← Volver a la lista de capítulos

Turbo Streams y seguridad

Publicado el 10 de febrero de 2022

En este capítulo aprenderemos cómo usar Turbo Streams de forma segura y evitar enviar transmisiones a usuarios equivocados.

Entendiendo Turbo Streams y la seguridad

Antes de utilizar Turbo Streams en producción, **debemos entender cómo funciona la seguridad . Sería terrible transmitir HTML que contenga datos confidenciales a un usuario que no debería recibirlos** .

Imaginemos que nuestro editor de cotizaciones fuera una aplicación más compleja donde las cotizaciones pertenecen a empresas y las empresas tienen muchos usuarios. En ese caso, no deberíamos difundir cotizaciones a la `Quotes#index` página de un usuario que no pertenece a nuestra empresa. **Eso sería un fallo de seguridad importante** .

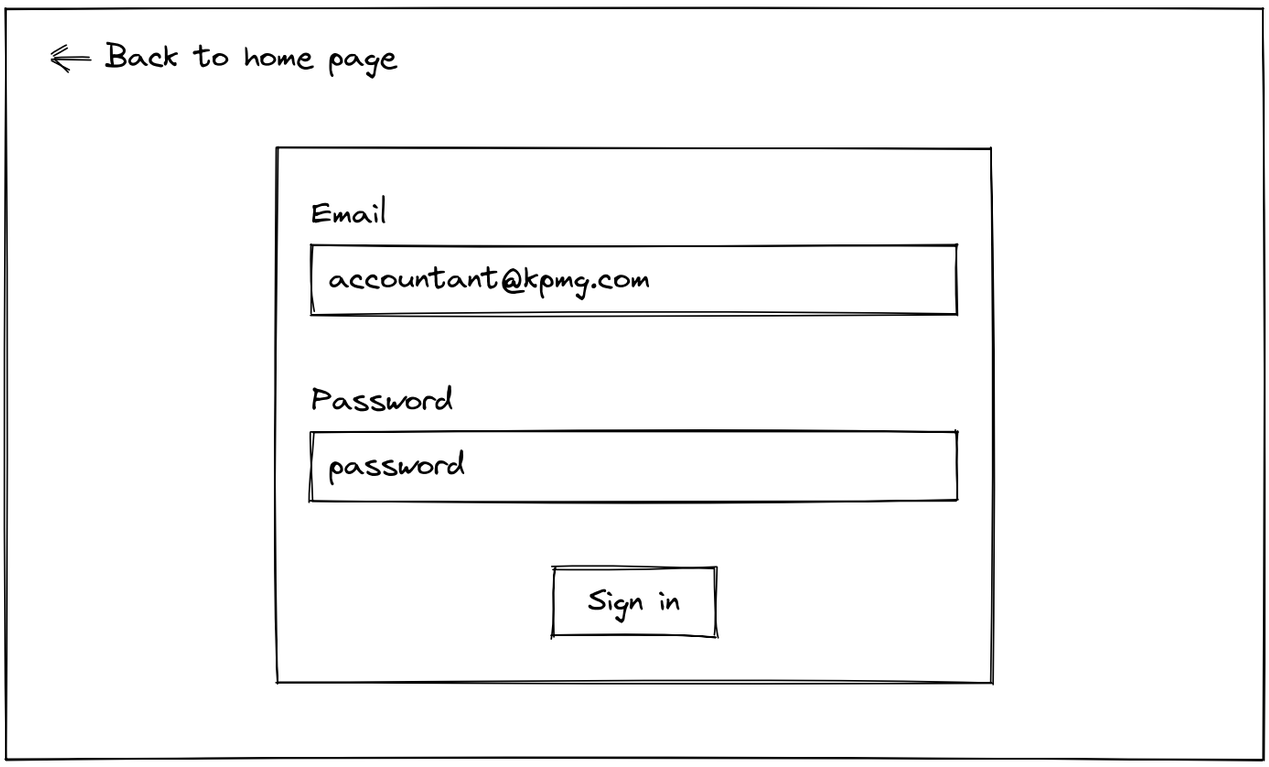
Agregaremos **usuarios** con la gema Devise y **empresas** a nuestra aplicación para simular este escenario del mundo real y comprender la seguridad de Turbo Streams. Luego, experimentaremos en el navegador para mostrar algunos de los problemas de seguridad que podrían surgir con Turbo Streams si no tenemos suficiente cuidado.

Lo que construiremos

Vamos a esbozar cómo se verá nuestra aplicación al final de este capítulo. Agregaremos una página de inicio con enlaces a la página de inicio de sesión cuando el usuario no haya iniciado sesión:

Boceto de la página de inicio cuando el usuario no ha iniciado sesión

Nuestros usuarios podrán iniciar sesión ingresando su correo electrónico y contraseña:

