PHOENIX CHANNELS & PRESENCE

KC ELIXIR MEETUP

PHOENIX CHANNELS

- Messaging system that provides soft-realtime features to Phoenix applications (or components :-)).
- Senders broadcast messages on topics. Receivers subscribe to topics to receive messages.
- Senders and Receivers *do not have to be Elixir processes*. Instead, they can be anything (Javascript client, iOS app, etc.) that can be taught to communicate through a Channel.

BENEFITS

- User demand for real time updates and communication
- Subscription-based
- Super scalable [1] [2]
- Integration between disparate platforms
- Stateful connection requires less overhead with each call

[1] https://dockyard.com/blog/2016/08/09/phoenix-channels-vs-rails-action-cable

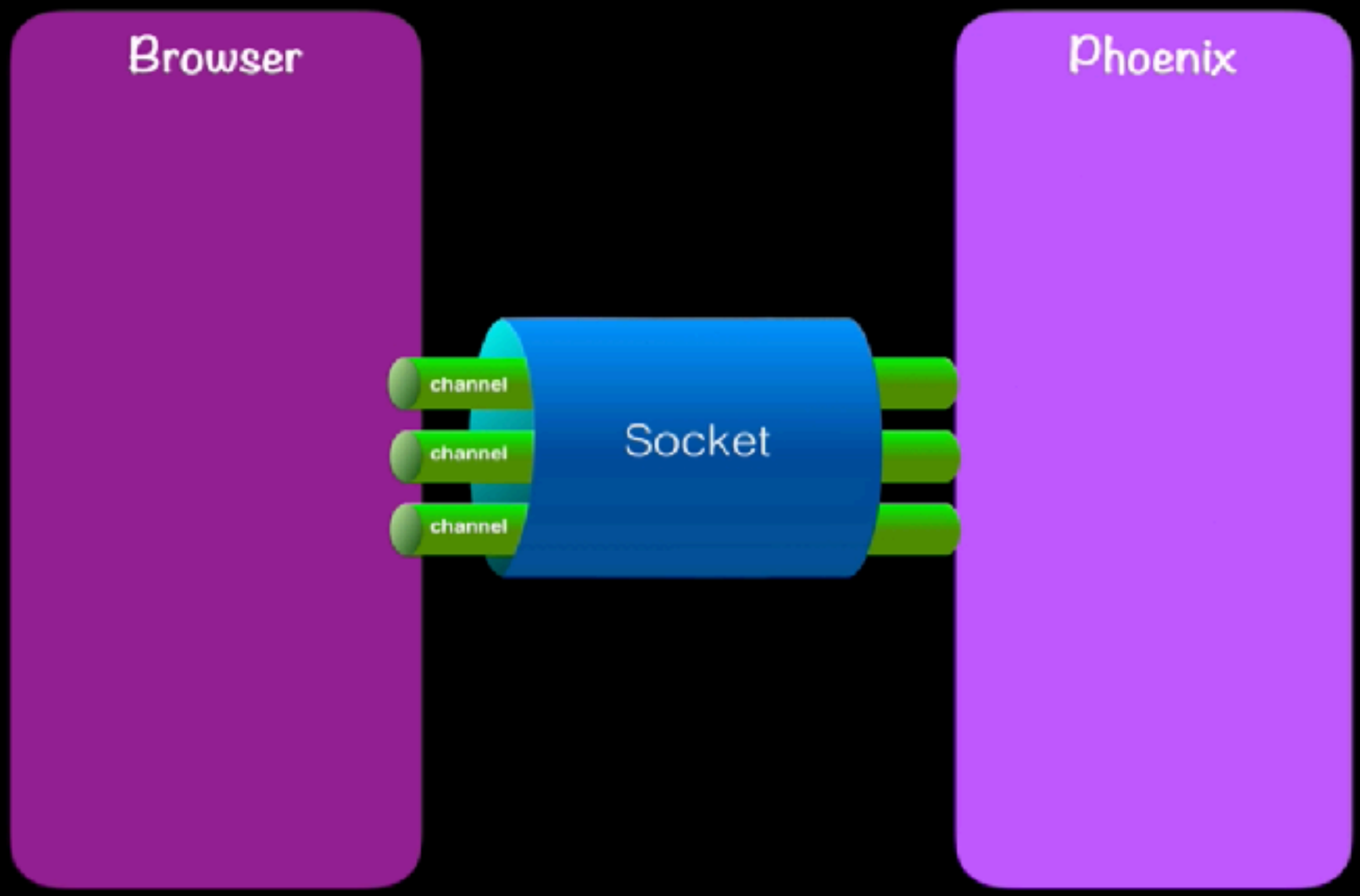
[2] http://phoenixframework.org/blog/the-road-to-2-million-websocket-connections

