Flamingo Server

Philip Lassen (vgh804), Nicolas Larsen (vgn209)

October 2018

1 Introduction

This was Both our first times working in Erlang. It took us a fair amount of time to get to grasps with how Erlang is structured. It was quite an interesting challenge thinking about everything as processes. A fair portion of our time was spent parsing examples and doing hands on exercises before we felt comfortable working with Erlang. We still ran into many issues when we started our implementations.

2 Implementation

2.1 Part I

Our implementation ended up as a mapping from a path to a function, args tuple. The server retained its global state, and we used the mapping named `LocalState` to save our paths. A new path is saved with the route function, which can take a list of paths, and map them to a specific function called an action.

To use the now function, a request checks whether or not a supplied pathname (with arguments) is in the mapping, if this is the case, it applies its argument and returns a reference to the request, along with a status code and the returned content of the function. This is mostly handled in our `loop/2` function.

2.2 Part II

For the hello module we decided to create a path for hello and goodbye. We had the choice of using one function for both of them and pattern matching on the result, in the case that "\hello", "\goodbye" are matched and then changing the string returned accordingly. We instead opted to use two route calls and two functions, one for "\hello" and and one for "\goodbye" respectively.

We struggled to figure out how to correctly represent and capture local state. Due to our inability to correctly encode local state it wasn't possible to implement the other modules to test our code against. They both depended on changing local states.

3 Assessment

We were able to get the flamingo server working for regular routes and responses. The Greetings example provided in the handout had the correct functionality. We also got our code to work for the hello module described in part 1.

We struggled to grasp the difference between local states and the global states as well as the utility of the global state. We also were unable to figure out how to correctly capture the local state. This meant that we were unable to implement the last modules asked for in Part II.

As always we ran into problems that took longer than expected, such as infinite waiting times and syntactical errors. And with more time I feel that we could improve our implementation.

4 Test

Our testing was done in the erlang shell. We used the greetings module as a starting point to make sure that our initial implementation worked. We also tested different error messages and response codes, making sure that they returned the correct response for the respective requests.

```
Eshell V10.1  (abort with ^G)
1> c(flamingo).
{ok,flamingo}
2> c(hello).
{ok,hello}
3> Server = hello:server().
<0.88.0>
4> hello:try_hello(Server).
{200,"Hello my friend"}
5> hello:try_bye(Server).
{200,"Sad to see you go."}
```

Part of the Erlang Shell session that we used to test our implementation of the Flamingo server and the hello module in Part II can be seen above. We weren't familiar with Erlang testing frameworks, and due to time constraints we stuck to a very hands on testing approach.

5 Appendix

5.1 flamingo.erl

```
-module(flamingo).

-export([new/1, request/4, route/4, drop_group/2, request_reply/2, loop/2, new_mapping/4]).

request_reply(Pid, Request) ->
    Pid ! {self(), Request},
    receive
        {Pid, Response} -> Response
    end.



new(_Global) -> {ok, spawn(flamingo, loop, [_Global, #{}])}.

request(_Flamingo, _Request, _From, _Ref) ->
    _Flamingo ! {_From, {request, _Request, _Ref}}.
    %receive
     %   {_Ref, Response} -> Response;
      %  _ -> "this failed epically"
    %end.

route(_Flamingo, _Path, _Fun, _Arg) ->
    _Flamingo ! {self(), {route, _Path, _Fun, _Arg}},
    receive
        {ok, _Flamingo} -> {ok, _Flamingo};
        _ -> "we super failed"
    end.

drop_group(_Flamingo, _Id) ->
    not_implemented.


loop(State, LocalState) ->
    Me = self(),
    receive
        {From, {request, {_Path, _Args}, _Ref}} ->
            case maps:find(_Path, LocalState) of
              {ok, Fun} ->
                From ! {_Ref, {200, apply(Fun, [{_Path, _Args},
                State, LocalState])}};
              {error} ->
                From ! {_Ref, {404, "Path not found"}}
            end,
            %% From ! {_Ref, {200, apply(maps:get(_Path, LocalState)
            , [{_Path, _Args}, State, LocalState])}},
            loop(State, LocalState);
        {From, {route, Path, Fun, Args}} ->
            NewMap = new_mapping(Path, Fun, Args, LocalState),
            From ! {ok, Me},
            loop(State, NewMap);
        _ -> "we Failed"
    end.

new_mapping(Path, Fun, Args, Map) ->
    case (Path) of
        [H|T] -> new_mapping(T, Fun, Args, Map#{H => Fun});
        [] -> Map
    end.
```

5.2 hello.erl

```
    -module(hello).
-export([server/0, try_hello/1, try_bye/1]).

hello({_Path, _}, _, _) ->
    "Hello my friend".

bye({_Path, _}, _, _) ->
    "Sad to see you go.".




server() ->
    {ok, F} = flamingo:new("Hi and Bye Server"),
    flamingo:route(F, ["/hello"], fun hello/3, none),
    flamingo:route(F, ["/goodbye"], fun bye/3, none),
    F.

try_hello(Server) ->
    Me = self(),
    Ref = make_ref(),
    flamingo:request(Server, {"/hello", []},
                     Me, Ref),
    receive
        {Ref, Reply} -> Reply
    end.

try_bye(Server) ->
    Me = self(),
    Ref = make_ref(),
    flamingo:request(Server, {"/goodbye", []},
                     Me, Ref),
    receive
        {Ref, Reply} -> Reply
    end.
```