← Volver a la lista de capítulos

Marcos Turbo anidados

Publicado el 14 de marzo de 2022

En este capítulo, crearemos nuestro último controlador CRUD para artículos de línea. Como los artículos de línea están anidados en las fechas de los artículos de línea, tendremos algunos desafíos interesantes que resolver con Turbo Frames.

Lo que construiremos en este capítulo

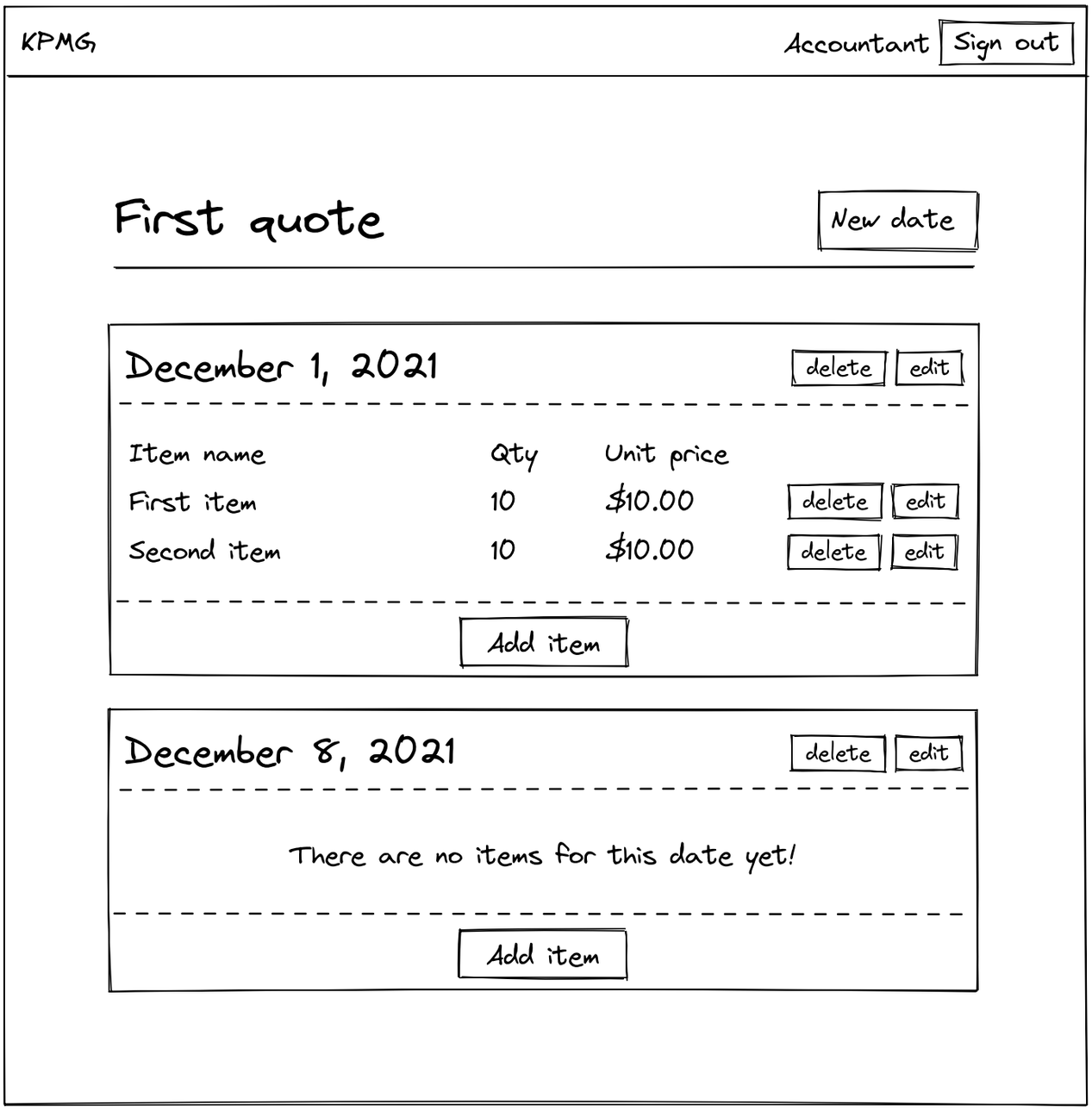
En este capítulo, casi finalizaremos nuestro editor de cotizaciones agregando **elementos de línea** a las **fechas de elementos de línea** que creamos en el capítulo anterior. Esos elementos de línea tendrán un `name` , un `description` , un `unit_price` , un `quantity` .

Si bien este capítulo trata sobre otro controlador CRUD, traerá consigo algunos desafíos interesantes, ya que tendremos muchos Turbo Frames anidados. También analizaremos cómo preservar el *estado* de nuestra `Quotes#show` página al realizar operaciones CRUD tanto en los modelos `LineItemDate` como en los `LineItem` modelos.

Antes de comenzar a codificar, veamos cómo funcionará nuestro editor de cotizaciones final . Creemos una cotización y luego naveguemos a la `Quotes#show` página. Luego, creemos algunas **fechas de partidas** y algunas **partidas** para tener una idea clara de cómo debería verse nuestro producto final.

Una vez que comprendamos bien cómo se comporta el editor de citas final , comencemos, como siempre, por hacer que nuestro controlador CRUD en el `LineItem` modelo funcione **sin Turbo Frames ni Turbo Streams** . Agregaremos funciones de Turbo Rails más adelante, una vez que nuestro controlador funcione correctamente.

Primero, hagamos algunos bocetos de cómo se comportará nuestra aplicación **sin Turbo Frames y Turbo Streams** . Cuando visitamos la `Quotes#show` página, ahora tenemos **fechas de partidas de línea** en nuestra cotización. Como cada **fecha de partida de línea** podrá tener varias **partidas de línea** , cada tarjeta **de fecha de partida de línea** tendrá un enlace "Agregar partida" para agregar **partidas de línea** para esta **fecha de partida de línea** específica :

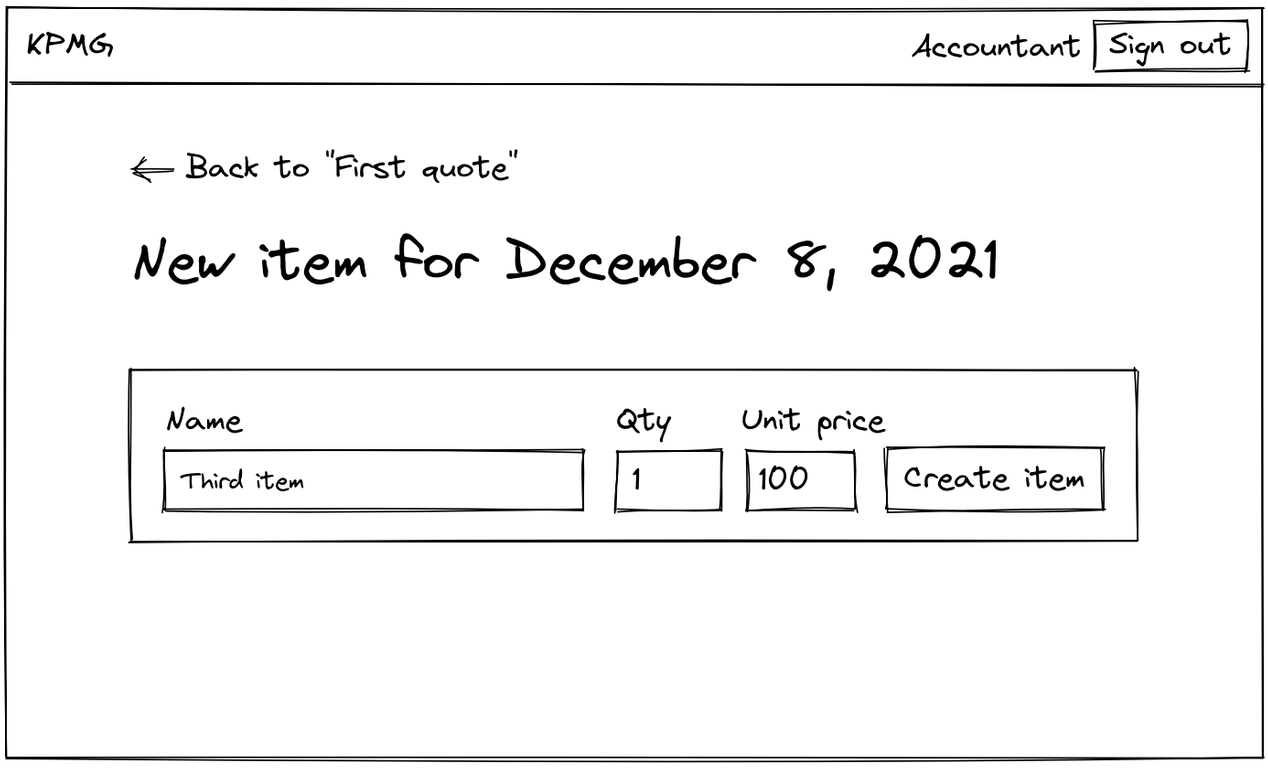


Boceto de la página de citas#mostrar con algunas fechas y algunos elementos

En la `Quotes#show` página, deberíamos poder agregar **partidas** a cualquiera de las **fechas de partidas** presentes en la cotización. Como estamos creando el CRUD **sin Turbo** , al hacer clic en el enlace "Agregar partida" para una **fecha de**

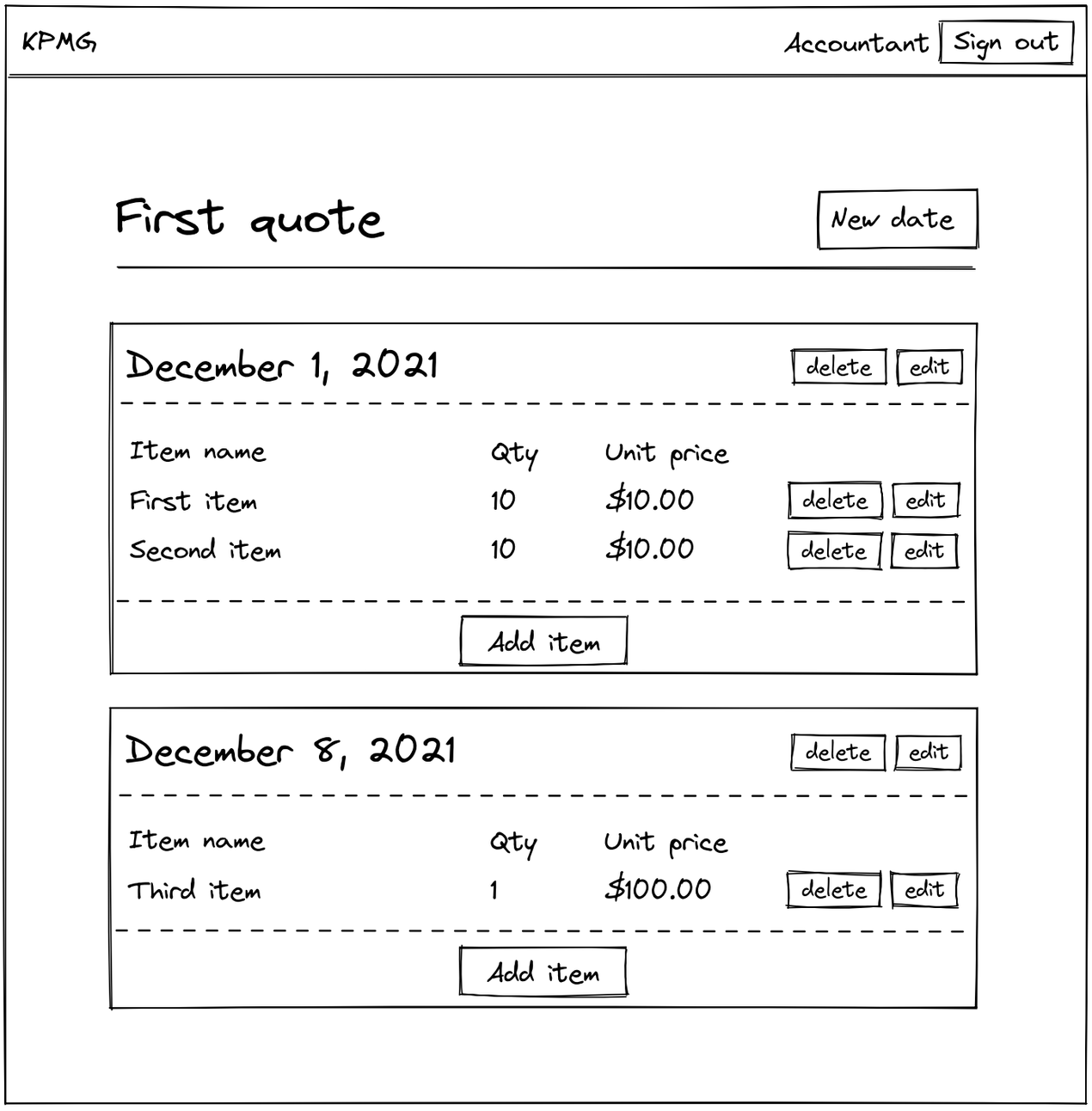
partida, accederemos a la `LineItems#new` página donde podemos agregar una **partida** para esta **fecha de partida** específica .

