Cooking in Erlang

ktn_recipe

Alguna vez viste codigo asi?

```
function(I) ->
 X = get_value_x(I),
 case X of
   bad_x ->
     do_thing_one(X);
   good_x ->
     Y = get_value_y(X),
     case Y of
       bad_y ->
         do_thing_two(X, Y);
       good_y ->
         Z = get_value_z(X, Y),
         case Z of
           bad_z ->
             do_thing_three(X, Y, Z);
           good_z ->
             do_thing_four(X, Y, Z)
         end
     end
 end.
```

```
function(I) ->
  X = get_value_x(I),
  function1(X).

function1(bad_x) ->
  do_thing_one(X);
function1(good_x) ->
  Y = get_value_y(X),
  function2(Y).
```

```
function2(bad_y) ->
  do_thing_two(X, Y);
function2(good_y) ->
  Z = get_value_z(X, Y),
  function3(Z).

function3(bad_z) ->
  do_thing_three(X, Y, Z);
function3(good_z) ->
  do_thing_four(X, Y, Z).
```

```
function(I) ->
  X = get_value_x(I),
  T1 =
    case X of
      bad  -> bad_x;
      good -> good_x
    end,

  Y = get_value_y(T1),
  T2 =
    case Y of
      bad  -> bad_y;
      good -> good_y
    end,

  Z = get_value_z(T1, T2)
  T3 =
    case Z of
      bad  -> bad_z;
      good -> good_z
    end,

  case {T1, T2, T3} of
    {bad_x, _, _} ->
      do_thing_one(X);
    {good_x, bad_y, _} ->
      do_thing_two(X, Y);
    {good_x, good_y, bad_z} ->
      do_thing_three(X, Y, Z);
    {good_x, good_y, good_z} ->
      do_thing_four(X, Y, Z)
  end.
```

ktn_recipe

```
ktn_recipe:run/2
ktn_recipe:run/4
ktn_recipe:verify/1
ktn_recipe:pretty_print/1
```

```
run(Mod, InitialState)

run(Transitions, ResultFun, ErrorFun, InitialState)
```

```
-callback transitions() -> transitions().
-callback process_result(term()) ->
term().
-callback process_error(term()) -> term().
```

BEFORE

```
delete_resource(Req, State) ->
  UserId         = State#state.user_id, % obtained from authentication
  ConversationId = State#state.conversation_id, % obtained from URL
  try
    Conversation = conversations:get_conversation(ConversationId),
    case Conversation of
      notfound ->
        Reason = {[{notfound, ConversationId}]},
        web_utils:return(?HTTP_NOT_FOUND, Reason, Req, State); % returns 404
      Conversation ->
        ContactId = conversations:get_contact_id(Conversation, UserId),
        Contact   = contacts:get_contact(UserId, ContactId),
```

```
      case contacts:get_status(Contact) of
        blocked_by ->
          conversations:delete(Conversation);
        _Status ->
          APIVersion = web_utils:get_version(Req),
          MinVersion = {date, {2014, 02, 14}},
          case web_utils:satisfies_version(APIVersion, MinVersion) of
            true ->
              conversations:clear_messages(Conversation, UserId);
            false ->
              contacts:block(UserId, ContactId),
              conversations:delete(UserId, ContactId)
          end,
      end,
      web_utils:return(?HTTP_NO_CONTENT, Reason, Req, State); % returns 204
  end
catch
  _:Error ->
    web_utils:return(?HTTP_INTERNAL_SERVER_ERROR, Req, State)
end.
```

AFTER

```
-module(proc_conversations_delete).

-export(
  [ transitions/0
  , process_result/1
  , process_error/2
  ]).
-export(
  [ get_conversation/1
  , get_contact/1
  , get_status/1
  , check_version/1
  ]).

transitions() ->
  [get_conversation, get_contact, get_status, check_version].
```

```
%% STEPS (these should, if possible, have no side effects)
get_conversation(#{conversation_id := ConversationId} = State) ->
  case conversations:get_conversation(ConversationId) of
    notfound ->
      {error, conversation_notfound};
    Conversation ->
      {ok, State#{conversation => Conversation}}
  end.
```

```
get_contact(#{conversation_id := ConversationId, user_id = UserId} = State) ->
  ContactId = conversations:get_contact_id(Conversation, UserId),
  Contact   = contacts:get_contact(UserId, ContactId),
  {ok, State#{contact_id => ContactId, contact => Contact}}.
```

```
get_status(#{contact := Contact} = State) ->
  case proplists:get_value(contact, State) of
    notfound ->
      {ok, State};
    Contact ->
      Status = contacts:get_status(Contact),
      {ok, State#{status => Status}}
  end.
```

```
check_version(#{api_version = APIVersion} = State) ->
  MinVersion       = {date, {2014, 02, 14}},
  SatisfiesVersion = web_utils:satisfies_version(APIVersion, MinVersion),
  {ok, State#{satisfies_version => SatisfiesVersion}}.
```

```erlang
%% RESULT PROCESSING
process_result(State) ->
  #{user_id := UserId
  , contact_id := ContactId
  , conversation := Conversation
  , status := Status
  , satisfies_version := SatisfiesVersion
  } = State,
  process_result(UserId, ContactId, Conversation, Status, SatisfiesVersion).
```

