SumoDB

https://github.com/inaka/sumo_db

A clean persistence layer for Erlang/OTP created to make your life easier

~$ whoami

Marcos Amilcar Almonacid

marcos@inaka.net / github: amilkr

hello@inaka.net
github: inaka
http://www.inaka.net

The God Module

```
handle_call({id, Id}, _From, State) ->

  Query = <<"select name from ", ?TABLE," where id = ?">>,
  emysql:prepare(get_name, Query),

  Result = case emysql:execute(mysql_pool, get_name, [Id]) of
    #result_packet{rows = []} -> {error, notfound};
    #result_packet{rows = Rows} -> process_results(Rows);
    Error -> lager:error("error: ~p with id ~p",[Error, Id]),{error,Error}
  end,

  {reply, Result, State};
```

No single responsibility

Untestable
&
Tightly Coupled

A lot of Duplicated Code

Hard to scale and refactor

```erlang
%INSERT
handle_call({find_or_update_nonce,Uid,Nonce,MaxTime,Force}, _From, State = #state{read_servers = ReadServers,write_servers = WriteServers}) ->
  emysql:prepare(find_nonce, <<"select nonce,TIME_TO_SEC(timediff(current_timestamp,created_at)) from ", ?USER_NONCE_TABLE " where uid = ?">>),
  Result = case post_sql:execute(random_server(ReadServers),find_nonce,[Uid]) of
    {result_packet,_,_,[],<<>>} ->
      emysql:prepare(insert_nonce, <<"insert into ", ?USER_NONCE_TABLE " (uid,nonce) VALUES (?,?)">>),
      post_sql:execute(random_server(WriteServers),insert_nonce,[Uid,Nonce]),
      Nonce;
    {result_packet,_,_,[[DbNonce,CreatedAt]],<<>>} ->
      case {CreatedAt,Force} of
        {Val,false} when Val =< MaxTime ->
          emysql:prepare(expire_nonce, <<"update ", ?USER_NONCE_TABLE " set created_at = date_add(created_at,interval - ? second) where uid  = ?">>),
          post_sql:execute(random_server(WriteServers),expire_nonce,[MaxTime,Uid]),
          DbNonce;
        {_Val,_} ->
          emysql:prepare(update_nonce, <<"update ", ?USER_NONCE_TABLE " set nonce  = ?, created_at = current_timestamp where uid  = ?">>),
          post_sql:execute(random_server(WriteServers),update_nonce,[Nonce,Uid]),
          Nonce
      end;
    Error -> lager:error("find_or_update_nonce: ~p",[Error]), {error,notfound}
  end,
  {reply, Result, State};

% SELECT
handle_call({get_post,Pid}, _From, State = #state{read_servers = ReadServers}) ->
  emysql:prepare(get_post,list_to_binary("select pid,content,short_id,status from " ++ ?POST_TABLE ++ " where pid = ?")),
  Result = case post_sql:execute(random_server(ReadServers),get_post,[Pid]) of
    {result_packet,_,_,[[_Pid,Data,ShortId,Status]],<<>>} -> json:encode(json:set_values([{short_id,ShortId},{status,Status}],json:decode(Data)));
    {result_packet,_,_,[],<<>>} -> lager:debug("get_post: pid ~p not found",[Pid]), {error,notfound};
    Error -> lager:error("get_post: error: ~p for Pid ~p",[Error,Pid]), {error,notfound}
  end,
  {reply, Result, State};

% UPDATE
handle_call({update_post_values,Pid,Values}, _From, State = #state{write_servers = WriteServers}) ->
    Post = json:decode(get_post(Pid)),
    UpdatedPost = json:set_values(Values,Post),

    ToBeApproved = case json:get_value(to_be_approved,UpdatedPost,<<"false">>) of
      <<"true">> -> 1;
      <<"false">> -> 0
    end,

    Status = json:get_value(status,UpdatedPost,1),

    Uploaded = case json:get_value(uploaded,UpdatedPost,<<"false">>) of
      true -> 1;
      <<"true">> -> 1;
      <<"false">> -> 0
    end,

    Ts = json:get_value(ts,UpdatedPost),

    emysql:prepare(update_post_values, list_to_binary("update " ++ ?POST_TABLE ++ " set content = ?, to_be_approved = ?, status = ?, ts = ?, uploaded = ? where pid = ?")),
    case post_sql:execute(random_server(WriteServers),update_post_values,[json:encode(UpdatedPost),ToBeApproved,Status,Ts,Uploaded,Pid]) of
      {ok_packet,_,_,_,_,_,_} -> ok;
      Error -> lager:error("update_post_values: error: ~p",[Error]), ok
    end,
  {reply, UpdatedPost, State};


% DELETE
handle_call({delete_topics, AllIds}, _From, State) ->
  AllIdsString = string:join([integer_to_list(X) || X <- AllIds], ", "),
  lager:debug("Deleting post topics with ids" ++ AllIdsString),
  emysql:prepare(delete_topics, list_to_binary("DELETE FROM " ++ ?TOPICS_TABLE ++ " where id IN (" ++ AllIdsString ++")")),
  case post_sql:execute(random_server(State#state.write_servers), delete_topics, []) of
