Episode 236

OmniAuth Part 2

In the last episode [watch[1], read[2]] we showed you how to add various forms of authentication to a user account in a Rails application using OmniAuth. By the end of that episode users who had already signed in with a username and password could also authenticate via third-party services such as Twitter. In this episode we'll continue to look at OmniAuth and will extend our application so that it can handle users who have signed out and want to sign back and also users who have not signed in at all.

Updating OmniAuth

A new version of OmniAuth that includes some important fixes has been released since the last episode was written. To update our application to use the new version we just need to run

```
$ bundle update
```

from the application's directory. This new version of OmniAuth changes the name of the request environment variable that it uses from `rack.auth` to `omniauth.auth` and so we'll have to update our application to reflect this.

/app/controllers/authentications_controller.rb

```
def create
  auth = request.env["ominauth.auth"]
  current_user.authentications.find_or_create_by_provder_and_uid↵
    (auth['provider'], auth['uid'])
  flash[:notice] = "Authentication successful."
  redirect_to authentications_url
end
```

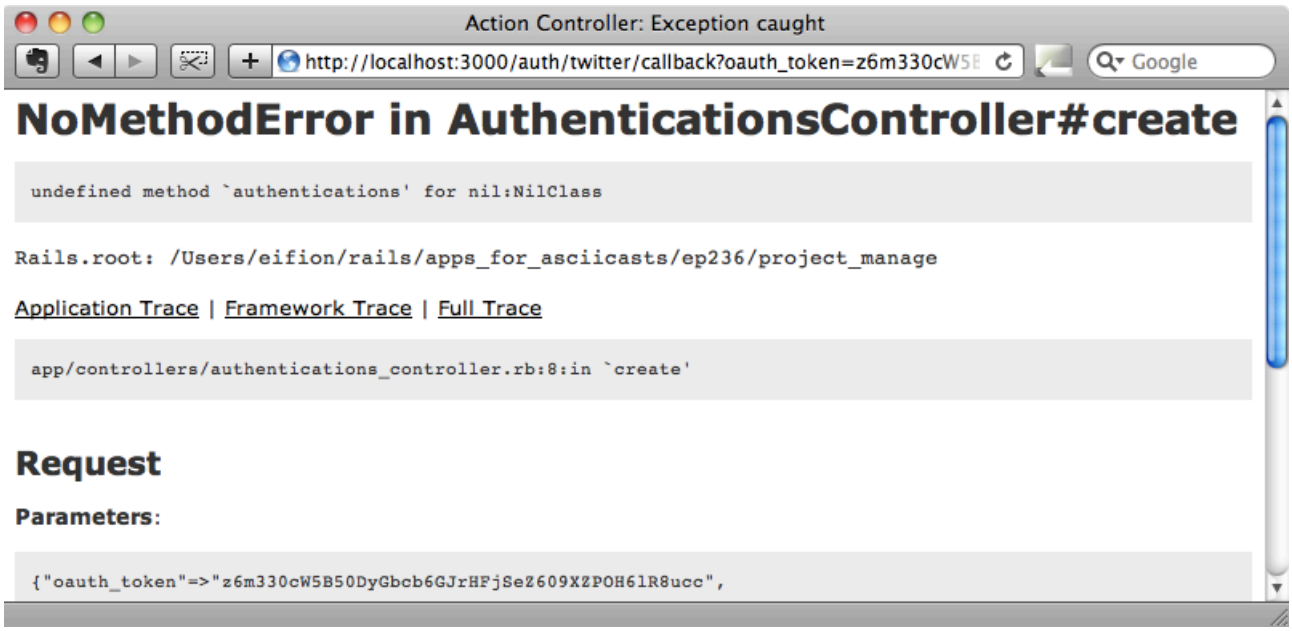
---

[1] http://railscasts.com/episodes/235-omniauth-part-1

[2] http://asciicasts.com/episodes/235-omniauth-part-1

Signing In With OmniAuth

The first change we'll make to the application is to enable it to allow existing users to sign in via Twitter. Currently if we try this we'll be taken to Twitter but when we authenticate there and are redirected back to our application we'll see an error message.



