Advanced Programming
Assignment 4 : Who Doesn't Love Emoji
Report

AP22 Assignment 4 Group 60
KRW521, CPJ395

October 7, 2022

1 Design and Implementation

1.1 Design and framework of the program

Functions are divided into three main parts: executable functions, client-side and server-side. The executable function will call the client side, and the client side will receive the parameters of the function and organize them into a message to the server side. The server side is a loop function that saves the state of the emoji list. The **start** function will start a server-side loop. Once it is started, it will always wait for a message from the client. The format of the client's message is: Pid of the main program, Operator, Parameters, where the Operator determines what operations the server should run, and the parameters are the parameters needed for those operations sent as messages along with the Operator. After the server side receives these messages, it calls the corresponding helper functions to process them. On the one hand, the server side updates its own state, and on the other hand, it sends the response message to the client after the processing is completed. Finally, when the client receives the response from the server, it parses the message and delivers the information the user needs. The exact execution flow of the program is shown in the figure below, where User can be seen as a shell, and Commend is the input to the shell.

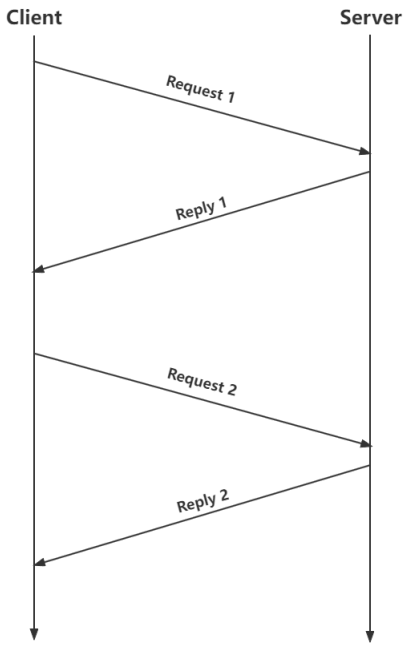

Figure 1: Communication between Client and Sever in the time dimension

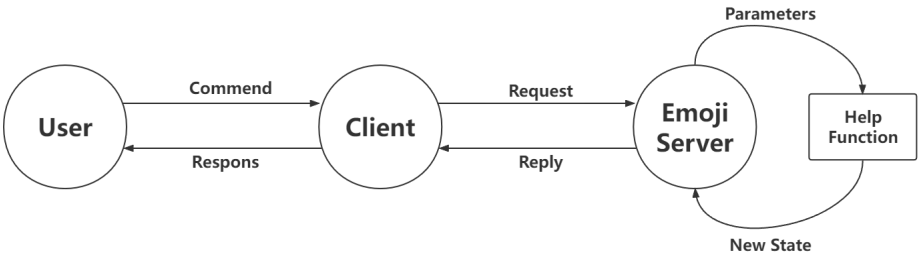

Figure 2: Execution flow of the program

1.2 Some helper functions

Besides the skeleton mentioned above, our program also has a series of Helper functions (Auxiliary functions) that will be called in the the function loop. The **duplicate** function is used to check if there are duplicate elements in the initialized list. The **getEmo** function is used to query the emoji corresponding to the **Shorcode** when the **lookup** function is executed. The **deleteAll** function is used to implement the delete function. The **renewState** function is used to update the state when the **lookup** is executed. **bindFun** function is used to implement **analytics**. **getStat** is used to implement **get_analytics**. **removeFun** is used to implement **remove_analytics**.

1.3 Implementation of key functions

The most difficult part of this assignment is the implementation of several functions related to **alias**, **lookup** and **analytics**. We believe that the binary tuple **{Shortcode, Emoji}** used for initialization is not enough to store aliases and bind functions. So we decided to extend this tuple and turn it into a 4-tuple, which is what the **withBind** function does. The **withBind** function expands the initialized binary tuple into a 4-tuple **{Shortcode, Emoji, RealName, FunList}**, where Shortcode and Emoji still store the previous RealName is used to store the original name of the alias, and FunList is called in the analytics function to save the function. The way it works is that when the alias function is called, a new element is created for the legal alias and its RealName is set to the original name of the previous Shortcode. This way, no matter how the alias is called, the RealName of the resulting alias element will always be the original name. The way it works is shown in the figure below.

