[Lesson 6.2: Carts (Sessions) and Line Items](#h.r1rl1u9amba8)

[Gem of the Day](#h.4ae365jxvgpm)

[Navbar and Shop Listing by Category](#h.6bxalac9rbpf)

[Shopping Cart Set Up](#h.56f8vpnqhn9x)

[Cart Model](#h.vyle25sbe21k)

[Line Items](#h.6916v0spyzfp)

[Add to Cart Button](#h.71svnjjcfye0)

[Cart View](#h.a5hn0xu6hdqc)

[Multiple Items of Same Product](#h.1pmee3hgqd66)

[Emptying Cart](#h.idinvgxvokne)

[Homework](#h.qqr1pa89uyis)

# Lesson 6.2: Carts (Sessions) and Line Items

## Gem of the Day

Friendly ID <https://github.com/norman/friendly_id>

## Navbar and Shop Listing by Category

Let’s update the navbar to include links to our storefront. The first thing we will do is update the logic in our controller to handle a cat\_id parameter that will modify what is in @products.

def all\_items

if params[:cat\_id]

@category = Category.find(params[:cat\_id])

@products = Product.where(category\_id: params[:cat\_id])

else

@products = Product.all

end

end

Now, let’s add our shop dropdown menu right after our home page link in the navbar - we are creating links to the all\_items page and passing in a parameter for the category ID which we will then use to only show those items on that page.

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-expanded="false">Shop<span class="caret"></span></a>

<ul class="dropdown-menu" role="menu">

<% Category.all.each do |cat| %>

<li><%= link\_to cat.name.capitalize, storefront\_all\_items\_path(cat\_id: cat.id) %></li>

<% end %>

<li class="divider"></li>

<li><%= link\_to "All Items", storefront\_all\_items\_path %></li>

</ul>

</li>

Finally, let’s add a heading2 to list which category is being shown on the all\_items page:

<% if params[:cat\_id] %>

<h3>All <%= @category.name.capitalize %></h3>

<% else %>

<h3>All Items</h3>

<% end %>

Commit your changes!

## Shopping Cart Set Up

Yesterday, we mentioned that our cart would be handled by 2 resources - orders and line items:

LineItem

* product\_id
* quantity
* line\_item\_total
* order\_id

Order

* sub\_total
* sales\_tax
* grand\_total
* customer\_id\*

\* how we'll eventually tie a user (customer) to what they want to buy

### Cart Model

Well, it’s a bit more complicated than that. We also need a temporary cart that will store info from the current session, then when they checkout, the app will generate the line items and order. To do this, we’ll keep a cart in the database and store its unique identifier, card.id, in the **session**. Every time a request comes in, we can recover the identity from the session and use it to find the cart in the database. We’ll talk more about sessions once we implement one little bit more.

To create the cart:

rails g scaffold Cart

rake db:migrate

Rails makes the current session look like a hash to the controller, so we will store the ID of the cart in the session by indexing it with the symbol :cart\_id. To do this, we create a new file in app/controllers/concerns called **current\_cart.rb**:

module CurrentCart

extend ActiveSupport::Concern

private

def set\_cart

@cart = Cart.find(session[:cart\_id])

rescue ActiveRecord::RecordNotFound

@cart = Cart.create

session[:cart\_id] = @cart.id

end

end

The set\_cart method essentially grabs the cart\_id from the session object and then attempts to find a cart associated with that id. If one doesn’t exist, it creates a new cart.

We placed this method in a CurrentCart module and marked it as private so that we can share this common code among controller and prevent Rails from making it an action on the controller.

### Line Items

A cart will contain the line items until we generate the order. Let’s scaffold the line items now:

rails g scaffold LineItem product:references cart:belongs\_to quantity:integer

Before migrating, edit the migration so that the quantity defaults to 1:

t.integer :quantity, default: 1

Now, run rake db:migrate.

At the model level, there is no difference between a simple reference and a “belongs\_to” relationship.

Let’s finish setting up the other side of the relationship in the Cart model:

class Cart < ActiveRecord::Base

has\_many :line\_items, dependent: :destroy

end

The dependent: :destroy part indicates that the existence of the line items is dependent on the existence of the cart. If we destroy a cart (delete it from the database), then we want Rails to also destroy any line items associated with that cart.

Finally, let’s set up the other side of the relationship in the Product model:

class Product < ActiveRecord::Base

belongs\_to :category

has\_many :line\_items

before\_destroy :ensure\_not\_referenced\_by\_any\_line\_item

...

private

# ensure that there are no line items referencing this product

def ensure\_not\_referenced\_by\_any\_line\_item

if line\_items.empty?

return true

else

errors.add(:base, 'Line Items present')

return false

end

end

end

end

Here we defined a **hook** method that Rails will call automatically whenever Rails attempts to destroy a row for Product in the database. The **errors** object is the same place that data validation errors are stored.