The reason this error is thrown is that the `AuthenticationController`'s `create` action expects there to be a currently logged-in user and tries to find or create a new authentication for that user. As we're trying to authenticate without having first logged in with a user name and password the `current_user` variable will be `nil`.

/app/controllers/authentications_controller.rb

```
def create
  auth = request.env["ominauth.auth"]
  current_user.authentications.find_or_create_by_provder_and_uid↵
    (auth['provider'], auth['uid'])
  flash[:notice] = "Authentication successful."
  redirect_to authentications_url
end
```

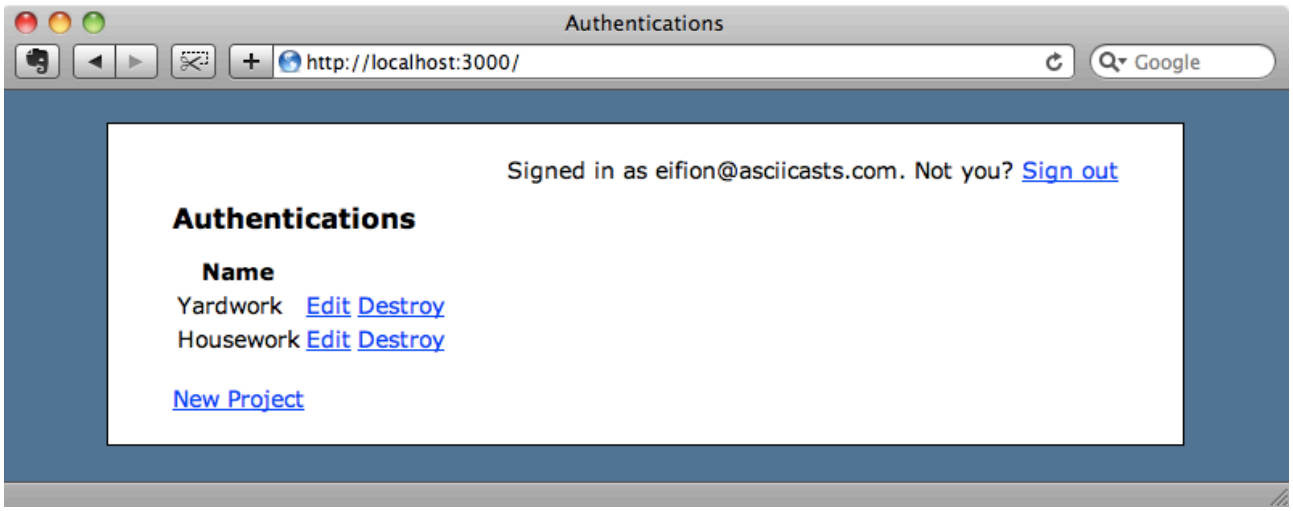
We'll need to change this code so that it tries to find the appropriate authentication and its associated user. If it finds one then the application will sign the user in, otherwise it will create one. We still need to handle the scenario where no authentication exists for a new user, but we'll come back to that later.

/app/controllers/authentications_controller.rb

```
def create
  omniauth = request.env["omniauth.auth"]
  authentication = Authentication.find_by_provder_and_uid ↵
    (omniauth['provider'], omniauth['uid'])
  if authentication
    flash[:notice] = "Signed in successfully."
    sign_in_and_redirect(:user, authentication.user)
  else
    current_user.authentications.create(:provider => omniauth ↵
      ['provider'], :uid => omniauth['uid'])
    flash[:notice] = "Authentication successful."
    redirect_to authentications_url
  end
end
```

The `create` method now looks for an `Authentication` based on the `provider` and `uid` parameters from the `omniauth.auth` environment variable. If it finds one then we want that user to be signed in, so we use Devise's `sign_in_and_redirect` method to sign in the user associated with that authentication. If a matching authentication isn't found then we'll create a new authentication for the current user.