    {ok_packet,_,_,_,_,_,_} -> ok;
    Error -> lager:error("delete_topics: error: ~p",[Error])
  end,
  {reply, ok, State};
```

```
%INSERT
handle_call({find_or_update_nonce,Uid,Nonce,MaxTime,Force}, _From, State = #state{read_servers = ReadServers,write_servers = WriteServers}) ->
  emysql:prepare(find_nonce, <<"select nonce,TIME_TO_SEC(timediff(current_timestamp,created_at)) from ", ?USER_NONCE_TABLE " where uid = ?">>),
  Result = case post_sql:execute(random_server(ReadServers),find_nonce,[Uid]) of
    {result_packet,_,_,[],<<>>} ->
      emysql:prepare(insert_nonce, <<"insert into ", ?USER_NONCE_TABLE " (uid,nonce) VALUES (?,?)">>),
      post_sql:execute(random_server(WriteServers),insert_nonce,[Uid,Nonce]),
      Nonce;
    {result_packet,_,_,[[DbNonce,CreatedAt]],<<>>} ->
      case {CreatedAt,Force} of
        {Val,false} when Val =< MaxTime ->
          emysql:prepare(expire_nonce, <<"update ", ?USER_NONCE_TABLE " set created_at = date_add(created_at,interval - ? second) where uid  = ?">>),
          post_sql:execute(random_server(WriteServers),expire_nonce,[MaxTime,Uid]),
          DbNonce;
        {_Val,_} ->
          emysql:prepare(update_nonce, <<"update ", ?USER_NONCE_TABLE " set nonce  = ?, created_at = current_timestamp where uid  = ?">>),
          post_sql:execute(random_server(WriteServers),update_nonce,[Nonce,Uid]),
          Nonce
      end;
    Error -> lager:error("find_or_update_nonce: ~p",[Error]), {error,notfound}
  end,
  {reply, Result, State};

% SELECT
handle_call({get_post,Pid}, _From, State = #state{read_servers = ReadServers}) ->
  emysql:prepare(get_post,list_to_binary("select pid,content,short_id,status from " ++ ?POST_TABLE ++ " where pid = ?")),
  Result = case post_sql:execute(random_server(ReadServers),get_post,[Pid]) of
    {result_packet,_,_,[[_Pid,Data,ShortId,Status]],<<>>} -> json:encode(json:set_values([{short_id,ShortId},{status,Status}],json:decode(Data)));
    {result_packet,_,_,[],<<>>} -> lager:debug("get_post: pid ~p not found",[Pid]), {error,notfound};
    Error -> lager:error("get_post: error: ~p for Pid ~p",[Error,Pid]), {error,notfound}
  end,
  {reply, Result, State};

% UPDATE
handle_call({update_post_values,Pid,Values}, _From, State = #state{write_servers = WriteServers}) ->
  Post = json:decode(get_post(Pid)),
  Updated = json:set_values(Values,Post),

  ToBeApproved = case json:get_value(to_be_approved,Updated,<<"false">>) of
    <<"true">> -> 1;
    <<"false">> -> 0
  end,

  Status = json:get_value(status,UpdatedPost,1),

  Uploaded = case json:get_value(uploaded,UpdatedPost,<<"false">>) of
    true -> 1;
    <<"true">> -> 1;
    <<"false">> -> 0
  end,

  Ts = json:get_value(ts,UpdatedPost),

  emysql:prepare(update_post_values, list_to_binary("update " ++ ?POST_TABLE ++ " set content = ?, to_be_approved = ?, status = ?, ts = ?, uploaded = ? where pid = ?")),
  case post_sql:execute(random_server(WriteServers),update_post_values,[json:encode(UpdatedPost),ToBeApproved,Status,Ts,Uploaded,Pid]) of
    {ok_packet,_,_,_,_,_,_} -> ok;
    Error -> lager:error("update_post_values: error: ~p",[Error]), ok
  end,
  {reply, UpdatedPost, State};


% DELETE
handle_call({delete_topics, AllIds}, _From, State) ->
  AllIdsString = string:join([integer_to_list(X) || X <- AllIds], ", "),
  lager:debug("Deleting post topics with ids" ++ AllIdsString),
  emysql:prepare(delete_topics, list_to_binary("DELETE FROM " ++ ?TOPICS_TABLE ++ " where id IN (" ++ AllIdsString ++")")),
  case post_sql:execute(random_server(State#state.write_servers), delete_topics, []) of
    {ok_packet,_,_,_,_,_,_} -> ok;
    Error -> lager:error("delete_topics: error: ~p",[Error])
  end,
  {reply, ok, State};
```

NIGHTMARE

The Sumo way

```
_NewUser = sumo:persist(mywebsite_user, User).

_Users = sumo:find_by(mywebsite_user, [{age, Age}]).

sumo:find_by(mywebsite_user, [{age, Age}], Limit, Offset).

sumo:delete(mywebsite_user, Id).

sumo:create_schema().
```

How does it work?


REPOSITORIES
HISTORY.COM

God Module

Multiple Entities

SQL driver

State translation

Business Logic

DB connection details

ActiveRecord

One Entity

SQL driver

State translation

Business Logic

DB connection details

STILL A MESS!

