AJAX IN RAILS

Let's fix the *scrolling back to the top* issue using JavaScript views in Rails.

RENDER JS VIEWS

Rails controllers can render different formats based on the request's "Accept" header:

```
# app/controllers/reviews_controller.rb
# [...]
  def create
    # ...
    if @review.save
      respond_to do |format|
        format.html { redirect_to restaurant_path(@restaurant) }
        format.js
      end
    else
  end
```

It will respond to requests that have `Accept: text/javascript` in their headers with a `js.erb` view.

IMPLEMENT THE `create.js.erb` VIEW

Add some JavaScript to insert the review partial at the beginning of the `#reviews` div.

```
document.getElementById('reviews').insertAdjacentHTML('afterBegin', '<%=j render @review %>')
```

WTH ARE THE `js.erb` VIEWS?

Don't think of the `js.erb` files as actual views (because we're not reloading the page or rendering new HTML). Think of it simply as **some JS code you run on the current page** instead of sending HTML at the end of your controller action.

CREATING JAVASCRIPT COMPONENTS WITH STIMULUS

SIMPLE COMPONENT: COLLAPSIBLE FORM

Let's create an "collapsible form" component for the `reviews#new` form, that expands when we click on a button.

Let's do it in pure **JavaScript**

STIMULUS JS

A JavaScript library that pairs well with Turbolinks.

Read the Handbook

STIMULUS IN RAILS

Quick setup in rails:

```
rails webpacker:install:stimulus
```

Check your app/javascript/packs/application.js

STIMULUS CONTROLLER

```
touch app/javascript/controllers/collapsible_form_controller.js
```

```
import { Controller } from "stimulus";

export default class extends Controller {
  connect() {
    console.log(`Hello from the ${this.identifier} controller!`);
  }
}
```

DATA-CONTROLLER

Connect the component to the `collapsible_form` controller by adding a `data-controller` attribute.

```
<!-- app/views/restaurants/show.html.erb -->

<div data-controller="collapsible-form"> <!-- create a container for our new component -->
  <button class="btn btn-outline-primary">Leave a review</button>

  <%= simple_form_for([ @restaurant, @review ], remote: true) do |f| %>
    <!-- [...] -->
  <% end %>
</div>
```

Set the `data-controller` in a div that contains both:

- the element listening to an event (the button)
- the element you want to update (the form)

DATA-TARGET

`data-<controller>-target` is the equivalent of `document.querySelector`

```
<%= simple_form_for([ @restaurant, @review ],
                    html: { data: { collapsible_form_target: 'form' } },
                    remote: true) do |f| %>
```

Simple form will generate a form tag like this:

```
<form action="..." data-collapsible-form-target="form" ... >
```

`data-<controller-name>-target="targetName"`

TARGETS

```
import { Controller } from "stimulus";

export default class extends Controller {
  static targets = [ 'form' ];

  connect() {
    console.log(this.formTarget);
  }
}
```

`this.formTarget` returns the first one, `this.formTargets` returns them all

CONNECT

Set the initial state of the form in the `connect()` action.

```
export default class extends Controller {
  static targets = [ 'form' ];

  connect() {
    this.formTarget.style.height = "0"
    this.formTarget.style.overflow = "hidden"
    this.formTarget.style.transition = "height 0.2s ease-in"
  }
}
```

DATA-ACTION

Listening to the `click` event on the button (`addEventListener`):

```
<!-- app/views/restaurants/show.html.erb -->

<div data-controller="collapsible-form">
  <button class="btn btn-outline-primary"
          data-action="click->collapsible-form#expandForm">Leave a review</button>
  <!-- [...] -->
</div>
```

Syntax: `event->controller-name#actionName`

ACTION

```
import { Controller } from "stimulus";

export default class extends Controller {
  static targets = [ 'form' ];
  // ...
  expandForm(event) {
    console.log(event);
  }
}
```

Let’s expand the form!

SETTINGS

Use data attributes to add settings to your component

```
  <div data-controller="collapsible-form" data-expanded-height="150px">
```

```
expandForm(event) {
  this.formTarget.style.height = this.element.dataset.expandedHeight
  event.currentTarget.remove() // Remove the button after expanding the form
}
```

NEW ACTION: SUBMIT ON ENTER

```
export default class extends Controller {
  // ...
  submitOnEnter(event) {
    if (event.key === 'Enter' && !event.shiftKey) {
      event.preventDefault()
      this.formTarget.style.height = "0"
      Rails.fire(this.formTarget, 'submit')
    }
  }
}
```

We use `Rails.fire` (instead of `form.submit`) in order to submit the form *remotely*.

Make sure to add the `rails-ujs` plugin to Webpack in your `environment.js` file.

```
environment.plugins.prepend('Provide',
  new webpack.ProvidePlugin({
    // ...
    Rails: ['@rails/ujs']
  })
);
```

ADDING THE `data-action` IN THE VIEW

```
<div data-controller="collapsible-form" data-expanded-height=150>
  <!-- [...] -->
  <%= f.input :content, as: :text,
    label: false,
    input_html: {
      placeholder: 'Press enter to submit your review.',
      data: { action: 'keydown->collapsible-form#submitOnEnter' }
    } %>
</div>
```

STIMULUS TAKEAWAYS

- `querySelector` is replaced by `data-<controller-name>-target="targetName"`
- `addEventListener` is replaced by `data-action`
- the `data-controller` wraps the other elements

SYNTAX

- `data-controller="controller-name"`
- `data-<controller-name>-target="targetName"`
- `data-action="event->controller-name#actionName"`

DEMO APP

You can test the final app in production.

HAPPY AJAXIFICATION!