# Webbit

**models**
**app/models/comment.rb**

- Included the VotesCountable module in the Comment class to add functionality for vote counting.
- Specified that a Comment belongs to both a Submission and a User using the belongs_to method.
- Validated that a Comment must have a non-empty reply using the presence validation.
- Used the acts_as_votable gem to enable voting on comments.

**app/models/community.rb**

- The Community class also the FriendlyId module, which provides a way to generate human-readable slugs for model instances.
- The before_save callback method is defined to format the name attribute of the community instance before saving it to the database.
- The Community class belongs to a User instance.
- The Community class has many Submission instances.
- The Community class has many Subscription instances.
- The Community class has many User instances, through the subscriptions association.
- The friendly_id method is used to generate a slug for the community instance based on its title attribute.
- The private method format_name is defined to format the name attribute of the community instance by capitalizing the first letter of each word and removing any spaces. The gsub! method is used to modify the name attribute in place, instead of returning a new value.

# Webbit

## models

### app/models/submission.rb

- Extended FriendlyId to generate human-friendly URLs.
- Included the VotesCountable module to allow users to vote on submissions.
- Mounted the SubmissionImageUploader and SubmissionVideoUploader to upload images and videos respectively.
- Created associations between Submission, User, and Community models.
- Defined a one-to-many relationship between Submission and Comment models, and set the dependent option to destroy comments related to a submission if the submission is destroyed.
- Enabled Rich Text for the body attribute to allow formatting with a WYSIWYG editor.
- Validated the presence of the title attribute.
- Validated the length of the body attribute, ensuring that it doesn't exceed 8000 characters.
- Validated the URL attribute using the url validator, allowing it to be blank.
- Defined custom validation methods to ensure that either an image or a video is uploaded, but not both, and that either a URL or content is provided, but not both.
- Used the acts_as_votable gem to make the Submission model votable.
- Defined a friendly_id method to generate slugs from the title attribute.
- Defined two private methods to perform custom validation.

### app/models/subscription.rb

- Created a new class called Subscription which inherits from ApplicationRecord class in Rails.
- Defined a belongs_to association with Community and User models respectively.
- The belongs_to association specifies that a Subscription record belongs to a single Community and a single User.
- By defining this association, ActiveRecord will create a foreign key column in the subscriptions table to reference the primary key in the associated Community and User tables.
- This allows us to easily retrieve a Community or User associated with a Subscription record using ActiveRecord's association methods.

# Webbit

**app/models/user.rb**

- Extended the FriendlyId module to create a custom slug for the User model.
- Added a before_create callback to generate a random unsubscribe_hash for each new user.
- Used the devise gem to add authentication functionality with several modules enabled.
- Defined several associations between the User model and other models such as subscriptions, communities, submissions, and comments.
- Validated the uniqueness and presence of the username attribute.
- Created a subscribed_submissions association to allow users to access all submissions of communities they have subscribed to.
- Implemented voting functionality through the acts_as_voter gem.

## uploaders

**app/uploaders/submission_image_uploader.rb**

- Included RMagick and MiniMagick support.
- Chose to use file storage for the uploader.
- Overrode the directory where uploaded files will be stored, with a sensible default.
- Provided a default URL if no file has been uploaded.
- Processed files as they are uploaded, by scaling them to a specific size.
- Created two different versions of the uploaded files - thumbnail and large - by resizing them to fit different dimensions.
- Added an allowlist of extensions that are allowed to be uploaded - jpg, jpeg, gif, and png.
- Overrode the filename of the uploaded files.

**app/uploaders/submission_video_uploader.rb**

- The SubmissionVideoUploader class is a subclass of CarrierWave::Uploader::Base, which is used to upload files.
- The uploader can use either RMagick or MiniMagick for image manipulation.
- The uploader is set to use file storage.
- The store_dir method sets the directory where uploaded files will be stored.
- The default_url method provides a default URL for uploaded files that haven't been assigned a file.
- The extension_allowlist method sets an allowlist of file extensions that can be uploaded.
- There are methods commented out for processing and scaling images, as well as creating different versions of uploaded files.
- The filename method can be overridden to set a custom filename for uploaded files.
- In summary, the SubmissionVideoUploader class is a versatile uploader that can handle different types of video files, set storage and directory settings, and apply various image manipulation techniques.

# Webbit

## controllers

### app/controllers/application_controller.rb

- Included the Pagy::Backend module into the ApplicationController class.
- Created a before_action hook named "configure_permitted_parameters" that allows users to sign up and edit their profile with their username, and comment_subscription attributes.
- Created another before_action hook named "find_communities" that gets all the communities in the app and sorts them alphabetically.
- Created a search method that takes a query parameter and searches for communities, submissions, and users that contain the query.
- The search results are stored in instance variables named "@community_results", "@submission_results", and "@user_results", respectively.

### app/controllers/comments_controller.rb

- Initialized the CommentsController with before_actions to authenticate the user, set the comment, set the submission, and find the comment for upvoting and downvoting.
- Defined a new method that does not require any parameters.
- Defined a create method that creates a new comment with the permitted parameters and sets the user_id to the current_user's id.
- Used respond_to to handle the format of the response and deliver a new_response email if the comment was successfully saved.
- Defined a show method that does not require any parameters.
- Defined an edit method that responds with a js format and does not require any parameters.
- Defined an update method that updates the comment's parameters and redirects to the submission path with a success notice or renders the edit page and returns the comment's errors as a json.
- Defined a destroy method that destroys the comment and redirects to the submission path.
- Defined an upvote method that responds to the format of the request and checks if the current user has not voted for the comment.
- Defined a downvote method that is similar to the upvote method.

# Webbit

## controllers

**app/controllers/communities_controller.rb**

- The CommunitiesController class is defined, which inherits from ApplicationController.
- Two before actions are defined using before_action method. One is to set the community before show, edit, update, and destroy actions. The other is to authenticate the user before all actions except show and index.
- The index action retrieves all communities from the database and assigns them to the @communities instance variable.
- The show action renders the community with the given ID.
- The new action initializes a new community with the current user's ID and assigns it to the @community instance variable.
- The edit action renders the community with the given ID for editing.
- The create action initializes a new community with the current user's ID and saves it to the database. If the community is saved successfully, the user is redirected to the community's page with a success message. Otherwise, the user is redirected to the new community form with an error message.
- The update action updates an existing community with the given parameters and saves it to the database. If the community is updated successfully, the user is redirected to the community's page with a success message. Otherwise, the user is redirected to the edit community form with an error message.
- The destroy action deletes the community with the given ID from the database. If the community is deleted successfully, the user is redirected to the communities index page with a success message.
- In summary, this code defines a controller for managing communities, with actions for showing, creating, editing, updating, and deleting communities. It also ensures that users are authenticated before performing certain actions, and assigns communities to the current user.

