Assignment for Module #4: Nested Resources, Security, and Pagination

The overall goal of this assignment is to assess your ability to implement:

- Nested resources
- Authenication
- Authorization
- Pagination

The functional goal of this assignment to implement a web application to manage TodoItems.

Functional Requirements

1. Start with the application completed in module 2. This will have defined all the models and relationships required for this assignment:

   - User
   - TodoList
   - TodoItem

   An Entity Relationship (ER) diagram is provided below to help depict each Model's relationship:

   ```
   +------+ 1    * +----------+ 1    * +----------+
   | User |--------| TodoList |--------| TodoItem |
   +------+        +----------+        +----------+
   ```

   We will not be using the optional `Profile` model class in this assignment.

2. Implement access to `TodoItem` as a nested resource of `TodoList`

3. Lock down the application to only authenticated users.

4. Limit access to resources associated with the logged in user.

5. Access unbounded collections of resources using pagination.

Getting Started

1. Start with a copy of your `todolists` solution from the module 2 assignment

2. From your `todolists` application root directory, remove the unit test from module 2.

   ```
   `-- spec
       `-- assignment_spec.rb
   ```

   ```
   $ rm spec/assignment_spec.rb
   ```

3. Download and extract the starter set of boostrap files for this assignment.

   ```
   |-- Gemfile
   |-- db
   |    `-- seeds.rb
   `-- spec
        -- start.rb
        -- nested_resources.rb
        -- security.rb
        -- authorization.rb
        -- authentication.rb
        -- pagination.rb
        -- end2end.rb
   ```

- Overwrite your existing Gemfile with the Gemfile from the bootstrap fileset. They should be nearly identical, but this is done to make sure the gems and versions you use in your solution can be processed by the automated Grader when you submit. Any submission should be tested with this version of the file.

  NOTE the Gemfile includes a section added for the last assignment:

  ```
  group :test do
      gem 'rspec-rails', '~> 3.0'
      gem 'capybara'
  end
  ```

  as well as a new definition for the following items:

  - `bcrypt` gem uncommented for use with `has_secure_password`
  - `tzinfo-data` gem conditionally included on Windows platforms
  - `will_paginate` added for implementing paging

    ```
    # Use ActiveModel has_secure_password
    gem 'bcrypt', '~> 3.1.7'

    # Windows does not include zoneinfo files, so bundle the tzinfo-data gem
    gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]

    gem 'will_paginate', '~> 3.0.6'
    ```

- Overwrite your existing `db/seeds.rb` file from the bootstrap fileset. This file contains some test data that will be useful during development and unit tests.
- Add the `spec/*.rb` files provided with the bootstrap fileset to the corresponding `spec/` directory within your `todolists` application. These files contain tests that will help determine whether you have completed the assignment.

4. Run the `bundle` command to make sure all gems are available.

   ```
   $ bundle
   ```

5. Run the rspec test(s) to receive feedback. `rspec` must be run from the root directory of your application. There are several test files provided for this assignment. Many of those files are designed to test your code at specific points as you proceed through the technical requirements of this assignment. As such, many tests will fail if executed after additional technical requirements have been completed. Initially, majority of tests will (obviously) fail until you complete the requirements necessary for them to pass.

   ```
   $ rspec
   ...
   (N) examples, 1 failure, (N) pending
   ```

   To focus test feedback on a specific step of the requirements, add the specific file (path included) with the tests along with "-e rq##" to the rspec command line to only evaluate a specific requirement. Pad all step numbers to two digits.

   ```
   $ rspec spec/start_spec.rb -e rq1.0
   ...
   (N) example, 0 failures
   ```

6. Implement your solution to the technical requirements and use the rspec tests to help verify your completed solution.

7. Submit your Rails app solution for grading.

Technical Requirements

1. Starting with a copy of your module 2 solution, this solution should already have `User`, `TodoList`, and `TodoItem` models defined with the following properties and relationships. This assignment does not use the `Profile` model class but it will not hurt to include it.

   - User
     - username - a string to hold account identity
     - password_digest - a string to hold password information
     - todo_lists - a 1:many relationship with `TodoList` (i.e., User `has_many` todo_lists).