AWESOME, I WANT TO CREATE A CHANNEL. WHAT DO I DO?

FIRST, START WITH A SOCKET

SOCKETS

- A User needs to connect to a Socket before a Channel can be joined.
- Channels depend on a Socket for communication.
- Sockets support simultaneous connections to multiple Channels.
- Sockets are configured in the user_socket.ex file which is generated by default in Phoenix projects.
- Authentication takes place in the Socket.
- Socket has an ‘assigns’ for persisting state.



https://codestool.coding-gnome.com/courses/take/elixir-for-programmers

SOCKETS - USER_SOCKET.EX

```elixir
defmodule TrainingCenterWeb.UserSocket do
  use Phoenix.Socket

  ## Channels
  channel "training_session:*", TrainingCenterWeb.GymChannel

  ## Transports
  transport :websocket, Phoenix.Transports.WebSocket
  # transport :longpoll, Phoenix.Transports.LongPoll

  def connect(%{"name" => name, "role" => role}, socket) do
    {:ok, assign(socket, :current_user, %{name: name, role: role})}
  end
```

training_center/lib/training_center_web/channels/user_socket.ex

SWEET, HOW DO I CONNECT TO A SOCKET FROM THE BROWSER?

PHOENIX'S SOCKET JAVASCRIPT LIBRARY

- Bundled with Phoenix
- Can edit included socket.js file or create new file and include it in app.js.

```
import {Socket} from "phoenix"

let socket = new Socket("/socket", {params: {token: window.userToken}})
socket.connect()
```

training_center/assets/js/gym_socket.js

EASY PEASY

ALRIGHT, SO THE SOCKET MADE LIKE STEREO MCS[1] AND GOT ITSELF CONNECTED. WHAT'S NEXT?

[1] https://www.youtube.com/watch?v=2S_gitKd7U4

CHANNEL. TIME.

LET'S TALK THROUGH THE PHOENIX.CHANNEL MODULE, STARTING WITH CALLBACKS AND THEN FUNCTIONS.

PHOENIX.CHANNEL – CALLBACKS

- We'll focus on these:
  - join(topic, auth_msg, socket)
  - handle_info(term, socket)
  - handle_in(event, msg, socket)

training_center/lib/training_center_web/channels/gym_channel.ex

PHOENIX.CHANNEL – CALLBACKS – JOIN/3 & HANDLE_INFO/2

- Phoenix.Channel.join/3 is the entry point to a Channel.
  - Channel authentication/authorization can be handled here
  - Join a Channel on a specific topic
  - Can subscribe to multiple topics on the same Channel

```
## Channels
channel "training_session:*", TrainingCenterWeb.GymChannel
```

training_center/lib/training_center_web/channels/user_socket.ex

- Phoenix.Channel.handle_info/2 accepts direct messages, just like GenServer.handle_info/2.

PHOENIX.CHANNEL - CALLBACKS - JOIN/3 & HANDLE_INFO/2

- Common to send an :after_join message on join/3 and handle data retrieval from the handle_info/2 function.

```elixir
def join("training_session:" <> training_session_id, _, socket) do
  send(self(), :after_join)
  {:ok, assign(socket, :training_session_id, training_session_id)}
end

def handle_info(:after_join, socket) do
  case role(socket) do
    "participant" ->
      push_participant_status(socket)
    "trainer" ->
      push_status(socket)
  end

  {:noreply, socket}
end
```