One Entity

Business Logic

State translation

SQL

DB driver details

One Entity

Business Logic

State translation

SQL

DB driver details

Domain Entity

One Entity

Business Logic

Domain Entity

State translation

SQL

Repository

DB driver details

One Entity

Business Logic

Domain Entity

State translation

SQL

Repository

DB driver details

Storage Backend

Domain Entity

=

sumo_doc

sumo_schema/0

Called when creating a schema in the db for entities of this type

sumo_schema/0

Called when creating a schema in the db for entities of this type

```
-spec sumo_schema() -> #sumo_schema{}.
sumo_schema() ->
  sumo:new_schema(?MODULE, [
    sumo:new_field(id, integer, [not_null, auto_increment, id]),
    sumo:new_field(title, string, [{length, 128}, not_null, unique]),
    sumo:new_field(content, text),
    sumo:new_field(author_id, integer, [index])
  ]).
```

sumo_sleep/1

Called when the entity is going to be persisted

sumo_sleep/1

Called when the entity is going to be persisted

```
-spec sumo_sleep(user()) -> proplists:proplists().
sumo_sleep(User) ->
  [
    {id, User#user.id},
    {email, User#user.id},
    {first_name, User#user.first_name},
    {last_name, User#user.last_name}
  ].
```

sumo_wakeup/1

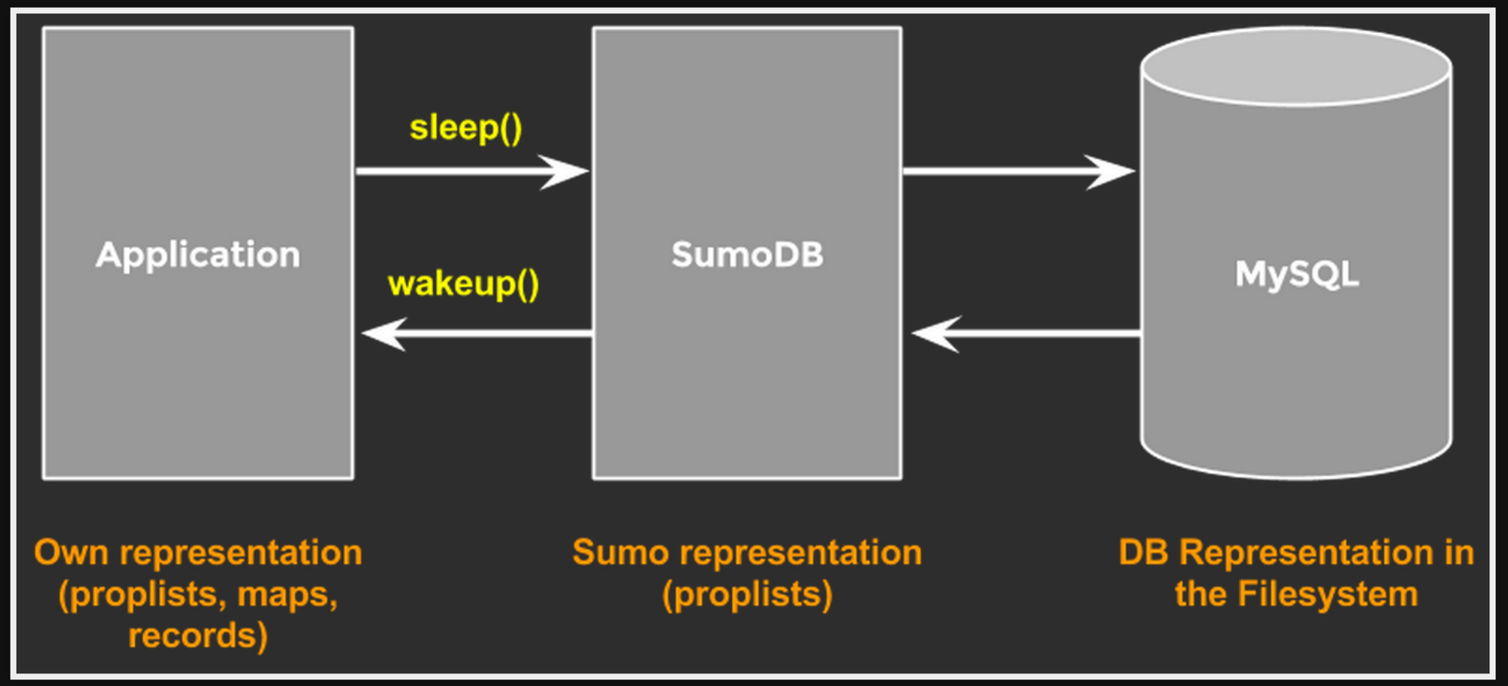
Called when loading the entity from the db

sumo_wakeup/1

Called when loading the entity from the db

```
-spec sumo_wakeup(proplists:proplist()) -> user().
sumo_wakeup(User) ->
  #user{
    id = proplists:get_value(id, User),
    email = proplists:get_value(email, User),
    first_name = proplists:get_value(first_name, User),
    last_name = proplists:get_value(last_name, User)
  }.
```

Sumo Doc - Life Cycle



Repository

=

sumo_repo

sumo repo functions

```
create_schema(Schema, State).

persist(Doc, State).

delete(DocName, Id, State).

delete_by(DocName, Conditions, State).

delete_all(DocName, State).

find_by(DocName, Conditions, State).

find_by(DocName, Conditions, Limit, Offset, State).
```

Storage Backend

=

sumo_backend

Domain Events

schema_created

created

updated

deleted

deleted_all

So...

