Source

models

app/models/admin_user.rb

- Defined the "AdminUser" class as a subclass of the "ApplicationRecord" class, indicating that it will be persisted in the database.
- Used the "devise" gem to add authentication functionality to the "AdminUser" model. This adds several modules that provide authentication features such as authentication with a password (database_authenticatable), password recovery (recoverable), remember me functionality (rememberable), and validation of password strength (validatable).
- Specified the modules that should be included in the "devise" configuration using the "devise" method and passing the symbols representing the desired modules.

app/models/category.rb

- Defined a one-to-many relationship between Category and CategoryItem models using the has_many method. This means that one Category can have many CategoryItems.
- Defined a many-to-many relationship between Category and Item models using the has_many method with the :through option. This means that a Category can have many Items through the CategoryItems table.
- Set the dependent option to :destroy on the has_many association with CategoryItem. This means that if a Category is destroyed, all its associated CategoryItems will also be destroyed.
- Validated the presence of title attribute in Category using the validates method. This means that a Category cannot be saved if the title attribute is blank.
- Validated the uniqueness of title attribute in Category using the validates method. This means that a Category cannot be saved if there is already a record with the same title.

app/models/category_item.rb

- Defined a belongs_to association with Item model.
- Defined a belongs_to association with Category model.
- In other words, I established a many-to-many relationship between the Item and Category models through the CategoryItem join model. Each CategoryItem instance represents a connection between an Item and a Category, allowing an Item to belong to multiple Categories and vice versa.
- To summarize, this code defines a Category model in Ruby on Rails with associations to CategoryItem and Item models, and includes validations for the title attribute to ensure its presence and uniqueness.

app/models/item.rb

- Added an association to CategoryItem using has_many and dependent: :destroy
- Added an association to Category using has_many and through: :category_items
- Added a validation to ensure that the title attribute is present using validates :title, presence: true
- Added a validation to ensure that the image attribute is present using validates :image, presence: true
- Used mount_uploader to specify that the image attribute should be uploaded using the ImageUploader class.

Source

models

app/models/user.rb

- There are several validations defined for the model using the validates method. These ensure that the name, username, email, stripe_token, and subscription_plan fields are present and unique where appropriate.
- The has_secure_password method is called to add authentication functionality to the model.
- The save_and_subscribe method is defined to handle creating a Stripe customer and subscription for a user. It first checks if the user is valid, then creates a Stripe customer using the user's email and a test credit card token. It then creates a subscription for the user based on their chosen subscription plan (either 'basic' or 'pro'). Finally, it saves the customer and subscription IDs to the database.
- The update_with_stripe method is defined to handle updating a user's subscription plan in Stripe and the database. It takes form_params as an argument, updates the model with those parameters, checks if it's valid, then updates the subscription in Stripe and saves the changes to the database.
- To summarize, the code defines a User model with validations and authentication, and provides methods for creating and updating Stripe subscriptions for users.

Source

controllers

app/controllers/accounts_controller.rb

- In the AccountsController class, the edit method is defined to display the user's account information.
- The update method is defined to update the user's account information with Stripe payment information. If the update is successful, the user is redirected to the root path with a success message. If the update fails, the user is redirected back to the edit page.
- The destroy method is defined to delete the user's account and subscription from Stripe and the database. After deleting the user and subscription, the session is reset and the user is redirected to the root path.
- The form_params method is defined as private to permit subscription_plan parameters to be passed in the request.
- In summary, the code defines the methods necessary to display, update, and delete a user's account and subscription information, while also resetting the user's session and redirecting to the root path as necessary.

app/controllers/application_controller.rb

- Created a helper method to make current_user and is_logged_in? available to views.
- Defined a before_action method to execute the current_user method before any action is executed in the whole application.
- Defined the current_user method to check if the session for the user id is present (user is logged in) and return the @current_user variable.
- Defined the is_logged_in? method to check if the session for the user id is present.
- Defined the force_login method to redirect to the root path and show an error message if the user is not logged in.
- Used instance variables to store and return the @current_user variable.

