Full Stack Functional - Haskell
Functional Programming

Jens Egholm Pedersen and Anders Kalhauge

Spring 2018

Setup

Clone the `exercise-haskell` repository

```
$ git clone https://github.com/
  cphbus-functional-programming/exercise-haskell.git
```

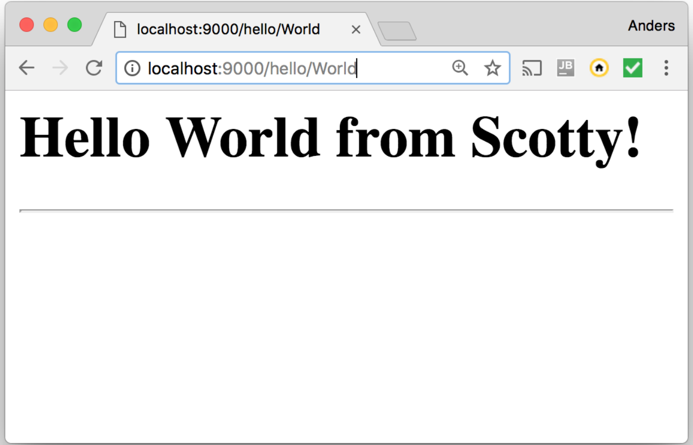
Run the haskell server:

```
$ cd exercise-haskell
$ stack setup
$ stack build
$ stack exec exercise-haskell-exe
```

The server starts listening on port 9000

Check Server

Start a browser and write `http://localhost:9000/hello/World` in the address:

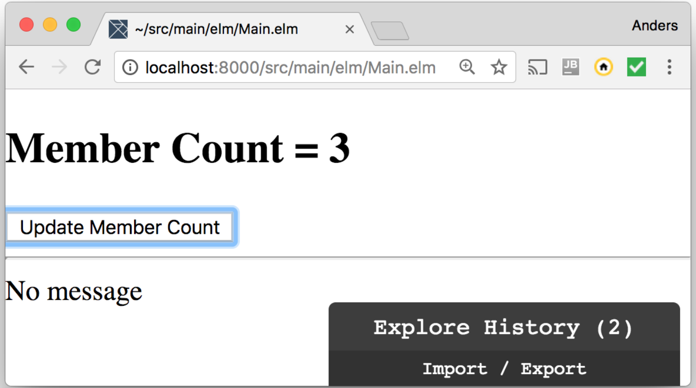


Installing Elm packages

Open a new terminal in the directory of the elm frontend, and start elm reactor:

```
$ cd ../exercise-elm
$ elm-reactor
```

In the browser address field write `http://localhost:8000/src/main/elm/Main.elm` and press the button[1]:



Start coding

Open your favourite editor, ie:

```
$ cd ../exercise-haskell
$ atom .
```

Open the `Main.hs` file. It is in the `src/` folder.

The initial Server code

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

import Network.Wai.Middleware.Cors
import Web.Scotty

main :: IO ()
main = do
  scotty 9000 $ do
    middleware simpleCors
    get "/hello/:name" $ do
      name <- param "name"
      html $ mconcat [ "<h1>Hello "
                     , name
                     , " from Scotty!</h1><hr/>"
                     ]
```

Exercise 1 - Create a member type

Create a `Member` data type. The member should have the same fields as the Member in the Elm application.

The `Member` data type should derrive the type classes `Show` and `Generic`. Remember to add the pragma and imports:

```haskell
{-# LANGUAGE DeriveGeneric #-}
module Main where

import GHC.Generics
import Data.Aeson (FromJSON, ToJSON)
```

in the top of `Main.hs`.

Make the `Member` data type an instance of `ToJSON` and `FromJSON`.

Exercise 2 - Tidying up

Add the following import:

```
import Prelude hiding (id)
```

to prevent name clashes between Preludes `id` (identity) function and our new `id` function.

Explain where that new `id` function is declared.

Exercise 3 - Insert member function

Create a function to insert new members in an `IntMap`. If the member is in the map, it should be updated. If the member is not in the map, the member's `id` should be set to the size of the map plus one.

The function shall have the following signatue:

```
import Data.IntMap (IntMap)
import qualified Data.IntMap.Strict as IntMap

insertMember :: Member
             -> IntMap Member
             -> (Member, IntMap Member)
```

Exercise 4 - A Mutable Variable

Create a mutable variable `MVar` to hold an `IntMap` of `Member`s. Call it `membersRef`.

You will need the following:

```
import Control.Concurrent ( newMVar
                          , readMVar
                          , takeMVar
                          , putMVar
                          )
```

cphbusiness COPENHAGEN BUSINESS ACADEMY

Exercise 5 - Get the member count

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Create an endpoint for the rest `GET` method with the url: `/member/count`. You should use `readMVar` to get the members from the `membersRef` variable.

```
ghci> :t readMVar
readMVar :: MVar a -> IO a
ghci> :t get
get :: RoutePattern -> ActionM () -> ScottyM ()
```

In other words: `get` expects an `ActionM` monad, but `readMVar` returns an `IO` monad, use `lift` to change type:

```
import Control.Monad.Trans.Class (lift)
...
  get "/member/count" $ do
    members <- lift $ readMVar membersRef
```

Exercise 6 - Get all members

Create an endpoint for the rest `GET` method with the url: `/member`. All members should be returned as a JSON list. You can use `IntMap.elems` to get the values (not keys) from an `IntMap`.

Exercise 7 - Get one member

Create an endpoint for the rest `GET` method with the url: `/member/:id`. The member with the id should be returned as a JSON object. You can use the `read` function to convert a string to an integer.

```
let id = (read idText) :: Int
```

Exercise 8 - Posting member

Create an endpoint for the rest `POST` method with the url: `/member`. Use the `insertMember` function to insert the member, and return the member, possibly with a new `id`.

The `MVar` reference is locked between calls to the `takeMVar` and the `putMVar` functions

What next?

- Ability to show a list of members.
- Ability to delete members.