Eventz

models

app/models/user.rb

- Defined a has_many association with the Registration class, with the dependent option set to :destroy. This means that when a user is deleted, all of their registrations will also be deleted.
- Defined a has_many association with the Like class, with the dependent option set to :destroy.
- Changed the has_many association with the Event class from "has_many :events, through: :likes" to "has_many :liked_events, through: :likes, source: :event". This allows us to call "user.liked_events" instead of "user.events" to get a list of events that the user has liked.
- Used the has_secure_password method to ensure that the user's password is encrypted and stored securely.
- Added validation for the presence of the user's name and the format and uniqueness of their email.

app/models/registration.rb

- Created a constant HOW_HEARD_OPTIONS with an array of strings representing the different ways a user can hear about an event.
- Added a validation for :how_heard attribute to make sure it is included in HOW_HEARD_OPTIONS.

app/models/like.rb

- Used the "belongs_to" method to create associations between the "Like" model and two other models, "Event" and "User".
- The "belongs_to" method creates a foreign key column for each associated model and sets up the appropriate relationships in the database.
- The "belongs_to" association means that a like "belongs to" both an event and a user, indicating that the "Like" model is used to represent the action of a user liking an event.

app/models/event.rb

- Used a callback function to set the slug for an Event before it is saved.
- Defined associations between an Event and its registrations, likes, likers, categorizations, categories, and main_image using has_many and has_one_attached.
- Validated the presence of an Event's name and location, and checked the length of its description to be at least 25 characters.
- Validated that an Event's price is a non-negative number and its capacity is a positive integer.
- Defined a custom validation function to check if the attached image file is acceptable.
- Defined scopes to query Events based on their past, upcoming, and free statuses.
- There is some incomplete code in the comment about validating the image file name and extension, which was replaced by ActiveStorage.

app/models/category.rb

- Created a class called Category which inherits from ApplicationRecord.
- Defined a has_many association with categorizations with the option dependent: :destroy which means if a category is deleted, all the associated categorizations will be deleted too.
- Defined another has_many association with events through the categorizations table.
- Added a validation for the name attribute of the Category model, checking for its presence and uniqueness.

Eventz

controllers

controllers/categories_controller.rb

- Implemented actions such as index, show, new, create, edit, update, and destroy for handling CRUD operations on category resources.
- Defined private methods such as category_params to sanitize and permit category attributes for mass assignment, and set_category to find a category record by its ID and set it as an instance variable.
- Rendered views such as index, show, new, edit, and form for displaying category information and handling user input.
- Set up before/after filters to execute specific code before or after certain actions, such as authenticate_user! to require authentication for certain actions and authorize_resource to check if the current user is authorized to perform a certain action on a category.

controllers/events_controller.rb

- Required signin for all actions except index and show using before_action method.
- Required admin privilege for all actions except index and show using before_action method.
- Set event for show, edit, update and destroy actions using before_action method.
- Defined index action to query upcoming, past, recent, or free events based on the filter parameter in the URL.
- Defined show action to find the event by id or slug, get the event's likers and categories, and check if the current user has already liked the event.
- Defined edit action to find the event by id.
- Defined update action to find the event by id and update the event with the permitted parameters.
- Used flash notice message after successfully updating the event.

controllers/likes_controller.rb

- Defined a before_action method to require the user to sign in before creating or destroying a like.
- Created a create method that adds a like to an event when a link or button is clicked. If the like is saved, it redirects to the event page.
- Created a destroy method that deletes a user's like when clicked. The method first checks if the user is authorized to delete the like, then deletes it and redirects to the event page.

Eventz

controllers

controllers/registrations_controller.rb

- Use before_action to run require_signin method before every action in the RegistrationsController except for new and create methods.
- Use before_action to run set_event method before index, new, create, and destroy methods.
- Define index method to retrieve all registrations for an event and assign them to the instance variable @registrations.
- Define new method to instantiate a new registration for an event and assign it to the instance variable @registration.
- Define create method to instantiate a new registration with the registration_params and assign it to the instance variable @registration. Set the user attribute of the registration object to the current user. If the registration object is saved successfully, decrement the capacity of the event by 1 and redirect to the event_registrations_url with a success message. If not, render the new template with an error status.
- Define destroy method to find the registration by id and assign it to the instance variable @registration. Also, assign the user of the registration object to the instance variable @user. If the current user is the owner of the registration, destroy the registration and redirect to the user's profile with a success message. If not, redirect to the event with an error message.
- Define registration_params method to permit only the how_heard parameter for registration.

