Lab 04: User Authentication & Authorization — Part II

Instructions: In this lab, we will look at how user authentication and authorization solutions can be developed in Rails using third-party plugins. "Devise" is a flexible authentication solution for Rails. After that, we will experiment with a nice authorization library "CanCan" for Ruby on Rails. CanCan is a basic framework for authorization that makes it easy to build roles and role-based authorization into your user model.

Credits

- Devise – `https://github.com/plataformatec/devise`
- CanCan – `https://github.com/ryanb/cancan`
- Thanks to Iqbal Farabi for the nice tutorial at `http://iqbalfarabi.net/`.

In this lab, we will build a project "Readit," a very simple version of "Reddit." If you don't know what Reddit is, see `http://www.reddit.com/`. But here we will implement only the features that allow registered users to submit content in the form of links. Later, we will allow moderators to edit links and administrators to edit and remove links.

Getting Started with Devise

We have already implemented an authentication module from scratch in the last lab session, but now we will look at a prominent third party solution "Devise," which is a flexible authentication solution for Ruby on Rails. It is a complete MVC solution based on the Rails engine. It has modular design, which means you only need to include the features you want. OK, let's get started:

1. Create a Rails project called `readit` using the PostgreSQL database.
2. Create a scaffold `Link` with name as string and URL as string. After that, add the following line to your `routes.rb`.

   ```
   root :to => 'links#index'
   ```

   Delete the file `public/index.html`. Check if your Web site is working.
3. Next, get Devise. Edit your `Gemfile`:

   ```
   gem 'devise'
   ```

   and then run `bundle install`.
4. Next we run the generator

   ```
   $ rails generate devise:install
   ```

   which installs an initializer for Devise. It also outputs a description of Devise's configuration options. You MUST take a look at it.

5. When you are done, you are ready to add Devise to any of your models using the generator. Let's create a User model using Devise, and you will see some magic. Just run:

   ```
   $ rails generate devise User
   ```

   This will create a `User` model and automatically configure it with Devise modules.

6. Edit `user.rb`. It should look like this:

   ```
   # Include default devise modules. Others available are:
   # :token_authenticatable, :confirmable,
   # :lockable, :timeoutable and :omniauthable
   devise :database_authenticatable, :registerable,
          :recoverable, :rememberable, :trackable, :validatable

   # Setup accessible (or protected) attributes for your model
   attr_accessible :email, :password, :password_confirmation, :remember_me
   # attr_accessible :title, :body
   ```

7. Run `rake routes` to check the routes that are automatically generated by Devise. See how the Devise routes are set up in `routes.rb` too.

8. Look at the database migration file, try to understand what it does, and run `rake db:migrate`.

9. Restart the server. Magically, you should now be able to sign up and sign in! But you cannot sign out. Why? **Hint:** Check the routes and think about the HTTP methods (See more information here: `http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html`).

10. Add the following code to the body in the `app/views/layouts/application.html.erb` file to provide user navigation links.

    ```
    <p class="notice"><%= notice %></p>
    <p class="alert"><%= alert %></p>
    <% if flash[:error] %>
      <div id="error">
        <%= flash[:error] %>
      </div>
    <% end %>

    <div id="container">
      <div id="user_status">
        <% if user_signed_in? %>
          Welcome <%= current_user.email %>! Not you?
         <%= link_to "Sign out", destroy_user_session_path, :method => :delete %>
        <% else %>
          <%= link_to "Sign up", new_user_registration_path %> or
          <%= link_to "Sign in", new_user_session_path %>.
        <% end %>
      </div>

      <%= yield %>
    </div>
    ```

11. Congratulations! You can now sign out, but you will notice that when you sign out, by default, Devise will automatically redirect you to the root page.

12. You may add the following line to any of your controllers such as the Link controller, so that this controller can only be accessed by authenticated users.

    ```
    before_filter :authenticate_user!
    ```

13. Since we are building an online link collection, we probably want to show our content to the world, right? Let's allow anyone to see our content by adding the following line to the Link controller.

    ```
    before_filter :authenticate_user!, :except => [:index, :show]
    ```

14. Verify that your Auth+Auth is working as expected.

Customizing Devise

Let's play around with Devise a litte bit. Notice that you use your email for authentication. Devise uses it by default when authenticating a user, but what if you want to use your username instead?

Fortunately, it is easy. What you need to do is to add a `username` column to the `User` table, change some of Devise's configuration, and edit the view. OK, let's make it happen.

1. To add a new column to your table, you can use a migration. Run:

   ```
   $ rails generate migration add_username_to_users username:string
   ```

   Then run `rake db:migrate`.

2. Go to the rails console and update your user information. You can use the class method `update_attribute()` here.

3. To change the Devise configuration, open the file `config/initializer/devise.rb`. After that, change `:email` in the following lines:

   ```
   config.authentication_keys = [ :email ]
   config.case_insensitive_keys = [ :email ]
   config.strip_whitespace_keys = [ :email ]
   ```

   to be `:username`. You will need to restart the server because you have changed the configuration.