Si hacemos clic en el enlace "Agregar artículo" para la **fecha del segundo artículo** , aquí hay un boceto de la página que podemos esperar:



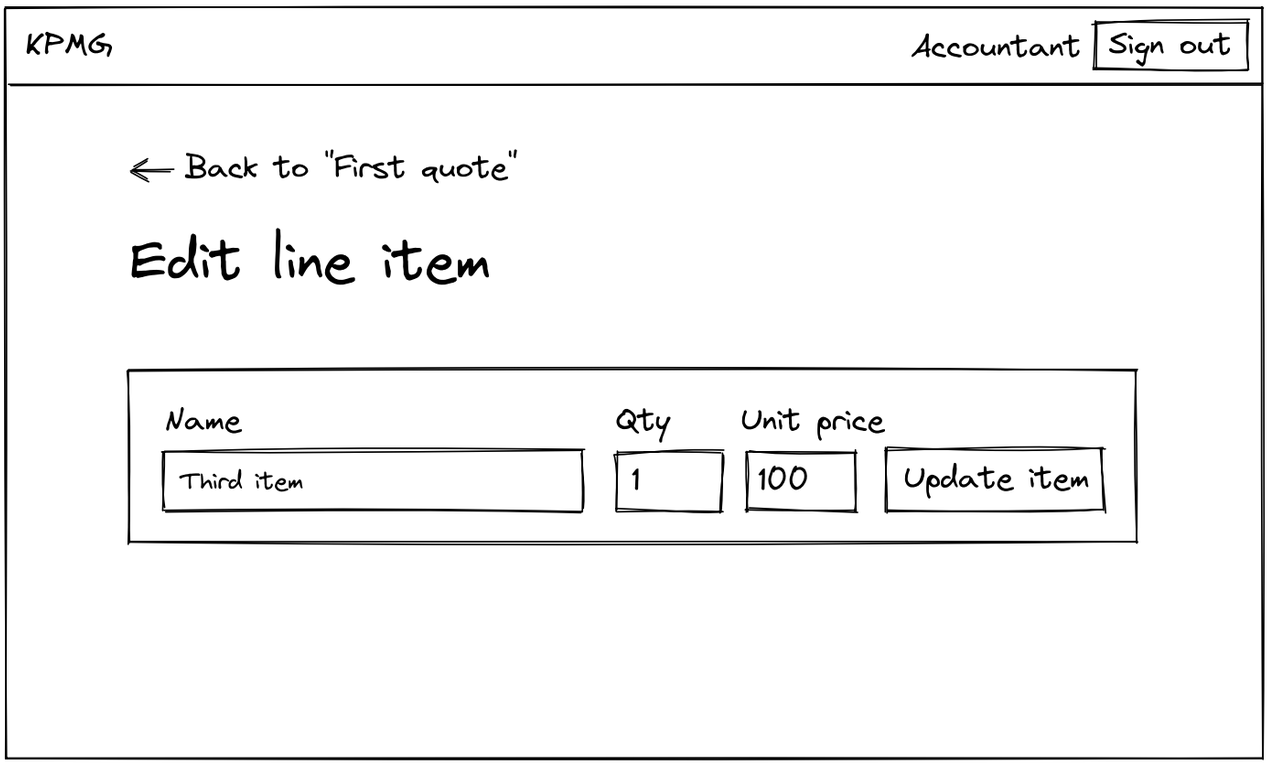
Boceto de la página LineItems#new para la segunda fecha

Si enviamos un formulario válido, seremos redirigidos a la `Quotes#show` página con la nueva **línea** agregada:



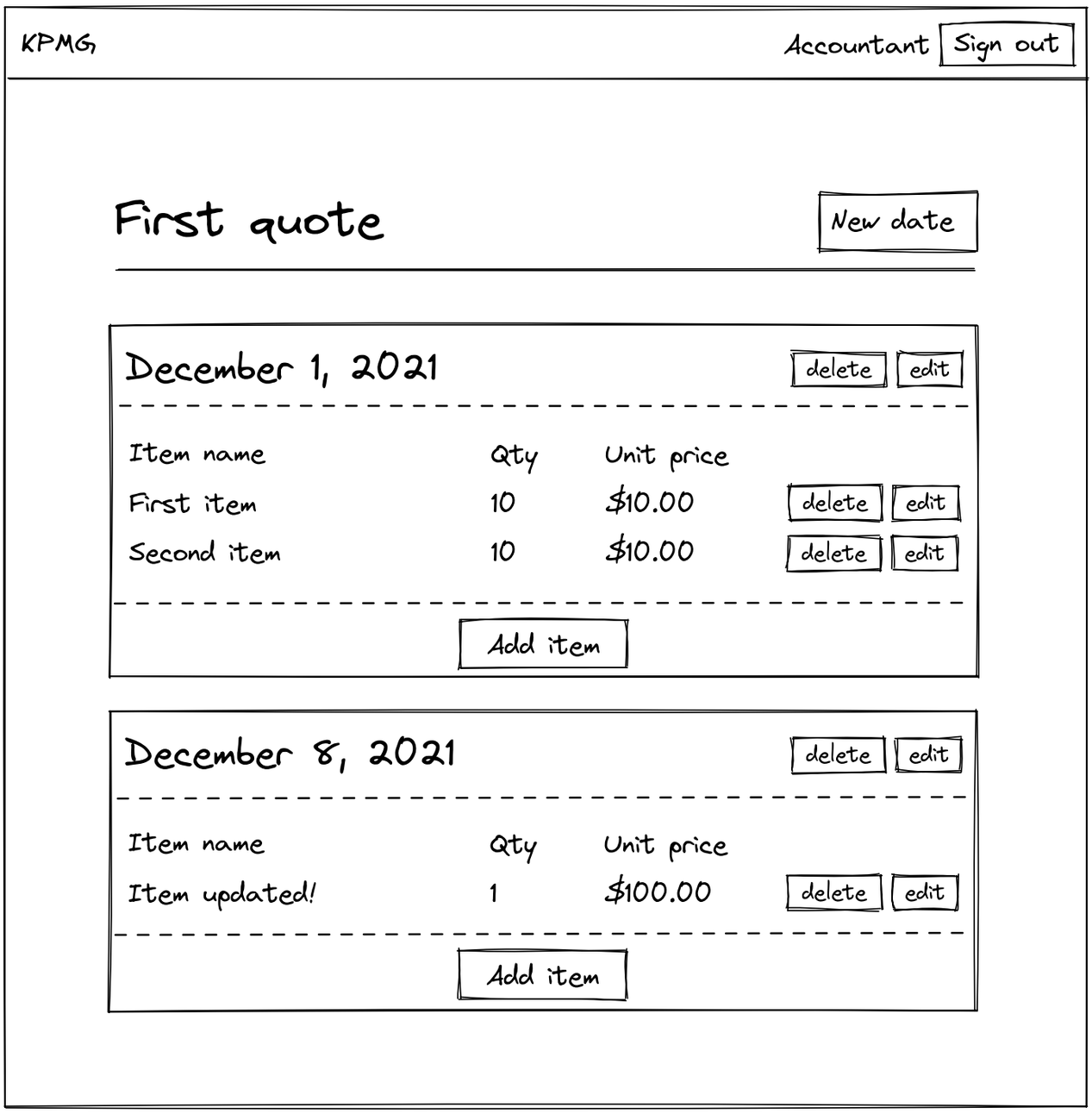
Boceto de la página Quotes#show con el elemento de línea creado agregado a la lista

Si decidimos actualizar la **línea de pedido** que acabamos de crear, podemos hacer clic en el enlace "Editar" de esta **línea de pedido** para navegar a la `LineItems#edit` página:



Boceto de la página de edición de artículos de línea

Si enviamos un formulario válido, seremos redirigidos a la `Quotes#show` página con el **elemento de línea** actualizado:



Boceto de la página de cotización#show con la línea de artículo actualizada

Por último, pero no por ello menos importante, podemos eliminar una **línea de pedido** haciendo clic en el enlace "Eliminar" correspondiente a esa **línea de pedido** . De ese modo, la **línea de pedido** se eliminará de la lista.

Ahora que los requisitos están claros, ¡es hora de comenzar a codificar!

Creando el modelo

Comencemos creando un modelo llamado `LineItem`. Este modelo tendrá cinco campos:

- una referencia a la **fecha de la línea de artículo** a la que pertenece
- Un nombre
- una descripción opcional
- un precio unitario

- una cantidad

Agregamos una referencia al `LineItemDate` modelo porque cada **ítem de línea** pertenece a una **fecha de ítem de línea** y cada **fecha de ítem de línea** tiene muchos **ítems de línea** . Generemos la migración:

```
bin/rails generate model LineItem \
  line_item_date:references \
  name:string \
  description:text \
  quantity:integer \
  unit_price:decimal{10-2}
```

Antes de ejecutar el `rails db:migrate` comando, debemos agregar restricciones a los campos `name` , `quantity` y , `unit_price` ya que queremos que siempre estén presentes en cada registro. Podemos hacer cumplir esta validación a nivel de base de datos gracias a la `null: false` restricción.

La migración final debería verse así:

```
# db/migrate/XXXXXXXXXXXXXX_create_line_items.rb

class CreateLineItems < ActiveRecord::Migration[7.0]
  def change
    create_table :line_items do |t|
      t.references :line_item_date, null: false, foreign_key: true
      t.string :name, null: false
      t.text :description
      t.integer :quantity, null: false
      t.decimal :unit_price, precision: 10, scale: 2, null: false

      t.timestamps
    end
  end
end
```

Ahora que nuestra migración está lista, podemos ejecutarla:

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

Agreguemos las asociaciones y las validaciones correspondientes en el `LineItem` modelo:

```
# app/models/line_item.rb

class LineItem < ApplicationRecord
  belongs_to :line_item_date

  validates :name, presence: true
  validates :quantity, presence: true, numericality: { only_integer: tru
  validates :unit_price, presence: true, numericality: { greater_than: 0

  delegate :quote, to: :line_item_date
end
```

Las validaciones del modelo refuerzan que:

- Los campos `name` , `quantity` , y `unit_price` deben estar presentes
- Los campos `unit_price` y `quantity` deben ser mayores que cero
- El `quantity` debe ser un número entero

También delegamos el `quote` método al `LineItem#line_item_date` método. De esta manera, las dos líneas siguientes son equivalentes:

```
line_item.line_item_date.quote
line_item.quote
```

Ahora que nuestro `LineItem` modelo está completo, agreguemos la `has_many` asociación en el `LineItemDate` modelo:

```
# app/models/line_item_date.rb

class LineItemDate < ApplicationRecord
  has_many :line_items, dependent: :destroy

  # All the previous code...
end
```

¡Nuestra capa de modelo ya está completa! Ahora trabajaremos en las rutas.

Agregar rutas para artículos de línea

Queremos realizar las siete acciones CRUD en el `LineItem` modelo excepto dos de ellas:

- No necesitaremos la `LineItem#index` acción ya que todos los elementos de línea ya estarán presentes en la `Quotes#show` página.
- No necesitaremos la `LineItem#show` acción, ya que no tendría sentido que viéramos una sola línea de artículo. Siempre queremos ver la cotización en su totalidad.

Estas dos excepciones se reflejan en el archivo de rutas a continuación:

```
# config/routes.rb

Rails.application.routes.draw do
  # All the previous routes

  resources :quotes do
    resources :line_item_dates, except: [:index, :show] do
      resources :line_items, except: [:index, :show]
    end
  end
end
```

Ahora que nuestras rutas están listas, ¡es hora de comenzar a diseñar nuestra aplicación con datos falsos!