controllers/sessions_controller.rb

- The new method doesn't do anything, it just renders the new.html.erb view.
- The create method looks for a User object in the database that matches the email parameter passed in from the login form.
- If a matching User object is found and the password entered matches the password of the User, the user_id is stored in the session and the user is redirected to either the intended URL (if one exists) or to their own show page. A flash message is also displayed welcoming the user back.
- If the email or password is invalid, a flash message is displayed immediately using flash.now[:alert]. The render method is used to display the new.html.erb view again with a 422 status code.
- The destroy method clears the user_id from the session and redirects the user to the events URL with a flash message indicating that they have been signed out.
- Overall, the code handles user authentication and session management in a Rails web application.

Eventz

controllers

controllers/users_controller.rb

- Defined two before_action methods - require_signin and require_correct_user - to ensure that only signed-in users can access certain actions.
- Defined the index action which fetches all users from the User model and assigns them to the @users instance variable.
- Defined the new action which creates a new User object and assigns it to the @user instance variable.
- Defined the create action which creates a new User object with the user_params parameters and saves it to the database. If successful, it sets the session user_id to the new user's ID and redirects to the user's show page. If not successful, it renders the new view with a status of unprocessable_entity.
- Defined the show action which finds the User object with the given ID parameter and assigns it to the @user instance variable. It also finds all Registrations associated with the user and assigns them to the @registrations instance variable, and finds all Events that the user has liked and assigns them to the @liked_events instance variable.
- Defined the edit action which finds the User object with the given ID parameter and assigns it to the @user instance variable. This action is only accessible to the correct user, as ensured by the require_correct_user before_action method.
- Defined the update action which finds the User object with the given ID parameter and updates its attributes with the user_params parameters. If successful, it redirects to the user's show page with a success notice. If not successful, it renders the edit view with a status of unprocessable_entity.
- Defined the destroy action which finds the User object with the given ID parameter and deletes it from the database. If successful, it sets the session user_id to nil and redirects to the events index page with a success notice. If not successful, it renders the show view with a status of unprocessable_entity.
- Defined a private user_params method which requires and permits the name, email, password, and password_confirmation parameters for a User object.

Eventz

helpers

app/helpers/events_helper.rb

- Defined a helper module named EventsHelper, which is accessible to any Event-related views.
- Defined a view helper method called price, which takes an event object as a parameter.
- The price method calls the free? method on the event object, which is defined in the Event Model.
- If the free? method returns true, the method returns the string "Free". Otherwise, it uses the number_to_currency method to format the event price and returns the result.
- Defined a view helper method called day_and_time, which takes an event object as a parameter.
- The day_and_time method checks if the starts_at attribute of the event object is not nil.
- If the starts_at attribute is not nil, the method formats it using the strftime method and returns the result.
- Defined a view helper method called main_image, which takes an event object as a parameter.
- The main_image method checks if the main_image attribute of the event object is attached.
- If the main_image attribute is attached, the method returns an image_tag with the URL of the attached image.
- If the main_image attribute is not attached, the method returns an image_tag with the URL of a placeholder image.

app/helpers/likes_helper.rb

- Defined a helper module named LikesHelper.
- Created a method called like_or_unlike_button that takes two arguments, event and like.
- If a like object is present, then display a "☆ Unlike" button using the button_to method and specify the path to the event_like_path with the HTTP method DELETE.
- If a like object is not present, then display a "★ Like" button using the button_to method and specify the path to the event_likes_path with the HTTP method POST.
- The button_to method always uses POST as the HTTP verb by default, so there's no need to specify it explicitly.

app/helpers/registrations_helper.rb

- Defined a helper module named RegistrationsHelper.
- The helper has a method named register_or_sold_out that takes an event object as an argument.
- The method checks if the event is sold out or not using the sold_out? method defined in the Event model.
- If the event is sold out, the method generates some HTML using the content_tag helper to display a "Sold Out!" text with a CSS class named 'sold-out'.
- If the event is not sold out, the method generates an HTML link using the link_to helper to register for the event.
- The link has the text "Register!" and a CSS class named 'register'.
- The helper method returns either the sold out HTML or the registration link HTML depending on the event's sold_out status.

