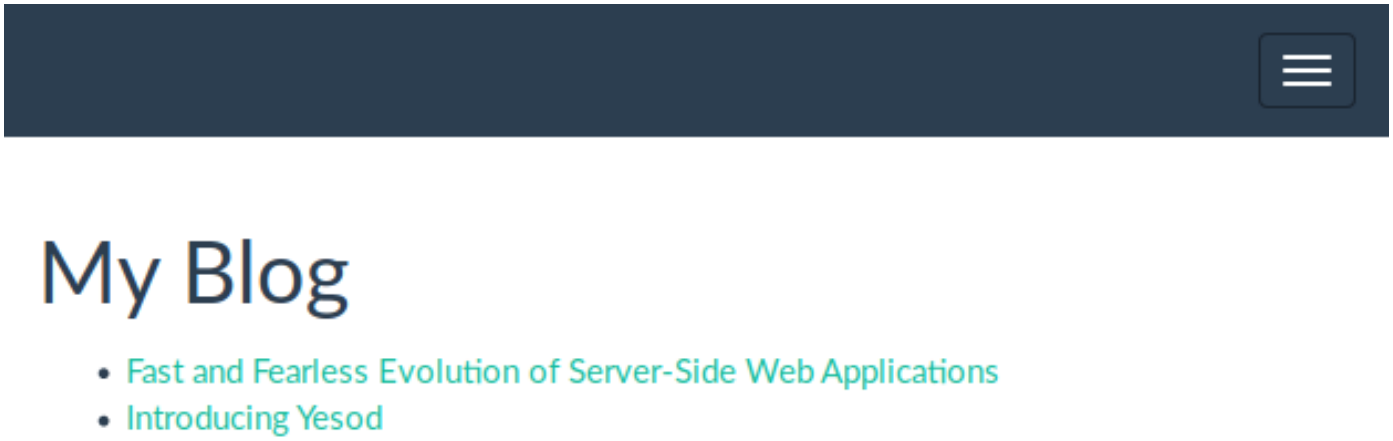
```
templates/article.hamlet
templates/article.cassius
templates/article.lucius
templates/article.julius
```

- Can refer to bindings in Haskell code
- Only include small parts, or use external resources

Lucius (CSS Templates)

```
.comments {
  margin-top: 3em;
}
.comments ul {
  list-style-type: none;
  padding: 0;
}
.comment {
  background: #eee;
  padding: .5em;
  margin-bottom: 1em;
}
.author {
  font-weight: bold;
}
.author:after {
  content: ':';
}
```

Home Page Result



Article Page Result

Introducing Yesod

Lorem ipsum dolor sit amet...

Comments

Carol: Wonderful post! Keep them coming.

Mallory: I have have opinions. You suck.

Yesod Forms

- Write forms using applicative or monadic style
- Use the same structure for rendering, parsing, and validation
- There are various renderers available

Comment Form

```
commentForm :: AForm Handler Comment
commentForm =
  Comment
  <$> areq textField (named "Name") Nothing
  <*> (unTextarea
       <$> areq textareaField (named "Comment") Nothing)
```

Rendering a Form

```
getArticleWithFormR :: ArticleId -> Handler Html
getArticleWithFormR id' = do
  article <- getArticle id'
  comments <- getArticleComments id'

  (commentFormWidget, commentFormEnc) <-
    generateFormPost (renderForm commentForm)

  defaultLayout $ do
    setTitle (Html.text (articleTitle article))
    $(widgetFile "article-with-form")
```

Including The Form Widget

```
<form role=form
      method=post
      action=@{ArticleCommentsR id'}
      enctype=#{commentFormEnc}>
  ^{commentFormWidget}
  <button type="submit" .btn .btn-default>Submit
```

