Assignments: Authorization

Requirements

- Starting with the previous assignment, **Authentication**, enhance the security features of your application by adding additional logic to control what actions users can and cannot perform within your application. Users will be authorized to perform actions based on 1) the presence or absence of an administrative role for each user, and 2) the resource ownership of each user (established by associations your application maintains).

- Establish the concept of an administrative role for users by adding a single boolean attribute to your existing User model. Rather than adding logic and views to your application for editing users, simply use the Rails Console to enable the admin role for at least one user. Admin users should be authorized to read, create, update, and destroy all resources, regardless of resource ownership.

- Non-admin users should be authorized to read and create all resources, but they should only be authorized to update and destroy the resources which they own. To establish the concept of resource ownership, create a one-to-many relationship between your User model and the models for your “owned” resources. For instance, a User has_many :planets, and a Planet belongs_to :user. Being that your users must be signed in to create a resource, simply establish this association when the resources are created (within the create actions of the various controllers).

- Apply the authorization constraints above to at least two different resources within your application.

- When a user attempts to perform an action for which they are not authorized, you should redirect the user to the application’s home page and display a message. Use the flash feature to do this.

- To improve the user experience of your application, don’t render links to actions for which users are not authorized. For instance, since non-admin users are not authorized to destroy resources owned by other users, do not display delete links for resources owned by other users. But, since admin users are always authorized to perform all actions, all links should be visible to admins.

- Minimize the amount of work required by using an open-source library. Specifically, use **CanCan**, an authorization library for Ruby on Rails created by Ryan Bates. Learn how to install, configure, and use CanCan by reading the README and other documentation published at the project’s GitHub repository.

Steps

1. Follow the first couple steps outlined in CanCan’s README to add CanCan to your Rails project. This involves adding a gem to the bottom of your Gemfile, running the bundle command to install the gem, and running a Rails generate command provided by CanCan to create an Ability class. When following the instructions in the README, use the steps for Rails 4.

   $ bin/rails generate migration add_admin_to_users admin:boolean
   $ bin/rake db:migrate

2. In order for users to become admin users, generate a database migration to add a new boolean attribute named admin to your User model. Don’t forget to apply the migration afterward. If you’re generating the User resource for the first time, feel free to add the attribute at that time. Be sure not to include :admin in the list of permitted parameters in the users controller, because you want to be absolutely certain that users are not able to grant themselves administrative capabilities (by maliciously injecting an additional form attribute, for instance).

   irb> user = User.find_by_email("dj@example.com")
   => #<User id: 1>
   irb> user.admin = true
   => true
   irb> user.save
   => true
4. Add logic to authorize admin users to perform any action to any resource, while restricting non-admin users to only reading and creating resources. To do this, add the following code to the top of the `initialize` method of the `Ability` class at `app/models/ability.rb`:

```ruby
if user.admin?
  can :manage, :all
else
  can [:read, :create], :all
end
```

5. Tell your resource controllers to enforce this authorization logic by adding `load_and_authorize_resource` immediately after `before_action :authenticate_user!` within each of your resource controllers. This tells CanCan to load and authorize the appropriate resource before each action within the controller. Because CanCan will now be loading your resources for you, take a moment to remove what should be the first line in each action within your resource controllers. For the Planet controller in the class example, this includes the following lines:

```ruby
@planets = Planet.all
@planet = Planet.new(planet_params)
@planet = Planet.find(params[:id])
```

6. Ensure that each of your resource controllers implement a private method that returns the permitted parameters for its resource. For example, for the planets controller, the method looks like this:

```ruby
def planet_params
end
```

7. Now, in order to authorize non-admin users to update and destroy the resources they own, you must first associate the resources to the user when the resources are created. To do this, first create a one-to-many relationship between your User model and the models for your “owned” resources. For each resource that will be owned by a user, add a `user_id` attribute to the model by generating and applying a database migration, add a `belongs_to :user` association to the model, and add the appropriate `has_many` association to the User model. Again, you may add the `user_id` attributes when you initially generate the resources. And be sure not to include `user_id` within the list of permitted parameters in your controllers. Otherwise, it would be possible for a user to maliciously assume ownership of another user’s resources.

8. Once the associations are in place, add logic to associate an individual resource to a user when the resource is created. This is relatively easy since your application already requires that users be signed in to create a resource. Simply add a line of code similar to the following to the top of the `create` action within each of your various resource controllers:

```ruby
@planet.user = current_user
```

9. Return to your `Ability` class and add one more line of code to authorize non-admin users to update and destroy the resources they own. Your updated code should look something like this:

```ruby
if user.admin?
  can :manage, :all
else
  can [:read, :create], :all
  can [:update, :destroy], :all, :user_id => user.id
end
```

10. Follow step 3 in the CanCan README to redirect to the home page and show a message when a user attempts to perform an unauthorized action.

11. Follow the first part of step 2 in the CanCan README to remove links to actions for which the signed-in user is not authorized.

**Resources**

- Use the example from class as a reference when completing the steps above.
For additional help installing, configuring, and using CanCan, refer to the README and wiki pages available at the project’s GitHub repository.

Submission

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