Eventz

views

app/views/events/_form.html.erb

- Created a Ruby on Rails form using the form_with helper method, passing in the event model as the argument.
- Rendered the shared/errors partial with the event object to display any validation errors on the form.
- Added a label for the name field and a text_field to accept user input for the name of the event.
- Added a label for the description field and a text_area to accept user input for the description of the event.
- Added a label for the location field and a text_field to accept user input for the location of the event.
- Added a label for the price field and a number_field to accept user input for the price of the event.
- Added a label for the capacity field and a number_field to accept user input for the capacity of the event.
- Added a label for the main_image field and a file_field to accept user input for the main image of the event, using ActiveStorage for file upload.
- Added a label for the starts_at field and a datetime_select to accept user input for the starting time of the event, with the ampm option enabled and a class of datetime for styling purposes.
- Added a collection_check_boxes to accept user input for the categories associated with the event, displaying all categories from the Category model.
- Added a submit button to submit the form.

app/views/events/edit.html.erb

- Rendered a form to edit the details of an event using the "form" partial.
- Passed the "event" instance variable, which contains the details of the event being edited, to the "form" partial for rendering.
- Interpolated the name of the event being edited into an HTML heading using the <%= %> tags.

app/views/events/index.html.erb

- Iterated through the @events instance variable using each method to display the details of each event on the page.
- Created a section element with a class of "event" to encapsulate the event details.
- Created a div element with a class of "image" to display the event's main image.
- Used the main_image helper method to display the event's main image if attached using ActiveStorage or the default image if not attached.
- Created a div element with a class of "summary" to display the event's summary.
- Used the link_to helper method to display the event's name as a link to the event's show page.
- Used the day_and_time helper method to display the event's date and time and the event's location.
- Used the price helper method to display the event's price.
- Used the truncate helper method to display a truncated version of the event's description.
- Checked if the current user is an admin using the current_user_admin? helper method to display an "Add New Event" button if the user is an admin.
- Commented out a section that was meant to display past events in a table.

Eventz

views

app/views/events/new.html.erb

- Created a new event view using HTML's <h1> tag to display the page title "Creating New Event".
- Used the Ruby on Rails syntax "<%=" and "%>" to embed Ruby code in the HTML view file.
- Rendered a partial view named "_form" passing the instance variable "@event" to it.
- The rendered "_form" partial contains a form for creating a new event, which is associated with the @event instance variable passed to it.
- This allows the user to fill in the necessary details for the event and submit the form, which is then handled by the controller to create a new event record in the database.

app/views/events/show.html.erb

- Displayed event image in the HTML template using the main_image helper function.
- Created a conditional statement to display a like/unlike button only if the user is logged in.
- Created a count for the number of likes on the event.
- Displayed the event's name, capacity, price, date and time, location, and description.
- Called the register_or_sold_out helper function to display either a registration link or a "sold out" message, depending on whether the event has reached capacity.
- Created a link to display a list of users who have registered for the event.
- Displayed a list of users who have liked the event (if any) and a list of categories the event belongs to (if any).
- Created an "Edit" button and a "Delete" button (if the current user is an admin) that links to the respective routes.

Eventz

views

app/views/registrations/index.html.erb

- Created a header element with the name of the event as a link to the event page using 'link_to' helper method.
- Iterated through the list of registrations using the '.each' method to display the list of registered users for the event.
- Created an unordered list with a class of 'registrations' using the '<ul>' tag.
- For each registration, created a list item using the '<li>' tag with the name of the registered user as a link to their profile page, using 'link_to' helper method, and displayed the source of how they heard about the event using 'how_heard' attribute.
- Displayed the list item using '<% end %>' tag to close the loop.

app/views/registrations/new.html.erb

- Created a registration form for an event using Ruby on Rails.
- Used a nested resource to pass the parent resource (@event) and the child resource (@registration) in the array of resources.
- Rendered shared errors for the @registration object in the form.
- Added a label and select field for "How did you hear about this event?" using Registration::HOW_HEARD_OPTIONS constant.
- Added a submit button to register for the event.
- Optional: Provided an alternative form that can also be used with a url and method to handle event registrations.

app/views/users/new.html.erb

- Created a Sign Up page using HTML and Ruby on Rails syntax.
- Used the <h1> tag to create a heading for the page.
- Utilized the Rails render method to render a partial template named "form".
- Passed a local variable user to the "form" partial using the user: @user syntax.
- This partial template contains a form that allows users to enter their information for signing up.
- The @user instance variable contains any pre-populated data for the form or user data that was previously entered but did not pass validation.