app/controllers/categories_controller.rb

- Created a CategoriesController class that inherits from the ApplicationController class.
- Defined an index method that retrieves all the categories from the database using the Category.all method and assigns them to an instance variable called @categories.
- Defined a show method that retrieves a specific category from the database using the Category.find method and passing in the params[:id] parameter to find the category with the corresponding ID.
- The retrieved category is then assigned to an instance variable called @category.

app/controllers/items_controller.rb

- Here's a concise explanation of the code:
- The ItemsController inherits from the ApplicationController, giving it access to its methods.
- The force_login method is executed before any action is taken, making sure that the user is logged in.
- The index action checks for a search query, and if present, filters the items by title, otherwise it displays all items.
- The show action retrieves and displays a specific item based on its ID.
- The download action retrieves an item's image from a remote URL and sends it as a file download to the user. The Down gem is used to download the file and send_file method is used to send it as an attachment. Alternatively, open-uri can be used to download the file and send_data method can be used to send it as an attachment.

Source

controllers

app/controllers/pages_controller.rb

- Created a PagesController class that inherits from the ApplicationController class.
- Defined a home method in the PagesController class.
- Used the is_logged_in? method to check if a user is logged in.
- If a user is logged in, redirected them to the items_path.
- The home method renders the home view by default if a user is not logged in.

app/controllers/sessions_controller.rb

- Defined a SessionsController that inherits from the ApplicationController.
- Implemented a new method that renders a new view to the user to log in.
- Implemented a create method that accepts form data from the user, and then sets a session with the user's ID if the user exists in the database and the entered password matches their password.
- If the user does not exist or their password is incorrect, flash an error message and render the 'new' view again.
- Implemented a destroy method that resets the user's session, flashes a success message, and redirects them to the root path.
- Defined a private method form_params that requires and permits username and password parameters from the session params.

app/controllers/users_controller.rb

- Created a UsersController that inherits from the ApplicationController.
- Defined a new action that creates a new user object.
- Defined a create action that creates a new user object with the parameters submitted from a form.
- Checked if the user object can be saved and subscribed.
- If saved, the session is created with the user_id and the user is redirected to the root path with a success message.
- If not saved, the user is redirected back to the new action.
- Defined a private method form_params that requires the user parameters and permits certain user attributes.

Source

helpers

app/helpers/items_helper.rb

- Created an ItemsHelper module.
- Defined a method called format_categories that takes in an array of categories as an argument.
- Mapped each category in the categories array into a link using the link_to method with the category's title and path.
- Joined the links together into a sentence using the to_sentence method.
- Marked the sentence as HTML safe using the html_safe method.
- Returned the resulting sentence from the method.
- In summary, the format_categories method takes an array of category objects, creates links for each category, and returns a formatted sentence containing all the links.

uploaders

app/uploaders/image_uploader.rb

- Included RMagick or MiniMagick support to allow for image manipulation.
- Chose to use fog storage for file storage.
- Overrode the default store directory for uploaded files to "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}".
- Provided a default URL for when there is no uploaded file using ActionController::Base.helpers.asset_path("fallback/" + [version_name, "default.png"].compact.join('_')).
- Created different versions of uploaded files with the medium and thumb versions, which are resized to fill and fit respectively.

Source

views

app/views/accounts/edit.html.erb

- Created an HTML heading with the text "Edit account" using the h1 tag.
- Used the simple_form_for helper method to create a form for the @user object, with the form's action URL set to the account_path.
- Added a select input field for the @user's subscription plan, with two options - "Basic subscription - $9.99/month" and "Pro subscription - $19.99/month" - using the f.input method and specifying the input type as :select.
- Set the collection option to an array of arrays, with the first element of each sub-array representing the display text of the option and the second element representing the value that will be passed to the database (either 'basic' or 'pro').
- Included a parameter include_blank set to false to ensure that the user has to choose one of the options.
- Added a submit button with the text "Update account" using the f.button method.
- Closed the form with the end keyword.
- Created a paragraph element containing a link with the text "Delete account" and a confirmation dialog box that asks the user if they are sure they want to delete their account.
- Set the link's href attribute to the account_path and its method attribute to :delete.