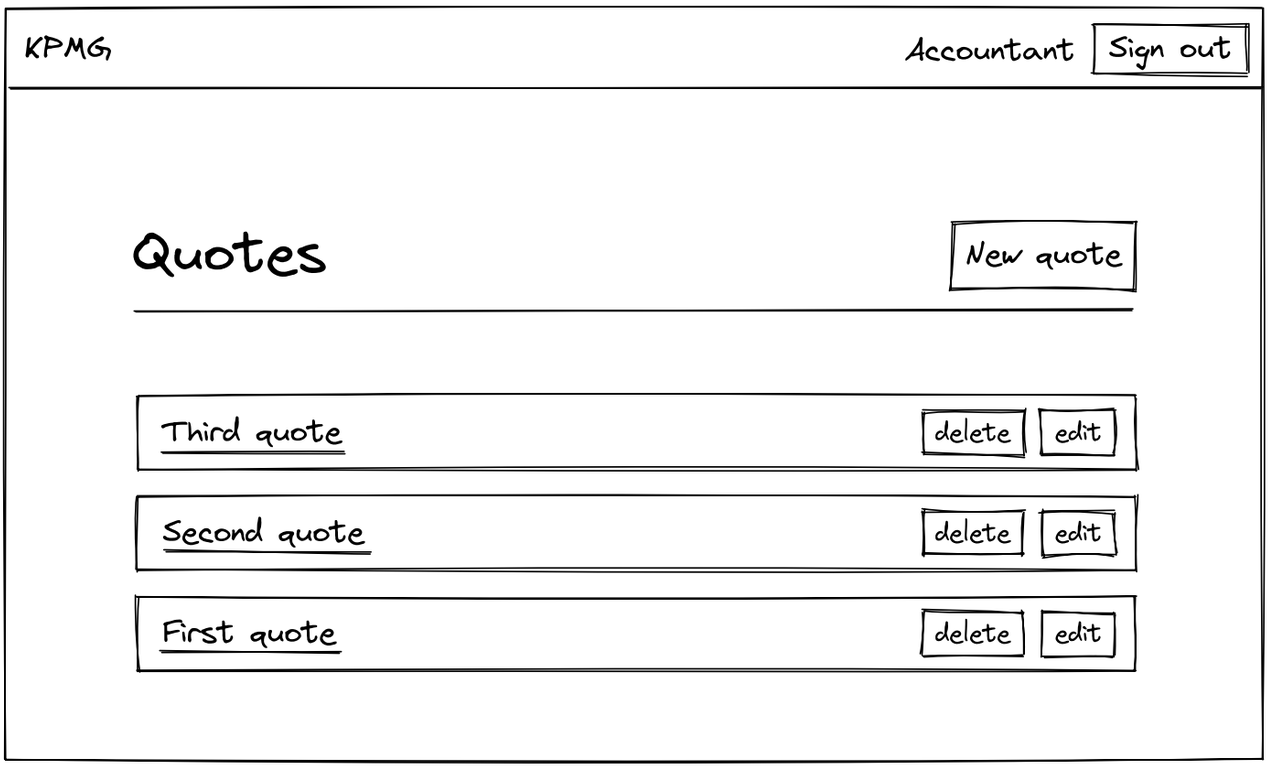


Boceto de la página de inicio de sesión

Nuestros usuarios serán redirigidos a la página de inicio cuando inicien sesión. En la barra de navegación, mostraremos el nombre de la empresa y *el nombre* del usuario en función de la dirección de correo electrónico. Podrán navegar hasta nuestro editor de cotizaciones haciendo clic en el botón "Ver cotizaciones":

Boceto de la página de inicio cuando el usuario ha iniciado sesión

Al hacer clic en el botón "Ver cotizaciones", los usuarios navegarán a la `Quotes#index` página que ahora tendrá una barra de navegación:



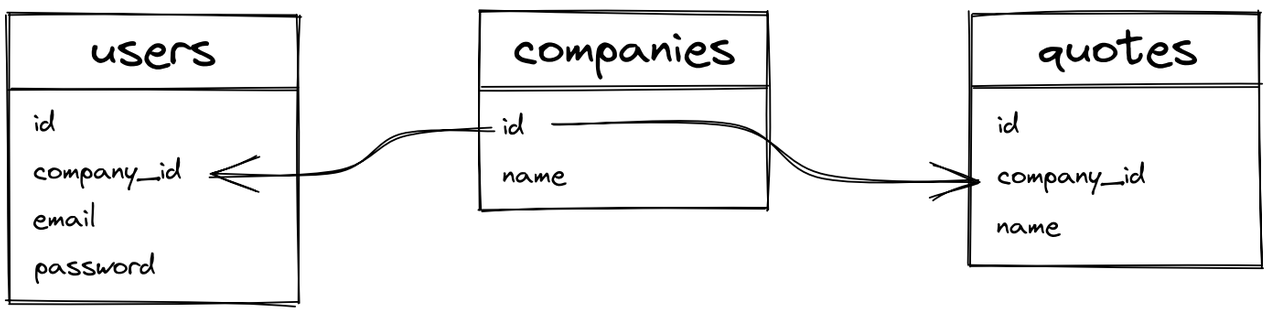
Boceto de la página de índice de citas con la barra de navegación

Con todas estas nuevas incorporaciones nuestra aplicación empezará a parecerse a una real y podremos hacer algunos experimentos en el navegador donde encontraremos algún problema de seguridad.

Los modelos `Quote`, `Company` y `User` estarán relacionados entre sí mediante las siguientes asociaciones:

- Un **usuario** pertenece a una **empresa**
- Una **cita** pertenece a una **empresa**
- Una **empresa** tiene muchos **usuarios**
- Una **empresa** tiene muchas **cotizaciones**

El esquema de base de datos que implementaremos se ilustra en el siguiente boceto:



Boceto del esquema de base de datos deseado

Introduciremos los datos que necesitamos con el `rails db:seed` comando para simular nuestro escenario del mundo real. En los partidos, necesitaremos dos empresas y tres usuarios:

- La primera empresa (**KPMG**) tendrá dos usuarios: un **contable** y un **gerente** .
- La segunda empresa (**PwC**) tendrá un solo usuario: un ***fisgón*** que nunca debería tener acceso a las cotizaciones de **KPMG .**

Creemos los modelos `Company` y `User` , y agreguemos las asociaciones necesarias para que coincidan con nuestro esquema de base de datos.

Añadiendo empresas a nuestra aplicación Rails

Vamos a crear el `Company` modelo:

```
rails generate model Company name
```

Editemos la migración porque queremos que nuestras empresas siempre tengan un nombre. Es bueno aplicar esto a nivel de base de datos agregando `null: false` una restricción de base de datos. Esto evitará que las empresas tengan un nombre vacío si, por alguna razón, se omiten las validaciones:

```
# db/migrate/XXXXXXXXXXXXXX_create_companies.rb

class CreateCompanies < ActiveRecord::Migration[7.0]
  def change
    create_table :companies do |t|
      t.string :name, null: false

      t.timestamps
    end
  end
end
```

Ahora podemos ejecutar la migración:

```
rails db:migrate
```

No olvidemos agregar la validación de presencia para el nombre en el `Company` modelo:

```
# app/models/company.rb

class Company < ApplicationRecord
  validates :name, presence: true
end
```

Solo necesitaremos las dos empresas de las que hablamos en la introducción: **KPMG** y **PwC** . Vamos a añadirlas a nuestro `companies.yml` archivo de accesorios:

```
# test/fixtures/companies.yml

kpmg:
  name: KPMG
```

```
pwc:
  name: PwC
```

Solo necesitamos esas dos empresas para que nuestro ejemplo del mundo real funcione. Para hablar de Turbo Streams y seguridad, no necesitamos crear el CRUD completo en las `CompaniesController` vistas asociadas. Continuemos agregando usuarios a nuestra aplicación.