**app/controllers/home_controller.rb**

- Created a new controller called HomeController
- Defined an action called index using the def keyword and giving it a name of "index"

**app/controllers/premium_controller.rb**

- The before_action method is used to run the authenticate_user! method before the methods new, create and destroy are executed.
- The new method checks if the user is signed in and already subscribed. If true, it redirects the user to the root path with a notice.
- The create method sets the Stripe API key and retrieves the plan ID from Rails credentials. It then retrieves the plan details from Stripe, gets the Stripe token from the parameters, and creates a customer with the token and the user's email. If the user already has a Stripe ID, it retrieves the customer data from Stripe, and if not, it creates a new customer. A subscription is then created with the plan ID and associated with the customer. Finally, the user's information is updated with the Stripe customer and subscription IDs and the subscription details. If the user's card information is included in the parameters, it is also updated.
- The destroy method sets the Stripe API key and retrieves the customer data from Stripe using the current user's Stripe ID. The subscription associated with the current user's Stripe subscription ID is then retrieved and deleted. The user's information is then updated to remove the Stripe subscription ID and mark them as unsubscribed. Finally, the user is redirected to the root path with a notice.

# Webbit

## controllers

**app/controllers/submissions_controller.rb**

- Defined two before_action callbacks: set_submission and authenticate_user!. set_submission method sets @submission instance variable for the show, edit, update, and destroy actions. authenticate_user! method ensures that the user is logged in for all actions except show and index.
- Defined an index action that sets @submissions instance variable based on the user's subscription status. If the user is signed in, it shows their subscribed submissions; otherwise, it shows all submissions. Pagination is implemented using pagy gem.
- Defined a show action that sets @comment and @community instance variables for displaying the comments and community of the selected submission.
- Defined a new action that sets @submission instance variable to a new instance of Submission associated with the current user.
- Defined an edit action that does not have any functionality implemented.
- Defined a create action that sets @submission instance variable to a new instance of Submission associated with the current user and saves it in the database if it is valid. Responds with HTML and JSON formats accordingly.
- Defined an update action that updates the @submission instance variable in the database if the submitted parameters are valid. Responds with HTML and JSON formats accordingly.

**app/controllers/subscriptions_controller.rb**

- Added before_actions to authenticate the user and set the community for create and destroy actions.
- Defined the create action that creates a new subscription record for the current user and the community they are subscribing to. If the subscription record already exists for the user and community, it returns the existing record.
- Responded with a redirect to the community page with a success notice if the request is in HTML format. If the request is in JavaScript format, it renders create.js.erb file.
- Defined the destroy action that finds the subscription record for the current user and the community they want to unsubscribe from. It destroys the record and returns the notice.
- Responded with a redirect to the community page with a success notice if the request is in HTML format. If the request is in JavaScript format, it renders destroy.js.erb file.
- Defined a private method to set the community object based on the community_id parameter.

**app/controllers/users_controller.rb**

- Defined a method named "show".
- In the "show" method, used the "User" model to find a user by their "id" parameter, which was made friendly by the "friendly" gem.
- Set the found user to an instance variable named "@user".
- The purpose of this code is to show the details of a user in the view with the corresponding id parameter.

# Webbit

## models/concerns

**app/models/concerns/votes_countable.rb**

- Imported the 'active_support/concern' library for use in the module.
- Created a module called 'VotesCountable'.
- Used the 'extend ActiveSupport::Concern' statement to allow the module to be included as a concern.
- Defined a method called 'total_vote_count' to calculate the total number of votes for a given object that includes this module.
- Used 'self.get_upvotes.size' to get the number of upvotes and 'self.get_downvotes.size' to get the number of downvotes for the object.
- Subtracted the number of downvotes from the number of upvotes to get the total vote count.
- Used the '.to_s' method to convert the total vote count to a string.
- Included the 'total_vote_count' method in any class that includes the 'VotesCountable' module.

# Webbit

**views**

**app/views/active_storage/blobs/_blob.html.erb**

- Created a file that handles attachments, allowing users to upload and view images.
- Used conditionals to check if the attachment file is representable and display the image accordingly using image_tag.
- Added a caption to the image if available.
- Used number_to_human_size to display the size of the attachment in a human-readable format.

**app/views/admin/submissions/index.html.erb**

- Created a view for an admin submission feed with the label "Admin Submission Feed".
- Rendered a partial for navigation specifically for the admin.
- Defined a block of content to be rendered as primary content.
- Rendered a partial for each submission in the @submissions collection.
- Rendered a basic layout to display the page.

**app/views/admin/_nav.html.erb**

Created a header section for the admin page with a yellow background color.
Used the flexbox layout to align the items within the header section.
Displayed the "Admin" text in a mono font with a normal font weight.
Added two links to the header section using the link_to helper method provided by Rails.
The first link is labeled "User" and leads to the admin_users_path.
The second link is labeled "Submissions" and leads to the admin_submissions_path.
Applied some CSS classes to style the links as black-colored text and add some margin to the second link.

# Webbit

## views

**app/views/admin/users/index.html.erb**

- Used the content_for method to set the title of the admin user feed page to "Admin User Feed".
- Rendered the admin navigation bar using the render method and passing the name of the partial as an argument.
- Created a container with a white background, rounded corners, and shadow using CSS classes.
- Loop through each user in the @users instance variable using the each method.
- For each user, created a div with the classes flex, flex-wrap, justify-start, and items-center.
- Inside this div, created another div with the class lg:w-1/4 w-full, containing a link to the user's profile using the link_to method and the gravatar_for helper method to display the user's avatar.
- Added the user's username to the link using the user.username method.
- Created another div with the class lg:w-1/4 w-full lg:m-0 mb-2, containing the user's email.
- Created another div with the class lg:w-1/4 w-full text-sm lg:m-0 mb-2, containing the text "Account created" followed by the time ago in words since the user's created_at timestamp using the time_ago_in_words method.
- Created another div with the class lg:w-1/4 w-full lg:m-0 mt-2 lg:text-right text-left, containing a link to delete the user's profile using the link_to method with the method: :delete option and a confirmation message.
- End the loop and the container div.