training_center/lib/training_center_web/channels/gym_channel.ex

PHOENIX.CHANNEL - CALLBACKS - HANDLE_IN/3

- handle_in/3 matches on an event sent into the Channel and then processes that event appropriately.

```elixir
def handle_in("complete_exercise", %{"name" => name}, socket) do
  status = Gym.complete_exercise(training_session_id(socket), current_user(socket).name, name)
  push(socket, "participant_status", %{html: TrainingSessionView.get_html(status)})
  broadcast!(socket, "status_updated", %{})
  {:noreply, socket}
end
```

training_center/lib/training_center_web/channels/gym_channel.ex

SO THOSE ARE THE CALLBACKS. THERE IS ACTUALLY A SPECIAL ONE WE'LL REVEAL SHORTLY. UNTIL THEN, HERE ARE SOME VERY USEFUL CHANNEL FUNCTIONS...

PHOENIX.CHANNEL – FUNCTIONS – PUSH/3

- push(socket, event, message)
- push/3 sends a message directly to one socket.
- This works well if you are wanting to return requested information to the Channel subscriber who requested it.

```
def handle_in("complete_exercise", %{"name" => name}, socket) do
  status = Gym.complete_exercise(training_session_id(socket), current_user(socket).name, name)
  push(socket, "participant_status", %{html: TrainingSessionView.get_html(status)})
  {:noreply, socket}
end
```

training_center/lib/training_center_web/channels/gym_channel.ex

PHOENIX.CHANNEL – FUNCTIONS – BROADCAST

- broadcast(socket, event, message)
  - Broadcasts an event to all Channel topic subscribers
- broadcast_from(socket, event, message)
  - Broadcasts an event to all Channel topic subscribers EXCEPT the one that originated the request
- Both have ! option that raises exception if broadcast fails

```elixir
def handle_in("complete_exercise", %{"name" => name}, socket) do
  broadcast!(socket, "status_updated", %{})
  {:noreply, socket}
end
```

training_center/lib/training_center_web/channels/gym_channel.ex

WHAT IF I WANT TO BROADCAST TO ONLY SOME CHANNEL SUBSCRIBERS?

HELLO INTERCEPTS!

INTERCEPTS

- Intercepts are defined in the Channel module.
- A special callback, handle_out/3, is invoked for the messages listed in Intercepts.
- Logic in this callback can suppress a broadcast message from going to specific sockets.

```
intercept [
  "status_updated"
]

def handle_out("status_updated", _msg, socket) do
  if is_trainer?(socket), do: push_status(socket)
  {:noreply, socket}
end
```

training_center/lib/training_center_web/channels/gym_channel.ex

COOL. SO I HAVE TO BE SUBSCRIBED TO A CHANNEL TO BROADCAST TO IT, RIGHT?

NOPE!

Chuck Testa™

BROADCAST FROM OUTSIDE CHANNEL

- The Phoenix Endpoint can be used to broadcast to a Channel topic.
- The code below is in a Phoenix Controller.

```elixir
def start_training_session(conn, params = %{"id" => id}) do
  Gym.start_training_session(id)
  Endpoint.broadcast("training_session:#{id}", "training_session_started", %{})
  redirect(conn, to: trainer_training_session_path(conn, :show, id))
end
```

training_center/lib/training_center_web/controllers/trainer/training_session_controller.ex

ENOUGH WITH THE SERVER, WHAT'S UP WITH THE BROWSER? HOW DOES IT GET IN ON THIS SWEET, SWEET NEW CHANNEL HOTNESS?

PHOENIX CHANNELS – JAVASCRIPT

- The Javascript Socket module provides a Channel class.
- The Channel class allows you to join a Channel as well as send and receive messages.