Diseño de elementos de línea

Las **fechas de los elementos** de la `Quotes#show` página actualmente no tienen ningún **elemento de la línea** . Solucionemos esto agregando algunos datos falsos a nuestros elementos.

Imaginemos que el editor de cotizaciones que estamos creando es para un software de eventos corporativos. Como los eventos pueden abarcar varias fechas, nuestras cotizaciones tendrán varias fechas y cada una tendrá varios elementos. En nuestro archivo de eventos, es posible que queramos agregar una sala donde los invitados puedan tener una reunión y una comida. Agreguemos esos elementos en el archivo de eventos:

```
# test/fixtures/line_items.yml

room_today:
  line_item_date: today
  name: Meeting room
  description: A cosy meeting room for 10 people
  quantity: 1
  unit_price: 1000

catering_today:
  line_item_date: today
  name: Meal tray
  description: Our delicious meal tray
  quantity: 10
  unit_price: 25

room_next_week:
  line_item_date: next_week
  name: Meeting room
  description: A cosy meeting room for 10 people
  quantity: 1
  unit_price: 1000

catering_next_week:
  line_item_date: next_week
  name: Meal tray
  description: Our delicious meal tray
  quantity: 10
  unit_price: 25
```

Ahora podemos volver a generar la base de datos ejecutando el `bin/rails db:seed` comando. Esas semillas nos permitirán diseñar nuestro editor de cotizaciones con datos falsos. Ahora abramos la aplicación en la `Quotes#show` página de la "Primera cotización". Queremos agregar dos elementos a cada **fecha de ítem de línea** en la página:

- La colección de artículos de línea para esta fecha
- El enlace para agregar nuevos artículos de línea para esta fecha

Agreguemos esos elementos completando el parcial para una sola línea de fecha del artículo:

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<%= turbo_frame_tag line_item_date do %>
  <div class="line-item-date">
    <div class="line-item-date__header">
      <!-- All the previous code -->
    </div>
    <div class="line-item-date__body">
      <div class="line-item line-item--header">
        <div class="line-item__name">Article</div>
        <div class="line-item__quantity">Quantity</div>
        <div class="line-item__price">Price</div>
        <div class="line-item__actions"></div>
      </div>

      <%= render line_item_date.line_items, quote: quote, line_item_date

      <div class="line-item-date__footer">
        <%= link_to "Add item",
                    [:new, quote, line_item_date, :line_item],
                    class: "btn btn--primary" %>
      </div>
    </div>
  </div>
<% end %>
```

Para representar cada elemento de línea, ahora tenemos que crear una parte para mostrar un solo elemento de línea:

```
<%# app/views/line_items/_line_item.html.erb %>

<div class="line-item">
  <div class="line-item__name">
    <%= line_item.name %>
    <div class="line-item__description">
      <%= simple_format line_item.description %>
    </div>
  </div>
  <div class="line-item__quantity-price">
    <%= line_item.quantity %>
    &times;
    <%= number_to_currency line_item.unit_price %>
  </div>
```

```
  <div class="line-item__quantity">
    <%= line_item.quantity %>
  </div>
  <div class="line-item__price">
    <%= number_to_currency line_item.unit_price %>
  </div>
  <div class="line-item__actions">
    <%= button_to "Delete",
                  [quote, line_item_date, line_item],
                  method: :delete,
                  class: "btn btn--light" %>
    <%= link_to "Edit",
                [:edit, quote, line_item_date, line_item],
                class: "btn btn--light" %>
  </div>
</div>
```

El `simple_format` asistente suele ser útil para representar texto que se escribió en un área de texto. Por ejemplo, imaginemos que un usuario escribió el siguiente texto en el campo de descripción de un **artículo de línea** :

```
- Appetizer
- Main course
- Dessert
- A glass of wine
```

El HTML generado por el `simple_format` ayudante en nuestra vista será:

```
<p>
  - Appetizers
  <br>
  - Main course
  <br>
  - Dessert
  <br>
  - A glass of wine
</p>
```

Como podemos ver, el formato que nuestro usuario quería al rellenar su área de texto se conserva gracias a los saltos de línea `<br>` . Si eliminamos el `simple_format` asistente, el texto se mostrará en una sola línea, lo cual no es lo que queremos aquí.

Las clases CSS `.line-item__quantity`, `.line-item__price`, y `.line-item__quantity-price` pueden parecer un poco redundantes, pero mostraremos las dos primeras clases CSS solo cuando el tamaño de la pantalla sea mayor que nuestro `tabletAndUp` punto de interrupción, y mostraremos la última clase CSS solo en dispositivos móviles.

Ahora que tenemos el código HTML, agreguemos un poco de CSS para que nuestro diseño sea un poco más atractivo. Lo primero que tenemos que hacer es finalizar nuestro `.line-item-date` componente que comenzamos en el capítulo anterior agregando los elementos `.line-item-date__body` y `.line-item-date__footer`:

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

.line-item-date {
  // All the previous code

  &__body {
    border-radius: var(--border-radius);
    background-color: var(--color-white);
    box-shadow: var(--shadow-small);
    margin-top: var(--space-xs);
    padding: var(--space-xxs);
    padding-top: 0;

    @include media(tabletAndUp) {
      padding: var(--space-m);
    }
  }

  &__footer {
    border: dashed 2px var(--color-light);
    border-radius: var(--border-radius);
    text-align: center;
    padding: var(--space-xxs);

    @include media(tabletAndUp) {
      padding: var(--space-m);
    }
  }
}
```

Ahora que nuestro `.line-item-date` componente CSS está completo, dediquemos un tiempo a diseñar cada **elemento de línea** individual . Escribiremos mucho CSS aquí, ya que crearemos:

- Nuestro `.line-item` componente base para diseñar un **artículo de una sola línea**
- Un `.line-item--header` modificador para diseñar la fila de encabezado de una colección de **elementos de línea**
- Un `.line-item--form` modificador para diseñar el formulario para crear y actualizar un **artículo de línea**

Todos esos componentes serán compatibles con **dispositivos móviles** y **tabletas, así como con pantallas más grandes** gracias a nuestro `tabletAndUp` punto de interrupción. Profundicemos en el código:

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

.line-item {
  display: flex;
  align-items: start;
  flex-wrap: wrap;
  background-color: var(--color-white);

  gap: var(--space-xs);
  margin-bottom: var(--space-s);
  padding: var(--space-xs);
  border-radius: var(--border-radius);

  > * {
    margin-bottom: 0;
  }

  &__name {
    flex: 1 1 100%;
    font-weight: bold;

    @include media(tabletAndUp) {
      flex: 1 1 0;
    }
  }

  &__description {
    flex-basis: 100%;
```

```
    max-width: 100%;
    color: var(--color-text-muted);
    font-weight: normal;
    font-size: var(--font-size-s);
  }

  &__quantity-price {
    flex: 0 0 auto;
    align-self: flex-end;
    justify-self: flex-end;
    order: 3;

    font-weight: bold;

    @include media(tabletAndUp) {
      display: none;
    }
  }

  &__quantity {
    flex: 1;
    display: none;

    @include media(tabletAndUp) {
      display: revert;
      flex: 0 0 7rem;
    }
  }

  &__price {
    flex: 1;
    display: none;

    @include media(tabletAndUp) {
      display: revert;
      flex: 0 0 9rem;
    }
  }

  &__actions {
    display: flex;
    gap: var(--space-xs);
    order: 2;
    flex: 1 1 auto;
```

```
    @include media(tabletAndUp) {
      order: revert;
      flex: 0 0 10rem;
    }
  }

  &--form {
    box-shadow: var(--shadow-small);

    .line-item__quantity,
    .line-item__price {
      display: block;
    }

    .line-item__description {
      order: 2;
    }
  }

  &--header {
    display: none;
    background-color: var(--color-light);
    margin-bottom: var(--space-s);

    @include media(tabletAndUp) {
      display: flex;
    }

    & > * {
      font-size: var(--font-size-s);
      font-weight: bold ;
      letter-spacing: 1px;
      text-transform: uppercase;
    }
  }
}
```

No olvidemos importar este nuevo archivo dentro de nuestro archivo de manifiesto:

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

// All the previous code
@import "components/line_item";
```

¡Eso fue **mucho** CSS, pero ahora todo debería verse bien! Si abrimos el navegador, deberíamos ver que nuestro diseño es *lo suficientemente bueno* .

Antes de pasar a la siguiente sección, observemos que actualmente tenemos un **problema de rendimiento** . Aunque está un poco fuera del alcance de este tutorial, es importante explicar qué sucede aquí. Si inspeccionamos los registros de nuestro servidor Rails al visitar la `Quotes#show` página, veremos un problema *de consulta N+1* aquí:

```
...
SELECT "line_items".* FROM "line_items" WHERE "line_items"."line_item_da
...
SELECT "line_items".* FROM "line_items" WHERE "line_items"."line_item_da
...
```

En el extracto de los registros anterior, consultamos la `line_items` tabla dos veces porque tenemos dos **fechas de artículos de línea** , pero la consultaríamos **n** veces si tuviéramos **n fechas de artículos de línea** . Esto se debe a que cada vez que representamos una nueva **fecha de artículo de línea** , también realizamos una solicitud para recuperar los **artículos de línea** asociados debido a esta línea:

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<%= render line_item_date.line_items, quote: quote, line_item_date: line
```

Una buena regla general para el rendimiento es que debemos consultar una tabla de base de datos solo una vez por ciclo de solicitud-respuesta .

Para evitar el problema *de la consulta N+1* , necesitamos cargar la colección de **artículos de línea** para cada **fecha de artículo de línea** con anticipación. Hagámoslo en la `QuotesController#show` acción:

```
# app/controllers/quotes_controller.rb

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

  def show
    @line_item_dates = @quote.line_item_dates.includes(:line_items).orde
  end
```

```
  # All the previous code...
end
```

Con esto `includes` agregado, deberíamos notar en los registros que ahora solo consultamos la `line_items` tabla **una vez** para mostrar la página:

```
SELECT "line_items".* FROM "line_items" WHERE "line_items"."line_item_da
```

Con ese problema de rendimiento resuelto, ¡ahora es hora de crear el nuestro `LineItemsController` y hacerlo funcionar!

Nuestro controlador CRUD estándar

Creación de líneas de pedido sin Turbo

Ahora que el esquema, el modelo, las rutas, el marcado y el diseño de nuestra base de datos están listos, es hora de comenzar a trabajar en el controlador. Como se mencionó en la introducción, primero crearemos un controlador estándar **sin Turbo Frames ni Turbo Streams; los agregaremos más adelante** .

Nuestro controlador contendrá las siete acciones del CRUD, excepto las acciones `#index` y `#show` . Comencemos por hacer que las acciones `#new` y `#create` funcionen:

```
# app/controllers/line_items_controller.rb

class LineItemsController < ApplicationController
  before_action :set_quote
  before_action :set_line_item_date

  def new
    @line_item = @line_item_date.line_items.build
  end

  def create
    @line_item = @line_item_date.line_items.build(line_item_params)

    if @line_item.save
```

```
      redirect_to quote_path(@quote), notice: "Item was successfully cre
    else
      render :new, status: :unprocessable_entity
    end
  end

  private

  def line_item_params
    params.require(:line_item).permit(:name, :description, :quantity, :u
  end

  def set_quote
    @quote = current_company.quotes.find(params[:quote_id])
  end

  def set_line_item_date
    @line_item_date = @quote.line_item_dates.find(params[:line_item_date
  end
end
```

Nuestro controlador es muy estándar y ya debería funcionar, pero nos falta la `line_items/new.html.erb` vista y el `line_items/_form.html.erb` parcial. Agreguemos esos dos archivos a nuestra aplicación:

```
<%# app/views/line_items/new.html.erb %>

<main class="container">
  <%= link_to sanitize("&larr; Back to quote"), quote_path(@quote) %>

  <div class="header">
    <h1>New item for <%= l(@line_item_date.date, format: :long) %></h1>
  </div>

