Live Qchatex

Web chat application that uses **sockets** to entirely handle client-server interactions.

Phoenix LiveView: Handles UI interactions with client's browser, such as HTML content updates, click events, forms submission, etc. It's a 2-way communication channel (websockets) so no need for polling.

Phoenix PubSub: Notifies when certain events occurred by sending messages to processes. Each process decides what "kind" of messages will "listen" by subscribing to certain topics.

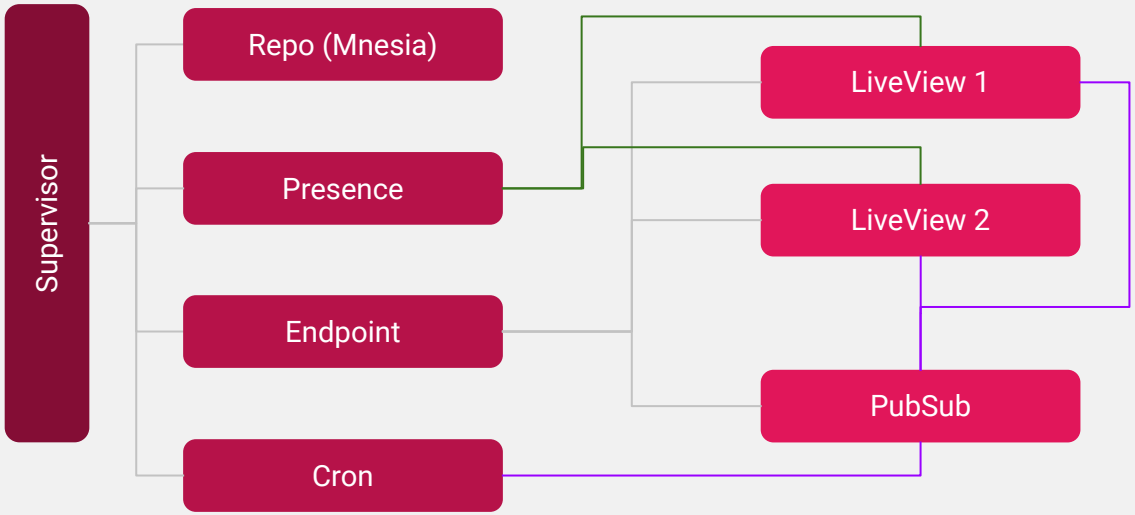
Phoenix Presence: Notifies when processes (sockets) joins/leaves a certain topic. As soon as a subscribed process (socket) ends or dies, a message is broadcasted to the other subscribed processes.

Erlang Mnesia + Memento: Temporarily stores data (such as users sessions, chat rooms and messages) in a way that it doesn't get lost if the server is restarted. Uses Erlang DETS behind scenes.

FIQUS
COOPERATIVA DE SOFTWARE

Supervision Tree

How the app is organised into distinct processes.