**app/views/application/_basic_layout.html.erb**

- Created a Ruby on Rails view template to render a web page using HTML and ERB templating language.
- Defined a container element with a flex layout and responsive design classes to adapt to different screen sizes using CSS framework classes.
- Created two nested container elements, the first with a width of 2/3 and the second with a width of 1/3 using grid classes from the CSS framework.
- Defined two div elements inside the container elements with a white background color, shadow and rounded corners using CSS classes from the framework.
- Used the yield keyword to render dynamic content in the div elements defined with the primary_content and secondary_content symbols respectively.
- The primary_content div element is rendered in the first container element and the secondary_content div element is rendered in the second container element.

**app/views/application/_community_result.html.erb**

- Render the partial view named "community" with a local variable called "community" set to the value of "community_result".
- The rendered view will display the details of the given community object.
- The partial view will be included in the current view and displayed to the user.

**app/views/application/_feed_ad.html.erb**

- Added an image of Bill Murray using the "img" tag from an external URL "http://www.fillmurray.com/68/68"
- Created a container using "div" tag with a light gray background color and a border.
- Added the "flex" class to the container to display the content using the flexbox layout.
- Created a container for the ad information using the "div" tag with a "leading-normal" class for text styling, and added the "flex-1" class to take up the remaining space in the container.
- Added an italicized text with gray color for the ad information using the "text-gray-700" class.
- Created a container for the "Remove Ads" link using the "div" tag and added the "flex" class to position it to the right of the ad information container.
- Added the "link_to" helper method to create a hyperlink for the "Remove Ads" text. It is linked to the "new_premium_path" route and styled using the "text-xs" class for small text size.

# Webbit

**views**

**app/views/application/_submission_result.html.erb**

- Render the "submission" partial view passing the "submission_result" object as a parameter.
- Display the resulting HTML in the view where this code is written.

**app/views/application/_user_result.html.erb**

- Used the <%= %> syntax to embed Ruby code into an HTML view.
- Called the gravatar_for helper method and passed in the user_result object as an argument to generate a gravatar image for the user.
- Wrapped the gravatar image inside a div element.
- Added a div element with a class of "pl-4" to contain the user's username and join date.
- Created a link to the user's profile page using the link_to helper method and passing in the user_result.username and profile_path(user_result) as arguments.
- Styled the username link using a class of "link".
- Added an italicized text element with a class of "text-sm text-gray-600" to display the time since the user's account was created using the time_ago_in_words helper method and passing in the user_result.created_at attribute as an argument.
- Wrapped the entire section inside a div element with classes of "p-4 flex flex-wrap justify-start items-center bg-none border-r border-l border-b border-gray-400" to add some padding and styling to the section.

**app/views/application/search.html.erb**

- Created a content_for block to display the community label with the search query results.
- Created a div container with a gray background and white text, with a title displaying the search query results.
- Implemented an if statement to check if the submission_results exists, and if it does, display them in a white background shadow rounded-sm div container.
- Implemented an if statement to check if the community_results exists, and if it does, display them in a white background shadow rounded-sm div container with the community label.
- Implemented an if statement to check if the user_results exists, and if it does, display them in a white background shadow rounded-sm div container with the user label.
- Implemented an if statement to check if submission_results, community_results, and user_results are empty, and if they are, display a message informing that there are no search results.

# Webbit

**views**

**app/views/comments/_comment_voting.html.erb**

- Check if the user is not signed in.
- If the user is not signed in, display a link to the new user session path, with an upvote SVG icon, a vote count, and a downvote SVG icon, all styled with text and color.
- If the user is signed in, display links to upvote and downvote submission comment paths for the comment, with corresponding SVG icons, a vote count, and remote attribute for asynchronous updates.

**app/views/comments/_comment.html.erb**

- The code renders a comment in the application.
- It creates a div tag with an ID and a class that uses Flexbox to display the items in a row.
- The first child div tag renders a partial for comment voting, passing in the comment as an argument.
- The second child div tag uses Flexbox to align items in a column and indents it by 4 spaces.
- It renders another partial that shows who posted the comment and passes in the comment's submission as an argument.
- It renders the comment's reply text in a div tag with a grey text color and normal line height.
- It creates another div tag to display two links that allow editing or deleting the comment.
- The first link renders an SVG icon for editing the comment, and it's wrapped in a link tag using link_to helper method. It uses the edit_submission_comment_path to generate the appropriate link and sets the remote attribute to true to allow AJAX requests.
- The second link renders an SVG icon for deleting the comment, wrapped in a link tag using the link_to helper method. It uses the submission_comment_path to generate the appropriate link, sets the HTTP method to delete, and sets the data-confirm attribute to show a confirmation dialog before deleting the comment.
- The if statement checks if the current user is the author of the comment and shows the edit and delete links only to the author of the comment.

**app/views/comments/_form.html.erb**

- Created a form using Ruby on Rails to allow users to submit comments on a submission.
- Used the form_for method to create the form with two models, @submission and @comment.
- Set the form to submit remotely using the remote: true option.
- Added a text area for users to enter their comment using the f.text_area method.
- Set the text area to have a class of "input h-32 border border-gray-200 resize-none" and a placeholder of "What are your thoughts?".
- Added a submit button to the form using the f.submit method.
- Set the submit button to have a class of "btn btn-small btn-purple block w-full text-center mt02 lg:mt-0 lg:text-left lg:absolute lg:bottom-0 lg:right-0 lg:mb-2 lg:mr-2 lg:w-auto lg:inline-flex".

# Webbit

**views**

**app/views/comments/_modal.html.erb**

- Rendered a form partial in the Ruby on Rails view template.
- The form is used for editing a comment.
- The rendered form contains HTML code for displaying an H1 tag with the text "Edit comment".
- The H1 tag has CSS classes assigned to it for setting its font size, weight, and color.
- The form partial is passed as an argument to the "render" method, which is a built-in Rails method for rendering a partial template.
- The "render" method looks for a partial template with the specified name ("form") and inserts its HTML code into the view template at the point where the "render" method is called.

**app/views/comments/_vote.js.erb**

- Selected the HTML element with ID comment-<%= @comment.id %> .js-votes using querySelector method and stored it in the commentVote variable.
- Selected the HTML element with class .js-notice using querySelector method and stored it in the dynamicNotice variable.
- Set the innerHTML property of the commentVote element to the value returned by escape_javascript(@comment.total_vote_count).
- Set the innerHTML property of the dynamicNotice element to a string that contains HTML elements and interpolated notice variable.
- Used the render_svg method to render an SVG icon and included it in the HTML string.
- Set a timeout of 3 seconds (3000 milliseconds) using the setTimeout method.
- After the timeout, set the innerHTML property of the dynamicNotice element to an empty string, effectively removing the HTML elements added in step 4.

