**Programarea web cu Haskell folosind framework-ul Spock**



Spock este un framework web Haskell care vine cu suport pentru rutare, middleware, json, blaze, sesiuni, cookie-uri, asistent pentru baze de date, protecție CSFR, stare globală

**Instalare Spock (versiunea curentă 0.13.0.0)**

Acesta poate fi instalat accesând link-ul <https://hackage.haskell.org/package/Spock>

Sau prin intermediul altor programe:

*Stack*

Link instalare stack: <https://docs.haskellstack.org/en/stable/README/>

Comanda instalare Spock prin intermediul stack: **stack install Spock**

*Cabal*

Link instalare cabal: <https://www.haskell.org/cabal/download.html>

Comanda instalare Spock prin intermediul cabal: **cabal install Spock**

Capabilitățile framework-ului Spock vor fi exemplificate printr-o aplicație, Bibliotecă, ce permite operații CRUD asupra unei liste de cărți. ([github repository](https://github.com/TheodorBogdan/haskell-web-app))

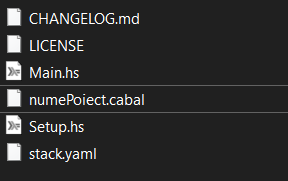
**Creare proiect nou**

Apelare comenzi:

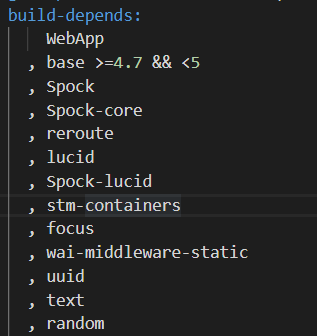
cabal init

stack init

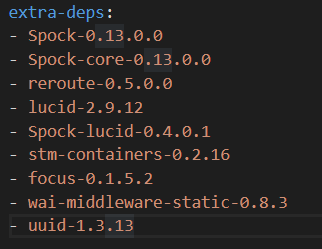
În urma rulării celor două comenzi, structura proiectului va arăta astfel:



În fișierul cu extensia .cabal sunt stocate informații despre proiect, printre care dependințele proiectului la build.



În fișierul stack.yaml reținem referințele către pachetele de care avem nevoie împreună cu versiunile aferente, depedințe ce nu pot fi rezolvate direct la build.



În **Main.hs**, pentru a porni server-ul importăm **Web.Spock** și **Web.Spock.Config** și rulăm funcția runSpock ce are semnătura:

**runSpock :: Port -> IO Middleware -> IO ()**

main :: IO ()

main = do

**runSpock 8080 (spock cfg app)**

**app** este de tip **SpockM conn0 sess0 st0 ()** și reprezintă modul de definire al unei aplicații în Spock.

Tipul SpockM acceptă 4 parametrii:

* Conexiunea la baza de date (conn)
* Tipul sesiunii (sess)
* Tipul stării (st)
* Tipul valorii returnate de monadic

**cfg** este de tip **SpockCfg conn0 sess0 st0**

Acesta poate fi inițializat cu **defaultSpockCfg** care acceptă o configurație de sesiune, un pool sau o conexiune la baza de date și o stare inițială.

cfg <- defaultSpockCfg () PCNoDatabase st

**Definirea rutelor**

Vom defini un tip alias, Server

type Server a = SpockM () () () a

app :: Server ()

app = get root (text "Hello!")

**Randare html**

app = get root (html "<h1>Hello!</h1>")

Folosirea pachetului **Lucid**, un DSL pentru HTML

import Web.Spock.Lucid (lucid)

import Lucid

app = get root $ lucid $ do

h1\_ "Heading example"

p\_ "Paragraph example"

**ServerState**

Vom defini un Server state care ne va ajuta în a menține starea obiectelor

**newtype ServerState = ServerState { books :: IORef [Book] }**

Tipul Server devine acum:

**type Server a = SpockM () () ServerState a**

ServerState trebuie asignat de asemenea configurației

state <- ServerState <$> newIORef [

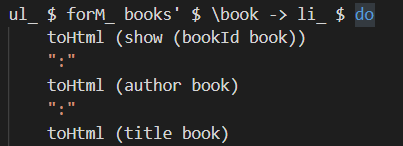
Book 1 "Author\_1" "Title\_1",

Book 2 "Author\_2" "Title\_2"

]

cfg <- defaultSpockCfg () PCNoDatabase state

Definirea de form-uri:



import Control.Monad (forM\_)

Citire date din state:

import Control.Monad.IO.Class (liftIO)

books' <- getState >>= (liftIO . readIORef . books)

Adăugarea de noi elemente se poate realiza prin verbul post.

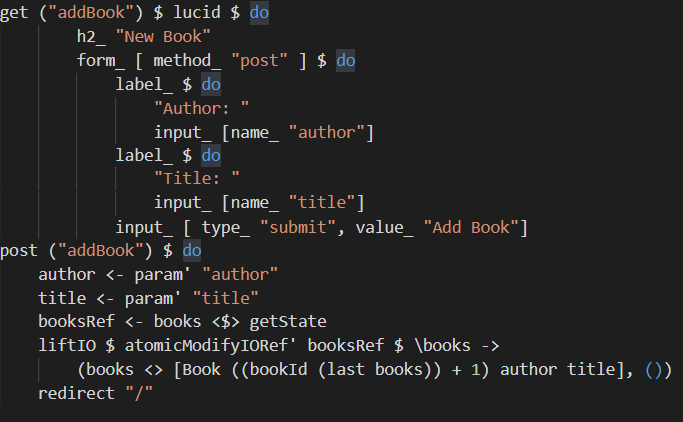
Funcția param’ este utilizată pentru a extrage valorile din câmpuri.

param' :: forall p (m :: \* -> \*) ctx.

(FromHttpApiData p, MonadIO m)

=> Text

-> ActionCtxT ctx m p



Funcția atomicModifyIORef’ este folosită pentru a actualiza lista de cărți

**atomicModifyIORef' :: IORef a -> (a -> (a, b)) -> IO b**