Agregar usuarios a nuestra aplicación con Devise

Utilizaremos la muy popular y ampliamente utilizada gema Devise para agregar usuarios a nuestra aplicación y autenticarlos.

Vamos a agregarlo a nuestro `Gemfile`:

```
# Gemfile

gem "devise", "~> 4.8.1"
```

Ahora instalemos la gema:

```
bundle install
bin/rails generate devise:install
```

Ahora podemos generar el `User` modelo con los generadores Devise:

```
bin/rails generate devise User
bin/rails db:migrate
```

Ahora que tenemos nuestro `User` modelo en funcionamiento, necesitamos una vista para iniciar sesión que coincida con los bocetos que describimos en la introducción. No permitiremos que los usuarios se registren porque tendríamos que codificar la lógica para asociar nuevos usuarios a sus empresas. No necesitamos todo este trabajo para hablar sobre Turbo Streams y la seguridad, así que simplifiquemos las cosas.

Solo necesitamos usuarios en nuestras semillas y una forma de iniciar sesión. Por lo tanto, deshabilitaremos todas las funciones de Devise para nuestro `User` modelo, excepto dos de ellas:

- La función para iniciar sesión de usuarios (`:database_authenticatable`)
- La función para validar el correo electrónico y la contraseña mediante las validaciones integradas de Devise (`:validatable`)

Así es como `User` debería verse nuestro modelo:

```
# app/models/user.rb

class User < ApplicationRecord
  devise :database_authenticatable, :validatable
end
```

Por último, pero no por ello menos importante, vamos a crear los accesorios de los que hablamos en la introducción:

```
# test/fixtures/users.yml

accountant:
  email: accountant@kpmg.com
  encrypted_password: <%= Devise::Encryptor.digest(User, 'password') %>

manager:
  email: manager@kpmg.com
  encrypted_password: <%= Devise::Encryptor.digest(User, 'password') %>

eavesdropper:
  email: eavesdropper@pwc.com
  encrypted_password: <%= Devise::Encryptor.digest(User, 'password') %>
```

Devise almacena el `encrypted_password` campo en la `users` tabla de la base de datos por razones de seguridad. Si queremos que nuestros accesorios tengan la cadena `"password"` como contraseña, debemos usar el mismo método que usaría la gema Devise para cifrar la contraseña en nuestros accesorios. Por eso usamos el `Devise::Encryptor.digest` método aquí.

Con nuestro `User` modelo completo, **tenemos que agregar las asociaciones entre usuarios, empresas y cotizaciones** . Eso es lo que haremos en la siguiente sección.

Nota : Al iniciar sesión con usuarios, es posible que se produzca un error de redirección al enviar un formulario no válido. Esto se debe a que **la gema Devise aún no es compatible con Turbo (versión 4.8.1)** . La forma más sencilla de evitar este error es **desactivar Turbo en los formularios de Devise** configurando el `data-turbo` atributo en `false` en los formularios de Devise, como aprendimos en el capítulo Turbo Drive .

No lo haremos en nuestro Tutorial, pero si llevamos nuestra aplicación a producción, tendríamos que hacerlo antes de que los usuarios reales prueben nuestra aplicación.

Usuarios, empresas y asociaciones de cotizaciones

No tenemos ninguna asociación entre nuestros modelos `User` , `Company` y `Quote` . Vamos a generar dos migraciones para poder agregar esas asociaciones.

```
bin/rails generate migration add_company_reference_to_quotes company:ref
bin/rails generate migration add_company_reference_to_users  company:ref
```

La primera migración agregará la `company_id` clave externa a quotes y la segunda agregará la `company_id` clave externa a users. Gracias a los generadores de Rails, estas dos migraciones están listas para una migración:

```
bin/rails db:migrate
```

Nota : Las migraciones fallarán si tenemos algunos usuarios o citas en nuestra base de datos porque los dos archivos de migración especifican `null: false` una restricción para la `company_id` clave externa en usuarios y citas. Las citas o usuarios en nuestra base de datos actualmente tienen un espacio en blanco `company_id` que entra en conflicto con la nueva restricción.

Si nuestro proyecto fuera una aplicación real que ya se encuentra en producción, primero tendríamos que completar el `company_id` campo para todos los usuarios y cotizaciones antes de agregar la `null: false` restricción.

Como nuestra aplicación aún no se encuentra en producción, simplemente podemos eliminar la base de datos, volver a crearla y volver a ejecutar nuestra migración:

```
bin/rails db:drop db:create db:migrate
```

¡Nuestra migración ahora se ejecuta con éxito!

Ahora que nuestras migraciones pasan y nuestro esquema de base de datos está completo, agreguemos las asociaciones en los modelos `User`, `Company` y `:Quote`

```
# app/models/user.rb

class User < ApplicationRecord
  devise :database_authenticatable, :validatable

  belongs_to :company
end
```

```
# app/models/company.rb

class Company < ApplicationRecord
  has_many :users, dependent: :destroy
  has_many :quotes, dependent: :destroy

  validates :name, presence: true
end
```

```
# app/models/quote.rb

class Quote < ApplicationRecord
  belongs_to :company

  # All the previous code
end
```

Actualicemos nuestros datos de acuerdo con esto. En la introducción mencionamos que nuestro **contador** y nuestro **gerente** deben pertenecer a

KPMG . Por otro lado, nuestro **espía** debe pertenecer a **PwC** . Esto es muy fácil de hacer con los datos:

```
# test/fixtures/users.yml

accountant:
  company: kpmg
  email: accountant@kpmg.com
  encrypted_password: <%= Devise::Encryptor.digest(User, 'password') %>

manager:
  company: kpmg
  email: manager@kpmg.com
  encrypted_password: <%= Devise::Encryptor.digest(User, 'password') %>

eavesdropper:
  company: pwc
  email: eavesdropper@pwc.com
  encrypted_password: <%= Devise::Encryptor.digest(User, 'password') %>
```

Actualicemos también nuestras cotizaciones, ya que pertenecen a una empresa. Todas las cotizaciones que hemos utilizado hasta ahora pertenecerán a **KPMG** y **PwC** no tendrá ninguna cotización por defecto:

```
# test/fixtures/quotes.yml

first:
  company: kpmg
  name: First quote

second:
  company: kpmg
  name: Second quote

third:
  company: kpmg
  name: Third quote
```

Nuestros accesorios ya están listos. Vamos a cargar la base de datos ejecutándolos `rails db:seed` en la consola. ¡Todo está listo!

