

[← Back to the list of chapters](#)


Another CRUD controller with Turbo Rails

Published on March 07, 2022

In this chapter, we will build the CRUD controller for the dates in our quotes. It is the perfect opportunity to practice what we have learned since the beginning of the tutorial!

Sponsor this project on Github!

This tutorial is open-source forever. If you want to support my work, you can sponsor it on Github! I will invite you to a repository with the tutorial's source code.

 [Become a sponsor](#)

What we will build in the following three chapters

Now that our users can create, update, and delete quotes, it's time to make our quote editor do something useful!

In the next three chapters, we will work on the `Quotes#show` page. By the time we finish those three chapters, our users will be able to add multiple dates to their quotes. Those dates will have multiple line items, each one having a name, an optional description, a quantity, and a unit price.

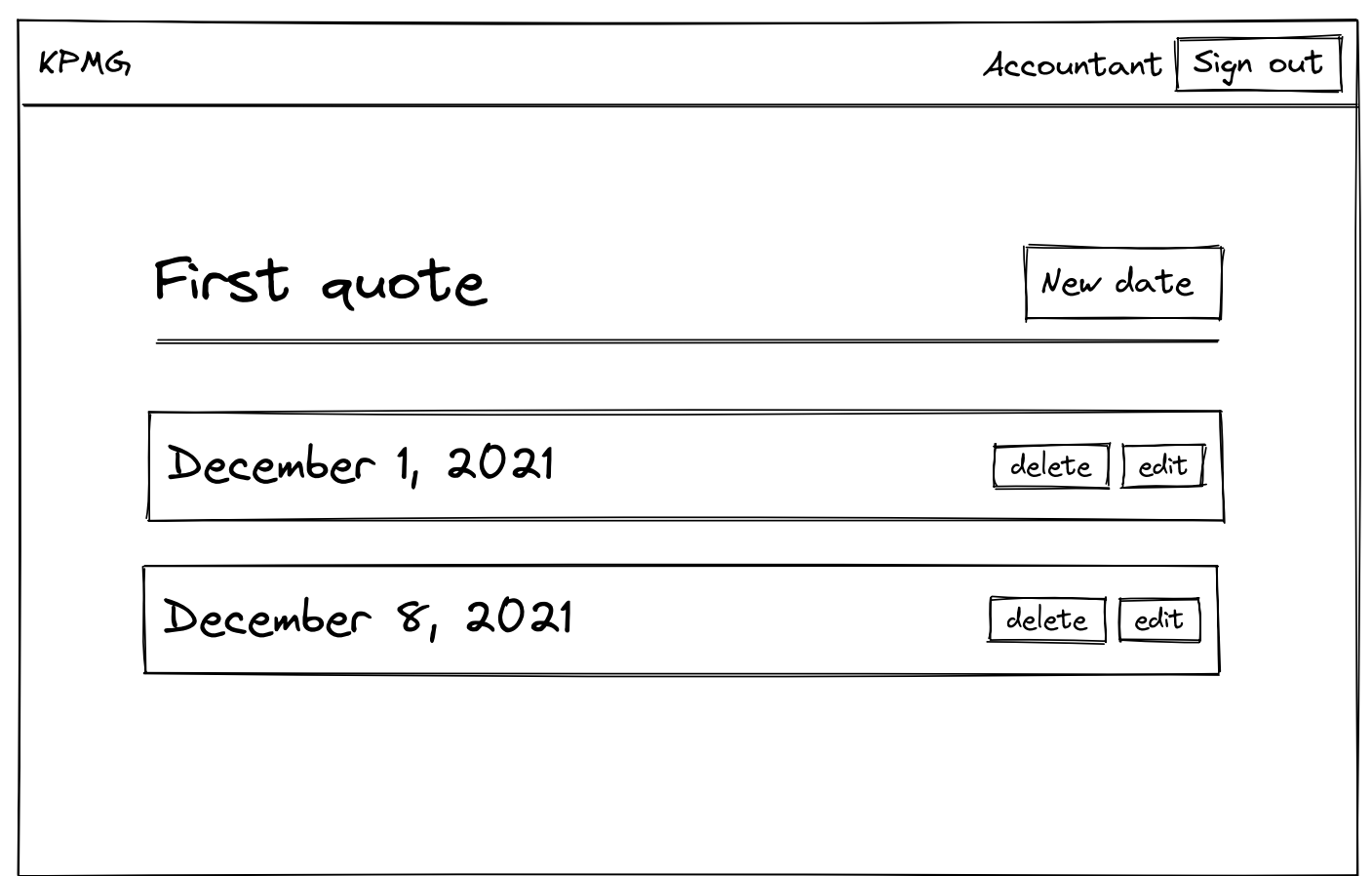
Before we start coding, let's try the [quote editor on hotrails.dev](#). Let's create a quote, then navigate to the `Quotes#show` page for this quote. We can create some dates and add line items to those dates. When we create, update, or delete line items, the total amount of the quote at the bottom of the page is updated.

Thanks to the power of Ruby on Rails with Turbo and what we learned in the previous chapters, it will be quite easy to do!

What we will build in this chapter

As in the **first chapter**, we will start by making the CRUD on dates **without using Turbo Frames and Turbo Streams**. We will always start this way as we need our controllers to work properly before making any improvement. It will then only require a few lines of code to slice the page into Turbo Frames.