     Add appropriate options to have the `User` model class delete a `TodoList` in a cascading fashion
   - TodoList
     - list_name - a string name assigned to the list
     - list_due_date - a date when todo items in the list are to be complete. This is a date. We are not concerned with the time of day.
     - user - a many:1 relationship with `User` (i.e., TodoList `belongs_to` User)
     - todo_items - a 1:many relationship with `TodoItem` (i.e., TodoList `has_many` todo_items).

     Add appropriate options to have the `TodoList` model class delete a `TodoItem` in a cascading fashion
   - TodoItem
     - due_date - date when the specific task is to be complete
     - title - a string with short name for specific task
     - description - a string with narrative text for specific task
     - completed - a boolean value (default=false), indicating whether item is complete
     - todo_list - a many:1 relationship with TodoList - TodoItem belongs_to TodoList

   ```
   $ rake db:migrate
   $ rspec spec/start_spec.rb
   ```

2. Add `has_secure_password` to the `User` model class. This will define a `password` property that will get processed into an encrypted hash stored in the `password_digest` database column. We won't use this capability immediately – but it is necessary to define it early in the assignment so that the data model works with the `db/seeds.rb` file in the next step.

   ```
   $ rspec spec/security_spec.rb -e rq02
   ```

3. Seed the database with the `db/seeds.rb` file. This will load sample `Users`, `TodoLists`, and `TodoItems`. If this loads correctly – your models and database are setup correctly and you are ready to start accessing the data through web pages produced by the controller and views.

   ```
   $ rake db:seed
   $ rails c
   > User.first.todo_lists.count
    => (N>0)
   ```

4. Use the `rails g scaffold_controller` command to create controller and view artifacts for `TodoLists` and `TodoItems`.

   ```
   $ rails g scaffold_controller TodoList list_name list_due_date:date
   $ rails g scaffold_controller TodoItem title due_date:date description:text completed:boolean
   ```

   Update `config/routes.rb` to

   - Access `:todo_list` resources at URI `/todo_lists`
   - Access `:todo_item` resources at URI `/todo_lists/:todo_list_id/todo_items`
   - Make the `todo_list#index` action the root of the application

(Hint: refer to module 4, lesson 1, lecture:`Nested Resources: Part1` for details on how this is done)

At this point, `TodoList` is defined as a global resource (with a root-level URI) and `TodoItem` is defined as a nested resource, always scoped below the `TodoList` it belongs to. Our application is not written to work that way, so expect some errors as we begin the modifications.

If you have not yet done so – please start the server and also take a look at your defined URI routes.

```
$ rails s   #in separate console
```

```
$ rake routes   #in original console
$ rspec spec/nested_resources_spec.rb -e rq04
```

5. Update the `TodoList` to display `TodoItems` as a nested resource in the `todo_list#show` page (`todo_lists/show.html.erb`).

   a. Copy the table from the `todo_items#index` page (`index.html.erb`) and paste the table into the `todo_lists#show` page (`todo_lists/show.html.erb`)
   b. Change global `@todo_items` references to scoped `@todo_list.todo_items` references below:

```
from: <% @todo_items.each do |todo_item| %>
to:   <% @todo_list.todo_items.each do |todo_item| %>
```

   c. Remove the `Edit` link for `TodoItems`
   d. Change the link_to parameters from global `todo_item` references to provide fully qualified `[@todo_list, todo_item]` references as an array.

```
from: <td><%= link_to 'Show', todo_item %></td>
to:   <td><%= link_to 'Show', [@todo_list, todo_item] %></td>

from: <td><%= link_to 'Destroy', todo_item, method: :delete, data: ...
to:   <td><%= link_to 'Destroy', [@todo_list, todo_item], method: :delete, data: ...
```

```
$ rspec spec/nested_resources_spec.rb -e rq05b
```

NOTE: This test case is for incremental testing only and WILL FAIL after authentication infrastructure is in place later in this assignment.

```
$ rspec spec/nested_resources_spec.rb -e rq05d
```

NOTE: This test case is for incremental testing only and WILL FAIL after authentication infrastructure is in place later in this assignment.

   e. Add a link to create a 'New Todo Item'. (hint: Use the `link_to` and `new_todo_list_todo_item_path(@todo_list)` helpers to produce a link tag)

```
$ rspec spec/nested_resources_spec.rb -e rq05e
```

