



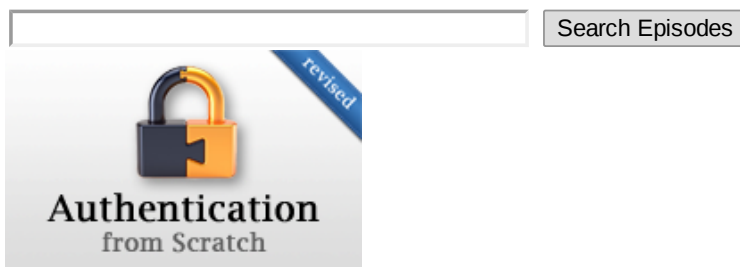




[Sign in through GitHub](#) | [Manage Subscription](#) | [Log Out](#)

-  watch on iTunes
-  follow on Twitter
-  follow on Facebook
-  subscribe to RSS feed
- [Browse Episodes](#)
- [RailsCasts Pro](#)
- [About](#)
- [Feedback](#)



#250 Authentication from Scratch (revised)

Feb 25, 2012 | 13 minutes | [Authentication](#)

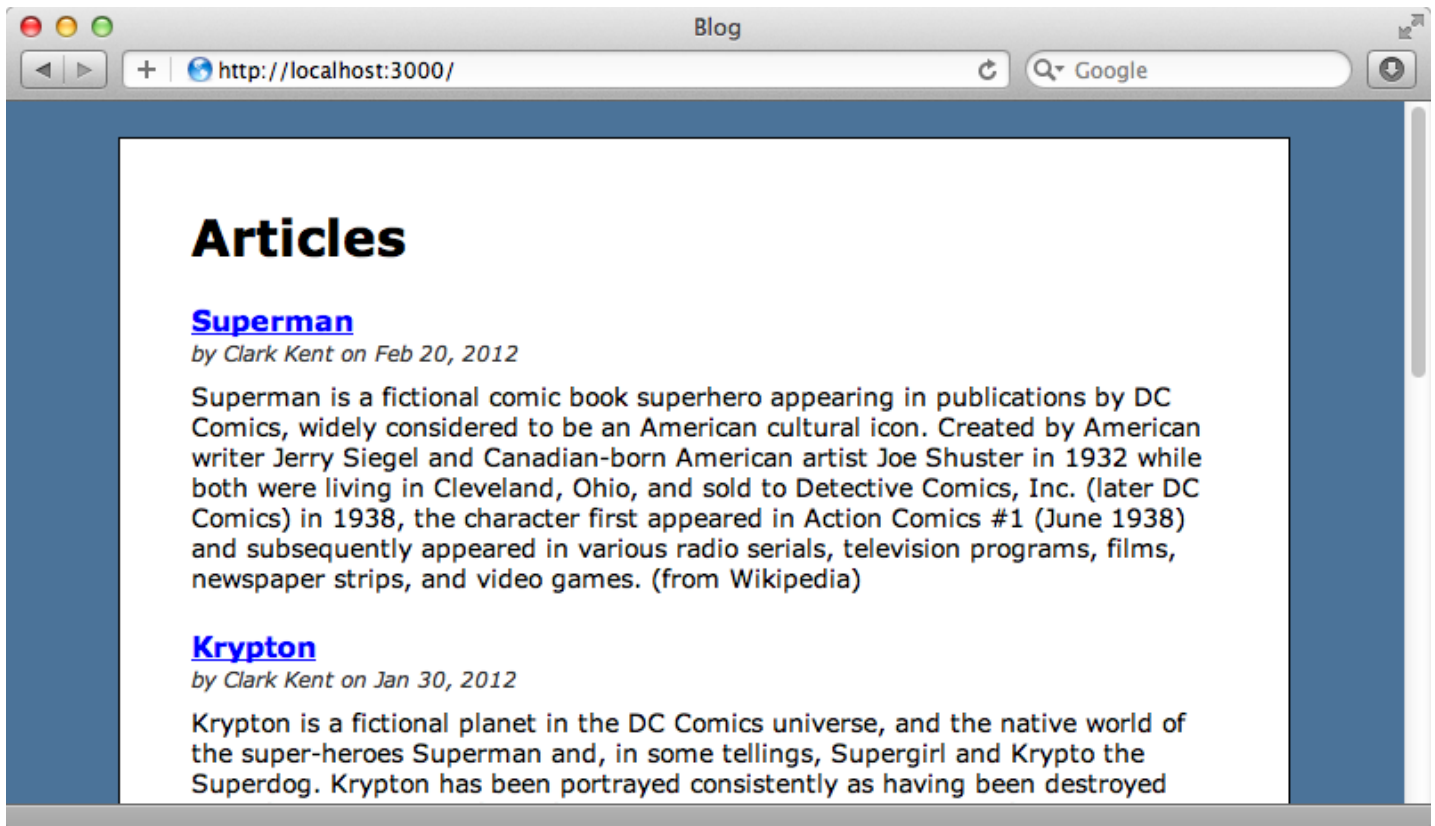
Simple password authentication is easy to do with `has_secure_password`. Here you will learn how to make a complete Sign Up, Log In, and Log Out process as well as restrict access to certain actions.

