Advanced Programming
Assignment 4
Emoji 😍

GRC196 SZW935

October 6, 2021

Note that both of us are retaking this class, so the warm-up is very similar to our submission from last year's rps assignment. Part 2 of the assignment was written from scratch.

1 Completeness 💯

We attempted every part of the specification with varying degrees of success. The faults are covered in section 2, here we will go over some notable design choices.

We used a map structure as the state, given as `#{Short, {Emo, Aliases, Analytics}}` where `Aliases` is a list of shortcodes and `Analytics` is another map `#{Label => {Fun, State}}`.

When looking up an emoji we first check if the short is in the state map, and if not we then use `loop_alias/2` to check all the alias lists (by turning the map into a list and iterating over). How we handled analytics functions is discussed in section 4.

We separated concerns with a few other auxiliary functions, the main ones being `handle_call/2` and `handle_nonblock/2` which is called when the loop receives a request. We also use `handle_analytics/2` whenever there is a lookup, which is expanded on more also in section 4.

2 Correctness 🐧

We wrote 48 tests in total to cover a broad spectrum of functionality, including efficiency and robustness. They all pass save for one test for robustness. More tests could have been written to cover edge cases of robustness, for example a test that ensures correct functions still get updated if there is one that raises an exception.

3 Efficiency ⏳

We admit that our understanding of what makes our erlang code efficient/non-efficient is limited, but as mentioned, we have written tests that operate on a larger emoji server (using `medium` from someemoji.erl) and they pass in a timely fashion. Thus we think our server is efficient enough for the scope of this library.

4 Robustness 🛡️

We implemented robustness to varying degrees of success. One thing we do is make use of the `make_ref()` functions to make sure we are only handling replies from the emoji server. Analytics functions are handled in lookups by creating one worker process to run them. This way if there are functions that raises an exception, it is caught (through the use of `process_flag(trap_exit, true)`) and the emoji is still returned.

Unfortunately, functions that fail to terminate are not handled, and if one exists then a response to a lookup is never received. Also, if one function raises an exception, none of the states of the analytics get updated. Our implementation is close to working; both these issues would be solved by having individual processes for each function, rather than one process that handles all functions. As a quick fix we considered using a timeout in `request_reply` for endless functions, but decided against it as that would cause even more problems than it would solve.

5 Maintainability ⚙️

We argue that our code is fairly maintainable based on our separation of concerns as mentioned in section 1. The `handle_call/2` function is quite verbose, so maybe the concerns could have been separated further. We also commented when necessary, though probably could have used `-spec` before each function for better understandability.

6 Other 👾

No comment.

Appendix

6.1 Emoji

```
-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) ->
  case length(Initial) == sets:size(sets:from_list(proplists:get_keys(Initial))) of
    true -> {ok, spawn(fun() -> loop(maps:from_list(init_state(Initial))) end)};
    false -> {error, "shortcodes must be unique"}
  end.

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

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

delete(E, Short) -> non_blocking(E, {delete, Short}).

lookup(E, Short) -> request_reply(E, {lookup, Short}).

analytics(E, Short, Fun, Label, Init) -> request_reply(E, {analytics, Short, Fun,
    Label, Init}).

get_analytics(E, Short) -> request_reply(E, {get_analytics, Short}).

remove_analytics(E, Short, Label) -> non_blocking(E, {remove_analytics, Short, Label})
    .

stop(E) -> request_reply(E, stop).

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

non_blocking(Pid, Msg) ->
  Pid ! {non_blocking, Msg}.

% {Shortcode => {Emo, [Aliases], [Label => {Fun, State}]}}
loop(State) ->
  receive

    % Handle worker normal exits
    {'EXIT', _, normal} ->
      loop(State);