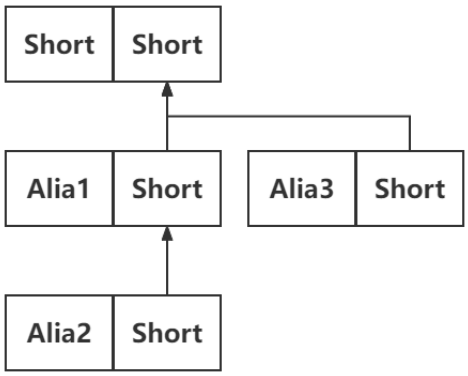

Figure 3: How RealName works

So when we want to remove a Shortcode and all its alias elements, we just need to iterate through the list and remove all the elements with the same RealName. This method also allows us to see the real name of the element very easily, we only need to see its **RealName**. This makes it easier to handle the function list of a Shortcode. As shown in the figure below, the FunList of all aliasedis is empty. When we want to manipulate the FunList by alias, we only need to find its real name element by its RealName and manipulate it. This way we can avoid errors caused by backing up the FunList too much, because we only need to maintain one function table.

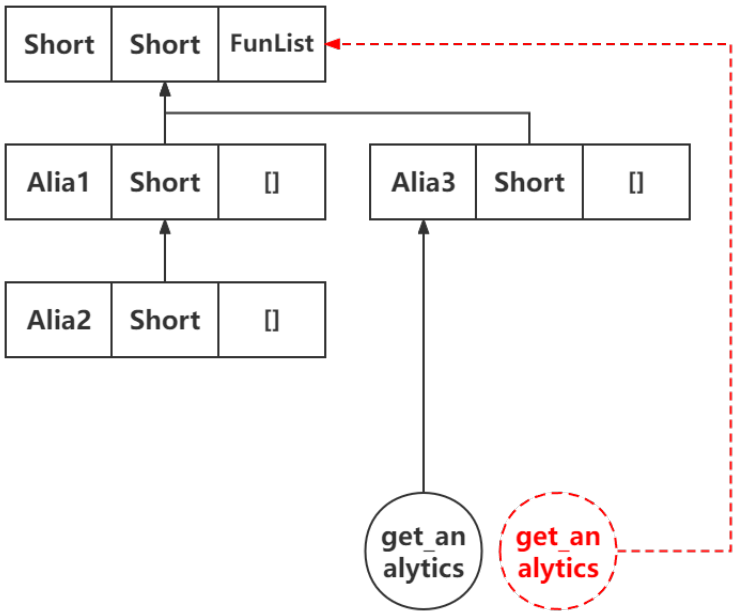

Figure 4: How FunList works

In our opinion, an alias is more like a pointer to the original memory, which doesn't hold any information.

2 Assessment of The Code

2.1 Completeness

All functions are completed, and the completion of all functions are as follows:

Class of Function	Function Name	Completion
Start	start	Completed
Basic functionality	new_shortcode	Completed
Basic functionality	alias	Completed
Basic functionality	delete	Completed
Basic functionality	lookup	Completed
Analytics	analytics	Completed
Analytics	get_analytics	Completed
Analytics	remove_analytics	Completed
Stop	stop	Completed
Intermediary	request_reply	Completed

2.2 Correctness

After running on the online TA, all test cases were ok. And the condition of correctness is as follows:

Class of Function	Function Name	Test Result
Start	start	OK
Basic functionality	new_shortcode	OK
Basic functionality	alias	OK
Basic functionality	delete	OK
Basic functionality	lookup	OK
Analytics	analytics	OK
Analytics	get_analytics	OK
Analytics	remove_analytics	OK
Stop	stop	OK
Intermediary	request_reply	OK

2.3 Efficiency

The efficiency of our program is also at a high level.