[Click to Play Video](#) ▶

Tweet { 17 }

- Download:
- [source code](#) Project Files in Zip (102 KB)
- [mp4](#) Full Size H.264 Video (25 MB)
- [m4v](#) Smaller H.264 Video (14.4 MB)
- [webm](#) Full Size VP8 Video (17.5 MB)
- [ogv](#) Full Size Theora Video (30.7 MB)
- [Show Notes](#)
- [ASCIICast](#)
- [46 Comments](#)
- [Similar Episodes](#)
- [Next Episode >](#)
- [< Previous Episode](#)

User authentication is required in almost every Rails application, including the blogging application shown below. This app currently has no authentication at all and we want to add some so that users can sign up and then log in to and out of the site.



There are several gems that will help us to add authentication to an app, such as Devise and OmniAuth. Sometimes though we just need simple password-based authentication and in these cases it's not difficult to create it from scratch. We'll do just that in this episode.

Adding a Signup Form

The first thing we need to do is to add a sign-up form for creating a new user and then add a link to it. This means that we'll need to generate a User model and a controller to handle the signup process. We'll use the resource generator to do this. This is similar to the scaffold generator but it doesn't fill in all the controller actions. We'll give the User model email and password_digest fields.

terminal

```
$ rails g resource user email password_digest
```

Having a password_digest field is important as it's the default name that's used with Rails' has_secure_password feature and we'll be using this feature later. This command will generate a User model, a UsersController and a database migration. We'll migrate the database before we continue.

terminal

```
$ rake db:migrate
```

Next we'll add has_secure_password to the User model. This was introduced in Rails 3.1 and adds some simple authentication support to the model using that password_digest column.

/app/models/user.rb

```
class User < ActiveRecord::Base
  has_secure_password
end
```

To get this working in Rails we'll also need to modify the gemfile and uncomment the line that includes the bcrypt-ruby gem as this gem handles hashing the password before its stored in the database.

/Gemfile

```
# To use ActiveRecord has_secure_password
gem 'bcrypt-ruby', '~> 3.0.0'
```

We need to run `bundle` to ensure that this gem is installed and to restart the server for the changes to be picked up.

Another important change to make in the User model is to call `attr_accessible` and specify the attributes that can be set by mass assignment and through the form. We'll set `email`, `password` and `password_confirmation` but you can set whatever fields you want to match your signup form.

`/app/models/user.rb`

```
class User < ActiveRecord::Base
  has_secure_password

  attr_accessible :email, :password, :password_confirmation

  validates_uniqueness_of :email
end
```

Having this in place means that if we add an `admin` column to the database it won't be possible for a malicious user to make themselves an admin user by sending a request including that value to the `UsersController`'s `update` action. We've also added a validation to the model to ensure that the email address is unique and we could add other validations here to validate the format of the email, the length of the password and so on. We don't need to validate the presence of the `password` and `password_confirmation` fields, however, as this will be handled by `has_secure_password`.

Our User model is looking good so let's move on to creating the signup form. We'll do this inside the `UsersController` that we generated earlier. This is blank by default so we'll need to add a couple of actions to handle creating new users. The code for these is fairly standard.

`/app/controllers/users_controller.rb`

```
class UsersController < ApplicationController
  def new
    @user = User.new
  end

  def create
    @user = User.new(params[:user])
    if @user.save
      redirect_to root_url, notice: "Thank you for signing up!"
    else
      render "new"
    end
  end
end
```

The `new` action will display the signup form and the `create` action is called when the form is submitted. This will validate the values from the form and if this passes it will redirect the browser back to the home page and show a message. If the validation fails the signup form will be redisplayed.

The signup form is shown below. Again, this is fairly standard. It uses `form_for` with the `@user` instance variable, has a section for displaying any errors and finally three fields for `email`, `password` and `password_confirmation`.

`/app/views/users/new.html.erb`

```
<h1>Sign Up</h1>

<%= form_for @user do |f| %>
  <% if @user.errors.any? %>
    <div class="error_messages">
      <h2>Form is invalid</h2>
      <ul>
        <% @user.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
</form>
```