```

```
    {non_blocking, Request} ->
      NewState = handle_nonblock(Request, State),
      loop(NewState);
    {From, Ref, stop} ->
      From ! {Ref, ok};
    {From, Ref, Request} ->
      {NewState, Res} = handle_call(Request, State),
      From ! {Ref, Res},
      loop(NewState)
  end.

handle_call(Request, State) ->
  case Request of
    {new, Short, Emo} ->
      case get_emoji(Short, State) of
        {ok, _} ->
          {State, {error, "shortcode already exists"}};
        _ ->
          NewState = maps:put(Short, {Emo, [], maps:new()}, State),
          {NewState, ok}
      end;

    {alias, Short1, Short2} ->
      case get_emoji(Short2, State) of
        {ok, _} ->
          {State, {error, "alias already exists"}};
        no_emoji ->
          case get_emoji(Short1, State) of
            {ok, {OrigShort, {Emo, Aliases, Analytics}}} ->
              NewState = maps:put(OrigShort, {Emo, Aliases ++ [Short2], Analytics},
    State),
              {NewState, ok};
            no_emoji ->
              {State, {error, "shortcode does not exist"}}
          end
      end;

    {lookup, Short} ->
      case get_emoji(Short, State) of
        {ok, Emoji} ->
          handle_analytics(State, Emoji);
        no_emoji ->
          {State, no_emoji}
      end;

    {analytics, Short, Fun, Label, Init} ->
      case get_emoji(Short, State) of
        {ok, {OrigShort, {Emo, Aliases, Analytics}}} ->
          case maps:find(Label, Analytics) of
            {ok, _} ->
              {State, {error, "label already exists for shortcode"}};
            _ ->
              NewAnalytics = maps:put(Label, {Fun, Init}, Analytics),
              NewState = maps:put(OrigShort, {Emo, Aliases, NewAnalytics}, State),
              {NewState, ok}
          end;
        no_emoji ->
```

```
          {State, {error, "shortcode does not exist"}}
      end;

    {get_analytics, Short} ->
      case get_emoji(Short, State) of
        {ok, {_, {_, _, Analytics}}} ->
          ReturnList = read_analytics(maps:to_list(Analytics)),
          {State, {ok, ReturnList}};
        no_emoji ->
          {State, {error, "shortcode does not exist"}}
      end
  end.

handle_nonblock(Request, State) ->
  case Request of
    {delete, Short} ->
      case lookup_alias(Short, maps:to_list(State)) of
        {ok, {OrigShort, _}} ->
          maps:remove(OrigShort, State);
        _ ->
          maps:remove(Short, State)
      end;

    {remove_analytics, Short, Label} ->
      case get_emoji(Short, State) of
        {ok, {OrigShort, {Emo, Aliases, Analytics}}} ->
          NewAnalytics = maps:remove(Label, Analytics),
          maps:put(OrigShort, {Emo, Aliases, NewAnalytics}, State);
        no_emoji ->
          State
      end
  end.

% Updates analytics as a worker process
handle_analytics(State, {Short, {Emo, Aliases, Analytics}}) ->
  Me = self(),
  process_flag(trap_exit, true),
  Worker = spawn_link(fun() ->
              NewAnalytics = maps:from_list(update_analytics(Short, maps:to_list(
    Analytics))),
              Me ! {Me, NewAnalytics}
            end),
  receive
    {Me, NewAnalytics} ->
      Updated = {Emo, Aliases, NewAnalytics},
      NewState = maps:put(Short, Updated, State),
      {NewState, {ok, Emo}};
    {'EXIT', Worker, _} ->
      {State, {ok, Emo}}
  end.

% Converts initial list to initial state of server
init_state([]) -> [];
init_state([{Short, Emo} | Shortcodes]) ->
  [{Short, {Emo, [], maps:new()}}] ++ init_state(Shortcodes).

% Gets the emoji for a given short/alias
```

```
get_emoji(Short, State) ->
  case maps:find(Short, State) of
    {ok, {Emo, Aliases, Analytics}} ->
      {ok, {Short, {Emo, Aliases, Analytics}}};
    _ ->
      lookup_alias(Short, maps:to_list(State))
  end.

% Finds short in aliases
lookup_alias(_, []) -> no_emoji;
lookup_alias(Alias, [{Short, {Emo, Aliases, Analytics}} | Shortcodes]) ->
  case lists:member(Alias, Aliases) of
    true -> {ok, {Short, {Emo, Aliases, Analytics}}};
    false -> lookup_alias(Alias, Shortcodes)
  end.

% Converts analytics to the expected return list
read_analytics([]) -> [];
read_analytics([{Label, {_, State}} | Analytics]) ->
  [{Label, State}] ++ read_analytics(Analytics).

% Run all the analytics for a shortcode, called as worker process
update_analytics(_, []) -> [];
update_analytics(Short, [{Label, {Fun, State}} | Analytics]) ->
  try Fun(Short, State) of
    NewState ->
      [{Label, {Fun, NewState}}] ++ update_analytics(Short, Analytics)
  catch
    Ex ->
      exit(Ex)
  end.
```