2.4 Robustness

Class of Function	Function Name	Robustness
Start	start	Strong
Basic functionality	new_shortcode	Strong
Basic functionality	alias	Strong
Basic functionality	delete	Strong
Basic functionality	lookup	Strong
Analytics	analytics	Strong
Analytics	get_analytics	Strong
Analytics	remove_analytics	Strong
Stop	stop	Strong
Intermediary	request_reply	Strong

2.5 Maintainability

As for the maintainability, this time, we separate the server and client by using the Intermediary, that is, function **request_reply**, making our program be easy to maintain.

Class of Function	Function Name	Maintainability
Start	start	Good
Basic functionality	new_shortcode	Good
Basic functionality	alias	Good
Basic functionality	delete	Good
Basic functionality	lookup	Good
Analytics	analytics	Good
Analytics	get_analytics	Good
Analytics	remove_analytics	Good
Stop	stop	Good
Intermediary	request_reply	Good

A Appendix: emoji.erl

```
-module(emoji).

-export([start/1, new_shortcode/3, alias/3, delete/2, lookup/2,
         analytics/5, get_analytics/2, remove_analytics/3,
         stop/1]).

% -type shortcode() :: string().
% -type emoji() :: binary().
% -type analytic_fun(State) :: fun((shortcode(), State) -> State).


start(Initial) ->
    ShortList =
        lists:map(
            fun(Emoji) ->
                case Emoji of
                    {Short, _} -> Short
                end
            end, Initial),
    case duplicate(ShortList) of
        true  -> {error, "duplicate shortcodes"};
        false ->
            try E = spawn(fun() -> loop(withBind(Initial)) end) of
                _        -> {ok, E}
            catch
                _:Reason -> {error, Reason}

            end
    end.


new_shortcode(E, Short, Emo) ->
    request_reply(E, {new_shortcode, Short, Emo}).

alias(E, Short1, Short2) ->
    request_reply(E, {alias, Short1, Short2}).

delete(E, Short) ->
```

```
39      request_reply(E, {delete, Short}).
40
41  lookup(E, Short) ->
42      request_reply(E, {lookup, Short}).
43
44  analytics(E, Short, Fun, Label, Init) ->
45      request_reply(E, {analytics, Short, Fun, Label, Init}).
46
47  get_analytics(E, Short) ->
48      request_reply(E, {get_analytics, Short}).
49
50  remove_analytics(E, Short, Label) ->
51      request_reply(E, {remove_analytics, Short, Label}).
52
53  stop(E) ->
54      request_reply(E, stop).
55
56
57  request_reply(Pid, Request) ->
58          Pid ! {self(), Request},
59          receive
60                  {Pid, Response} -> Response
61          end.
62
63
64  loop(EMap) ->
65      ShortList =
66          lists:map(
67              fun(Emoji) ->
68                  case Emoji of
69                      {Short, _, _, _} -> Short
70                  end
71              end, EMap),
72      receive
73          {From, {new_shortcode, Short, Emo}} ->
74              case lists:member(Short, ShortList) of
75                  true  ->
76                      From!{self(), {error, "new_shortcode error: |"++
    ↪  Short ++"| already exists"}},
77                      loop(EMap);
```

```
78                 false ->
79                     From!{self(), ok},
80                     loop([{Short, Emo, Short, []}|EMap])
81             end;
82         {From, {alias, Short1, Short2}} ->
83             case lists:member(Short1, ShortList) of
84                 false ->
85                     From!{self(), {error, "alias error: |"++ Short1 ++"|
    ↪  does not exist"}},
86                     loop(EMap);
87                 true  ->
88                     case lists:member(Short2, ShortList) of
89                         true  ->
90                             From!{self(), {error, "alias error: |"++
    ↪  Short2 ++"| already exists"}},
91                             loop(EMap);
