← Volver a la lista de capítulos

Mensajes flash con Hotwire

Publicado el 24 de febrero de 2022

En este capítulo, aprenderemos cómo agregar mensajes flash con Turbo y cómo hacer una bonita animación con Stimulus.

Añadiendo mensajes flash a nuestro controlador CRUD

Ahora que tenemos un controlador CRUD funcional para nuestro `Quote` modelo, queremos agregar mensajes flash para mejorar la usabilidad de nuestra aplicación. En este capítulo, veremos cómo funcionan los mensajes flash con Turbo.

Antes de agregar mensajes flash *con* Turbo, necesitamos hacer que funcionen *sin* Turbo, como solíamos hacer antes de Ruby on Rails 7. Para hacer esto, deshabilitaremos Turbo en toda la aplicación, como aprendimos en el capítulo Turbo Drive :

```
// app/javascript/application.js

import "./controllers"

// The two following lines disable Turbo on the whole application
import { Turbo } from "@hotwired/turbo-rails"
Turbo.session.drive = false
```

Ahora que Turbo está deshabilitado en toda la aplicación, podemos probar en el navegador que nuestro editor de citas se comporta como esperaríamos que lo hiciera sin Turbo. Cada **clic en un vínculo abre una nueva página** y **cada envío de formulario redirecciona** a la página `Quotes#index` o a la `Quotes#show` página.

Con Turbo desactivado, es hora de empezar a trabajar en nuestros mensajes flash. Ya configuramos mensajes flash en el primer capítulo cuando las acciones `#create` , `#update` , y `#destroy` tienen éxito en nuestro trabajo `QuotesController` gracias a la `notice` opción:

```
class QuotesController < ApplicationController
  # All the previous code

  def create
    @quote = current_company.quotes.build(quote_params)

    if @quote.save
      respond_to do |format|
        format.html { redirect_to quotes_path, notice: "Quote was succes
        format.turbo_stream
      end
    else
      render :new, status: :unprocessable_entity
    end
  end

  # All the previous code

  def update
    if @quote.update(quote_params)
      redirect_to quotes_path, notice: "Quote was successfully updated."
    else
      render :edit, status: :unprocessable_entity
    end
  end

  # All the previous code

  def destroy
    @quote.destroy

    respond_to do |format|
```

```
      format.html { redirect_to quotes_path, notice: "Quote was successf
      format.turbo_stream
    end
  end
end
```

Nota : Si no está familiarizado con la `notice` notación de los mensajes flash, las dos sintaxis siguientes son equivalentes:

```
# Syntax 1
redirect_to quotes_path, notice: "Quote was successfully created."

# Syntax 2
flash[:notice] = "Quote was successfully created."
redirect_to quotes_path
```

Prefiero usar la primera sintaxis porque es de una sola línea, ¡pero siéntete libre de usar la segunda si lo deseas!

Incluso si los mensajes flash están configurados correctamente, **actualmente no los mostramos en las vistas** . Asegurémonos de que mostramos esos mensajes flash correctamente para el formato HTML antes de hablar de Turbo. Para ello, primero crearemos el mensaje flash parcial que contendrá el marcado para un solo mensaje flash:

```
<%# app/views/layouts/_flash.html.erb %>

<% flash.each do |flash_type, message| %>
  <div class="flash__message">
    <%= message %>
  </div>
<% end %>
```

Representaremos este mensaje flash de forma parcial en cada página de la aplicación directamente en el diseño:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <!-- All the head code -->
  </head>

  <body>
    <%= render "layouts/navbar" %>

    <div class="flash">
      <%= render "layouts/flash" %>
    </div>

    <%= yield %>
  </body>
</html>
```

Vamos a probar que todo esté conectado correctamente creando, actualizando o destruyendo una cita. ¡El mensaje de Flash aparece como se esperaba! Vamos a agregar un poco de CSS para que queden más bonitos:

```
// app/assets/stylesheets/components/_flash.scss

.flash {
  position:fixed;
  top: 5rem;
  left: 50%;
  transform: translateX(-50%);

  display: flex;
  flex-direction: column;
  align-items: center;
  gap: var(--space-s);

  max-width: 100%;
  width: max-content;
  padding: 0 var(--space-m);

  &__message {
    font-size: var(--font-size-s);
    color: var(--color-white);
    padding: var(--space-xs) var(--space-m);
    background-color: var(--color-dark);
    animation: appear-then-fade 4s both;
    border-radius: 999px;
  }
}
```

La `.flash` clase CSS es el contenedor de nuestros mensajes flash. Tiene una posición fija en la pantalla. A cada mensaje flash individual se le aplica un estilo gracias a la `.flash__message` clase CSS. Usaremos una animación personalizada para nuestros mensajes flash llamada `appear-then-fade`. Agreguemos un archivo para animaciones en nuestra arquitectura CSS:

```
// app/assets/stylesheets/config/_animations.scss

@keyframes appear-then-fade {
  0%, 100% {
    opacity:0
  }
  5%, 60% {
    opacity:1
  }
}
```

Con esos dos archivos agregados, agreguémoslos a nuestro archivo de manifiesto para que sean parte del paquete CSS:

```
// app/assets/stylesheets/application.sass.scss

@import "components/flash";
@import "config/animations";
```

Ahora que hemos añadido nuestro CSS, probemos la visualización de nuestros mensajes Flash en el navegador. ¡Nuestros mensajes Flash ahora tienen el estilo adecuado!

Sin embargo, hay un pequeño problema con nuestra implementación actual. Cuando pasamos el ratón sobre el área del mensaje flash, **el cursor del ratón cambia incluso cuando el mensaje flash ya no es visible**. Esto se debe a que, incluso si nuestro mensaje flash tiene una opacidad de cero, sigue estando presente en el DOM y por encima del resto del contenido de la página. **Para resolver este problema, debemos eliminar los mensajes flash del DOM cuando alcanzan una opacidad de cero**.

Aquí es donde agregaremos la **única línea de JavaScript que necesitamos en todo el tutorial**. Crearemos un pequeño controlador de estímulo que elimina el mensaje de Flash cuando `appear-then-fade` finaliza la animación.

Para hacer esto, escribamos el comando para generar un nuevo controlador de estímulo llamado `removals`:

```
bin/rails generate stimulus removals
```

Esto agrega un nuevo controlador de estímulo que se importa automáticamente en el `app/javascript/controllers/index.js` archivo:

```
// app/javascript/controllers/index.js

import { application } from "./application"

import HelloController from "./hello_controller.js"
application.register("hello", HelloController)

import RemovalsController from "./removals_controller.js"
application.register("removals", RemovalsController)
```

Como podemos ver, tenemos un `HelloController` que se generó automáticamente cuando creamos nuestra aplicación al principio del tutorial con el `bin/rails new` comando. Eliminémoslo ya que no lo necesitaremos:

```
bin/rails destroy stimulus hello
```

Este comando debería eliminar nuestro `HelloController` y actualizar el archivo de índice de los controladores:

```
// app/javascript/controllers/index.js

import { application } from "./application"

import RemovalsController from "./removals_controller.js"
application.register("removals", RemovalsController)
```

Nota : Parece haber un pequeño error en Rails donde las dos líneas que registran `HelloController` no se eliminan correctamente del archivo de índice de los controladores cuando se ejecuta el `bin/rails destroy stimulus hello` comando. Si ese es tu caso, puedes simplemente eliminarlas manualmente o ejecutar el `bin/rails stimulus:manifest:update` comando.

Implementemos ahora nuestro controlador de estímulo:

```
// app/javascript/controllers/removals_controller.js

import { Controller } from "@hotwired/stimulus"

export default class extends Controller {
  remove() {
    this.element.remove()
  }
}
```

Este controlador tiene una función simple llamada `remove`. Cuando llamamos a esta función, se elimina el nodo DOM donde está conectado el controlador.

Si esto es un poco abstracto por ahora, demostremos cómo funciona usando nuestro controlador en nuestros mensajes flash para eliminarlos del DOM cuando finaliza su animación:

```
<%# app/views/layouts/_flash.html.erb %>

<% flash.each do |flash_type, message| %>
  <div
    class="flash__message"
    data-controller="removals"
    data-action="animationend->removals#remove"
  >
    <%= message %>
  </div>
<% end %>
```

La biblioteca Stimulus nos permite vincular el comportamiento de JavaScript definido en los controladores de estímulo a HTML gracias a las convenciones de nombres en los atributos de datos .

El fragmento HTML anterior sugiere que cada mensaje flash está conectado a un `RemovalsController` agradecimiento mediante el `data-controller="removals"` atributo de datos. Cuando finaliza la animación, se llama a la función `remove` del agradecimiento mediante el atributo de

datos. `RemovalsController data-action="animationend->removals#remove`

Si hacemos una prueba en el navegador y creamos, actualizamos o destruimos una cita, deberíamos ver que el mensaje flash aparece en la página. Cuando la animación termina, el mensaje flash se elimina del DOM. Si inspeccionamos el DOM después de unos segundos, ¡el mensaje flash desaparece! Si pasamos el mouse sobre el área del mensaje flash, el cursor ya no cambia porque el mensaje flash se eliminó por completo del DOM.

¡Excelente! Ahora que todo está conectado para una aplicación que no utiliza Turbo, es hora de asegurarnos de que tengamos el mismo comportamiento con Turbo habilitado.

Mensajes Flash con Turbo en Rails 7

En primer lugar, eliminemos **las** líneas que agregamos al principio del capítulo para deshabilitar Turbo en toda la aplicación:

```
// app/javascript/application.js

import "./controllers"
import "@hotwired/turbo-rails"
```

Probemos en el navegador. Observamos que el comportamiento Turbo ha vuelto, pero los mensajes Flash han desaparecido. Expliquemos qué sucede.

Mensajes flash con Hotwire en la `#create`acción

Veamos la `QuotesController#create` acción. Para el formato HTML, el mensaje flash se establece gracias a la `notice` opción. Sin embargo, no se menciona ningún mensaje flash para el formato Turbo Stream:

```
# app/controllers/quotes_controller.rb

def create
  @quote = current_company.quotes.build(quote_params)

  if @quote.save
    respond_to do |format|
```

```
      format.html { redirect_to quotes_path, notice: "Quote was successf
      format.turbo_stream
    end
  else
    render :new
  end
end
```

Agreguemos el mismo mensaje flash para el formato Turbo Stream:

```
# app/controllers/quotes_controller.rb

def create
  @quote = current_company.quotes.build(quote_params)

  if @quote.save
    respond_to do |format|
      format.html { redirect_to quotes_path, notice: "Quote was successf
      format.turbo_stream { flash.now[:notice] = "Quote was successfully
    end
  else
    render :new
  end
end
```

Usamos `flash.now[:notice]` aquí y no `flash[:notice]` porque **las respuestas de Turbo Stream no redireccionan a otras ubicaciones** , por lo que el flash tiene que aparecer en la página ahora mismo.

Si ahora hacemos la prueba en el navegador, el mensaje Flash sigue sin aparecer en la página. Esto se debe a que no se menciona qué hacer con el mensaje Flash en la plantilla que se muestra cuando se crea correctamente una cita:

