[Lesson 6.3: Cart Security & Orders](#h.imim044j4wsb)

[Gem of the Day](#h.hl1sq9jaj0ln)

[Review: What We Did So Far](#h.6mw2vff0i5wd)

[Security](#h.gfiy9g54qkxv)

[Checking Out](#h.b30jxdfy5qvj)

[What is an Order?](#h.i9ohjjv6et67)

[Checkout Button](#h.p0uogj4z24dh)

[New Order (Checkout) View](#h.urqhsprguo9o)

[Creating the Order](#h.kso5y58lnbn7)

[Homework](#h.i3mpj7rf4yo4)

# Lesson 6.3: Cart Security & Orders

## Gem of the Day

Stripe vs Active Merchant: <https://www.ruby-toolbox.com/categories/Payments>

Gant is a big fan of Stripe - it’s popular and a gateway.

Stripe: <https://github.com/stripe/stripe-ruby>

Active Merchant: <https://github.com/Shopify/active_merchant>

## Review: What We Did So Far

6.1

* Branch shop
* Products and paperclip image
* Categories and seeds
* All items view with panels and modals

6.2

* Navbar and filter categories
* Cart scaffold
* Module for setting cart
* Line items scaffold, relationships, validations to prevent orphan records
* Add cart button (line item create which also uses the set cart method from the module)
* Cart view and totals
* Fixed multiples

## Security

Let’s take a minute to talk about security and do a few things to make our app more secure. First, try navigating to a cart that doesn’t exist:

<http://localhost:3000/carts/fakecart>

You can see that the app triggers an error - in production that error wouldn’t be clear to the user. So, let’s write some code to trigger some messages in the Carts Controller:

class CartsController < ApplicationController

before\_action :set\_cart, only: [:show, :edit, :update, :destroy]

rescue\_from ActiveRecord::RecordNotFound, with: :invalid\_cart

…

private

…

def invalid\_cart

logger.error "Attempt to access invalid cart #{params[:id]}"

redirect\_to storefront\_all\_items\_path, notice: 'Invalid cart'

end

end

Reload that invalid page and confirm that it redirects to the all\_items path, gives an error on the page, and also logs the attempt in your log.

That’s an improvement, but our biggest problem is that currently anyone can access anyone else’s cart as long as they guess a valid id. Let’s change that by **disallowing the cart id** parameter in the **line items controller**. Scroll down to the line item params and simple delete “, :cart\_id”.

## Checking Out

So, we’ve creating a lot of cool stuff, but a user still can’t check out and actually buy the products in their cart. Today we will add a simplified version of that functionality.

### What is an Order?

An order, in it’s most basic sense, is a set of line items along with details of the purchase transaction like customer contact information and payment option.

Let’s create the Order scaffold and add order ID to the Line Items model:

rails g scaffold Order name:string address:text pay\_type:string user:references

rails g migration add\_order\_to\_line\_item order:references

Double-check your migrations, then run rake db:migrate. Then, set up our relationships - remember that dependent destroy will delete all line items of an order if the order itself is deleted:

class **Order** < ActiveRecord::Base

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

belongs\_to :user

class **LineItem** < ActiveRecord::Base

belongs\_to :order

belongs\_to :product

belongs\_to :cart

class **User** < ActiveRecord::Base

# Include default devise modules. Others available are:

# :confirmable, :lockable, :timeoutable and :omniauthable

devise :database\_authenticatable, :registerable,

:recoverable, :rememberable, :trackable, :validatable

has\_many :orders

Then, commit your changes!

### Checkout Button

To start the checkout process, we need a user to click a Checkout button from their shopping Cart view. Let’s add this button before the Continue Shopping button on the Cart show view:

<%= link\_to "Checkout", new\_order\_path, class: "btn btn-info btn-sm" %>

Now, we need to jump in the Orders controller to do a few things:

1. make the user log in
2. give the order access to the cart
3. make sure there is something in the cart
4. if there is something in the cart, give them a form to enter checkout details like contact and payment

For #1 and #2, let’s add this to the top of the Orders controller:

class OrdersController < ApplicationController

include CurrentCart

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

before\_action :authenticate\_user!

