Getting Started with Yesod (in anger)

Ben Sima[1]

2018-04-10 Tue

[1] ben@bsima.me

Abstract

Haskell can be a complex, powerful, highly abstract language. Usually these are not features you want in web development, however. That's why popular frameworks in languages such as Ruby or Python emphasize, "simple", "easy" and "it just works". This talk will share with you a Haskell secret: coding for the web in Haskell can be clear, simple, and enjoyable to write, potentially even more so than in Ruby/Python. This talk will introduce the Yesod web framework and illustrate how Yesod's simple abstractions corral Haskell's complexity into an enjoyable and safe value-level language for web programming.

What is Yesod?

- web framework (MVC)
- set of libraries
- a few "DSL"s
- a typeclass(es)

Pros and Cons of Yesod

- + Batteries included
- + Established tech (SQL, HTML, forms, MVC)
- + Fast & reusable widgets
- + Type safe routing, form validation, DB queries
- + Cross compilation? Mobile/desktop?
- ± jQuery
- ± Opinionated
- - Non-SPA
- - SQL

Alternatives

- miso or reflex-dom
  - Haskell in the browser, GHCjs
- servant
  - type-level routing
- acid-state (happstack)
  - give ACID guarantees to any Haskell datatype
- snap / scotty
  - lightweight "web frameworks"

Yesod architecture - MVC

```
src/
  Handler/Home.hs
  Helper/Mailchimp.hs
  Settings.hs
  Application.hs
  Foundation.hs
templates/
  home.hamlet
  home.{lucius,cassius}
  home.julius
config/
  models
  routes
  settings.yml
```

Model (Persistent)

Defining an example model

DSL (config/models)

```
Entry
  author UserId
  content Markdown
  title Text Maybe
  pubdate UTCTime
  deriving Show Eq

User
  name Text
```

Generated Haskell

```
data Entry = Entry
  { entryAuthor :: UserId
  , entryContent :: Markdown
  , entryTitle :: Maybe Text
  , entryPubdate :: UTCTime
  }
  deriving (Show, Eq, ...)

data User = User
  { userName :: Text }
```

Routes

config/routes:

```
/home HomeR GET
/entries EntriesR GET POST
/entries/#EntryId EntryR GET POST DELETE
!/entries/new EntryNewR GET POST
```

Controller (Haskell)

```
getEntryR :: EntryId -> Handler Html
getEntryR entryId = do
  entry <- runDB $ get404 entryId
  author <- runDB $ get (entryAuthor entry)
  defaultLayout $(widgetFile "entry")
```

View (Shakespeare)

```
<div #entry-#{entryId entry}>
  $maybe title <- entryTitle entry
    <h1>#{title}
  $nothing
    <div .no-title>
  <p .by-line>
    By #{userName author} on #{fmtDate $ entryPubdate entry}
  <article>
    #{markdownToHtml $ entryContent entry}
```

Forms

Displaying a form:

```
entryForm :: AForm Handler Entry
entryForm = Entry
  <$> areq textField "Title" Nothing
  <*> areq markdownField "Article" Nothing


getEntryNewR :: Handler Html
getEntryNewR = do
  (widget, enctype) <- generateFormPost entryForm
  defaultLayout $(widgetFile "entry-new")
```

Forms

Posting a form:

```
entryForm :: AForm Handler Entry
entryForm = Entry
  <$> areq textField "Title" Nothing
  <*> areq markdownField "Article" Nothing


postEntryNewR :: Handler Html
postEntryNewR = do
  ((res, widget), enctype) <- runFormPost entryForm
  (userId, user) <- requireAuthPair
  case res of
    FormSuccess entry -> do
      entryId <- runDB $ insert $ entry {entryAuthor = userId}
      redirect $ EntryR entryId
    _ -> defaultLayout $(widgetFile "entry-new")
```

Forms

entry-new.hamlet

```
<form #new-entry-form .form-control method=post
      action=@{EntryNewR} enctype=#{enctype}>
  ^{widget}
  <button .btn.btn-default>Submit!
```

Widgets

```
addToLibrary :: UserId -> RefId -> Widget
addToLibrary uid refId = do
  libFormId <- newIdent
  mlib <- handlerToWidget $ runDB $ getBy $ UniqEntry uid refId
  let (icon, txt) = case mlib of
        Nothing -> ("oi-plus", "Add to your library") :: (Text,
        Just _ -> ("oi-check", "In your library") :: (Text, Tex
  toWidget [julius|
<JS from next slide>
|]
  toWidget [whamlet|
<form .form-horizontal ##{libFormId}>
  <button .card-text.btn.btn-success.clickable.align-middle typ
    <i .oi.#{icon}.align-middle aria-hidden="true">
    <span .align-middle>
      #{txt}
|]
```

Type safe JS (kinda)

```
$("##{rawJS libFormId}").submit(function(event){
  event.preventDefault();
  $.ajax({
    url: '@{LibraryR}',
    type: 'POST',
    contentType: "application/json",
    data: JSON.stringify({refId: #{toJSON $ fromSqlKey refId}})
    success: function (data) {
      $("##{rawJS libFormId} > button > i").removeClass("oi-plu
      $("##{rawJS libFormId} > button > i").addClass("oi-check'
      $("##{rawJS libFormId} > button > span").text("In your li
    },
    error: function (data) {
      console.warn("Error: " + data);
    },
  });
});
```

Content Negotiation - TypedContent

```
getEntryR :: EntryId -> Handler TypedContent
getEntryR entryId = do
  entry <- runDB $ get404 entryId
  selectRep $ do
    provideRep $ defaultLayout $ [whamlet|#{entry}|]
    provideJson entry
    provideRepType "text/csv" $ Csv.encode entry
```

The Yesod Typeclass

https://www.stackage.org/haddock/nightly-2018-04-03/yesod-core-1.6.2/Yesod-Core.html

Yesod Auth

Yesod ships with authentication:

- OpenID
- OAuth
- Email/Password

Help - Where do I go when I get stuck?

- IRC on Freenode (I'm `bsima`)
  - `#yesod`
  - `#haskell-beginners`
  - `#haskell`
- Yesod book: yesodweb.com/book
- Haskell-cafe / Haskell-beginners mailing lists
- Slides: github.com/bsima/talks