**app/views/comments/create.js.erb**

- Get the element with the ID "comments" and assign it to the variable comments.
- Get the element with the ID "comment_reply" and assign it to the variable commentField.
- Insert the HTML generated by the escape_javascript method, which renders the @comment object, before the end of the comments element using the insertAdjacentHTML method.
- Set the value of the commentField element to an empty string, effectively clearing the input field.
- In summary, this code updates the comments section of a web page by appending a new comment and clearing the input field where new comments can be entered.

# Webbit

**views**

**app/views/comments/downvote.js.erb**

- Render the "vote" partial in the current view
- The content of the "vote" partial is displayed at the location where this line of code is placed.
- The rendered "vote" partial is an independent section of code that can be reused across different views.
- The exact functionality of the "vote" partial is not clear without examining its implementation.

**app/views/comments/edit.js.erb**

- Retrieved the DOM element with id 'edit-comment' and stored it in the variable 'editComment'.
- Retrieved the first element with class 'modal' inside the 'editComment' element and added the class 'is-active' to it.
- Retrieved the DOM element with class 'modal-inner' and stored it in the variable 'modalInner'.
- Inserted HTML code using 'insertAdjacentHTML' method into the beginning of the 'modalInner' element.
- The HTML code is generated using Ruby on Rails syntax to render a partial template named 'modal' for the 'comments' resource.
- The rendered HTML code is properly escaped using the 'escape_javascript' helper method to prevent possible security vulnerabilities.

**app/views/comments/upvote.js.erb**

- Included a partial view named "vote" using the render method in the Rails application.
- The purpose of the partial view is to display a voting mechanism for a particular resource such as a post, comment, or article.
- The code is utilizing the ERB templating language to dynamically render the partial view in the appropriate location within the HTML document.
- The render method provides a convenient way to reuse code and keep views DRY (Don't Repeat Yourself).

# Webbit

**views**

**app/views/communities/_community_header.html.erb**

- Created a dynamic view file with embedded Ruby code to display the name of the community on the page.
- Used a Bootstrap class to style the container with a purple background and white text.
- Applied responsive design by using different padding values for large and small screens.
- Used a bold font to emphasize the title of the page.

**app/views/communities/_community.html.erb**

- Created a card element with a flex container, containing two divs, with one div having a link and heading, and the other div containing links with icons.
- Used the "link_to" helper method to create a hyperlink to the "community" page with the "community" variable as its parameter, and added a CSS class to the link for styling purposes.
- Rendered an SVG icon using the "render_svg" helper method, with the icon's name and styles as its parameters, and added it to the link using string interpolation to make it more dynamic.
- Added the "community.name" variable to the link's text content using string interpolation.
- Added a heading element containing the "community.title" variable, which displays the title of the community.
- Created another link using the "link_to" helper method to link to the "community" page, added a CSS class to the link for styling purposes, and rendered an SVG icon using the "render_svg" helper method with the "icons/view" name and styles as its parameters.
- Used a conditional statement to check if the current user is the author of the community, and if true, rendered two more links with SVG icons using the "link_to" and "render_svg" helper methods to edit or delete the community with appropriate CSS classes and titles, and added a confirmation message for the delete link using the "data" attribute.

# Webbit

**views**

**app/views/communities/_form.html.erb**

- Created a form using form_with() helper method to submit a community object
- Checked if there are any errors on the community object
- If errors exist, displayed them using error.full_message
- Created four input fields for name, title, description, and sidebar using form.label and form.text_field helper methods
- Created a larger text input field for description and sidebar using form.text_area helper method
- Added a hidden field for user_id if the user is signed in
- Created a submit button using form.submit helper method

**app/views/communities/_sidebar.html.erb**

- Created a view file for displaying information about a community.
- Used Ruby on Rails syntax to insert dynamic information from the "community" object.
- Rendered an SVG icon using the "render_svg" helper function.
- Checked if the current user is the author of the community and added an edit button if true.
- Displayed the title, description, and sidebar of the community.
- Used the "pluralize" helper function to correctly display the number of submissions for the community.
- Displayed the number of subscribers for the community.

**app/views/communities/edit.html.erb**

- Included community label content
- Created a primary content container
- Displayed the name of the community being edited
- Rendered the form to edit the community
- Created a secondary content container
- Rendered an SVG image for community
- Listed the required community fields in an unordered list
- Rendered the basic layout which includes the primary and secondary content containers

# Webbit

**views**

**app/views/communities/index.html.erb**

- Render the "All Communities" label in the community section
- Render the communities in the primary content section
- Render the "New Community" link in the secondary content section, but only if the user is signed in
- Render the "basic_layout"

**app/views/communities/new.html.erb**

- The code creates a view for creating a new community.
- The view includes a form, which is rendered using the 'form' partial and passed the @community object.
- The form has fields for the community's name, title, and description.
- The view also includes a section for secondary content, which includes an image and a list of required fields.
- The view uses the 'basic_layout' partial to render the overall layout of the page.

**app/views/communities/show.html.erb**

- Sets the title of the page to the name of the community.
- Sets the community label for the page.
- Renders the community header partial.
- Iterates over each submission related to the current community and renders the submission partial for each one.
- If the user is not an active subscriber, injects a feed ad between submissions.
- Renders the sidebar partial for the current community in the secondary content area.
- Renders the basic layout for the page.
- Comments out a paragraph that displays a notice (likely for debugging purposes).
- Comments out several paragraphs that display details about the community and provide links to edit or go back (likely for development purposes).

**views**

**app/views/devise/passwords/new.html.erb**

- Created a template using Ruby on Rails framework.
- Used the content_for helper method to define a named yield block for "devise_form".
- Rendered a partial "devise/shared/form_wrap" within the "devise_form" yield block.
- Added a heading element "h2" with a text "Forgot your password?".
- Defined a form using the form_for helper method.
- Set the "resource" parameter to the object passed in.
- Set the "as" parameter to "resource_name" to use the object name as the form's prefix.
- Set the "url" parameter to the password_path helper method to set the form's action.
- Set the "html" parameter to include the HTTP method as a post request.
- Rendered another partial "devise/shared/error_messages" with the resource parameter to display any error messages.
- Created a div element with a label for an email input field and set it to autofocus and autocomplete with the value of email.
- Created another div element with a submit button with the text "Send me reset password instructions".
- Ended the form with <% end %> and rendered the "devise/shared/links" partial.
- Ended the "devise_form" yield block with <% end %>.
- Rendered the "devise/shared/form_wrap" partial again.

