Another Pin Co

models
app/models/admin_user.rb

- Included default devise modules in the AdminUser model using the devise method.
- The included modules are :database_authenticatable, :recoverable, :rememberable, and :validatable.
- These modules provide authentication features for the AdminUser model, such as secure password storage, password recovery, session management, and validation of login credentials.

app/models/cart.rb

- Created a Cart class that inherits from ApplicationRecord.
- Added a has_many association to the OrderItem model, allowing each Cart instance to have multiple associated OrderItem instances.
- Defined a total_quantity method for the Cart class that calculates the total quantity of all OrderItems associated with the Cart instance.
- Defined a total_price method for the Cart class that calculates the total price of all OrderItems associated with the Cart instance, based on the product price and quantity of each OrderItem.
- Defined a total_price_in_dollars method for the Cart class that calculates the total price of all OrderItems associated with the Cart instance, but uses the product's price in dollars instead of the default currency.
- Used the each method to iterate over all OrderItems associated with the Cart instance and perform the necessary calculations.
- Assigned the final total quantity or price to an instance variable for later use or display.
- The methods can be called on any instance of the Cart class, returning the appropriate calculated value.

app/models/order_item.rb

- Created a model class OrderItem that extends ApplicationRecord.
- Defined associations to other models with belongs_to: Order and Cart models. The optional: true option allows for the association to be optional, meaning that an OrderItem object can exist without an associated Order or Cart object.
- Also defined an association to the Product model but without the optional option, which means that an OrderItem object must have an associated Product object.
- Defined a validation for the quantity attribute using validates method with numericality validator. The validator checks if the quantity value is greater than or equal to zero.

Another Pin Co

models

app/models/order.rb

- Defined a class called Order that inherits from ApplicationRecord.
- Added a has_many association with order_items.
- Added validation rules for the first_name, last_name, email, address_1, city, and country fields.
- Enabled nested attributes for order_items.
- Defined a method called add_from_cart that takes a cart object and moves its items to the order.
- Defined a method called save_and_charge that saves the order and charges the customer using Stripe. It first checks if the order is valid, creates a new Stripe customer, creates a new payment source for the customer, and charges the customer. If an error occurs, it handles the error and adds it to the model errors. If the order is not valid, it returns false.
- Defined a method called total_price that calculates the total price of the order in cents.
- Defined a method called total_price_in_dollars that calculates the total price of the order in dollars.

app/models/product.rb

- Here's a concise explanation of what I did in the given code:
- Validated the presence of title, price, and description using the validates method.
- Established a one-to-many association between Product and OrderItem using the has_many method.
- Mounted five image uploaders using the mount_uploader method and specifying the ProductImageUploader as the uploader class.
- Defined a price_in_dollars method to convert the price from cents to dollars.
- Overall, I developed the Product model, which represents a product in an e-commerce application, and added validations for its attributes, a one-to-many association with order items, and image uploaders. I also implemented a method to convert the product's price from cents to dollars.

uploaders

app/uploaders/product_image_uploader.rb

- Included CarrierWave::RMagick and CarrierWave::MiniMagick to provide support for image manipulation.
- Used :fog as the storage option to choose what kind of storage to use for this uploader.
- Overridden the directory where uploaded files will be stored by defining the store_dir method. It will store the uploaded files in the "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}" directory.
- Defined different versions of the uploaded files using the version method for :large, :medium, and :thumb.
- Applied resize_to_fill method to the uploaded files for each version. The :large version will resize the image to 1000x1000, :medium to 660x660, and :thumb to 220x220.

controllers

app/controllers/application_controller.rb

- The before_action method is used to call the current_cart method before any other actions are executed in the controller.
- The helper_method macro is used to make the current_cart method available in the views as well.
- The current_cart method is defined, which is responsible for giving a user a cart.
- When a user arrives on the website, the current_cart method checks if they already have a cart by looking at the session[:cart_id] value.
- If they already have a cart, the method retrieves the cart from the database using Cart.find(session[:cart_id]).
- If they don't have a cart, the method creates a new one using Cart.create, and sets the session[:cart_id] to the ID of the new cart.
- Finally, the @current_cart variable is set to the newly created or retrieved cart, which can be used throughout the application.
- In summary, this code sets up a mechanism for creating and managing shopping carts for users of a web application. It ensures that each user has a unique cart, and allows the cart to be accessed from any view or action in the application.

controllers

app/controllers/carts_controller.rb

- Initialized a CartsController class that inherits from the ApplicationController class.
- Defined a show method within the CartsController class.
- The show method renders a view that displays the current state of the user's cart.
- Since the cart object has been set up for every view on the website due to the ApplicationController, there is no need to explicitly initialize or set up the cart object in the show method.
- The show method executes implicitly when the user visits the "show" page for their cart.

