How to Make a CRUD SPA (Hotwire Best Fundamentals)

Requirements:

Rails 7+

Ruby 3+

Decent Knowledge of Normal CRUD layouts in Ruby on Rails.

(Create Read Update Destroy)

Making Creating Posts Realtime

- First create post scaffold with Title and Body.
- add root "posts#index" to config/routes.rb
- We need to render the form on the index page, so add this line to the index page <%= turbo_frame_tag id="new_post", src:new_post_path %>
- That line of code will correspond to the new page and try and render it.
- which we need to change (or we can change the _form.html.erb inside)
- **Wrap all of posts/new.html.erb with <%= turbo_frame_tag id="new_post" do %> Old Content<% end %>** OR Wrap all of posts/_form.html.erb with <%= turbo_frame_tag id="new_post" do %> Old Content <% end %>
- I prefer the first option, but do both so you have a better understanding how this works.

Now you will have the form displaying on your Index page. Very good

- Then Go posts_controller.rb and replace the create action with

```
def create @post = Post.new(post_params) respond_to do |format|
if @post.save format.turbo_stream else format.html { render
:new, status: :unprocessable_entity } end end end
```

We're mainly looking at the line format.turbo_stream here.

This is looking for a file which haven't created yet.

The file it is looking for is called **create**.turbo_stream.erb. it's named after the method it is in.

We need to create that file, it does not come preinstalled.

So Create a File Called create.turbo_stream.erb in app/views/posts folder

Then in that file add this line of code

<%= turbo_stream.append "posts", @post %>

What this does is add the new "@post" to the "posts" div on the index page. (Default div ID)

To Explain <%= turbo_stream.append "posts", @post %> a little further.

if you remember in the create action in the posts controller (Code Above)

we are creating a post, and we call it "@post"

we call it "@post" because of the line @post = Post.new(post_params)

So This line of code adds the "@post" instance variable to the div.

Now to make the experience more realtime, go to index.html.erb

We don't need the following line in index.html.erb.

<%= link_to "Show this post", post %>

Delete it or comment it out.

Great Job, Now

So Now we have our form on our index page (same place as the posts) and when you click create a new post is automatically added to the "posts" div.

All in Realtime, though the form does not clear which we will fix in a second.

Clearing The Form After Creating A Post

Assuming the Turbo Frame is in the Form Part, so.

<%= turbo_frame_tag id="new_post" do %>

Form Content

<% end %>

Then this method will, work so make sure you have it setup like this.

Add this Code to create.turbo_stream.erb

```
<%= turbo_stream.replace 'new_post', partial: 'form', locals: {
post: Post.new } %>
```

And that's Done! Now the form will clear After submitting.

Making Deleting Posts Realtime

Now we will make Deleting posts RealTime.

To Achieve this:

- Go To Show.html.erb and copy this line to the bottom of _post.html.erb

```
<%= button_to "Destroy this post", @post, method: :delete %>
```

In _posts.html.erb Change it so that it's using the post variable and not the "@post" variable

Like This <%= button_to "Destroy this post", **post**, method: :delete %>

Now that you have this line in _posts.html.erb, Remove it from the Show.html.erb as it is no longer needed at all.

Done and Dusted

Adding Inline Editing

what we are going to do, is add turbo frames to our form and our post partial. When an element in the post partial is clicked, we will render the form for that element on the same page. when the form is submitted, we will update the element without refreshing the whole page. Simple

In _post.html.erb The important parts we are focusing on are these lines

```
<% frame_id = dom_id(post, "title_turbo_frame")%> <%= form_with
model: post, data: { turbo_frame: frame_id} do %> <%=
turbo_frame_tag frame_id do %> <p> <strong>Title:</strong> <%=
post.title %> </p> <%= link_to "Edit Title", edit_post_path(post)%>
<% end %> <% end %>
```

We have turbo frames that correspond to these lines of code in _form.html.erb

```
<% if post.persisted? %> <%= form_with(model: post) do |form| %> <%
frame_id = dom_id(post, "title_turbo_frame")%> <%= turbo_frame_tag
frame_id do %> <div> <%= form.label :title, style: "display: block"
%> <%= form.text_area :title %> </div> <%= form.button "Save"%> <%=
link_to "Cancel", post%> <% end %> <% end %> <% end %>
```

In the first Box of code, when the edit link is clicked, it goes to the edit path, but it scans all the code and finds a matching turbo frame to one it is already in, therefore it substitutes the current turbo frame for the latter. I.E Code Box number 1 turns into Code Box number 2 on the same page..

Then we have a small form where we can edit the title and save it. We can also cancel if we do not want to edit anything.

You have to copy the title code in the form and place it outside the form as well as inside. The code on the outside should only be rendered if editing the title. But the code on the inside should be rendered regardless. This is because we always want the user to be able to create a post.

<% if post.persisted? %>Code<% end %> This means to only render the code if the post is being edited. as you don't want to render the form if it's not what the user wanted to do.

The line `<% frame_id = dom_id(post, "title_turbo_frame") %>` is used to generate a unique identifier for the Turbo Frame tag that wraps around a specific piece of content related to the post's title. This identifier is crucial for Turbo Streams to identify the correct frame to update when changes occur.

Here's how it works:

- `dom_id(post, "title_turbo_frame")` : This is a Rails helper method `dom_id` used to generate a unique identifier for a DOM element based on the provided object (`post`) and an optional prefix (`"title_turbo_frame"`). It ensures that the identifier is unique within the scope of the HTML document.
- `<%= turbo_frame_tag frame_id do %>` : This is a Turbo Frames tag that creates a container that Turbo Streams can target for updates. The `frame_id` generated by `dom_id` is used as the `id` attribute of the Turbo Frame. This ensures that the Turbo Stream updates sent from the server will be directed to the correct frame.

So, essentially, this code is creating a Turbo Frame with a unique identifier based on the post's title, which can then be updated dynamically using Turbo Streams without affecting other parts of the page.

That's my Single Page Application Tutorial Blog post

Hope you got it all right. Like this blog post and youtube video.