Listing 1: emojis

6.2 Tests for Emoji

```
-module(test_emoji).

-export([test_all/0]).

% We'll use EUnit
-include_lib("eunit/include/eunit.hrl").

test_all() -> eunit:test(testsuite(), [verbose]).

testsuite() ->
    [ {"Basic behaviour", spawn,
       [ test_start_server(),
         test_start_server_one_shortcode(),
         test_start_server_not_unique_shortcode(),
         test_start_server_not_unique_binary(),
         test_start_server_small(),
         test_start_server_medium(),

         test_stop_server(),
         test_stop_server_multiple(),

         test_new_shortcode_unique(),
         test_new_shortcode_non_unique(),
         test_new_shortcode_already_alias(),

         test_alias(),
         test_alias_non_existing_shortcode(),
         test_alias_existing_alias(),

         test_lookup_existing(),
         test_lookup_non_existing(),
         test_lookup_alias(),
         test_lookup_from_list_of_aliases(),
         test_lookup_alias_of_alias(),

         test_delete_shortcode(),
         test_delete_alias(),
         test_delete_non_existing(),

         test_analytics(),
         test_analytics_alias(),
         test_analytics_multiple(),
         test_analytics_non_unique_label(),
         test_analytics_non_existing_short(),

         test_get_analytics_empty(),
         test_get_analytics_init(),
         test_get_analytics_multiple_funs_init(),
         test_get_analytics_new_alias(),
         test_get_analytics_after_lookups(),
         test_get_analytics_multiple_funs_after_lookups(),
         test_get_analytics_after_lookups_alias(),
         test_get_analytics_non_existing_short(),

         test_remove_analytics_shortcode(),
```

```
56          test_remove_analytics_alias(),
57          test_remove_analytics_one_out_of_many(),
58          test_remove_analytics_non_existing_label(),
59          test_remove_analytics_non_existing_shortcode(),
60
61         % load (efficiency) tests
62          test_medium_new(),
63          test_medium_lookup(),
64          test_medium_delete(),
65          test_medium_alias(),
66          test_medium_get_analytics(),
67          test_medium_remove_analytics(),
68
69         % robustness tests
70          test_analytics_broken_fun(),
71          test_analytics_forever_fun()
72        ]
73       }
74     ].
75
76 % analytics functions from example
77 hit(_, N) -> N+1.
78 accessed(SC, TS) ->
79   Now = calendar:local_time(),
80   [{SC,Now} | TS].
81 broken(_, _) -> throw("I don't like you").
82 forever(SC, State) -> State ++ forever(SC, State).
83
84
85 test_start_server() ->
86   {"We can call start/1 and it does not crash",
87     fun () ->
88       ?assertMatch({ok, _}, emoji:start([]))
89     end }.
90
91 test_start_server_one_shortcode() ->
92   {"We can call start/1 with one shortcode and it does not crash",
93     fun () ->
94       ?assertMatch({ok, _}, emoji:start([{"smiley", <<240,159,152,131>>}]))
95     end }.
96
97 test_start_server_not_unique_shortcode() ->
98   {"We can call start/1 with two non-unique shortcodes and it produces an error",
99     fun () ->
100      ?assertMatch({error, _}, emoji:start([{"smiley", <<240,159,152,131>>},
101                                            {"smiley", <<240,159,164,166>>}]))
102    end }.
103
104 test_start_server_not_unique_binary() ->
105   {"We can call start/1 with two non-unique binaries and it works",
106     fun () ->
107       ?assertMatch({ok, _}, emoji:start([{"smiley", <<240,159,152,131>>},
108                                          {"facepalm", <<240,159,152,131>>}]))
109    end }.
110
111 test_start_server_small() ->
112   {"We can call start/1 with small list and it does not crash",
```

```
113     fun () ->
114       ?assertMatch({ok, _}, emoji:start(someemoji:small()))