app/controllers/orders_controller.rb

- In the OrdersController, a show action is defined that retrieves an order object by its id and assigns it to the instance variable @order.
- In the same controller, a new action is defined that creates a new Order object and adds items to it from the current cart.
- The create action creates a new Order object with parameters passed from the form, adds items to it from the current cart, and saves it if payment is successfully charged via Stripe. If payment fails, it renders the new template to show the error message.
- The form_params method is a private method that allows only specific parameters to be passed to the create action, preventing unauthorized access to other parameters.
- When an order is successfully created and charged, the user's session is reset, a success flash message is displayed, and the user is redirected to the order's show page.
- If an order fails to be created, the user stays on the new page and an error message is displayed.
- A private method add_from_cart is used to add items from the cart to the order object.
- An OrderMailer is used to send a receipt email to the customer once the order is created.
- In summary, this code defines a controller for handling orders in a Ruby on Rails application. It has methods for displaying, creating, and adding items to an order, as well as for sending a receipt email to the customer. The form_params method ensures that only permitted parameters are passed to the create action, and the add_from_cart method adds items from the cart to the order object.

controllers

app/controllers/order_items_controller.rb

- Created an OrderItemsController that inherits from the ApplicationController.
- Implemented the create method to find a product by its ID, get the quantity from form data, and add the product to the current cart with the specified quantity.
- Set a success message to flash when the product is added to the cart, then redirected to the product's page.
- Implemented the update method to recognize the path of the previous request, find the product and order item by their IDs, and update the order item with new form parameters.
- Set a success message to flash when the cart is updated, and redirected to the product's page or the previous controller and action, depending on the referrer.
- Implemented the destroy method to find the product and order item by their IDs, delete the order item, set a success message to flash, and redirected to the cart's page.
- Created a private method to permit the quantity parameter in form data.

app/controllers/pages_controller.rb

- Created a PagesController class that inherits from ApplicationController.
- Defined a home method that sets @featured_products instance variable to all products where is_featured is true.
- Defined an info method that does not have any code in it.
- These methods will be used as actions to handle HTTP requests and return responses to the client.

app/controllers/products_controller.rb

- Defines a controller called ProductsController that inherits from ApplicationController.
- Defines two actions, index and show.
- The index action retrieves all products from the database and assigns them to the instance variable @products.
- The show action finds a specific product based on the ID provided in the params hash and assigns it to the instance variable @product.
- Checks if the product is already in the cart by finding an order item with the product and the current cart.
- If the order item does not exist in the cart, creates a new order item with the current cart, the product, and a quantity of 1.
- The order item is assigned to the instance variable @order_item.

views

app/views/carts/show.html.erb

- Checked if there are any order items in the current cart using @current_cart.order_items.any?
- If there are any order items, created a table to display the item details.
- For each order item in the current cart, created a table row to display the item details such as product image, title, price, quantity, and a button to update the quantity using simple_form_for.
- Added a link to remove the item from the cart using link_to and confirmed the action using data: { confirm: "Are you sure?" }.
- Displayed the total price of all items in the cart using @current_cart.total_price_in_dollars.
- Added a link to continue to checkout if there are items in the cart.
- If there are no items in the cart, displayed a message "There are no items in your cart".

app/views/layouts/application.html.erb

- Created a header section with the Another Pin Co brand name and a navigation bar containing links to the shop, information, and cart pages.
- Displayed the total price of the items in the cart using the number_to_currency helper method.
- Implemented a conditional statement to display a flash success message, if present.
- Included a footer section with the copyright year.

app/views/order_mailer/receipt.html.erb

- Display a heading "Thank you for your order." using an H1 HTML tag.
- Display the order ID using the instance variable @order.id within an HTML paragraph tag.
- Display the first and last name of the person who made the order using the instance variables @order.first_name and @order.last_name within an HTML paragraph tag.
- Display the total price of the order in dollars using the instance variable @order.total_price_in_dollars and the number_to_currency helper method within an HTML paragraph tag.
- Display a message "Thanks for buying!" using an HTML paragraph tag.

Another Pin Co

views

app/views/orders/new.html.erb

- Added a checkout form for an order using the Simple Form gem in Ruby on Rails.
- Created input fields for the customer's first name, last name, email, country, address 1, address 2, city, and postal code.
- Displayed any validation errors for the Stripe token on the page.
- Added input fields for the credit card number, expiration date, and CVC.
- Styled the credit card input fields using the Stripe Elements API by adding custom fonts and setting styles for the input fields.
- Created a Stripe object using the Stripe API key provided in the Rails application credentials.
- Created a Stripe card element for the credit card input fields and mounted it on the page.

app/views/orders/show.html.erb

- Created an HTML page with a header and three paragraphs.
- Used embedded Ruby (ERB) to dynamically generate one of the paragraphs to include a link to the root path of the application.
- The page displays a message thanking the user for their order and indicating that delivery will take 7-10 days.
- The link allows the user to easily navigate back to the home page of the application.

app/views/pages/home.html.erb

- Render the "Homepage" view template which will be used to display the home page of the web application.
- Check if there are any featured products available to display on the homepage.
- If there are any featured products available, then render a section on the homepage with a title "Featured pins".
- Render a partial template named "/products/product" for each of the featured products using the "@featured_products" collection.
- If there are no featured products available, then the section for featured products will not be displayed on the homepage.

