My first attempt at producing something remotely close to my Propositional Checker website. A modified version of the RqDataUpload.hs source from the Happstack Crash Course.

The advice given to me is that I need to have two pages generated. A ServerPart Response is a page. One page will setup the form to collect a proposition from the student and post it, and the next page will parse that information and generate a Response page, which will display whether the proposition is correct or incorrect, and possibly provide feedback.

Possibly use 'lookInput' with the RqData monad. It actually turned out that I could just use 'look'.

I will need to process the string I get and then display the result.

```
{-# LANGUAGE OverloadedStrings #-}
import Control.Monad                          (msum)
import Happstack.Server                       (Response, ServerPart,
                                              Method(GET, POST), methodM
                                              , defaultBodyPolicy, dir, getDataFn
                                              , look, lookInput, fileServe, nullDir
                                              , notFound
                                              , nullConf, ok, simpleHTTP, toResponse
                                              , seeOther)
import Text.Blaze                             as B
import Text.Blaze.Html4.Strict                as B hiding (map)
import Text.Blaze.Html4.Strict.Attributes as B hiding (dir, title)

import PropChecker as T
import PropParser as P
```

For now I will import my tautology check until I can further understand how happstack works.

```
main :: IO ()
main = simpleHTTP nullConf $ propcheck
```

For 'dir "feedback"' has methodM POST attached to it in order match on the the specific HTTP request, and if it does match, then produce the page. This way, anyone trying to make a request on 'feedback' will in this case go back to the home page. Alternatively, we can nest another msum in 'dir "feedback"' and generate an error message that refers to specifically to this case. I have taken this approach because it allows me to decide if in the future I decide to remove the 'errorPage' or add more possibilities simply by adding or subtratcing elements from the nested msum.

nullDir will check if the path is non-empty, and if it is, it the handler will move onto the next item, which is notExist, my personalized 404 message. I prefer my own, because I believe it can add a sense of user-friendliness.

It's interesting to note, that since I am using overloaded strings to make life easier using blazeHtml, the string "/" needs to have it's type declared explicitly because it doesn't know which type to choose. Another way around this is to move propCheck into separate module, in order to avoid having to declare the types explicitly, and this will be preferred when the website becomes more complex, in order to have cleaner code.

```
propcheck :: ServerPart Response
propcheck =
    msum [ dir "feedback" $ msum [ methodM POST >> feedback
                                 , errorPage
                                 ]
         , dir "static" $ fileServe [] "."
         , nullDir >> propForm
         , notExist
         ]
```

– , seeOther (”/” :: String) (toResponse (”/” :: String))

```
propForm :: ServerPart Response
propForm = ok $ toResponse $
    html $ do
      B.head $ do
        title "Propositional Equivalance Checker"
      B.h1 $ do
        text "Peter's Mega Ultra Propositional Equivalence Checker"
      B.body $ do
        p (string $ question1)
        p "Given:"
        ul $ li $ text "R1L is not equivalent to R1T"
        ul $ li $ text "R2L is not equivalent to R2T"
      B.div $ do

      B.div $ do
        p (string $ instructions)
        form ! enctype "multipart/form-data" ! B.method "POST" ! action "/feedback" $ d
             input ! type_ "text" ! name "user_prop" ! size "40" ! maxlength "40"
             input ! type_ "submit" ! name "check_prop" ! value "Check this proposition
             input ! type_ "reset" ! name "clear_prop" ! value "Clear"

instructions :: String
instructions =  "Enter a proposition that describes this situation. "

question1 :: String
question1 =  "[...] the king explained to the prisoner that each of the two "
          ++ "rooms contained either a lady or a tiger, but it could be that "
          ++ "there were tigers in both rooms, or ladies in both rooms, or "
          ++ "then again, maybe one room contained a lady and the other room "
          ++ "a tiger."
```

So far, I know how to get data from a form and display it, as well as do any necessary calculations that gives me a value to display. My next problem is getting the error message from the Parser in case the user enters a string that can't be parsed, and displaying this error message to the user, which should be helpful to the user.

Current Situation:

1. If a string that can't be parsed is encountered then 'feedback' does not load at all.
2. I still need to organize the layout of my page and how responses for each case will be handled

```
feedback :: ServerPart Response
feedback =
   do r <- getDataFn (defaultBodyPolicy "/tmp/" 1000 1000 1000) $ look "user_prop"
      ok $ toResponse $
         html $ do
           B.head $ do
             title "Prop Feedback"
           B.h2 $ do
             text "Feedback on given Proposition"
             img ! src "/static/lambda.gif" ! alt "lambda" ! width "40" ! height "40"
           B.body $ do
             mkBody r
    where
      mkBody (Left errs) =
          do p $ "The following error occurred:"
             mapM_ (p . string) errs
      mkBody (Right theprop) = do
                B.h3 $ do
                  text "Analysis"
                isEquivalent (P.evalProp theprop prop1 rests1) theprop

isEquivalent               :: String -> String -> Html b
isEquivalent "True"  prop  =  p (string $ "Your proposition '" ++ prop ++
                                "' correctly describes this situation.")
isEquivalent "False" prop  =  do
                                p (string $ "Your proposition '" ++ prop ++
                                 "' incorrectly describes this situation.")
                                B.h4 $ text "Here's a tip: "
                                p (string $ "When " ++ evalDisagree prop prop1 rests1 +
                                 "Carefully look over the " ++
                                 "information that is being given to you and try again.
isEquivalent error   prop  =  do
                                p (string $ ("'" ++ prop ++ "' is not a correctly" ++
                                  " written proposition."))
                                B.h4 $ text "Check "
                                p (string $ betterError error)



-- Enter your restrictions here
rests1 :: Rests
rests1 =  [ Not (Equiv (Var 'a') (Var 'c'))
          , Not (Equiv (Var 'b') (Var 'd'))
          ]

prop1 :: String
prop1 =  "(a v b) & (c v d)"

errorPage :: ServerPart Response
errorPage = ok $ toResponse $
    html $ do
      B.head $ do
        title "Error Page"
      B.h1 $ do
        text "Oops!"
      B.body $ do
        p $ "It seems you tried to check your feedback without submitting a proposition
      B.div $ do
        p $ ""

notExist :: ServerPart Response
notExist = notFound $ toResponse $
    html $ do
      B.head $ do
        title "Error Page"
      B.h1 $ do
        text "Sorry!"
      B.body $ do
        p $ "This page doesn't exist, maybe you can ask for it to be in the next versio
      B.div $ do
        p $ ""


-- This provides a better error message
betterError             :: String -> String
betterError err         =  drop 8 (filter (/= ')') err)
```