Part3 Add AJAX, Checkout, User authentication

– Add AJAX

AJAX is a technique that javascript in browser and send async request to server and use the response to update the browser without refreshing the whole page.

change the “add to cart” button so that it will send AJAX request,

```
<%= button_to 'Add to Cart', lineitems_path(product_id: product), class: 'add_to_cart',
remote:true %>
```

the server need to send back a javascript code that will be executed in browser to replace the HTML shopping cart content with new content.

since when add_to_cart is clicked, it posts to create action of lineitem controller, and request javascript format, we need to add a js format case:

```
  def create

    product = Product.find(params[:product_id])
    @line_item = @cart.add_item(product.id)

    #@line_item = LineItem.new(line_item_params)

    respond_to do |format|
      if @line_item.save
         format.html { redirect_to shopper_url }
          format.js
         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
```

if you do not specify redirect_to or render for .js format, by default, it will look for “create“ view to render

create a create view:
app/views/lineintems/create.js.erb with following content:

```
$('#cart').html("<%= escape_javascript render(@cart)%>”);
```

this is jQuery code. it will be returned to the browser and ‘eval’ed by the browser. javascript will be run inside browser and locate the element with id #cart and replace the content of the shopping cart with new content. the new content if “render(@cart)”

the ‘escape_javascript’ make sure quotes are escaped

‘#cart’ is used because in application.scss, we have ‘cart’ as the shopping cart id:

```
<div id="cart">
<%= render(@cart)%>
</div>
```

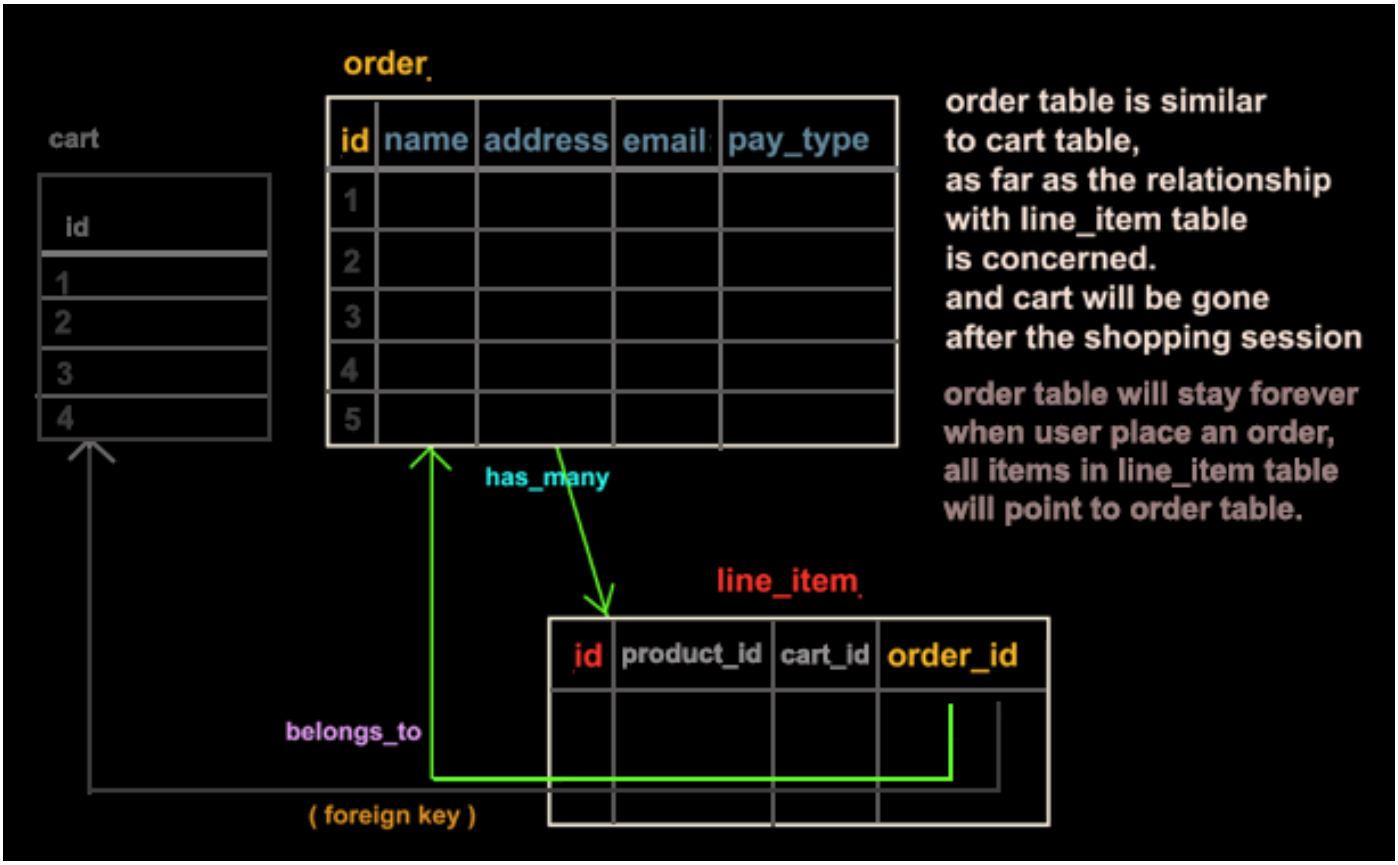
to check if AJAX works, add a text in your layout (anywhere outside your cart div)

```
<div style="color:yellow">
<h1>testing</h1>
</div>
```

```
<div id="cart">
 <%= render(@cart)%>
 </div>
```