### Add to Cart Button

Now let’s add an Add to Cart button for each product. Because we scaffolded Line Items, we can use the Create method that the scaffold generated. Each time we add to a cart, we are actually creating Line Items. Rails will handle creating the cart if it does not already exist.

When we talked about HTTP verbs and the scaffold methods, we learned that the create method uses the POST verb. The Rails link\_to method uses GET by default, but the button\_to method uses POST, so we will use that here.

So, in the all\_items view, just add this button to the panel:

<%= button\_to 'Add to Cart', line\_items\_path(product\_id: product), class: "btn btn-success btn-sm" %>

We have to make a few changes before that functionality will work. In the **Line Items Controller**, we need to find the shopping cart for the current session or create one if it doesn’t already exist. Then, we need to add that product to the cart and display the cart contents. We built part of this code in our CurrentCart module, but now we need to use it.

class LineItemsController < ApplicationController

include CurrentCart

before\_action :set\_cart, only: [:create]

Now, scroll down to the create method so that we can add just a few lines of code to recognize the product\_id parameter and build the cart (we’re also removing the notice about the line item being created):

def create

product = Product.find(params[:product\_id])

@line\_item = @cart.line\_items.build(product: product)

respond\_to do |format|

if @line\_item.save

format.html { redirect\_to @line\_item.cart }

format.json { render :show, status: :created, location: @line\_item }

else

format.html { render :new }

format.json { render json: @line\_item.errors, status: :unprocessable\_entity }

end

end

end

### Cart View

Last but not least, let’s add some data to the cart show view:

<h2>Your Shopping Cart</h2>

<table class="table">

<thead>

<tr>

<th>Product</th>

<th>Quantity</th>

<th>Unit Price</th>

<th>Total Price</th>

</tr>

</thead>

<tbody>

<% @cart.line\_items.each do |item| %>

<tr>

<td><%= item.product.name %></td>

<td><%= item.quantity %> &times;</td>

<td><%= number\_to\_currency item.product.price %></td>

<td><%= number\_to\_currency item.total\_price %></td>

</tr>

<% end %>

</tbody>

</table>

<h3>Order Subtotal: <%= number\_to\_currency @cart.subtotal %></h3>

<%= link\_to "Continue Shopping", storefront\_all\_items\_path, class: "btn btn-success btn-sm" %>

<%= button\_to "Empty Cart", @cart, method: :delete, data: {confirm: "Are you sure?"}, class: "btn btn-danger btn-sm", form: {style: 'display:inline-block;'} %>

To make this work, we need to create the **total\_price** methods for both line items and carts, in their respective models:

class LineItem < ActiveRecord::Base

...

def total\_price

product.price \* quantity

end

end

class Cart < ActiveRecord::Base

...

def subtotal

line\_items.to\_a.sum { |item| item.total\_price }

end

end

Go ahead and run your server and test out adding an item to the cart. If everything works as expected, commit your changes!

### Multiple Items of Same Product

Now, let’s make it so that when we add the same product to the cart, it doesn’t generate a new line item, it only updates the quantity. We need an **add\_product** method in our **Cart Model** to handle this:

class Cart < ActiveRecord::Base

has\_many :line\_items, dependent: :destroy

def add\_product(product\_id)

current\_item = line\_items.find\_by(product\_id: product\_id)

if current\_item

current\_item.quantity += 1

else

current\_item = line\_items.build(product\_id: product\_id)

end

current\_item

end

end

We also need to update the **create** method in the **Line Items Controller** to take in this method:

def create

product = Product.find(params[:product\_id])

@line\_item = @cart.add\_product(product.id)

Now when we add a duplicate product, it updates the quantity on the line item rather than creating a new line.

### Emptying Cart

Let’s update the redirect after emptying a cart. Go to the Carts Controller, destroy method. We are going to make sure that the user is destroying their own cart first, and then remove the cart from the session before redirecting to all\_items

def destroy

@cart.destroy if @cart.id == session[:cart\_id]

session[:cart\_id] = nil

respond\_to do |format|

format.html { redirect\_to storefront\_all\_items\_url, notice: 'Your cart is currently empty.' }

format.json { head :no\_content }

end

end

Commit your changes!

## Homework

Make things prettier. Add the ability to delete individual line items from the cart - add buttons to each row that are linked to the destroy method for Line Items. Optional: Change dog show views to modals on Dog index.

Learn more about sessions here: <http://www.justinweiss.com/blog/2015/03/17/how-rails-sessions-work/>

Learn more about session, cookies, and authentication here: <http://www.theodinproject.com/ruby-on-rails/sessions-cookies-and-authentication>