Hotrails

← Back to the list of chapters

# Adding a quote total with Turbo Frames

*Published on March 15, 2022*

In this chapter, we will add a sticky bar containing the total of the quote. This total will be updated every time we create, update, or delete a line item.

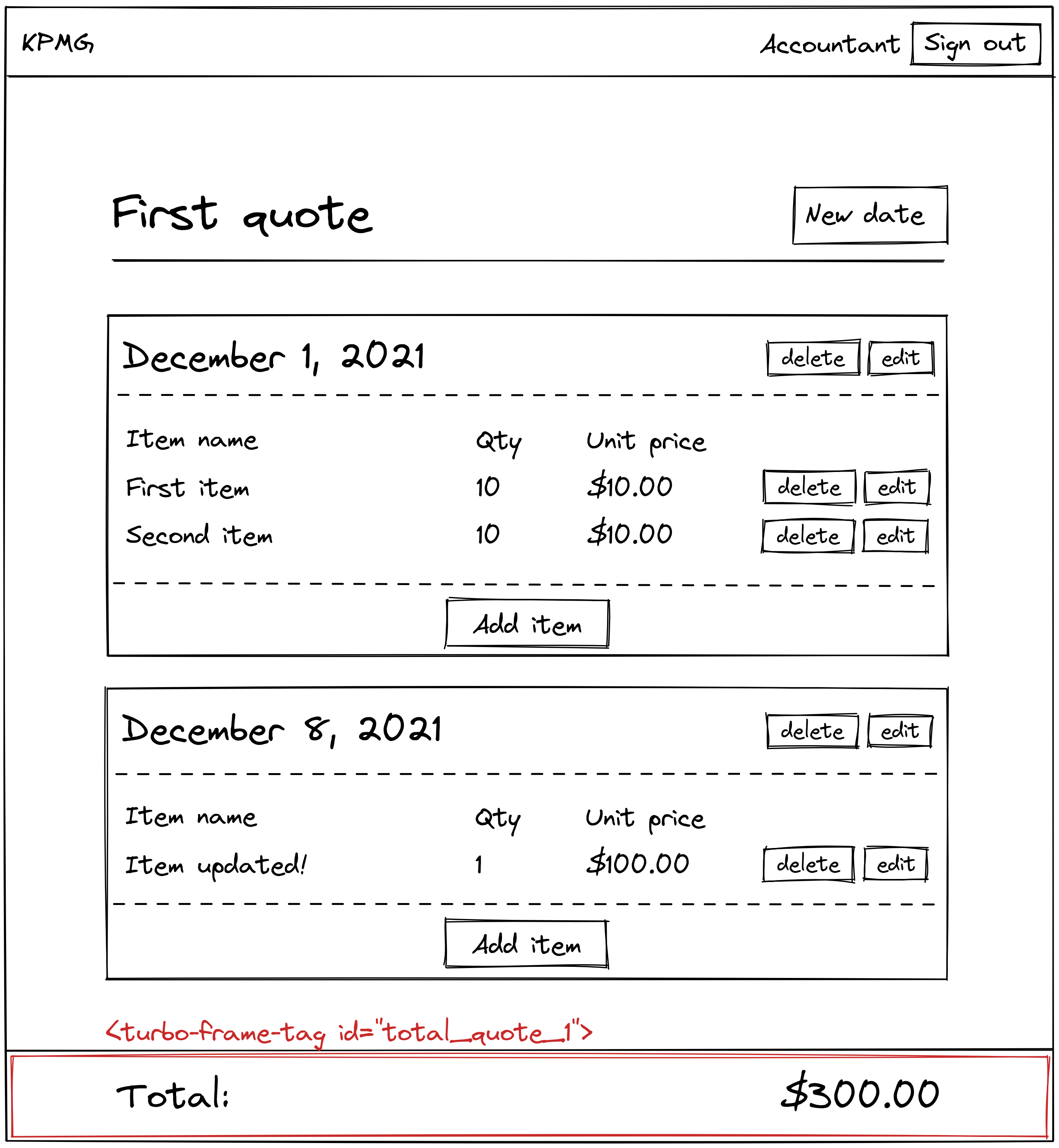
## What we will build in this chapter

In this last chapter, we will finalize our quote editor and this Turbo Rails tutorial! We will add a sticky bar to the `Quotes#show` page containing the total amount of the quote.

