virtual-hom

7 October 2016

Composition with lenses

- Given a *`View a = a -> HTML (Callback a)`*
- And a *`Lens’ b a`*
- Make a `View b`

Types

Elements

```haskell
data Elem cb a

attributes :: Lens' (Elem cb a) (Map Text Text)
content    :: Lens' (Elem cb a) Text
children   :: Lens' (Elem cb a) [Elem cb a]
callbacks  :: Lens' (Elem cb a) (Callbacks cb)
namespace  :: Lens' (Elem cb a) Text
```

```haskell
div & children .~ [
      h1 "Hello, world",
      p & content .~ "I am a paragraph!"]
```

Types
Callbacks

```haskell
data Callbacks cb

blur   :: Lens' (Callbacks cb) (Maybe (GenericEventData -> cb))
...
change :: Lens' (Callbacks cb) (Maybe (ValueChangedData -> cb))
...
```

```haskell
-- cb = Int -> m Int
button & callbacks . click ?~ (\_ i -> return $ i + 1)
```

Types

Components

```haskell
-- | Create a component with internal state `s` and external state `p`
component :: Functor m => s -> ((s, p) -> [Elem ((s, p) -> m (s, p)) ()]) -> Component m p
```

```haskell
counterComp :: Monad m => Component m ()
counterComp = component 0 $ \(state, _) ->
    [row & children .~ [
      p & content .~ ("This button has been clicked " <> (T.pack $ show state) <> " times"),
      btnDefault
        & content .~ "Click"
        & callbacks . click ?~ (\_ (s, p) -> return (succ s, p))
    ]]
```

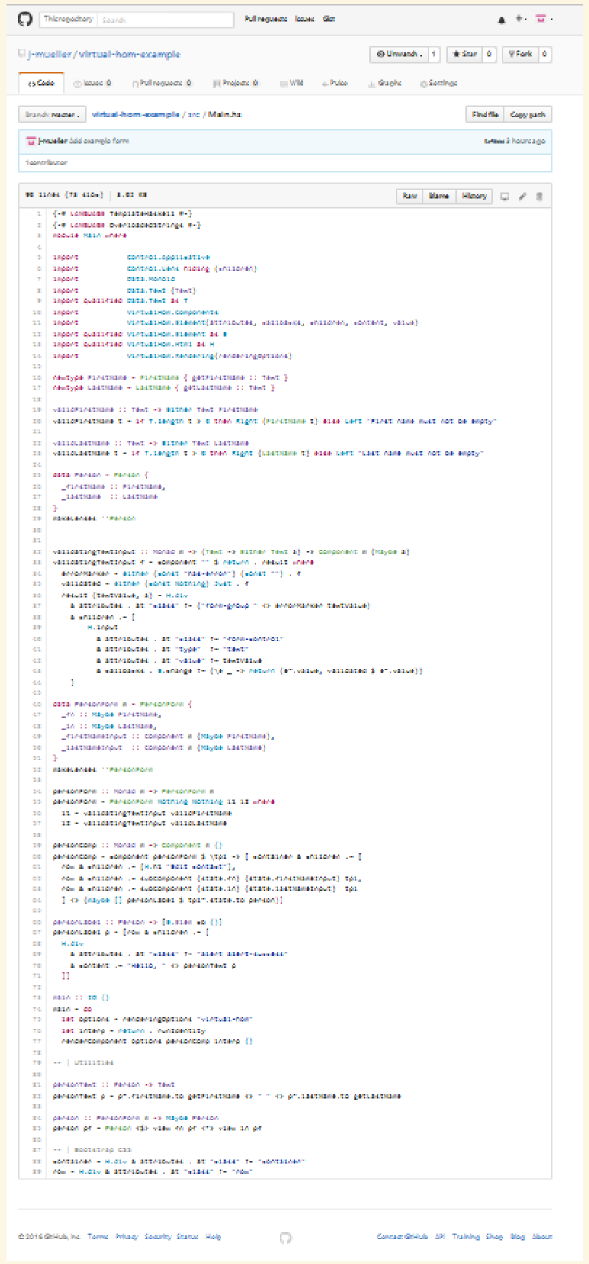
Types

Combining Components

```haskell
-- | Create a component with internal state `s` and external state `p`
component :: Functor m => s -> ((s, p) -> [Elem ((s, p) -> m (s, p)) ()]) -> Component m p

instance Functor m => Monoid (Component m a)

-- | Use a sub-component that is part of the state and modifies the state
subComponent :: Functor f => Lens' a b -> Lens' a (Component f b) -> a -> [Elem (a -> f a) ()]

-- | Change the type of a component using a lens.
specialise :: Functor f => Lens' a b -> Component f b -> Component f a

-- | Show a component only if a prism gets a value
on :: Functor f => Prism' a b -> Component f b -> Component f a
```

