

# Phoenix Tutorial - OSCON 17

Marc Sugiyama - Erlang Solutions

Marc's Mac -

- updated phoenix: mix archive.install  
[https://github.com/phoenixframework/archives/raw/master/phoenix\\_new.ez](https://github.com/phoenixframework/archives/raw/master/phoenix_new.ez)
- updated node: brew upgrade node

Required Software

- Erlang
- Elixir
- Phoenix - mix archive.install  
[https://github.com/phoenixframework/archives/raw/master/phoenix\\_new.ez](https://github.com/phoenixframework/archives/raw/master/phoenix_new.ez)
- Posgres - brew install postgresql; initdb /usr/local/var/postgres
- Node JS - brew install node.js

## Script

1. start postgres:
  - a. **postgres -D /usr/local/var/postgres**
  - b. psql postgres to log in
    - i. \l lists database
    - ii. may need to "drop database pchat\_dev" to begin at a clean starting point
2. Create app to chat with pictures: pchat
3. **mix phoenix.new pchat**
4. **cd pchat**
5. create application's database
  - a. check login credentials to postgres in config/dev.exs
  - b. **mix ecto.create**
    - i. asks to install local rebar
    - ii. get errors, but says "stop normal" so ok?
6. smoke test
  - a. **iex -S mix phoenix.server**
  - b. see start page: <http://localhost:4000>
  - c. keep it running - in development mode updates are dynamic
    - i. Talk about Erlang BEAM
7. phoenix directory structure
  - a. **\_build** - build artifacts
  - b. **config** - runtime configuration
  - c. **deps** - dependent applications
  - d. **lib** - non-webserver code - state spans web requests

- i. lib/pchat/endpoint.ex - requests routed here first, then passed to router
  - ii. lib/pchat/repo.ex - connects the phoenix app to our database via Ecto
  - iii. lib/pchat.ex - top level supervisor
- e. node\_modules - node.js build - for compiling static assets
- f. priv - "private" files
  - i. priv/static - static assets
- g. test - test code
- h. web - webserver code - state exists only for the duration of a web request
  - i. channels - websocket
  - ii. controllers
  - iii. models
  - iv. static - builds into priv/static
  - v. templates
  - vi. views
  - vii. router.ex
- 8. route.ex
  - a. module names start with Caps
  - b. dots create a namespace since Module names must be unique
  - c. "use" incorporates definitions from other modules
    - i. implicitly adds imports, macros, etc.
  - d. pipelines let us chain together actions to take on requests, scope identifies the routing table
    - i. atoms
    - ii. do ... end
    - iii. strings
    - iv. macros create a DSL
    - v. GET on root calls PageController.index
    - vi. DSL - based on macros
- 9. controllers/page\_controller
  - a. defines index/2, corresponding to the route
  - b. function names and variables start with lower case
  - c. \_params - means it's ok that variable is not used
  - d. renders "index.html" - Phoenix finds templates/page/index.html.eex
    - i. .eex is embedded elixir - templating engine - compiled into functions, not interpreted at runtime.
  - e. function calls do not require parens
    - i. demonstrate shell
      - 1. call functions
      - 2. line editing
      - 3. variables
        - a. explain data is immutable although variables can be rebound
- 10. Build a chat where you communicate via pictures