app/views/categories/index.html.erb

- Loop through each category object in the @categories collection
- For each category, create a paragraph (<p>) element containing a link (<%= link_to %>) to the category's title (category.title) and the category's path (category_path(category))
- Render the resulting HTML as part of an <h1> element with the text "Categories" (<h1>Categories</h1>)

app/views/categories/show.html.erb

- Render a page that displays a category's title using <h1>
- Display a collection of items belonging to that category in a grid format using a partial template items/item
- Provide a link to go back to the list of all categories using link_to and categories_path

Source

views

app/views/items/_category.html.erb

- Created a link that displays the title of a category and directs the user to the category's show page when clicked.
- Used the Rails link_to helper method to generate the link with the appropriate URL and HTML attributes.
- Passed in the category's title and the category_path(category) helper method as arguments to the link_to method.
- Wrapped the link in a span element with a pipe character (|) on either side for styling purposes.

app/views/items/_item.html.erb

- Created a link using the link_to method to generate an HTML hyperlink that points to a specific URL.
- The item_path(item) argument passed to link_to is a helper method that generates a URL for the given item object.
- The class: "item" argument adds a CSS class to the hyperlink for styling purposes.
- Used the image_tag method to generate an HTML img tag with the medium version of the item's image URL.
- The item.title method is used to display the item's title inside a p tag.

app/views/items/index.html.erb

- Created a search form that takes in a query parameter through a text input field with a name attribute of "q".
- Used the @items instance variable to render a collection of items as a grid using a partial template called "item".
- Checked if the @items collection is empty and if the query parameter is present. If both are true, rendered a message "Nothing here, try searching something else!".
- Commented out a block of code that presumably contained an iteration over each item in @items and displaying an image using the image_tag helper function.

app/views/items/show.html.erb

- Created a view template file that displays a title, categories, description, image, and a download link for an item.
- Used the instance variable @item to access the attributes of a specific item object.
- Used the pluralize method to display the correct plural form of the word "Category" based on the number of categories associated with the item.
- Created a custom helper method format_categories in the items_helper.rb file to display the names of the categories associated with the item.
- Used the simple_format method to format the description text into paragraphs.
- Used the image_tag method to display the item's medium-sized image.
- Created a link using the link_to method that points to the download action of the items controller and passes the ID of the item as a parameter.

Source

views

app/views/layouts/application.html.erb

- Included necessary meta tags and linked stylesheets and scripts in the head section of the HTML file.
- Created a body section with conditional statements to check for the presence of flash messages and display them accordingly.
- Added a header section that displays the site name as a link to the home page or the items index page depending on the current controller and action.
- Created a navigation section that displays links based on the user's login status, including links to the items index page, categories index page, account editing page, and logout action if the user is logged in, and links to the sign-up and login pages if the user is not logged in.
- Defined the content and footer sections that wrap around the dynamic content of the site and the credits for the image sources.

app/views/pages/home.html.erb

- Created a header tag <h1> with the text "Source".
- Added two link tags <%= link_to %> that direct users to the sign-up and log-in pages respectively.
- Used the new_user_path and new_session_path helper methods to generate the URLs for the sign-up and log-in pages.
- Added classes "button" and "button-outline" to the link tags to style them as buttons.
- Included a welcoming message "Welcome to source - an exclusive image stock photo subscription service".

app/views/sessions/new.html.erb

- Created a view file for the login page with an H1 tag and a simple form using the simple_form_for helper method from the SimpleForm gem.
- Passed the symbol :session as the argument for the simple_form_for method to create a form for a session.
- Passed the session_path as the URL option for the simple_form_for method, which is the path to the login session in the Rails application.
- Used the block variable f to define the input fields for the form using the f.input method provided by SimpleForm gem.
- Created input fields for :username and :password.
- Created a submit button with the text "Log in" using the f.button method.
- Wrapped the entire form with ERB tags to embed Ruby code in HTML.
- Rendered the form in the view file.

