Contents

1 Introdução

Neste tutorial faremos um blog usando o Yesod Frameword. O Yesod é um Haskell web Framework para o desenvolvimento produtivo de RESTful, aplicações web de tipo seguro e de alta performance.

1.1 Começando..

1.1.1 Pacotes a serem instalados

- psql (postgres)
- stack
- yesod
- yesod-bin

1.2 Criando um novo projeto

Com tudo devidamente instalado, criamos um novo projeto usando o Stack, cuja a sintaxe é *stack new <project name> <template>*. Para esse caso:

- *stack new myBlog yesod-sqlite && cd myBlog*

Para construir as bibliotecas

- *stack exec – yesod devel*

Agora você pode testar sua aplicaçando em `http://localhost:3000/`. Após isso, criamos uma sandbox para nosso projeto usando o Cabal:

- *cabal sandbox init*

Agora instalamos apenas as dependencias necessárias para nossa aplicação:

- *cabal install –only-dependencies*

Em seguida podemos criar nossa aplicação e testá-la:

- *cabal build*
- *yesod-devel*

1.3 Criando Rotas

Antes de irmos a frente, criamos um novo git repo.

- *git init*
- *git add .*
- *git commit -m "myBlog" .*

Agora estamos prontos para seguir em frente. A primeira coisa que temos que fazer para uma aplicação de blog, é uma página que nos permita criar blogposts. Uma página que nos permita criar amostras de dados e ver como o yesod cria rotas. Não faremos tudo na mão, então no terminal:

- *yesod add-handler*

Isto abrirá um menu, para esse caso respondemos:

- PostNew
- /post/new
- GET

Podemos ver que houve algumas mudanças com o comando:

- *git status*

Agora estamos prontos para continuar com este Handler. Handler/PostNew.hs e editamos o que está lá para:

```
module Handler.PostNew where

import Import

getPostNewR :: Handler Html
getPostNewR = do
  defaultLayout $ do
    $(widgetFile "posts/new")
```

Em seguida criaremos uma nova página, um template. Criamos uma nova pasta dentro de templates. templates/posts. A seguir criamos um documento em equivalente ao html, o hamlet. **new.hamlet**
Para testarmos, editamos para:

```
<h1> Create a post!
```

Interprete o Handler e pode-se testar no browser. `http://localhost:3000/posts/new`

"Insert copyright statement here" não é necessáriamente requerido. Isso vem da função *defaultLayout* que chama o dafault-layout-wrapper.hamlet e lá há um campo que está especificado o seguinte:

```
<body>
   <div class="container">
     <header>
     <div id="main" role="main">
       ^{pageBody pc}
     <footer>
       #{appCopyright $ appSettings master}
```

Então é só apagar a seguinte linha:

#{appCopyright $ appSettings master}

1.4 Criando Forms

Fazemos algumas mudanças no Handler:

```
module Handler.PostNew where

import Import
import Yesod.Form.Bootstrap3

data BlogPost = BlogPost
{ title :: Text
}

blogPostForm :: AForm Handler BlogPost
blogPostForm = BlogPost
    <$> areq textField (bfs ("Title" :: Text)) Nothing

getPostNewR :: Handler Html
getPostNewR = do
    (widget, enctype) <- generateFormPost $ renderBootstrap3 BootstrapBasicForm blogPos
    defaultLayout $ do
$(widgetFile "posts/new")
```

E também algumas mudanças no template:

```
<h1> Create a post!

<form method=post action=@{PostNewR} enctype=#{enctype}>
  ^{widget}
  <button class="btn btn-default">Create Post!
```

Agora precisamos de algo para gerar novos blogposts. Adicionaremos dois pacotes em nosso myBlog.cabal. Na guia **build-depends**, ao fim dela adicione:

, markdown
, yesod-text-markdown

Agora adicionaremos mais alguns recursos em nosso Handler:

```
module Handler.PostNew where
```

```
import Import
import Yesod.Form.Bootstrap3
import Text.Markdown (Markdown)
import Yesod.Text.Markdown

data BlogPost = BlogPost
{ title :: Text
, article :: Markdown
}

blogPostForm :: AForm Handler BlogPost
blogPostForm = BlogPost
    <$> areq textField (bfs ("Title" :: Text)) Nothing
    <*> areq markdownField (bfs ("Article" :: Text)) Nothing

getPostNewR :: Handler Html
getPostNewR = do
    (widget, enctype) <- generateFormPost $ renderBootstrap3 BootstrapBasicForm blogPos
    defaultLayout $ do
$(widgetFile "posts/new")
```

Para deixarmos o campo de "Articles" um pouco maior, então criamos um novo documento css (lucius). Seu caminho é /templates/posts/new/new.lucius

```
textarea {
    min-height: 400px;
  }
```

1.5 Definindo rotas onde os blogpost serão gerados

Começamos adicionando um método para o construtor *PostNewR*. Seu caminho é, /config/routes.