- a. websocket browser clients for live updates
  - b. post with name and picture URL
- 11. setup client
  - a. **cp ../../REF/pchat-static-assets/web .**
    - i. copy new templates into web/templates for pchat from web/templates
    - ii. copy new static assets into web/static for pchat from web/static
  - b. notice phoenix notices updated files
- 12. Channel
  - a. Create a room - **mix phoenix.gen.channel Room room**
  - b. add channel room to **user\_socket**
    - i. uncomment channel line
  - c. Pchat.RoomChannel module (web/channels/room\_channel.ex)
  - d. customize to our needs
    - i. talk about tuples, maps
    - ii. remove authorization
    - iii. handle\_in new:msg
    - iv. remove rest
  - e. post pictures:
    - i. <http://res.freestockphotos.biz/pictures/17/17885-cat-close-up-pv.jpg>
    - ii. [http://fc09.deviantart.net/fs70/i/2012/148/7/8/tabby\\_cat\\_1\\_by\\_lakela-d51d61e.jpg](http://fc09.deviantart.net/fs70/i/2012/148/7/8/tabby_cat_1_by_lakela-d51d61e.jpg)
    - iii. <http://4.bp.blogspot.com/-XPeYMMrDxfw/UA2dQHIXeVI/AAAAAAAAAKE/XXuEUEUlldY/s1600/Dog-2.jpg>
    - iv. <http://www.parakeetcare.org/parakeet-pictures/green-parakeets-beak.jpg>
- 13. REST interface to inject message - need a way to process the REST request and then send the user/img to all of the users connected to the room. Keep track of all the clients and then send a message to all of them. We'll use a process to render all of the chats.
  - a. web/router.ex - uncomment scope /api
  - b. add **resources "/post", PostController**
    - i. try in browser, get an error: **http://localhost:4000/post**
  - c. **mix phoenix.routes**
  - d. copy controller/page\_controller.ex to **post\_controller.ex**
    - i. change module name
    - ii. function is "post"
    - iii. decode params as map, keys are strings "user" and "img", use =>
    - iv. status = Pchat.PostHandler.post(user, img)
    - v. json conn, %{"status": status}
    - vi. [http://localhost:4000/api/post?user=marc&img=https://s2.graphiq.com/sites/default/files/stories/t2/tiny\\_cat\\_12573\\_8950.jpg](http://localhost:4000/api/post?user=marc&img=https://s2.graphiq.com/sites/default/files/stories/t2/tiny_cat_12573_8950.jpg)
    - vii. explain pattern matching
      - 1. demo on shell
- 14. GenServer
  - a. lib/pchat/post\_handler.ex

- i. API
    - 1. channel registers its pid
    - 2. controller posts img url
  - b. framework for writing processes that handle requests
    - i. has an event loop
    - ii. single thread of execution
    - iii. programmed via callbacks
  - c. start\_link - starts the process that will be our server
    - i. talk about processes, process model
      - 1. processes are light weight:
        - a. for \_ <- 1..100000, do: spawn(fn() -> :timer.sleep(1000) end)
      - 2. actor model
        - a. send pid, message
        - b. receive do msg -> msg after 0 -> :ok end
      - 3. linking is how processes keep track of one another
  - d. init returns the initial loop state that is passed into the callbacks
  - e. cast/handle\_cast for async calls - use it here to register the room channel so we can send it messages
    - i. Process.monitor lets us know if the process exits
    - ii. VM sends us a message we see as handle\_info callback
  - f. post/handle\_call
    - i. pattern matching in state to see if we should do something
    - ii. reply to caller, so call to post can return a value
      - 1. mention that API hides message passing
    - iii. Pchat.RoomChannel.post for registered pid
      - 1. anonymous functions
      - 2. higher order functions
  - g. add a post API to the channel to send message to channel pid using regular Elixir message, add corresponding handle\_info
  - h. add call to PChat.PostHandler.register in Channel's join
- 15. Supervisor: pchat.ex
  - a. add GenServer as worker: **worker(Pchat.PostHandler, [])**,
  - b. talk about supervision trees
    - i. strategies
    - ii. error handling
- 16. Need to restart because we changed lib directory
  - a. get a shell **ix -S mix phoenix.server**
  - b. :observer.start
  - c. find pchat app in applications
  - d. find PostHandler in pids
  - e. show process state
- 17. Add archive of messages

