Visão detalhada de Channels: Criando uma aplicação Chat

Roteiro

- Definição
- Controllers, Views e Templates
- Channels
- Websocket x Longpoll

Definição

Forma de comunicação a distância, utilizando computadores ligados à internet, na qual o que se digita no teclado de um deles aparece em tempo real no vídeo de todos os participantes do bate-papo.

Definição

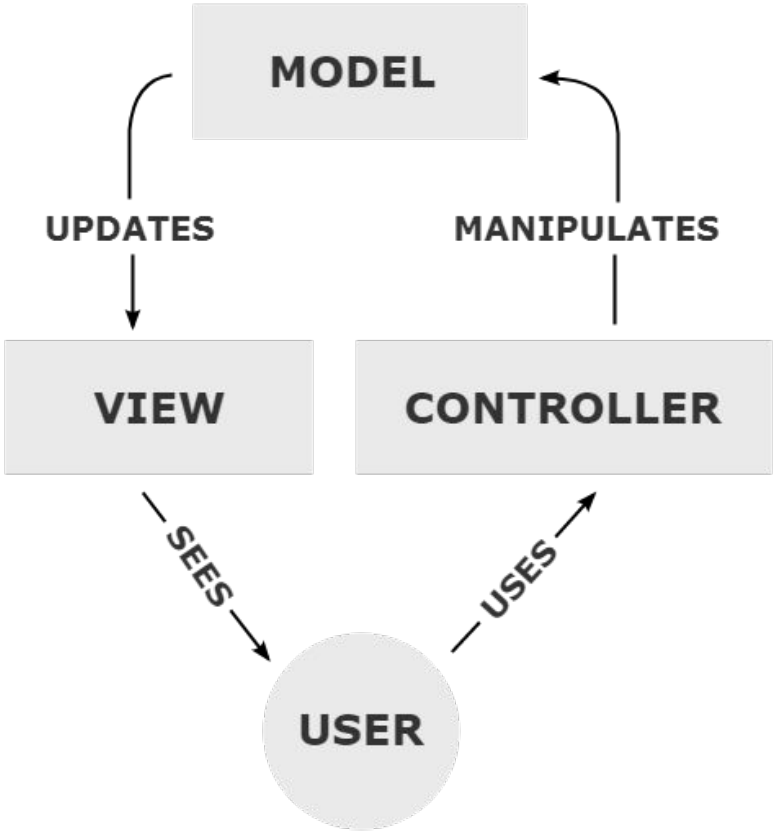
O que o nosso chat estará apto a fazer:

- Criar usuário e salvando no banco
- Mandar mensagem (porém sem persistência)
- Presença de online

COMANDOS

```
mix phx.new chat

{:plug_cowboy, “~> 1.0”}

mix deps.get

mix ecto.create

mix phx.server
```

Controllers, Views e Templates



COMANDOS

mix phx.gen.html Conversation Room rooms nome descricao topico

resources, “/rooms”, RoomController

mix ecto.migrate

COMANDOS

assets/css/app.css:

```
body {
  min-height: 2000px;
  padding-top: 70px;
}
```

comentar no bootstrap:

```
/* Customize container */
@media (min-width: 768px) {
  .container {
    max-width: 730px;
  }
}
```

Remover a div Header em lib/chat_web/templates/layout/app.html.eex

COMANDOS

lib/chat_web/templates/room/index.html.eex

```
<div class="row">
  <div class="col-md-3">
    <h3>Rooms</h3>
    <ul class="list-group">
      <li class="list-group-item">Ufra</li>
    </ul>
  </div>
  <div class="col-md-9">
    <div class="jumbotron">
      <div class="page-header">
        <h2>Welcome to Chat</h2>
      </div>
      <div class="page-header">
        <h2><small>Escolha uma sala para entrar</small></h2>
        <h2><small>ou <a href="/rooms/new" class="btn btn-success">Crie</a> uma nova</small></h2>
      </div>
    </div>
  </div>
</div>
```