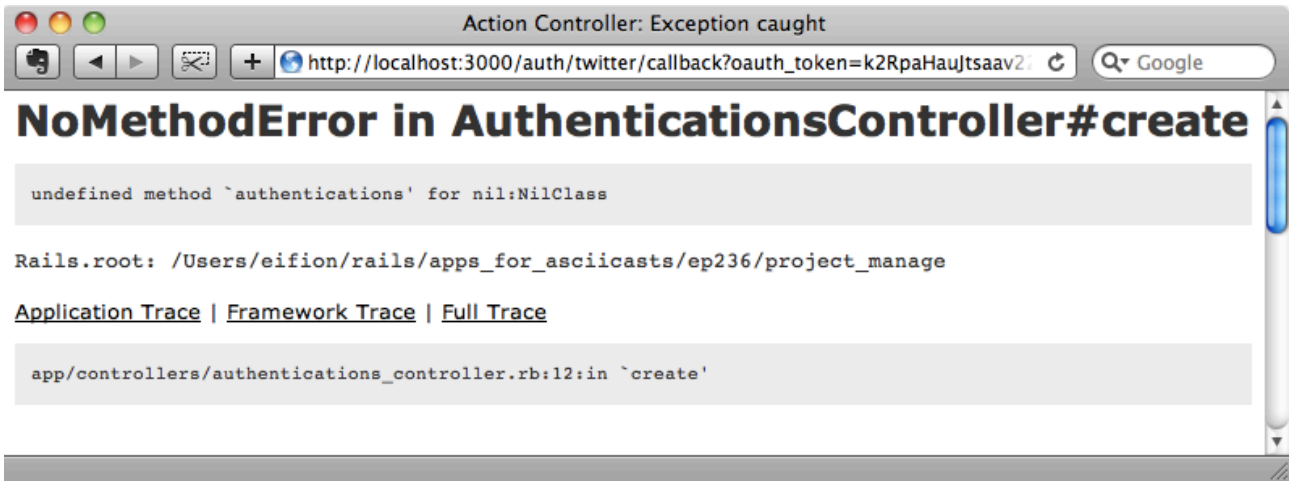
We can test this out by visiting the authentications page while we're not logged in and signing in via Twitter. We'll should be signed in to the application and then redirected back to the home page.



Handling Brand-New Users

Next we'll need change our application to handle brand-new users coming to our site and trying to sign in through Twitter. Our application won't work for these users as it stands as the code in the `create` action still expects a `current_user` when it fails to find an existing authentication.

We can see what will happen in this case by signing out of both Twitter and our app then logging in with a different Twitter account. Twitter redirects back to our application but we get an error message as the code in the `create` action tries to retrieve authentications for the non-existent `current_user`.



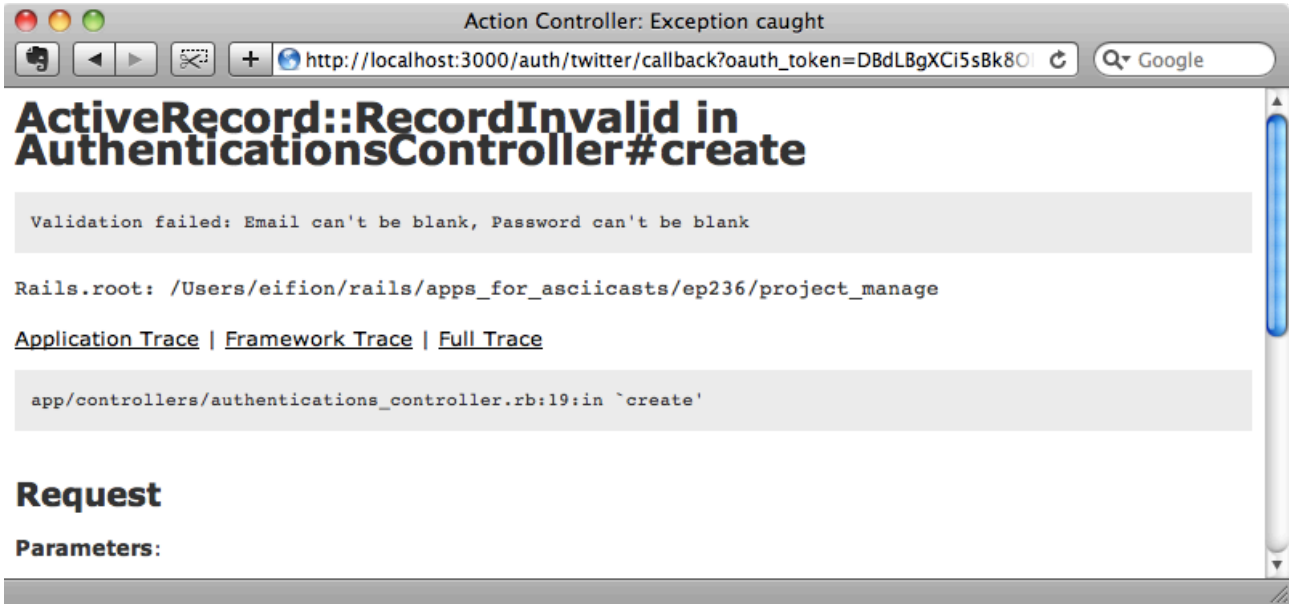
To fix this we need to add another condition in the `create` action so that it can handle this situation.

