CS 421 Programming Languages

Final Project

Michael Neill Hartman

mnh5@illinois.edu

NetID: **mnh5**

GitHub repo: **https://github.com/hartmanm/hs_server**

---

hs_server

A bash script builds a docker container with Haskell, mounts a directory, and runs a local webserver from the container with content pulled from the mounted files / directories and the project/src/routes file

Overview

motivation

I frequently find myself interacting with filesystems on different machines using a combination of various linux command line tools, such as ls, du, diff, grep, sed, cp, tar. I feel my use frequently falls into specific patterns like: scp/rsync a directory or archive to a machine, find a configuration file, edit the file with information from another configuration file, run an executable. This becomes more complicated as usernames, system paths, multiple matching filenames and other complexities often cause typical attempts at simplification to fail with edge cases. What if there was a tool that could simplify such tasks by leveraging Haskell, producing a simple local web server interface?

---

goals

As Haskell has powerful generic processing capabilities it seems like a good fit to implement this kind of a visualized simplification/automation tool. I thought it would allow me to explore existing Haskell libraries and tools, and produce a tool that affords some useful functionality, showcasing the capabilities afforded by functional programming. My goals were:

- research existing Haskell resources and libraries
- Implement some minimal basis of filesystem walking/ingesting concepts
- Implement basic minimum functional Haskell localhost web server visualization / interface
- create documentation / git repository for the project

reach goals

- Put the project inside a docker container, and allow for volume mapping to the container to allow docker required portability for the project
- Allow for metaprogramming with the web interface

accomplishments

I was able to achieve all of my goals and one of my reach goals with this project. If you have docker installed, you can clone the repo on any computer running macOS or linux, cd to hs_server/project , and run : bash hs_server

This should automatically build the docker container and execute the hs_server mounting the repo directory as a volume. Another directory can be mounted by passing its fullpath as a parameter.

If you open: http://localhost:3001 in a browser you can interact with the generated content.

Implementation

(a) the major tasks or capabilities of your code;

When run without a parameter, bash hs_server will mount the repo volume and add the default routes in the src/routes file as seen below:

goals

- research existing Haskell resources and libraries
- I researched and tested out many different libraries before finding ones that I was able to create a working implementation with. I was surprised how many libraries there are for Haskell.
- Implement some minimal basis of filesystem walking/ingesting concepts
- In Main.hs I have implemented functions that use Haskell libraries to walk the filesystem and create a map of all the file paths and the char representation of their content.
- Implement basic minimum functional Haskell localhost web server visualization / interface
- This map I call the: html_content_map. I used a Pretty Print library to render the content of this map as a html page of html_links, each of which is a link to a route serving the files contents.
- The html_content_map also contains links and content added to the src/routes file which allows adding routes and html content without mounting a directory
- create documentation / git repository for the project
- This report and the github repo, along with the comments in the code constitute this.

reach goals

- Put the project inside a docker container, and allow for volume mapping to the container to allow docker required portability for the project

- A bash script builds a docker container with Haskell, mounts a directory, and runs a local webserver from the container with content pulled from the mounted files / directories and the project/src/routes file

```
open a web browser and go to :  http://localhost:3001

[+] Building 0.0s (17/17) FINISHED                                                docker:desktop-linux
 => [internal] load .dockerignore                                                                0.0s
 => => transferring context: 2B                                                                  0.0s
 => [internal] load build definition from Dockerfile                                             0.0s
 => => transferring dockerfile: 757B                                                             0.0s
 => [internal] load metadata for docker.io/library/ubuntu:jammy                                  0.0s
 => [with_haskell 1/4] FROM docker.io/library/ubuntu:jammy                                       0.0s
 => [internal] load build context                                                                0.0s
 => => transferring context: 3.00kB                                                              0.0s
 => CACHED [with_haskell 2/4] RUN    apt-get update -y                                           0.0s
 => CACHED [with_haskell 3/4] RUN    apt-get install haskell-platform -y                         0.0s
 => CACHED [with_haskell 4/4] RUN    cabal update                                                0.0s
 => CACHED [with_dependencies 1/7] RUN    mkdir -p                      /server/app              0.0s
 => CACHED [with_dependencies 2/7] COPY   src/server.cabal              /server                  0.0s
 => CACHED [with_dependencies 3/7] COPY   src/initial_main              /server/app/Main.hs      0.0s
 => CACHED [with_dependencies 4/7] RUN    cd /server;                   cabal install            0.0s
 => CACHED [with_dependencies 5/7] COPY   src/run_hs_server             /                        0.0s
 => CACHED [with_dependencies 6/7] RUN    chmod 755                     /run_hs_server           0.0s
 => CACHED [with_dependencies 7/7] COPY   src/Main.hs                   /server/app/Main.hs      0.0s
 => CACHED [with_runtime_config 1/1] COPY   generated/modified_routes    /server                 0.0s
 => exporting to image                                                                           0.0s
 => => exporting layers                                                                          0.0s
 => => writing image sha256:cc388e31e08e9c65e781f95d19fce1ae325f4e9f2c8e046f038f441dcfd6bb       0.0s
 => => naming to docker.io/library/hs_server_di:latest                                           0.0s
```