```
<%# app/views/quotes/create.turbo_stream.erb %>

<%= turbo_stream.prepend "quotes", @quote %>
<%= turbo_stream.update Quote.new, "" %>
```

Para que nuestro mensaje flash funcione con las respuestas de Turbo Stream, necesitamos agregar una línea para indicarle a Turbo que *anteponga* los

mensajes flash a una lista o *actualice* el contenido del contenedor del mensaje flash.

La acción Turbo Stream que utilizamos depende del efecto que queremos conseguir. Si vamos a apilar mensajes flash y queremos que el efecto de la aplicación abarque una sola página, podemos utilizar *prepend* . Si vamos a tener un solo mensaje flash en la pantalla a la vez, podemos utilizar *replace* .

Utilicemos la acción *prepend* en nuestro tutorial:

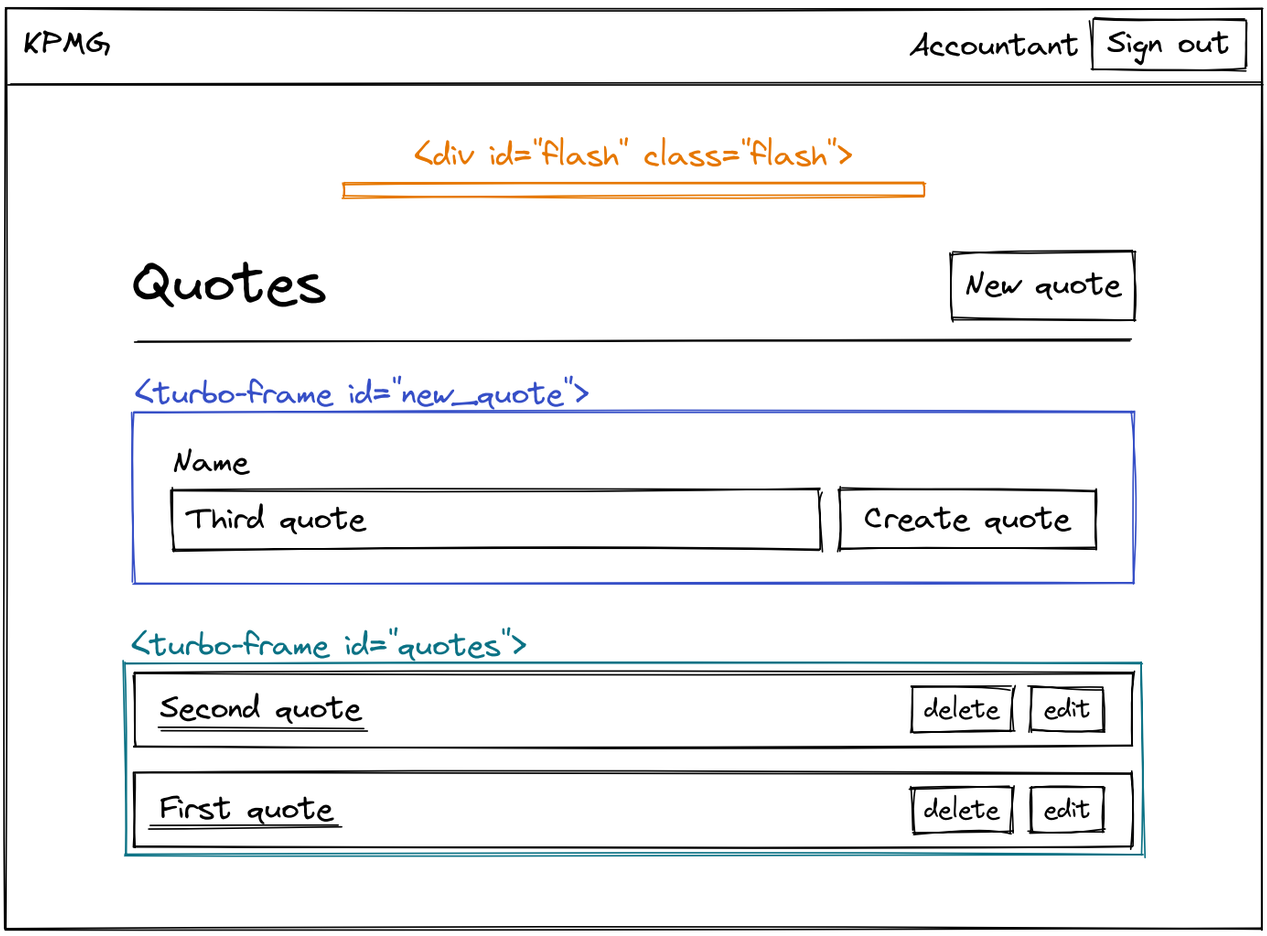
```
<%# app/views/quotes/create.turbo_stream.erb %>