Let's sketch what we will build first. When we visit the `Quotes#show` page, we should be able to see all the dates for the quote:



Sketch of the `Quotes#show` page with some dates

As we are first building the CRUD **without Turbo**, clicking on the "New date" link will take us to the `LineItemDates#new` page:

KPMG

Accountant

Sign out

⌕ Back to "First quote"

New date

Date

December 3, 2021

Create date

Sketch of the LineItemDates#new page

If we submit a valid form, we will be redirected to the Quotes#show page with the new date added. The dates will be **ordered in ascending order**:

KPMG

Accountant

Sign out

First quote

New date

December 1, 2021

delete

edit

December 3, 2021

delete

edit

December 8, 2021

delete

edit

Sketch of the Quotes#show page with the created date added to the list

If we decide to update the date we just created, we can click on the "Edit" link for this date to navigate to the LineItemDates#edit page:

KPMG

Accountant

Sign out

← Back to "First quote"

Edit date

Date

December 5, 2021

Update date

Sketch of the `LineItemDates#edit` page

If we submit a valid form, we will be redirected to the `Quotes#show` page with the date updated. The dates should still be **ordered in ascending order**:

KPMG

Accountant

Sign out

First quote

New date

December 1, 2021

delete

edit

December 5, 2021

delete

edit

December 8, 2021

delete

edit

Sketch of the `Quotes#show` page with the updated date

Last but not least, we can delete a date by clicking on the "Delete" link for this date. The date is then removed from the list.

Now that the requirements are clear, it's time to start coding!

Creating the model

Let's start by creating a model named `LineItemDate` with a date field and a reference to the quote it belongs to. We add this reference because each line item date belongs to a quote, and each quote has many line item dates. Let's generate the migration:

```
bin/rails generate model LineItemDate quote:references date:date
```

Before running the `rails db:migrate` command, we must add some constraints on the migration file:

- The date **must be present** on every `LineItemDate` record. We will add some validations in the model, but we should still add the `null: false` constraint to enforce the presence of the date at the database level even if, for some reasons, validations are skipped.
- We want to prevent a quote from having multiple times the same date. To enforce this at the database level, we add a uniqueness constraint for the couple `quote_id` and `date`.
- We want to be able to order line item dates by ascending date. We should add an index on the database field for performance reasons when we know we will use it for ordering purposes.

The final migration looks like this:

```
# db/migrate/XXXXXXXXXXXXX_create_line_item_dates.rb

class CreateLineItemDates < ActiveRecord::Migration[7.0]
  def change
    create_table :line_item_dates do |t|
      t.references :quote, null: false, foreign_key: true
      # Adding null: false constraint on date
      t.date :date, null: false

      t.timestamps
    end

    # Adding uniqueness constraint for the couple date and quote_id
    add_index :line_item_dates, [:date, :quote_id], unique: true
    # Adding index to the date field for performance reasons
    add_index :line_item_dates, :date
  end
end
```

```
end  
end
```

Now that our migration is ready, we can run it:

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

Let's add the associations and the corresponding validations on the `LineItemDate` model and a scope to order our line item dates in the ascending order:

```
# app/models/line_item_date.rb  
  
class LineItemDate < ApplicationRecord  
  belongs_to :quote  
  
  validates :date, presence: true, uniqueness: { scope: :quote_id }  
  
  scope :ordered, -> { order(date: :asc) }  
end
```

The validation line here enforces that:

- The date must be present on each line item date thanks to the `presence: true` option
- A quote won't be able to have the same date twice thanks to the `uniqueness: { scope: :quote_id }` option

Let's now add the `has_many` association on the `Quote` model:

```
# app/models/quote.rb  
  
class Quote < ApplicationRecord  
  has_many :line_item_dates, dependent: :destroy  
  
  # All the previous code...  
end
```

Our model layer is now complete! Let's next work on the routes.

Adding routes for line item dates

We want to perform all the seven CRUD actions on the `LineItemDate` model except two of them:

- We won't need the `LineItemDates#index` action as all dates will already be present on the `Quotes#show` page.
- We won't need the `LineItemDates#show` action as it would make no sense for us to view a single line item date. We always want to see the quote as a whole.

Those two exceptions are reflected in the routes file below:

```
# config/routes.rb

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

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

This is a very clean routes file as we only use RESTful resources. In the next section, we will add some fake data to our fixtures and our seeds to be able to design our `Quotes#show` page!

Designing line item dates

The `Quotes#show` page is currently empty. Let's start by adding some fake data to our first quote in the fixtures:

```
# test/fixtures/line_item_dates.yml

today:
  quote: first
  date: <%= Date.current %>

next_week:
```

```
quote: first
date: <%= Date.current + 1.week %>
```

We can now seed the database again by running the `bin/rails db:seed` command. Those seeds will enable us to design our quote editor with fake data. Let's now open the application on the `Quotes#show` page for the "First quote" and start designing the page. For now, our `Quotes#show` page markup looks like this:

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

<main class="container">
  <%= link_to sanitize("&larr; Back to quotes"), quotes_path %>
  <div class="header">
    <h1>
      <%= @quote.name %>
    </h1>
  </div>
</main>
```

To match our sketches, we need to add a link to the `LineItemDates#new` page and a line to render the collection of line item dates:

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

<main class="container">
  <%= link_to sanitize("&larr; Back to quotes"), quotes_path %>