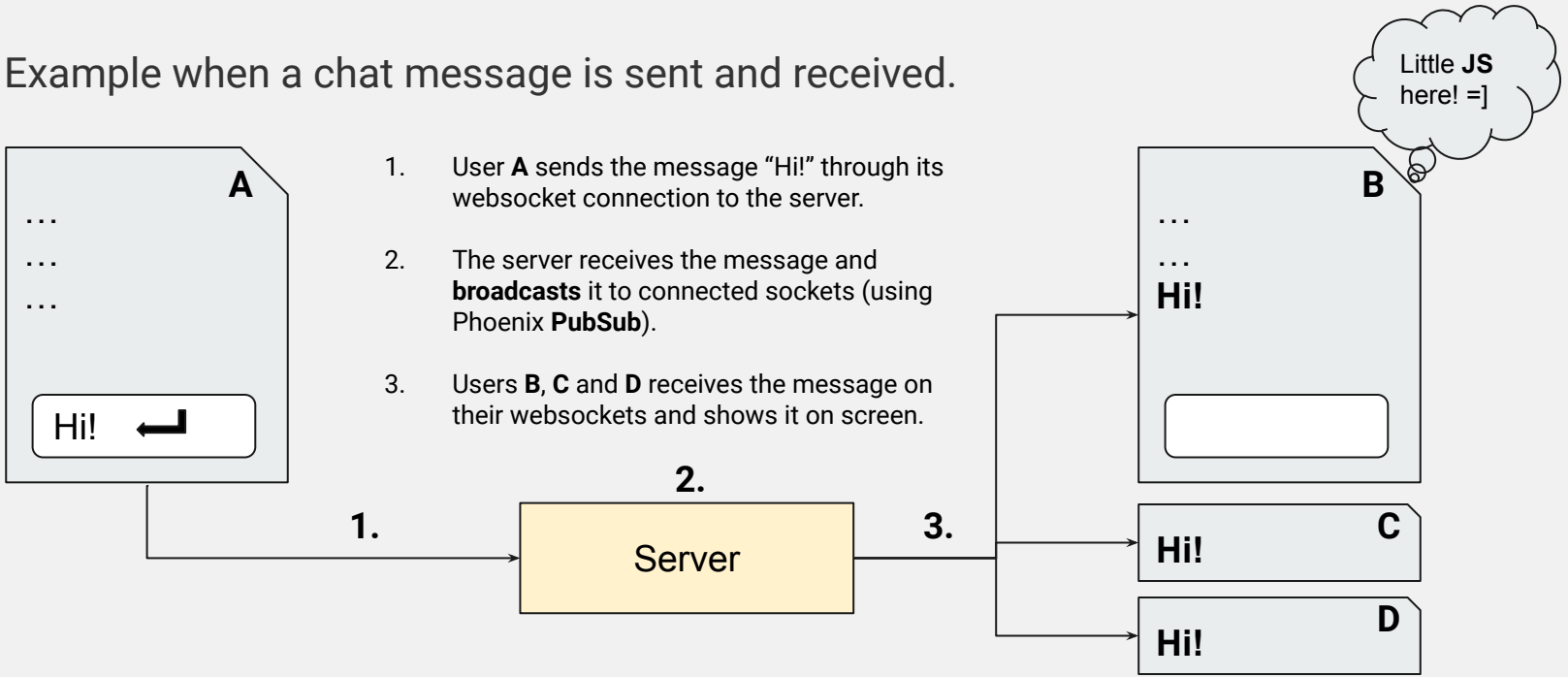


Subscriptions and links:

- Track messages
- Broadcast messages
- Process link

Phoenix LiveView

Example when a chat message is sent and received.



1. User **A** sends the message “Hi!” through its websocket connection to the server.
2. The server receives the message and **broadcasts** it to connected sockets (using Phoenix **PubSub**).
3. Users **B**, **C** and **D** receives the message on their websockets and shows it on screen.

Phoenix PubSub

Managing the communications between all the LiveView processes.

It is the **core engine** that allows us to **interact with LiveViews** by providing a way to **push real-time changes** to clients browsers about events happened in **another** process (that can be a other LiveView or **any** process).

- Each client/websocket/live-view/process is **subscribed** to certain internal **topics** on the **server** and will **react** when events/messages arrives to the **process's mailbox**.

- Then the **live-view** process/socket could **send to the client** an update to be displayed on its browser screen, such as online chats/users counters, public chats list, chat members and messages… basically everything!

That makes Phoenix **LiveView + PubSub** a perfect fit for a real-time chat app!
But what's about a client leaving a chat or even closing the browser....?
Phoenix Presence to the rescue!

Phoenix LiveView + PubSub

Example when a chat message is sent and received.

User **A** (sender)

```
PubSub.subscribe(LiveQchatex.PubSub, "chats/#{chat_id}")
```

```
def handle_event("newmsg", %{"msg" => msg}, socket) do
   PubSub.broadcast_from(LiveQchatex.PubSub, self(),
       "chats/#{chat_id}", {:chat_msg, msg})
   {:noreply, socket |> update_messages(msg)}
 end
```

User **B** (receiver)

```
PubSub.subscribe(LiveQchatex.PubSub, "chats/#{chat_id}")
```

```
def handle_info({:chat_msg, msg}, socket) do
   {:noreply, socket |> update_messages(msg)}
end

defp update_messages(%{:assigns => assigns} = socket, msg) do
   socket |> assign(:msgs, assigns.msgs ++ [msg])
end
```

By updating **socket.assigns** we “tell” LiveView to **automatically** re-render **just** the portion of the changed content (its **diff**).