PHOENIX CHANNELS – JAVASCRIPT

- The join() method simply joins a Channel on a topic.

```
channel = socket.channel("training_session:1", {})
channel.join()
        .receive("ok", resp => {
          console.log("connected: " + resp)
        })
        .receive("error", resp => {
          alert(resp)
          throw(resp)
        })
```

training_center/assets/js/gym_socket.js

PHOENIX CHANNELS – JAVASCRIPT

- The push(msg, object) method sends a message to the Channel.

```
channel.push("complete_exercise", {name: name})
```

training_center/assets/js/gym_socket.js

PHOENIX CHANNELS – JAVASCRIPT

- The on(msg, function) method is executed upon receiving a matching message from the Channel.

```
channel.on("status_updated", payload => {
  console.log("Received status updated message")
  let trainingSessionContainer = document.getElementById("trainingSessionContainer")
  trainingSessionContainer.innerHTML = payload.html
})
```

training_center/assets/js/gym_socket.js

WHEW! THAT WAS PLENTY. LET'S TAKE A QUICK SPIN THROUGH PRESENCE-LAND AND CALL IT A DAY, HUH?

PHOENIX PRESENCE

- Provides tracking of Processes and Channels.
- Allows for discovery of which people, systems, etc. are currently connected and “online”.

BENEFITS

- No single point of failure
- No single source of truth
- Automatically replicates across cluster
- SO simple to implement

https://dockyard.com/blog/2016/03/25/what-makes-phoenix-presence-special-sneak-peek

HOW DOES IT WORK?

PRESENCE – SERVER SIDE

- Presence manages its state through two functions
  - Presence.track/3
  - Presence.list/1

PRESENCE.TRACK/3

- When a user join a Channel, Presence.track/3 can be called.
- This allows Presence to monitor your connection status to that Channel.
- Can add lightweight metadata for the third argument.

```
Presence.track(socket, current_user(socket).name, %{
  online_at: inspect(System.system_time(:seconds))
})
```

training_center/lib/training_center_web/channels/gym_channel.ex

PRESENCE.LIST/1

- Presence.list/1 will provide a list of everyone currently connected to the Channel. It takes the Channel Socket as a parameter.

```
Presence.list(socket)
```

training_center/lib/training_center_web/channels/gym_channel.ex

PRESENCE_DIFF

- Presence Diff is managed behind the scenes.
- Anytime Presence detects someone joining or leaving a Channel, it will broadcast a “presence_diff” message to all connected subscribers.
- The “presence_diff” message contains only information about what has changed.
- You can manage who receives these notifications by adding an intercept and handle_out(“presence_diff”, ...) callback in the Channel.

ALRIGHT, SO HOW DO WE SET THIS UP?

PRESENCE MODULE

- You'll need to define a Presence module in the Channels directory.

```
defmodule TrainingCenterWeb.Presence do
  use Phoenix.Presence,
    otp_app: :training_center,
    pubsub_server: TrainingCenter.PubSub
end
```

training_center/lib/training_center_web/channels/presence.ex

ADD TO APPLICATION SUPERVISION

- We'll update the Supervision children list in application.ex to include the Presence module we just defined.

```
children = [
  supervisor(TrainingCenterWeb.Endpoint, []),
  supervisor(TrainingCenterWeb.Presence, [])
]
```

training_center/lib/training_center/application.ex

PHOENIX PRESENCE – JAVASCRIPT

- The Javascript for Presence is pretty straight forward.
- import {Presence} from “phoenix”
- Create an empty presences object.
- Set up two channel.on() callbacks.
  - channel.on(“presence_state”, ...)
  - channel.on(“presence_diff”, ...)

training_center/assets/js/gym_socket.js

PHOENIX PRESENCE – JAVASCRIPT – SYNCDIFF

- Listens for changes to Phoenix.Tracker
- “Tracker servers use a heartbeat protocol and CRDT [1] to replicate presence information across a cluster in an eventually consistent, conflict-free manner.” [2]

```
channel.on("presence_diff", diff => {
  presences = Presence.syncDiff(presences, diff)
})
```

training_center/assets/js/gym_socket.js

[1] https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type

[2] https://hexdocs.pm/phoenix_pubsub/Phoenix.Tracker.html

PHOENIX PRESENCE - JAVASCRIPT - SYNCSTATE

```
channel.on("presence_state", state => {
  presences = Presence.syncState(presences, state)
})
```

training_center/assets/js/gym_socket.js

THAT'S IT!

LET'S LOOK AT A QUICK DEMONSTRATION.

DEMONSTRATION

- https://samdev.us

CODE

- https://github.com/sammarten/kc-elixir-meetup-june-2018