  <div class="header">
    <h1>
      <%= @quote.name %>
    </h1>

    <%= link_to "New date",
              new_quote_line_item_date_path(@quote),
              class: "btn btn--primary" %>
  </div>

  <%= render @line_item_dates, quote: @quote %>
</main>
```


To render this collection of line item dates, we first need to retrieve those line items in the `QuotesController#show` action:

```
# app/controllers/quotes_controller.rb

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

  def show
    @line_item_dates = @quote.line_item_dates.ordered
  end

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

We used the `ordered` scope on our line item dates collection to **order them in ascending order**. Now that we properly retrieve our collection from the database, it's time to create the HTML for a single line item date:

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

<div class="line-item-date">
  <div class="line-item-date__header">
    <h2 class="line-item-date__title">
      <%= l(line_item_date.date, format: :long) %>
    </h2>

    <div class="line-item-date__actions">
      <%= button_to "Delete",
                  [quote, line_item_date],
                  method: :delete,
                  class: "btn btn--light" %>

      <%= link_to "Edit",
                  [:edit, quote, line_item_date],
                  class: "btn btn--light" %>
    </div>
  </div>
</div>
```

Most of the markup is wrapped inside a `div` with a `.line-item-date__header` class. This is because we will have a `.line-item-date__body` and a `.line-item-date__footer` class in the next chapter that will contain the line items of our quote and a link to create a new line item. In order to minimize the

amount of CSS/HTML we will have to change, we are taking a little shortcut here.

Note: I use polymorphic routes here for readability purposes. I want to make the lines of code a bit shorter on this tutorial, so you don't have to scroll the code sections. If you are not familiar with polymorphic routes, the two following lines are equivalent (but the second one is longer):

```
<%= button_to "Delete", [quote, line_item_date] %>
<%= button_to "Delete", quote_line_item_date_path(quote, line_item_date)
```

It is also possible to use them in controllers. For example, the two following lines for code are equivalent:

```
redirect_to @quote
redirect_to quote_path(@quote)
```

If you want to learn more about them, here is a link to [the documentation](#).

Now that we have the HTML markup, let's add a little bit of CSS to make our line item dates a bit nicer:

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

.line-item-date {
  margin-top: var(--space-xl);
  margin-bottom: var(--space-xxs);

  &__header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: var(--space-xs);
  }

  &__title {
    font-size: var(--font-size-xl);

    @include media(tabletAndUp) {
```

```
    font-size: var(--font-size-xxl);
  }
}

&__actions {
  display: flex;
  gap: var(--space-xs);
}
}
```

Let's not forget to import this new file inside our manifest file:

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

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

Everything should look nice now! We can inspect our design in the browser and see that it is *good enough*. It is now time to start working on our controller.

Our standard CRUD controller

Creating line item dates without Turbo

Now that our database schema, model, routes, markup, and design are done, it's time to start working on the controller. As mentioned in the introduction, we will first build a standard controller **without Turbo Frames and Turbo Streams; we will add them later**.

Our controller will contain all the seven actions of the CRUD except the `#index` and the `#show` actions. Let's start by making the `#new` and `#create` actions work:

```
# app/controllers/line_item_dates_controller.rb

class LineItemDatesController < ApplicationController
  before_action :set_quote

  def new
    @line_item_date = @quote.line_item_dates.build
```

```
end

def create
  @line_item_date = @quote.line_item_dates.build(line_item_date_params)

  if @line_item_date.save
    redirect_to quote_path(@quote), notice: "Date was successfully cre
  else
    render :new, status: :unprocessable_entity
  end
end

private

def line_item_date_params
  params.require(:line_item_date).permit(:date)
end

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

Our controller is very standard and should already work, but we are missing the `line_item_dates/new.html.erb` view and the `line_item_dates/_form.html.erb` partial. Let's add those two files to our application:

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

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

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

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

We don't need a fancy design for our `LineItemDates#new` page as we will use Turbo later to only extract the form from the page and insert it in the

Quotes#show page. However, it should still be usable for people using legacy browsers that don't support Turbo. Let's add the markup for our form:

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

<%= simple_form_for [quote, line_item_date], html: { class: "form line-i
  <% if line_item_date.errors.any? %>
    <div class="error-message">
      <%= line_item_date.errors.full_messages.to_sentence.capitalize %>
    </div>
  <% end %>

  <%= f.input :date, html5: true, input_html: { autofocus: true } %>
  <%= link_to "Cancel", quote_path(quote), class: "btn btn--light" %>
  <%= f.submit class: "btn btn--secondary" %>
<% end %>
```

With those two files created, let's test the creation of a new date in our browser. Everything works as expected!

Refactoring the error notification message

There is an opportunity here to refactor how we handle errors inside our forms. We may notice that we use exactly the same way to display errors on the quote form and on the line item date form:

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

<% if quote.errors.any? %>
  <div class="error-message">
    <%= quote.errors.full_messages.to_sentence.capitalize %>
  </div>
<% end %>
```

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

<% if line_item_date.errors.any? %>
  <div class="error-message">
    <%= line_item_date.errors.full_messages.to_sentence.capitalize %>
  </div>
<% end %>
```

We want the way we display errors to be consistent throughout our whole application. Let's create a helper that we will use in all our forms to ensure errors are always treated the same way:

```
# app/helpers/application_helper.rb

module ApplicationHelper
  # All the previous code

  def form_error_notification(object)
    if object.errors.any?
      tag.div class: "error-message" do
        object.errors.full_messages.to_sentence.capitalize
      end
    end
  end
end
```