  <%= render "form",
             quote: @quote,
             line_item_date: @line_item_date,
             line_item: @line_item %>
</main>
```

No necesitamos un diseño sofisticado para nuestra `LineItems#new` página, ya que utilizaremos Turbo más adelante para extraer el formulario de la página e

insertarlo en `Quotes#show` ella. Sin embargo, debería seguir siendo utilizable para las personas que utilizan navegadores antiguos que no admiten Turbo. Agreguemos el marcado para nuestro formulario:

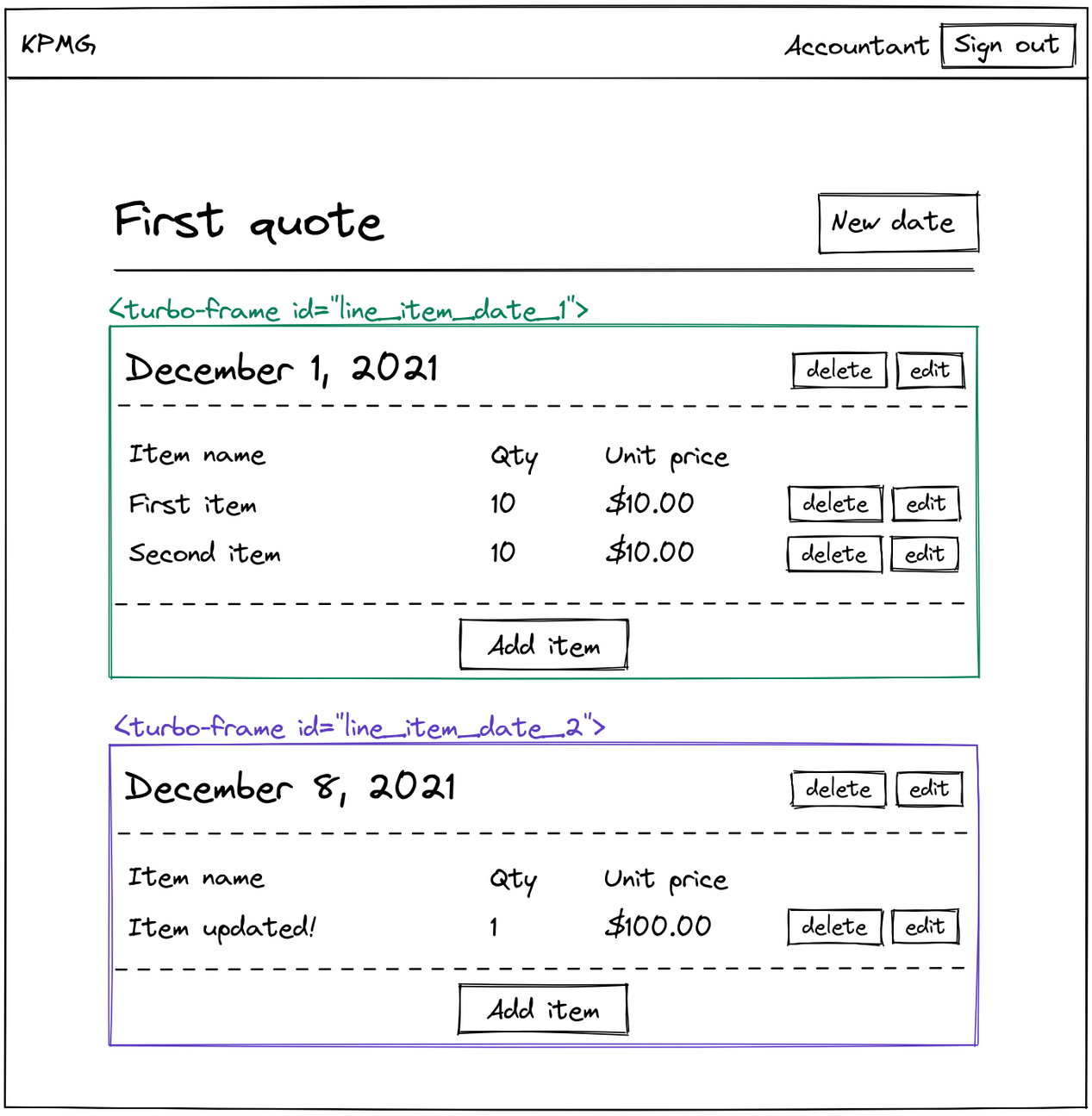
```
<%# app/views/line_items/_form.html.erb %>

<%= simple_form_for [quote, line_item_date, line_item],
                    html: { class: "form line-item line-item--form" } do

  <%= form_error_notification(line_item) %>

  <%= f.input :name,
              wrapper_html: { class: "line-item__name" },
              input_html: { autofocus: true } %>
  <%= f.input :quantity,
              wrapper_html: { class: "line-item__quantity" } %>
  <%= f.input :unit_price,
              wrapper_html: { class: "line-item__price" } %>
  <%= f.input :description,
              wrapper_html: { class: "line-item__description" } %>

  <div class="line-item__actions">
    <%= link_to "Cancel", quote_path(quote), class: "btn btn--light" %>
    <%= f.submit class: "btn btn--secondary" %>
  </div>
<% end %>
```

En este formulario, reutilizamos el `form_error_notification` asistente que creamos en el capítulo anterior. También reutilizamos la `.line-item` clase CSS en combinación con el `.line-item--form` modificador para darle estilo al formulario.

Probemos en nuestro navegador. **¡No funciona como se esperaba** ! La tarjeta **de fecha de la línea de pedido** desaparece y tenemos el siguiente error en la consola del navegador:

```
Response has no matching <turbo-frame id="line_item_date_123456"> elemen
```

Esto se debe a que nuestro enlace "Agregar elemento" ya está anidado dentro de un Turbo Frame que agregamos en el capítulo anterior como se describe en el siguiente boceto:



Boceto de la muestra de citas con los Turbo Frames del capítulo anterior

Debido a esos Turbo Frames, Turbo interceptará todos los clics en enlaces y envíos de formularios dentro de esos Turbo Frames y esperará un Turbo Frame con el mismo ID en la respuesta. Primero queremos que nuestro CRUD funcione **sin Turbo Frames ni Turbo Streams** .

Para evitar que Turbo intercepte nuestros clics en enlaces y envíos de formularios, utilizaremos el `data-turbo-frame="_top"` atributo data como se explicó en el primer capítulo sobre Turbo Frames . Agreguemos este atributo data a nuestro enlace "Agregar elemento":

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<!-- All the previous code -->

<div class="line-item-date__footer">
  <%= link_to "Add item",
```

```
                [:new, quote, line_item_date, :line_item],
                data: { turbo_frame: "_top" },
                class: "btn btn--primary" %>
</div>

<!-- All the previous code -->
```

Anticipemos también y agreguemos el mismo atributo de datos al enlace "Editar" y al formulario "Eliminar" en la línea parcial del artículo, ya que tendremos el mismo problema allí:

```
<%# app/views/line_items/_line_item.html.erb %>

<!-- All the previous code -->

<div class="line-item__actions">
  <%= button_to "Delete",
                [quote, line_item_date, line_item],
                method: :delete,
                form: { data: { turbo_frame: "_top" } },
                class: "btn btn--light" %>
  <%= link_to "Edit",
              [:edit, quote, line_item_date, line_item],
              data: { turbo_frame: "_top" },
              class: "btn btn--light" %>
</div>

<!-- All the previous code -->
```

Probemos ahora nuestra acción `#new` and `#create` en el navegador. ¡Ahora todo funciona como se esperaba!

Tomemos unos segundos para completar el archivo de traducciones con el texto que queremos para las etiquetas, marcadores de posición y botones de envío:

```
# config/locales/simple_form.en.yml

en:
  simple_form:
    placeholders:
      quote:
        name: Name of your quote
```

```
      line_item:
        name: Name of your item
        description: Description (optional)
        quantity: 1
        unit_price: $100.00
    labels:
      quote:
        name: Name
      line_item:
        name: Name
        description: Description
        quantity: Quantity
        unit_price: Unit price
      line_item_date:
        date: Date

  helpers:
    submit:
      quote:
        create: Create quote
        update: Update quote
      line_item:
        create: Create item
        update: Update item
      line_item_date:
        create: Create date
        update: Update date
```

Con ese archivo completado, el texto del botón de envío será "Fecha de creación" cuando creemos un `LineItemDate` y "Fecha de actualización" cuando actualicemos un `LineItemDate`.

Actualización de elementos de línea sin Turbo

Ahora que nuestras acciones `#new` y `#create` funcionan, hagamos el mismo trabajo para las acciones `#edit` y `#update`. Comencemos con el controlador:

```
class LineItemsController < ApplicationController
  before_action :set_quote
  before_action :set_line_item_date
  before_action :set_line_item, only: [:edit, :update, :destroy]

  # All the previous code
```

```
  def edit
  end

  def update
    if @line_item.update(line_item_params)
      redirect_to quote_path(@quote), notice: "Item was successfully upd
    else
      render :edit, status: :unprocessable_entity
    end
  end

  private

  # All the previous code

  def set_line_item
    @line_item = @line_item_date.line_items.find(params[:id])
  end
end
```

Sabemos que también necesitaremos la `set_line_item` devolución de llamada para la `#destroy` acción, por lo que podemos anticiparnos y agregarla a la lista de acciones que requieren esta devolución de llamada.

Ahora que nuestras acciones `#edit` y `#update` están implementadas, agreguemos la `LineItems#edit` vista para poder probar en el navegador:

```
<%# app/views/line_items/edit.html.erb %>

<main class="container">
  <%= link_to sanitize("&larr; Back to quote"), quote_path(@quote) %>

  <div class="header">
    <h1>Edit item</h1>
  </div>

  <%= render "form",
             quote: @quote,
             line_item_date: @line_item_date,
             line_item: @line_item %>
</main>
```

Como podemos observar, la `LineItems#edit` vista es muy similar a la `LineItems#new` vista anterior. Solo cambia el título. Como ya construimos el formulario en la sección anterior, estamos listos para experimentar en el navegador. Todo funciona como se espera; ¡solo falta una acción más!

Eliminar elementos de línea sin Turbo

La `#destroy` acción es la más sencilla de las cinco, ya que no requiere una vista. Solo tenemos que eliminar el **elemento de línea** y luego redirigir a la `Quotes#show` página:

```
# app/controllers/line_items_controller.rb

class LineItemsController < ApplicationController
  # All the previous code

  def destroy
    @line_item.destroy

    redirect_to quote_path(@quote), notice: "Item was successfully destr
  end

  # All the previous code
end
```

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

Nuestro controlador CRUD ya está funcionando, pero queremos que todas las interacciones se realicen en la misma página. Gracias a la potencia de Turbo, solo se necesitarán unas pocas líneas de código para dividir nuestra página en partes que se puedan actualizar de forma independiente.

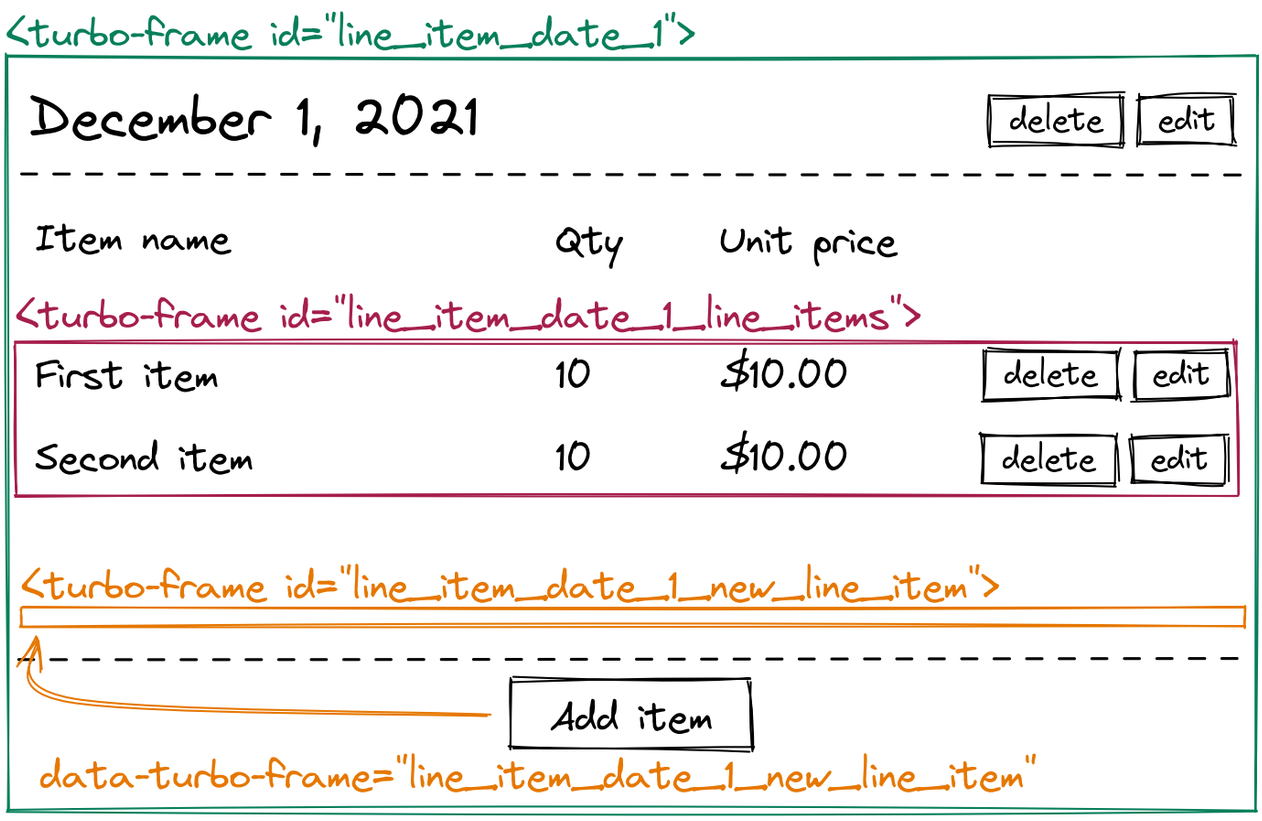
Cómo agregar Turbo Frames y Turbo Streams

Creación de líneas de pedido con Turbo

Ahora que nuestro controlador CRUD funciona como se esperaba, es momento de mejorar la experiencia del usuario para que todas las interacciones ocurran en `Quotes#show` la página.

