Assignment: Module #4 - Secure Information

Objective

By the end of this assignment, you will be able to:

- Design and implement API access to a database-backed resource on a Rails server
- Define and enforce an authorization policy for API access the resource
- Design and implement user access to the API resource through an AngularJS (1.x) web client
- Enforce an authorization policy within the web client that is consistent with the authorization policy enforced by the API
- Build on the skills from modules 1-3.
  - Configure a Rails application for use with a database
  - Provision resources for deployment
  - Deploy an API server and web client to the Internet (taking care of dependencies)
  - Require HTTPS for communication with web client and server API
  - Deploy a packaging of the SPA optimized for production

This assignment has many options

- All assignment options are intended to extend the course example developed in the lectures. It is not intended that you start from scratch. By extending the course example covered in the lectures, you will not only have a decent starting point – you will also have a base that is familiar to the peer grader to be able to more easily follow the details of your solution. You may augment or modify the course examples as needed.
- All assignment options have a minimal set of requirements that could easily be far short of what a full capability would contain. You are to limit your implementations to only what is required to provide a single list, show, and modify set of scenarios for users with different assigned roles.
- All assignment options are notional. You may use your imagination to fill in the details of the technical solution that will meet the overall goals of the assignment.
- All assignment options have security requirements with the need for users, credentials, and roles. Some must be pre-assigned by the deployer. Others may be added during runtime use.

General user account/role rules:

- Mike and Carol are application-wide administrators
- Alice is your first choice as an originator that remains an organizer. Carol is an alternate choice for this type of user.
- Greg and Marsha are your first choices for organizers that are not originators
- Jan, Cindy, Peter, and Bobby are candidates for pre-assigned members.
- Sam should not have any special roles. Login is the best he can do.

Requirements (310 min)

1. (0min – assumes you have been following along with lectures) You are required to start with the `security-assignment` tag from the course example and make you changes from there. If you are not using Asset Pipeline, then you may copy the course examples from the Asset Pipeline area into your development area of choice and work from that point (from a Git perspective, it is just a movement of files).

```
$ git clone https://github.com/jhu-ep-coursera/capstone_demoapp.git security-assignment
$ cd security-assignment
$ git checkout security-assignment
You are in 'detached HEAD' state ...
$ git checkout -b getting-started
Switched to a new branch 'getting-started'
```

```
$ git remote add staging https://git.heroku.com/abc-123.git #your staging repository
$ git push --force staging getting-started:master
$ heroku run rake db:migrate --remote staging
$ heroku run rake ptourist:reset_all --remote staging
$ heroku restart --remote staging
```

2. (15min – assumes you may need to craft a custom rakefile) Your solution must use the same "Brady Bunch" users and credentials setup for the course example application. If you need the peer grader to login as a specific user with assigned roles – state the name they are to login as and they will use the standard credentials from the ptourist rakefile.
3. (15min) Pick an assignment option. You do not have to implement the entire capability, but what you implement should be within the scope of the option.
4. (60min) Design and implement API access to new database-backed resource(s) on a Rails server. The backend can be either ActiveRecord or Mongoid, but an open choice is not always feasible in all cases.
5. (30min) Design and implement authorization policy enforcement to the new resource(s). Your policy must include the rules for index, show, create, update, and destroy of the resource.
   - The policy must be expressed (i.e., evaluatable by a peer grader with access to source code) in a Pundit policy class and not explicitly implemented within the controller.
6. (120 min) Design and implement a limited set of functionality within an AngularJS (1.x) web client that accesses your new resource(s) and demonstrates the ability to do the following.
   - The ability to list instances of one of your new resource(s)
   - The ability to view an instance of your new resource(s)
   - The ability to modify an instance of your new resource(s). This can include delete, and update the core object, or the modification of a relationship.
7. (60min) Enforce the resource authorization policy within the web client.
   - Information displayed to users must be appropriate for their assigned roles
   - Controls available for users to select or click must be appropriate for their assigned roles
8. (10min) Deploy the solution to the Internet (e.g., Heroku, Github pages) so that it is accessible to peer graders to evaluate.
   - Require HTTPS for communication with web client and server API
   - Deploy a packaging of the SPA optimized for production

Optional Implement and end-to-end Capybara/feature spec that tests/mimics the actions you want your peer grader to follow. You can create more spec files, but this should be one targeted at honing in one what they minimally need to do when evaluating your solution.

- the ability to list resource instances of your new resource(s)
- the ability to view a resource instance of your new resource(s)
- the ability to modify an instance of your new resource(s)

Options

You are to chose *one* of the following options to implement in this assignment. You do not need to implement the entire capability to receive full credit. Just authorized access to a single index, show, and modify action is the intent here. Anything else should be added outside of the scope of the assignment.