/app/controllers/authentications_controller.rb

```
def create
  omniauth = request.env["omniauth.auth"]
  authentication = Authentication.find_by_provider_and_uid ↵
    (omniauth['provider'], omniauth['uid'])
  if authentication
    flash[:notice] = "Signed in successfully."
    sign_in_and_redirect(:user, authentication.user)
  elsif current_user
    current_user.authentications.create(:provider => omniauth ↵
      ['provider'], :uid => omniauth['uid'])
    flash[:notice] = "Authentication successful."
    redirect_to authentications_url
  else
    user = User.new
    user.authentications.build(:provider => omniauth ↵
      ['provider'], :uid => omniauth['uid'])
    user.save!
    flash[:notice] = "Signed in successfully."
    sign_in_and_redirect(:user, user)
  end
end
```

This method now handles three different conditions. If a matching authentication is found it signs in the user associated with that authentication. If there is no matching authentication, but there is a currently signed-in user an authentication is created for that user. Finally, if there is no current user then a new user is created, a new authentication is built for that user and the user is signed in.

If we try logging in as a new user via Twitter now we get a different error message when Twitter redirects back to our application.



This time the application throws the error "Validation failed: Email can't be blank, Password can't be blank". Devise reports these validation errors on the `User` model because it assumes that these two fields exist on the model.

In the `User` model we've included `:validatable` in the list of options that we've passed to the `devise` method and it's here that the validations for the email and password fields are set.

/app/models/user.rb

```
class User < ActiveRecord::Base
  has_many :authentications
  # Include default devise modules. Others available are:
  # :token_authenticatable, :lockable, :timeoutable
  # :confirmable and :activatable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

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

The simplest way to fix this would be to remove the `:validatable` option. This would work if we only wanted to support authentication via Twitter but we also want to keep the ability for people to log in with a username and password so this isn't an option as it would remove the validation in this case. Alternatively we could save the new user without validating it by replacing

```
user.save!
```

in the `create` action with

```
user.save(:validate => false)
```

This will skip the validation when we save the user but of course it would mean that we could end up with invalid data in the database. The `User` model has a `username` field and we want each username to be unique. Skipping the validation here could easily lead to duplicated usernames.

Unfortunately there is no easy solution to this problem. What we'll do is keep the validation and redirect the user to a form where they can fix any problems if the validation fails when we try to save the new user. We'll change the code in `create` so that if the validation fails when we try to save a new user they'll be redirected the new user registration page. We don't want to lose the user's OmniAuth information when this happens and so we'll store it in the session. The OmniAuth hash can sometimes contain too much data to fit in the cookie session store so we'll remove the `extra` key. This key stores a lot of information that we don't need to get the user registered so it is safe to remove it.

/app/controllers/authentications_controller.rb

```
user = User.new
user.authentications.build(:provider => omniauth['provider'], ↵
  :uid => omniauth['uid'])
if user.save
  flash[:notice] = "Signed in successfully."
  sign_in_and_redirect(:user, user)
else
  session[:omniauth] = omniauth.except('extra')
  redirect_to new_user_registration_url
end
```

The next thing we need to do is to customize the way the registration controller behaves. This is buried deep inside the Devise code but we can override it by creating a new registration controller.