Para aclarar los requisitos, primero esbocemos el comportamiento deseado. ***A partir de ahora, ampliaremos los bocetos en una sola línea de fecha para que sigan siendo legibles*** .

Cuando un usuario visita la `Quotes#show` página y hace clic en el botón "Añadir artículo" **para una fecha específica** , queremos que el formulario aparezca en la `Quotes#show` página justo encima del botón "Añadir artículo" en el que acabamos de hacer clic. ¡Haremos esto con Turbo Frames, por supuesto! Para que funcione, tenemos que conectar el enlace "Añadir artículo" a un Turbo Frame vacío gracias al `data-turbo-frame` atributo data.



Boceto de una fecha concreta con Turbo Frames

Para que Turbo reemplace correctamente el Turbo Frame en la `Quotes#show` página, el Turbo Frame en la `LineItems#new` página debe tener la misma identificación.

Observamos que los identificadores de Turbo Frame son más largos que en los capítulos anteriores. **Los Turbo Frames deben tener identificadores únicos en la página para funcionar correctamente** . Como tenemos varias fechas en la página, si el identificador del Turbo Frame vacío fuera solo `new_line_item` , o el identificador de la lista de **elementos de línea** fuera solo `line_items` , tendríamos varios Turbo Frames con el mismo identificador.

Expliquemos por qué **los Turbo Frames de la misma página deben tener diferentes identificadores** . Si hiciéramos lo mismo que en los capítulos

anteriores, nuestra `create.turbo_stream.erb` vista se vería así:

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

<%= turbo_stream.update LineItem.new, "" %>
<%= turbo_stream.append "line_items", @line_item %>
```

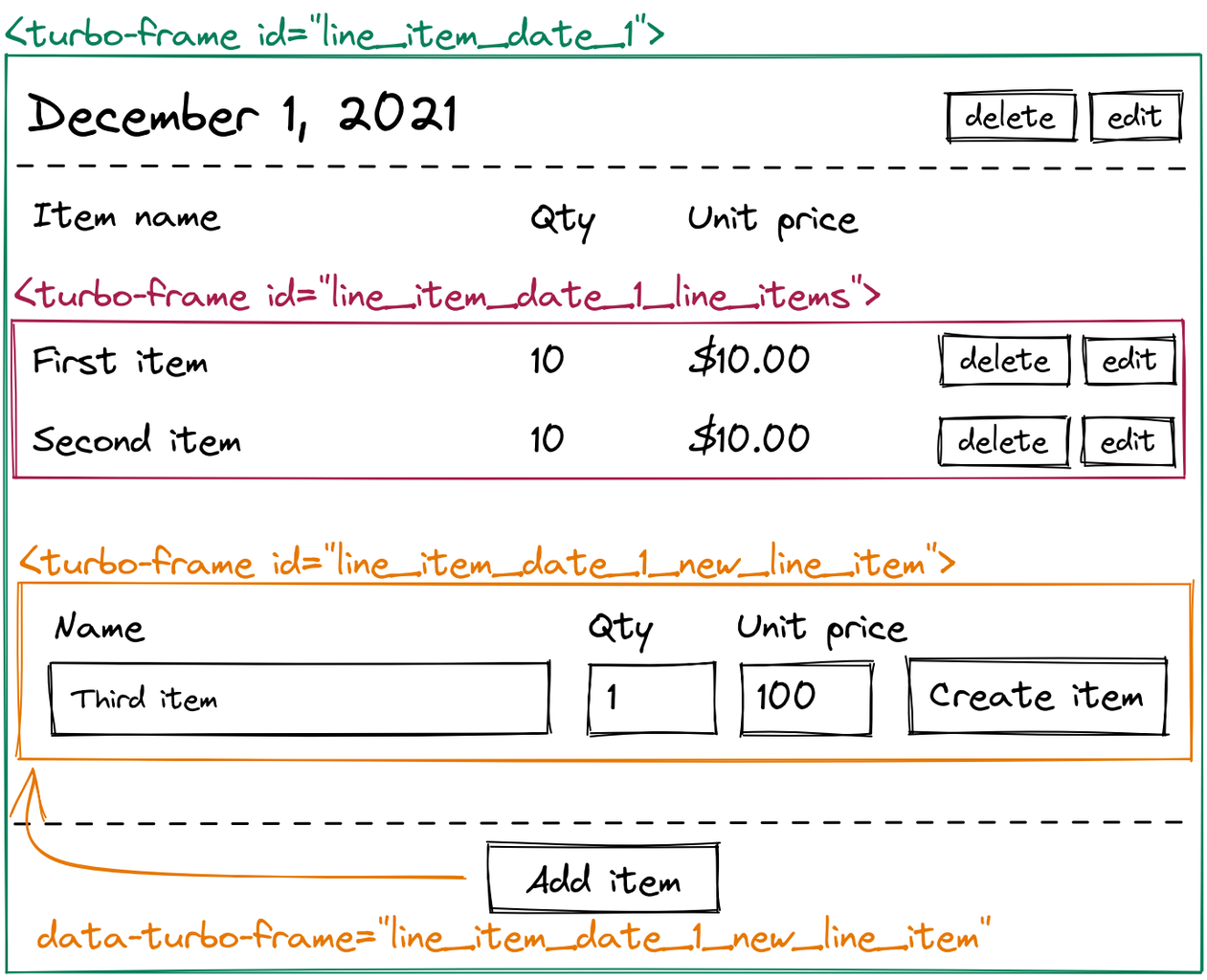
Si hay varias **fechas de artículos de línea** en la cotización, entonces aparecerán los identificadores `new_line_item` y `line_items` varias veces en la `Quotes#show` página. ¿Cómo podría Turbo adivinar qué hacer si hay varias veces el mismo identificador? ¡Nuestro **artículo de línea** creado probablemente se *agregaría* a la lista de artículos de línea de la fecha incorrecta!

Una buena convención es anteponer a los identificadores que normalmente tendríamos el `dom_id` del recurso principal para resolver este problema . De esa manera, nos aseguramos de que nuestros identificadores sean únicos.

Para que Turbo funcione correctamente, necesitamos un Turbo Frame con la misma identificación en la `LineItems#new` página:

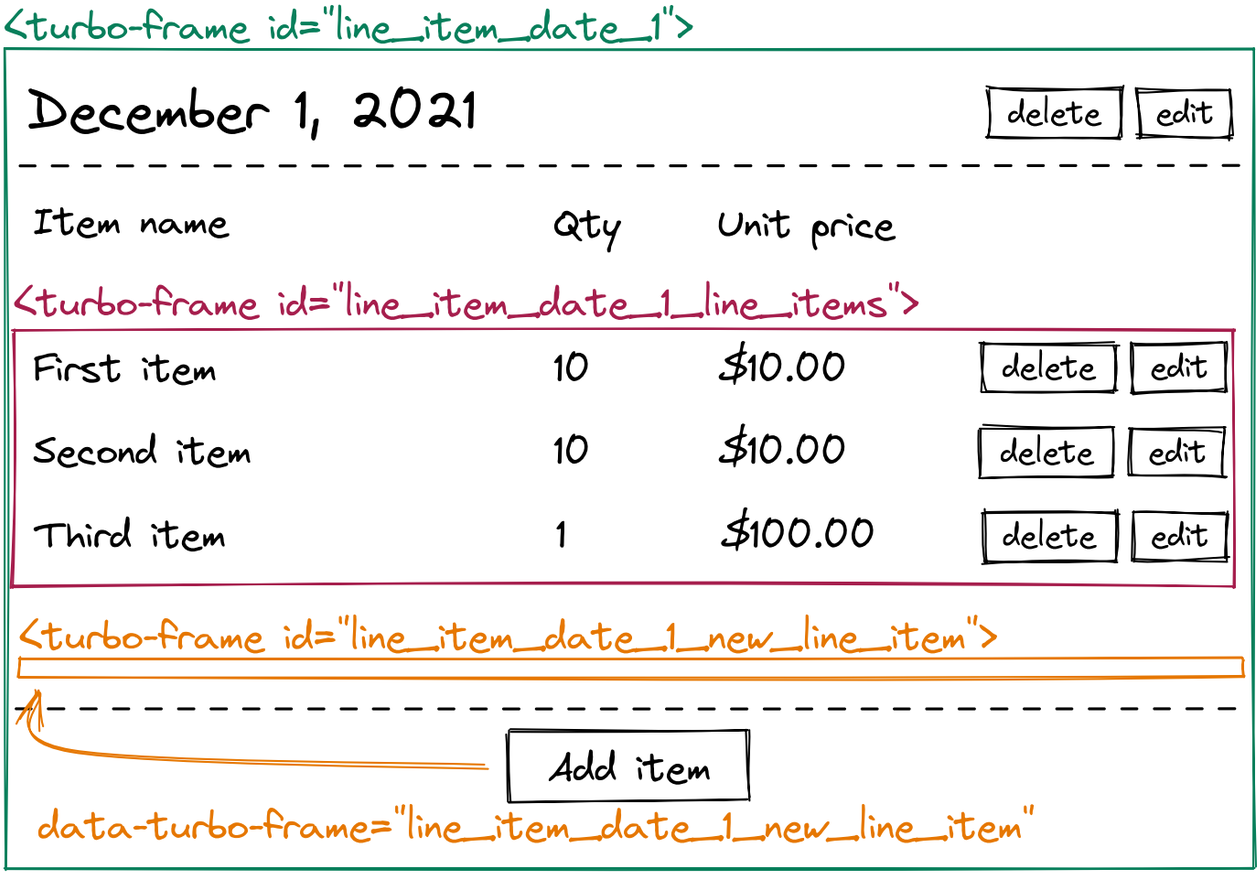
Boceto de la página LineItems#new con un Turbo Frame

Con esos Turbo Frames en su lugar, cuando un usuario haga clic en el botón "Nuevo elemento", Turbo reemplazará exitosamente el Turbo Frame vacío en la `Quotes#show` página con el Turbo Frame que contiene el formulario en la `LineItems#new` página:



Boceto de una fecha específica con el formulario de la página LineItems#new

Cuando el usuario envía el formulario, queremos que el **elemento de línea** creado se *agregue* a la lista de **elementos de línea** para esta fecha específica:



Boceto de una fecha específica con el elemento creado adjunto

Ahora que los requisitos están claros, solo deberían hacer falta unas pocas líneas de código para hacerlo realidad, ¡gracias al poder de Turbo Frames y Turbo Streams!

Comencemos a trabajar en la primera parte: hacer que el formulario aparezca en la `Quotes#show` página cuando un usuario haga clic en el botón "Agregar elemento". En cada **línea de fecha de elemento** , agreguemos un Turbo Frame vacío y conectemos el botón "Agregar fecha" a él:

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<%= turbo_frame_tag line_item_date do %>
  <div class="line-item-date">
    <!-- All the previous code -->
    <div class="line-item-date__body">
      <div class="line-item line-item--header">
        <!-- All the previous code -->
      </div>

      <%= render line_item_date.line_items, quote: quote, line_item_date

      <%= turbo_frame_tag dom_id(LineItem.new, dom_id(line_item_date)) %

      <div class="line-item-date__footer">
        <%= link_to "Add item",
                    [:new, quote, line_item_date, :line_item],
                    data: { turbo_frame: dom_id(LineItem.new, dom_id(lin
                    class: "btn btn--primary" %>
      </div>
    </div>
  </div>
<% end %>
```