92                         false ->
93                             From!{self(), ok},
94                             loop([{Short2, getEmo(Short1, EMap),
    ↪  realShort(Short1, EMap), []}|EMap])
95                     end
96             end;
97         {From, {delete, Short}} ->
98             case lists:member(Short, ShortList) of
99                 false ->
100                    loop(EMap);
101                true  ->
102                    From!{self(), ok},
103                    loop(deleteAll(realShort(Short, EMap), EMap))
104            end;
105        {From, {lookup, Short}} ->
106            case lists:member(Short, ShortList) of
107                false ->
108                    From!{self(), no_emoji},
109                    loop(EMap);
110                true  ->
111                    From!{self(), {ok, getEmo(Short, EMap)}},
112                    loop(renewState(Short, realShort(Short, EMap), EMap))
113            end;
114        {From, {analytics, Short, Fun, Label, Init}} ->
```

```
115             case lists:member(Short, ShortList) of
116                 false ->
117                     From!{self(), {error, "analytics error: |"++ Short
    ↪  ++"| does not exist"}},
118                     loop(EMap);
119                 true  ->
120                     case labelExist(realShort(Short, EMap), Label, EMap)
    ↪  of
121                         true  ->
122                             From!{self(), {error, "analytics error: |"++
    ↪  Label ++"| already exists"}},
123                             loop(EMap);
124                         false ->
125                             From!{self(), ok},
126                             loop(bindFun(realShort(Short, EMap), Fun,
    ↪  Label, Init, EMap))
127                     end
128             end;
129         {From, {get_analytics, Short}} ->
130             case lists:member(Short, ShortList) of
131                 false ->
132                     From!{self(), {error, "get_analytics error: |"++
    ↪  Short ++"| does not exist"}},
133                     loop(EMap);
134                 true  ->
135                     From!{self(), {ok, getStat(realShort(Short, EMap),
    ↪  EMap)}},
136                     loop(EMap)
137             end;
138         {From, {remove_analytics, Short, Label}} ->
139             case lists:member(Short, ShortList) of
140                 false ->
141                     From!{self(), {error, "remove_analytics error: |"++
    ↪  Short ++"| does not exist"}},
142                     loop(EMap);
143                 true  ->
144                     From!{self(), ok},
145                     loop(removeFun(realShort(Short, EMap), Label, EMap))
146             end;
147         {From, stop} -> From!{self(), ok};
```

```
148         {From, _} ->
149             From!{self(), "syntax error"},
150             loop(EMap)
151     end.
152
153
154
155
156 duplicate(ShortList)->
157     case ShortList of
158         [] -> false;
159         [Head|Tail] ->
160             case lists:member(Head, Tail) of
161                 true  -> true;
162                 false -> duplicate(Tail)
163             end
164     end.
165
166 withBind(List) ->
167     case List of
168         [] -> [];
169         [{Short, Emo}|Rest] -> [{Short, Emo, Short, []}|withBind(Rest)]
170     end.
171
172 realShort(Short, EMap) ->
173     case EMap of
174         [] -> noting;
175         [{Short_, _, RealShort, _}|Rest] ->
176             case Short_ =:= Short of
177                 true  -> RealShort;
178                 false -> realShort(Short, Rest)
179             end
180     end.
181
182 getEmo(Short, EMap) ->
183     case EMap of
184         [] -> noting;
185         [{Short_, Emo, _, _}|Rest] ->