115     end }.
116
117 test_start_server_medium() ->
118   {"We can call start/1 with medium list and it does not crash",
119     fun () ->
120       ?assertMatch({ok, _}, emoji:start(someemoji:medium()))
121     end }.
122
123 test_stop_server() ->
124   {"We stop a server, check it returns ok",
125     fun () ->
126       {ok, S} = emoji:start([]),
127       ?assertEqual(ok, emoji:stop(S))
128     end }.
129
130 test_stop_server_multiple() ->
131   {"We start then stop multiple servers, check it returns ok",
132     fun () ->
133       {ok, S} = emoji:start([]),
134       {ok, S1} = emoji:start(someemoji:small()),
135       {ok, S2} = emoji:start(someemoji:medium()),
136       ?assertEqual(ok, emoji:stop(S)),
137       ?assertEqual(ok, emoji:stop(S1)),
138       ?assertEqual(ok, emoji:stop(S2))
139     end }.
140
141 test_new_shortcode_unique() ->
142   {"Register new unique shortcode",
143     fun () ->
144       {ok, S} = emoji:start([]),
145       ?assertEqual(ok, emoji:new_shortcode(S, "smiley",
146                                            <<240,159,152,131>>))
147     end }.
148
149 test_new_shortcode_non_unique() ->
150   {"Register new non-unique shortcode, error",
151     fun () ->
152       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
153       ?assertMatch({error, _}, emoji:new_shortcode(S, "smiley",
154                                            <<240,159,152,131>>))
155     end }.
156
157 test_new_shortcode_already_alias() ->
158   {"Register new shortcode that is already an alias, error",
159     fun () ->
160       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
161       ok = emoji:alias(S, "smiley", "happy"),
162       ?assertMatch({error, _}, emoji:new_shortcode(S, "happy",
163                                            <<240,159,152,131>>))
164     end }.
165
166 test_alias() ->
167   {"Register alias, no error",
168     fun () ->
169       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
```

```
170       ?assertEqual(ok, emoji:alias(S, "smiley", "happy"))
171     end }.
172
173 test_alias_non_existing_shortcode() ->
174   {"Register alias for non-existing shortcode, error",
175     fun () ->
176       {ok, S} = emoji:start([]),
177       ?assertMatch({error, _}, emoji:alias(S, "smiley", "happy"))
178     end }.
179
180 test_alias_existing_alias() ->
181   {"Register alias that already exists, error",
182     fun () ->
183       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>},
184                              {"facepalm", <<240,159,152,131>>}]),
185       ok = emoji:alias(S, "smiley", "happy"),
186       ?assertMatch({error, _}, emoji:alias(S, "facepalm", "happy"))
187     end }.
188
189 test_lookup_existing() ->
190   {"Lookup an existing shortcode",
191     fun () ->
192       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
193       ?assertEqual({ok, <<240,159,152,131>>}, emoji:lookup(S, "smiley"))
194     end }.
195
196 test_lookup_non_existing() ->
197   {"Lookup a non-existing shortcode, error",
198     fun () ->
199       {ok, S} = emoji:start([]),
200       ?assertEqual(no_emoji, emoji:lookup(S, "smiley"))
201     end }.
202
203 test_lookup_alias() ->
204   {"Register alias and then look it up",
205     fun () ->
206       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
207       ok = emoji:alias(S, "smiley", "happy"),
208       ?assertEqual({ok, <<240,159,152,131>>}, emoji:lookup(S, "happy"))
209     end }.
210
211 test_lookup_from_list_of_aliases() ->
212   {"Register alias for a shortcode that has multiple aliases and then look it up",
213     fun () ->
214       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
215       ok = emoji:alias(S, "smiley", "happy"),
216       ok = emoji:alias(S, "smiley", "content"),