Note how the `new_todo_list_todo_item_path(@todo_list)` is formed from what is produced in `rake routes`.

```
$ rake routes
new_todo_list_todo_item GET /todo_lists/:todo_list_id/todo_items/new(.:format) todo_items#new
```

- we want to invoke `todo_items#new` when we create a new `TodoItem`
- that action is mapped to using the `/todo_lists/:todo_list_id/todo_items/new(.:format)` URI and GET method. We are required to supply a :todo_list_id
- `new_todo_list_todo_item_path` is formed by adding `_path` to `new_todo_list_todo_item`
- the :todo_list_id is filled in by passing in a `@todo_list` when calling it
- GET is provided by `new_todo_list_todo_item_path`

Notice that the TodoItems now display on the todo_list#show page by navigating to a specific TodoList. However, the TodoItem URIs are not yet implemented in the TodoItem controller (next step).

```
$ rspec spec/nested_resources_spec.rb -e rq05
```

6. Modify the TodoItem controller to work as a nested resource by implementing the following. Note that your views with TodoItem URI references will not work until these changes are made and the links and forms are updated to include the scoping TodoList for each referenced TodoItem. The unit tests, however, will be able to make calls into your back-end to determine all URIs are implemented properly – prior to moving on to the views.

   a. Remove the old URI comments or replace them to have the following form since all calls to a TodoItem will now be scoped below a TodoList. Use the todo_item output of `rake routes` to give you a head start.

```
$ rake routes
#METHOD /todo_list/:todo_list_id/todo_items
#METHOD /todo_list/:todo_list_id/todo_items/:id
```

   b. Remove the todo_item#index method and views/todo_items/index pages. This will no longer be called since all TodoItem displays will be scoped to a particular TodoList. We will get the TodoList and call todo_list.todo_items() instead.

   c. Add a private helper method called set_todo_list that sets the @todo_list instance variable from the :todo_list_id property passed in via the params. (Hint: try the following in the `rails console` if you need practice locating a TodoList by id)

```
$ rails c
> list_id=TodoList.first.id
> @todo_list=TodoList.find(list_id)
```

   d. Update the private helper method called set_todo_item to scope its find command to the todo_items of a specific @todo_list list. (Hint: try the following in the `rails console` if you need practice locating a TodoItem by id scoped to a TodoList)

```
$ rails c
> list_id=TodoList.first.id
> @todo_list=TodoList.find(list_id)
> item_id=@todo_list.todo_items.first.id
> @todo_item=@todo_list.todo_items.find(item_id)
```

   e. Invoke the set_todo_list method before each method in the controller is executed using a before_action

   f. Update the todo_item#new action to return a new TodoItem instance that is initialized to reference its parent @todo_list, which is provided by set_todo_list.
   (Hint: try the following the `rails console` if you need practice creating a new instance of a TodoItem associated with a TodoList. Notice the new TodoItem is never saved to the database during this call. However, what is passed back to the form is a TodoItem prototype that has its foreign key reference set to the TodoList so that TodoList can be referenced when the TodoItem is finally created in a follow-on POST)

```
$ rails c
> @todo_list=TodoList.first
> @todo_item=@todo_list.todo_items.new
```

   g. Update the todo_item#create to create a new TodoItem instance based on the todo_item_params as before. Except now create this instance associated with the @todo_list provided by set_todo_list. (Hint: try the following in the `rails console` if you need practice creating a new instance of a TodoItem associated with a TodoList. Notice that in this case – save() is being called on the todo_list, causing the new TodoItem to be inserted into the database.)

```
$ rails c
> @todo_list=TodoList.first
> @todo_item=@todo_list.todo_items.new(title:"my item")
> @todo_list.save()
```

h. Update the HTML redirect of the `todo_item#create`, `todo_item#update`, and `todo_item#destroy` methods to do to the `todo_list#index` page. (Hint: use the `@todo_list` variable within `redirect_to` to express the `todo_list#index` page URI)

7. Update `TodoList` and `TodoItem` views to adjust the links and forms in these views to work with the updated URIs and `TodoItem` controller.

   a. Update the links on the `todo_items#show` page (`todo_items/show.html.erb`) to include the `TodoList` the `TodoItem` is a member of.

   - Change the `Edit` link_to path parameter from the global `edit_todo_item_path` (that no longer exists) to the new `edit_todo_list_todo_item_path`. This new method requires both `@todo_list` and 'todo_item' passed in as separate arguments (not as an array – as in previous requirement).