<%= turbo_stream.prepend "quotes", @quote %>
<%= turbo_stream.update Quote.new, "" %>
<%= turbo_stream.prepend "flash", partial: "layouts/flash" %>
```

Esta última línea le indica a Turbo que anteponga al nodo DOM de id `flash` el contenido del `layouts/flash` parcial. Actualmente no tenemos un nodo DOM de id `flash` , por lo que debemos agregarlo al diseño de la aplicación:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- All the head code -->
  </head>

  <body>
    <%= render "layouts/navbar" %>

    <div id="flash" class="flash">
      <%= render "layouts/flash" %>
    </div>

    <%= yield %>
  </body>
</html>
```

Probemos en el navegador y creemos **dos citas muy rápidamente** . Deberíamos ver que los **dos mensajes Flash aparecen en la pantalla y desaparecen cuando termina la animación** .

Hagamos un boceto rápido de lo que sucede. Cuando estamos a punto de crear una cotización, nuestra página se ve así:

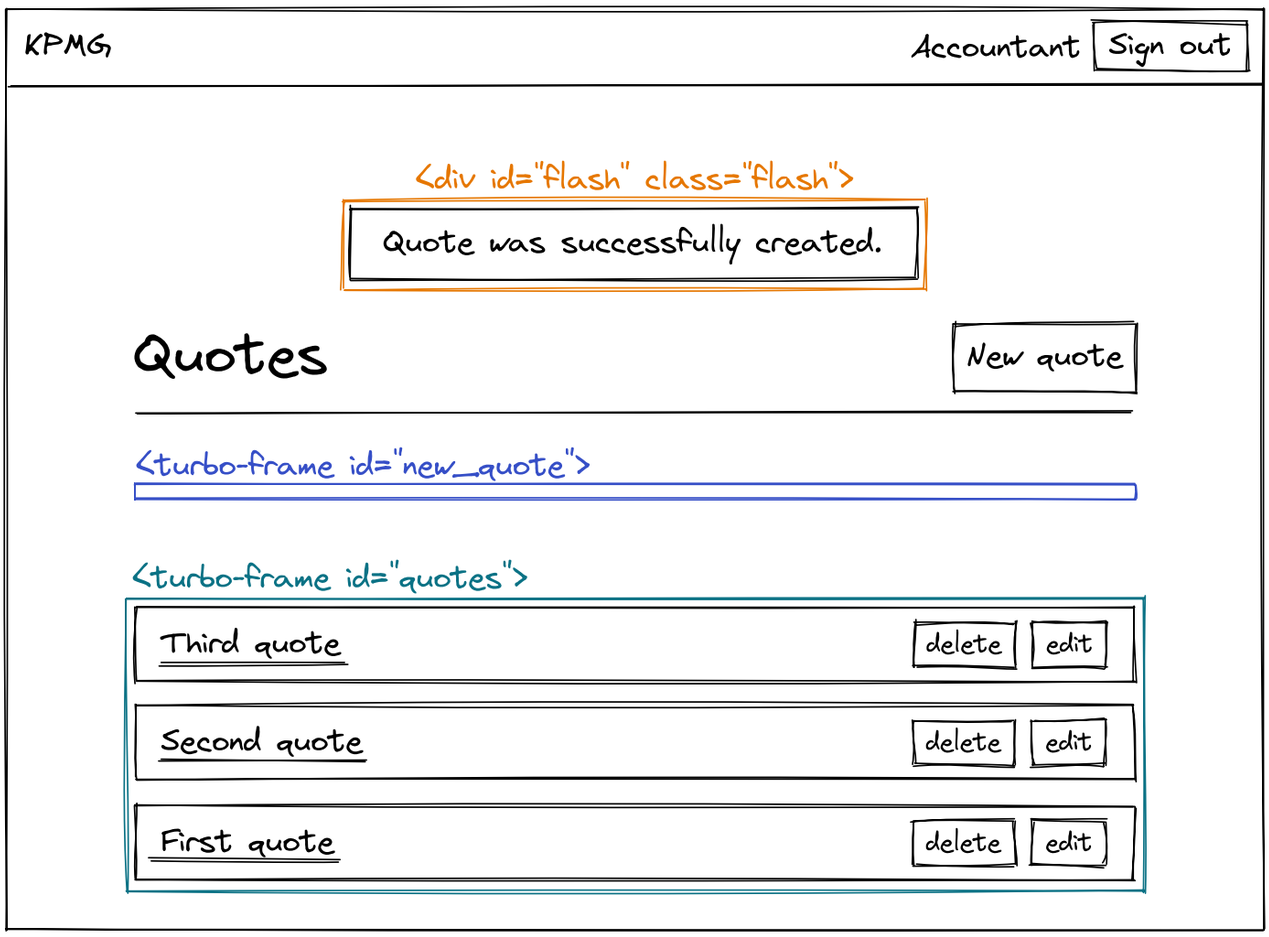


Boceto de la página de índice de cotizaciones cuando estamos a punto de crear una cotización

Cuando enviamos el formulario, se crea la cotización y `create.turbo_stream.erb` se muestra la vista. Esta vista le indica a Turbo que realice tres acciones:

- Anteponga la cita creada al nodo DOM con id `#quotes`
- Vacíe el contenido del nodo DOM con id `#new_quote`
- Anteponer el contenido del mensaje flash parcial al nodo DOM con id `#flash`

Cuando se ejecutan esas tres acciones, nuestra vista final se ve así:



Boceto de la página Quotes#index cuando acabamos de crear una cita