```
$ rails g controller registrations
```

We'll need views to go with this controller and we can copy over the ones that Devise uses by using the `devise:views` generator.

```
$ rails g devise:views
```

This copies the views from the Devise engine into the `/app/views/devise` directory in our application. In this directory is a `registrations` directory that contains two `erb` files. We'll need to move these two files into `/app/views/registrations` so that they work with our new controller.

Next we need to change the routes file so that we tell Devise to use our controller for registrations rather than its default one.

`/config/routes.rb`

```
ProjectManage::Application.routes.draw do |map|
  match '/auth/:provider/callback' => 'authentications#create'
  devise_for :users, :controllers => { :registrations => ↵
    'registrations' }
  resources :projects
  resources :tasks
  resources :authentications
  root :to => 'projects#index'
end
```

In the controller we've just generated we want to override some of Devise's registrations controller's functionality so we'll modify to so that it inherits from `Devise::RegistrationsController` rather than from `ApplicationController`. Devise's `RegistrationsController` class has a method called `build_resource` that builds a `User` model inside the `new` and `create` actions. By overriding this method we can add some custom behaviour to the user model that is created and add an associated authentication based on the information that we saved in the OmniAuth session variable. What we'll do here is very similar to what we do in the `create` action in the `AuthenticationsController` so we'll extract this functionality out into a new method in the `User` model that we'll call `apply_omniauth`.

/app/models/user.rb

```
def apply_omniauth(omniauth)
  authentications.build(:provider => omniauth['provider'], ↵
    :uid => omniauth['uid'])
end
```

We can now use this method in the `AuthenticationController`'s `create` action.

/app/controllers/authentications_controller.rb

```
user = User.new
user.apply_omniauth(omniauth)
if user.save
  flash[:notice] = "Signed in successfully."
  sign_in_and_redirect(:user, user)
else
  session[:omniauth] = omniauth.except('extra')
  redirect_to new_user_registration_url
end
```

And also in our new `RegistrationsController`.

/app/controllers/registrations_controller.rb

```
class RegistrationsController < Devise::RegistrationsController
  private
  def build_resource(*args)
    super
    if session[:omniauth]
      @user.apply_omniauth(session[:omniauth])
      @user.valid?
    end
  end
end
```

We only want to do this if the session variable exists, so we check for it first. Also we'll validate the user so that the validations are shown in the `new` action when a user is redirected there.

Let's see if this new code works. If we visit the application while not signed in and authenticate via Twitter we'll now be redirected back to the sign up page as we don't yet have a user account and we'll see the validation errors.

We don't want to validate the password field when the user has other means of authentication so in the `User` model we'll disable the password validations. Devise gives us a way to do this by overriding the boolean `password_required?` method. We only want to validate the password when the user has no other authentications or when they are trying to add a password. We also want to delegate to `super` so that we get the overridden behaviour is applied as well.

`/app/models/user.rb`

```
def password_required?
  (authentications.empty? || !password.blank?) && super
end
```

We want to hide the password fields in the signup form when they're not required and we can use `password_required?` in the view code to do this.

/app/views/registrations/new.html.erb

```
<h2>Sign up</h2>

<%= form_for(resource, :as => resource_name, :url =>
registration_path(resource_name)) do |f| %>
  <%= devise_error_messages! %>

  <p><%= f.label :email %><br />
  <%= f.text_field :email %></p>

<% if @user.password_required? %>
  <p><%= f.label :password %><br />
  <%= f.password_field :password %></p>

  <p><%= f.label :password_confirmation %><br />
  <%= f.password_field :password_confirmation %></p>
<% end %>
  <p><%= f.submit "Sign up" %></p>
<% end %>