then, change "testing" to be different text, for example "testing again"
then click "add to cart" button, you will not see "testing" changed to "testing again", but refresh will see it.

- create order table



conceptually order table is a cart table with shopper information. its information will be kept forever. when user place an order, an order is created to hold all the content of the shopping cart. so basically, the way to create an order record, is to:

- store user info (address, email, how to pay)
- point all lineitems to this order record

1 order: has many line items
1 line item: belong to 1 order

```
$rails g scaffold Order name address:text email paytype

$rails g migration AddOrderToLineitems order:references
```

add associations and validations in model:

```
class LineItem < ActiveRecord::Base
  belongs_to :product
  belongs_to :cart

  belongs_to :order

  def total_price
    product.price * quantity
  end

end
```

and

```
class Order < ActiveRecord::Base

  has_many :lineitems, dependent: :destroy

  PAYMENT_TYPES = ["Check", "Credit Card", “Venmo”, “Paypal”]

  validates :name, :address, :email, presence: true

  validates :paytype, inclusion: PAYMENT_TYPES

end
```

add checkout button to shopping cart
/app/views/carts/_cart.html.erb

```
<%= button_to 'Check Out', new_order_path, method: :get %>
```

modify controller:

first, set cart, then modify actions

```
class OrdersController < ApplicationController

  include CurrentCart
```

```
  before_action :set_cart, only: [:new, :create]

  before_action :set_order, only: [:show, :edit, :update, :destroy]
```

second, modify new action
check if cart is empty

```
  # GET /orders/new
  def new

    if @cart.lineitems.empty?
       redirect_to shopper_url, notice: "your cart is empty"
       return
     end

    @order = Order.new
  end
```

change the view for order table

/app/views/orders/new.html.erb.
change to this:

```
<h1>New Order</h1>

<div class="order_form">
<fieldset>
<legend>Please Enter Your Info</legend>

<%= render 'form' %>

</fieldset>
<%= link_to 'Back', orders_path %>
```

note:
you can also use:
<%= render 'form', order: @order %>
in the new view
then use form_for(order) in the partial

_form partial

app/views/orders/_form.html.erb

```
<%= form_for(@order) do |f| %>
  <% if @order.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@order.errors.count, "error") %> prohibited
this order from being saved:</h2>

      <ul>
      <% @order.errors.full_messages.each do |message| %>
    <li><%= message %></li>
      <% end %>
```

```
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :name %><br>
    <%= f.text_field :name, size: 40 %>
  </div>
  <div class="field">
    <%= f.label :address %><br>
    <%= f.text_area :address, rows: 3, cols: 40 %>
  </div>
  <div class="field">
    <%= f.label :email %><br>
    <%= f.text_field :email, size: 40 %>
  </div>
  <div class="field">
    <%= f.label :paytype %><br>
    <%= f.select :paytype, Order::PAYMENT_TYPES,
prompt: 'Select a payment method' %>
  </div>
  <div class="actions">
    <%= f.submit ‘Place Order’%>
  </div>
<% end %>
```

Add CSS for order form

add this to the order css

```
.order_form {
  fieldset {
      background: #efe;

    legend {
          color: #dfd;
                background: #141;
                      font-family: sans-serif;
                            padding: 0.2em 1em;
                                }
                                  }

  form {
      label {
            width: 5em;
                  float: left;
                        text-align: right;
                              padding-top: 0.2em;
                                    margin-right: 0.1em;
                                          display: block;
                                              }
```

```
    select, textarea, input {
          margin-left: 0.5em;
              }

    .submit {
          margin-left: 4em;
              }

    br {
          display: none
              }
                }
                }
```