Ahora que nuestro mensaje flash funciona para la `QuotesController#create` acción, agreguemos mensajes flash a nuestras acciones `QuotesController#update` y `QuotesController#destroy`. ¡Siéntete libre de intentar hacerlo tú mismo antes de leer las dos secciones siguientes!

Mensajes flash con Hotwire en la `#update`acción

A diferencia de la `#create` acción, la `#update` acción no tiene una vista específica para las respuestas de Turbo Stream. Si probamos en nuestro navegador, el mensaje flash no aparece en la página.

Como se explicó en el Capítulo 4 , debido a que el formulario de edición de la cita se encuentra dentro de un Turbo Frame, este Turbo Frame está aislado del resto de la página. Al actualizar una cita, incluso si la respuesta contiene el mensaje flash, **Turbo solo extraerá y reemplazará el Turbo Frame correspondiente a la cita que se acaba de actualizar** .

Si queremos añadir mensajes flash a nuestra `#update` acción, tenemos que crear una vista Turbo Stream tal como hicimos con la `#create` acción. Lo primero que tenemos que hacer es cambiar el `#update` método en el controlador:

```
def update
  if @quote.update(quote_params)
    respond_to do |format|
      format.html { redirect_to quotes_path, notice: "Quote was successf
      format.turbo_stream { flash.now[:notice] = "Quote was successfully
    end
  else
    render :edit, status: :unprocessable_entity
  end
end
```

Ahora que nuestro controlador admite el formato Turbo Stream, tenemos que crear una vista Turbo Stream:

```
<%# app/views/quotes/update.turbo_stream.erb %>

<%= turbo_stream.replace @quote %>
<%= turbo_stream.prepend "flash", partial: "layouts/flash" %>
```

Probémoslo en nuestro navegador. Como podemos ver, la cita se reemplaza cuando se actualiza y el mensaje flash se muestra como se espera.