So...
How can we use it?

```erlang
-module(user).
-behaviour(sumo_doc).

-record(user, {
  id :: pos_integer(),
  username :: binary()
}).

-type user() :: #user{}.
-export_type([user/0]).

-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE, [
    sumo:new_field(id, integer, [not_null, auto_increment, id]),
    sumo:new_field(username, string, [{length, 128}, not_null, unique])
  ]).

-spec sumo_wakeup(proplists:proplist()) -> user().
sumo_wakeup(User) ->
  #user{
    id = proplists:get_value(id, User),
    username = proplists:get_value(username, User)
  }.

-spec sumo_sleep(user()) -> proplists:proplists().
sumo_sleep(User) ->
  [{id, User#user.id},
   {username, User#user.id}].

-spec new(binary()) -> user().
new(Username) ->
  #user{username = Username}.
```

```
-behaviour(sumo_doc).

-record(user, {
  id :: pos_integer(),
  username :: binary()
}).
```

```
-export_type([user/0]).

-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE, [
    sumo:new_field(id, integer, [not_null, auto_increment, id]),
    sumo:new_field(username, string, [{length, 128}, not_null, unique])
  ]).

-spec sumo_wakeup(proplists:proplist()) -> user().
sumo_wakeup(User) ->
  #user{
    id = proplists:get_value(id, User),
    username = proplists:get_value(username, User)
  }.

-spec sumo_sleep(user()) -> proplists:proplists().
sumo_sleep(User) ->
  [{id, User#user.id},
   {username, User#user.id}].

-spec new(binary()) -> user().
new(Username) ->
  #user{username = Username}.
```

```erlang
-module(user).
-behaviour(sumo_doc).

-record(user, {
  id :: pos_integer(),
  username :: binary()
}).

-type user() :: #user{}.
-export_type([user/0]).

sumo_schema() ->
  sumo:new_schema(?MODULE, [
    sumo:new_field(id, integer, [not_null, auto_increment
    sumo:new_field(username, string, [{length, 128}, not_
  ]).

-spec sumo_wakeup(proplists:proplist()) -> user().
sumo_wakeup(User) ->
  #user{
    id = proplists:get_value(id, User),
    username = proplists:get_value(username, User)
  }.

-spec sumo_sleep(user()) -> proplists:proplists().
sumo_sleep(User) ->
  [{id, User#user.id},
   {username, User#user.id}].

-spec new(binary()) -> user().
new(Username) ->
  #user{username = Username}.
```

```erlang
-module(user).
-behaviour(sumo_doc).

-record(user, {
  id :: pos_integer(),
  username :: binary()
}).

-type user() :: #user{}.
-export_type([user/0]).

-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE, [

-spec sumo_wakeup(proplists:proplist()) -> user().
sumo_wakeup(User) ->
  #user{
    id = proplists:get_value(id, User),
    username = proplists:get_value(username, User)
  }.

-spec sumo_sleep(user()) -> proplists:proplists().
sumo_sleep(User) ->
  [{id, User#user.id},
   {username, User#user.id}].

new(Username) ->
  #user{username = Username}.
```

```erlang
-module(user).
-behaviour(sumo_doc).

-record(user, {
  id :: pos_integer(),
  username :: binary()
}).

-type user() :: #user{}.
-export_type([user/0]).

-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE, [
    sumo:new_field(id, integer, [not_null, auto_increment, id]),
    sumo:new_field(username, string, [{length, 128}, not_null, unique])
  ]).

-spec sumo_wakeup(proplists:proplist()) -> user().
sumo_wakeup(User) ->
  #user{
    id = proplists:get_value(id, User),
    username = proplists:get_value(username, User)
  }.

-spec sumo_sleep(user()) -> proplists:proplists().
sumo_sleep(User) ->
```

```erlang
-spec new(binary()) -> user().
new(Username) ->
  #user{username = Username}.
```

```
{sumo_db, [
  {docs, [
    {user, user_repo}
  ]},

  {repositories, [
    {user_repo, sumo_repo_mysql, [
      {storage_backend, mysql_backend},
      {workers, 10}
    ]}
  ]},

  {storage_backends, [
    {mysql_backend, sumo_backend_mysql, [
      {username, "user"},
      {password, "pass"},
      {host, "127.0.0.1"},
      {port, 3306},
      {database, "db"}
      {poolsize, 5}
    ]}
  ]}
]}
```

```
  {docs, [
    {user, user_repo}
  ]},
```

```
  {repositories, [
    {user_repo, sumo_repo_mysql, [
      {storage_backend, mysql_backend},
      {workers, 10}
    ]}
  ]},

  {storage_backends, [
    {mysql_backend, sumo_backend_mysql, [
      {username, "user"},
      {password, "pass"},
      {host, "127.0.0.1"},
      {port, 3306},
      {database, "db"}
      {poolsize, 5}
    ]}
  ]}
]}
```

```
{sumo_db, [
  {docs, [
    {user, user_repo}
  ]},
  {repositories, [
    {user_repo, sumo_repo_mysql, [
      {storage_backend, mysql_backend},
      {workers, 10}
    ]}
  ]},
        {username, "user"},
        {password, "pass"},
        {host, "127.0.0.1"},
        {port, 3306},
        {database, "db"}
        {poolsize, 5}
      ]}
    ]}
]}
```

```
{sumo_db, [
  {docs, [
    {user, user_repo}
  ]},

  {repositories, [
    {user_repo, sumo_repo_mysql, [
      {storage_backend, mysql_backend},
```

```
{storage_backends, [
  {mysql_backend, sumo_backend_mysql,
    {username, "user"},
    {password, "pass"},
    {host, "127.0.0.1"},
    {port, 3306},
    {database, "db"}
    {poolsize, 5}
  ]}
]}
```