- **What's Next?**
  - Allow for metaprogramming with the web interface
- I would have liked to have added more extensive processing of the files, and allowed for metaprogramming with the web interface by adding post http method support to the hs_server. But this was a reach goal I didn't expect to have time to implement

(b) components of the code;

hs_server : a bash script that triggers building and executing

src/hs_server.dockerfile : dockerfile used to build a ubuntu:jammy based Haskell container

src/initial_main : a placeholder main.hs file used to build the docker image

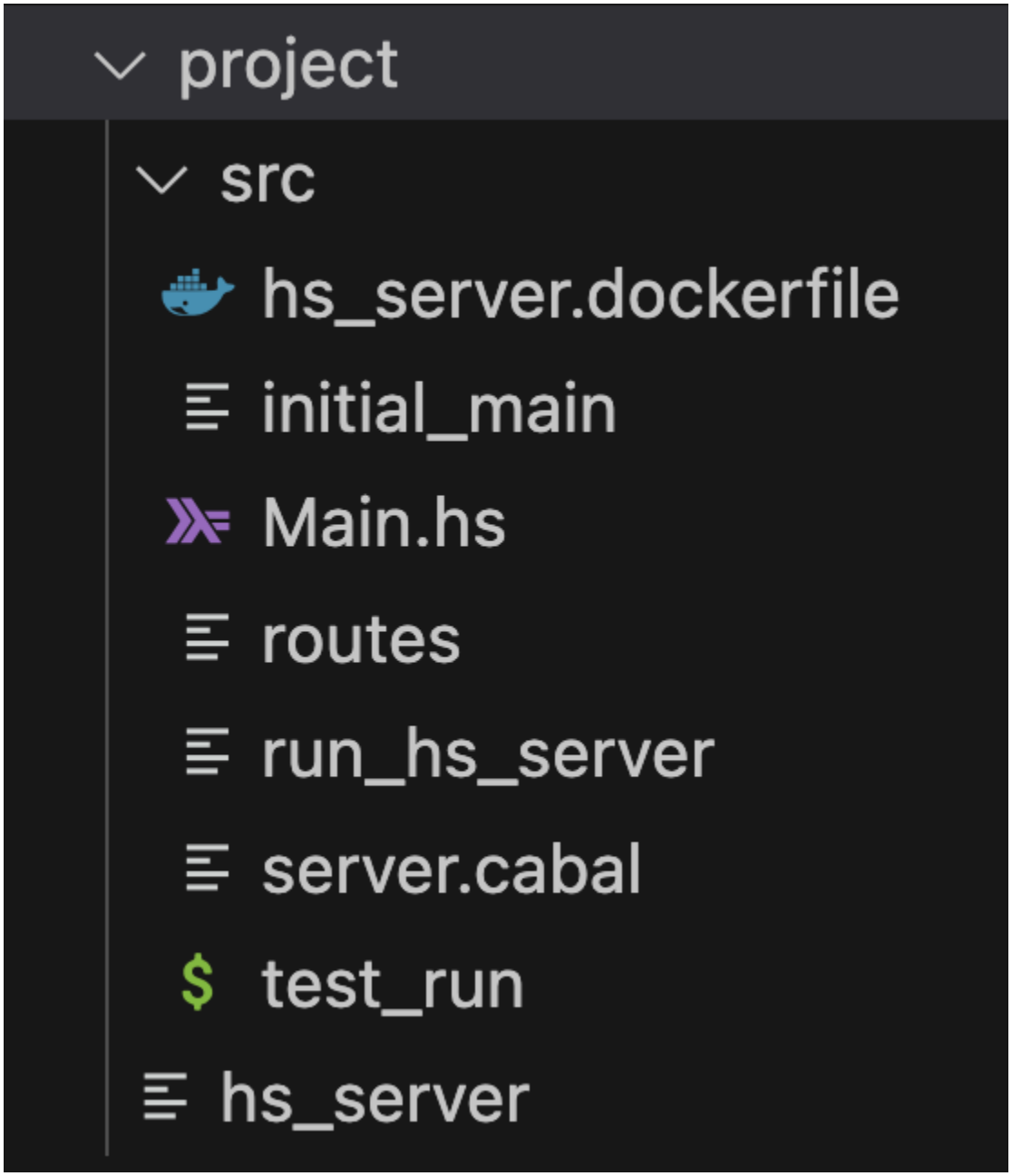
src/Main.hs : the src for the server executable