     ```
     from: <%= link_to 'Edit', edit_todo_item_path(@todo_item) %> |
     to:   <%= link_to 'Edit', edit_todo_list_todo_item_path(@todo_list, @todo_item) %> |
     ```

   - Change the `Back` link_to path parameter from to global `edit_items_path` (that no longer exists) to the `todo_listi#show` page it is a member of. This requires using the `@todo_list`.

     ```
     from: <%= link_to 'Back', todo_items_path %>
     to:   <%= link_to 'Back', @todo_list %>
     ```

     ```
     $ rspec spec/nested_resources_spec.rb -e rq07a
     ```

   b. Update the links on the `todo_items#edit` page (`todo_items/edit.html.erb`) to include the `TodoList` the `TodoItem` is a member of.

   - Change the `Show` link_to path parameter from a global `@todo_item` reference to include its `@todo_list`. This requires using both `@todo_list` and '@todo_item' passed in as separate arguments as an array.

     ```
     from: <%= link_to 'Show', @todo_item %> |
     to:   <%= link_to 'Show', [@todo_list, @todo_item] %> |
     ```

   - Change the `Back` link_to path parameter from a global `todo_items_path` (that no longer exists) to reference the `TodoList` it is a member of. This new method requires the `@todo_list` passed in as a single argument.

     ```
     from: <%= link_to 'Back', todo_items_path %>
     to:   <%= link_to 'Back', @todo_list %>
     ```

   c. Update the form parameters on the `TodoItems` form partial page (`todo_items/_form.html.erb`) to include the `TodoList` the `TodoItem` is a member of.

   - Change the link_to parameters from global `todo_item` references to provide fully qualified '[@todo_list, @todo_item]' references as an array.

     ```
     from: <%= form_for(@todo_item) do |f| %>
     to:   <%= form_for([@todo_list, @todo_item]) do |f| %>
     ```

     ```
     $ rspec spec/nested_resources_spec.rb -e rq07c
     ```

   d. Update the links on the `todo_items#new` page (`todo_items/new.html.erb`) to include the `TodoList`.

   - change the `Back` link_to path parameter from a global `todo_items_path` (that no longer exists) to reference the `TodoList` it is a member of. This new method requires the `@todo_list` passed in as a single argument.

     ```
     from: <%= link_to 'Back', todo_items_path %>
     to:   <%= link_to 'Back', @todo_list %>
     ```

     ```
     $ rspec spec/nested_resources_spec.rb -e rq07d
     ```

   e. Make the display of `completed` conditional on the `todo_item` being edited versus new. Users should not be allowed to see/change the `completed` property for a new `TodoItem`. (Hint: edited objects are persisted and can be tested using `persisted?`. Objects can also be tested with `.new_record?`)

```
$ rspec spec/nested_resources_spec.rb -e rq07e
```

```
$ rspec spec/nested_resources_spec.rb -e rq07
```

8. Verify that you have implemented a password login capability for the `User` model. You implemented this in an earlier step to allow the provided `db/seeds.rb` to immediately work with passwords. This this should just be a sanity check and review of how `has_secure_password` works.

   - Using the `rails console`, verify that you fail authentication when using the wrong password for a specific `User`. You can locate the `username` and assigned `password` in the `db/seeds.rb` file.

     ```
     $ rails c
     > user=User.where(username:"rich").first
     > user.authenticate("wrongpassword")
      => false
     ```

   - Using the `rails console`, verify that you can authenticate using a valid password for a specific `User`.

     ```
     > user=User.where(username:"rich").first
     > user.authenticate("123abc")
      => #<User id: 277, username: "rich", password_digest: "$2a...
     ```

   - Using the `rails console`, verify that you can authenticate and get the `TodoLists` for an authenticated `User`.