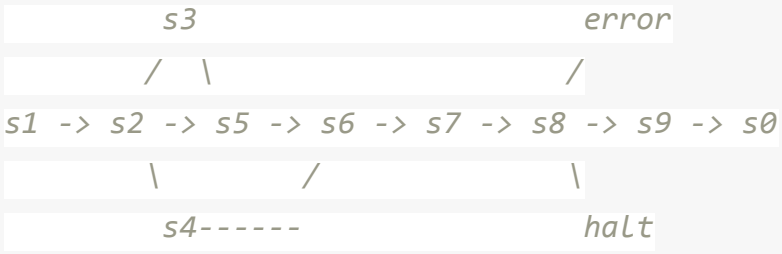
```
process_result(UserId, ContactId, Conversation, blocked_by, _SatisfiesVersion) ->
  conversations:delete(Conversation),
  {ok, ?HTTP_NO_CONTENT};
process_result(UserId, _ContactId, Conversation, _Status, true) ->
  conversations:clear_messages(Conversation, UserId),
  {ok, ?HTTP_NO_CONTENT};
process_result(UserId, ContactId, _Conversation, _Status, false) ->
  contacts:block(UserId, ContactId),
  conversations:delete(UserId, ContactId),
  {ok, ?HTTP_NO_CONTENT}.
```

```
%% ERROR HANDLERS
%% Here we can either format the error nicely for the client who invoked the
%% procedure, or roll back changes and side effects.
process_error(conversation_notfound, #{conversation_id := ConversationId} = State) ->
  {error, {notfound, ConversationId}}.
```

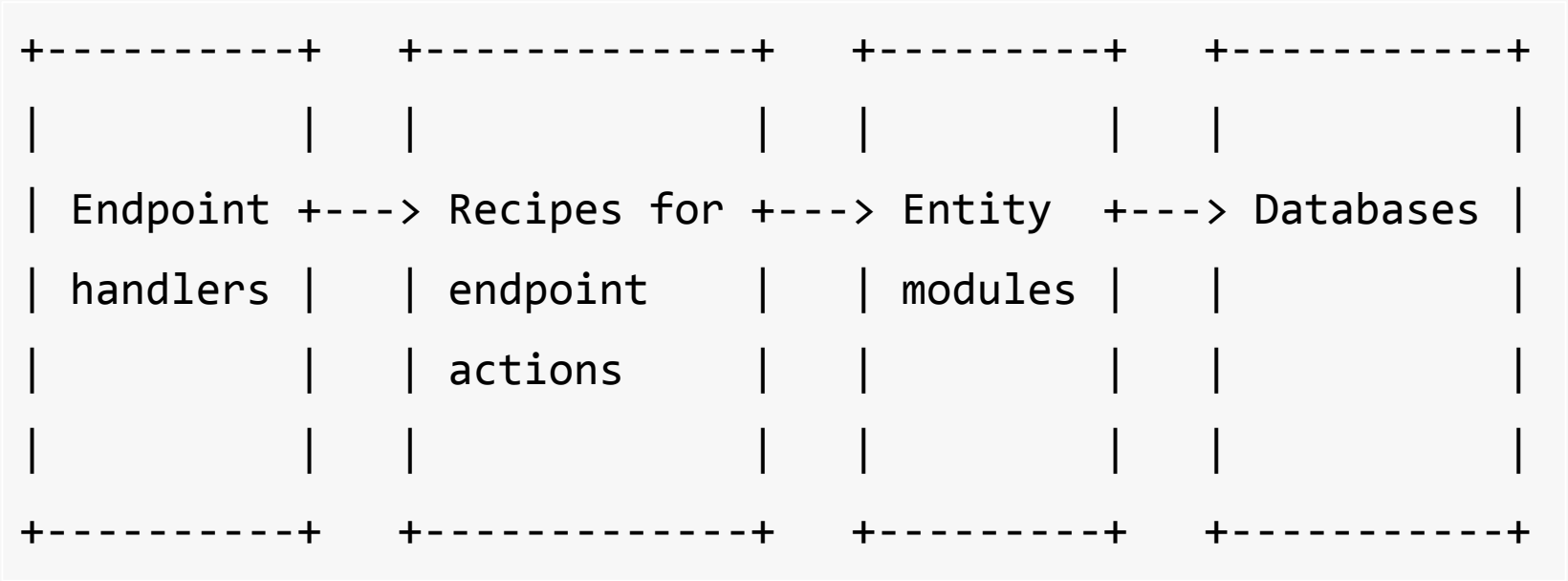
```
delete_resource(Req, #state{} = State) ->
  try
    UserId         = State#state.user_id,
    ConversationId = State#state.conversation_id,
    APIVersion     = web_utils:get_version(Req),

    % Initialize the procedure state with input from the request.
    ProcState =
      [ {user_id,         UserId}
      , {conversation_id, ConversationId}
      , {api_version,     APIVersion}
      ],
    case procedure:run(proc_conversation_delete, ProcState) of
      {ok, Result} ->
        web_utils:return(Result, Req, State)
      {error, {notfound, ConversationId}} ->
        web_utils:return(?HTTP_NOT_FOUND, Req, State)
    end
  catch
    _:Error ->
      web_utils:return(?HTTP_INTERNAL_SERVER_ERROR, Req, State)
  end.
```


BUT WAIT
SANDSTRÖM
ONTV COMPANY
THERE'S MORE!
quickmeme.com




```
transitions() ->
 [ s1
 , {s2, i1, s3}
 , {s2, i2, s4}
 , {s2, i3, s5}
 , {s3, i4, s5}
 , {s4, i5, s6}
 , s5
 , {s6, ok, fun ktn_recipe_example:s7/1}
 , {fun ktn_recipe_example:s7/1, ok, s8}
 , s8
 , {s8, i6, error}
 , {s8, i7, halt}
 , fun ktn_recipe_example:s9/1
 , s0
 ].
```


Endpoint handlers
Recipes for endpoint actions
Entity modules
Databases