

# Embed a Phoenix page into WordPress

Aron Wolf



# Links

- ▶ **Repo:**

[https://github.com/aronwolf90/phoenix\\_embedded](https://github.com/aronwolf90/phoenix_embedded)



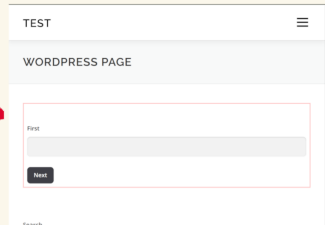
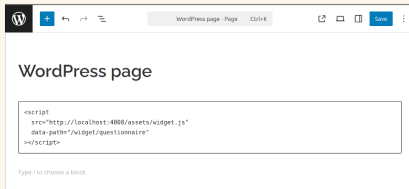
- ▶ **Presentation:**

[https://github.com/aronwolf90/  
phoenix-embedded-presentation/blob/master/main.  
pdf](https://github.com/aronwolf90/phoenix-embedded-presentation/blob/master/main.pdf)



# Goal

- ▶ Embed a small html snipped into WP that renders the LiveView page.

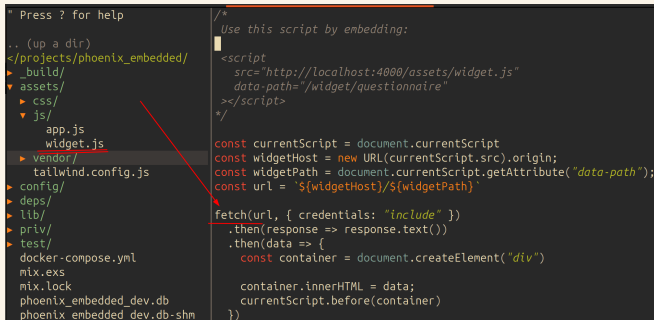


## Steps needed to render a LiveView page

- ▶ Load html through a normal http request.
- ▶ Mount LiveSocket through a js script.

# Fetch html

- ▶ Fetch html.
- ▶ Create container.
- ▶ Add fetched html as innerHtml to the container.
- ▶ Attach container above script.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. A red arrow points from the `widget.js` file in the file explorer to the `fetch` function in the code editor.

```
* Press ? for help

.. (up a dir)
</projects/phoenix_embedded/
  >_build/
  >assets/
    >css/
    >js/
      app.js
      widget.js
  >vendor/
    tailwind.config.js
  >config/
  >deps/
  >lib/
  >priv/
  >test/
    docker-compose.yml
    mix.exs
    mix.lock
    phoenix_embedded_dev.db
    phoenix_embedded_dev.db-shm

/*
Use this script by embedding:

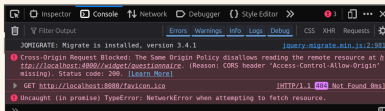
<script
  src="http://localhost:4000/assets/widget.js"
  data-path="/widget/questionnaire"
></script>
*/

const currentScript = document.currentScript
const widgetHost = new URL(currentScript.src).origin;
const widgetPath = document.currentScript.getAttribute("data-path");
const url = `${widgetHost}/${widgetPath}`

fetch(url, { credentials: "include" })
  .then(response => response.text())
  .then(data => {
    const container = document.createElement("div")

    container.innerHTML = data;
    currentScript.before(container)
  })
```

# Fix cors error



```
test.exs
deps/
lib/
  phoenix_embedded/
  phoenix_embedded_web/
    components/
    controllers/
    live/
      endpoint.ex
      gettext.ex
      router.ex
      telemetry.ex
  phoenix_embedded.ex
  phoenix_embedded_web.ex

end

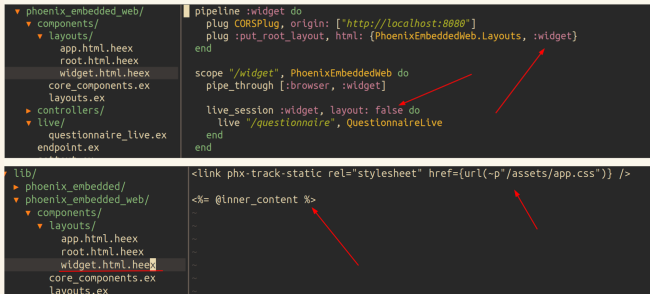
pipeline :widget do
  plug CORSPlug, origin: ["http://localhost:8080"]
  plug :put_root_layout, html: {PhoenixEmbeddedWeb.Layouts, :widget}
end

scope "/widget", PhoenixEmbeddedWeb do
  pipe_through [:browser, :widget]

  live_session :widget do
    live "/questionnaire", QuestionnaireLive
  end
end
```

# Use separate layout

- ▶ Layout without header or body.
- ▶ Without LiveView layout.
- ▶ With stylesheet link.