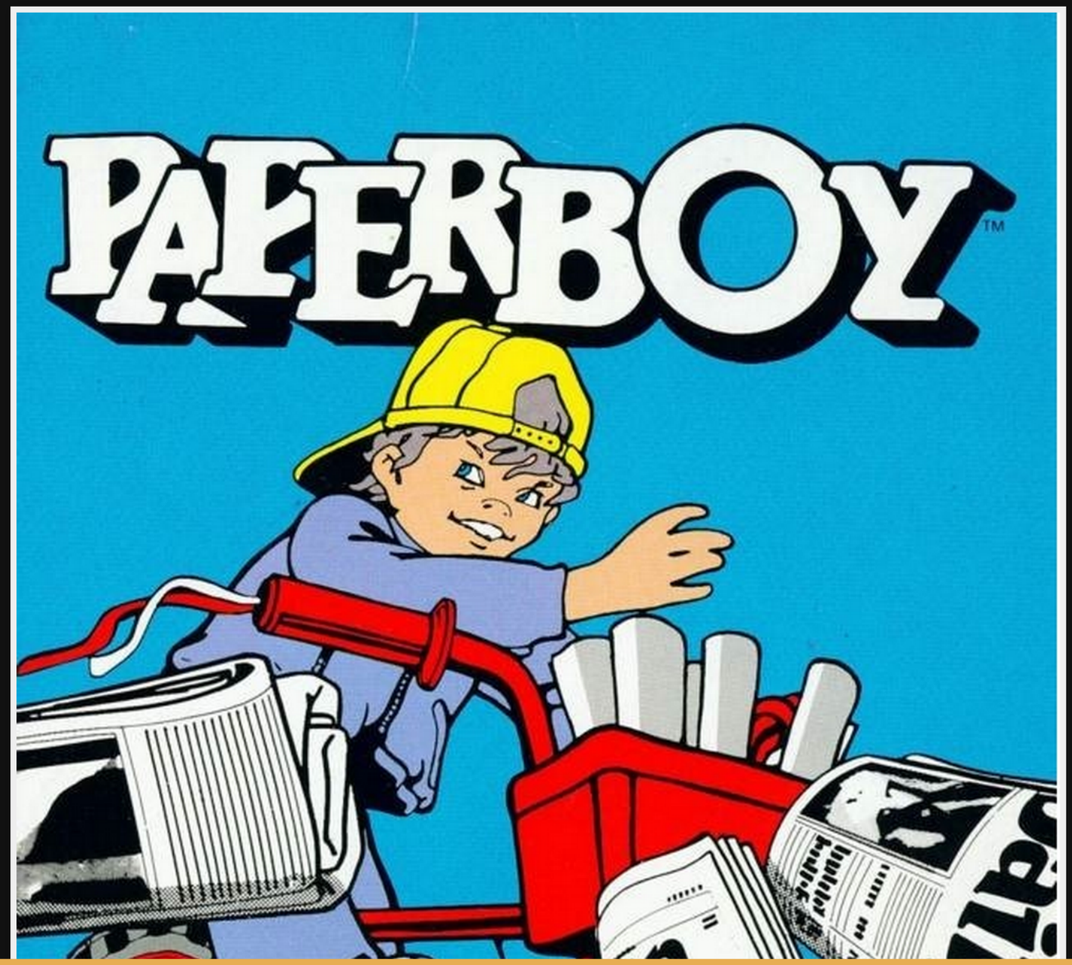
And That's it!