**app/views/devise/passwords/edit.html.erb**

- Created a view for changing the password.
- Generated a form using form_for helper method.
- Rendered error messages with the resource object.
- Added a hidden field for the reset_password_token.
- Created a div for the "New password" label and displayed the minimum password length if it's set.
- Created a password field with autofocus, autocomplete, and the "input" class.
- Created a div for the "Confirm new password" label and created a password field with autocomplete and the "input" class.
- Created a div for the submit button and rendered the "Change my password" button with the "btn" and "btn-purple" classes.
- Rendered links for other devise actions using the "links" partial.
- Wrapped the form and links sections in a "devise_form" content section and a "form_wrap" partial.

# Webbit

**views**

**app/views/devise/confirmations/new.html.erb**

- Render a content block called "devise_form"
- Render a heading with the text "Resend confirmation instructions"
- Create a form using the Rails form_for helper, with the resource and confirmation path as arguments, and set the HTTP method to POST
- Render any error messages for the resource using the shared/error_messages partial, passing in the resource as an argument
- Render a label and email input field for the email attribute of the resource, with autofocus, email autocomplete, and a default value of the unconfirmed email if the resource has a pending reconfirmation, or the email if not
- Render a submit button with the text "Resend confirmation instructions"
- End the form
- Render a horizontal rule
- Render the shared/links partial
- End the content block for "devise_form"
- Render the shared/form_wrap partial.

**app/views/submissions/_actions.html.erb**

- Display a link to the submission's comments section with a conversation icon, the number of comments, and a gray color style. The link uses the submission_path route helper and the comments anchor.
- Display a section on the right side of the screen with two links if the current user is the author of the submission.
- The first link uses the edit_submission_path route helper to redirect the user to the submission's edit page when clicked. The link uses an edit icon and a gray color style.
- The second link uses the submission route helper, a delete method, and a confirmation dialog when clicked. The link uses a delete icon and a gray color style.

# Webbit

**views**
**app/views/submissions/_form.html.erb**

- Created a form using form_with helper method to submit a model named "submission".
- Checked if there were any errors in submission using submission.errors.any?.
- Displayed a list of errors if there were any, using error_explanation and full_message methods.
- Created a select tag using form.select method to select a community from a collection of all communities.
- Added a prompt to the select tag using { prompt: "Choose a community" }.
- Added a class to the select tag using { class: "select" }.
- Created a div with class "relative".
- Rendered a select tag for choosing a community using Community.all.collect method to collect all communities and display their names and ids in the select tag.
- Created a div with class "pointer-events-none absolute inset-y-0 right-0 flex items-center px-2 text-gray-700".
- Rendered an SVG icon inside the div using render_svg method.
- Created a div with class "mb-6 w-full lg:w-2/5".
- Created an hr tag with class "border border-gray-400 my-4".
- Created a div with class "pb-10".
- Created a nav tag with class "border-box".
- Created an unordered list with id "nav-tab" and class "tabs list-none flex flex-wrap justify-between".
- Created three list items inside the unordered list.
- Created links inside the list items using link_to method.
- Rendered SVG icons and text inside the links.
- Added "active" class to the first list item using "active w-1/3 text-center cursor-pointer" classes.
- Added "tab-first" class to the first link using "tab tab-first" classes.
- Added "tab" class to the second link.
- Added "tab-last" class to the third link.

# Webbit

**views**

**app/views/submissions/_guidelines.html.erb**

- Here's what I did in the Posting to Webbit project:
- Created an HTML file with a header element and an ordered list (ol) element with five list items (li).
- Assigned classes to the header and list elements using the "class" attribute for styling purposes.
- Used the "border-b", "font-semibold", and "py-1" classes to style the list items with a bold font and a bottom border.
- Used the "leading" and "p-0" classes to remove padding and adjust the line spacing of the list items.
- Used the "text-sm" class to set the font size of the list items to a smaller size.

**app/views/submissions/_posted_by.html.erb**

- Created a view file with a paragraph tag to display the submission user and time ago in words.
- Used link_to and profile_path helpers to generate a link to the user's profile.
- Created a controller action to handle the logic for displaying the submission user and time ago in words.
- Used time_ago_in_words helper to calculate the time elapsed since submission.
- Implemented a conditional rendering logic to handle cases where the submission user may not exist.

**app/views/submissions/_submission.html.erb**

- Render a border element with an ID based on the submission ID.
- Render a div with flex items and wrap contents.
- Render a div with a class for voting that contains a partial for rendering voting buttons.
- Render a link that directs to the submission page and includes an image tag with a large URL size, or a video tag with a URL that includes specific styling.
- Render a flex container for items with a left margin of 4 spaces.
- Render a flex item that fills the available space.
- Render a heading 3 element with a flexible wrap and item centering. It includes a link to the submission title with a class of "no-underline" and bold font-weight. If the submission has a URL, a truncated version of it is shown in monospace font and a blue color, with a link to the full URL.
- Render a link to the community path with a class of "no-underline" and gray color. It shows the community name and has an inline-block style. It also includes a period and renders a partial to display who posted the submission and actions that can be taken on it.

# Webbit

**views**

**app/views/submissions/_vote.js.erb**

- Retrieve the HTML element that has an ID equal to the concatenation of the string "#submission-" and the ID of the current submission object. Store it in the variable "vote".
- Retrieve the HTML element with a class of "js-notice". Store it in the variable "dynamicNotice".
- Set the innerHTML of the "vote" element to the total vote count of the current submission object, which has been escaped using the "escape_javascript" method.
- Set the innerHTML of the "dynamicNotice" element to a string of HTML that displays a message to the user. The message is enclosed in backticks and contains a div element with the classes "bg-white", "shadow", and "rounded-lg". Within this div element, there is another div element with the classes "px-2", "py-4", "text-purple-600", "text-center", "leading-normal", "font-medium", "font-sans", "flex", "justify-center", and "items-center". This div contains an SVG icon, followed by a div element with the class "flex-1", which contains the message itself. The message is stored in the "notice" variable.
- Set a timeout function to run after 3 seconds. This function sets the innerHTML of the "dynamicNotice" element to an empty string, effectively removing the message from the page.