186             case Short_ =:= Short of
187                 true  -> Emo;
```

```
188                 false -> getEmo(Short, Rest)
189             end
190     end.
191
192 deleteAll(Short, EMap) ->
193     case EMap of
194         [] -> [];
195         [{Short_, Emo, RealShort, FunList}|Rest] ->
196             case RealShort =:= Short of
197                 true  -> deleteAll(Short, Rest);
198                 false -> [{Short_, Emo, RealShort,
    ↪  FunList}|deleteAll(Short, Rest)]
199             end
200     end.
201
202 renewState(Short, RealShort, EMap) ->
203     case EMap of
204         [] -> [];
205         [{Short_, Emo, RealShort_, FunList}|Rest] ->
206             case (RealShort =:= RealShort_) and (RealShort =:= Short_) of
207                 true  -> [{Short_, Emo, RealShort_, runFun(Short,
    ↪  FunList)}|Rest];
208                 false -> [{Short_, Emo, RealShort_,
    ↪  FunList}|renewState(Short, RealShort, Rest)]
209             end
210     end.
211
212 runFun(Short, FunList) ->
213     case FunList of
214         [] -> [];
215         [{Fun, Label, State}|Rest] ->
216             try Fun(Short, State) of
217                 NewState -> [{Fun,Label,NewState}|runFun(Short,Rest)]
218             catch
219                 _:_      -> [{Fun,Label,State}|runFun(Short,Rest)]
220             end
221     end.
222
223 labelExist(Short, Label, EMap) ->
224     case EMap of
```

```
225         [] -> false;
226         [{Short_, _, _, FunList}|Rest] ->
227             case Short_ =:= Short of
228                 false -> labelExist(Short, Label, Rest);
229                 true  ->
230                     case findLable(Label, FunList) of
231                         true  -> true;
232                         false -> labelExist(Short, Label, Rest)
233                     end
234             end
235     end.
236
237 findLable(Label, FunList) ->
238     case FunList of
239         [] -> false;
240         [{_, Label_, _}|Rest] ->
241             case Label_ =:= Label of
242                 true  -> true;
243                 false -> findLable(Label, Rest)
244             end
245     end.
246
247 bindFun(Short, Fun, Label, Init, EMap) ->
248     case EMap of
249         [] -> [];
250         [{Short_, Emo, RealShort, FunList}|Rest] ->
251             case Short_ =:= Short of
252                 false -> [{Short_, Emo, RealShort,
↪   FunList}|bindFun(Short, Fun, Label, Init, Rest)];
253                 true  -> [{Short_, Emo, RealShort, [{Fun, Label,
↪   Init}|FunList]}|Rest]
254             end
255     end.
256
257 getStat(Short, EMap) ->
258     case EMap of
259         [] -> nothing;
260         [{Short_, _, _, FunList}|Rest] ->
261             case Short_ =:= Short of
262                 true  -> showStat(FunList);
```

```
263                 false -> getStat(Short, Rest)
264             end
265     end.
266
267 showStat(FunList) ->
268     case FunList of
269         [] -> [];
270         [{_, Label, State}|Rest] -> [{Label, State}|showStat(Rest)]
271     end.
272
273 removeFun(Short, Label, EMap) ->
274     case EMap of
275         [] -> [];
276         [{Short_, Emo, RealShort, FunList}|Rest] ->
277             case Short_ =:= Short of
278                 true  -> [{Short_, Emo, RealShort, remove(Label,
    ↪  FunList)}|Rest];
279                 false -> [{Short_, Emo, RealShort,
    ↪  FunList}|removeFun(Short, Label, Rest)]
280             end
281     end.
282
283 remove(Label, FunList) ->
284     case FunList of
285         [] -> [];
286         [{Fun, Label_, State}|Rest] ->
287             case Label_ =:= Label of
288                 true  -> Rest;
289                 false -> [{Fun, Label_, State}|remove(Label, Rest)]
290             end
291     end.
292
```