For now, our application has only two helpers, so it's fine to keep those helpers in the `ApplicationHelper`. However, as our application grows, it will become important to organize those helpers into logical units. We could have a dedicated `FormHelper` for all form-related code! However, this is not our concern here, and we simply want to remove the duplication. With that helper in place, we can use it in both our views:

```
<%= app/views/line_item_dates/_form.html.erb %>

<%= All the previous code %>
<%= form_error_notification(line_item_date) %>
<%= All the previous code %>
```

```
<%= app/views/quotes/_form.html.erb %>

<%= All the previous code %>
<%= form_error_notification(quote) %>
<%= All the previous code %>
```

Thanks to this helper, our final line item date form looks like this:

```
<%= app/views/line_item_dates/_form.html.erb %>

<%= simple_form_for [quote, line_item_date], html: { class: "form line-i
```

```
<%= form_error_notification(line_item_date) %>

<%= f.input :date, html5: true, input_html: { autofocus: true } %>
<%= link_to "Cancel", quote_path(quote), class: "btn btn--light" %>
<%= f.submit class: "btn btn--secondary" %>
<% end %>
```

With our helper in place, we **only needed five lines of code to design our form**. If we test in the browser, we should see that everything still works as expected, after we refactored our code!

Let's just take a few seconds to fill the translations file with the text we want for the labels and the submit buttons:

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

en:
  simple_form:
    placeholders:
      quote:
        name: Name of your quote
    labels:
      quote:
        name: Name
      line_item_date:
        date: Date

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

With that file completed, the text of the submit button will be "Create date" when we create a `LineItemDate` and "Update date" when we update a `LineItemDate`.

Updating line item dates without Turbo

Now that our `#new` and `#create` actions are working, let's do the same work for the `#edit` and `#update` actions. Let's start with the controller:

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

  # All the previous code

  def edit
  end

  def update
    if @line_item_date.update(line_item_date_params)
      redirect_to quote_path(@quote), notice: "Date was successfully updated"
    else
      render :edit, status: :unprocessable_entity
    end
  end

  private

  def set_line_item_date
    @line_item_date = @quote.line_item_dates.find(params[:id])
  end

  # All the previous code
end
```

We know that we are going to need the `set_line_item_date` callback for the `#destroy` action as well, so we can anticipate and add it to the list of actions that require this callback.

Now that our `#edit` and `#update` actions are implemented, let's add the `LineItemDates#edit` view to be able to test in the browser:

```
<%# 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>

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

As we can notice, the `LineItemDates#edit` view is very similar to the `LineItemDates#new` view. Only the title changes. As we already built the form in the previous section, we are ready to experiment in the browser. Everything works as expected, only one more action to go!

Deleting line item dates without Turbo

The `#destroy` action is the simplest of all five as it doesn't require a view. We only need to delete the line item date and then redirect to the `Quotes#show` page:

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

  def destroy
    @line_item_date.destroy

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

  # All the previous code
end
```

Let's test it in our browser, and it works as expected! We could do one more thing to prevent users from deleting dates unintentionally. Let's add a confirmation message when they click the "Delete" button on a line item date. To confirm form submissions with Turbo, we need to add the `data-turbo-confirm="Your message"` to the `<form>` HTML tag.

Let's do this on the "Delete" buttons for line item dates:

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

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

```
<%= button_to "Delete",  
  quote_line_item_date_path(quote, line_item_date),  
  method: :delete,  
  form: { data: { turbo_confirm: "Are you sure?" } },  
  class: "btn btn--light" %>
```

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

The `button_to` helper generates a form in HTML. Here is what the HTML should look like if we inspect the DOM:

```
<form data-turbo-confirm="Are you sure?" class="button_to" method="post"  
  <input type="hidden" name="_method" value="delete" autocomplete="off">  
  <button class="btn btn--light" type="submit">Delete</button>  
  <input type="hidden" name="authenticity_token" value="long_token" auto  
</form>
```

The important part here is to notice that the `data-turbo-confirm` data attribute is on the `<form>` tag. When we click on the "Delete" button for a line item date, a confirmation alert now appears on the screen!

Our CRUD controller is now working as expected, but we now want all interactions to happen on the same page. Thanks to the power of Turbo, it will only require a few lines of code to slice our page into pieces that can be updated independently.

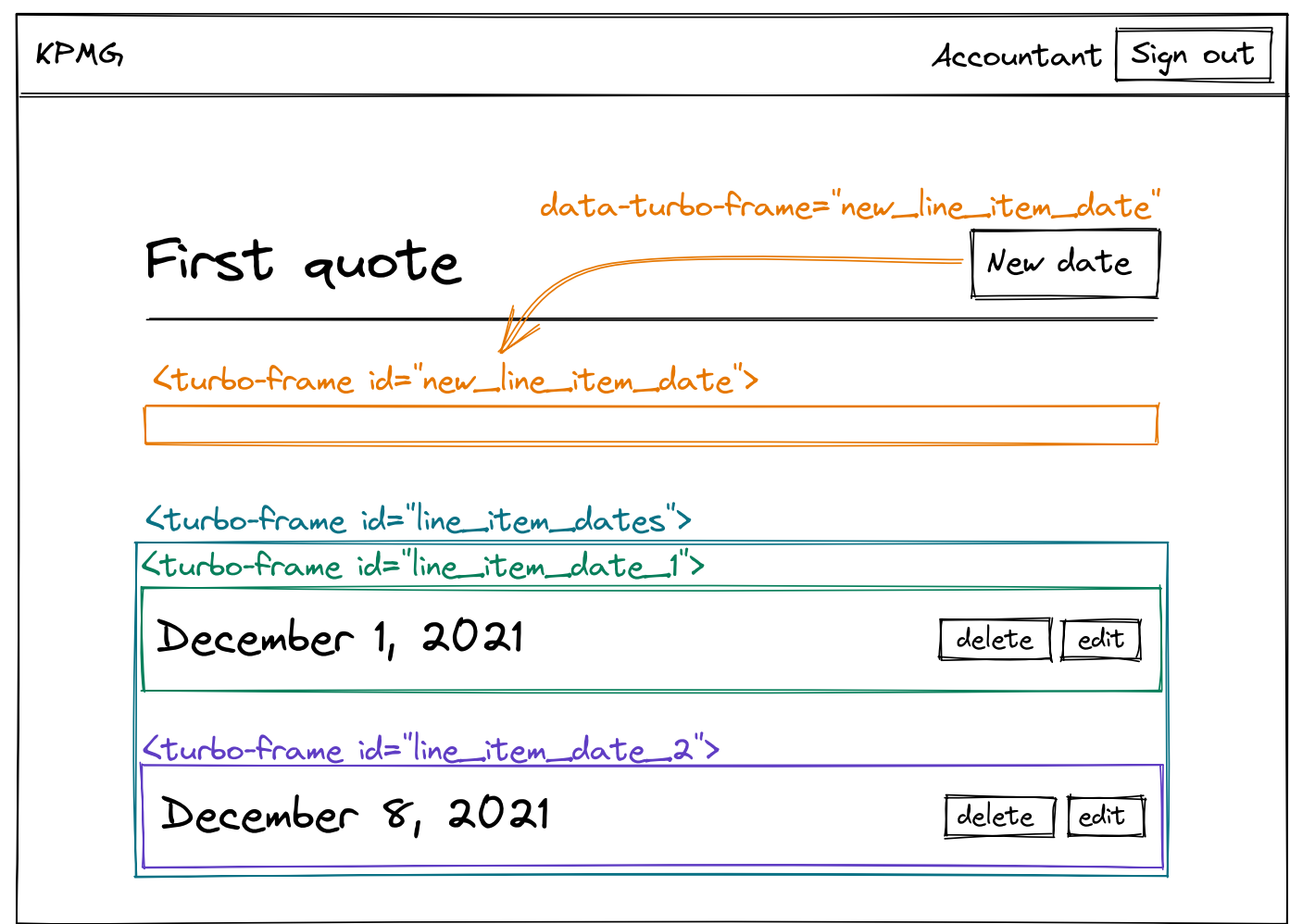
Adding Turbo Frames and Turbo Streams

Now that our CRUD controller is working as expected, it's time to improve the user experience so that all the interactions happen on `Quotes#show` page. The experience will be very similar to the one we had for the `Quotes#index` page.

Creating line item dates with Turbo

To be clear on the requirements, let's first sketch the desired behavior. What we want is that when a user visits the `Quotes#show` page and clicks on the "New date" button, the form appears on the `Quotes#show` page. We will do this with Turbo Frames, of course! To make it work, we have to connect the

"New date" link to an empty Turbo Frame thanks to the data-turbo-frame data attribute:



Sketch of the `Quotes#show` page with Turbo Frames

For Turbo to properly replace the Turbo Frame on the `Quotes#show` page, the Turbo Frame on the `LineItemDates#new` page must have the same id. By convention, this id is `new_line_item_date` as it is the `dom_id` of a new instance of our `LineItemDate` model:



Sketch of the `LineItemDates#new` page with a Turbo Frame

With those Turbo Frames in place, when a user clicks on the "New date" button, Turbo will successfully replace the empty Turbo Frame on the Quotes#show page with the Turbo Frame containing the form on the LineItemDates#new page.

KPMG

Accountant

Sign out

First quote

New date

<turbo-frame id="new_line_item_date">

Date

Create date

<turbo-frame id="line_item_dates">

<turbo-frame id="line_item_date_1">

December 1, 2021

delete

edit