src/routes : configuration and content file

src/run_hs_server : a docker entrypoint wrapper to launch the server executable

src/server.cabal : Haskell package configuration file

src/test_run : runs tests, mounting the src/tests directory (which is created if not existing)



(c) status of the project -- what works well, what works partially, and and what is not implemented at all

what works well:

- A bash script builds a docker container with Haskell, mounts a directory, and runs a local webserver from the container with content pulled from the mounted files / directories and the project/src/routes file
- hs_server is simple and automated; it is easy to mount a different directory, or add routes to the src/routes file

what works partially:

- In general hs_server works as intended, but it does have limitations. When the executable is generated, the files are processed and incorporated into the executable as the html_content_map. This makes all file html_link interactions very fast, but also prevents very large directories or files from being able to be processed by hs_server. I have not determined exactly what the limits are, but I tested and confirmed that the linker and compiler are not able to incorporate anything too large.
- Another potential limitation is that each binary file is binned to bytes, i.e. converted into its char 0-255 representation; which is what the value of the html_content_map contains as its html_content.

what is not implemented at all:

- I didn't implement css at all for the main page of hs_server. However, it can be implemented as is with routes added to the src/routes file.

Tests

unit tests:

the primary test is building and running by invoking:

```
bash hs_server
```

If this works you can use it.

feature tests:

verify attempting to mount a non existent path:

```
# with no actual mountpoint

bash hs_server <a non existent path>
```

Test using test_run:

src/test_run

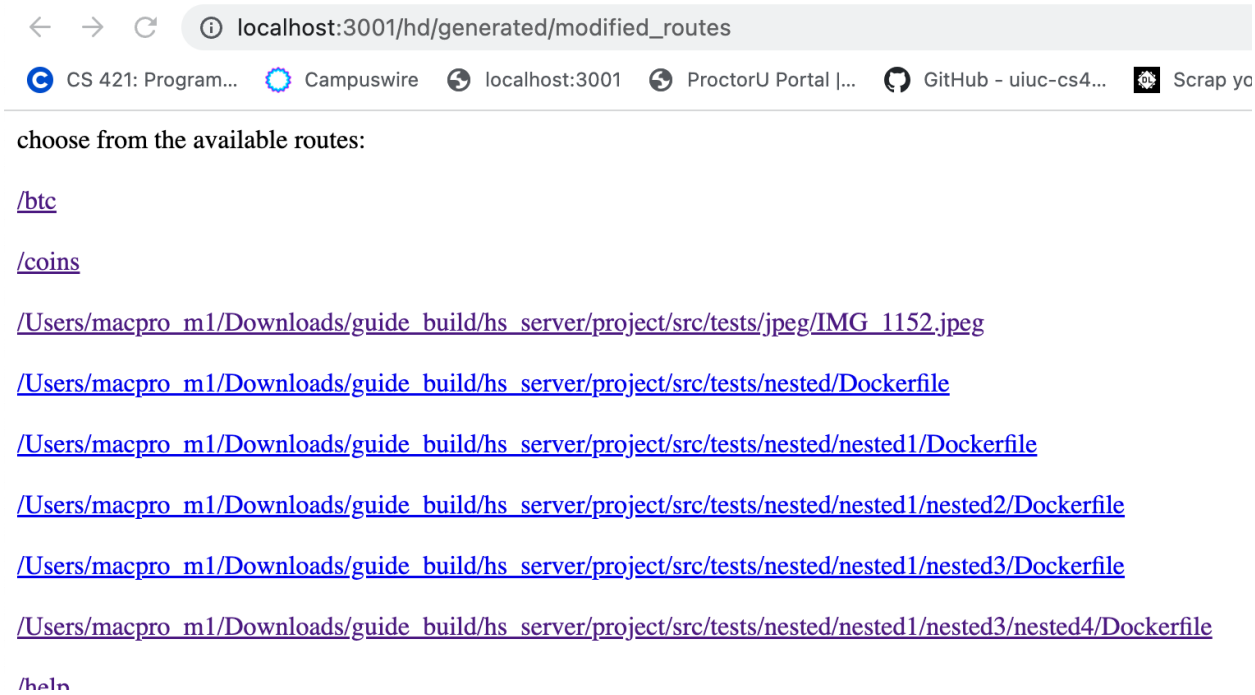
cd hs_server/project/src

bash test_run

This will launch the server mounting the hs_server/project/src/tests directory

Open your browser and click on the link:

This should display something similar to:



This is running the jpeg, nested directories, and empty directories tests.

The jpeg test verifies that a jpeg is converted into its char 0-255 representation

The nested directories test verifies that multiple nested directories are walked correctly and the files in them are served.

The empty directory test verifies that an empty directory will be processed without an error

These tests exercise the concept(s) implemented

Verifying that they operate as expected.

Listing

hs_server :

a bash script that triggers building and executing

requires docker and sed, note wrapper only works on macOS or Linux

will build a docker container and run it mounting the fullpath passed as a parameter. If no parameter is passed the repository directory will be mounted.

src/hs_server.dockerfile :

dockerfile used to build a ubuntu:jammy based Haskell container

When hs_server runs docker build, it copies this file to another file called Dockerfile

And uses export DOCKER_BUILDKIT=1

This caches the build on your machine allowing you to modify the src/routes file or change the mounted volume and rebuild the docker container using the cache

src/initial_main :

a placeholder main.hs file used to build the docker image, allows for the docker build cache to work effectively

src/Main.hs :

the src for the server executable

The core functions are:

```
-- starts with an input path and walks its subdirectories returning all the files

walk_path_recursively :: FilePath -> IO ([(FilePath,[FilePath])],[FilePath])

walk_path_recursively path = do
```

```
-- the modified_routes file which contains

-- <key> <route html content>

-- is parsed using <a space or multiple spaces> as the IFS between <key> <route html
content>

-- and a newline char as the ROUTE_IFS

generate_map_from_routes_html_data :: FilePath -> IO (Map.Map String String)
```

```
-- generate a series of html_links from the html_content_map

-- as the default html (index or home)

-- modifying the internal mount_point to the host path

-- for the links

pp_html_content_map :: Map.Map String String -> PP.Doc
```

```
-- main function for generating html links for each file and processing each files
contents

-- returns an updated html_content_map with the additional generated links and
html_content

add_mounted_directories_and_files_to_html_content_map :: Map.Map String String ->
[(FilePath, [FilePath])] -> IO (Map.Map String String)
```

```
-- either returns the value of a constant k v pulled from

-- the modified_routes file -> html_content_map or

-- renders (displays the Pretty Printed html) the series of html_links from the
html_content_map

-- as the default html (index or home)

get_value_or_render :: B.ByteString -> Map.Map String String -> String
```