<%= render :partial => "devise/shared/links" %>
```

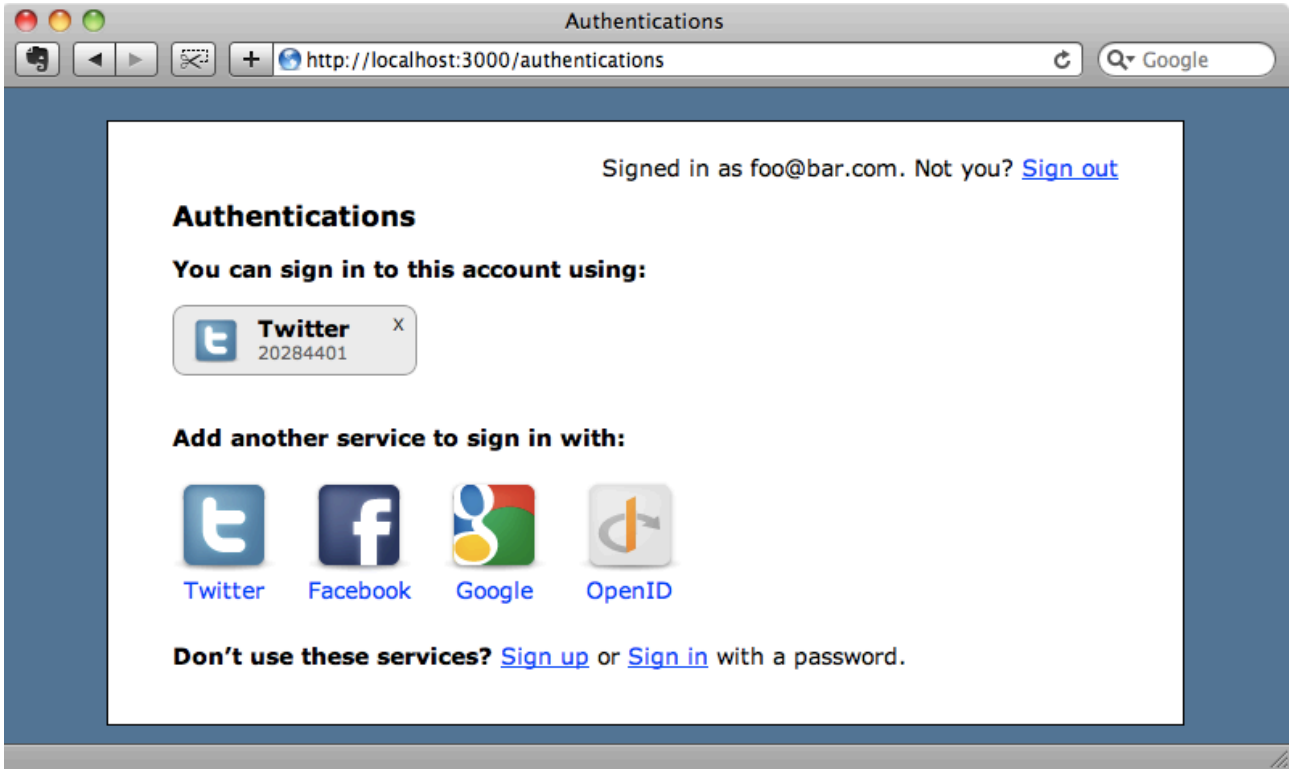
If we reload the form now we'll see that there is only one validation error displayed and that the password fields are not shown.

There's one more thing we need to do inside the `RegistrationsController` and that is to remove the OmniAuth data from the session once the user has successfully registered. We can do that by overriding the `create` action.

/app/controllers/registration_controllers.rb

```
def create
  super
  session[:omniauth] = nil unless @user.new_record?
end
```

Now, when the user successfully registers the OmniAuth session data will be removed. If we fill in an email address in the form now we'll be able to sign up successfully and when we go to the authentications page we'll see our Twitter authentication listed.



Adding a Service To OmniAuth

We'll finish off by showing how easy it is to add another service to OmniAuth by adding OpenID support. Our application is currently using WEBrick in development mode and OpenID has some problems when used with WEBrick because it uses some long URLs so we'll switch over to Mongrel.

To do this we just need to add a reference to Mongrel in our application's `Gemfile`. We'll specify version `1.2.0.pre` as this works with Rails.

/Gemfile