<turbo-frame id="line_item_date_2">

December 8, 2021

delete

edit

Sketch of the Quotes#show page with the form from the LineItemDates#new page

When the user submits the form, we want the created line item date to be inserted on the Quotes#show page at the right position so that dates remain in the **ascending order**. If a quote already has a date that is before the one we just created, we should insert the created date's HTML in the correct order as described in the sketch below:

https://www.hotrails.dev/turbo-rails/turbo-rails-crud

20/36

KPMG

Accountant

Sign out

First quote

New date

< turbo-frame id="new_line_item_date">

Date

December 3, 2021

Create date

< turbo-frame id="line_item_dates">

< turbo-frame id="line_item_date_1">

December 1, 2021

delete

edit

< turbo-frame id="line_item_date_2">

December 8, 2021

delete

edit

< turbo-stream action="after" target="line_item_date_1">

< turbo-frame id="line_item_date_3">

December 3, 2021

delete

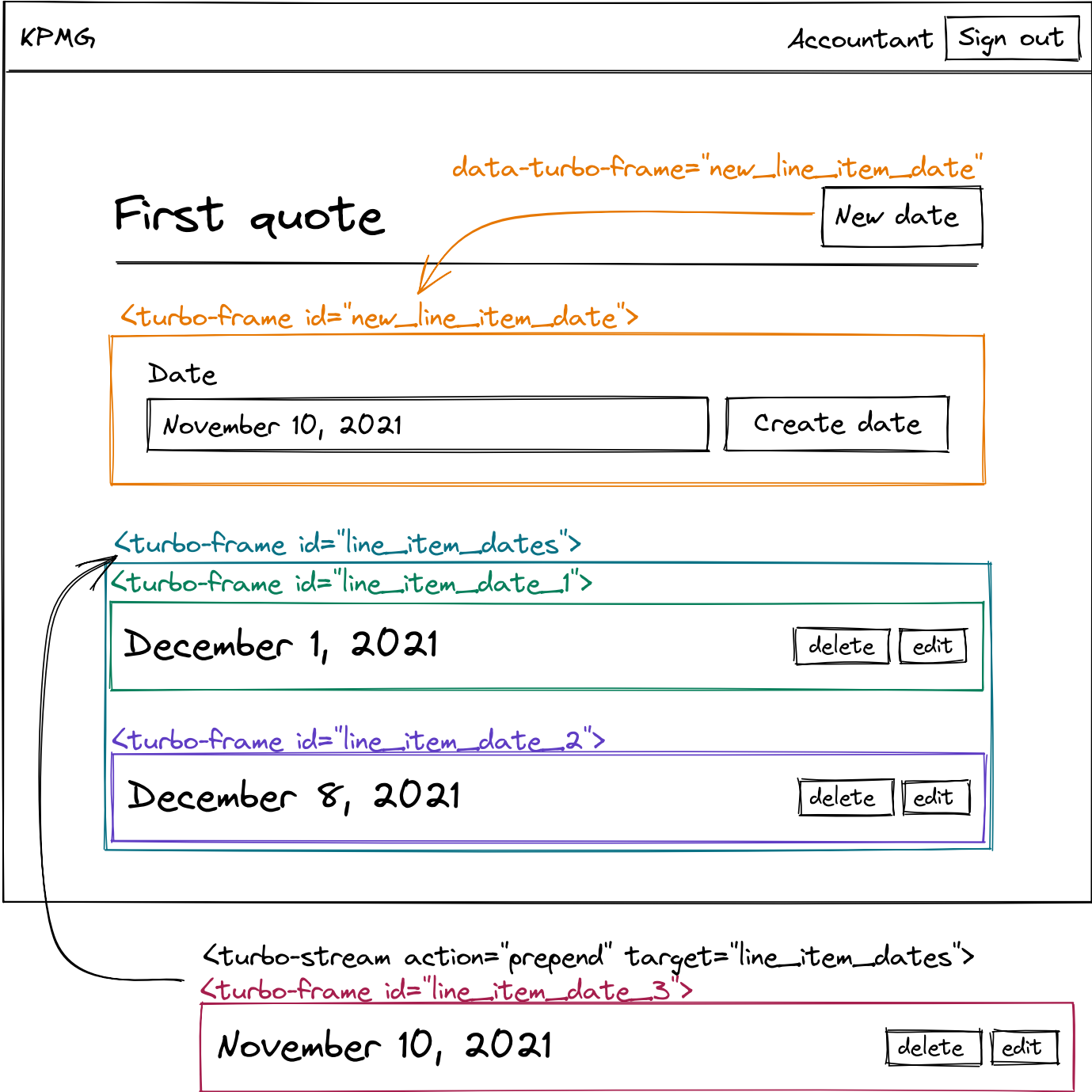
edit

Sketch of the created date being inserted right after the previous date

On the other hand, if the quote has no date earlier than the one that was just created, the HTML of the created date should be inserted at the beginning of the list:

https://www.hotrails.dev/turbo-rails/turbo-rails-crud

21/36



Sketch of the created date being inserted at the beginning of the list

Now that the requirements are clear, it should only take a few lines of code to make it real, thanks to the power of Turbo Frames and Turbo Streams!

Let's start working on the first part: making the form appear on the Quotes#show page when a user clicks on the "New date" button. This is the same work as on the Quotes#index page. On the Quotes#show page, let's add an empty Turbo Frame with the id of new_line_item_date and link the "New date" button to it:

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

<main class="container">
  <%= link_to sanitize("&larr; Back to quotes"), quotes_path %>

  <div class="header">
    <h1>
      <%= @quote.name %>
    </h1>
  </div>
</main>
```

```
</h1>

<%= link_to "New date",
  new_quote_line_item_date_path(@quote),
  data: { turbo_frame: dom_id(LineItemDate.new) },
  class: "btn btn--primary" %>

</div>

<%= turbo_frame_tag LineItemDate.new %>
<%= render @line_item_dates, quote: @quote %>
</main>
```