Como se mencionó anteriormente, para los recursos anidados, queremos anteponer el `dom_id` del recurso con el `dom_id` del padre. El `dom_id` ayudante toma un prefijo opcional como segundo argumento. Podríamos usar el `dom_id` ayudante para seguir nuestra convención:

```
line_item_date = LineItemDate.find(1)

dom_id(LineItem.new, dom_id(line_item_date))
# => line_item_date_1_new_line_item
```

Este enfoque funciona bien, pero es difícil de leer. También tiene un caso extremo:

```
dom_id("line_items", dom_id(line_item_date))
# This does not return "line_item_date_1_line_items"
# It raises an error as "line_items" does not respond to `#to_key`
# and so can't be transformed into a dom_id
```

En lugar de confiar `dom_id` directamente en el ayudante, creemos un ayudante para que nuestras identificaciones sean más fáciles de generar/leer y garantizar que todos los desarrolladores de nuestro equipo usen la misma convención:

```
# app/helpers/application_helper.rb

module ApplicationHelper
  # All the previous code

  def nested_dom_id(*args)
    args.map { |arg| arg.respond_to?(:to_key) ? dom_id(arg) : arg }.join
  end
end
```

Con este asistente en su lugar, es mucho más fácil generar y leer nuestro `dom_ids` :

```
line_item_date = LineItemDate.find(1)

nested_dom_id(line_item_date, LineItem.new)
# => line_item_date_1_new_line_item

nested_dom_id(line_item_date, "line_items")
# => line_item_date_1_line_items
```

Actualicemos la vista para utilizar nuestra nueva convención:

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<%= turbo_frame_tag line_item_date do %>
  <div class="line-item-date">
    <!-- All the previous code -->
    <div class="line-item-date__body">
```

```
      <div class="line-item line-item--header">
        <!-- All the previous code -->
      </div>

      <%= render line_item_date.line_items, quote: quote, line_item_date

      <%= turbo_frame_tag nested_dom_id(line_item_date, LineItem.new) %>

      <div class="line-item-date__footer">
        <%= link_to "Add item",
                    [:new, quote, line_item_date, :line_item],
                    data: { turbo_frame: nested_dom_id(line_item_date, L
                    class: "btn btn--primary" %>
      </div>
    </div>
  </div>
<% end %>
```

Ahora que nuestros Turbo Frames tienen los identificadores esperados en la `Quotes#show` página, necesitamos tener Turbo Frames coincidentes en la `LineItems#new` página para que Turbo intercambie los marcos. Envolvamos nuestro formulario en una etiqueta Turbo Frame:

```
<%# app/views/line_items/new.html.erb %>

<main class="container">
  <%= link_to sanitize("&larr; Back to quote"), quote_path(@quote) %>

  <div class="header">
    <h1>New item for <%= l(@line_item_date.date, format: :long) %></h1>
  </div>

  <%= turbo_frame_tag nested_dom_id(@line_item_date, LineItem.new) do %>
    <%= render "form",
               quote: @quote,
               line_item_date: @line_item_date,
               line_item: @line_item %>
  <% end %>
</main>
```

Experimentemos en el navegador. Al hacer clic en el botón "Agregar elemento", el formulario debería aparecer en la posición esperada para esta

fecha.

Al igual que en los capítulos anteriores, cuando enviamos un formulario **no válido , los errores aparecen en la página como se espera.**

We have to give Turbo more precise instructions when submitting a **valid** form thanks to a Turbo Stream view. We want to perform two actions:

1. Remove the form we just submitted from the DOM
2. Append the created **line item** to the list of **line items** for this specific date

Let's edit our `LineItemsController#create` action to respond to the `turbo_stream` format:

```
# app/controllers/line_items_controller.rb

class LineItemsController < ApplicationController
  # All the previous code...

  def create
    @line_item = @line_item_date.line_items.build(line_item_params)

    if @line_item.save
      respond_to do |format|
        format.html { redirect_to quote_path(@quote), notice: "Item was
        format.turbo_stream { flash.now[:notice] = "Item was successfull
      end
    else
      render :new, status: :unprocessable_entity
    end
  end

  # All the previous code...
end
```

Let's create our view that will perform the two actions that we want:

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

<%# Step 1: empty the Turbo Frame containing the form %>
<%= turbo_stream.update nested_dom_id(@line_item_date, LineItem.new), ""
```

```
<%# Step 2: append the created line item to the list %>
<%= turbo_stream.append nested_dom_id(@line_item_date, "line_items") do
  <%= render @line_item, quote: @quote, line_item_date: @line_item_date
<% end %>

<%= render_turbo_stream_flash_messages %>
```

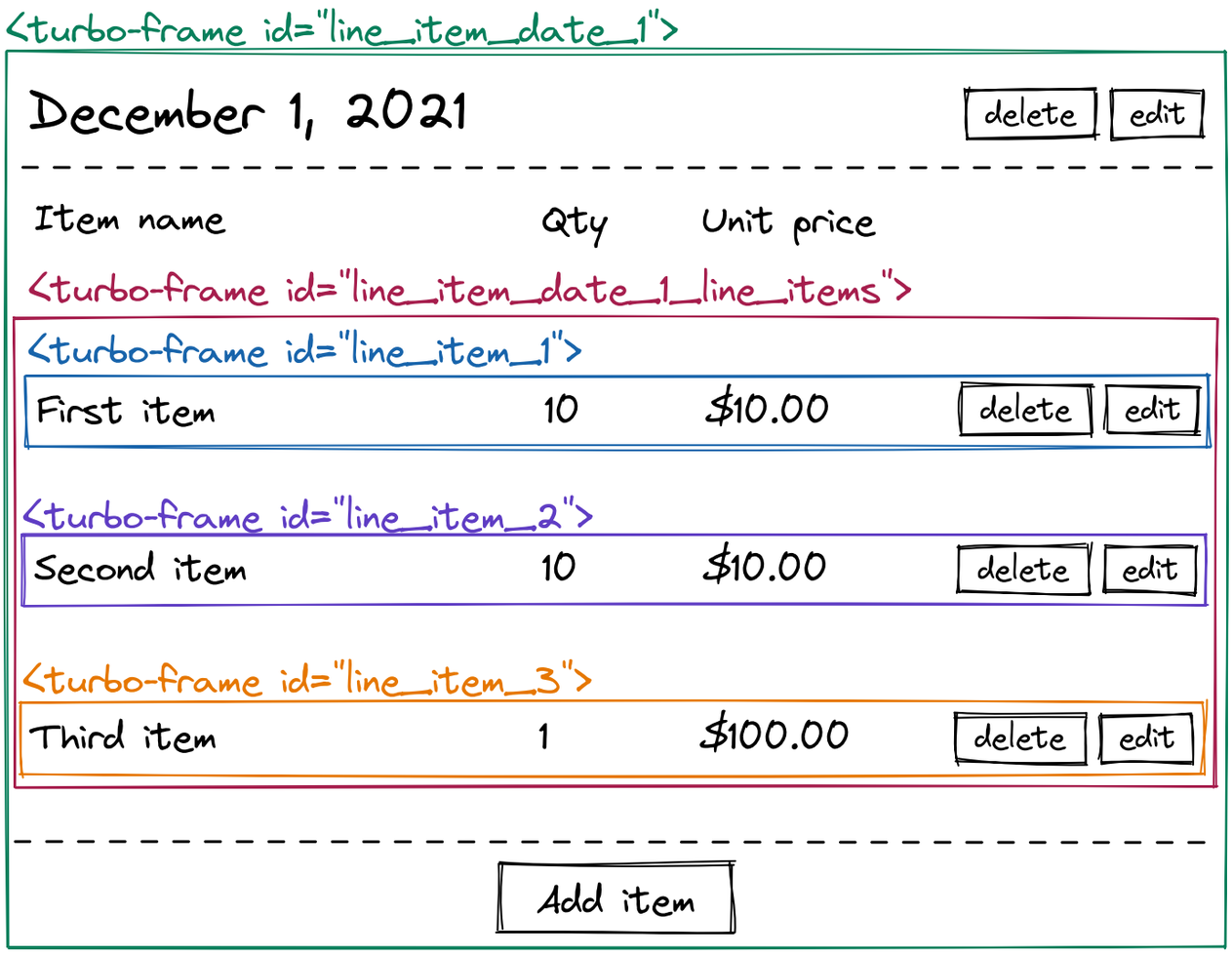
The last thing we need to do is to add a Turbo Frame to wrap the list of **line items** for each specific date:

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<!-- All the previous code -->
<%= turbo_frame_tag nested_dom_id(line_item_date, "line_items") do %>
  <%= render line_item_date.line_items, quote: quote, line_item_date: li
<% end %>
<!-- All the previous code -->
```

Let's test it in our browser, and everything should work as expected. That was a lot of work! We are almost there. The `#edit`, `#update`, and `#destroy` actions will be easier to implement now that nearly everything is in place.

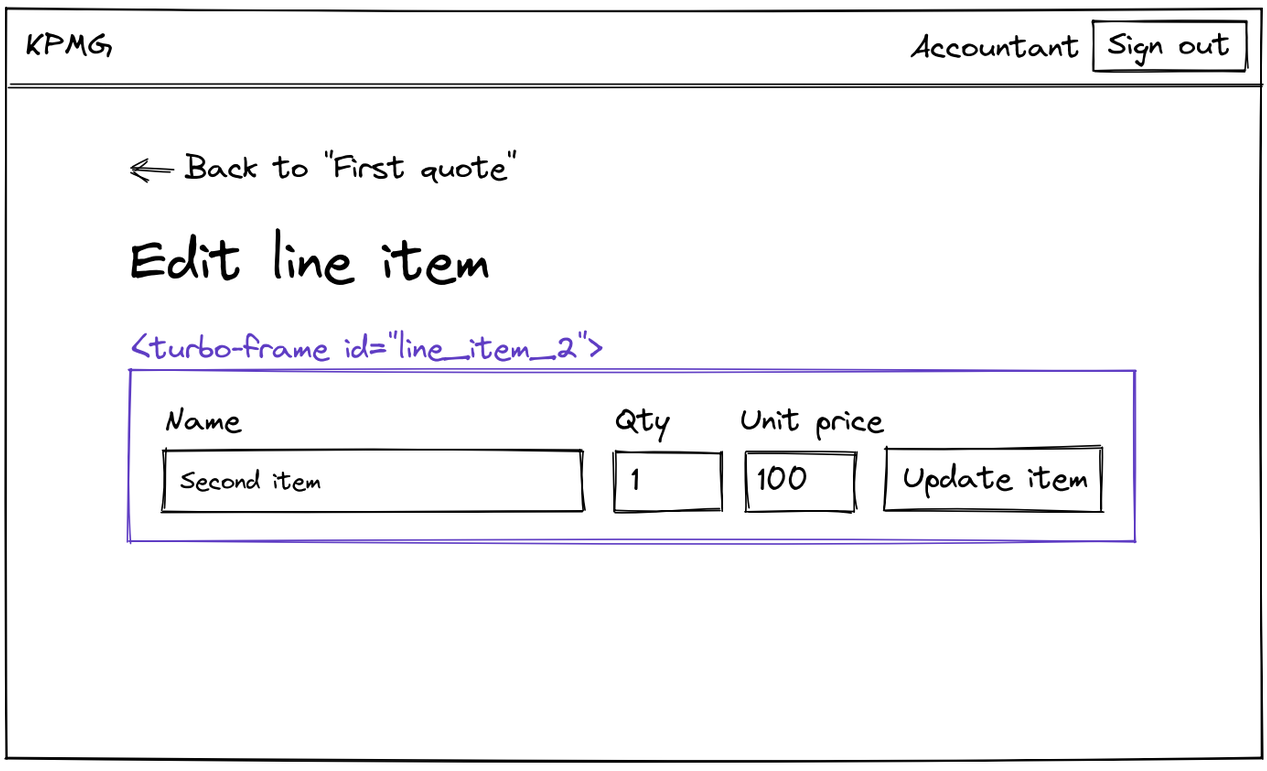
Updating line items with Turbo

Like we did for the `#new` and `#create` actions, we want to make the `#edit` and `#update` actions for a quote happen on the `Quotes#show` page. We already have most of the Turbo Frames we need, but we are going to need each line item also to be wrapped inside a Turbo Frame as described in the sketch below:



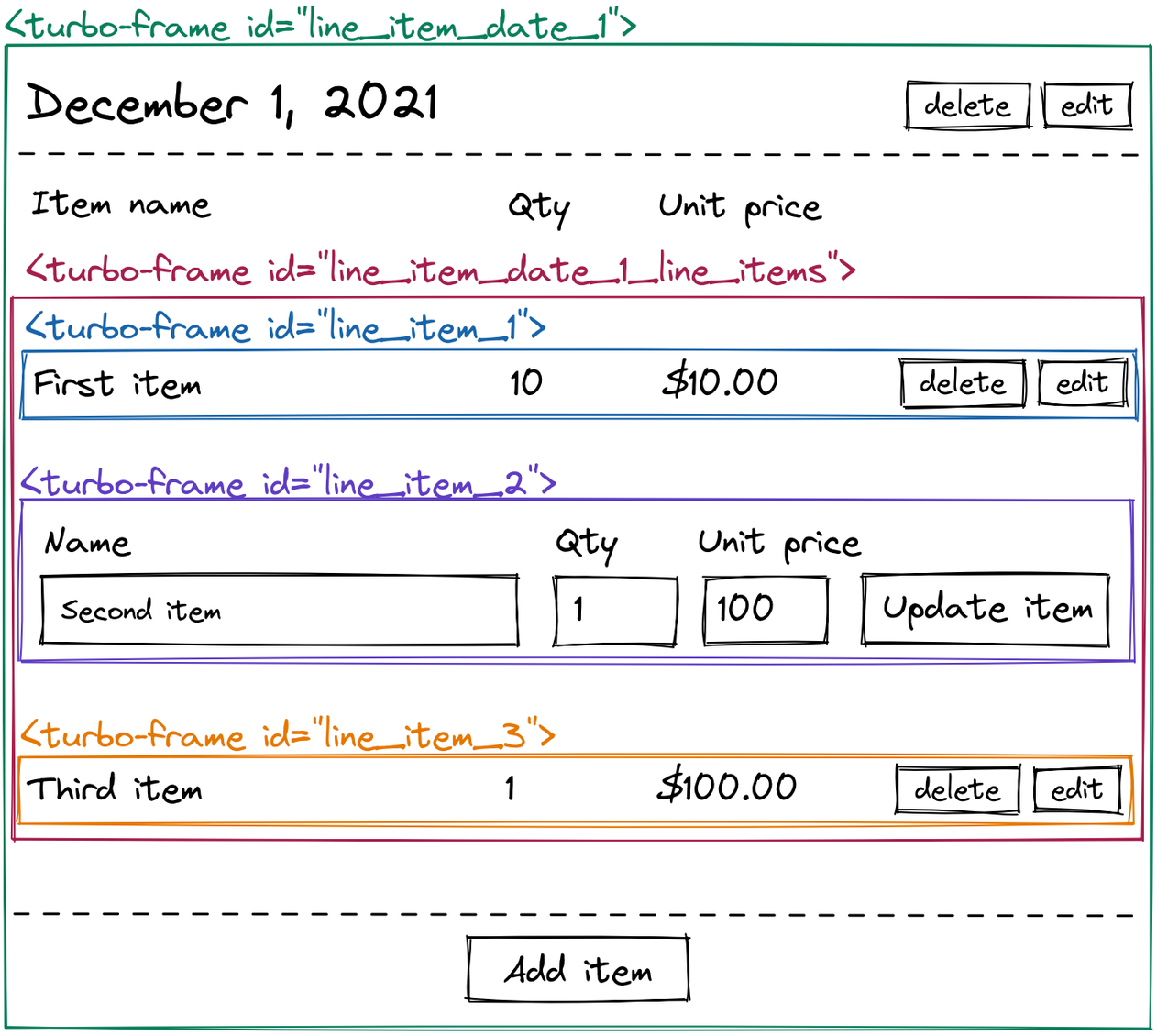
Sketch of a specific date with Turbo Frames

When clicking on the "Edit" link for the **second line item** of our sketch that is within a Turbo Frame of id `line_item_2`, Turbo expects to find a Turbo Frame with the same id on the `LineItems#edit` page as described in the sketch below:



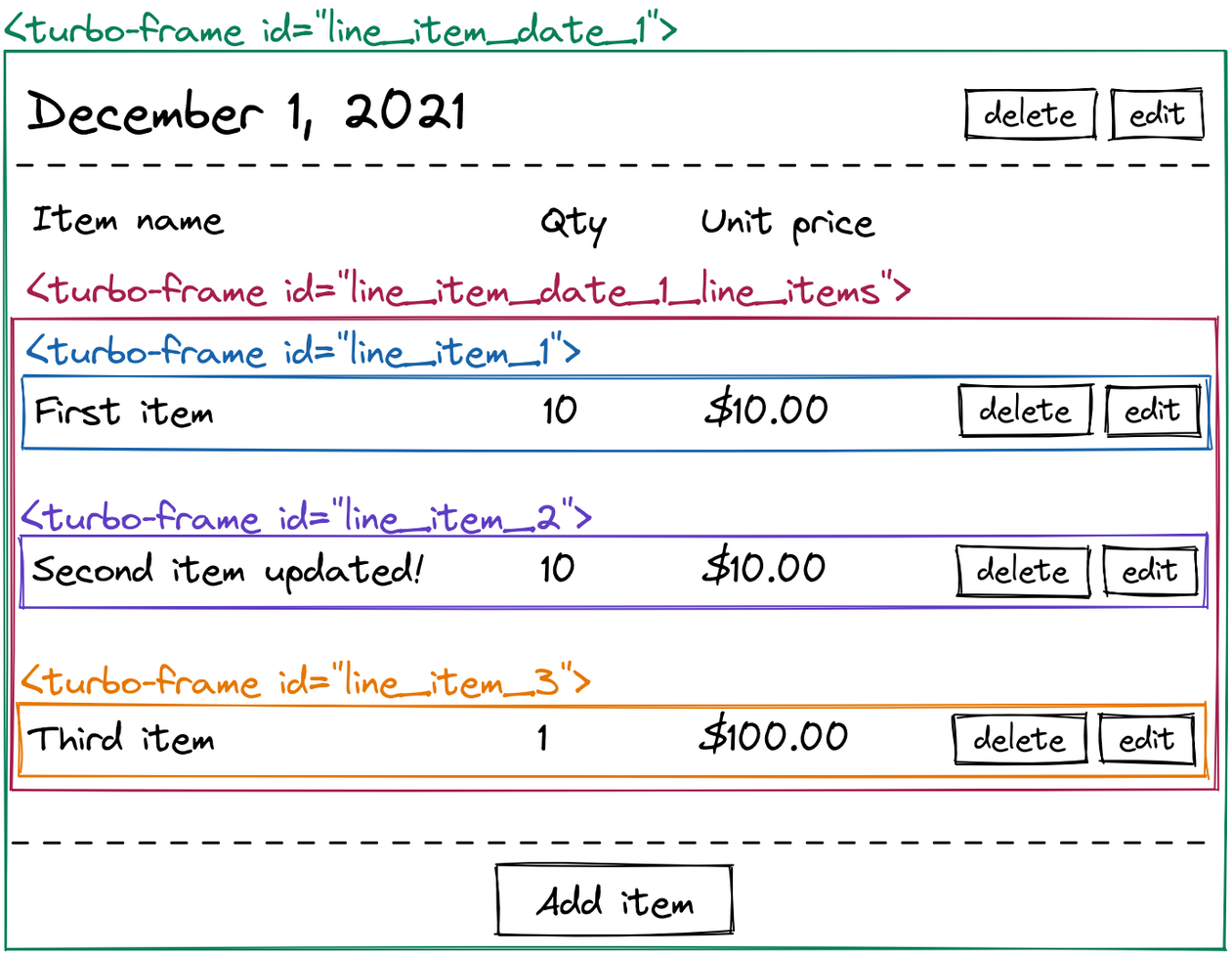
Sketch of the LineItems#edit page with a Turbo Frame of the same id

With those Turbo Frames in place, Turbo will be able to replace the line item with the form from the `LineItems#edit` page when clicking on the "Edit" link of a line item:



Sketch of specific date with the edit form for the second item

When submitting the form, we want the form to be replaced with the final quote:



Sketch of the specific date after the form was submitted

Now that the requirements are clear, it's time to start coding. The first part of the job is to make the edit form successfully replace the HTML of **line items** on the `Quotes#show` page. To do this, let's wrap every item in a Turbo Frame:

```
<%# app/views/line_items/_line_item.html.erb %>

<%= turbo_frame_tag line_item do %>
  <div class="line-item">
    <!-- All the previous code -->
  </div>
<% end %>
```

We also need to remove the `data-turbo-frame="_top"` data attribute from the "Edit" link:

```
<%# app/views/line_items/_line_item.html.erb %>

<!-- All the previous code -->
<%= link_to "Edit",
            [:edit, quote, line_item_date, line_item],
            class: "btn btn--light" %>
<!-- All the previous code -->
```

Now that we wrapped our **line items** in Turbo Frames, we need to wrap the form in the `LineItems#edit` page in a Turbo Frame as well:

```
<%# app/views/line_items/edit.html.erb %>

<main class="container">
  <%= link_to sanitize("&larr; Back to quote"), quote_path(@quote) %>

  <div class="header">
    <h1>Edit item</h1>
  </div>

  <%= turbo_frame_tag @line_item do %>
    <%= render "form",
               quote: @quote,
               line_item_date: @line_item_date,
               line_item: @line_item %>
  <% end %>
</main>
```

With this Turbo Frame in place, we can test in the browser. When clicking on the "Edit" button on a **line item**, the form successfully replaces the **line item** on the `Quotes#show` page.

If we submit an invalid form, everything already works as expected. If we submit a valid form, the line item date is successfully replaced but we miss the flash message. To solve this, we will need a Turbo Stream view. Let's first enable our controller to render a Turbo Stream view:

```
# app/controllers/line_items_controller.rb

def update
  if @line_item.update(line_item_params)
    respond_to do |format|
      format.html { redirect_to quote_path(@quote), notice: "Item was su
      format.turbo_stream { flash.now[:notice] = "Item was successfully
    end
  else
    render :edit, status: :unprocessable_entity
  end
end
```

Let's now create the `update.turbo_stream.erb` view to replace the line item form with the line item partial and render the flash message:

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

<%= turbo_stream.replace @line_item do %>
  <%= render @line_item, quote: @quote, line_item_date: @line_item_date
<% end %>

<%= render_turbo_stream_flash_messages %>
```

Let's test it in the browser; everything should work as expected!