Agregar una página de inicio a nuestra aplicación

Ahora que tenemos usuarios en la aplicación, debemos ayudarlos a iniciar sesión fácilmente. Como se describe en los bocetos del comienzo del capítulo:

1. Los usuarios deben estar autenticados para acceder al editor de cotizaciones y solo deben ver las cotizaciones que pertenecen a su empresa.
2. Los usuarios deben poder navegar al formulario de inicio de sesión desde la página de inicio incluso cuando no estén autenticados.

Para resolver el primer punto anterior, debemos asegurarnos de que nuestros usuarios estén autenticados en todas partes de la aplicación. Vamos a hacer que esto se cumpla gracias `ApplicationController` al método Devise `authenticate_user!`:

```
# app/controllers/application_controller.rb

class ApplicationController < ActionController::Base
  before_action :authenticate_user!
end
```

Para resolver el segundo punto, necesitamos que los usuarios no autenticados puedan acceder al formulario de inicio de sesión; de lo contrario, no podrían iniciar sesión. Agreguemos una excepción a nuestra devolución de llamada:

```
# app/controllers/application_controller.rb

class ApplicationController < ActionController::Base
  before_action :authenticate_user!, unless: :devise_controller?
end
```

También necesitamos una página de inicio desde la que nuestros usuarios puedan navegar hasta el formulario de inicio de sesión. Vamos a crear una `PagesController` con una `home` acción como ruta raíz. Este controlador es público, por lo que omitiremos la necesidad de estar autenticado:

```
# app/controllers/pages_controller.rb

class PagesController < ApplicationController
  skip_before_action :authenticate_user!
```

```
  def home
  end
end
```

Ahora hagamos que la `home` acción sea la ruta raíz de nuestra aplicación:

```
# config/routes.rb

Rails.application.routes.draw do
  root to: "pages#home"

  # All the other routes
end
```

Por último, agreguemos la vista correspondiente y hagámosla lucir como nuestros bocetos:

```
<%# app/views/pages/home.html.erb %>

<main class="container">
  <h1>Quote editor</h1>
  <p>A blazing fast quote editor built with Hotwire</p>

  <% if user_signed_in? %>
    <%= link_to "View quotes", quotes_path, class: "btn btn--dark" %>
  <% else %>
    <%= link_to "Sign in", new_user_session_path, class: "btn btn--dark"
  <% end %>
</main>
```

No diseñaremos nuestra página de destino más que eso, pero dedicaremos más tiempo a la barra de navegación, ya que estará visible en todas partes de nuestra aplicación.

Primero, agreguemos el marcado para nuestra barra de navegación. Primero, usaremos *marcadores de posición* para el nombre de la empresa y el nombre del usuario actual, pero pronto lo cambiaremos. Por ahora, centrémonos en el HTML:

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

```
<header class="navbar">
  <% if user_signed_in? %>
    <div class="navbar__brand">
      Company name
    </div>
    <div class="navbar__name">
      Current user name
    </div>
    <%= button_to "Sign out",
                  destroy_user_session_path,
                  method: :delete,
                  class: "btn btn--dark" %>
  <% else %>
    <%= link_to "Sign in",
                new_user_session_path,
                class: "btn btn--dark navbar__right" %>
  <% end %>
</header>
```

Para representar nuestra barra de navegación en cada página de nuestra aplicación, podemos representarla directamente desde el diseño de la aplicación:

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

<!DOCTYPE html>
<html>
  <!-- All the <head> content -->

  <body>
    <%= render "layouts/navbar" %>
    <%= yield %>
  </body>
</html>
```

Ahora que nuestra barra de navegación se muestra en toda la aplicación, escribiremos un poco de CSS para darle estilo:

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

.navbar {
  display: flex;
  align-items: center;
```

```
  box-shadow: var(--shadow-large);
  padding: var(--space-xs) var(--space-m);
  margin-bottom: var(--space-xxl);
  background-color: (var(--color-white));

  &__brand {
    font-weight: bold;
    font-size: var(--font-size-xl);
    color: var(--color-text-header);
  }

  &__name {
    font-weight: bold;
    margin-left: auto;
    margin-right: var(--space-s);
    color: var(--color-text-header);
  }

  &__right {
    margin-left: auto;
  }
}
```

No olvidemos importar este archivo CSS a nuestro archivo de manifiesto:

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

// All the previous imports
@import "components/navbar";
```

Todo debería empezar a verse exactamente como los bocetos de la primera sección de este capítulo, excepto que no aplicaremos estilo al formulario de inicio de sesión. Probemos que todo esté conectado correctamente en el navegador. **Antes de probar, asegurémonos de que nuestras semillas estén listas** con el `bin/rails db:seed` comando.

Inicie sesión con nuestro usuario de contador. Para ello, navegue hasta la página de inicio y haga clic en el botón "Iniciar sesión". Ingrese el correo electrónico (`accountant@kpmg.com`) y la contraseña (`password`) para el usuario de contador en la página de inicio de sesión. Nuestra barra de navegación se ve bien, pero aún necesitamos cambiar el nombre de la empresa y el nombre del usuario actual de forma dinámica.

Empecemos por lo más fácil: el nombre del archivo `current_user`. Lo usaremos para ver qué usuario del dispositivo está conectado para que nos resulte más fácil el desarrollo. Agreguemos un `#name` método a nuestro `User` modelo para adivinar el nombre del usuario a partir de la dirección de correo electrónico:

```
# app/models/user.rb

class User < ApplicationRecord
  devise :database_authenticatable, :validatable

  belongs_to :company

  def name
    email.split("@").first.capitalize
  end
end
```

También podemos agregar una prueba a nuestro método para garantizar que se comporte como esperamos:

```
# test/models/user_test.rb

require "test_helper"

class UserTest < ActiveSupport::TestCase
  test "name" do
    assert_equal "Accountant", users(:accountant).name
  end
end
```

Ahora que estamos seguros de que nuestro método muestra el resultado correcto, podemos actualizar el HTML en nuestra barra de navegación:

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

<header class="navbar">
  <% if user_signed_in? %>
    <div class="navbar__brand">
      Company name
    </div>
    <div class="navbar__name">
```

```
      <%= current_user.name %>
    </div>
    <%= button_to "Sign out",
                  destroy_user_session_path,
                  method: :delete,
                  class: "btn btn--dark" %>
  <% else %>
    <%= link_to "Sign in",
                new_user_session_path,
                class: "btn btn--dark navbar__right" %>
  <% end %>
</header>
```

También queremos agregar el nombre de la empresa a la barra de navegación. Como tenemos un `current_user` , sería bueno tener un , `current_company` ya que nuestros usuarios siempre pertenecen a una empresa.

Agreguemos esta lógica al método, `ApplicationController` ya que lo usaremos `current_company` más adelante en nuestros controladores y vistas. Para usar el `current_company` método en nuestras vistas, debemos convertirlo en un ayudante. Podemos hacerlo gracias al `helper_method` método:

```
# app/controllers/application_controller.rb

class ApplicationController < ActionController::Base
  before_action :authenticate_user!, unless: :devise_controller?

  private

  def current_company
    @current_company ||= current_user.company if user_signed_in?
  end
  helper_method :current_company
end
```

Ahora que tenemos nuestro `current_company` ayudante, podemos usarlo en nuestras vistas y especialmente en nuestra barra de navegación:

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

<header class="navbar">
  <% if user_signed_in? %>
```

```
    <div class="navbar__brand">
      <%= current_company.name %>
    </div>
    <div class="navbar__name">
      <%= current_user.name %>
    </div>
    <%= button_to "Sign out",
                  destroy_user_session_path,
                  method: :delete,
                  class: "btn btn--dark" %>
  <% else %>
    <%= link_to "Sign in",
                new_user_session_path,
                class: "btn btn--dark navbar__right" %>
  <% end %>
</header>
```

Eso fue mucho código de configuración, pero casi lo logramos. Solo necesitamos arreglar nuestras pruebas porque requeríamos que los usuarios iniciaran sesión en el editor de citas y agregamos algunas asociaciones. Después de eso, estaremos listos para analizar Turbo Streams y la seguridad mediante la experimentación con el navegador.

Arreglando nuestras pruebas

¡Nuestras pruebas del sistema no funcionan en este momento! Ejecutémoslas con el `bin/rails test:system` comando.

El primer error que podemos notar es que nuestros usuarios ahora necesitan iniciar sesión antes de manipular las comillas. Para iniciar sesión de los usuarios en las pruebas del sistema, nos basaremos en los ayudantes de la `Warden` gema para evitar codificarlos nosotros mismos. Devise está construido sobre `Warden` y el `Warden::Test::Helpers` módulo contiene ayudantes que nos ayudarán a iniciar sesión de los usuarios en nuestras pruebas gracias al `login_as` método.

Para utilizarlos, primero incluyámoslos `Warden::Test::Helpers` en nuestra `ApplicationSystemTestCase` clase.

```
# test/application_system_test_case.rb
```

```
require "test_helper"

class ApplicationSystemTestCase < ActionDispatch::SystemTestCase
  include Warden::Test::Helpers

  driven_by :selenium, using: :headless_chrome, screen_size: [1400, 1400
end
```

Ahora podemos usar los ayudantes del `Warden::Test::Helpers` módulo en nuestras pruebas del sistema, ya que todas nuestras clases de pruebas del sistema heredan de `ApplicationSystemTestCase`. Lo que tenemos que hacer es iniciar sesión con nuestro usuario contable antes de cada ejecución de prueba en la prueba del sistema de cotizaciones. Para ello, usaremos el `login_as` ayudante del `Warden::Test::Helpers` bloque de configuración:

```
# test/system/quotes_test.rb

require "application_system_test_case"

class QuotesTest < ApplicationSystemTestCase
  setup do
    login_as users(:accountant)
    @quote = Quote.ordered.first
  end

  # All the previous code
end
```

Volvamos a ejecutar las pruebas del sistema con el `bin/rails test:system` comando. Algunas pruebas siguen fallando porque necesitamos que las cotizaciones estén asociadas con una empresa. Actualicemos `QuotesController` para usar la asociación con el `Company` modelo.

El `QuotesController#index` método solo debe mostrar las cotizaciones que pertenecen a la empresa del usuario actual. Para ello, usemos las asociaciones:

```
# app/controllers/quotes_controller.rb

def index
  @quotes = current_company.quotes.ordered
end
```

Además, al crear la cotización, debemos asegurarnos de que la cotización esté asociada con la empresa del usuario actual:

```
# app/controllers/quotes_controller.rb

def create
  # Only this first line changes to make sure the association is created
  @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
```

Para todas las demás acciones, debemos asegurarnos de que la cotización que manipulamos tenga como alcance el `current_company` fin de evitar que los usuarios manipulen cotizaciones que no pertenecen a su empresa, evitando así problemas de seguridad:

```
class QuotesController < ApplicationController
  before_action :set_quote, only: [:show, :edit, :update, :destroy]
  # All the previous code

  private

  def set_quote
    # We must use current_company.quotes here instead of Quote
    # for security reasons
    @quote = current_company.quotes.find(params[:id])
  end