```
gem 'mongrel', '1.2.0.pre2'
```

Next we need to modify our OmniAuth config file and and add OpenId to the list of providers.

/config/initializers/omniauth.rb

```
require 'openid/store/filesystem'
Rails.application.config.middleware.use OmniAuth::Builder do
  provider :twitter, 's3dXXXXXXXX', 'lR23XXXXXXXXXXXXXXXXX'
  provider :open_id, OpenID::Store::Filesystem.new('/tmp')
end
```

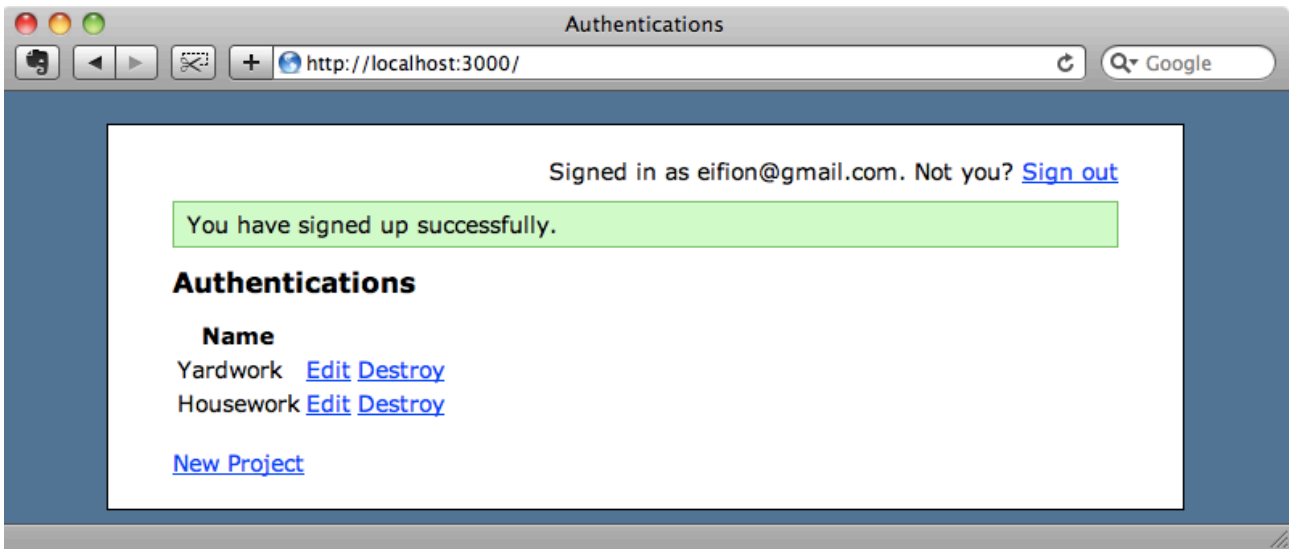
We're using a file-system store with OpenID. If your host doesn't provide this then you could use memcached or an ActiveRecord store, but OpenID does require a persistent store. Note that OpenID doesn't include the filesystem store automatically so we need to `require` it at the top of the file.

Lastly we need to go into the `User` model and change the way that the `apply_omniauth` method works. OpenID provides an email parameter and we'll use it in the `User` model unless the user already has an email address.

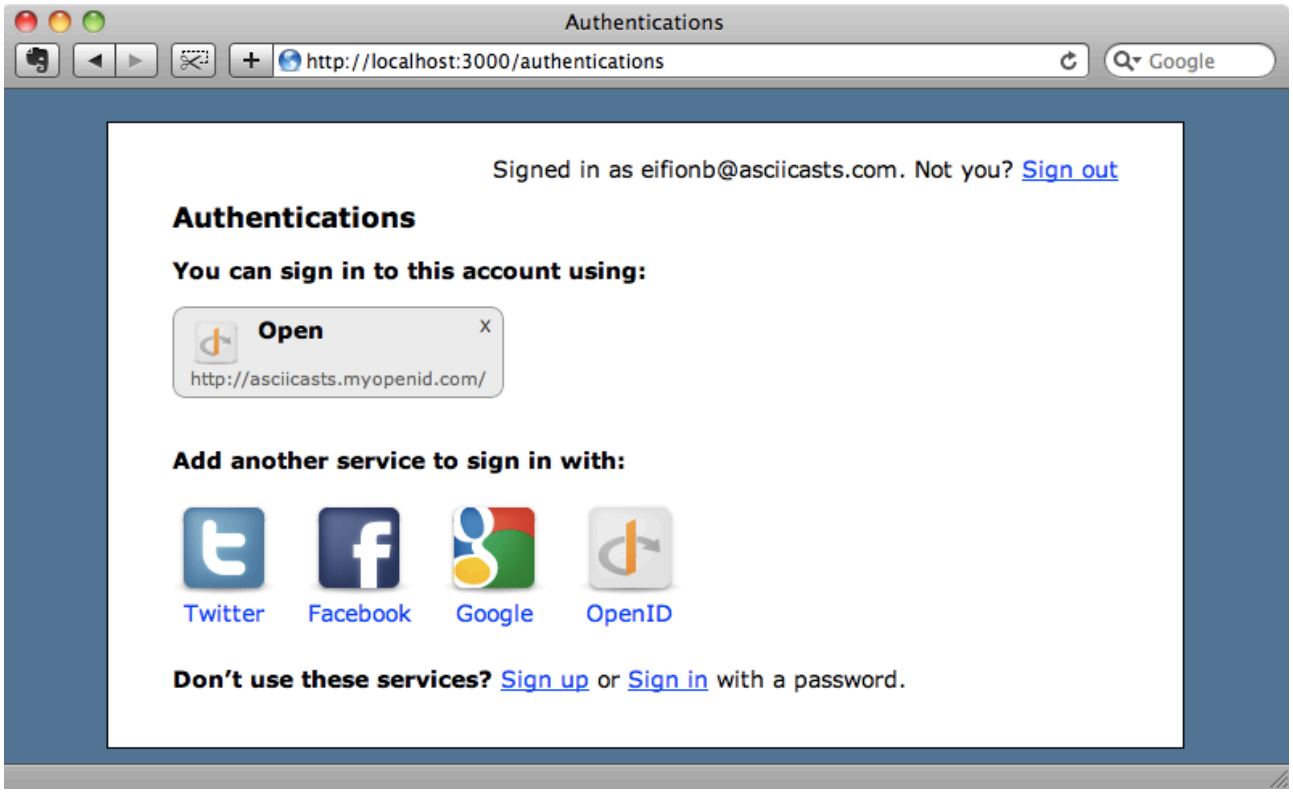
/app/models/user.rb

