0.1. SSR Counter

```elixir
defmodule YapWeb.CounterLive do
 use YapWeb, :live_view

 @impl true
 def render(assigns) do
  ~H"""
  <Layouts.app flash={@flash}>
   <div class="flex flex-col items-center justify-center">
    <div class="p-6 w-80 text-center">
     <h1 class="text-3xl font-bold mb-4">Counter</h1>
     <!-- 2. Show initial count -->
     <p class="text-4xl font-semibold text-primary mb-6">{@count}</p>
     <div class="flex justify-center space-x-6">
      <!-- 3. Make buttons react to clicks -->
      <.button phx-click="increment">+</.button>
      <.button phx-click="decrement">-</.button>
     </div>
    </div>
   </div>
  </Layouts.app>
  """
 end

 @impl true
 def mount(_params, _session, socket) do
  # 1. Assign the initial count
  {:ok, assign(socket, count: 0)}
 end

  # 4. Write the event handlers
 @impl true
 def handle_event("increment", _params, socket) do
  {:noreply, assign(socket, count: socket.assigns.count + 1)}
 end

 @impl true
 def handle_event("decrement", _params, socket) do
  {:noreply, assign(socket, count: socket.assigns.count - 1)}
 end
end
```

0.2. Global Counter

```elixir
defmodule YapWeb.GlobalCounterLive do
  {...}
end
```

```elixir
defmodule YapWeb.Router do
  live "/counter/global", GlobalCounterLive, :show
end
```

Change the navigate path.

```html
# lib/yap_web/components/layouts.ex
<.link navigate={~p"/counter/global"} class="btn btn-ghost">Global Counter</.link>
```

Test it and show that other clients don't see changes. So add broadcasting.

```elixir
defmodule Yap.GlobalCounter do
  use Agent

  @topic "global_counter"

  def subscribe do
    Phoenix.PubSub.subscribe(Yap.PubSub, @topic)
  end

  def increment do
    Agent.update(__MODULE__, fn count -> count + 1 end)
    broadcast()
  end

  def decrement do
    Agent.update(__MODULE__, fn count -> count - 1 end)
    broadcast()
  end

  defp broadcast do
    Phoenix.PubSub.broadcast(Yap.PubSub, @topic, {:global_counter, get()})
  end
end
```

Then call it from the liveview.

```elixir
defmodule YapWeb.GlobalCounterLive do
  # 1. Add the subscribe to mount
  def mount(_params, _session, socket) do
    if connected?(socket), do: GlobalCounter.subscribe()
    ...
  end

  # 2. Add handle_info to handle the broadcast
  @impl true
  def handle_info({:global_counter, count}, socket) do
    {:noreply, assign(socket, count: count)}
  end

  # 3. Rewrite the event handlers
  @impl true
  def handle_event("increment", _params, socket) do
    GlobalCounter.increment()
    {:noreply, socket}
  end

  @impl true
  def handle_event("decrement", _params, socket) do
    GlobalCounter.decrement()
    {:noreply, socket}
  end
end
```

0.3. User Form (Name)

Setup a cookie for identifying the user.

```elixir
defmodule YapWeb.Router do

  import YapWeb.UserTracking

  pipeline :browser do
    ...
    plug :ensure_anonymous_cookie
  end

  scope "/", YapWeb do
    live_session :user_tracking, session: {YapWeb.UserTracking, :build_session, []} do
      live "/form", UserFormLive, :show
    end
  end
end
```

Show the user assigned id.

```elixir
defmodule YapWeb.UserFormLive do
        # 2. Show it
       <h1 class="text-2xl mt-4">USER NAME ({@whoami})</h1>

  # 1. Add id to the assigns
  def mount(_params, %{"whoami" => whoami}, socket) do
    {:ok,
     socket
     |> assign(:whoami, whoami)
           ...
     }
  end
end
```

Get the user data saved in the memory cache.

```elixir
defmodule YapWeb.UserFormLive do

  # 2. Show it
  <h1 :if={@name} class="text-2xl mt-4">{@name}</h1>

  # 1. Get initial name from memory cache
  def mount(_params, %{"whoami" => whoami}, socket) do
    data = MemoryCache.get(whoami, %{name: nil})

    {:ok,
     socket
     |> assign(:name, name)
           ...
     }
  end
end
```