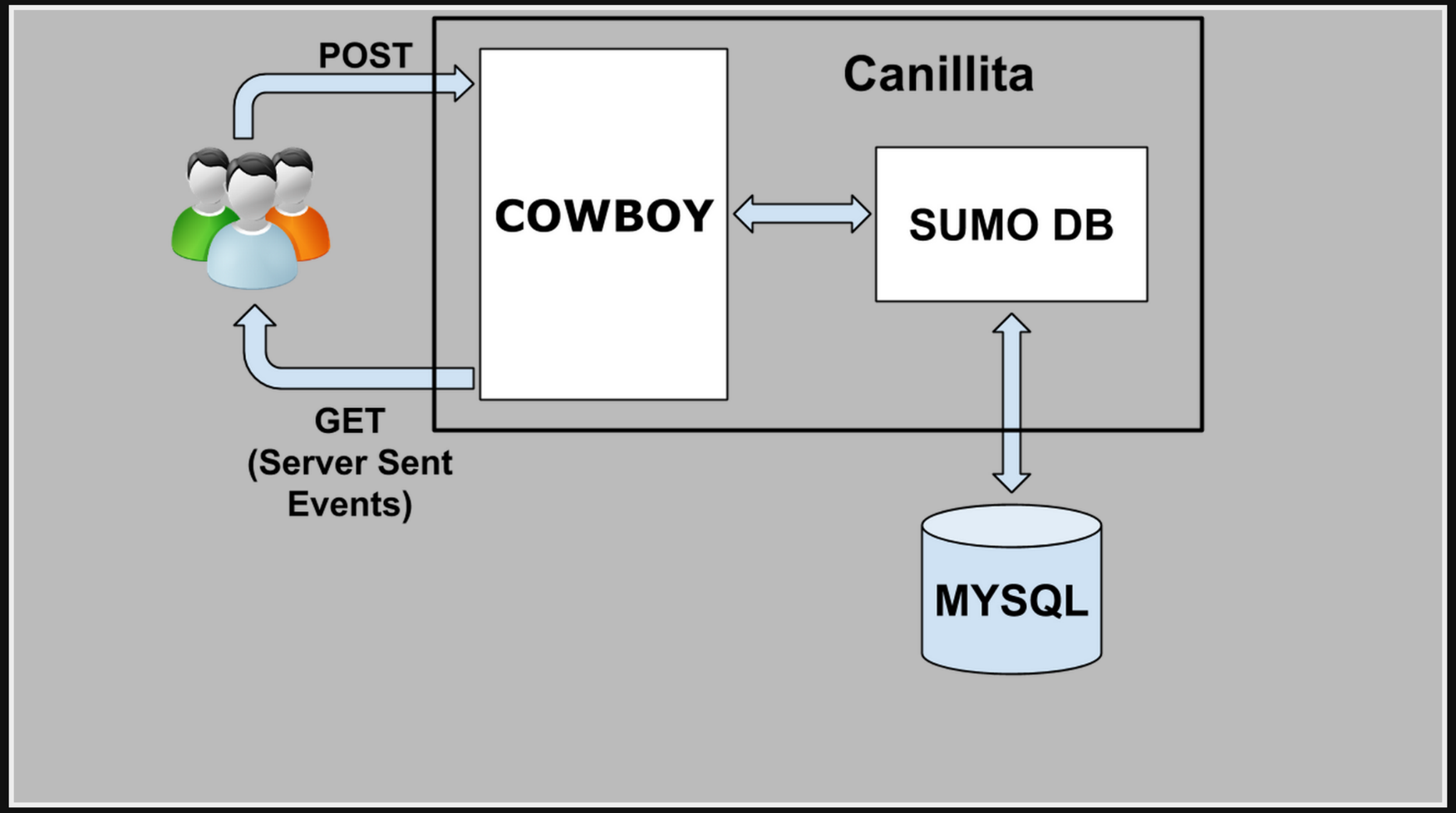
```erlang
-module(user_service).

-export([new/1, get/1, delete/1]).

-spec new(binary()) -> user:user().
new(Username) ->
  User = user:new(Username),
  sumo:persist(user, User).

-spec get(pos_integer()) -> user:user().
get(Id) ->
  sumo:find(user, Id).

-spec delete(pos_integer()) -> ok.
delete(Id) ->
  sumo:delete(user, Id).
```

SumoDB + Cowboy

Canillita

SumoDB + Cowboy




POST
Canillita
COWBOY
SUMO DB
GET
(Server Sent
Events)
MYSQL

GET /news

```
curl -vX GET http://localhost:4004/news
```

```
> GET /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
>
< HTTP /1.1 200 OK
< transfer-encoding: chunked
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
< content-type: text/event-stream
<
event: old_news_flash
data: The first news flash
data: This is an old news flash. keep waiting for more
```

GET /news

```
curl -vX GET http://localhost:4004/news
```

```
> GET /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
>
< HTTP /1.1 200 OK
< transfer-encoding: chunked
< connection: keep-alive
< server: Cowboy
< date: Thu, 07 Nov 2013 14:31:10 GMT
```

```
event: old_news_flash
data: The first news flash
data: This is an old news flash. keep waiting for more
```

POST /news

```
curl -vX POST http://localhost:4004/news -H"Content -Type:application/json"
  -d'{ "title": "This is the title for the news flash",
       "content": "And this is the content...." }'
```

```
> POST /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
> Content-Type:application/json
> Content-Length: 50
>
< HTTP /1.1 204 No Content
< connection: keep-alive
< server: Cowboy
< date: Fri, 08 Nov 2013 20:06:01 GMT
< content-length: 0
<
```