```
def apply_omniauth(omniauth)
  self.email = omniauth['user_info']['email'] if email.blank?
  authentications.build(:provider => omniauth['provider'], ↵
    :uid => omniauth['uid'])
end
```

We can try this out now by signing in with an OpenID account. When we do we'll be redirected to our OpenID provider's website so that we can log in there and then we're redirected back to our application and signed in. The email address from our OpenID account is automatically applied to our account details here.



If we look at our authentications page as well we'll see the OpenID authentication listed.



The title for the OpenID authentication on the page above is wrong so we'll quickly fix it. The title for each authentication is rendered using the following code:

/app/views/authentications/index.html.erb

```
<div class="provider">
  <%= authentication.provider.titleize %>
</div>
```

Using `titleize` doesn't really work in this case so instead we'll write a new method in the `Authentication` class and use that here instead.

/app/views/authentications/index.html.erb

```
<div class="provider">
  <%= authentication.provider_name %>
</div>
```

The `provider_name` method is straightforward and looks like this.

/app/models/authentication.rb

```
class Authentication < ActiveRecord::Base
  belongs_to :user

  def provider_name
    if provider == 'open_id'
      "OpenID"
    else
      provider.titleize
    end
  end
end
```

That's all we have time for in this episode. OmniAuth is an impressive gem, especially given the number of services it can connect to. Once you get its foundations established then it's easy to add services as you need them. Shoehorning them into an existing user account system is a little tricky and we've not covered all of the details here but there should been enough here to give you a good start when it comes to using OmniAuth with Devise.