Use your imagination. There is nothing firm that any of these options require except that the solution must include a flavor of access control based on authentication, application roles, and and instance roles.

Option: Types of Things

Implement a means to identify and search for a type of Thing. You may design the types to be static (e.g., a museum is always a museum), or use dynamic tags (e.g., Things having easy wheelchair access)

Example Solution Choices: * List * Only authenticated users can constrain lists to Things having certain tags * View * Only authenticated users can see tags assigned to Thing * Modify * Only Thing organizers can assign tags

Option: Business and Service Offering

Implement a means for a Business (can be considered a Thing – your Things become businesses) to have one or more Service Offerings.

Example Solution Choices: * List * Only authenticated users can access the list of Service Offerings * View * Only members of the Business with the Service Offering can see non-public fields tracked within the Service Offering * Modify * Only Business organizers and create Service Offerings associated with their Business

Option: Customer and Inquiries

Implement a means for a customer of the site to make an inquiry. Only inquiries made by authenticated users can be retrieved by their originator. Your implementation does not have to include the processing of the inquiry – please concentrate on submitting inquiries and obtaining list of inquiries the authenticated user has initiated.

Example Solution Choices: * List * An organizer of an inquiry can list their inquiries * View * An organizer of an inquiry can view their inquiry. The same URL given to an unauthorized user will not display the inquery. * Modify * Only authenticated users can submit an Inquiry (and thus become the organizer for)

Option: Site Member and Inquiries

Implement a means for Thing organizers to access inqueries made on the site. Your implementation does not have to include the creating of the inquiry – please focus on the business user accessing the inqueries.

Example Solution Choices: * List * Only Business/Thing members can list all inquiries (i.e., Mike and Sam cannot see inquiries) * View * Only Business/Thing members can view an inquery * Modify * Only Business/Thing organizers can edit an inquiry

Option: Trip Logs

Implement a means for a Tour Guide to create a Trip (e.g., a special type of Thing) and allow members of that trip to post Images associated with the trip.

Example Solution Choices: * List * Only authenticated users can see the list of trips * View * Only the Trip member can view non-public fields of a trip * Modify * Only members of a Tour Guide Thing can create Trips

Option: Role Editor

Implement a means to manage roles for Things. The course example focuses on automatic organizer roles for creators and the enforcement of roles. However, it does not provide for the runtime assignment of roles like Thing member.

Example Solution Choices: * List * Members can list the complete member list of their Thing * Admin role can see a list of all user accounts * View * Admin role can see what roles a user has * Modify * Admin role can assign originator roles to resource types (e.g., Thing) * Organizers can modify member assignments to their Thing * Organizers can modify organizer assignments to their Thing

Rubric

1. Identify which assignment option you implemented and describe what the peer grader will be able to do with the enhancement relative to the rubric requirements. Feel free to boast but focus on what the peer grader needs to know first.
    - (5) Which assignment option did they pick?
2. Supply the base, public URL of your web client that hosts your solution for this assignment.
    - (5) When you view page source, is the CSS and Javascript minimized (i.e., does it have unnecessary comments and new line formatting removed) and concatenated (i.e., reduced number of URLs required for the browser to download)?

3. Supply the URL of a branch or tag in your public Git repository that represents the version of source code deployed for the Rails portion of your solution. This may be the same URL as the SPA version if you are using a single source tree.
   - (5) Is the Rails controller providing access to the resource under test delegating access decisions to a separate policy class?
   - (10) Is the policy class factoring in the user roles and specific resource instance accessed to make its decisions?
4. Supply the URL of a branch or tag in your public Git repository that represents the version of source code deployed for the SPA portion of your solution. This may be the same URL as the Rails version if you are using a single source tree.
   - (15) Is the policy being enforced based on application and resource instance roles obtained from the server? i.e., Authorizations are not soley being based on whether the user is authenticated.
5. Describe a single scenario a user can execute thru the UI that will demonstrate a query of the back end and the resulting collection will be a function of the user's login. (In the course example, Images were annotated when the user was their owner and Things were limited to only users with membership to that thing.)
   - (20) Were you able to become the prescribed user and see different results listed based on the login and the explanation?
6. Describe a single scenario where the content of what is displayed is different based on the user's login. (In the course example, only members of a Thing could see its notes field).
   - (20) Were you able to become the prescribed authentication state and see different results displayed based on the login and explanation?
7. Describe a single scenario where a user can make a change to the database based on being authenticated as a user with a certain assigned role.
   - (10) Were you able to become the prescribed authenticated state and make the specified change?
   - (10) Were you prevented from performing the change when being in an alternate user state?

Last Updated: 2017-02-07