COMANDOS

lib/chat_web/templates/room/index.html.eex

```
<div class="row">
  <div class="col-md-3">
    <h3>Rooms</h3>
    <ul class="list-group">
1°        - <li class="list-group-item">Ufra</li>
1°        + <%= inspect @rooms %>


2°        - <%= inspect @rooms %>

2°        +<%= for room <- @rooms do %>
2°        +    <%= link room.nome, to: room_path(@conn, :show, room.id) %>
2°        +<% end %>
```

COMANDOS

lib/chat_web/templates/layout/app.html.eex

```
<head>

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW
/dAiS6JXm" crossorigin="anonymous">

<link rel="stylesheet" href="<%= static_path(@conn, "/css/app.css") %>">

</head>
```

COMANDOS

lib/chat_web/templates/layout/app.html.eex

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93
hXpG5KkN" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXu
svfa0b4Q" crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76
PVCmYl" crossorigin="anonymous"></script>

<script src="<%= static_path(@conn, "/js/app.js") %>"></script>
```

COMANDOS

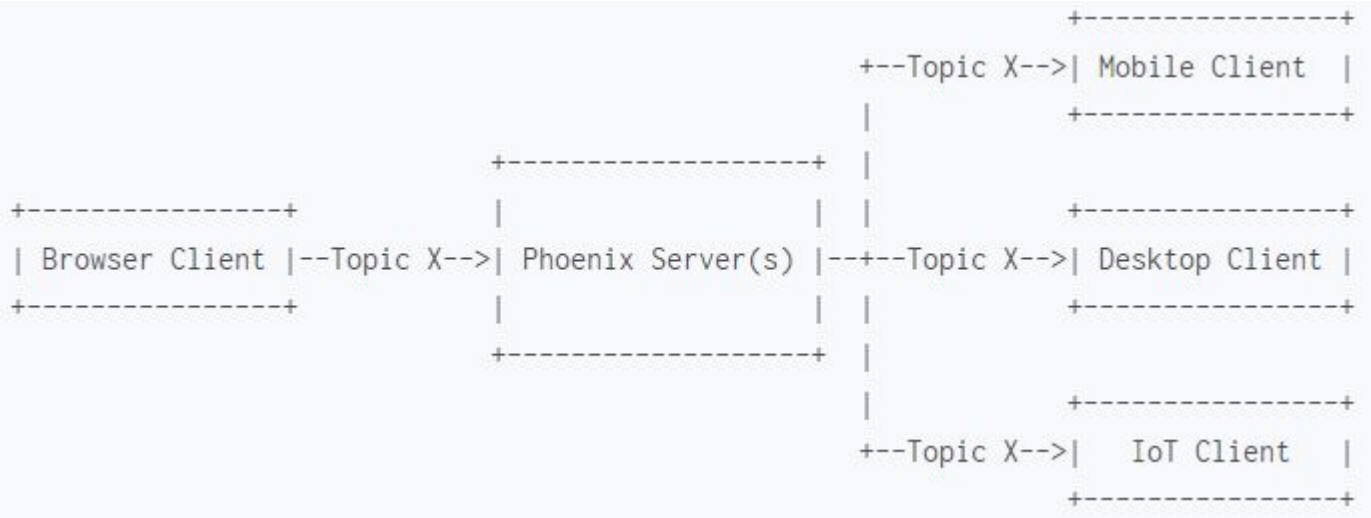
remover body app.css

lib/prater_web/templates/layout/app.html.eex - importar abaixo da div container

