HOW I CANCAN

Andrew Briening (abriening)

NOT AUTHENTICATION

- HTTP 1.1 Spec
- Authorization Header and "401 Unauthorized" are inaccurate
- Should be Authentication
- Devise, Authlogic, OmniAuth, Sorcery, Clearance, or ActiveModel::SecurePassword with ActionController::HttpAuthentication

AUTHORIZATION

- 403 Forbidden
- CanCan, Declarative Authorization, and a sharp drop-off to other solutions
- https://www.ruby-toolbox.com/categories/rails_authorization

CANCAN

- https://github.com/ryanb/cancan
- No runtime dependencies
- Decoupled from Rails, but includes plenty of helpers
- http://railscasts.com/episodes/192-authorization-with-cancan

INSTALL CANCAN

- Authentication & current_user should already exist
- gem ‘cancan’
- bundle install
- rails g cancan:ability

ABILITY

```ruby
class Ability
  include CanCan::Ability

  def initialize(user)
    # ...
  end
end
```

ABILITIES

- Role based?
- Type based?
- Anything-you-want based?

EXAMPLES

```ruby
class Ability
  include CanCan::Ability

  def initialize(user)
    user ||= User.new # guest user

    if user.role? :admin
      can :manage, :all
    else
      can :read, :all
      can :create, Comment
      can :update, Comment do |comment|
        comment.user == user
      end
    end
  end
end
```

EXAMPLES

```ruby
class Ability
  include CanCan::Ability

  def initialize(user)
    user ||= User.new # guest user

    admin if user.role?(:admin)
    moderator if user.role?(:moderator)
    translator if user.role?(:translator)
  end

  def admin
    can :manage, :all
  end

  def moderator
    # ...
  end

  def translator
    # ...
  end
end
```

EXAMPLES

```ruby
class Ability
  include CanCan::Ability

  module Admin
    def apply_rules
      # ...
    end
  end

  def initialize(user)
    user ||= User.new # guest user

    extend Admin if user.role?(:admin)
    # ...
    apply_rules if respond_to? :apply_rules
  end
end
```

CHECKING ABILITIES

```
Ability.new(@user).can?(:destroy, @project)
Ability.new(@user).cannot?(:destroy, @project)
```

```
# controller instance methods
authorize! :read, @article

def authorize!(*args)
  @_authorized = true
  current_ability.authorize!(*args)
end

# controller class methods
load_and_authorize_resource

load_resource

authorize_resource

check_authorization
```

CHECKING ABILITIES

```
# before
.form-actions
  = link_to t('.back'), worlds_path
  = link_to t('.edit'), edit_world_path(@world)
  = link_to t('.destroy'), world_path(@world), :method => "delete" ...
```

```
# after
.form-actions
  - if can? :read, World
    = link_to t('.back'), worlds_path
  - if can? :update, @world
    = link_to t('.edit'), edit_world_path(@world)
  - if can? :destroy, @world
    = link_to t('.destroy'), world_path(@world), :method => "delete" ...
```

ALIASES & SPECIAL CASES

```
:manage, :all, :read, :create, :update
```

```
def matches_action?(action)
  @expanded_actions.include?(:manage) || ...
end

def matches_subject?(subject)
  @subjects.include?(:all) || ... || ...
end
```

```
def default_alias_actions
  {
    :read => [:index, :show],
    :create => [:new],
    :update => [:edit],
  }
end
```

BEWARE MANAGE ALL

```ruby
# in ability.rb
  can :manage, World, :owner => { :id => user.id }

# in controller
def explode
  authorize! :explode, @world
  @world.explode # ;)
end
```

Can inadvertently allow actions

BEWARE MANAGE ALL

```
# in ability.rb
  can [:create, :read, :update], World, :owner => { :id => user.id }
# in controller
  authorize! :explode, @world # the world is safe again
```

"Deny, Allow" by explicitly defining each action

Acceptance testing is important

BEWARE MANAGE ALL

```ruby
# in ability.rb
  can [:create, :read, :update, :destroy], World
# in controller
  authorize! :manage, @world # nope!
```

authorize!(:manage) and can?(:manage, ...) only work if the ability was explicitly defined as can(:manage, ...)

USE INSTANCES WHEN POSSIBLE

```ruby
# rule
can :create, World, owner: { id: @user }

# This doesn't work, can? usage is not reciprocal with can rules
# :owner is silently ignored
can? :create, World, owner: { id: @user } # yep
can? :create, World, owner: { id: @other_user } # that's ok

# Pass in an instantiated record
can? :create, World # yes, surprisingly
can? :create, World.new(owner: @user) # yep
can? :create, World.new(owner: @other_user) # nope!
```

KEEP IT RESTFUL

```
# in ability.rb
  can [:create, :read, :update, :destroy], World
```

Stick to CRUD methods, :create, :read, :update, :destroy

KEEP IT RESTFUL

```ruby
# seems intuitive
# in ability.rb
  can :play, World

# check in worlds controller
  def play
    can? :play, @world
    # ...
  end

# but gets awkward; how do I "unplay"?
  can :stop_server, World, server: { world: { owner: {id: user} } }
  can? :stop_server, @world
```

```ruby
# better
  can [:create, :destroy], Server, world: {owner: {id: user}}

# check
  can? :create, Server.new(@world)
  can? :destroy, @server
```

CONTROLLER HELPERS

```
# class methods
load_and_authorize_resource

load_resource

authorize_resource

check_authorization
```

- Helpful in getting things setup fast, DRY, but ...
- Breaks single responsibility
- Adds complexity when need to override method of loading

CONTROLLER HELPERS

```
# world_controller, before
  def index
    authorize! :read, World
    @worlds = worlds.all
    respond_with @worlds
  end

  def show
    @world = worlds.find(params[:id])
    authorize! :read, @world
    respond_with @world
  end

# world_controller, after
  load_and_authorize_resource
  def index
    respond_with @worlds
  end

  def show
    respond_with @world
  end
```

ACCESSIBLE BY

```ruby
World.accessible_by current_ability

# chain with scopes
World.active.accessible_by current_ability

# chain with collection associations
@world.players.accessible_by(current_ability)

# converted to scope
can :update, World, owner: user

# Boom! Can’t use block syntax and accessible_by
can :update, World do |world|
  world.owner == user
end
```

CANCAN::ACCESSDENIED

```
class ApplicationController < ActionController::Base

  def authorization_error
    # 403 Forbidden response
    respond_to do |format|
      format.html{ render '/rescues/access_denied', :status => 403 }
      format.xml{  render :xml => 'Access Denied',  :status => 403 }
      format.json{ render :json => 'Access Denied', :status => 403 }
    end
  end

  rescue_from CanCan::AccessDenied, :authorization_error
end
```

410 GONE