  # All the previous code
end
```

Ejecutemos nuestras pruebas del sistema una vez más con el `bin/rails test:system` comando. Todas deberían estar en verde. Ejecutemos *todas*

nuestras pruebas con el `bin/rails test:all` comando. ¡Todas pasan! Ahora estamos listos para hablar sobre Turbo Streams y la seguridad.

Seguridad y Turbo Streams

Esta fue una configuración muy larga, pero estamos listos para discutir cómo funciona la seguridad con Turbo Streams.

Comencemos mostrando un gran problema que tenemos en nuestra aplicación. Para ello, abramos dos ventanas del navegador **una al lado de la otra para poder ver las actualizaciones en tiempo real** :

- Una ventana del navegador en modo predeterminado
- Una ventana del navegador en navegación privada

En la ventana del modo predeterminado, iniciemos sesión con nuestro **contador** definido en el `users.yml` archivo de accesorios (el correo electrónico es `accountant@kpmg.com` y la contraseña es `password`).

En la ventana de navegación privada, iniciemos sesión con nuestro **dispositivo espía** (el correo electrónico es `eavesdropper@pwc.com` y la contraseña también es `password`).

Ahora, hagamos que ambos usuarios naveguen a la `Quotes#index` página. Con la cuenta del **contador** , creemos una cotización llamada "Cotización secreta". La "Cotización secreta" aparece en la ventana privada en la que el **espía** ha iniciado sesión.

Aquí hay un problema de seguridad crítico : la cotización que creó el **contador** se transmitió al **usuario espía** . Esos dos usuarios **pertenecen a empresas diferentes** y **nunca deberían tener acceso a las cotizaciones de la otra empresa** .

Si actualizamos la página de navegación privada con nuestra cuenta **de espía** , la "cita secreta" desaparece. Esto se debe a que el HTML que contiene la "cita secreta" se transmitió al usuario **espía** incluso si delimitamos correctamente la lista de citas en `current_company` la `QuotesController#index` acción:

```
# app/controllers/quotes_controller.rb

def index
  @quotes = current_company.quotes.ordered
end
```

Este es un problema de Turbo Streams. Antes de utilizar Turbo Streams en producción, primero debemos comprender cómo funciona la seguridad a un alto nivel. Analicemos por qué tenemos esta brecha de seguridad y cómo solucionarla.

Seguridad de Turbo Stream en profundidad

Es fundamental comprender cómo funciona la seguridad antes de utilizarla `turbo-rails` en producción. **La seguridad es un tema complejo, pero la solución es sencilla en este caso** .

Comencemos nuestro viaje notando algo. Abramos nuestras herramientas de desarrollo e inspeccionemos el DOM con la sesión de nuestro **contable y la sesión de nuestro espía** (la ventana de navegación predeterminada y la privada). En la `Quotes#index` página de ambas ventanas del navegador, deberíamos encontrar una `<turbo-cable-stream-source>` etiqueta. Fue generada por el `turbo_stream_from` ayudante que llamamos en la `Quotes#index` página:

```
<%# app/views/quotes/index.html.erb %>

<%# This line generate the <turbo-cable-stream-source> tag %>
<%= turbo_stream_from "quotes" %>

<%# All previous content %>
```

Copiemos `turbo-cable-stream-source` tanto de la sesión del contador como de la sesión del espía.

```
<!-- Accountant's session -->
<turbo-cable-stream-source
  channel="Turbo::StreamsChannel"
  signed-stream-name="InF1b3RlcyI=--eba9a5055d229db025dd2ed20d069d87c36a
>
```

```
</turbo-cable-stream-source>

<!-- Eavesdropper's session -->
<turbo-cable-stream-source
  channel="Turbo::StreamsChannel"
  signed-stream-name="InF1b3RlcyI=--eba9a5055d229db025dd2ed20d069d87c36a
>
</turbo-cable-stream-source>
```

¿Observa cómo **los dos nombres de flujo firmados son iguales** ? Empezamos a entender por qué ambos usuarios reciben las transmisiones de creación, actualización y eliminación de cotizaciones.

Nota : El nombre del atributo para la transmisión es `signed-stream-name` en lugar de simplemente `stream-name` . El `turbo_stream_from` asistente firma automáticamente el nombre de la transmisión para evitar que los usuarios manipulen su valor y obtengan acceso a transmisiones privadas. ¡La gema turbo-rails se encarga de este problema de seguridad por nosotros!

En los bocetos utilizo una versión simplificada `stream-name` de nuestro modelo mental. En la realidad, el `stream-name` signo se firma automáticamente y no tenemos que pensar en ello.

Nota : la suya `signed-stream-name` no será igual a la mía porque no estará firmada con la misma clave privada. Rails genera una clave privada diferente para cada nueva aplicación Ruby on Rails. Lo importante aquí es que los `signed-stream-name` atributos sean los mismos tanto para el **contador** como para el **espía** .

Ambos usuarios se han suscrito al atributo `Turbo::StreamsChannel` de agradecimiento `channel` . Todas las comunicaciones entre el publicador (el servidor) y el suscriptor (el cliente) de Turbo Streams se realizarán a través del atributo `Turbo::StreamsChannel` .

Sin embargo, ambos usuarios tienen el mismo `signed-stream-name` . En `ActionCable` , la función de un stream es enviar transmisiones a los suscriptores. Por lo tanto, si tenemos el mismo `signed-stream-name` en la

página del **contador** y en la del **espía** `Quotes#index` , ambos recibirán las mismas transmisiones. ¡Es por eso que cuando se crea una cotización, el HTML correspondiente se transmite tanto al **contador** como al **espía** !

Esbocemos lo que sucede. Cuando el **contador** crea la "cita secreta", esta se transmite a la secuencia denominada "citas". Tanto el **contador** como el **espía** reciben las transmisiones mientras se transmiten desde la secuencia "citas".

El contador está transmitiendo desde el flujo de cotizaciones.

El espía también está transmitiendo desde el flujo de cotizaciones.

Para resolver nuestro problema de seguridad, esos `signed-stream-name` atributos deben ser ***diferentes*** .

Ahora que entendemos mejor el problema, veremos cómo solucionarlo en la siguiente sección.

Solución del problema de seguridad de Turbo Streams

En el capítulo anterior , instruimos al modelo de cotizaciones para transmitir creaciones, actualizaciones y eliminaciones a los usuarios:

```
# app/models/quote.rb

class Quote < ApplicationRecord
  # All the previous code

  broadcasts_to ->(quote) { "quotes" }, inserts_by: :prepend
end
```

La parte que nos interesa aquí es la lambda que se pasa como primer argumento al `broadcasts_to` método:

```
->(quote) { "quotes" }
```

Lo que esto hace en segundo plano es que `Turbo::StreamsChannel` transmitirá las creaciones, actualizaciones y eliminaciones de citas a través de la `"quotes"` transmisión, ya que esta lambda siempre devolverá la cadena `"quotes"` . La transmisión se recibe en la `Quotes#index` página porque los usuarios están suscritos a la transmisión gracias a la siguiente línea:

```
<%# app/views/quotes/index.html.erb %>

<%# This line generate the <turbo-cable-stream-source> tag %>
<%= turbo_stream_from "quotes" %>

<%# All previous content %>
```

Queremos tener **el mismo** `signed-stream-name` para el **contador** y el **gerente** y **uno diferente** para el **espía** . Para hacer esto, tenemos que cambiar el nombre del flujo donde se transmitirán las cotizaciones HTML. Hacer esto `turbo-rails` es muy simple:

```
class Quote < ApplicationRecord
  # All the previous code

  broadcasts_to ->(quote) { [quote.company, "quotes"] }, inserts_by: :pr
end
```

En esencia, el nombre del flujo firmado se genera a partir de la matriz devuelta por la lambda que es el primer argumento del `broadcasts_to` método. Las reglas para transmisiones seguras son las siguientes:

1. **Los usuarios que comparten transmisiones deben hacer que lambda devuelva una matriz con los mismos valores .**
2. **Los usuarios que no deberían compartir transmisiones deberían hacer que lambda devuelva una matriz con diferentes valores .**

En nuestro ejemplo, la empresa de cotización es la *misma* para los usuarios de accesorios **contable** y **gerente** . Para ambos, la lambda devuelve una matriz con los mismos valores, de modo que puedan compartir la creación, actualización y eliminación de cotizaciones.

La compañía de la cotización es *diferente* para el **espía** , por lo que no podrá recibir las transmisiones.

Para que nuestra función vuelva a funcionar, debemos actualizar la `turbo_stream_from` página `Quotes#index` , ya que acabamos de cambiar el nombre del flujo en nuestra `Quotes#index` vista para que coincida con los valores dentro de lambda:

```
<%# app/views/quotes/index.html.erb %>

<%= turbo_stream_from current_company, "quotes" %>
```

Experimentemos nuevamente en el navegador. Abramos la `Quotes#index` página en una ventana predeterminada del navegador con la sesión de nuestro **contador** `Quotes#index` y la página en una ventana de navegación privada con la sesión de nuestro **espía** . Creemos una nueva cita con el nombre "Cita secreta" con la sesión de nuestro **contador** . ¡Esta vez, no se agregó a la lista de citas del **espía** !

Al inspeccionar el DOM, podemos ver que los `signed-stream-name` valores ya no son los mismos en la página del **contador** y en la del **espía** `Quote#index` .

```
<!-- Accountant's session -->
<turbo-cable-stream-source
  channel="Turbo::StreamsChannel"
  signed-stream-name="IloybGtPaTh2YUc5MGQybHlaUzFqYjNWeWMyVXZRMjl0Y0dGdW
>
</turbo-cable-stream-source>

<!-- Eavesdropper's session -->
<turbo-cable-stream-source
  channel="Turbo::StreamsChannel"
  signed-stream-name="IloybGtPaTh2YUc5MGQybHlaUzFqYjNWeWMyVXZRMjl0Y0dGdW
>
</turbo-cable-stream-source>
```

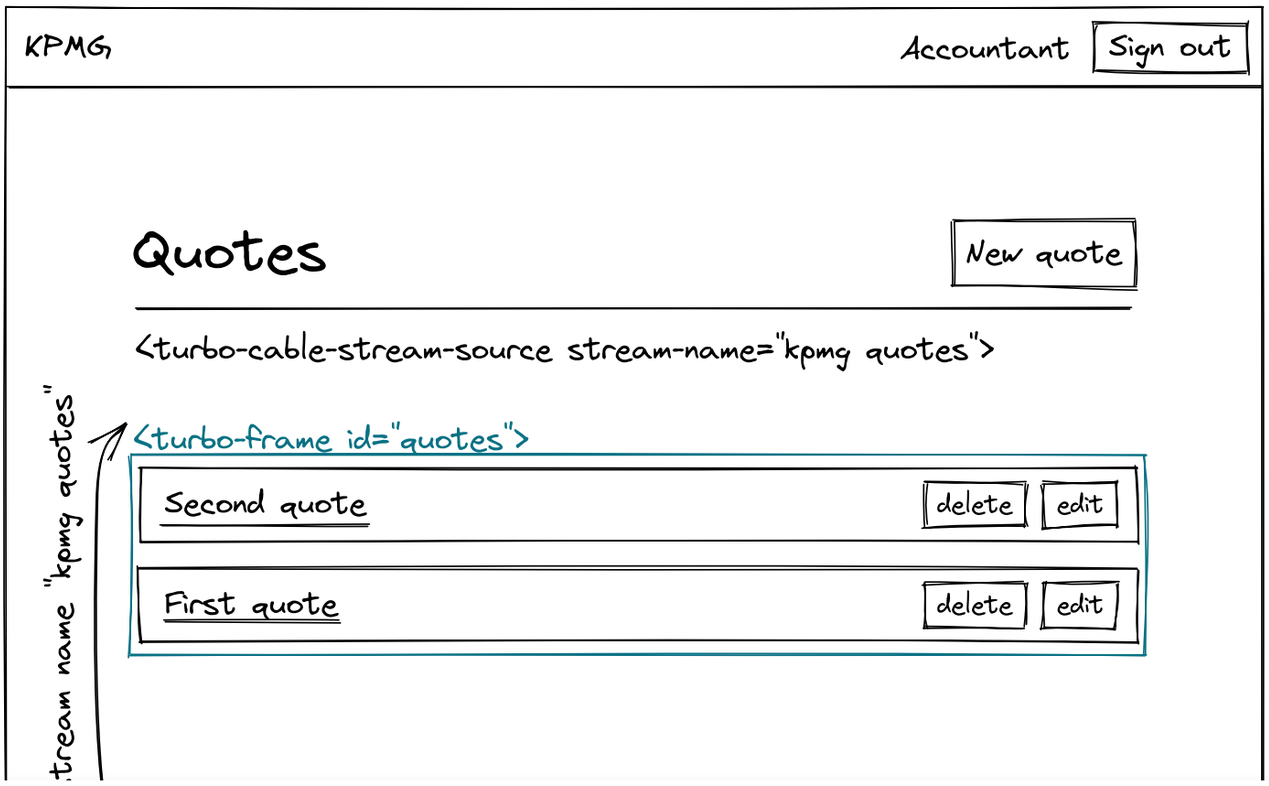
Ahora, hagamos la prueba con las cuentas del **contador** y del **gerente** . Cierremos la sesión **del espía** en la ventana privada e iniciemos la sesión del **gerente** . Con ambos usuarios en la `Quotes#index` página, creemos una cotización de "Cotización compartida" con la cuenta del **contador** . La cotización creada debería anteponerse en tiempo real a la lista de cotizaciones en la ventana privada del **gerente . ¡Ahora todo funciona como se esperaba!**

Al inspeccionar el DOM, podemos ver que los `signed-stream-name` valores son los mismos en la página del **contador** y en la página del **gerente** `Quote#index` .