217       ok = emoji:alias(S, "smiley", "glad"),
218       ?assertEqual({ok, <<240,159,152,131>>}, emoji:lookup(S, "content"))
219     end }.
220
221 test_lookup_alias_of_alias() ->
222   {"Register alias that already exists, error",
223     fun () ->
224       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
225       ok = emoji:alias(S, "smiley", "happy"),
226       ok = emoji:alias(S, "happy", "glad"),
```

```
227       ?assertEqual({ok, <<240,159,152,131>>}, emoji:lookup(S, "glad"))
228     end }.
229
230 test_delete_shortcode() ->
231   {"Delete an existing shortcode and then lookup, no_emoji",
232     fun () ->
233       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
234       emoji:delete(S, "smiley"),
235       ?assertEqual(no_emoji, emoji:lookup(S, "smiley"))
236     end }.
237
238 test_delete_alias() ->
239   {"Delete an alias and then look it up, no_emoji",
240     fun () ->
241       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
242       ok = emoji:alias(S, "smiley", "happy"),
243       emoji:delete(S, "happy"),
244       ?assertEqual(no_emoji, emoji:lookup(S, "happy"))
245     end }.
246
247 test_delete_non_existing() ->
248   {"Delete a non-existing shortcode/alias, no error, then lookup, no_emoji",
249     fun () ->
250       {ok, S} = emoji:start([]),
251       emoji:delete(S, "smiley"),
252       ?assertEqual(no_emoji, emoji:lookup(S, "smiley"))
253     end }.
254
255 test_analytics() ->
256   {"Create an analytics",
257     fun () ->
258       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
259       ?assertEqual(ok, emoji:analytics(S, "smiley", fun hit/2, "Counter", 0))
260     end }.
261
262 test_analytics_alias() ->
263   {"Create an analytics for an alias",
264     fun () ->
265       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
266       ok = emoji:alias(S, "smiley", "happy"),
267       ?assertEqual(ok, emoji:analytics(S, "happy", fun hit/2, "Counter", 0))
268     end }.
269
270 test_analytics_multiple() ->
271   {"Create multiple analytics for a short",
272     fun () ->
273       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
274       ok = emoji:alias(S, "smiley", "happy"),
275       ?assertEqual(ok, emoji:analytics(S, "happy", fun hit/2, "Counter", 0)),
276       ?assertEqual(ok, emoji:analytics(S, "smiley", fun accessed/2, "Accessed", []))
277     end }.
278
279 test_analytics_non_unique_label() ->
280   {"Create analytics with a non-unique label, error",
281     fun () ->
282       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
283       ok = emoji:alias(S, "smiley", "happy"),
```

```
284       ok = emoji:analytics(S, "happy", fun hit/2, "Counter", 0),
285       ?assertMatch({error, _}, emoji:analytics(S, "smiley", fun accessed/2, "Counter",
          []))
286     end }.
287
288 test_analytics_non_existing_short() ->
289   {"Create analytics for a short that does not exist, error",
290     fun () ->
291       {ok, S} = emoji:start([]),
292       ?assertMatch({error, _}, emoji:analytics(S, "smiley", fun accessed/2, "Accessed"
          , []))
293     end }.
294
295 test_get_analytics_empty() ->
296   {"Get analyitics for shortcode with no analytics",
297     fun () ->
298       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
299       ?assertEqual({ok, []}, emoji:get_analytics(S, "smiley"))
300     end }.
301
302 test_get_analytics_init() ->
303   {"Get initial state for analyitics function",
304     fun () ->
305       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
306       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
307       ?assertEqual({ok, [{"Counter", 0}]}, emoji:get_analytics(S, "smiley"))
308     end }.
309
310 test_get_analytics_multiple_funs_init() ->
311   {"Get initial state for multiple analyitics function",