     ```
     > user=User.where(username:"rich").first
     > user.authenticate("123abc").todo_lists.count
      => 49 #seed data randomly generated
     ```

     ```
     $ rspec spec/security_spec.rb -e rq08
     ```

9. Create a new controller to manage the user's `session` when interacting with the server.

   a. Use the `rails g controller` command to create a `Sessions` controller with the following actions:

   - new
   - create
   - destroy

   b. Clean up the `config/routes.rb` file edited by the `rails g controller` command to be the following:

   - generated:

     ```
     get 'sessions/new'
     get 'sessions/create'
     get 'sessions/destroy'
     ```

   - change to:

     ```
     resources :sessions, only: [:new, :create, :destroy]
     ```

   c. Map the GET `/login` action to `sessions#new` in `config/routes.rb`. Have this be referred to as the `login` resource so `rake routes` generates a `login`_path helper.

```
get "/login" => "sessions#new", as: "login"
```

   d. Map the DELETE `/login` action to `sessions#destroy` in `config/routes.rb`. Have this referred to as the `logout` resource so `rake routes` reports a `logout`_path helper.

```
delete "/logout" => "sessions#destroy", as: "logout"
```

```
$ rspec spec/security_spec.rb -e rq09
```

10. Implement the `Sessions` controller class and view. This should permit a caller to willingly navigate to the `/login` page, login with a correct password, and proceed to the root URI. Nothing will stop an un-authenticated user from accessing the same list at this time. (Hint: the information to complete this step is contained in module 4, lesson 2, lecture:`Sessions and Controller View`)

    a. Leave the `new` method in its default state. This will cause the route to continue straight to `views/sessions/new.html.erb`.

    b. Update the `sessions#new` page (`views/sessions/new.html.erb`) to declare a form:

- for a `User` model type :user; this will cause the properties of the form to be assigned to an instance of a `User`
- with a `sessions_path` URI; this will cause a POST to the sessions#create URI to be invoked when a submit is called
- with a :username text_field; this will assign the user input to the `user[username]` property
- with a :password password_field; this will obfuscate the user's password while being typed and assign the user input to the `user[password]` property
- with a submit action; this will submit the form to the server when pressed

```
<h1>Login</h1>
<%= form_for(:user, url: sessions_path) do |f| %>
  <div class="field">
      <%= f.label :username %> <br/>
      <%= f.text_field :username %>
  </div>
  <p/>
  <div class="field">
      <%= f.label :password %> <br/>
      <%= f.password_field :password %>
  </div>
  <p/>
  <div class="actions">
      <%= f.submit "Login" %>
  </div>
<% end %>
```

```
$ rspec spec/security_spec.rb -e rq10b
```

    c. Implement the `create` method as follows:

- get the user's `username` and `password` from the submitted form
- find the user based on `username`
- authenticate the user using the supplied `password`
- if authenticated
- store the user.id in the `session`
- redirect the caller to the `root_path` of the application and supply a flash.notice message announcing successful login
- if not authenticated
- redirect the caller to the `login_path` and supply an flash.alert message

```
$ rspec spec/security_spec.rb -e rq10c
```

    d. Implement the `destroy` method as follows:

- reset the `session`, wiping out the user's session and everything in it
- redirect the caller to the `login_path` with a flash.notice message announcing successful logout

  ```
  $ rspec spec/security_spec.rb -e rq10d
  ```

e. Remove the `destroy` and `create` pages in the view, generated by `rails g controller` since they are not being used.

```
$ rspec spec/security_spec.rb -e rq10d
```

11. Require users to authenticate with your application prior to accessing anything except the login page. At the completion of these steps, no one should be able to access anything except the login page – until they successfully authenticate. (Hint: the information to complete this step is contained within module 4, lesson 2, lecture: `Authorization`)

    a. Define a `logged_in? helper method in the`ApplicationController`class that evaluates to`true' if the there is a user associated with the session.
    b. Define a `current_user` helper method in the `ApplicationController` class that finds and returns the `User` instance associated with the session.
    c. Expose `logged_in?` and `current_user` as helper methods outside of the controller using `helper_method`. Note these methods were already available to all controller sub-classes. This designation makes them available to the views as well.
    d. Define a `ensure_login` helper method in the `ApplicationController` class that redirects the caller to the `login_path` if they are not logged in. Note this method is available to all controller sub-classes in the application.
    e. Define that all methods perform `ensure_login` before they are called using `before_action`.
    f. Create an exception to the above rule so that `sessions#new` and `sessions#create` can be accessed by an unauthenticated user – otherwise no one will be able to access the login page.

```
class SessionsController < ApplicationController
  skip_before_action :ensure_login, only: [:new, :create]
```

    g. Update the `views/layouts/application.html.erb` page to include user/logout information based on the current session state. With this snippet in place – you should be able to login and see the current_user.username displayed in the right, top corner of the display.

```
<% if logged_in? %>
  <div style='float: right;'>
    Logged in as <%= current_user.username %> |
    <%= link_to "Logout", logout_path, method: :delete %>
  </div>
<% end %>
```

```
$ rspec spec/authentication_spec.rb -e rq11
```

