Ember CLI & ember-simple-auth-devise

~~16/Jun/2014~~ **errata**

20/Oct/2014

We will build a Rails project and an Ember project from scratch. The backend will manage authentication with the gem **Devise**. The frontend will be able to `login`, to the Rails backend, using the library **Ember Simple Auth** (ESA) with its Devise extension.

The complete source code for the app is hosted in Github. Check the **ember-cli-simple-auth-devise** repository for the most up to date code.

Tools needed:

```
>_
```

```
ember -v
version: 0.1.1
node: 0.10.29
npm: 2.1.2
# ember-simple-auth 0.6.7

rails -v
Rails 4.0.2
```

First step, create a project folder:

```
>_
```

```
mkdir ember-cli-simple-auth-devise
cd ember-cli-simple-auth-devise
```

Backend

Rails and Devise

Create a Rails project that uses Devise.

ember-cli-simple-auth-devise>_

```
rails new my-backend
cd my-backend

# Basic Devise install
echo "gem 'devise'" >> Gemfile
bundle install
rails generate devise:install
rails generate devise User
```

Configure Devise

Devise dropped token support, so we are adding it back to our backend. We will also add a controller that will handle Ember Simple Auth requests.

my-backend>_

```
rails generate migration add_authentication_token_to_users \
                         authentication_token:string
rake db:migrate

rails generate controller sessions
```

Auto-generate an authentication token on model creation.

my-backend/app/models/user.rb

```
class User < ActiveRecord::Base
  before_save :ensure_authentication_token

  # leave the device line
  # devise :database_authenticatable etc.

  def ensure_authentication_token
    if authentication_token.blank?
      self.authentication_token = generate_authentication_token
    end
  end

  private

    def generate_authentication_token
      loop do
        token = Devise.friendly_token
        break token unless User.where(authentication_token: token).first
      end
    end
end
```

Enable Devise to respond to JSON requests.

my-backend/app/controllers/sessions_controller.rb

```ruby
# Note that we are extending
# from: Devise::SessionsController
class SessionsController < Devise::SessionsController
  def create
    respond_to do |format|
      format.html { super }
      format.json do
        self.resource = warden.authenticate!(auth_options)
        sign_in(resource_name, resource)
        data = {
          user_token: self.resource.authentication_token,
          user_email: self.resource.email
        }
        render json: data, status: 201
      end
    end
  end
end
```

Use the controller that we just made in the previous step.

my-backend/config/routes.rb

```ruby
#replace this line
#devise_for :users
devise_for :users, controllers: { sessions: 'sessions' }
```

Authenticate users with their email and authentication token.

my-backend/app/controllers/application_controller.rb

```ruby
class ApplicationController < ActionController::Base
  before_filter :authenticate_user_from_token!

  # leave the line:
  # protect_from_forgery

  private

    def authenticate_user_from_token!
      authenticate_with_http_token do |token, options|
      user_email = options[:user_email].presence
      user       = user_email && User.find_by_email(user_email)

      if user && Devise.secure_compare(user.authentication_token, token)
        sign_in user, store: false
      end
    end
  end
```

```
end
```

Disable sessions and cookies

When using Ember Simple Auth with a Devise backend, it doesn't make sense to keep the server side session active as it will send cookies to the client which are actually redundant when using the authentication token mechanism.

We will disable completely creating sessions. If for some reason you have to keep the session, there are other options, you can read about them here: **ember-simple-auth#201**.

my-backend/config/initializers/session_store.rb

```
Rails.application.config.session_store :disabled
```

my-backend/app/controllers/application_controller.rb

```
protect_from_forgery with: :null_session
```

Dummy data

We will register two users to the database, so we can test logging-in later.

my-backend/db/seeds.rb

```
User.create([
  {email: 'green@mail.com',
   password: '12345678', password_confirmation: '12345678'},
  {email: 'pink@mail.com',
   password: '12345678', password_confirmation: '12345678'}
])
```

Load the data and start the server.

my-backend>_

```
rake db:seed
rails server
```

Frontend

Ember CLI

Create a very simple Ember application with links to three pages:

- index
- protected
- login

ember-cli-simple-auth-devise>_

```
ember new my-frontend
cd my-frontend

ember generate route application
ember generate route protected
ember generate route login
ember generate controller login
ember generate template index

echo "landing page" > app/templates/index.hbs
echo "this is a protected page" > app/templates/protected.hbs
```

my-frontend/app/templates/application.hbs

```
<h2 id='title'>Frontend</h2>

{{#link-to 'index'}}Home{{/link-to}}
{{#link-to 'protected'}}Protected{{/link-to}}

<hr>
{{outlet}}
```

You may test the app.

my-frontend>_

```
ember server
# visit http://0.0.0.0:4200
```

Ember Simple Auth

Install

We will be using Ember Simple Auth packaged as an ember cli addon.

my-frontend>_

```
npm install --save-dev ember-cli-simple-auth
npm install --save-dev ember-cli-simple-auth-devise
ember generate ember-cli-simple-auth
ember generate ember-cli-simple-auth-devise
```

my-frontend/config/environment.js

```
module.exports = function(environment) {
  ENV['simple-auth'] = {
    authorizer: 'simple-auth-authorizer:devise'
  };
}
```

Edit routes

We will use the route mixins provided by Ember Simple Auth.