```
<!-- Accountant's session -->
<turbo-cable-stream-source
  channel="Turbo::StreamsChannel"
  signed-stream-name="IloybGtPaTh2YUc5MGQybHlaUzFqYjNWeWMyVXZRMjl0Y0dGdW
>
</turbo-cable-stream-source>

<!-- Manager's session -->
<turbo-cable-stream-source
  channel="Turbo::StreamsChannel"
  signed-stream-name="IloybGtPaTh2YUc5MGQybHlaUzFqYjNWeWMyVXZRMjl0Y0dGdW
>
</turbo-cable-stream-source>
```

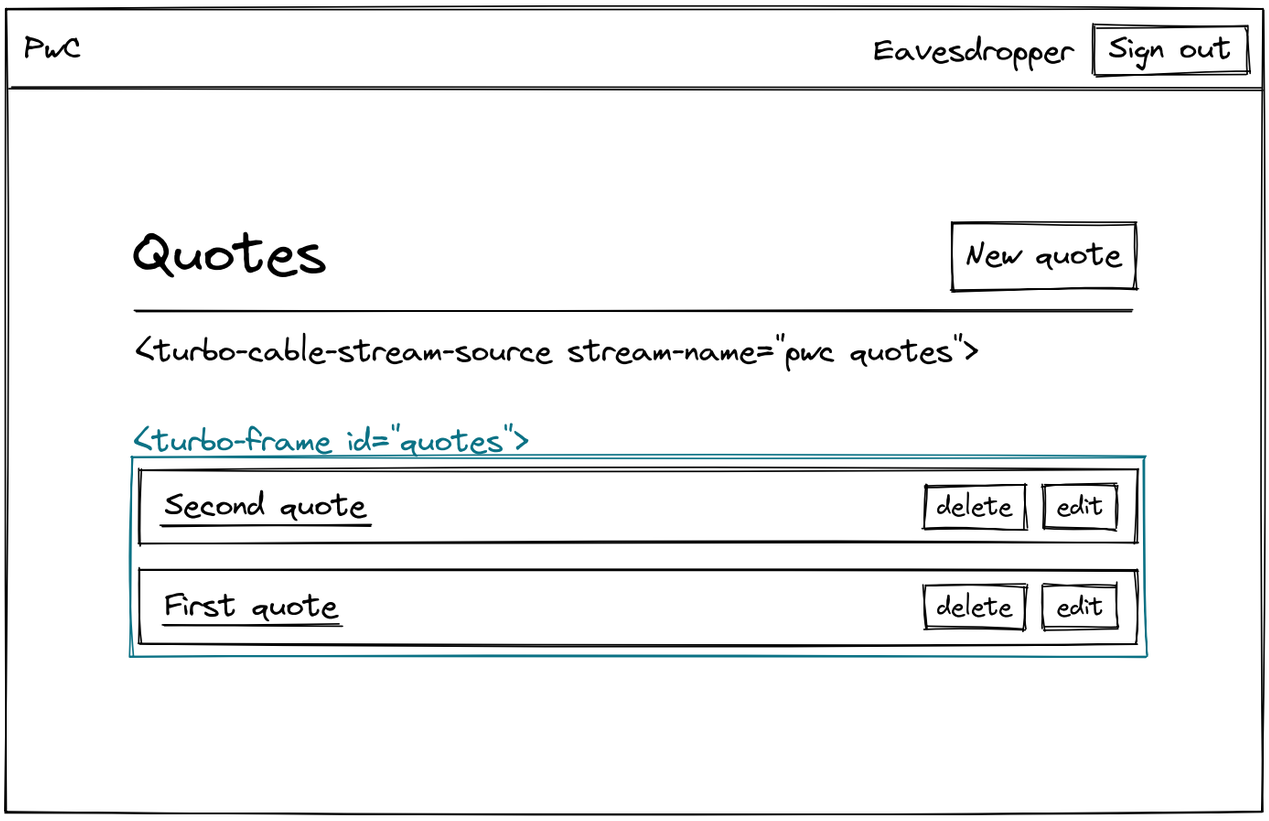
Esbocemos lo que sucede aquí. Tanto el **contador** como el **gerente** reciben la transmisión porque comparten el mismo nombre de transmisión:



El contable recibe la transmisión

El gerente recibe la transmisión

El **espía** no recibe la transmisión porque el `<turbo-cable-stream-source>` elemento no tiene el mismo `stream-name` atributo:



El espía no recibe la transmisión

Al utilizar Turbo Streams en producción, debemos asegurarnos de que el `broadcasts_to` método en el modelo y el `turbo_stream_from` método en la vista estén configurados adecuadamente para evitar problemas de seguridad.

Envolver

Los Turbo Streams son una herramienta fantástica, pero debemos tener cuidado a quién transmitimos información si no queremos crear problemas de seguridad importantes.

En este capítulo, vimos que esto debe configurarse en las devoluciones de llamadas del modelo que activan las transmisiones. La lambda en el primer argumento del `broadcasts_to` método debe devolver el **mismo valor** para los usuarios que comparten transmisiones y **uno diferente** para los usuarios que no deben compartir las transmisiones.

← 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