Now that we have a `turbo_frame_tag` with an id of `new_line_item_date` on the `Quotes#show` page, we need to have a `turbo_frame_tag` with the same id on the `LineItemDates#new` page for Turbo to be able to swap the two frames. Let's wrap our form in a Turbo Frame tag:

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

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

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

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

Now, when clicking on the "New date" button, the form appears in the `Quotes#show` page!

When we submit an **invalid** form, the errors appear on the page as expected. This is because, our `LineItemDates` controller renders the `LineItemDates#new` view on an invalid submission. As the form is within a Turbo Frame of id `new_line_item_date` and the rendered view contains a Turbo Frame with the same id, Turbo is smart enough to replace the frame automatically. As discussed in previous chapters, responses to invalid form submission should have the `unprocessable_entity` status for Turbo to display errors.

However, when we submit a **valid** form, as the form is within a Turbo Frame of id `new_line_item_date` and the response redirects to the `Quotes#show` page that contains an empty frame with this id, the form disappears as the Turbo Frame containing the form is replaced by an empty one. Nothing happens with the created date because Turbo is unable to guess what we should do with it. To satisfy the behavior we sketched above, we need a `create.turbo_stream.erb` template to instruct Turbo to:

1. Replace the Turbo Frame with id `new_line_item_date` with an empty one
2. Add the created line item date to the list **at the correct position**

While step 1 is exactly the same as what we did for the `Quotes#index` page, step 2 is more complicated. Indeed, to insert the newly created line item date at the correct position, we have to identify the correct position of the new line item date in the ordered list of dates. Let's learn how to do that. First, let's edit our `LineItemDatesController#create` action to respond to the `turbo_stream` format:

```
# app/controllers/line_item_dates_controller.rb

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

  def create
    @line_item_date = @quote.line_item_dates.build(line_item_date_params)

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

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

Now let's think about what we want to achieve. We want to insert the newly created line item date at the correct position in the ordered list. The order of

dates is **ascending**. That means that if our quote has dates before the new date, we should add our new date just *after* the latest of them. Otherwise, we should *prepend* the date right at the beginning of the list of dates.

Let's code this in our Turbo Stream view:

```
<%= app/views/line_item_dates/create.turbo_stream.erb %>

<%= Step 1: remove the form from the Quotes#index page %>
<%= turbo_stream.update LineItemDate.new, "" %>

<%= Step 2: add the date at the right place %>
<%= if previous_date = @quote.line_item_dates.ordered.where("date < ?", @
  <%= turbo_stream.after previous_date do %>
    <%= render @line_item_date, quote: @quote %>
  <%= end %>
<%= else %>
  <%= turbo_stream.prepend "line_item_dates" do %>
    <%= render @line_item_date, quote: @quote %>
  <%= end %>
<%= end %>

<%= render_turbo_stream_flash_messages %>
```

This piece of code could be improved, and we will refactor it soon, but let's understand what it does first. Step 1 is the same as for the quotes. We simply empty the Turbo Frame containing the form.

Step 2 is a bit more complicated. We first retrieve the line item date that is just before the one that we just created. If it exists, we add the new line item date right *after* this one. If not, we *prepend* the new line item date to the list of line item dates.

To make it work, we have to wrap our line item dates in a `turbo_frame_tag` with the `id` of `line_item_dates` in case we need to *prepend* the created line item date to the list:

```
<%= app/views/quotes/show.html.erb %>

<%= All the previous code... %>
```

```
<%= turbo_frame_tag "line_item_dates" do %>
  <%= render @line_item_dates, quote: @quote %>
<% end %>

<%= All the previous code... %>
```

We must also wrap each individual line item date in a `turbo_frame_tag`. This is because we must be able to identify each line item date by a unique id in case we need to insert the created date right *after* one of them:

```
<%= 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>
<% end %>
```

Let's test it in our browser, and it should work! Now let's refactor our code and add some tests. The first thing we need to do is to extract the *previous date* logic into the `LineItemDate` model:

```
# app/models/line_item_date.rb

class LineItemDate < ApplicationRecord
  # All the previous code...

  def previous_date
    quote.line_item_dates.ordered.where("date < ?", date).last
  end
end
```

Now let's replace the logic in our view with the method that we just created:

```
<%= app/views/line_item_dates/create.turbo_stream.erb %>

<%= Step 1: remove the form from the Quotes#index page %>
<%= turbo_stream.update LineItemDate.new, "" %>

<%= Step 2: add the date at the right place %>
<% if previous_date = @line_item_date.previous_date %>
  <%= turbo_stream.after previous_date do %>
```

```
<%= render @line_item_date, quote: @quote %>
<% end %>
<% else %>
  <%= turbo_stream.prepend "line_item_dates" do %>
    <%= render @line_item_date, quote: @quote %>
  <% end %>
<% end %>

<%= render_turbo_stream_flash_messages %>
```

Let's also test this method to make sure it works as expected:

```
# test/models/line_item_date_test.rb

require "test_helper"

class LineItemDateTest < ActiveSupport::TestCase
  test "#previous_date returns the quote's previous date when it exists" do
    assert_equal line_item_dates(:today), line_item_dates(:next_week).previous_date
  end