Destroying line items with Turbo

The last feature we need is the ability to remove line item dates from our quote. To do this, we first have to support the Turbo Stream format in the `#destroy` action in the controller:

```
# app/controllers/line_items_controller.rb

def destroy
  @line_item.destroy

  respond_to do |format|
    format.html { redirect_to quote_path(@quote), notice: "Date was succ
    format.turbo_stream { flash.now[:notice] = "Date was successfully de
  end
end
```

In the view, we only have to remove the line item and render the flash message:

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

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

Let's not forget to remove the `data-turbo-frame="_top"` data attribute from the "Delete" button:

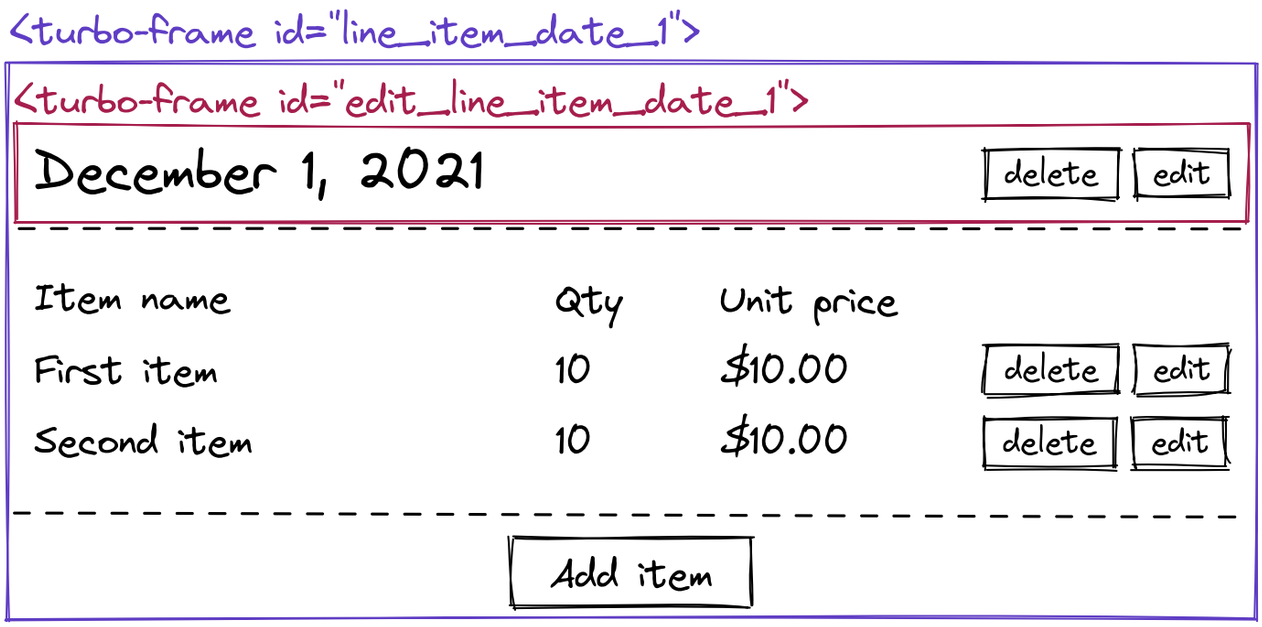
```
<%# app/views/line_items/_line_item.html.erb %>

<!-- All the previous code -->
<%= button_to "Delete",
              [quote, line_item_date, line_item],
              method: :delete,
              class: "btn btn--light" %>
<!-- All the previous code -->
```

We can finally test in our browser that everything works as expected. The behavior is almost exactly the same as the one we had for quotes!

Editing line item dates with Turbo

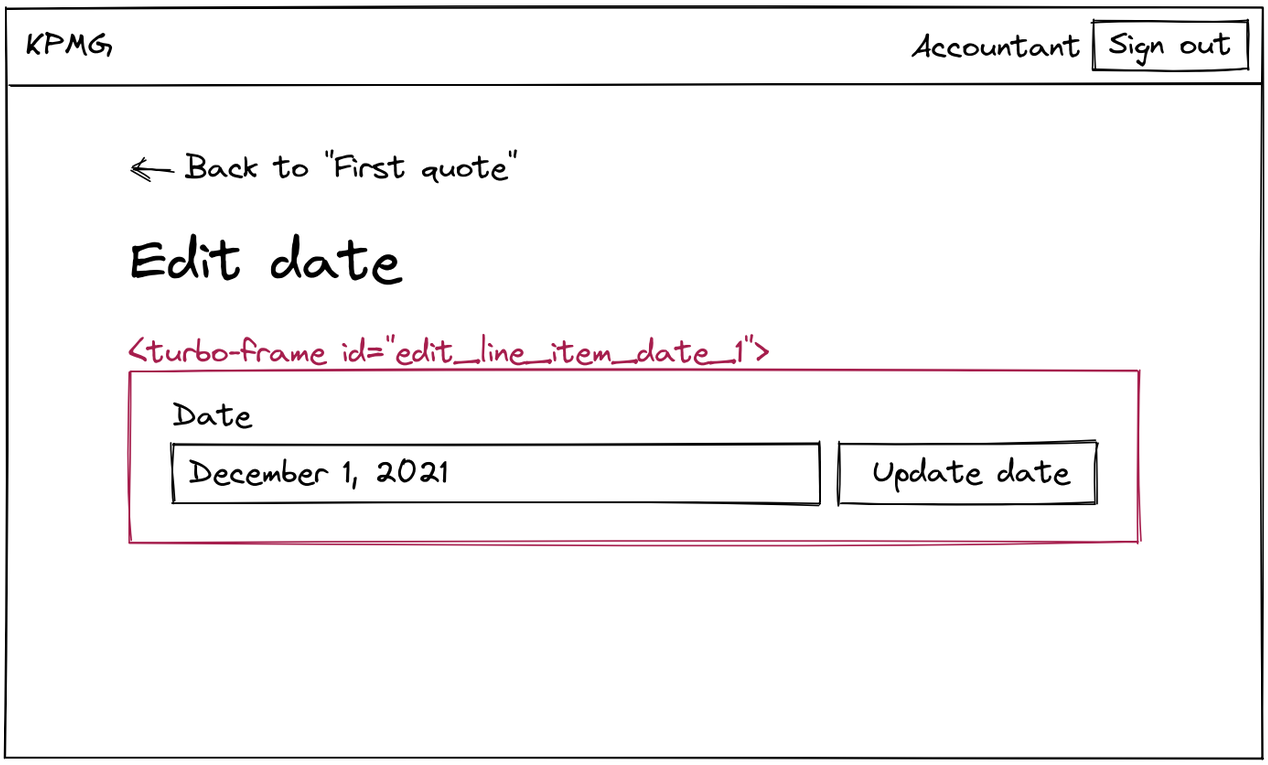
The actions for our line items are now complete! However, we have a small issue: the whole **line item date** card is replaced by the edit form when clicking on the "Edit" link for a **line item date**. We would like only the card's header contaning the date to be replaced.

Let's wrap only the header of the **line item date** card inside another Turbo Frame with a unique id by prefixing its `dom_id` with "edit":



Sketch of a specific date on the Quotes#show page

For Turbo to be able to replace the Turbo Frame, we need a Turbo Frame with the same id on the `LineItemDates#edit` page:



Sketch of the LineItemDates#edit page with a Turbo Frame

Now, when clicking on the "Edit" button for this specific date, Turbo will only replace the header of the **line item date** card:

Sketch of the LineItemDates#edit page with a Turbo Frame

Now that the requirements are clear, it's time to start coding. Let's first start by adding the Turbo Frame with the "edit" prefix to the line item date partial:

```
<%# app/views/line_item_dates/_line_item_date.html.erb %>

<%= turbo_frame_tag line_item_date do %>
  <div class="line-item-date">
    <%= turbo_frame_tag dom_id(line_item_date, :edit) do %>
```

```
      <div class="line-item-date__header">
        <!-- All the previous code -->
      </div>
    <% end %>
    <div class="line-item-date__body">
      <!-- All the previous code -->
    </div>
  </div>
<% end %>
```

We also need to add the "edit" prefix to the `LineItemDates#edit` page:

```
<%# app/views/line_item_dates/edit.html.erb %>

<main class="container">
  <%= link_to sanitize("&larr; Back to quote"), quote_path(@quote) %>

  <div class="header">
    <h1>Edit date</h1>
  </div>

  <%= turbo_frame_tag dom_id(@line_item_date, :edit) do %>
    <%= render "form", quote: @quote, line_item_date: @line_item_date %>
  <% end %>
</main>
```

Let's test it in the browser. Now when clicking on the edit link for a **line item date**, only the card's header is replaced by the form from the `LineItemDates#edit` page.

Preserving state with Turbo Rails

Until now, we managed to preserve the *state* of our application all the time by making pieces of our page really independent. However, there is a small glitch in our application right.

To demonstrate our *state* issue, let's navigate on the `Quotes#show` page for the first quote and open a few **line item** forms for the first **line item date** by clicking on the "Edit" button for those **line items**. Let's then update the first **line item date**. The forms are now closed again!

Esto se debe a que, para mantener nuestras fechas en orden ascendente, eliminamos por completo la tarjeta **de fecha del elemento de línea** del DOM y luego la volvemos a adjuntar en la posición correcta en la lista. Por supuesto, si eliminamos por completo y volvemos a adjuntar la **fecha del elemento de línea** , perderemos el *estado* de los **elementos de línea** dentro de esta fecha, ya que el parcial se representa con todos los formularios cerrados de forma predeterminada.

Aquí llegamos al límite de lo que podemos hacer con Turbo Rails sin escribir ningún código JavaScript personalizado. Si quisiéramos conservar el *estado* de la aplicación en la `Quotes#show` página al actualizar la **fecha de un elemento de línea** , tendríamos dos soluciones:

- No reordene los elementos al utilizar el formato Turbo Stream
- Reordenar elementos en el frontend con un controlador de estímulo

Aunque se trate de un error menor, es importante conocer las limitaciones de lo que Turbo puede hacer por sí solo. En este tutorial, simplemente ignoraremos este error.

Probando nuestro código con pruebas del sistema

Nuestro trabajo no estaría completo si no agregáramos algunas pruebas. Siempre deberíamos escribir al menos pruebas del sistema para asegurarnos de que se cubra el camino correcto. Si cometemos un error, podemos corregirlo antes de enviar nuestro código a producción.

Agreguemos un archivo de prueba del sistema para probar la ruta correcta del CRUD en nuestros elementos de línea:


```
# test/system/line_items_test.rb

require "application_system_test_case"

class LineItemSystemTest < ApplicationSystemTestCase
  include ActionView::Helpers::NumberHelper

  setup do
    login_as users(:accountant)
```

```
  @quote          = quotes(:first)
  @line_item_date = line_item_dates(:today)
  @line_item      = line_items(:room_today)

  visit quote_path(@quote)
end

test "Creating a new line item" do
  assert_selector "h1", text: "First quote"

  within "##{dom_id(@line_item_date)}" do
    click_on "Add item", match: :first
  end
  assert_selector "h1", text: "First quote"

  fill_in "Name", with: "Animation"
  fill_in "Quantity", with: 1
  fill_in "Unit price", with: 1234
  click_on "Create item"

  assert_selector "h1", text: "First quote"
  assert_text "Animation"
  assert_text number_to_currency(1234)
end

test "Updating a line item" do
  assert_selector "h1", text: "First quote"

  within "##{dom_id(@line_item)}" do
    click_on "Edit"
  end
  assert_selector "h1", text: "First quote"

  fill_in "Name", with: "Capybara article"
  fill_in "Unit price", with: 1234
  click_on "Update item"

  assert_text "Capybara article"
  assert_text number_to_currency(1234)
end

test "Destroying a line item" do
  within "##{dom_id(@line_item_date)}" do
    assert_text @line_item.name
```

```
    end

    within "##{dom_id(@line_item)}" do
      click_on "Delete"
    end

    within "##{dom_id(@line_item_date)}" do
      assert_no_text @line_item.name
    end
  end
end
```

Si ejecutamos el `bin/rails test:all` comando, nos daremos cuenta de que tenemos dos pruebas anteriores que corregir. Como tenemos demasiados enlaces "Editar" y "Eliminar" con el mismo nombre, Capybara no sabrá en cuál hacer clic y generará un `Capybara::Ambiguous` error.

Para solucionar ese problema, debemos ser más específicos con los identificadores que usamos en nuestros `within` bloques:

```
# test/system/line_item_dates_test.rb

# All the previous code

test "Updating a line item date" do
  assert_selector "h1", text: "First quote"

  within id: dom_id(@line_item_date, :edit) do
    click_on "Edit"
  end

  assert_selector "h1", text: "First quote"

  fill_in "Date", with: Date.current + 1.day
  click_on "Update date"

  assert_text I18n.l(Date.current + 1.day, format: :long)
end

test "Destroying a line item date" do
  assert_text I18n.l(Date.current, format: :long)

  accept_confirm do
```

```
    within id: dom_id(@line_item_date, :edit) do
      click_on "Delete"
    end
  end

  assert_no_text I18n.l(Date.current, format: :long)
end

# All the previous code
```

Ejecutemos todas las pruebas con `bin/rails test:all` el comando, ¡y ahora todas deberían estar en verde!

Envolver

En este capítulo, casi hemos finalizado nuestro editor de citas. Aprendimos a administrar Turbo Frames anidados y a mantener nuestro código legible gracias a las convenciones de nombres de Turbo Frames.

En el próximo capítulo, finalizaremos por completo nuestro editor de citas. ¡Nos vemos allí!

← anterior Siguiente →