- a. **mix phoenix.gen.model Msg msgs who:string msg:string**
  - i. look at model
    - 1. changeset is a pipeline to validate the data
    - 2. automatically adds primary key
    - 3. timestamps are inserted\_at and updated\_at
  - ii. migration
    - 1. creates the table, also used for migrating to new versions
  - iii. **mix ecto.migrate**
  - iv. look at postgres
    - 1. psql postgres
    - 2. \l to list databases
    - 3. \c pchat\_dev to connect to the database
    - 4. \dt to list tables
    - 5. \d msgs to see msgs table definition
- b. insert messages
  - i. add code to Pchat.RoomChannel.handle\_in to create a changeset and Repo.insert
- c. Reload Picture Chat page - causes recompile
- d. Post message
- e. See log of INSERT
- f. In psql, **select \* from msgs;**

## room\_channel.ex

```
defmodule Pchat.RoomChannel do
  use Pchat.Web, :channel

  def join("room:lobby", _payload, socket) do
    {:ok, socket}
  end

  def handle_in("new:msg", payload, socket) do
    broadcast! socket, "new:msg", %{user: payload["user"], body: payload["body"]}
    {:noreply, socket}
  end
end
```

## router.ex

Add for API route

```
# Other scopes may use custom stacks.
```

```
scope "/api", Pchat do
```

```
  pipe_through :api
```

```
  get "/post", PostController, :post
```

```
end
```

## room\_channel.ex - for REST

```
defmodule Pchat.RoomChannel do
  use Pchat.Web, :channel

  def post(pid, user, img) do
    send(pid, {:post, user, img})
  end

  def join("rooms:lobby", _msg, socket) do
    Pchat.PostHandler.register(self)
    {:ok, socket}
  end

  def handle_in("new:msg", msg, socket) do
    broadcast! socket, "new:msg", %{user: msg["user"], body: msg["body"]}
    {:reply, {:ok, %{msg: msg["body"]}}, socket}
  end

  def handle_info({:post, user, img}, socket) do
    push socket, "new:msg", %{user: user, body: img}
    {:noreply, socket}
  end

end
```



## room\_channel.ex - add message archive

```
def handle_in("new:msg", payload, socket) do
  user = payload["user"]
  msg = payload["body"]
  changeset = Pchat.Msg.changeset(%Pchat.Msg{}, %{who: user, msg: msg})
  {:ok, _} = Repo.insert(changeset)
  broadcast! socket, "new:msg", %{user: user, body: msg}
  {:noreply, socket}
end
```

## post\_controller.ex

```
defmodule Pchat.PostController do
  use Pchat.Web, :controller

  def index(conn, %{ "user" => user, "img" => img }) do
    status = Pchat.PostHandler.post(user, img)
    json conn, %{ "status": status }
  end
end
```

## lib/pchat/post\_handler.ex

```
defmodule Pchat.PostHandler do
  use GenServer

  def start_link do
    GenServer.start_link(__MODULE__, [], name: __MODULE__)
  end

  def init([]) do
    {:ok, []}
  end

  def register(pid) do
    GenServer.cast(__MODULE__, {:register, pid})
  end

  def handle_cast({:register, pid}, pids) do
    Process.monitor pid
    {:noreply, [pid | pids]}
  end

  def post(user, img_url) do
    GenServer.call(__MODULE__, {:post, user, img_url})
  end

  def handle_call({:post, _user, _img_url}, _from, []) do
    {:reply, "no_room", []}
  end

  def handle_call({:post, user, img_url}, _from, pids) do
    postfn = fn pid -> Pchat.RoomChannel.post(pid, user, img_url) end
    Enum.each(pids, postfn)
    {:reply, "ok", pids}
  end

  def handle_info({:DOWN, _ref, :process, pid, _}, pids) do
    pids = Enum.filter(pids, fn p -> p != pid end)
    {:noreply, pids}
  end
end
```

Start Elixir Shell from job control:  
s 'Elixir.IEx'

Erlang shell functions:  
:c.i