This total amount will be updated every time:

- a **line item** is created, updated, or deleted
- a **line item date** is deleted, as the line item date may contain some items

Here is a sketch of what our quote editor will look like:



*Sketch of the Quotes#show page with the total amount of the quote*

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

---

**Note**: Now that we have more experience with Turbo Frames, let's discuss when to use a `<turbo-frame>` tag or a `<div>`.

**We must use a Turbo Frame when we need Turbo to intercept clicks on links and form sumbissions for us**. For example, the content of the `line_items/_line_item.html.erb` partial must be wrapped inside a Turbo Frame because when we click on the "Edit" button for a **line item**, we want Turbo to replace the content of this Turbo Frame with the form extracted from the `LineItems#edit` page. This feature wouldn't work with a simple `div`.

On the other hand, **we don't *need* a Turbo Frame when we only target an id of the DOM in a Turbo Stream view**. For example, we didn't use a Turbo Frame for flash messages. Instead, we used a `div` with an id of `flash` to *prepend* new messages. We could have used a `div` with the id of `quotes` on the `Quotes#index` page instead of a Turbo Frame, because the `quotes` id is only used in Turbo Stream views to *prepend* quotes to the list. It never serves to intercept clicks on links and form submissions.

My *personal preference* here is always to use Turbo Frame tags as it makes it obvious that the id is used somewhere in a Turbo Stream view. This is why we use Turbo Frames everywhere in this tutorial. However, it is perfectly fine if you disagree and prefer to use a `div` when you don't need a Turbo Frame.

With that subtle difference explained, let's continue with the tutorial!

---

## Designing the sticky navbar

Before working with Turbo, let's add the sticky navbar to our `Quotes#show` page and take some time to design it:

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

<main class="container">
  <!-- All the previous code -->
</main>

<%= render "quotes/total", quote: @quote %>
```

Let's add the markup for the partial containing the quote total:

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

<footer class="quote-total">
  <div class="quote-total__inner container">
    <div>Total:</div>
    <div><%= number_to_currency quote.total_price %></div>
  </div>
</footer>
```

We need to add the `#total_price` method in the `Quote` model to be able to display the total amount for a quote:

```
# app/models/quote.rb

class Quote < ApplicationRecord
  # All the previous associations
  has_many :line_items, through: :line_item_dates

  # All the previous code

  def total_price
    line_items.sum(&:total_price)
  end
end
```

As we can see in the code we just added, the total price of a quote is the sum of the total prices of each **line item** of this quote. Let's make sure we implement the `#total_price` method for a single **line item**:

```
# app/models/line_item.rb

class LineItem < ApplicationRecord
  # All the previous code

  def total_price
    quantity * unit_price
  end
end
```

The total price of a single **line item** is its quantity times its unit price. Let's quickly add a few tests for the two methods we just implemented:

```
# test/models/line_item_test.rb

require "test_helper"

class LineItemTest < ActiveSupport::TestCase
  test "#total_price returns the total price of the line item" do
    assert_equal 250, line_items(:catering_today).total_price
  end
end
```

```ruby
# test/models/quote_test.rb

require "test_helper"

class QuoteTest < ActiveSupport::TestCase
  test "#total_price returns the sum of the total price of all line item
    assert_equal 2500, quotes(:first).total_price
  end
end
```

Let's test in the browser, and we should now see the total price of the quote on our page. Let's style our sticky bar with CSS before moving to the next section:

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

.quote-total {
  position: fixed;
  bottom: 0;
  width: 100%;

  font-size: var(--font-size-xl);
  font-weight: bold;
  background-color: var(--color-white);
  box-shadow: var(--shadow-large);

  padding-top: var(--space-xs);
  padding-bottom: var(--space-xs);

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

  &__inner {
    display: flex;
    align-items: center;
    justify-content: space-between;
  }
}
```

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

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

@import "components/quote_total";
```

Let's test it in our browser, and everything should work as expected except that we need a bit more margin in the bottom of our `Quotes#show` page for the last **line item date** not to overlap with the sticky bar containing the quote total. Let's add a utility class to add bottom margin when required:

```
// app/assets/stylesheets/utilities/_margins.scss

.mb-xxxxl {
  margin-bottom: var(--space-xxxxl);
}
```