```
<div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3
bg-white border-bottom box-shadow">
  <h5 class="my-0 mr-md-auto font-weight-normal">
    <a href="/" class="navbar-brand text-dark"><strong>Chat</strong></a>
  </h5>
</div>
```

Channels

Conceitualmente, os canais são bem simples. Os clientes se conectam e se inscrevem em um ou mais tópicos, seja isso `chat_publico` ou `atualiza_post_do:user1`. Qualquer mensagem enviada em um tópico, seja do servidor ou de um cliente, é enviada para todos os clientes inscritos nesse tópico



Os canais podem suportar qualquer tipo de cliente: um navegador, aplicativo nativo, relógio inteligente, dispositivo incorporado ou qualquer outra coisa que possa se conectar a uma rede.

Channels

Rotas do Canal

Rotas de canal são definidas em manipuladores de soquete, como `HelloWeb.UserSocket` no exemplo acima. Eles correspondem na sequência de tópicos e despacham solicitações correspondentes para o módulo do Canal.

O caractere de estrela `*` atua como um correspondente de caractere curinga, portanto, na rota de exemplo a seguir, os pedidos `room:lobby` e `room:123` os dois serão despachados para o `RoomChannel`.

```
channel "room:*", HelloWeb.RoomChannel
```

Cada canal irá implementar uma ou mais cláusulas de cada uma destas quatro funções de retorno de chamada - `join/3`, `terminate/2`, `handle_in/3`, e `handle_out/3`.

Channels

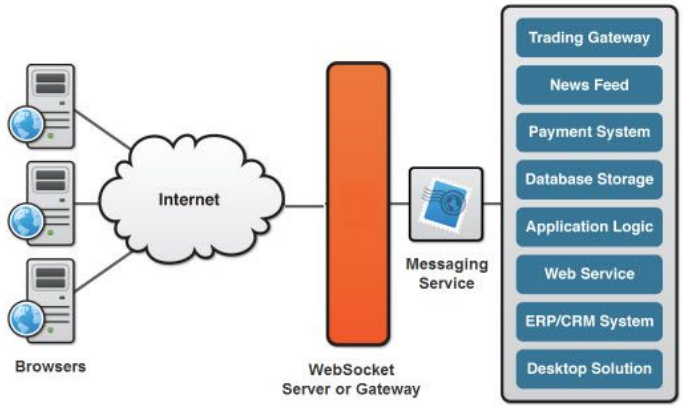
Mensagens

O Phoenix.Socket.Message módulo define uma estrutura com as seguintes chaves, que denota uma mensagem válida. Dos documentos Phoenix.Socket.Message .

- topic- O tópico da cadeia ou o "topic:subtopic"namespace de pares, como "messages"ou"messages:123"
- event - O nome do evento de string, por exemplo "phx_join"
- payload - A carga útil da mensagem
- ref - A referência única da string

Websocket

WebSocket é uma tecnologia que permite a comunicação bidirecional por canais full-duplex sobre um único soquete Transmission Control Protocol (TCP).



Longpoll

O Long Polling é uma técnica que simula uma indisponibilidade do servidor para manter uma conexão HTTP aberta. Essa técnica foi criada a partir da necessidade de comunicação em tempo real com um servidor *web*.

Quando um cliente faz uma requisição o servidor simula uma indisponibilidade de dados e faz com que a requisição HTTP não tenha resposta enquanto não houver mudanças no modelo dos dados da aplicação.

COMANDOS

mix phx.gen.schema Auth.User users email:unique username:unique encrypted_password

null: false

```
# lib/chat_web/router.ex
scope "/", ChatWeb do
  resources "/sessions", SessionController, only: [:new, :create]
end
```

COMANDOS

touch lib/chat_web/controllers/session_controller.ex

touch lib/chat_web/views/session_view.ex

touch lib/chat_web/templates/session/new.html.eex

lib/chat_web/controllers/session_controller.ex

```
defmodule ChatWeb.SessionController do
  use ChatWeb, :controller
   def new(conn, _params) do
    render conn, "new.html"
  end
end
```

lib/chat_web/views/session_view.ex

```
defmodule ChatWeb.SessionView do
 use ChatWeb, :view
end
```

lib/chat_web/templates/session/new.html.eex

```
<h1>Login</h1>
```

Suporte ssas

→ cd assets
→ npm install sass-brunch --save-de
→ cd ..

mv assets/css/app.css assets/css/app.scss

