CS 4990: Ruby on Rails Web Application Development

Assignment: Callbacks

Requirements

- Add functionality to a Ruby on Rails application to display recent site activity via an activity feed and to welcome new users via email.

- Create an `Activity` model with a string attribute named `message` and a polymorphic `belongs_to` association that will be used to link back to the resource that triggered the activity.

- For each of at least two different resources, create an observer and define `after_create` and `after_update` callbacks that each create a new `Activity` associated to the resource that was created or updated.

- On a page within your application, such as your home page, display an activity feed which contains the 10 most recent activities. For each `Activity`, render the activity’s message as a link to the associated resource. Create and use a scope on the `Activity` model to sort activity records in reverse-chronological order, and to limit the results to 10 records.

- Create a mailer to send a welcome email to a user by creating a new mailer class named `UserMailer` with a method named `welcome_email` that sends an email to a user’s registered email address. Create both HTML and plain-text email templates for `welcome_email`, and within each, render a welcome message that includes the user’s first name and a link to your application’s `root_url`.

- Create an observer named `UserObserver` with an `after_create` callback that uses `UserMailer` to send a welcome email to each new user.

[Optional] In order for recipients to receive the email messages your application sends, configure your application to use Amazon’s Simple Email Service (SES) and deploy your application to Heroku.

Steps

Activity Feed

1. Generate an `Activity` model with a string attribute named `message`, as well as an integer attribute named `resource_id` and a string attribute named `resource_type` to accommodate a polymorphic `belongs_to` association named `resource`. Don’t forget to apply the generated database migration.

```ruby
$ bin/rails generate model activity message:string resource_id:integer resource_type:string
$ bin/rake db:migrate
```

2. Within your new `Activity` model at `app/models/activity.rb`, create a `belongs_to :resource` association and define it as a polymorphic association.

```ruby
belongs_to :resource, :polymorphic => true
```

3. Add the `rails-observers` gem to your project by adding it to your `Gemfile` and then running `bundle` in Terminal.

4. For each of the models through which you want to record activity, generate an observer using the name of the model. For instance, the Rails generate commands below create observers at `app/models/planet_observer.rb` and `app/models/moon_observer.rb`.

```ruby
$ bin/rails generate observer planet
$ bin/rails generate observer moon
```

5. Tell Rails to use these new observers by registering them. Within `config/application.rb`, add a configuration line that specifies a comma-separated list of the observers you created, similar to the following. Don’t forget to restart your Rails server after modifying configuration files like this one.

```ruby
config.active_record.observers = :planet_observer, :moon_observer
```
6. For each of your observers, define methods `after_create` and `after_update`. These callback methods each accept a single argument: an instance of the model they’re observing. Within each method, create a new `Activity` by specifying the given model instance as the activity’s resource, as well as a message that appropriately describes the activity.

```ruby
def after_create(planet)
    Activity.create(:resource => planet, :message => "A new planet named #{planet.name} was discovered")
end

def after_update(planet)
    Activity.create(:resource => planet, :message => "Updated data for planet #{planet.name} is available")
end
```

7. In order to display the most recent activities, define a scope for the `Activity` model within `app/models/activity.rb` that returns up to 10 records in reverse-chronological order.

```ruby
scope :most_recent, -> { order("created_at").reverse_order.limit(10) }
```

8. Within the controller action for the page on which you’d like to display the activity feed, add an instance variable that contains a collection of the most recent `Activity` records (using the scope you just created). For instance, to display the activity feed on your home page, modify the `index` action within `app/controllers/home_controller.rb`.

```ruby
def index
    @activities = Activity.most_recent
end
```

9. Finally, display the activity feed. Within the corresponding view template, use the instance variable you defined in the controller to render a list of activities. Render each activity’s message as a link to its associated resource.

```html
<div id="activity">
    <h3>Recent Discoveries:</h3>
    <ul>
    <%= @activities.each do |activity| %>
        <li><%= link_to activity.message, activity.resource %></li>
    <% end %>
    </ul>
</div>
```