Create the order in order controller to add items

```
  # POST /orders
  # POST /orders.json
  def create
    @order = Order.new(order_params)

     @order.add_items_from_cart(@cart)

     respond_to do |format|
       if @order.save
          Cart.destroy(session[:cart_id])
          session[:cart_id] = nil

          format.html { redirect_to shopper_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
```

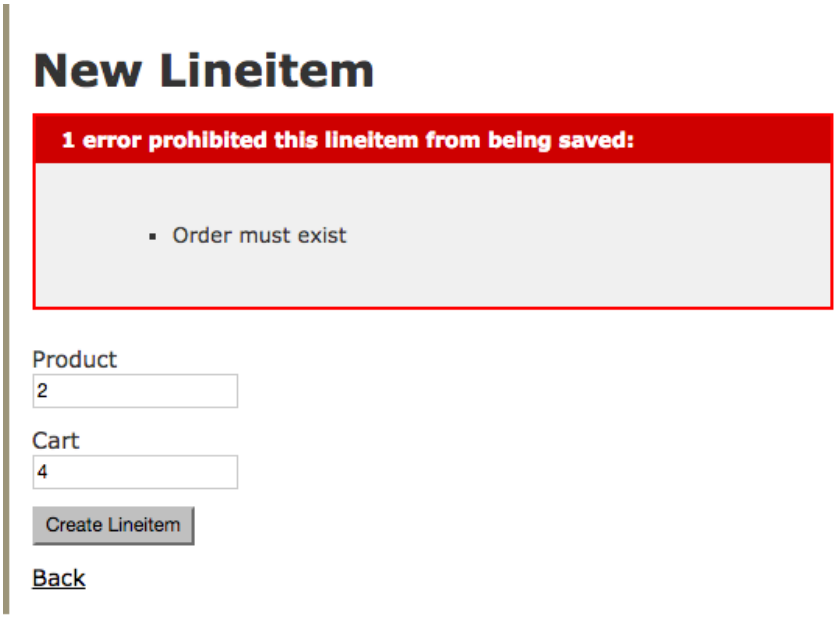
define add_items_from_cart method in model

```
class Order < ActiveRecord::Base

  has_many :lineitems, dependent: :destroy

  PAYMENT_TYPES = ["Check", "Credit Card", “Venmo”, “Paypal”]

  validates :name, :address, :email, presence: true

  validates :paytype, inclusion: PAYMENT_TYPES
```

```ruby
  def add_items_from_cart(cart)
    cart.lineitems.each do |item|
      item.cart_id = nil
      item.order_id = self.id
    end
  end
end
```

this is to transfer from cart to the order,
we will keep the lineitems that are ordered in the lineitems table, just that it will point to different parent (from cart to order)

if you see this when click on “add to cart”



This happens because Rails 5 introduced a constraint that you can not create child (line item) without parent (order)

so solution is, add optional: true to the lineitem model

```ruby
class Lineitem < ApplicationRecord
  belongs_to :product
  belongs_to :cart
  belongs_to :order, optional: true

  def totalprice
    quantity * product.price
  end
end
```

add interface to create admin users
admin users is kept in user table

$ rails g scaffold User name password_digest

model to add validates:

```
class User < ActiveRecord::Base
  validates :name, presence: true, uniqueness: true
  has_secure_password  #so that we can call user.authenticate method
end
```

in Gemfile uncomment out this:
[so that has_secure_password and be used]

```
# Use ActiveModel has_secure_password
gem 'bcrypt', '~> 3.1.7'
```

install this gem:

> bundle install (or just bundle. default to bundle install)

modify the following actions in user controller
users#create
users#update
users#index

redirect to users list, instead of showing a user

```
  # POST /users.json
  def create
    @user = User.new(user_params)

    respond_to do |format|
      if @user.save
    format.html { redirect_to users_url, notice: “User
#{@user.name} was successfully created.” }
        format.json { render :show, status: :created, location:
@user }
      else
        format.html { render :new }
        format.json { render json: @user.errors,
status: :unprocessable_entity }
      end
    end
  end
```