For #3 and #4, let’s add some code to the New method in the Order Controller - basically, if nothing is in the cart, we redirect them to the storefront with a notice, and we return immediately - in other words, we exit the checkout:

def new

if @cart.line\_items.empty?

redirect\_to storefront\_all\_items\_url, notice: "Your cart is empty."

return

end

@order = Order.new

@order.user\_id = current\_user.id

end

### New Order (Checkout) View

Now that we can move to the new order form, let’s make it look better. First, let’s go to the Order New view and add an h1 and h3, and edit the back button to go back to the cart rather than the list of orders:

<h1>Checkout</h1>

<h3>Please Enter Your Details</h3>

<%= render 'form' %>

<%= link\_to 'Back to Cart', cart\_path(session[:cart\_id]) %>

Now, let’s create an array of payment types in the Order model:

PAYMENT\_TYPES = ["Check", "Credit Card", "PayPal"]

Then, let’s edit the form itself:

<%= form\_for(@order) do |f| %>

...

<div class="field form-group">

<%= f.label :name %><br>

<%= f.text\_field :name, class: "form-control", placeholder: "Jane Smith" %>

</div>

<div class="field form-group">

<%= f.label :address %><br>

<%= f.text\_area :address, class: "form-control", placeholder: "100 St. Charles Ave\nNew Orleans, LA 70115" %>

</div>

<div class="field form-group">

<%= f.label :pay\_type %><br>

<%= f.select :pay\_type, Order::PAYMENT\_TYPES, {prompt: "Select a payment method"}, {class: "form-control"} %>

</div>

<div class="actions">

<%= f.submit "Place Order", class: "btn btn-success" %>

</div>

<% end %>

Finally, let’s add some validation to make sure all fields are filled out - go to the Order model:

class Order < ActiveRecord::Base

belongs\_to :user

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

validates :name, :address, :user\_id, presence: true

PAYMENT\_TYPES = ["Credit Card", "PayPal", "Check", "Monopoly Money"]

validates :pay\_type, inclusion: PAYMENT\_TYPES

end

Commit your changes!

## Creating the Order

Now we need to update the Create method in the Order controller to accommodate our needs:

* Grabbing the form data to populate a new order object
* Add the line items from our cart to the order using a method that we will create in the step after this
* Validate and save the order - if there is a failure, let them edit.
* After order is successfully saved, delete the cart and redisplay the storefront with a notice that the order was placed.

def create

@order = Order.new(order\_params)

@order.user\_id = current\_user.id

@order.add\_line\_items\_from\_cart(@cart)

respond\_to do |format|

if @order.save

Cart.destroy(session[:cart\_id])

session[:cart\_id] = nil

format.html { redirect\_to storefront\_all\_items\_url, notice: 'Thank you for your order!' }

format.json { render :show, status: :created, location: @order }

else

format.html { render :new }

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

end

end

end

Now, we need to actually create the **add\_line\_items\_from\_cart** method in the Order Model which will do two things while looping through the line\_items:

* Set the cart\_id to nil to prevent the line\_item from disappearing when we destroy the cart (after we put it in the order).
* Add the line item to the collection of line items for the order

def add\_line\_items\_from\_cart(cart)

cart.line\_items.each do |item|

item.cart\_id = nil

line\_items << item

end

end

Now let’s check that it is working by checking out. First try checking out without entering anything on the new order page. Then, enter info and checkout. That’s nice, but did it do anything? Let’s check in rails console using ap Order.all, then ap LineItem.all. Pretty cool! Let’s commit our changes.

Classroom Challenge:

* Update the orders index to show the user email.
* Update the orders view to list the line items associated with an order.

## Homework

* Add a shopping cart button to navbar. Bonus - only have it show up if it exists.
* Add an Orders link to the navbar. We will change links based on admin status later.
* Think of a personal app you would like to work on over the next 2 weeks.
* If you have not already, finish yesterday’s homework.