views
app/views/pages/info.html.erb

- Created layout for an info/about page for AnotherPin Co.

app/views/products/_product.html.erb

- Used the link_to helper method to generate a hyperlink to the product_path route.
- Utilized the image_tag helper method to display the product's image using its medium size url.
- Displayed the product title and price by using the title and price_in_dollars attributes of the product object.
- Converted the product price to a currency format using the number_to_currency helper method.
- Wrapped the product information in a div element with a class of "product" for styling purposes.

app/views/products/index.html.erb

- Utilized the "@products" collection to display a list of products on the webpage.
- Rendered the "product" partial for each product in the "@products" collection.
- Created a webpage that displays a list of products using the Ruby on Rails framework.
- Completed the task using Ruby on Rails conventions and best practices.

app/views/products/show.html.erb

- Displayed the title of the product using the h2 tag and the value stored in the @product.title variable.
- Displayed the product description using the simple_format helper method and the value stored in the @product.description variable.
- Created a form using the simple_form_for helper method to allow users to add the product to their cart.
- Bound the form to the @product and @order_item variables to create a new order item when the user submits the form.
- Added an input field for the user to enter the quantity of the product they want to add to their cart.
- Added a submit button with the label "Add to cart" to submit the form.
- Added a link to the "Back to all products" page using the link_to helper method and the products_path variable.

Another Pin Co

admin

app/admin/admin_users.rb

- Registered AdminUser with ActiveAdmin framework
- Permitted the parameters email, password, and password_confirmation for AdminUser
- Defined the index view to display selectable column, id column, email, current_sign_in_at, sign_in_count, created_at, and action buttons for AdminUser
- Added filters for email, current_sign_in_at, sign_in_count, and created_at for AdminUser in the index view
- Defined the form view to display email, password, and password_confirmation inputs for AdminUser
- Added actions buttons for the form view for AdminUser

app/admin/dashboard.rb

- Registered the "Dashboard" page with ActiveAdmin
- Set the priority of the page to 1 and the label to "active_admin.dashboard"
- Defined the content of the dashboard with a title of "active_admin.dashboard"
- Created a container div with a class of "blank_slate_container" and an id of "dashboard_default_message"
- Added a span with a class of "blank_slate"
- Added another span with the text "active_admin.dashboard_welcome.welcome"
- Added a small tag with the text "active_admin.dashboard_welcome.call_to_action"
- Defined two columns using the "columns" method
- Added a panel to the first column with the title "Recent Posts"
- Listed the 5 most recent posts using the "Post.recent(5)" method and iterated through them with the "map" method
- Added a link to each post's title using the "link_to" method and the "admin_post_path(post)" argument
- Added a panel to the second column with the title "Info"
- Added a paragraph tag with the text "Welcome to ActiveAdmin"

Another Pin Co

admin

app/admin/orders.rb

- Created an ActiveAdmin resource for the "Order" model.
- Defined the permitted parameters for assignment using the "permit_params" method. Allowed parameters include first name, last name, email, country, address 1, address 2, city, postal code, and order items attributes such as ID, product ID, and quantity.
- Defined the index page for the "Order" model, including a selectable column, name (concatenated from first and last name), email, country, and actions.
- Defined the form for creating/editing an order, including basic information such as first name, last name, and email, and address information such as address 1, address 2, city, postal code, and country.
- Defined the "order_items" section of the form using the "has_many" method to allow for the creation/editing of multiple items on the same order, with fields for product and quantity.
- Added actions to the form, such as a "submit" button.

app/admin/products.rb

- Registered Product as an ActiveAdmin resource.
- Allowed certain parameters to be permitted for assignment using permit_params.
- Overwrote the default admin form for the Product resource with a personalized form.
- Defined the content to be displayed on the products index admin page using index do.
- Created a form for the Product resource using form do.
- Defined the inputs for the Product form including title, price, description, collection_date, is_featured, is_sold_out, image_1, image_2, image_3, image_4, and image_5.
- Defined the description input as a quill_editor.
- Defined the is_featured and is_sold_out inputs with custom labels.
- Created a section for the Images inputs.
- Ended the form.

Another Pin Co

mailers

app/mailers/order_mailer.rb

- Created an OrderMailer class which inherits from the ApplicationMailer class in Ruby on Rails.
- Defined a receipt method in the OrderMailer class.
- The receipt method takes an argument 'order', which is an instance of the Order class.
- Assigned the order instance to an instance variable @order, so that it can be used in the email view.
- Using the mail method, an email is sent to the email address associated with the order instance.
- The email has a subject line, "Thank you for ordering from Another Pin Co."

config

config/routes.rb

- Configured Devise gem for admin users authentication using ActiveAdmin configuration.
- Added routes for the ActiveAdmin dashboard.
- Defined resources for products and nested resources for order items to manage multiple products with their order items.
- Defined resources for pages to allow multiple pages in the application.
- Defined resources for orders to allow users to place multiple orders.
- Defined a singular resource for the cart to restrict a user to have only one cart at a time.
- Defined a custom route 'info' to display information about the application.
- Set the root route to 'pages#home' for the homepage.