  test "#previous_date returns nil when the quote has no previous date" do
    assert_nil line_item_dates(:today).previous_date
  end
end
```

This is a minimal test but it gives us confidence that our application is working 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 almost everything is in place.

Updating line item dates 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 all the Turbo Frames that we need on the `Quotes#show` page as described in the sketch below:

KPMG

Accountant

Sign out

First quote

New date

<turbo-frame id="line_item_dates">

<turbo-frame id="line_item_date_1">

December 1, 2021

delete

edit

<turbo-frame id="line_item_date_3">

December 3, 2021

delete

edit

<turbo-frame id="line_item_date_2">

December 8, 2021

delete

edit

Sketch of the Quotes#show page

When clicking on the "Edit" link for the **second date** of our sketch that is within a Turbo Frame of id `line_item_date_3` , Turbo expects to find a Turbo Frame with the same id on the `LineItemDates#edit` page as described in the sketch below:

KPMG

Accountant

Sign out

← Back to "First quote"

Edit date

<turbo-frame id="line_item_date_3">

Date

December 3, 2021

Update date

Sketch of the LineItemDates#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 date on the `Quotes#show` page with the form from the `LineItemDates#edit`

page when clicking on the "Edit" link of a line item date:

KPMG

Accountant

Sign out

First quote

New date

<turbo-frame id="line_item_dates">

<turbo-frame id="line_item_date_1">

December 1, 2021

deleteedit

<turbo-frame id="line_item_date_3">

Date

December 3, 2021

Update date

<turbo-frame id="line_item_date_2">

December 8, 2021

deleteedit

Sketch of the Quotes#show with the edition form

When submitting the form, we will fall back into the same problem that we had for the #new and #create actions. We need to insert the updated line item date at the right position on the list. We will use the same method as we did in the create.turbo_stream.erb view. When submitting the form, we will first remove the date that we are currently updating from the view and then insert it at the right place, just like we did for the #create action. This is described in the sketch below:

KPMG

Accountant

Sign out

First quote

New date

<turbo-frame id="line_item_dates">

<turbo-frame id="line_item_date_1">

December 1, 2021

delete

edit

<turbo-frame id="line_item_date_3">

Date

December 15, 2021

Update date

<turbo-frame id="line_item_date_2">

December 8, 2021

delete

edit

<turbo-stream action="remove" target="line_item_date_3">

<turbo-stream action="after" target="line_item_date_2">

December 15, 2021

delete

edit

Sketch of the Quotes#show page when submitting the form

The final state of the Quotes#show page should have the dates ordered in ascending order:

KPMG

Accountant

Sign out

First quote

New date

<turbo-frame id="line_item_dates">

<turbo-frame id="line_item_date_1">

December 1, 2021

deleteedit

<turbo-frame id="line_item_date_2">

December 8, 2021

deleteedit

<turbo-frame id="line_item_date_3">

December 15, 2021

deleteedit

Sketch of the Quotes#show page with the dates in the right order

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 a date on the Quotes#show page. To do this, we only have to wrap the form on the LineItemDate#edit page within a Turbo Frame:

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

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

https://www.hotrails.dev/turbo-rails/turbo-rails-crud

31/36

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 doesn't guarantee our dates will be in the right order. To ensure the dates are always in the right order, we must make a Turbo Stream view that will be very similar to the one we make for our `#create` action in the previous section. Let's first enable our controller to render a Turbo Stream view:

```
# app/controllers/line_item_dates_controller.rb

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

Let's now create the `update.turbo_stream.erb` view. We will use exactly the same logic as for the `#create` action, except this time, we won't empty the Turbo Frame containing the form but instead remove it completely! Once removed, we can append it again at the right position, just like described in our sketches:

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

<%# Step 1: remove the form %>
<%= turbo_stream.remove @line_item_date %>

<%# Step 2: insert the updated date at the correct position %>
<% if previous_date = @line_item_date.previous_date %>
  <%= turbo_stream.after previous_date do %>
    <%= render @line_item_date, quote: @quote %>
  <% end %>
<% else %>
  <%= turbo_stream.prepend "line_item_dates" do %>
    <%= render @line_item_date, quote: @quote %>
  <% end %>
<% end %>
```



```
<%= render_turbo_stream_flash_messages %>
```

Let's test it in the browser, everything should work as expected! The last action we have to do is the easiest one, so let's keep going!

Destroying line item dates 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_item_dates_controller.rb

def destroy
  @line_item_date.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 simply have to remove the line item date and render the flash message:

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

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

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!

Testing our code with system tests

Our work wouldn't be really complete if we didn't add a few tests. We should always write at least system tests to make sure the happy path is covered. That

way if we make a mistake, we can correct it before pushing our code into production.

Let's add a system test file to test the happy path of the CRUD on our line item dates:

```
# test/system/line_item_dates_test.rb

require "application_system_test_case"

class LineItemDatesTest < ApplicationSystemTestCase
  setup do
    login_as users(:accountant)

    @quote = quotes(:first)
    @line_item_date = line_item_dates(:today)

    visit quote_path(@quote)
  end

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

    click_on "New date"
    assert_selector "h1", text: "First quote"
    fill_in "Date", with: Date.current + 1.day

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

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

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

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

end
```

We just added tests for the happy path of the creation, update and deletion of a line item date. We can run unit tests and system tests simultaneously with the `bin/rails test:all` command. Let's run them, and they should be all green!

Wrap up

In this chapter, we did another example of a CRUD controller on a resource just like we did for quotes, the only difference is that this time, we had to maintain the correct order of the dates on the `Quotes#show` page. To do this, we had to learn how to insert partials at a precise position in the DOM thanks to `turbo_stream.after`.

We also learned to prompt a confirmation alert to users thanks to the `data-turbo-confirm` data attribute that should be placed on `<form>` tags.

The next chapter will be the last big chapter, and we will talk about nested Turbo Frames. See you there!

[< previous](#)[next >](#)

Get notified when I write new articles

If you liked this article and want to keep up with Ruby on Rails and Hotwire, you can subscribe to my newsletter (no spam, no tracking, unsubscribe any time)!

Subscribe to the newsletter



Github



Twitter



Newsletter

Made with  remotely