The ApplicationRouteMixin provides the authenticate and invalidate actions.

my-frontend/app/routes/application.js

```
import Ember from 'ember';
import ApplicationRouteMixin from 'simple-auth/mixins/application-route-mixin';

export default Ember.Route.extend(ApplicationRouteMixin);
```

The AuthenticatedRouteMixin makes a route available only to logged in users.

my-frontend/app/routes/protected.js

```
import Ember from 'ember';
import AuthenticatedRouteMixin from 'simple-auth/mixins/authenticated-route-mixin';

export default Ember.Route.extend(AuthenticatedRouteMixin);
```

Edit controllers

my-frontend/app/controllers/login.js

```
import Ember from 'ember';
import LoginControllerMixin from 'simple-auth/mixins/login-controller-mixin';

export default Ember.Controller.extend(LoginControllerMixin, {
  authenticator: 'simple-auth-authenticator:devise'
});
```

Templates and Ember Simple Auth helpers

my-frontend/app/template/application.hbs

```
{{#if session.isAuthenticated}}
  <button {{ action 'invalidateSession' }}>Logout</button>
{{else}}
  {{#link-to 'login'}}Login{{/link-to}}
{{/if}}
#<hr>
```

my-frontend/app/templates/login.hbs

```
<form {{action 'authenticate' on='submit'}}>
  <label for='identification'>Login</label>
  {{input id='identification' placeholder='Enter Login'
           value=identification}}
  <label for='password'>Password</label>
  {{input id='password' placeholder='Enter Password'
           type='password' value=password}}
  <button type='submit'>Login</button>
</form>
```

Try it out

my-frontend>_

```
ember server --proxy http://0.0.0.0:3000
# Visit http://0.0.0.0:4200

# Don't forget to have the rails server running too.
```

That is all, logging-in should be working now.

Errors

This are errors that, if you followed all the instructions, should not be appearing.

Error 422 - Unprocessable Entity

In the console you get a 422 error accompanied by a rails server error:

> POST http://localhost:4200/token 422 (Unprocessable Entity)
>
> Can't verify CSRF token authenticity Completed 422 Unprocessable Entity in 29ms

Solution

ember-cli needs to keep track of the CSRF token, this can be solved with **rails-csrf**. We will make changes to both the backend and the frontend.

Rails

my-backend>_

```
rails generate controller api/csrf
```

my-backend/app/controllers/api/csrf_controller.rb

```
# add an index action
  def index
    render json: { request_forgery_protection_token => form_authenticity_token }.to_json
end
```

my-backend/config/routes.rb

```
# add a namespace
namespace :api do
  get :csrf, to: 'csrf#index'
end
```

Ember

Install rails-csrf

my-frontend>_

```
bower install rails-csrf#0.0.5 --save
```

my-frontend/Brocfile

```
app.import('vendor/rails-csrf/dist/named-amd/main.js', {
  'rails-csrf': [
    'service'
  ]
});
```

my-frontend/app/app.js

```
// add this line before the one below
loadInitializers(App, 'rails-csrf');

//export default App;
```

my-frontend/app/routes/application.js

```
-export default Ember.Route.extend(ApplicationRouteMixin);
+export default Ember.Route.extend(ApplicationRouteMixin, {
+  beforeModel: function () {
+    this._super.apply(this, arguments);
+    return this.csrf.fetchToken();
+  }
+});
```

my-frontend/config/environment.js

```
  module.exports = function(environment) {
    var ENV = {
+     railsCsrf: {
+       csrfURL: 'api/csrf'
+     }
    };
  };
```

Everything should be working now, start both servers.

Error 404

```
POST http://localhost:4200/token 404 (Not Found)
# chances are you didn't specify a proxy when starting the ember server
```

Error 500

```
POST http://localhost:4200/token 500 (Internal Server Error)
# chances are you specified the wrong proxy when starting the ember server
```

Error 408 or 500 - Time out

In the console you get a 408 error:

```
POST http://localhost:4200/token 408 (Request Time-out)
```

Or a 500 error accompanied by an ember server error:

```
POST http://localhost:4200/token 500 (Internal Server Error)
```

```
Error: socket hang up
  at createHangUpError
```

Solution

my-frontend/server/index.js

```
//app.use(bodyParser());
```

This error appeared in previous versions of ember-cli, but no longer does. To know more see **ember-cli#723**.

22 Comments givan.se Login

Sort by Best Share Favorite

Join the discussion…

Patrick Baselier · 5 months ago

Wow! Works great.

Are you planning to update the post with ember-cli updates?

3 · Reply · Share ›