Eventz

views

app/views/users/_form.html.erb

- Implemented a Ruby on Rails user registration and account update feature.
- Created a form with fields for name, email, password, and password confirmation using the 'form_with' helper method.
- Used the 'render' method to display error messages in the form for the user object.
- Used the 'label' method to create labels for each form field.
- Created text fields for the 'name', 'email', 'password', and 'password_confirmation' fields using the 'text_field' and 'password_field' methods.
- Used an if statement to check if the user object is a new record and displayed the appropriate submit button with the 'submit' method.
- Used the 'email_field' method to create an email field for the email input.
- Ensured that the name field had autofocus enabled to allow for a better user experience.

app/views/users/edit.html.erb

- Rendered a HTML h1 tag with the text "Edit Account" using ERB syntax
- Rendered a partial view called "form" with a local variable named "user" set to the value of the instance variable "@user". The partial view contains a form for editing user account information.

app/views/users/index.html.erb

- Used the pluralize method to display the number of users (@users) in an H1 heading.
- Created an unordered list with a class of "users".
- Used a loop to iterate through each user (@users.each do |user|).
- For each user, created a list item (li) that contains:
- A link to the user's name using the link_to method.
- The word "created".
- The time since the user was created, calculated using the time_ago_in_words method.
- Closed the loop.
- Closed the unordered list.
- In summary, this code generates a list of users with links to their profiles, along with the time since each user was created.

app/views/users/show.html.erb

- Defined a Ruby on Rails view file for the User page
- Displayed the user's name and email using instance variables
- Checked if the current user is the same as the displayed user and displayed "Edit Account" and "Delete Account" links if so
- Checked if there are any registrations associated with the user and displayed them if so
- For each registration, displayed the associated event's image, name, and registration date using instance variables and methods like link_to and time_ago_in_words
- Provided an option to unregister from the event using a "Unregister" button with a confirmation dialog
- Checked if there are any liked events associated with the user and displayed them if so
- For each liked event, displayed the event's name and image using instance variables and link_to method

Eventz

views

app/views/sessions/new.html.erb

- Created a sign-in page for the Ruby on Rails web application
- Included a form that will submit the user's email and password
- Used the form_with method to generate the form
- Specified the form's submission URL as session_path
- Added a label for the email field and generated an email input field using the email_field method, with the autofocus option set to true
- Added a label for the password field and generated a password input field using the password_field method
- Included a submit button with the label "Sign In" using the submit method.

app/views/shared/_errors.html.erb

- Checked if there are any errors with the object using the if statement.
- Created a section with a class of "errors" to display the error messages.
- Added a header with the text "Oops! Your form could not be saved." to inform the user of the error.
- Added a subheader with the text "Please correct the following [number of errors]:", using the pluralize method to correctly format the number of errors.
- Created an unordered list to display the error messages.
- Used a do loop to iterate through each error message in object.errors.full_messages.
- Added each error message to a list item using the li tag and displayed it within the unordered list using the message variable.

Eventz

views

views/layouts/_flash.html.erb

- Checked if there is a message associated with flash[:notice] by using an if statement with the flash[:notice] variable.
- If there is a message, it creates a div with a class "flash notice" and displays the message using the <%= flash[:notice] %> syntax.
- Checked if there is a message associated with flash[:alert] by using an if statement with the flash[:alert] variable.
- If there is a message, it creates a div with a class "flash alert" and displays the message using the <%= flash[:alert] %> syntax.
- To summarize, this code snippet checks if there are any messages stored in the flash object associated with the :notice or :alert keys, and if there are, it displays them in a styled div element with appropriate CSS classes.

views/layouts/_footer.html.erb

- Created a footer for the website using the HTML <footer> tag with a paragraph <p> with the footer content.

views/layouts/_header.html.erb

- Created a header section with a navigation bar.
- Included a link to the home page with a logo image.
- Added a list of links to the navigation bar with upcoming events, past events, free events, and recent events.
- Checked if there is a current user logged in.
- If a current user is available, displayed the user's name and a link to their profile page.
- If no current user is available, provided links to sign in, sign up, and sign out pages.

views/layouts/application.html.erb

- Rendered the header, flash messages, and footer partials within the body tag of the HTML document.
- Used the yield keyword to render the specific view file for the current page being visited by the user.

Eventz

config

config/storage.yml

- added amazon aws s3 information to use it with production environment

config/environments/production.rb

- defined amazon aws s3 as the service to be used by active storage in the production environment