/ *posts/new PostNewR GET POST*

Então agora implementaremos esse construtor em nosso Handler. Vamos até /config/models e lá criaremos um novo modelo para nosso blogpost.

```
BlogPost
```

```
    title Text
    article Markdown
```

Então adicionamos as bibliotecas necessárias para nosso modelo em /config/Model.hs

```
import Text.Markdown (Markdown)
import Yesod.Text.Markdown ()
```

Podemos agora remover nosso *data BlogPost* para implementar o nosso construtor, que será salva em um banco de dados. Nosso Handler ficará assim:

```
module Handler.PostNew where

import Import
import Yesod.Form.Bootstrap3
import Yesod.Text.Markdown

blogPostForm :: AForm Handler BlogPost
blogPostForm = BlogPost
    <$> areq textField (bfs ("Title" :: Text)) Nothing
    <*> areq markdownField (bfs ("Article" :: Text)) Nothing

getPostNewR :: Handler Html
getPostNewR = do
    (widget, enctype) <- generateFormPost $ renderBootstrap3 BootstrapBasicForm blogPos
    defaultLayout $ do
$(widgetFile "posts/new")

postPostNewR :: Handler Html
postPostNewR = do
  ((res, widget), enctype) <- runFormPost $ renderBootstrap3 BootstrapBasicForm blogPos
  case res of
    FormSuccess blogPost -> do
      _ runDB $ insert blogPost
      error "TODO"
       _ -> defaultLayout $(widgetFile "posts/new")
```

Agora criaremos uma página que nos permite criar blogposts. No terminal digitamos:

- yesod add-handler
- PostDetails
- /posts/#BlogPostId
- GET

  Isso irá gerar um problema de overlapping, para solucionar vamos para /config/routs e editamos a seguinte linha como a seguir:

- *posts*!#BlogPostId PostDetailsR GET

  Agora editaremos o construtor *PostDetails.hs* para que a página salve as postagens.

```
module Handler.PostDetails where

import Import

getPostDetailsR :: BlogPostId -> Handler Html
getPostDetailsR blogPostId = do
  blogPost <- runDB $ get404 blogPostId
  defaultLayout $ do
    $(widgetFile "postDetails/post")
```

Criamos também uma nova pasta e um novo arquivo para essas manipulações. /templates/postDetails/post.hamlet. E lá digitamos.

```
<h1>#{blogPostTitle blogPost}

<article .jumbletron>
  #{blogPostArticle blogPost}
```

Em PostNew.hs queremos que o usuário seja redirecionado quando inserirmos um blogpost, redirecionado-o para página de postagens.

```
postPostNewR :: Handler Html
postPostNewR = do
  ((res, widget), enctype) <- runFormPost $ renderBootstrap3 BootstrapBasicForm blogPos
```

```
  case res of
    FormSuccess blogPost -> do
      blogPostId <- runDB $ insert blogPost
      redirect $ PostDetailsR blogPostId
    _ -> defaultLayout $(widgetFile "posts/new")
```

Agora indexaremos tudo na página inicial, para isso editamos o arquivo /config/routes.

- / HomeR GET

Em seguida editamos o arquivo Home.hs para que apareça todos os posts feitos.

```
module Handler.Home where

import Import

-- This is a handler function for the GET request method on the HomeR
-- resource pattern. All of your resource patterns are defined in
-- config/routes
--
-- The majority of the code you will write in Yesod lives in these handler
-- functions. You can spread them across multiple files if you are so
-- inclined, or create a single monolithic file.
getHomeR :: Handler Html
getHomeR = do
    allPosts <- runDB $ selectList [] [Desc BlogPostId]
    defaultLayout $ do
$(widgetFile "posts/index")
```

Feito isso, falta agora criar apenas o hamlet da página inicial. Seu caminho é /templates/posts/index.hamlet

```
<h1> All Posts

<div .jumbotron>
  <ul>
    $forall Entity id post <- allPosts
      <h4>
<li>
  <a href=@{PostDetailsR id}>#{blogPostTitle post}
```

Com isso concluimos nosso tutorial.