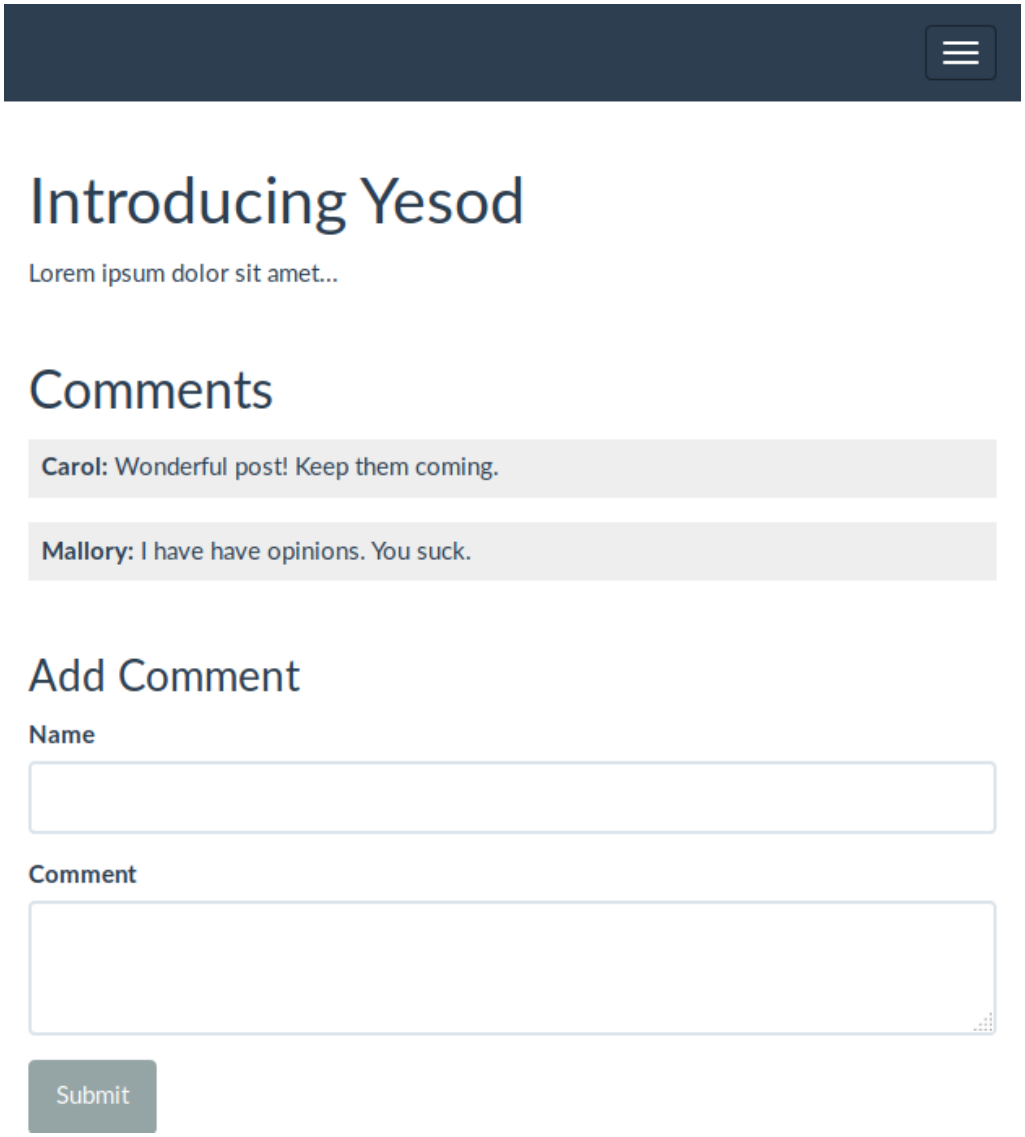
Parsing and Validating the Form

```haskell
postArticleCommentsR :: ArticleId -> Handler Html
postArticleCommentsR id' = do
  article <- getArticle id'
  comments <- getArticleComments id'

  ((result, commentFormWidget), commentFormEnc) <-
    runFormPost (renderForm commentForm)

  case result of
    FormSuccess comment -> do
      addArticleComment id' comment
      redirect (ArticleWithFormR id')
    _ ->
      defaultLayout $ do
        setTitle (Html.text (articleTitle article))
        $(widgetFile "article-with-form")
```

Comment Form Result



@owickstrom

Yesod Recap

- Very capable, hit the ground running
- We only looked at some core features
- Worth learning