312     fun () ->
313       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
314       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
315       ok = emoji:analytics(S, "smiley", fun accessed/2, "Accessed", []),
316       {ok, Analytics} = emoji:get_analytics(S, "smiley"),
317       ?assertEqual(true, lists:member({"Counter", 0}, Analytics)),
318       ?assertEqual(true, lists:member({"Accessed", []}, Analytics))
319     end }.
320
321 test_get_analytics_new_alias() ->
322   {"Create analytics, set alias, then get analytics of alias",
323     fun () ->
324       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
325       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
326       ok = emoji:alias(S, "smiley", "happy"),
327       ?assertEqual({ok, [{"Counter", 0}]}, emoji:get_analytics(S, "happy"))
328     end }.
329
330 test_get_analytics_after_lookups() ->
331   {"Get analyitics after three lookups",
332     fun () ->
333       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
334       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
335       {ok, _} = emoji:lookup(S, "smiley"),
336       {ok, _} = emoji:lookup(S, "smiley"),
337       {ok, _} = emoji:lookup(S, "smiley"),
338       ?assertEqual({ok, [{"Counter", 3}]}, emoji:get_analytics(S, "smiley"))
```

```
339     end }.
340
341 test_get_analytics_multiple_funs_after_lookups() ->
342   {"Get multiple analytics after three lookups",
343     fun () ->
344       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
345       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
346       ok = emoji:analytics(S, "smiley", fun accessed/2, "Accessed", []),
347       {ok, _} = emoji:lookup(S, "smiley"),
348       {ok, _} = emoji:lookup(S, "smiley"),
349       {ok, _} = emoji:lookup(S, "smiley"),
350       {ok, Analytics} = emoji:get_analytics(S, "smiley"),
351       ?assertEqual(true, lists:member({"Counter", 3}, Analytics)),
352       % have to match this way due to time constantly changing
353       ?assertMatch(_, maps:get("Accessed", maps:from_list(Analytics)))
354     end }.
355
356 test_get_analytics_after_lookups_alias() ->
357   {"Create analytics, lookup alias, then get analytics of shortcode",
358     fun () ->
359       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
360       ok = emoji:alias(S, "smiley", "happy"),
361       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
362       {ok, _} = emoji:lookup(S, "happy"),
363       {ok, _} = emoji:lookup(S, "happy"),
364       ?assertEqual({ok, [{"Counter", 2}]}, emoji:get_analytics(S, "smiley"))
365     end }.
366
367 test_get_analytics_non_existing_short() ->
368   {"Get analytics for a non-existing short",
369     fun () ->
370       {ok, S} = emoji:start([]),
371       ?assertMatch({error, _}, emoji:get_analytics(S, "smiley"))
372     end }.
373
374 test_remove_analytics_shortcode() ->
375   {"Remove analytics from a shortcode and then get, empty list",
376     fun () ->
377       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
378       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
379       emoji:remove_analytics(S, "smiley", "Counter"),
380       ?assertEqual({ok, []}, emoji:get_analytics(S, "smiley"))
381     end }.
382
383 test_remove_analytics_alias() ->
384   {"Add analytics for shortcode, add alias, remove analytics from alias and then get,
      empty list",
385     fun () ->
386       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
387       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
388       ok = emoji:alias(S, "smiley", "happy"),
389       emoji:remove_analytics(S, "happy", "Counter"),
390       ?assertEqual({ok, []}, emoji:get_analytics(S, "happy"))
391     end }.
392
393 test_remove_analytics_one_out_of_many() ->