```
 stylesheets: {
    joinTo: "css/app.css",
    order: {
      after: ["priv/static/css/app.scss"]
    }
  },
  templates: {
    joinTo: "js/app.js"
  }
},
```

lib/chat_web/templates/session/new.html.eex

```
<div class="auth-form-wrapper">
  <%= form_for @conn, session_path(@conn, :create), [as: :session, class: "form-signin"], fn f -> %>
    <div class="text-center mb-4">
      <h1 class="h3 mb-3 font-weight-normal">Sign In</h1>
    </div>

    <div class="form-label-group">
      <%= text_input f, :email, class: "form-control", placeholder: "Email address", required: true, autofocus: true %>
      <%= label f, :email, "Email address", class: "control-label" %>
    </div>

    <div class="form-label-group">
      <%= password_input f, :password, class: "form-control", placeholder: "Password", required: true %>
      <%= label f, :password, class: "control-label" %>
    </div>

    <%= submit "Sign in", class: "btn btn-lg btn-primary btn-block" %>
  <% end %>
</div>
```

touch assets/css/auth_form.scss

https://github.com/franknfjr/elixir-phoenix/blob/master/dia4/chat/assets/css/login_form.scss

ChatWeb.SessionController

```
defmodule ChatWeb.SessionController do
  use ChatWeb, :controller
  alias Chat.Auth

  def new(conn, _params) do
    render conn, "new.html"
  end

  def create(conn, %{"session" => %{"email" => email, "password" => password}}) do
    case Auth.sign_in(email, password) do
      {:ok, user} ->
        conn
        |> put_session(:current_user_id, user.id)
        |> put_flash(:info, "You have successfully signed in!")
        |> redirect(to: room_path(conn, :index))

      {:error, _reason} ->
        conn
        |> put_flash(:error, "Invalid Email or Password")
        |> render("new.html")
    end
  end

end
```

touch lib/chat/auth/auth.ex

```
defmodule Chat.Auth do
  alias Chat.Repo
  alias Chat.Auth.User
   def sign_in(email, password) do
    user = Repo.get_by(User, email: email)
     cond do
      user && user.encrypted_password == password ->
        {:ok, user}
      true ->
        {:error, :unauthorized}
    end
  end

  def current_user(conn) do
     user_id = Plug.Conn.get_session(conn, :current_user_id)
     if user_id, do: Repo.get(User, user_id)
    end

    def user_signed_in?(conn) do
     !!current_user(conn)
    end

 end
```

lib/chat_web/templates/layout/app.html.eex

```
<%= if Chat.Auth.user_signed_in?(@conn) do %>
  <nav class="my-2 my-md-0 mr-md-3">
    Signed in as: <strong><%= Chat.Auth.current_user(@conn).username %></strong>
  </nav>
 <% end %>
```

iex -S mix phx.server

```
iex> %Chat.Auth.User{} |>
  Chat.Auth.User.changeset(%{email: "user@user.com", encrypted_password:
"123123", username: "user"}) |>
  Chat.Repo.insert()
```

```
delete "/sign_out", SessionController, :delete # routes
```

Controller do Sessions

```
def delete(conn, _params) do
  conn
  |> Auth.sign_out()
  |> redirect(to: room_path(conn, :index))
end
```

auth.ex

```
def sign_out(conn) do
  Plug.Conn.configure_session(conn, drop: true)
end
```

add em baixo do nav ->

```
    <%= if Chat.Auth.user_signed_in?(@conn) do %>

      <nav class="my-2 my-md-0 mr-md-3">
        Signed in as: <strong><%= Chat.Auth.current_user(@conn).username %></strong>
      </nav>
      <%= link "Sign Out", to: session_path(@conn, :delete), method: :delete, class: "btn
btn-outline-primary" %>
    <% else %>
      <%= link "Sign In", to: session_path(@conn, :new), class: "btn btn-outline-primary" %>
    <% end %>
```

COMANDOS

resources "/registrations", RegistrationController, only: [:new, :create]

touch lib/chat_web/controllers/registration_controller.ex

touch lib/chat_web/views/registration_view.ex

touch lib/chat_web/templates/registration/new.html.eex