Handle the form submit.

```elixir
defmodule YapWeb.UserFormLive do

  # 1. Add the event target
  <.form for={@form} phx-submit="submit-user-data">

  # 2. Write the event handler
  def handle_event("submit", %{"user" => %{"name" => name}}, socket) do
    case validate_name(name) do
      [] ->
        save_user_data!(socket, :name, name)

        {:noreply, assign(socket, name: name)}

      errors ->
        form = to_form(%{"name" => name}, as: "user", errors: errors)

        {:noreply, assign(socket, form: form)}
    end
  end
end
```

Validate the form.

```elixir
defmodule YapWeb.UserFormLive do
  # 1. Add the event target
  <.form for={@form} phx-change="validate-user-data" phx-submit="submit-user-data">

  # 2. Write the event handler
  def handle_event("validate-user-data", %{"user" => %{"name" => name}}, socket) do
    errors = validate_name(name)

    form = to_form(%{"name" => name}, as: "user", errors: errors)

    {:noreply, assign(socket, form: form)}
  end
end
```

0.4. User Form (Avatar)

```elixir
defmodule YapWeb.UserFormLive do
  # 1. add the avatar upload component
  <.upload_avatar uploads={@uploads} />


  # 3. add avatar to the assigns in mount chaging the defaults
  # maybe delete the cookie and start over
  def mount(_params, %{"whoami" => whoami}, socket) do
    data = MemoryCache.get(whoami, %{name: nil, avatar: nil})
      ...
    |> assign(:avatar, data.avatar)
      ...
  end

  # 2. Write the event handlers
  @impl true
  def handle_event("validate-upload", _params, socket) do
    {:noreply, socket}
  end

  @impl true
  def handle_event("submit-upload", _params, socket) do
    [avatar] =
      consume_uploaded_entries(socket, :avatar, fn %{path: path}, _entry ->
        dest = Path.join("priv/static/uploads", Path.basename(path))

        File.cp!(path, dest)

        save_user_data!(socket, :avatar, Path.basename(dest))

        {:ok, dest}
      end)

    {:noreply, assign(socket, avatar: avatar)}
  end

  @impl true
  def handle_event("cancel-upload", %{"ref" => ref}, socket) do
    {:noreply, cancel_upload(socket, :avatar, ref)}
  end
end
```

Then start showing the avatar.

```elixir
defmodule YapWeb.UserFormLive do
    <img :if={@avatar} src={"/uploads/#{@avatar}"} class="w-32 h-32 rounded-full" />
end
```

0.5. Phoenix Generators

Run the generator.

```shell
mix phx.gen.live Blog Post posts title:string body:text published:boolean
```

Add the routes.

```elixir
live "/posts", PostLive.Index, :index
live "/posts/new", PostLive.Form, :new
live "/posts/:id", PostLive.Show, :show
live "/posts/:id/edit", PostLive.Form, :edit
```

Run the migrations.

```shell
mix ecto.migrate
```

Add the link to the layout.

```html
<.link navigate={~p"/posts"} class="btn btn-ghost">Posts</.link>
```

0.6. JS Hooks

```js
const Hooks = {}

Hooks.AutoResizeTextarea = {
  mounted() {
    this.resizeTextarea()

    this.el.addEventListener('input', () => {
      this.resizeTextarea()
    })
  },
  updated() {
    this.resizeTextarea()
  },
  resizeTextarea() {
    this.el.style.height = 'auto'
    this.el.style.height = `${this.el.scrollHeight}px`
  }
}

const csrfToken = document.querySelector("meta[name='csrf-token']").getAttribute("content")
const liveSocket = new LiveSocket("/live", Socket, {
 hooks: Hooks,
 longPollFallbackMs: 2500,
 params: {_csrf_token: csrfToken}
})
```

Add the hook to the body input from the post.

```html
<.input field={@form[:body]} type="textarea" phx-hook="AutoResizeTextarea" label="Body" />
```

0.7. Generate Auth

Run the generator and accept the LiveView defaults.

```shell
mix phx.gen.auth Accounts User users --hashing-lib argon2
```

Download new dependencies and run migrations.

```shell
mix deps.get
mix ecto.migrate
```