B Appendix: test_emoji.erl

```
1
2  -module(test_emoji).
3
4  -export([test_all/0]).
5
6  % We'll use EUnit
7  -include_lib("eunit/include/eunit.hrl").
8
9  test_all() -> eunit:test(testsuite(), [verbose]).
10
11 testsuite() ->
12     [ {"Basic behaviour", spawn,
13        [ test_start_server()
14        , test_shortcode_smiley()
15        , test_shortcode_smiley_lookup()
16        ,test_smiley_alias_and_lookup()
17        ,test_smiley_alias_and_error()
18        ,test_smiley_alias_and_ask_list()
19        ,test_smiley_alias_and_delete()
20        ]
21       },
22       {"Analytics", spawn,
23        [ test_register_analytics()
24        ,test_register_function()
25        ,test_2register_function()
26        ,test_2register_function_with_alias()
27        ,test_2register_function_with_multiple_alias()
28        ,test_register_remove_with_alias()
29        ]
30       },
31       {"Scale", spawn,
32       [ test_scale_small()
33        ,test_scale_medium()
34       ]
35      }
36     ].
37
38 test_start_server() ->
```

```
39      {"We can call start/1 and it does not crash",
40       fun () ->
41         ?assertMatch({ok, _}, emoji:start([]))
42       end }.
43
44  test_shortcode_smiley() ->
45      {"Register new shortcode",
46       fun () ->
47         {ok, S} = emoji:start([]),
48         ?assertEqual(ok, emoji:new_shortcode(S, "smiley",
49                                              <<240,159,152,131>>))
50       end }.
51  test_shortcode_smiley_lookup() ->
52        {"Register and lookup new shortcode",
53         fun () ->
54           {ok, S} = emoji:start([]),
55           emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
56           ?assertMatch({ok, <<240,159,152,131>>}, emoji:lookup(S,
    ↪  "smiley"))
57         end }.
58
59  test_smiley_alias_and_lookup() ->
60  {"Register and lookup new shortcode with alias",
61    fun () ->
62      {ok, S} = emoji:start([]),
63      emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
64      emoji:alias(S, "smiley", "smiley1"),
65      ?assertMatch({ok, <<240,159,152,131>>}, emoji:lookup(S, "smiley1"))
66    end }.
67
68  test_smiley_alias_and_error() ->
69    {"make an alias error",
70      fun () ->
71        {ok, S} = emoji:start([]),
72        emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
73        ?assertMatch({error, _}, emoji:alias(S, "smiley", "smiley"))
74      end }.
75
76  test_smiley_alias_and_ask_list() ->
77    {"test the alias registration",
```

```
78      fun () ->
79        {ok, S} = emoji:start([]),
80        emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
81        emoji:alias(S, "smiley", "smiley1"),
82        ?assertEqual({ok,<<240,159,152,131>>}, emoji:lookup(S, "smiley1"))
83      end }.
84
85  test_smiley_alias_and_delete() ->
86    {"delete shrotcode and its alias",
87      fun () ->
88        {ok, S} = emoji:start([]),
89        emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
90        emoji:alias(S, "smiley", "smiley1"),
91        emoji:delete(S, "smiley"),
92        ?assertEqual(no_emoji, emoji:lookup(S, "smiley1"))
93      end }.
94
95  test_register_analytics() ->
96    {"Register analytics function for a new shortcode",
97      fun () ->
98        {ok, S} = emoji:start([]),
99        emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
100       Hit = fun(_, N) -> N+1 end,
101       emoji:analytics(S, "smiley",  Hit, "Hit", 0),
102       ?assertEqual({ok, [{"Hit", 0}]}, emoji:get_analytics(S, "smiley"))
103     end }.
104
105 test_register_function() ->
106   {"Register analytics function and test it by lookup",
107     fun () ->
108       {ok, S} = emoji:start([]),
109       emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
110       Hit = fun(_, N) -> N+1 end,