4. You have to generate the Devise views, so that you can edit the forms and add the username field to them. To generate the Devise views, run:

   ```
   $ rails generate devise:views
   ```

5. Let's edit the sign in form. Change the lines:

   ```
   <div><%= f.label :email %><br />
   <%= f.email_field :email %></div>
   ```

   to

   ```
   <div><%= f.label :username %><br />
   <%= f.text_field :username %></div>
   ```

   That's it! Check if it works.

Finally, you are recommended to read the Devise documentation. You can add more features such as email confirmation or password resetting to your Web site. Furthermore, you can integrate authentication using OpenID, Twitter, Google, Yahoo, and Facebook.

There are many third-party authentication plugins available for Ruby on Rails. You are recommended to play with those as well. :-)

Role-based Authorization using CanCan

Next we are going to use a nice authorizaton gem called "CanCan." It allows us to, among other possibilities, add role-based authorization to our `User` model.

1. Add your `Gemfile` the line:

   ```
   gem 'cancan'
   ```

   Run `bundle install`.

2. User permissions are defined in an Ability class, so let's generate one.

   ```
   $ rails generate cancan:ability
   ```

3. Go to the Ability class. Add the following line in the method `initialize`.

   ```
   can :read, :all
   ```

   This means that anyone can only read on all models.

4. Let's edit the index view for `LinkController`.

   ```
   <% if can? :update, Link %>
     <td><%= link_to 'Edit', edit_link_path(link) %></td>
   <% else %>
     <td>Edit</td>
   <% end %>
   <% if can? :destroy, Link %>
     <td><%= link_to 'Destroy', link, method: :delete, data: { confirm: 'Are you sure?' } %></td>
   <% else %>
     <td>Destroy</td>
   <% end %>
   ```

   Restart the server then check if it works. Notice however that even though we cannot click on the link, we can still access it from the correct URL, for example, `http://localhost:3000/links/1/edit`.

5. To prevent this, add the following line

   ```
   authorize! :update, @link
   ```

   to the action `edit` in `LinkController`. Refresh the page and see the results. You should see an error page. It is always better to catch exception and generate a user-friendly error page. In the Application controller, add

   ```
   rescue_from CanCan::AccessDenied do |exception|
     flash[:error] = "Access denied."
     redirect_to root_url
   end
   ```

   However, we need to specify an authorized access for every controller action in our application. This is quite tedious to do. Instead of doing that, we will use a nice method called `load_and_authorize_resource` in the controller.

   Once this is done, we do not need to call the method `authenticate_user!` from Devise as we did in the first section anymore, so remove it. Restart the server and check the results.

   Note also that you don't have to set the resource's instance variable in each action, since `load_and_authorize_resource` will properly detect the type of action and fetch it for you. Cool!

6. OK, it is time for role management! Let's first create a scaffold `Role` with string field `name`.
7. Next, manually create the migration file for the HABTM table. Run:

   ```
   $ rails generate migration UsersHaveAndBelongToManyRoles
   ```

   then go to the migration file and edit it as follows:

   ```
   class UsersHaveAndBelongsToManyRoles < ActiveRecord::Migration
     def self.up
       create_table :roles_users, :id => false do |t|
         t.references :role, :user
       end
     end

     def self.down
       drop_table :roles_users
     end
   end
   ```

8. Run `rake db:migrate`.
9. Modify the model files too. The Role model should look like this:

   ```
   class Role < ActiveRecord::Base
     has_and_belongs_to_many :users
   end
   ```

   Similarly, for the User model, add `has_and_belongs_to_many :roles` to the model, add `:role_ids` to `attr_accessible`, and add a method to identify the role of the corresponding user.

   ```
   def role?(role)
    return !!self.roles.find_by_name(role.to_s)
   end
   ```

10. Add some roles. Maybe "admin" and "moderator."
11. Add the following code to the sign up form:

    ```
    <% for role in Role.find(:all) %>
      <div>
        <%= check_box_tag "user[role_ids][]", role.id, @user.roles.include?(role) %>
        <%= role.name %>
      </div>
    <% end %>
    ```

12. OK, you may need to restart the server. Once finished, try adding one admin user.
13. Let's move back to the Ability class and modify as follows:

    ```
    user ||= User.new # guest user

    if user.role? :admin
      can :manage, :all
    elsif user.role? :moderator
      can [:read, :update], Link
    else
      can :read, Link
    end
    ```

Restart the server, check the results, and try adding a moderator user. Finished? It works like a charm, doesn't it? :-)

Congratulations! That's it for today. Enjoy the problem set 2 and hope to see your nice auth+auth features in your project. :-)