GET /news

```
curl -vX GET http://localhost:4004/news
```

```
> GET /news HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:4004
> Accept: */*
>

< HTTP /1.1 200 OK
< transfer -encoding: chunked
< connection: keep-alive
```

```
event: old_news_flash
data: The first news flash
data: This is an old news flash. keep waiting for more

event: news_flash
data: This is the title for the news flash
data: And this is the content...
```

```erlang
-module(canillita_news).
-behaviour(sumo_doc).

-record(news_flash, {
  id          :: pos_integer(),
  title       :: binary(),
  content     :: binary()
}).

-export([new/2, get_id/1, get_title/1, get_content/1]).
-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE,
    [sumo:new_field(id,           integer,  [id, not_null, auto_increment]),
     sumo:new_field(title,        text,     [not_null]),
     sumo:new_field(content,      text,     [not_null])]).

-spec sumo_sleep(#news_flash{}) -> proplists:proplists().
sumo_sleep(NewsFlash) ->
  [{id,          NewsFlash#news_flash.id},
   {title,       NewsFlash#news_flash.title},
   {content,     NewsFlash#news_flash.content}].

-spec sumo_wakeup(proplists:proplists()) -> #news_flash{}.
sumo_wakeup(NewsFlash) ->
  #news_flash{
    id          = proplists:get_value(id, NewsFlash),
    title       = proplists:get_value(title, NewsFlash),
    content     = proplists:get_value(content, NewsFlash)
  }.

new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  #news_flash{title = Title, content = Content}.

get_id(NewsFlash) -> NewsFlash#news_flash.id.
get_title(NewsFlash) -> NewsFlash#news_flash.title.
get_content(NewsFlash) -> NewsFlash#news_flash.content.
```

```
-behaviour(sumo_doc).

-record(news_flash, {
  id           :: pos_integer(),
  title        :: binary(),
  content      :: binary()
}).

sumo_schema() ->
  sumo:new_schema(?MODULE,
    [sumo:new_field(id,           integer,  [id, not_null, auto_increment]),
     sumo:new_field(title,        text,     [not_null]),
     sumo:new_field(content,      text,     [not_null])]).

-spec sumo_sleep(#news_flash{}) -> proplists:proplists().
sumo_sleep(NewsFlash) ->
  [{id,          NewsFlash#news_flash.id},
   {title,       NewsFlash#news_flash.title},
   {content,     NewsFlash#news_flash.content}].

-spec sumo_wakeup(proplists:proplists()) -> #news_flash{}.
sumo_wakeup(NewsFlash) ->
  #news_flash{
    id          = proplists:get_value(id, NewsFlash),
    title       = proplists:get_value(title, NewsFlash),
    content     = proplists:get_value(content, NewsFlash)
  }.

new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  #news_flash{title = Title, content = Content}.

get_id(NewsFlash) -> NewsFlash#news_flash.id.
get_title(NewsFlash) -> NewsFlash#news_flash.title.
get_content(NewsFlash) -> NewsFlash#news_flash.content.
```

```erlang
-module(canillita_news).
-behaviour(sumo_doc).

-record(news_flash, {
  id          :: pos_integer(),
  title       :: binary(),
  content     :: binary()
}).

-export([new/2, get_id/1, get_title/1, get_content/1]).
```

```erlang
sumo_schema() ->
  sumo:new_schema(?MODULE,
    [sumo:new_field(id,           integer,  [id, not_null, auto_increment]),
     sumo:new_field(title,        text,     [not_null]),
     sumo:new_field(content,      text,     [not_null])]).
```

```erlang
-spec sumo_sleep(#news_flash{}) -> proplists:proplists().
sumo_sleep(NewsFlash) ->
  [{id,          NewsFlash#news_flash.id},
   {title,       NewsFlash#news_flash.title},
   {content,     NewsFlash#news_flash.content}].

-spec sumo_wakeup(proplists:proplists()) -> #news_flash{}.
sumo_wakeup(NewsFlash) ->
  #news_flash{
    id          = proplists:get_value(id, NewsFlash),
    title       = proplists:get_value(title, NewsFlash),
    content     = proplists:get_value(content, NewsFlash)
  }.

new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  #news_flash{title = Title, content = Content}.

get_id(NewsFlash) -> NewsFlash#news_flash.id.
get_title(NewsFlash) -> NewsFlash#news_flash.title.
get_content(NewsFlash) -> NewsFlash#news_flash.content.
```

```erlang
-module(canillita_news).
-behaviour(sumo_doc).

-record(news_flash, {
  id          :: pos_integer(),
  title       :: binary(),
  content     :: binary()
}).

-export([new/2, get_id/1, get_title/1, get_content/1]).
-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE,
    [sumo:new_field(id,           integer,  [id, not_null, auto_increment]),

-spec sumo_sleep(#news_flash{}) -> proplists:proplists().
sumo_sleep(NewsFlash) ->
  [{id,          NewsFlash#news_flash.id},
   {title,       NewsFlash#news_flash.title},
   {content,     NewsFlash#news_flash.content}].

-spec sumo_wakeup(proplists:proplists()) -> #news_flash{}.
sumo_wakeup(NewsFlash) ->
  #news_flash{
    id          = proplists:get_value(id, NewsFlash),
    title       = proplists:get_value(title, NewsFlash),
    content     = proplists:get_value(content, NewsFlash)
  }.

  Now = {datetime, calendar:universal_time()},
  #news_flash{title = Title, content = Content}.

get_id(NewsFlash) -> NewsFlash#news_flash.id.
get_title(NewsFlash) -> NewsFlash#news_flash.title.
get_content(NewsFlash) -> NewsFlash#news_flash.content.
```

```erlang
-module(canillita_news).
-behaviour(sumo_doc).

-record(news_flash, {
  id          :: pos_integer(),
  title       :: binary(),
  content     :: binary()
}).

-export([new/2, get_id/1, get_title/1, get_content/1]).
-export([sumo_schema/0, sumo_wakeup/1, sumo_sleep/1]).

sumo_schema() ->
  sumo:new_schema(?MODULE,
    [sumo:new_field(id,            integer,  [id, not_null, auto_increment]),
     sumo:new_field(title,         text,     [not_null]),
     sumo:new_field(content,       text,     [not_null])]).

-spec sumo_sleep(#news_flash{}) -> proplists:proplists().
sumo_sleep(NewsFlash) ->
  [{id,          NewsFlash#news_flash.id},
   {title,       NewsFlash#news_flash.title},
   {content,     NewsFlash#news_flash.content}].

-spec sumo_wakeup(proplists:proplists()) -> #news_flash{}.
sumo_wakeup(NewsFlash) ->
  #news_flash{
    id         = proplists:get_value(id, NewsFlash),
```

```erlang
new(Title, Content) ->
  Now = {datetime, calendar:universal_time()},
  #news_flash{title = Title, content = Content}.

get_id(NewsFlash) -> NewsFlash#news_flash.id.
get_title(NewsFlash) -> NewsFlash#news_flash.title.
get_content(NewsFlash) -> NewsFlash#news_flash.content.
```