Phoenix Presence

Detecting when processes joins or leaves certain topics.

It provides a way to **notify subscribed processes** that **another process** was connected or disconnected to a given topic. It can also notify **updates** when a process changes its “presence information”.

- Same as **PubSub**, each client/websocket/live-view/process is **subscribed** to certain internal **topics** on the **server** and will **react** when events/messages arrives to the **process’s mailbox**.
- Then, same as **PubSub**, the **live-view** process/socket could **send to the client** an update to be displayed on its browser screen, such as member join/quit a chat room, member typing status, nickname changes, etc.

When an user **leaves** a chat room or **closes** its browser, the underlying socket/process is **destroyed** and Presence will **notify** the subscribers that this **PID** is no longer amongst the living!

FIQUS
COOPERATIVA DE SOFTWARE

Phoenix Presence

Example when an user leaves the chat room.

User **A** (sender)

```
PubSub.subscribe(LiveQchatex.PubSub, "chats/#{chat_id}")
Presence.track(self(), "chats/#{chat_id}", user.id, user)
```

User **B** (receiver)

```
PubSub.subscribe(LiveQchatex.PubSub, "chats/#{chat_id}")
Presence.track(self(), "chats/#{chat_id}", user.id, user)
```

The user **closes** its browser, so this **PID** dies!

```
def handle_info(%{event: "presence_diff", topic: topic, payload: _p}, socket) do
    {:noreply, socket |> assign(members: Presence.list(topic))}
end
```

By updating **socket.assigns** we “tell” LiveView to **automatically** re-render **just** the portion of the changed content (its **diff**).

Erlang Mnesia + Memento

An out-of-the-box distributed database engine running on the BEAM.

It uses **ETS** or “Erlang Term Storage” to manage **ram based key-value** storage objects, so it’s **super fast**.
The chat app uses **D**ETS since it needs to **persist the data to disk** in order to keep it between server restarts.

- You can **store any valid struct** as it is in its native representation, no conversions or mutations needed.
- You can have **multiple nodes** running Mnesia and easily connect them together to **distribute** the load.

The chat app uses **Memento** as a **wrapper** to abstract Mnesia usage from **Elixir**.
It provides a friendly interface to run **CRUD** operations in an easy and simple way.

The app only needs to define 3 “model” structs: **User, Chat and Message**.

App Features

Some interesting features supported by the app.

- Chats and its messages are **kept** for a certain period of time, even if the chat room is empty.
- A **cron** process (GenServer) that periodically **clears** expired users, chats and messages.
- A "**heartbeat**" is sent from idle clients in order to refresh users and chats expiration times.
- Chat rooms can be **private or public** and can be changed on-the-fly (only by chat room owner/creator).
- Public chats are **listed** at home page, dynamically updating its titles and members amount.
- Support for update chat **title** (only by chat room owner/creator).
- Support for update user **nickname** by clicking yourself on members list or sending a message: `/nick MyName`
- The chat member list displays 3 dots "..." while an user is **typing** (with a *3s* timeout).
- Clicking in a chat member inside a chat room, will issue a **private chat request**.
- Private chat **invites** will be displayed on screen and will open a new chat window if accepted.
- A private chat between 2 users is **just another** private chat room like any other, so ir re-uses the same logic.

FIQUS
COOPERATIVA DE SOFTWARE

Thanks! =]

Some useful links for more info about this wonderful technologies!

App Repo: https://github.com/fiqus/lqchatex
Live Demo: https://lqchatex.fiqus.coop/

Phoenix LiveView: https://github.com/phoenixframework/phoenix_live_view
Phoenix PubSub: https://hexdocs.pm/phoenix_pubsub/
Phoenix Presence: https://hexdocs.pm/phoenix/Phoenix.Presence.html

Erlang Mnesia: https://learnyousomeerlang.com/mnesia
Memento: https://github.com/sheharyarn/memento

Visit us at **https://fiqus.coop**