Airship

Airship

- Inspired by Webmachine from Erlang
- Define RESTful resources
- Override fields in the default resource
- Tie together resources with routing

Airship Routes

```
appRoutes :: Resource IO -> RoutingSpec IO ()
appRoutes static = do
  "articles" </> var "articleId" #> articleResource
  "static"   </> star            #> static
```

Defining Resources

```
articleResource :: Resource IO
articleResource =
  defaultResource
  {
    -- overrides ...
  }
```

resourceExists

```
...

  , resourceExists =
      routingParam "articleId" >>= articleExists

...
```

contentTypesProvided

```
...

  , contentTypesProvided =
    let htmlResponse (Just article) =
          return (textResponse (renderArticle article))
        htmlResponse Nothing =
          return response404
    in return [("text/html", routingParam "articleId"
                             >>= getArticle
                             >>= htmlResponse)]

...
```

404 Not Found

```
$ curl -i 'localhost:3000'
HTTP/1.1 404 Not Found
Transfer-Encoding: chunked
Date: Tue, 12 Dec 2017 15:43:29 GMT
Server: Warp/3.2.13
Content-Type: text/html

Not found!
```

405 Method Not Allowed

```
$ curl -i -X PUT 'localhost:3000/articles/1'
HTTP/1.1 405 Method Not Allowed
Transfer-Encoding: chunked
Date: Tue, 12 Dec 2017 15:44:21 GMT
Server: Warp/3.2.13
Allow: GET,HEAD,POST
```

406 Not Acceptable

```
$ curl -i -H 'Accept: text/plain' 'localhost:3000/articles/1'
HTTP/1.1 406 Not Acceptable
Transfer-Encoding: chunked
Date: Tue, 12 Dec 2017 15:48:27 GMT
Server: Warp/3.2.13
```

200 OK

```
$ curl -i 'localhost:3000/articles/1'
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Tue, 12 Dec 2017 15:45:29 GMT
Server: Warp/3.2.13
Content-Type: text/html

<h1>Airship Webmachines!</h1><p>Lorem ipsum...</p>
```

Airship Overrides

```
allowMissingPost
allowedMethods
contentTypesAccepted
contentTypesProvided
deleteCompleted
deleteResource
entityTooLarge
  forbidden
generateETag
implemented
isAuthorized
isConflict
knownContentType
lastModified
languageAvailable
malformedRequest
movedPermanently
movedTemporarily
multipleChoices
previouslyExisted
  processPost
resourceExists
serviceAvailable
uriTooLong
validContentHeaders
```

Airship Considerations

- It is more low-level/barebones
- Again, “build your own framework”
- Suited for RESTful APIs

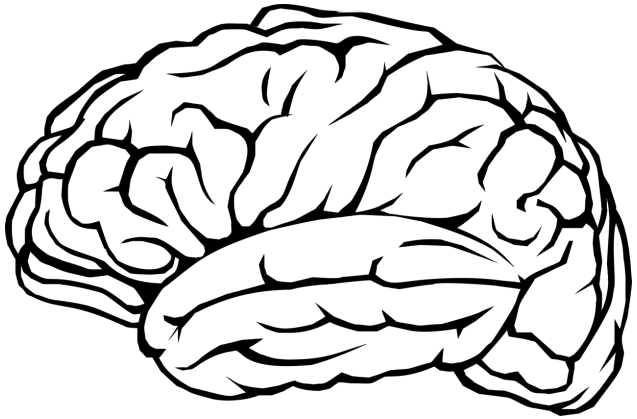
Client-Side Technologies

Client-Side Technologies

Need to do a single-page app?

- PureScript, Elm, etc
- Consider Haskell for your backend
- With Servant, you can use `servant-purescript` or `servant-elm`

Summary



Evolve software fearlessly using better tools for modeling and communication.

Spend your complexity budget carefully.

Explore the wonderful world of functional and statically typed server-side web.

Links

- Slides and code: github.com/owickstrom/fast-and-fearless-evolution-of-server-side-webapps
- Website: https://wickstrom.tech
- Twitter: @owickstrom

Questions?