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),

  case json_decode(Body) of
    {Params} ->
      Title   = proplists:get_value(<<"title">>,    Params, <<"Generic News">>),
      Content = proplists:get_value(<<"content">>,  Params, <<"">>),

      NewsFlash = canillita_news:new(Title, Content),
      sumo:persist(canillita_news, NewsFlash),
      notify(NewsFlash),

      {true, Req1, State};
    {bad_json, Reason} ->
      {ok, Req2} = cowboy_req:reply(400, [], jiffy:encode(Reason), Req1),
      {halt, Req2, State}
  end.

handle_get(Req) ->
  {ok, Req1} =
    cowboy_req:chunked_reply(
      200, [{"content-type", <<"text/event-stream">>}], Req),

  LatestNews = lists:reverse(sumo:find_all(canillita_news)),

  lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Req1)
    end, LatestNews),

  ok = pg2:join(canillita_listeners, self()),

  {loop, Req1, {}}.
```

```
29 handle_post(Req, State) ->
30   {ok, Body, Req1} = cowboy_req:body(Req),
31
32   case json_decode(Body) of
33     {Params} ->
34       Title   = proplists:get_value(<<"title">>,    Params, <<"Generic News">>),
35       Content = proplists:get_value(<<"content">>,  Params, <<"">>),
36
37       NewsFlash = canillita_news:new(Title, Content),
38       sumo:persist(canillita_news, NewsFlash),
39       notify(NewsFlash),
40
41       {true, Req1, State};
42     {bad_json, Reason} ->
43       {ok, Req2} = cowboy_req:reply(400, [], jiffy:encode(Reason), Req1),
44       {halt, Req2, State}
45   end.
46
47 handle_get(Req) ->
48   {ok, Req1} =
49     cowboy_req:chunked_reply(
50       200, [{"content-type", <<"text/event-stream">>}], Req),
51
52   LatestNews = lists:reverse(sumo:find_all(canillita_news)),
53
54   lists:foreach(
55     fun(NewsFlash) ->
56       send_flash(NewsFlash, Req1)
57     end, LatestNews),
58
59   ok = pg2:join(canillita_listeners, self()),
60
61   {loop, Req1, {}}.
```

```
handle_post(Req, State) ->
  {ok, Body, Req1} = cowboy_req:body(Req),

  case json_decode(Body) of
    {Params} ->
      Title   = proplists:get_value(<<"title">>,    Params, <<"Generic News">>),
      Content = proplists:get_value(<<"content">>,  Params, <<"">>),

      NewsFlash = canillita_news:new(Title, Content),
      sumo:persist(canillita_news, NewsFlash),
      notify(NewsFlash),

      {true, Req1, State};
    {bad_json, Reason} ->
      {ok, Req2} = cowboy_req:reply(400, [], jiffy:encode(Reason), Req1),
      {halt, Req2, State}
  end.

handle_get(Req) ->
  {ok, Req1} =
    cowboy_req:chunked_reply(
      200, [{<<"content-type">>, <<"text/event-stream">>}], Req),

  LatestNews = lists:reverse(sumo:find_all(canillita_news)),

  lists:foreach(
    fun(NewsFlash) ->
      send_flash(NewsFlash, Req1)
    end, LatestNews),

  ok = pg2:join(canillita_listeners, self()),

  {loop, Req1, {}}.
```

Current State

MySQL - MongoDB

Redis - SQLite3

Mnesia - ETS

DynamoDB

Last Words

SumoDB

erlang + databases

SumoDB

is our way...

SumoDB

is our way...

and it can be yours

Thank You!

https://github.com/inaka/sumo_db

https://github.com/inaka/erlang_training

Questions?