The screenshot displays the Phoenix Embedded Web IDE interface, showing the file explorer on the left and the code editor on the right. The file explorer shows the project structure for `phoenix_embedded_web`, with the `layouts` directory selected. The code editor shows the `pipeline` configuration and the `widget.html.heex` template. Red arrows point from the text in the list to specific parts of the code.

```
phoenix_embedded_web/  
  components/  
    layouts/  
      app.html.heex  
      root.html.heex  
      widget.html.heex  
      core_components.ex  
      layouts.ex  
    controllers/  
    live/  
      questionnaire_live.ex  
      endpoint.ex  
  lib/  
    phoenix_embedded/  
    phoenix_embedded_web/  
      components/  
        layouts/  
          app.html.heex  
          root.html.heex  
          widget.html.heex  
          core_components.ex  
          layouts.ex  
  test/  
    test_helper.ex  
    test_cases.ex
```

```
pipeline :widget do  
  plug CORSPlug, origin: ["http://localhost:8080"]  
  plug :put_root_layout, html: {PhoenixEmbeddedWeb.Layouts, :widget}  
end  
  
scope "/widget", PhoenixEmbeddedWeb do  
  pipe_through [:browser, :widget]  
  
  live_session :widget, layout: false do  
    live "/questionnaire", QuestionnaireLive  
  end  
end
```

```
<link phx-track-static rel="stylesheet" href={url(~p"/assets/app.css")} />  
  
<%= @inner_content %>  
  
<%= @inner_content %>
```

# Mount LiveSocket

- ▶ Copy LiveSocket code from app.js.
- ▶ Add csrf token to layout.

```
▼ phoenix_embedded_web/  
  ▼ components/  
    ▼ layouts/  
      app.html.heex  
      root.html.heex  
      widget.html.heex
```

```
<link phx-track-static rel="stylesheet" href={url(~p"/assets/app.css")} />  
<meta name="widget-csrf-token" content={get_csrf_token()} />  
%<= @inner_content %>
```

```
* Press ? for help
```

```
.. (up a dir)  
./projects/phoenix_embedded/  
  ▼ build/  
  ▼ assets/  
    ▼ css/  
    ▼ js/  
      app.js  
      widget.js  
  ▼ vendor/  
  tailwind.config.js  
  ▼ config/  
    config.exs  
    dev.exs  
    prod.exs  
    runtime.exs  
    test.exs  
  ▼ deps/  
  ▼ lib/  
  phoenix_embedded/
```

```
const url = `${widgetHost}/${widgetPath}`  
  
fetch(url, { credentials: "include" })  
  .then(response => response.text())  
  .then(data => {  
    const container = document.createElement("div")  
  
    container.innerHTML = data;  
    currentScript.before(container)  
  
    const csrfToken = document.querySelector("meta[name='widget-csrf-token']").getAttribute("content")  
    const liveSocket = new LiveSocket(`${widgetHost}/live`, Socket, {  
      longPollFallbackMs: 2500,  
      params: { _csrf_token: csrfToken }  
    })  
  
    liveSocket.href = url  
    liveSocket.connect()  
    window.liveSocket = liveSocket  
  })
```



# Multi step form

- ▶ Use one LiveView page.
- ▶ Hide/show parts when one clicks on next.
- ▶ Use global form to make reconnection work correctly.
- ▶ Use hidden fields for internal assigns.









```
def page(assigns) do
  ~H"""
  <span class={[@page != @current_page && "hidden"]}>
    {render_slot(@inner_block)}
  </span>
  """
end
```

```
@impl true
def render(assigns) do
  ~H"""
  <.simple_form
    id="questionnaire"
    for={@form}
    phx-change="validate"
    phx-submit="submit"
    class=""
  >
    <input type="hidden" name="current_page" value={@current_page} />

    <.page page="first" current_page={@current_page} >
      <.input field={@form[:first]} label="First" />

      <.button class="mt-4">Next</button>
    </.page>
  """
end
```

# Demo



TEST

WORDPRESS PAGE

First

Next

# Debug

- ▶ Open external page.
- ▶ Open console.
- ▶ Paste something like the following in the console.

```
// remove current page content
```

```
main = document.querySelectorAll("main")[0]  
new_main = document.createElement("main")  
main.replaceWith(new_main)
```

```
// load LiveView page
```

```
script = document.createElement("script")  
script.src = "http://localhost:4000/assets/widget.js"  
script.setAttribute("data-path", "/widget/questionnaire")  
new_main.appendChild(script)
```

Thank You!

Questions?

[aronwolf90@gmail.com](mailto:aronwolf90@gmail.com)