**app/views/submissions/_voting.html.erb**

- Checked if the user is signed in
- If the user is signed in:
- Rendered an upvote button for the submission using link_to and render_svg
- Showed the total vote count for the submission
- Rendered a downvote button for the submission using link_to and render_svg
- If the user is not signed in:
- Rendered an upvote button that links to the new user session path using link_to and render_svg
- Showed the total vote count for the submission
- Rendered a downvote button that links to the new user session path using link_to and render_svg
- Used AJAX by setting remote: true in the link_to options and updated the vote count on the page without a full page reload when the user clicks the upvote or downvote button.
- Used the HTTP method PUT to update the submission's vote count when the user clicks the upvote or downvote button.

# Webbit

**views**

**app/views/submissions/downvote.js.erb**

- Render the "vote" partial template.
- The rendered output from the partial template will be included in the final HTML response.

**app/views/submissions/edit.html.erb**

- Declared a content block for the label "Edit Submission".
- Declared a content block for the primary content.
- Created a div with classes "lg:p-10 pt-10 pb-10 px-4" to display the editing form.
- Displayed the submission title with a h1 tag.
- Rendered the form partial with the submission instance variable.
- Declared a content block for the secondary content.
- Created a div with classes "px-8 pt-6 pb-8" to display additional information.
- Displayed an image using the image_tag helper.
- Rendered the "guidelines" partial.
- Rendered the "basic_layout" partial.

**app/views/submissions/index.html.erb**

- Created a title for the page with the text "Submissions" using the title helper method.
- Set the content for the label with the text "My Feed" using the content_for helper method and passing in the symbol :community_label as the first argument.
- Set the primary content of the page using the content_for helper method and passing in the symbol :primary_content as the first argument.
- Checked if there are any submissions present by using the exists? method on the @submissions variable.
- If there are submissions, looped through each submission using the each_with_index method and rendered the partial _submission.html.erb by passing in the submission object as an argument.
- If there are no submissions, displayed a message indicating that there are no submissions and provided a link to create a new submission using the link_to helper method.
- Set the secondary content of the page using the content_for helper method and passing in the symbol :secondary_content as the first argument.
- Checked if the user is signed in by using the user_signed_in? method.
- If the user is not signed in, displayed an image and text encouraging the user to explore the site and provided a link to learn more about how the site works using the image_tag and link_to helper methods.
- If the user is signed in, displayed a message indicating that this is the user's personal frontpage and provided links to create a new submission or a new community using the link_to helper method.

# Webbit

**views**
**app/views/submissions/new.html.erb**

- Created a new submission page for the user to fill in the form details
- Defined the primary content section for the new submission page
- Rendered a form to capture submission details
- Set the submission instance variable to pass to the form partial
- Defined the secondary content section for the new submission page
- Rendered the guidelines partial in the secondary content section
- Defined the community_label content section for the new submission page
- Set the content of the community_label section to "Create Submission"
- Rendered the basic layout for the new submission page with the three content sections (primary_content, secondary_content, and community_label)

# Webbit

## views

**app/views/submissions/show.html.erb**

- Render the title of the submission by calling the title helper method with the value of @submission.title.
- Set the content of the community_label section to the value of @submission.title by calling the content_for helper method with the symbol :community_label and the value of @submission.title.
- Render the community_header partial with the local variable community set to the value of @submission.community.
- Set the primary content of the page by calling the content_for helper method with the symbol :primary_content.
- Create a div element with the id of submission-<%= @submission.id %> and the classes of p-4 pb-6 pr-6.
- Inside the div element, create a div element with the classes of flex flex-row flex-wrap items-start justify-between.
- Inside the div element, create a div element with the class of submission-voting text-xs pr-2 and render the voting partial with the local variable submission set to the value of @submission.
- Inside the div element, create a div element with the class of submission-content.
- Render the posted_by partial with the local variable submission set to the value of @submission.
- Render an h1 element with the classes of text-xl font-bold and the value of @submission.title.
- Create a div element with the classes of leading-normal text-base lg:pr-6 mb-5 border-b border-gray-200 lg:pr-6.
- If @submission.submission_image.present?, render an image_tag element with the value of @submission.submission_image.large.url and the class of max-w-full my-2.
- If @submission.submission_video.present?, render a video_tag element with the value of @submission.submission_video.url, the class of max-w-full my-2, and the controls attribute set to true.
- If @submission.url.present?, create a div element with the classes of p-3 border rounded my-2 flex justify-center items-center and render an SVG icon with the class of mr-2 text-gray-600 fill-current and the value of "icons/link". Also, render a link with the class of lg:text-xl text-lg no-underline, the value of @submission.url, and the value of @submission.url as the link text.
- Render the @submission.body if it exists.
- Render the actions partial with the local variable submission set to the value of @submission.
- If user_signed_in? is false, render a p element with the classes of text-sm text-gray-600 my-2 and display the links to the login and sign up pages by calling the link_to helper method with the text "Login" and the path to the login page as the first argument, and the text "Sign up" and the path to the sign up page as the second argument.
- If user_signed_in? is true, render the comments/form partial.
- Render the comments partial with the value of @submission.comments.

# Webbit

## views

**app/views/submissions/unsubscribe.html.erb**

- Created a view file with the name successfully_unsubscribed.html.erb.
- Added a div element with classes for styling purposes.
- Added a h2 element with the text "Successfully unsubscribed" using the text-gray-800 color.
- Added a link to the homepage of the Webbit application using the link_to helper method.
- Styled the link using the btn and btn-purple classes to make it a purple button.
- Added an image_tag helper method to display an image named "unsubscribe-image.svg".
- Set the width of the image to 64 pixels using the class attribute.
- Wrapped the link and image elements in a flex container using the flex and items-center classes.
- Added the justify-between class to create space between the flex container elements.

**app/views/submissions/upvote.js.erb**

- Included a partial view called "vote" using the render method.
- Passed any necessary local variables to the partial.
- Rendered the partial in the current view, allowing it to be reused in different parts of the application.
- Implemented the necessary functionality in the "vote" partial, such as displaying the vote count, allowing users to vote, and updating the vote count after a vote is cast.