```
defmodule ChatWeb.RegistrationController do
  use ChatWeb, :controller
  alias Chat.Auth
  def new(conn, _params) do
    render conn, "new.html", changeset: conn
  end
  def create(conn, %{"registration" => registration_params}) do
    case Auth.register(registration_params) do
      {:ok, user} ->
        conn
        |> put_session(:current_user_id, user.id)
        |> put_flash(:info, "You have successfully signed up!")
        |> redirect(to: room_path(conn, :index))
      {:error, changeset} ->
        render(conn, "new.html", changeset: changeset)
    end
  end
end
```

lib/chat_web/views/registration_view.ex

```
defmodule ChatWeb.RegistrationView do
  use ChatWeb, :view
end
```

```
<div class="auth-form-wrapper">
 <%= form_for @changeset, registration_path(@conn, :create), [as: :registration, class: "form-signin"], fn f -> %>
  <div class="text-center mb-4">
   <h1 class="h3 mb-3 font-weight-normal">Sign Up</h1>
  </div>

  <div class="form-label-group">
   <%= text_input f, :email, class: "form-control", placeholder: "Email address", required: true, autofocus: true %>
   <%= label f, :email, "Email address", class: "control-label" %>
   <%= error_tag f, :email %>
  </div>

  <div class="form-label-group">
   <%= text_input f, :username, class: "form-control", placeholder: "User name", required: true %>
   <%= label f, :username, "User name", class: "control-label" %>
   <%= error_tag f, :username %>
  </div>

  <div class="form-label-group">
   <%= password_input f, :password, class: "form-control", placeholder: "Password", required: true %>
   <%= label f, :password, class: "control-label" %>
   <%= error_tag f, :password %>
  </div>

  <div class="form-label-group">
   <%= password_input f, :password_confirmation, class: "form-control", placeholder: "Password confirmation", required: true %>
   <%= label f, :password_confirmation, "Password confirmation", class: "control-label" %>
   <%= error_tag f, :password_confirmation %>
  </div>

  <%= submit "Sign Up", class: "btn btn-lg btn-primary btn-block" %>
 <% end %>
</div>
```

auth.ex

```
def register(params) do
  User.registration_changeset(%User{}, params) |> Repo.insert()
end
```

```
    <% else %>
      <%= link "Sign In", to: session_path(@conn, :new), class: "btn btn-outline-primary" %>
      <%= link "Sign Up", to: registration_path(@conn, :new), class: "btn btn-outline-primary ml-md-3"
%>
    <% end %>
```

```
defp deps do
  [
    # ...
    {:comeonin, "~> 4.0"},
    {:bcrypt_elixir, "~> 1.0"}
  ]
end
```

iex -S mix phx.server

```
Chat.Repo.get_by(Chat.Auth.User, email: "user@example.com") |>
Chat.Repo.delete()

password = Comeonin.Bcrypt.hashpwsalt("password")

%Chat.Auth.User{} |>
  Chat.Auth.User.changeset(%{email: "user@example.com", encrypted_password:
password, username: "user"}) |>
  Chat.Repo.insert()
```

Atualizando o Auth

isso:

user.encrypted_password == password

por isso:

Comeonin.Bcrypt.checkpw(password, user.encrypted_password)

```
@doc false
def changeset(%User{} = user, attrs) do
  user
  |> cast(attrs, [:email, :username])
  |> validate_required([:email, :username])
  |> validate_length(:username, min: 3, max: 30)
  |> unique_constraint(:email)
  |> unique_constraint(:username)
end
```

```
@doc false
def registration_changeset(%User{} = user, attrs) do
  user
  |> changeset(attrs)
  |> validate_confirmation(:password)
  |> cast(attrs, [:password], [])
  |> validate_length(:password, min: 6, max: 128)
  |> encrypt_password()
end
```