—

```
  # PATCH/PUT /users/1
  # PATCH/PUT /users/1.json
  def update
    respond_to do |format|
      if @user.update(user_params)
        format.html { redirect_to users_url, notice: “User
#{@user.name} was successfully updated.” }
        format.json { render :show, status: :ok, location: @user }
      else
        format.html { render :edit }
        format.json { render json: @user.errors,
status: :unprocessable_entity }
      end
    end
  end


  def index
    #@users = User.all
    @users = User.order(:name)
  end
```

change user index view to add notice flash

```
<h1>Listing Users</h1>
<%if notice %>
<p id="notice"><%= notice %></p>
  <% end %>
```

change the app/views/users/_form.html.erb to add legend

```
<div class='order_form'>

<%= form_for(@user) do |f| %>
  <% if @user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@user.errors.count, "error") %> prohibited
this user from being saved:</h2>

      <ul>
      <% @user.errors.full_messages.each do |message| %>
        <li><%= message %></li>
```

```
      <% end %>
      </ul>
    </div>
  <% end %>
```

```
<fieldset>
<legend>Enter user details</legend>
  <div class="field">
    <%= f.label :name %><br>
    <%= f.text_field :name %>
  </div>
  <div class="field">
    <%= f.label :password %><br>
    <%= f.password_field :password %>
  </div>
  <div class="field">
    <%= f.label :password_confirmation %><br>
    <%= f.password_field :password_confirmation %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
</fieldset>
          <% end %>
</div>
```

go to /users in browser

if you get cannot load bcript error
restart your server

create new user using the interface.

——

create actions for admin to check number of orders:

$ **rails g controller admin index**

just one page to list order number

change the index view to display number of orders:

```
<% if notice %>
<p id="notice"><%= notice %></p>
<% end %>

<h1>Welcome</h1>

It's <%= Time.now %>
We have <%= pluralize(@total_orders, "order") %>.
```

change controller

```
class AdminController < ApplicationController
  def index
    @total_orders = Order.count
  end
end
```

create actions to authenticate user

$ **rails g controller access new create destroy**

```
access#new:  for admin user logon
access#create: for authenticate admin user
access#destroy: for admin user logout
```

change view for user logon

app/views/access/new.html.erb

```
<div class="order_form">
  <% if flash[:alert] %>
    <p id="notice"><%= flash[:alert] %></p>
  <% end %>
  <%= form_tag do %>
    <fieldset>
      <legend>Please Log In</legend>
      <div>
        <%= label_tag :name, 'Name:' %>
        <%= text_field_tag :name, params[:name] %>
      </div>
      <div>
     <%= label_tag :password, 'Password:' %>
        <%= password_field_tag :password, params[:password] %>
      </div>
      <div>
        <%= submit_tag "Login" %>
      </div>
    </fieldset>
  <% end %>
</div>
```

change actions in default methods to check user

```
class AccessController < ApplicationController
  def new

   def new
    if session[:user_id]
      redirect_to admin_url, notice: "already logged on"
      return
    end
  end

  #this is actually “post ‘/logon’
  def create
    user = User.find_by(name: params[:name])
    if user and user.authenticate(params[:password])
      session[:user_id] = user.id
      redirect_to admin_url
    else
      redirect_to login_url, alert: "Invalid user/password combination"
    end
  end


  def destroy
    session[:user_id] = nil
    redirect_to shopper_url, notice: "Logged out"
  end

end
```

change route so that “admin” point to “admin#index” action
“login” point to “access#new” action
also add route point to “access#create” and “access#destroy”

add logout button to side bar (and other links)

```
 <div id="columns">
    <div id="side">
       <div id = 'cart'>
          <%= render @cart %>
       </div>
       <ul>
        <li><a href="http://www....">Home</a></li>
        <li><a href="http://www..../faq">Questions</a></li>
        <li><a href="http://www..../news">News</a></li>
```

```
        <li><a href="http://www..../contact">Contact</a></li>
        </ul>

         <% if session[:user_id] %>
           <ul>
           <li><%= link_to 'Orders',   orders_path   %></li>
           <li><%= link_to 'Products', products_path %></li>
           <li><%= link_to 'Users',    users_path    %></li>
           </ul>
          <%= button_to 'Logout', logout_path, method: :delete
%>
         <% end %>
```

set access

change application controller so that the “authorize action” will be taken first

application_controller.rb

```
class ApplicationController < ActionController::Base

  before_action :authorize

  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

 def authorize
    unless User.find_by(id: session[:user_id])
      redirect_to login_url, notice: "Please log in"
    end
  end
end
```

whitelist some controllers to skip authorize

add
skip_before_action :authorize

to the controllers you don’t want authorize action to be called

—

Add a search form at side bar so that you are able to search for a product.