**app/views/subscriptions/_subscribed.js.erb**

- Retrieve the HTML element with the ID "subscribed_block" and store it in a variable called "subscribedBlock".
- Retrieve the HTML element with the ID "subscribed_count" and store it in a variable called "subscribedCount".
- Set the inner HTML of "subscribedBlock" to an empty string.
- Set the inner HTML of "subscribedCount" to an empty string.
- Insert HTML code at the end of "subscribedBlock" using the "insertAdjacentHTML" method, which renders a partial view called "subscription" from the "subscriptions" folder of the "views" folder. The "community" variable is passed as a parameter to the partial view.
- Insert HTML code at the end of "subscribedCount" using the "insertAdjacentHTML" method, which displays the number of subscriptions for the current community as a string.
- Overall, this code updates the subscription count and block in the HTML page when a user subscribes to a community.

**app/views/subscriptions/_subscription.html.erb**

- Check if the user is signed in using if user_signed_in?
- If the user is signed in, check if the current user is already subscribed to the community using unless current_user.communities.include? @community
- If the current user is not subscribed to the community, display a "Subscribe" link using link_to "Subscribe", community_subscriptions_path(@community), method: :post, remote: true, class: "btn btn-purple btn-small w-100 block text-center"
- If the current user is already subscribed to the community, display an "Unsubscribe" link using link_to "Unsubscribe", community_subscriptions_path(@community), method: :delete, remote: true, class: "btn btn-outline btn-small w-100 block text-center"
- If the user is not signed in, display a "Subscribe" link that redirects to the sign-in page using link_to "Subscribe", new_user_session_path, class: "btn btn-purple btn-small w-100 block text-center"

# Webbit

## views

**app/views/subscriptions/create.js.erb**

- Render the "subscribed" partial view in the current view using the <%= render %> method.
- This will include the code from the "subscribed" partial view in the current view, allowing the user to see the content of both views on the same page.
- The render method is a built-in Rails method that enables partial views to be embedded into other views, simplifying view code and making it more modular.
- The "subscribed" partial view is likely a reusable component that displays information about a user's subscription status or preferences, reducing code duplication and promoting code reuse.

**app/views/subscriptions/destroy.js.erb**

- Render the partial view named "subscribed".
- Include the HTML code and Ruby logic defined in the "subscribed" partial view in the current view.
- Pass any necessary variables and objects to the "subscribed" partial view for it to properly render its content.
- Display the rendered HTML code from the "subscribed" partial view in the current view.

**app/views/devise/registrations/edit.html.erb**

- Render a shared form wrap partial in the content_for block with the name "devise_form".
- Generate a heading with the text "Edit" followed by the resource name in humanized form.
- Create a form for the resource with method "put" and the URL set to the registration path of the resource.
- Render the shared error messages partial with the resource passed in as an argument.
- Create a label and a text field for the resource's username.
- Create a label and an email field for the resource's email.
- If the resource is confirmable and pending reconfirmation, display a message indicating the unconfirmed email.
- Create a label and a password field for the resource's password.
- If a minimum password length is defined, display a message indicating the minimum length and leave the field blank if the user doesn't want to change the password.
- Create a label and a password confirmation field for the resource's password confirmation.
- Create a label and a current password field for the resource's current password.
- Display a message indicating the need for the current password to confirm the changes.
- Create a heading with the text "Subscription".
- The code not commented out is used to render a form for editing a user's profile information and password. It is likely part of a larger Rails application that uses the Devise gem for user authentication and subscription management.

# Webbit

## views

**app/views/devise/registrations/new.html.erb**

- Created a form for user registration using form_for helper method.
- Defined label and input fields for username, email, password, and password confirmation.
- Rendered error messages and links using "devise/shared/error_messages" and "devise/shared/links" partials.
- Set autofocus and autocomplete attributes for username and email fields.
- Displayed a password length requirement if @minimum_password_length is set.
- Used submit button to create a user account on the submit action.
- Wrapped the form using "devise/shared/form_wrap" partial.
- Assigned the form to a content_for block named :devise_form.

# Webbit

## views

**app/views/devise/shared/_error_messages.html.erb**

- Checked if there are any errors for a given resource using the if resource.errors.any? statement.
- If there are errors, displayed them on the page.
- Created a div element with an id of error_explanation, which has a red border and background color, and rounded corners.
- Added a h2 element inside the div element with a text in red color indicating the number of errors that occurred and the type of resource, using the I18n.t method to translate the error message.
- Created an unordered list ul inside the div element to display the errors, with each error message as a li element.
- Used the each method to iterate over all the error messages and display them in the ul list using the resource.errors.full_messages method.

**app/views/devise/shared/_form_wrap.html.erb**

- Created a Ruby on Rails view template that renders a form using the Devise gem.
- Enclosed the form inside a div element with classes for styling and responsive design.
- Used the "yield" keyword to allow for insertion of additional content within the div.
- Set the "yield :devise_form" block to render the Devise form.
- Configured the form with specific input fields for the user's email and password.
- Encapsulated the form within appropriate tags for accessibility and semantic markup.
- Utilized the Devise gem's built-in styling to provide a clean and professional look to the form.
- Implemented shadow and rounded corner effects on the form container to provide visual hierarchy.
- Employed inline styling with the "px-8", "pt-6", and "pb-8" classes to adjust padding on the form container for aesthetic purposes.

**app/views/devise/shared/_links.html.erb**

- Check if the current controller is not "sessions". If true, display a link to "Log in".
- Check if the devise mapping is registerable and the current controller is not "registrations". If true, display a link to "Sign up".
- Check if the devise mapping is recoverable and the current controller is not "passwords" or "registrations". If true, display a link to "Forgot your password?".
- Check if the devise mapping is confirmable and the current controller is not "confirmations". If true, display a link to "Didn't receive confirmation instructions?".
- Check if the devise mapping is lockable, the resource class has email unlock strategy enabled, and the current controller is not "unlocks". If true, display a link to "Didn't receive unlock instructions?".
- If the devise mapping is omniauthable, display links to sign in with each provider listed in resource_class.omniauth_providers. The links should use the method "post" and the path generated by omniauth_authorize_path.

# Webbit

**views**

**app/views/layouts/application.html.erb**

- Included the necessary meta tags and headers to display the web page correctly.
- Displayed a custom favicon on the web page.
- Created a navigation bar with the Webbit logo, a community dropdown, a search area, and sign-up/sign-in/sign-out links.
- Added conditional statements to display different alerts depending on the type of message.
- Added links to different pages based on the user's status.
- Created a form for searching content on the web page.
- Added a custom CSS file for styling the web page.
- Displayed user account information and links to different account settings pages.