```
defp encrypt_password(changeset) do
  case changeset do
    %Ecto.Changeset{valid?: true, changes: %{password: password}} ->
      put_change(changeset, :encrypted_password,
Comeonin.Bcrypt.hashpwsalt(password))
    _ ->
      changeset
  end
end
```

```
schema "users" do
  # ...
  field :password, :string, virtual: true
  field :password_confirmation, :string, virtual: true
end
```

```
touch lib/chat_web/plugs/authenticate_user.ex

touch lib/chat_web/plugs/set_current_user.ex
```

```
defmodule ChatWeb.Plugs.SetCurrentUser do
  import Plug.Conn
  alias Chat.Repo
  alias Chat.Auth.User
  def init(_params) do
  end
  def call(conn, _params) do
    user_id = Plug.Conn.get_session(conn, :current_user_id)
    cond do
      current_user = user_id && Repo.get(User, user_id) ->
        conn
        |> assign(:current_user, current_user)
        |> assign(:user_signed_in?, true)
      true ->
        conn
        |> assign(:current_user, nil)
        |> assign(:user_signed_in?, false)
    end
  end
end
```

```
defmodule ChatWeb.Plugs.AuthenticateUser do
  import Plug.Conn
  import Phoenix.Controller
  alias ChatWeb.Router.Helpers
  def init(_params) do
  end
  def call(conn, _params) do
    if conn.assigns.user_signed_in? do
      conn
    else
      conn
      |> put_flash(:error, "You need to sign in or sign up before continuing.")
      |> redirect(to: Helpers.session_path(conn, :new))
      |> halt()
    end
  end
end
```

```
pipeline :browser do
  # ...
  plug ChatWeb.Plugs.SetCurrentUser
end
```

/templates/layout/app.html.eex

```
<%= if @user_signed_in? do %>
  <nav class="my-2 my-md-0 mr-md-3">
    Signed in as: <strong><%= @current_user.username %></strong>
```

```
→ mix ecto.gen.migration add_user_id_to_rooms
```

```
 def change do
    alter table(:rooms) do
      add :user_id, references(:users)
    end
  end
```

```
mix ecto.migrate
```

```
schema "rooms" do
  # ...
  belongs_to :user, Chat.Auth.User
  # ...
end


schema "users" do
  # ...
  has_many :rooms, Chat.Conversation.Room
  # ...
end
```

conversation.ex

```
def create_room(user, attrs \\ %{}) do
  user
  |> Ecto.build_assoc(:rooms)
```

roomcontroller.ex

```
 alias Chat.Auth.Authorizer

 plug ChatWeb.Plugs.AuthenticateUser when action not in [:index]
plug :authorize_user when action in [:edit, :update, :delete]

def create(conn, %{"room" => room_params}) do
   case Conversation.create_room(conn.assigns.current_user, room_params) do
```

```
defp authenticate_user(conn, _params) do
  if conn.assigns.user_signed_in? do
    conn
  else
    conn
    |> put_flash(:error, "You need to sign in or sign up before continuing.")
    |> redirect(to: session_path(conn, :new))
    |> halt()
  end
end
```

```
defp authorize_user(conn, _params) do
  %{params: %{"id" => room_id}} = conn
  room = Conversation.get_room!(room_id)
  if Authorizer.can_manage?(conn.assigns.current_user, room) do
    conn
  else
    conn
    |> put_flash(:error, "You are not authorized to access that page")
    |> redirect(to: room_path(conn, :index))
    |> halt()
  end
```

lib/chat_web/templates/room/show.html.eex

```
<%= if Prater.Auth.Authorizer.can_manage?(@current_user, @room) do %>
  <div>
    <%= link "Edit", to: room_path(@conn, :edit, @room.id), class: "btn btn-default"
%>
    <%= link "Delete", to: room_path(@conn, :delete, @room), method: :delete,
data: [confirm: "Are you sure?"], class: "btn btn-danger" %>
  </div>
<% end %>
```