Mensajes flash con Hotwire en la `#destroy`acción

El código que escribiremos para la `#destroy` acción es similar al que acabamos de escribir para las acciones `#create` , y `#update` . Primero, agreguemos el mensaje flash que queremos mostrar cuando el controlador responda a una solicitud de Turbo Stream:

```
def destroy
  @quote.destroy

  respond_to do |format|
    format.html { redirect_to quotes_path, notice: "Quote was successful
    format.turbo_stream { flash.now[:notice] = "Quote was successfully d
  end
end
```

Al igual que para las acciones `#create` y `#update` , también tenemos que *anteponer* nuestros nuevos mensajes flash a la lista de mensajes flash:

```
<%# app/views/quotes/destroy.turbo_stream.erb %>

<%= turbo_stream.remove @quote %>
<%= turbo_stream.prepend "flash", partial: "layouts/flash" %>
```

¡Lo probamos en nuestro navegador y funciona como se esperaba!

Refactorizando nuestros mensajes flash con un ayudante

En las tres vistas Turbo Stream que acabamos de crear, usamos la misma línea en todas partes para representar mensajes flash:

```
<%= turbo_stream.prepend "flash", partial: "layouts/flash" %>
```

Esta línea de código ya se repite en tres de nuestras vistas y hay varias **razones para cambiarla** . Por ejemplo, podríamos decidir usar *update* en lugar de *prepend* o cambiar la ruta del `layouts/flash` parcial a `components/flash` .

Para ser inmunes a esos cambios, una estrategia es eliminar la duplicación en nuestro código. La eliminación de la duplicación se conoce comúnmente como el principio DRY (Don't Repeat Yourself). De esa manera, si `layouts/flash` más adelante se realizan cambios en `components/flash` , solo tenemos un único lugar para cambiarlo.

En nuestro ejemplo, eliminaremos la duplicación creando un asistente. Definiremos el método en `ApplicationHelper` :

```
# app/helpers/application_helper.rb

module ApplicationHelper
  def render_turbo_stream_flash_messages
    turbo_stream.prepend "flash", partial: "layouts/flash"
  end
end
```

Ahora podemos usar ese ayudante en nuestras tres vistas de Turbo Stream:

```
<%= turbo_stream.prepend "quotes", @quote %>
<%= turbo_stream.update Quote.new, "" %>
<%= render_turbo_stream_flash_messages %>
```

```
<%# app/views/quotes/update.turbo_stream.erb %>

<%= turbo_stream.replace @quote %>
<%= render_turbo_stream_flash_messages %>
```

```
<%# app/views/quotes/destroy.turbo_stream.erb %>

<%= turbo_stream.remove @quote %>
<%= render_turbo_stream_flash_messages %>
```

Con este asistente, ahora podemos cambiar de forma segura el comportamiento de nuestros mensajes flash en toda nuestra aplicación. Nuestro código ahora se ve muy limpio y ¡ya no necesitamos más mensajes flash!

Envolver

Los mensajes Flash son una herramienta importante para brindar más información a los usuarios.

Para que los mensajes flash funcionen con Hotwire en Rails 7 se requiere un poco más de configuración que en versiones anteriores de Rails, pero ahora podemos agregarles efectos interesantes fácilmente. Por ejemplo, es posible apilar mensajes flash cuando se realizan varias operaciones en un breve período de tiempo.

En el próximo capítulo veremos otra herramienta muy importante para la experiencia de usuario en nuestras aplicaciones: los estados vacíos. ¡Nos vemos allí!

← anterior Siguiente →

Recibir notificaciones cuando escriba nuevos artículos

Si te gustó este artículo y quieres estar al día con Ruby on Rails y Hotwire, ¡puedes suscribirte a mi boletín (sin spam, sin seguimiento, cancelar la suscripción en cualquier momento)!

Suscríbete al boletín

Github Gorjeo Hoja informativa

Hecho con remotamente