---

**Welcome Email**

1. Start by generating a new mailer named `UserMailer`.

```bash
$ bin/rails generate mailer user_mailer
```

2. Open the generated mailer at `app/mailers/user_mailer.rb` and create a new method named `welcome_email` for sending a welcome email. The method should accept a single argument: an instance of a `User` model. Much like a controller action, the method should create one or more instance variables to contain any data needed by the corresponding view template, which in this case is just `@user`. Finally, call the special method `mail` and specify the recipient and subject for the email to be sent.

```ruby
def welcome_email(user)
    @user = user
    mail(:to => user.email, :subject => "Welcome to Planetarium!")
end
```

3. Create two view templates for the welcome email within the directory `app/views/user_mailer/`: one that contains an HTML-formatted email (`welcome_email.html.erb`), and one that contains a plain-text email (`welcome_email.text.erb`). These two templates will be used to create a single, multi-part email, so they should contain the same content. Within the templates, you may reference any instance variables defined within the corresponding mailer method, as well as call helper methods, including URL helpers. For
instance, the following plain-text template renders the user’s first name and a link to the application’s root_url.

Hi <%= @user.first_name %>,

Welcome to Planetarium!

Get started by visiting us at: <%= root_url %>

Have a great day!

4. In order for URLs to work correctly within email templates, you must first tell Rails what host to use when generating the URLs. To do so, add a line similar to the following to config/application.rb. If you’re deploying your application to a production environment like Heroku, specify the host for your production environment. Otherwise, use any host for testing purposes, such as example.com.

```ruby
config.action_mailer.default_url_options = { :host => "theverse.herokuapp.com" }
```

5. To test your new mailer, open the Rails console and enter something like the following. No email messages will be sent in your development environment, but you can see what would be sent by looking near the end of your development log file at log/development.log.

```
irb> user = User.first
irb> UserMailer.welcome_email(user).deliver
```

6. In order to send a welcome email to each user when they register, create a new observer named UserObserver.

```
$ bin/rails generate observer user
```

7. Don’t forget to register the new observer by adding it to the config.active_record.observers list within config/application.rb.

```ruby
config.active_record.observers = :planet_observer, :moon_observer, :user_observer
```

8. Within the observer, define a callback method after_create that accepts a single argument: an instance of the User model. Within the method, simply send a welcome email just like you did in the console.

```ruby
def after_create(user)
  UserMailer.welcome_email(user).deliver
end
```

[Optional] Amazon SES

1. After all that hard work, email messages sent by your application won’t be delivered without a bit more work. By configuring your application to use Amazon’s Simple Email Service (SES) in the production environment, your email messages will be delivered to their recipients. First, open the Amazon AWS Management Console, navigate to the SES page, and click SMTP Settings. On this page, click the “Create My SMTP Credentials” button and follow the instructions to create an SMTP username and password. You’ll use these credentials in the next step. Now click Email Addresses under Verified Senders and then click the “Verify a New Email Address” button. Follow the instructions to verify an email address that you own and have access to. This will be the email address from which your application will send email. Also be aware that all new Amazon SES accounts are initially limited to “sandbox” access, meaning you must also verify any email addresses to which your application will send email. While this is reasonable for testing purposes, it’s not practical for production usage. To remove this restriction, click Dashboard, click the “Request Production Access” button, and follow the instructions.

2. Add the following to your production environment configuration at config/environments/production.rb. Be sure to enter your SMTP username and password from the previous step.

```ruby
config.action_mailer.delivery_method = :smtp
```
config.action_mailer.smtp_settings = {
    :address => "email-smtp.us-east-1.amazonaws.com",
    :user_name => "YOUR_SMTP_USERNAME",
    :password => "YOUR_SMTP_PASSWORD",
    :authentication => "plain",
    :enable_starttls_auto => true
}

3. Within your mailer at app/mailers/user_mailer.rb, change the :from email address to the Verified Sender address you verified with Amazon SES earlier.

```ruby
default :from => "YOUR_VERIFIED_SENDER_EMAIL"
```

4. Finally, deploy your app to Heroku and email messages sent by your application will be delivered to their recipients by Amazon SES.

Resources

- Use the example from class as a reference when completing the steps above.
- Rails::Observers on GitHub
- Ruby on Rails Guides: Action Mailer Basics

Submission

- Show your completed assignment to the instructor during class or office hours to receive credit.