12. Update the application so that authenticated users can only have access to `TodoLists` associated with their specific user. This mostly involves updating the `TodoListController` to change all global `TodoList` commands to be scoped `current_user.todo_lists` commands. (Hint: If you need some practice accessing `TodoLists` for an authenticated user, try the following commands in the `rails console`.

```
$ rails c
> user_id=User.where(username:"rich").first.id
> current_user=User.find(user_id)
> current_user.todo_lists.count
 => 49 #random assignment -- some number greater than 0
```

This mostly involves changing the following from/to.

```
from: TodoList.x
to:   current_user.todo_lists.x
```

At this point, logged in users should only be able to see their TodoLists

```
$ rspec spec/authentication_spec.rb -e rq12
```

13. Add pagination to your application to help scale and manage methods that can return unbounded collections of information.

    a. Verify the will_paginate gem is added to your Gemfile.

    b. Update the todolist#index action to return a page of TodoLists associated with the current_user that are up to 8 objects per_page. (Hint: If you are not familiar with how will_paginate works, you can get some familiarity using the rails c and Active Record commands. will_paginate adds an additional method to all model classes to be able to break find command results into pages and page results.)

```
$ rails c
> 3.times {|n| p TodoList.paginate(page:n+1, per_page:1)}
> p TodoList.paginate(page:1, per_page:1).total_pages
 => 101
```

The page number will be available in the params[:page] property of the call.

    c. Add will_paginate to your todolists#index page and apply it to your @todo_lists result from the controller.

At this point, logged in users should only be able to see their TodoLists and

```
$ rspec spec/pagination_spec.rb
```

14. Perform an end-to-end check of your work. Before you do, you must remove the confirmation dialogs from your Destroy links since we are not using a webdriver that supports javascript for this assignment. Inside the todo_list show.html.erb file you will need to change the destroy link to eliminate the confirmation dialog:

```
from:  <%= link_to 'Destroy', [@todo_list, todo_item], method: :delete, data: { confirm: 'Are you sure?
to: <%= link_to 'Destroy', [@todo_list, todo_item], method: :delete %>
```

Do the same for todo_list index.html.erb

```
from: <%= link_to 'Destroy', todo_list, method: :delete, data: { confirm: 'Are you sure?' } %>
to: <%= link_to 'Destroy', todo_list, method: :delete %>
```

    a. Login to the application as rich
    b. Access the first TodoList on the second page
    c. Complete the first TodoItem in that TodoList if not completed
    d. Create a new TodoItem for that TodoList
    e. Delete a TodoItem from that TodoList
    f. Create a new TodoList
    g. Delete a TodoList

```
rspec spec/end2end_spec.rb
```

Self Grading/Feedback

Some unit tests have been provided in the bootstrap files and provide examples of tests the grader will be evaluating for when you submit your solution. They must be run from the project root directory.

```
$ rspec (file)
...
(N) examples, 0 failures
```

You can run as many specific tests you wish be adding `-e rq## -e rq##`

```
$ rspec (file) -e rq01 -e rq02
```

Note that some of the earlier specs cannot be run once security has been fully enabled. Use the `end2end` test when complete. Each of the individual requirements list specific specs that can be used during the time of that development.

Submission

Submit an .zip archive (other archive forms not currently supported) with your solution root directory as the top-level (e.g., your Gemfile and sibling files must be in the root of the archive and *not* in a sub-folder. The grader will replace the spec files with fresh copies and will perform a test with different query terms.

```
|-- app
|   |-- assets
|   |-- controllers
|   |-- helpers
|   |-- mailers
|   |-- models
|   `-- views
|-- bin
|-- config
|-- config.ru
|-- db
|-- Gemfile
|-- Gemfile.lock
|-- lib
|-- log
|-- public
|-- Rakefile
|-- README.rdoc
|-- test
`-- vendor
```

Last Updated: 2015-10-25