394   {"Add two analytics for shortcode, remove one, and then get, shows state of other
```

```
    analytics",
395     fun () ->
396       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
397       ok = emoji:analytics(S, "smiley", fun hit/2, "Counter", 0),
398       ok = emoji:analytics(S, "smiley", fun accessed/2, "Accessed", []),
399       emoji:remove_analytics(S, "smiley", "Accessed"),
400       ?assertEqual({ok, [{"Counter", 0}]}, emoji:get_analytics(S, "smiley"))
401     end }.
402
403 test_remove_analytics_non_existing_label() ->
404   {"Remove analytics from a non-existing label, no error, then get, empty list",
405     fun () ->
406       {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
407       emoji:remove_analytics(S, "smiley", "Counter"),
408       ?assertEqual({ok, []}, emoji:get_analytics(S, "smiley"))
409     end }.
410
411 test_remove_analytics_non_existing_shortcode() ->
412   {"Remove analytics from a non-existing shortcode/alias, no error, then get, no_emoji
    ",
413     fun () ->
414       {ok, S} = emoji:start([]),
415       emoji:remove_analytics(S, "smiley", "Counter"),
416       ?assertEqual(no_emoji, emoji:lookup(S, "smiley"))
417     end }.
418
419 test_medium_new() ->
420   {"Create new shortcode with larger emoji server",
421     fun () ->
422       {ok, S} = emoji:start(someemoji:medium()),
423       ?assertEqual(ok, emoji:new_shortcode(S, "thisisanewemoji",
424                                            <<240,159,152,131>>))
425     end }.
426
427 test_medium_lookup() ->
428   {"Lookup shortcode with larger emoji server",
429     fun () ->
430       {ok, S} = emoji:start(someemoji:medium()),
431       ?assertMatch({ok, _}, emoji:lookup(S, "pensive face"))
432     end }.
433
434 test_medium_delete() ->
435   {"Delete shortcode with larger emoji server",
436     fun () ->
437       {ok, S} = emoji:start(someemoji:medium()),
438       emoji:delete(S, "pensive face"),
439       ?assertEqual(no_emoji, emoji:lookup(S, "pensive face"))
440     end }.
441
442 test_medium_alias() ->
443   {"Delete shortcode with larger emoji server",
444     fun () ->
445       {ok, S} = emoji:start(someemoji:medium()),
446       ok = emoji:alias(S, "pensive face", "newalias"),
447       ?assertMatch({ok, _}, emoji:lookup(S, "pensive face"))
448     end }.
449
```

```
test_medium_get_analytics() ->
  {"Add analytics, lookup, then get analytics with larger emoji server",
    fun () ->
      {ok, S} = emoji:start(someemoji:medium()),
      ok = emoji:analytics(S, "pensive face", fun hit/2, "Counter", 0),
      {ok, _} = emoji:lookup(S, "pensive face"),
      ?assertEqual({ok, [{"Counter", 1}]}, emoji:get_analytics(S, "pensive face"))
    end }.

test_medium_remove_analytics() ->
  {"Add analytics then remove with larger emoji server",
    fun () ->
      {ok, S} = emoji:start(someemoji:medium()),
      ok = emoji:analytics(S, "pensive face", fun hit/2, "Counter", 0),
      {ok, _} = emoji:lookup(S, "pensive face"),
      emoji:remove_analytics(S, "pensive face", "Counter"),
      ?assertMatch({ok, []}, emoji:get_analytics(S, "pensive face"))
    end }.

test_analytics_broken_fun() ->
  {"Create analytics with a function that throws an error, then lookup",
    fun () ->
      {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
      ok = emoji:analytics(S, "smiley", fun broken/2, "Broken", 0),
      ?assertMatch({ok, _}, emoji:lookup(S, "smiley"))
    end }.

test_analytics_forever_fun() ->
  {"Create analytics with a function that loops forever, then lookup",
    fun () ->
      {ok, S} = emoji:start([{"smiley", <<240,159,152,131>>}]),
      ok = emoji:analytics(S, "smiley", fun forever/2, "Forever", []),
      ?assertMatch({error, _}, emoji:lookup(S, "smiley"))
    end }.
```

Listing 2: Tests