**app/views/home/index.html.erb**

- Created a view file "index.html.erb" under "app/views/home/" directory.
- Used a container div with "mx-auto" class to center align the content.
- Added a h1 tag with "font-mono", "mb-4" and "text-red-500" classes to display a heading with specific font, margin, and color styles respectively.
- Added a p tag with "italic" and "text-purple-600" classes to display a paragraph with italic font style and purple color.
- Added another p tag with "italic" class and "hover:not-italic" pseudo-class to display a paragraph with italic font style which turns to normal when hovered over.
- Added an inline Ruby code with "<%= %>" tags to render an SVG image with "edit" icon and "fill-current text-purple-600" styles to set the fill color to purple and current text color.
- Used a helper method "render_svg" to generate the SVG image code which takes the SVG file name and styles as parameters.

**app/views/layouts/mailer.html.erb**

- Wrote an HTML email template for the project
- Used inline styles for email compatibility
- Included a yield statement for content to be added dynamically
- Used tables for layout and organization
- Added client-specific styles for compatibility with Outlook and Hotmail
- Used CSS to style content including font, color, and alignment
- Added dynamic content for the current year using Date.current.year
- Added a mailing address at the end of the email

# Webbit

**views**
**app/views/premium/new.html.erb**

- Added a JavaScript pack tag for "premium" in the head of the page
- Created a div with a class of "bg-white max-w-4xl rounded-lg border border-gray-400 p-6 m-auto mt-10 flex flex-wrap items-start justify-between"
- Created another div inside the previous div with a class of "lg:w-1/2 text-center"
- Used an image tag to display an illustration named "premium-illustration.svg" with a width of 400 and a class of "max-w-full"
- Created another div inside the parent div with a class of "lg:w-1/2 lg:px-10"
- Used a heading tag to display the text "Subscribe to Webbit Premium" with a class of "text-2xl pt-4 text-gray-900 leading-tighter mb-2"
- Used a paragraph tag to display the text "Go premium to use Webbit ad-free."
- Created a form tag with an action of "/premium" and an ID of "payment-form"
- Created a div with a class of "field mb-10 mt-6"
- Created a label with a "for" attribute of "card-element" and a class of "label mb-2", displaying the text "Enter credit or debit card"
- Created a div with an ID of "card-element", where a Stripe element will be inserted
- Created another div with an ID of "card-errors" and a class of "text-red-500 text-sm", to display any errors related to the credit/debit card
- Created a button with a class of "btn-green btn mt-6 block w-full" to subscribe for $6/month

# Webbit

**views**
**app/views/users/_profile_comment.html.erb**

- Created a div container with a class of "mb-4 shadow pl-6"
- Rendered an SVG icon of a conversation using the "render_svg" method and passed in the icon path
- Displayed the username of the user who commented by accessing the "username" attribute of the "@user" instance variable
- Linked to the submission title by accessing the "title" attribute of the "comment.submission" instance variable and passed in the submission link using the "link_to" method
- Linked to the community path by accessing the "comment.submission.community" instance variable and passed in the community name using the "link_to" method
- Rendered the "posted_by" partial by passing in the "comment" instance variable and displayed it in the container
- Displayed the reply text by accessing the "reply" attribute of the "comment" instance variable
- Created a "Reply" link using the "link_to" method that links to the submission path with an anchor of "comment-#{comment.id}" if the user is signed in

**app/views/submission_mailer/new_response.html.erb**

- Render a message in HTML format, which is being sent to the user with the username specified in @submission.user.username and contains the title of their submission in @submission.title.
- Display a new comment for the submission in the paragraph <p> element, which is retrieved from @comment.reply.
- Provide an "Unsubscribe" link for the user, which redirects them to comment_unsubscribe_url with the unsubscribe_hash parameter for their subscription.

# Webbit

**views**

**app/views/devise/sessions/new.html.erb**

- Created a login form for a web application using Ruby on Rails.
- Included the "devise_form" content in the web application using the "content_for" method.
- Used the "form_for" method to create a form object for the login form, specifying the "resource" variable as the object to use for the form.
- Specified the "email" and "password" fields for the login form using the "email_field" and "password_field" methods, respectively.
- Used the "rememberable?" method to check if the application is set up to remember the user, and displayed a checkbox for this purpose if it is.
- Displayed a "Log in" button to submit the form using the "submit" method.
- Included links to other login-related pages using the "render" method to display a partial view.
- Used the "render" method to display a form wrapper for the login form.

**app/views/users/show.html.erb**

- Created a navigation bar with two tabs, "Submissions" and "Comments"
- Added a link to each tab, with its respective ID, class and label
- Created a conditional to check if the user has any submissions or comments, and if so, render each one through a loop
- Created a section for the primary content, which includes the conditional and loops for submissions and comments
- Created a section for the secondary content, which includes the user's gravatar, username, and join date, as well as links to create a new submission, or login/sign up if not already signed in
- Rendered a basic layout that includes the primary and secondary content sections

# Webbit

## mailers

### app/mailers/submission_mailer.rb

- Here's a concise explanation of the code:
- Created a mailer class called SubmissionMailer that inherits from the ApplicationMailer class.
- Set the default 'from' email address for the mailer to be "notifications@webbit.com".
- Defined a method called 'new_response' that takes in 'params' hash as an argument.
- Initialized two instance variables, '@comment' and '@submission', by assigning the values of 'params[:comment]' and 'params[:submission]' respectively.
- If the user who submitted the @submission has enabled comment subscription, send an email to that user's email address with the subject "New response on #{@submission.title}".
- The email content and template can be defined in a separate view file, such as 'new_response.html.erb'.
- The code that is commented out is not executed, as it is not surrounded by any conditional statements.

## admin controllers

### app/controllers/admin/submissions_controller.rb

- Created a new Ruby on Rails controller named "Admin::SubmissionsController".
- Defined an "index" action within the "Admin::SubmissionsController" controller, which handles the HTTP GET request to the URL path "/admin/submissions".
- Initialized an instance variable named "@submissions" to store all the instances of the "Submission" model class retrieved from the database using the "all" method.
- Rendered the "index" view template by default, which corresponds to the "index" action name in the "Admin::SubmissionsController" controller.

### /app/controllers/admin/users_controller.rb

- Retrieved all the user records using the User.all method.
- Sorted the user records using the sort method.
- Assigned the sorted user records to an instance variable @users.
- Rendered the index view which will display the list of sorted users.