… and others

Example


This repository
Search
Pull requests
Issues
Gist
j-mueller / virtual-hom-example
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
Settings
Branch: master
virtual-hom-example / src / Main.hs
Find file
Copy path
j-mueller Add example form
3 hours ago
1 contributor
89 lines (73 sloc) | 3.02 KB
Raw
Blame
History
© 2016 GitHub, Inc. Terms Privacy Security Status Help
Contact GitHub API Training Shop Blog About

```haskell
{-# LANGUAGE TemplateHaskell #-}
{-# LANGUAGE OverloadedStrings #-}
module Main where

import           Control.Applicative
import           Control.Lens hiding (children)
import           Data.Monoid
import           Data.Text (Text)
import qualified Data.Text as T
import           VirtualHom.Components
import           VirtualHom.Element(attributes, callbacks, children, content, value)
import qualified VirtualHom.Element as E
import qualified VirtualHom.Html as H
import           VirtualHom.Rendering(renderingOptions)

newtype FirstName = FirstName { getFirstName :: Text }
newtype LastName = LastName { getLastName :: Text }

validFirstName :: Text -> Either Text FirstName
validFirstName t = if T.length t > 0 then Right (FirstName t) else Left "First name must not be empty"

validLastName :: Text -> Either Text LastName
validLastName t = if T.length t > 0 then Right (LastName t) else Left "Last name must not be empty"

data Person = Person {
  _firstName :: FirstName,
  _lastName  :: LastName
}
makeLenses ''Person


validatingTextInput :: Monad m => (Text -> Either Text a) -> Component m (Maybe a)
validatingTextInput f = component "" $ return . render where
  errorMarker = either (const "has-error") (const "") . f
  validated = either (const Nothing) Just . f
  render (textValue, _) = H.div
    & attributes . at "class" ?~ ("form-group " <> errorMarker textValue)
    & children .~ [
        H.input
          & attributes . at "class" ?~ "form-control"
          & attributes . at "type"  ?~ "text"
          & attributes . at "value" ?~ textValue
          & callbacks . E.change ?~ (\e _ -> return (e^.value, validated $ e^.value))
    ]

data PersonForm m = PersonForm {
  _fn :: Maybe FirstName,
  _ln :: Maybe LastName,
  _firstNameInput :: Component m (Maybe FirstName),
  _lastNameInput  :: Component m (Maybe LastName)
}
makeLenses ''PersonForm

personForm :: Monad m => PersonForm m
personForm = PersonForm Nothing Nothing i1 i2 where
  i1 = validatingTextInput validFirstName
  i2 = validatingTextInput validLastName

personComp :: Monad m => Component m ()
personComp = component personForm $ \tpl -> [ container & children .~ [
  row & children .~ [H.h1 "Edit contact"],
  row & children .~ subComponent (state.fn) (state.firstNameInput) tpl,
  row & children .~ subComponent (state.ln) (state.lastNameInput)  tpl
  ] <> (maybe [] personLabel $ tpl^.state.to person)]

personLabel :: Person -> [H.Elem ev ()]
personLabel p = [row & children .~ [
  H.div
    & attributes . at "class" ?~ "alert alert-success"
    & content .~ "Hello, " <> personText p
  ]]

main :: IO ()
main = do
  let options = renderingOptions "virtual-hom"
  let interp = return . runIdentity
  renderComponent options personComp interp ()

-- | Utilities

personText :: Person -> Text
personText p = p^.firstName.to getFirstName <> " " <> p^.lastName.to getLastName

person :: PersonForm m -> Maybe Person
person pf = Person <$> view fn pf <*> view ln pf

-- | Bootstrap CSS
container = H.div & attributes . at "class" ?~ "container"
row = H.div & attributes . at "class" ?~ "row"
```

Resources

- Github: https://github.com/j-mueller/virtual-hom
- Example app: https://github.com/j-mueller/virtual-hom-example

- Related work
  - https://arianvp.me/lenses-and-prisms-for-modular-clientside-apps/
  - https://github.com/zrho/purescript-optic-ui