This function is called to process the route provided in the server loop:

```
run (get_port html_content_map) $ \req respond -> do

let path = rawPathInfo req

-- serve only (http get method) for the routes contained in the html_content_map

-- note this is not serving by accessing the files,

-- when the executable is generated the files are processed and incorporated

-- into the executable as the html_content_map

-- this makes all file html_link interactions very fast

let response = case requestMethod req of

"GET" -> get_value_or_render path html_content_map

_ -> "Error: unsupported HTTP method!"

respond $ responseLBS status200 [(hContentType, "text/html")] (LBCS.pack response)
```

src/routes :

configuration and content file of the form:

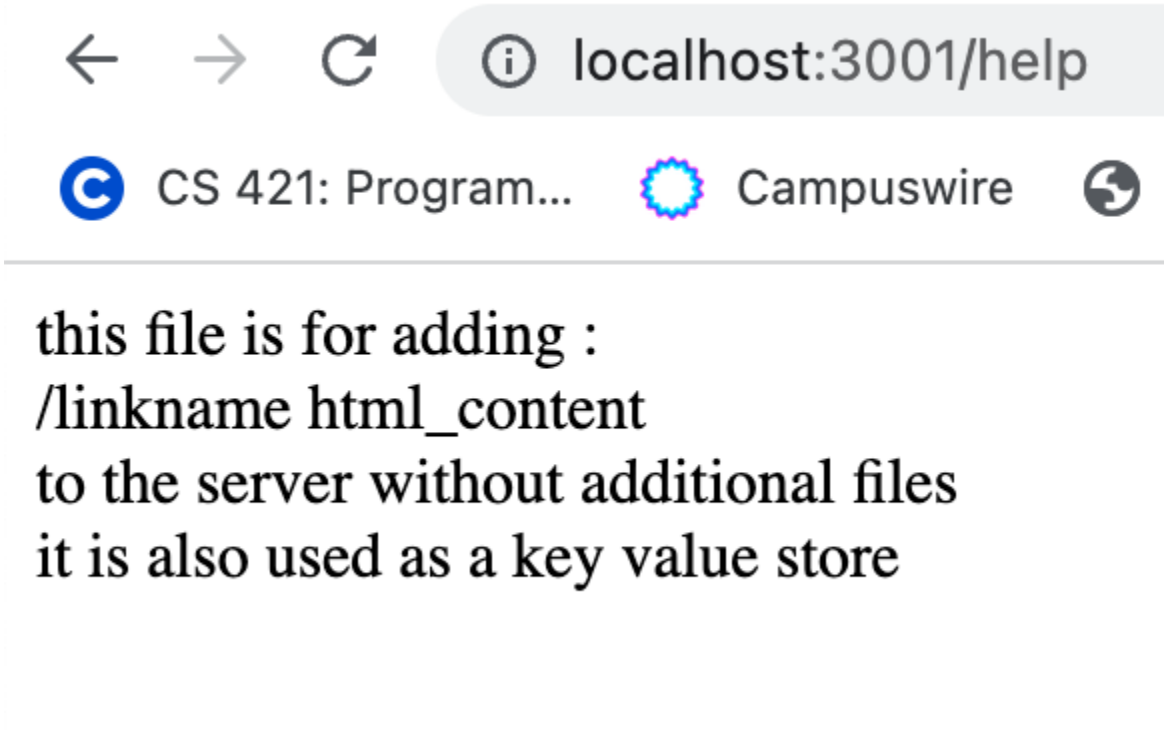
```
--  the modified_routes file which contains
-- <key>  <route html content>
-- is parsed using <a space or multiple spaces> as the IFS between <key>  <route html content>
-- and a newline char as the ROUTE_IFS
```

For example the /help route looks like this in the file:

```
12
13    /help                    this file is for adding : <br>  /linkname
      html_content    <br> to the server without additional files <br> it
      is also used as a key value store
14
```

When hs_server is run this route looks like this:

If you click on the /help route you are redirected to its hosted content:



src/run_hs_server :

a docker entrypoint wrapper to launch the server executable. When docker loads the image as a running container it executes this script, which in turn launches the server executable

src/server.cabal :

Haskell package configuration file with the dependencies for hs_server. These are downloaded and installed to the docker image. It is worth noting that this project could easily be used as a build template for docker based haskell projects.

src/test_run :

runs tests, mounting the src/tests directory as described above in the tests section

Group

I am the only member of this group.