111       emoji:analytics(S, "smiley",  Hit, "Hit", 0),
112       emoji:lookup(S,"smiley"),
113       ?assertEqual({ok, [{"Hit", 1}]}, emoji:get_analytics(S, "smiley"))
114     end }.
115
116 test_2register_function() ->
117   {"Register 2 analytics functions and test them by lookup",
```

```
118     fun () ->
119       {ok, S} = emoji:start([]),
120       emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
121       Hit = fun(_, N) -> N+1 end,
122       Last = fun (S1, _) -> S1 end,
123       emoji:analytics(S, "smiley",  Hit, "Hit", 0),
124       emoji:analytics(S, "smiley",  Last, "Last", none),
125       emoji:lookup(S,"smiley"),
126       ?assertEqual({ok, [{"Last", "smiley"} ,{"Hit", 1}]},
    ↪  emoji:get_analytics(S, "smiley"))
127     end }.
128
129 test_2register_function_with_alias() ->
130   {"Register 2 analytics functions for a short code with alias and test
    ↪  them by lookup",
131     fun () ->
132       {ok, S} = emoji:start([]),
133       emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
134       emoji:alias(S, "smiley", "smiley1"),
135       Hit = fun(_, N) -> N+1 end,
136       Last = fun (S1, _) -> S1 end,
137       emoji:analytics(S, "smiley1",  Hit, "Hit", 0),
138       emoji:analytics(S, "smiley1",  Last, "Last", none),
139       emoji:lookup(S,"smiley1"),
140       ?assertEqual({ok, [{"Last", "smiley1"} ,{"Hit", 1}]},
    ↪  emoji:get_analytics(S, "smiley1"))
141     end }.
142
143 test_2register_function_with_multiple_alias() ->
144   {"Register 2 analytics functions for a short code with multiple aliases
    ↪  and test them by lookup",
145     fun () ->
146       {ok, S} = emoji:start([]),
147       emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
148       emoji:alias(S, "smiley", "smiley1"),
149       emoji:alias(S, "smiley", "smiley2"),
150       Hit = fun(_, N) -> N+1 end,
151       Last = fun (S1, _) -> S1 end,
152       emoji:analytics(S, "smiley1",  Hit, "Hit", 0),
153       emoji:analytics(S, "smiley1",  Last, "Last", none),
```

```
154       emoji:lookup(S,"smiley1"),
155       ?assertEqual({ok, [{"Last", "smiley1"} ,{"Hit", 1}]},
    ↪  emoji:get_analytics(S, "smiley2"))
156     end }.
157
158 test_register_remove_with_alias()->
159   {"Removing registered analytics function with multiple aliases",
160   fun () ->
161     {ok, S} = emoji:start([]),
162     emoji:new_shortcode(S, "smiley",<<240,159,152,131>>),
163     emoji:alias(S, "smiley", "smiley1"),
164     emoji:alias(S, "smiley", "smiley2"),
165     Hit = fun(_, N) -> N+1 end,
166     Last = fun (S1, _) -> S1 end,
167     emoji:analytics(S, "smiley1",  Hit, "Hit", 0),
168     emoji:analytics(S, "smiley1",  Last, "Last", none),
169     emoji:remove_analytics(S, "smiley", "Hit"),
170     ?assertEqual({ok, [{"Last", none}]}, emoji:get_analytics(S,
    ↪  "smiley"))
171   end }.
172
173 test_scale_small() ->
174   {"initial with small amount of emoji and lookup",
175   fun () ->
176     InitialList = someemoji:small(),
177     {ok, S} = emoji:start(InitialList),
178     ?assertEqual({ok,<<""/utf8>>}, emoji:lookup(S,"boot"))
179   end }.
180
181 test_scale_medium() ->
182   {"initial with small amount of emoji and lookup",
183   fun () ->
184     InitialList = someemoji:medium(),
185     {ok, S} = emoji:start(InitialList),
186
    ↪  ?assertEqual({ok,<<240,159,145,168,240,159,143,190,226,128,141,240,159,146,187>>},
187     emoji:lookup(S,"man technologist: medium-dark skin tone"))
188   end }.
189
190 % test_after_stop()->
```

```
191 %   {"test after stop the server",
192 %   fun () ->
193 %     InitialList = someemoji:small(),
194 %     {ok, S} = emoji:start(InitialList),
195 %     emoji:stop(S),
196 %     ?assertMatch({error, _}, emoji:lookup(S,"boot"))
197 %   end }.
198
```