Source

views

app/views/users/new.html.erb

- Created a sign-up page with a header that says "Sign up" using an H1 tag.
- Used simple_form_for helper method to create a form for the @user instance variable, which is an instance of the User model.
- Added form inputs for the name, username, email, password, password_confirmation, and subscription_plan fields using f.input.
- Used the as: :select option to display a dropdown menu for the subscription_plan field, with options for a basic and pro subscription.
- Added a div with the id "card-errors" to display any errors that occur during credit card processing.
- Added a div with the id "card" to display the credit card form fields.
- Added a submit button to the form with the text "Sign up" and the id "sign-up-button".
- Used JavaScript to create a Stripe object with the Stripe API key stored in the Rails credentials, and mounted a credit card form to the div with the id "card".
- Added a submit event listener to the form that prevents the default form submission behavior.
- Used the createToken method from the Stripe object to create a token for the credit card information, and appended a hidden input field to the form with the token ID.
- Submitted the form using the submit method if the token was successfully created, or displayed an error message if there was an error.

app/admin/admin_users.rb

- Registered an AdminUser model in the ActiveAdmin dashboard.
- Permitted the parameters email, password, and password_confirmation for AdminUser.
- Created an index view for AdminUser with selectable columns for id, email, current_sign_in_at, sign_in_count, and created_at.
- Added filters for email, current_sign_in_at, sign_in_count, and created_at on the index view.
- Created a form for AdminUser that allows for editing of email, password, and password_confirmation.
- Added actions to the form for AdminUser to allow for saving changes.

app/admin/categories.rb

- Added a resource for the Category model to be managed in the ActiveAdmin dashboard.
- Uncommented the permit_params method to allow the title parameter to be assigned.
- If needed, additional parameters could be permitted by modifying the permitted array.
- Added a conditional statement to check if the current user is an admin and if the action is to create a new record, then the other parameter is also permitted.

Source

views

app/admin/dashboard.rb

- Registered a new page in ActiveAdmin named "Dashboard"
- Set the priority of the Dashboard menu to 1 and labeled it with "active_admin.dashboard"
- Defined the content of the Dashboard with a title and a div container
- Created a span element inside the div container to display a welcome message and a call to action
- Defined two columns in the Dashboard
- Inside the first column, created a panel named "Recent Posts" that displays a list of 5 recent posts with a link to each post's admin page
- Inside the second column, created a panel named "Info" that displays a welcome message
- End the Dashboard content block.

app/admin/items.rb

- Implemented ActiveAdmin functionality to manage Item model data.
- Created permit params to allow access to the title, description, image, and category_ids fields in the Item model.
- Defined the form view for editing and creating Item data, using the form block provided by ActiveAdmin.
- Added input fields for title, description, and image to the form.
- Included a checkbox list of categories for the Item, using the as: :check_boxes option in the form block.
- Sorted the categories in ascending order by their title attribute.
- Included the standard f.actions block to display the form submission buttons.

app/admin/users.rb

- Registered a model called "User" with ActiveAdmin.
- Listed the permitted parameters for the "User" model using the "permit_params" method. This includes the "name", "username", "email", "password", and "password_confirmation" attributes.
- Added an "index" view for the "User" model that displays a table with selectable columns for "id", "name", "username", "email", "subscription_plan", and "is_subscription_active".
- Added "actions" to the index view that allow editing and deleting of individual "User" records.
- Added a "form" view for creating and editing "User" records that includes input fields for "name", "username", and "email".
- The "inputs" method groups the input fields together under a heading of "Personal Information".
- The "actions" method adds the "Submit" and "Cancel" buttons to the form.