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

```
// app/assets/stylesheets/application.sass.scss
// All the previous code

// Utilities
@import "utilities/margins";
```

We can now use this utility class in our `Quotes#show` view:

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

<main class="container mb-xxxxl">
  <!-- All the previous code -->
</main>

<%= render "quotes/total", quote: @quote %>
```

Let's test in the browser. The quote's total amount is displayed as expected. However, the total isn't updated when we create, update or delete a **line item**. We have the same issue when we delete a **line item date** that contains some **line items**. We will solve these issues with Turbo Stream views in the next section.

## Updating our view with Turbo Streams

As mentioned in the previous section, we need to update the quote total when:

- We create, update or delete a **line item**
- We destroy a **line item date** as it may contain some **line items**

Updating independent pieces of the page is easy with Turbo Streams. Let's simply re-render the `quotes/_total.html.erb` partial every time we perform one of those operations:

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

<%# All the previous code %>

<%= turbo_stream.update dom_id(@quote, :total) do %>
  <%= render "quotes/total", quote: @quote %>
<% end %>
```

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

<%# All the previous code %>

<%= turbo_stream.update dom_id(@quote, :total) do %>
  <%= render "quotes/total", quote: @quote %>
<% end %>
```

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

<%# All the previous code %>

<%= turbo_stream.update dom_id(@quote, :total) do %>
  <%= render "quotes/total", quote: @quote %>
<% end %>
```

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

<%# All the previous code %>

<%= turbo_stream.update dom_id(@quote, :total) do %>
  <%= render "quotes/total", quote: @quote %>
<% end %>
```

Let's not forget to add a Turbo Frame tag or a `div` to wrap the sticky navbar containing the quote total:

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

<main class="container mb-xxxxl">
  <!-- All the previous code -->
</main>

<%= turbo_frame_tag dom_id(@quote, :total) do %>
  <%= render "quotes/total", quote: @quote %>
<% end %>
```

We can now test in the browser, and everything should work as expected. Slicing our page in independent pieces is very easy, thanks to Turbo!

## Adding system tests to our application

Let's not forget to add a few more lines to our system tests to ensure our application behaves as expected. We want to make sure that the total amount of the quote is correctly updated right after we create, update, or delete a **line item**:

```
# test/system/line_items_test.rb

require "application_system_test_case"

class LineItemsTest < ApplicationSystemTestCase
  # All the previous code

  test "Creating a new line item" do
    # All the previous code
    assert_text number_to_currency(@quote.total_price)
  end

  test "Updating a line item" do
    # All the previous code
    assert_text number_to_currency(@quote.total_price)
  end

  test "Destroying a line item" do
    # All the previous code
    assert_text number_to_currency(@quote.total_price)
  end
end
```

We also want to make sure the quote total is updated when we destroy a **line item date** containing **line items**:

```ruby
# test/system/line_item_dates_test.rb

require "application_system_test_case"

class LineItemDatesTest < ApplicationSystemTestCase
  # We must include this module to be able to use the
  # `number_to_currency` method in our test
  include ActionView::Helpers::NumberHelper

  # All the previous code

  test "Destroying a line item date" do
    # All the previous code
    assert_text number_to_currency(@quote.total_price)
  end
end
```

Let's run all of our tests one more time with the `bin/rails test:all` command. They should all be green!

## Conclusion

This was the last chapter of this Turbo Rails tutorial inspired by a real-world project I had to work on! I hope you enjoyed reading it as much as I enjoyed writing it! I also hope that you learned a lot about Turbo!

If you want to deploy your application to Heroku, here is a YouTube demo by DHH with a timestamp to where he talks about deployment on Heroku.

If you want to learn more about Hotwire and Turbo, I recommend having a look a the hotwire-example-template repository by Sean Doyle at Thoughtbot. All the examples in the repository are practical examples you will encounter in almost every Ruby on Rails application.

If you enjoyed this tutorial and got a lot of value from it, you can sponsor it directly from my Github profile. I will use your money to maintain this website and, maybe, create some more tutorials in the future!

Feel free to share this tutorial with everyone who might be interested!

← previous