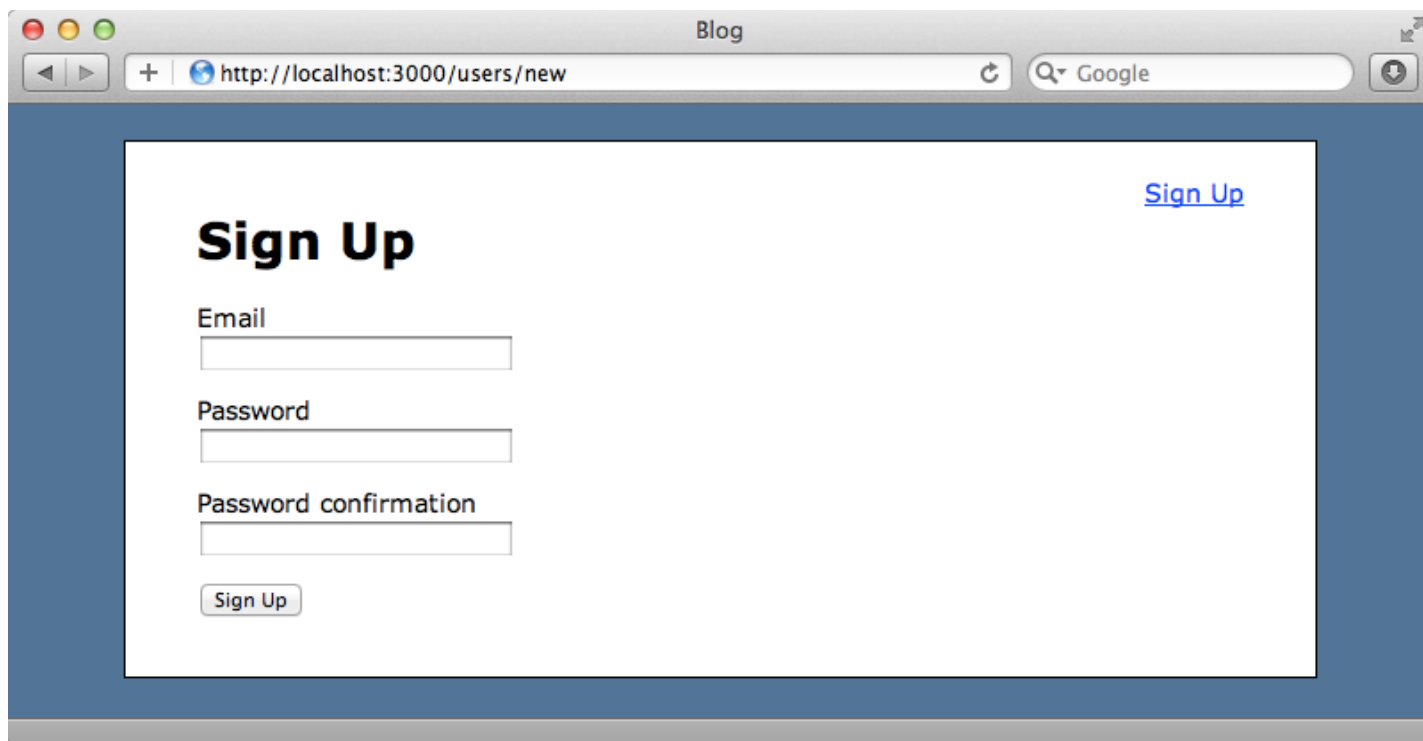
```
<div class="field">
  <%= f.label :email %><br />
  <%= f.text_field :email %>
</div>
<div class="field">
  <%= f.label :password %><br />
  <%= f.password_field :password %>
</div>
<div class="field">
  <%= f.label :password_confirmation %><br />
  <%= f.password_field :password_confirmation %>
</div>
<div class="actions"><%= f.submit "Sign Up" %></div>
<% end %>
```

Now all we need is a link to this form. We'll put it in the application's layout file.

/app/views/layouts/application.html.erb

```
<div id="user_header">
  <%= link_to "Sign Up", new_user_path %>
</div>
```

When we reload the home page now the "Sign Up" link is visible and clicking it will take us to our new form.

A screenshot of a web browser window titled "Blog". The address bar shows "http://localhost:3000/users/new". The page has a blue header and footer. The main content area is white and contains a "Sign Up" form. The form has a title "Sign Up" in large bold letters. Below the title are three input fields labeled "Email", "Password", and "Password confirmation". At the bottom of the form is a "Sign Up" button. In the top right corner of the white area, there is a blue link labeled "Sign Up".

If we fill in this form incorrectly we'll see validation errors as we'd expect. Once we've filled it in successfully we'll be redirected back to the home page and we'll see a flash message that tells us we've signed up.



Logging In And Out

So far we haven't done any actual authentication, we've just created a User record. For authentication we need a login form. We'll create that now and also add a "Log In" link next to the "Sign Up" one. First we'll create a new SessionsController to handle logging in and give it a new action.

terminal

```
$ rails g controller sessions new
```

We'll need to adjust the routes that were generated by this command. The SessionsController is a RESTful-style controller so we'll delete the generated get "sessions/new" route and add a new sessions resource.

/config/routes.rb

```
Blog::Application.routes.draw do
```

```
  resources :sessions
  resources :users
```

```
  root to: 'articles#index'
  resources :articles
end
```

We can leave the new action blank in the controller but we will modify the associated view as this is where the login form will go.

/app/views/sessions/new.html.erb

```
<h1>Log In</h1>

<%= form_tag sessions_path do %>
  <div class="field">
    <%= label_tag :email %><br />
    <%= text_field_tag :email, params[:email] %>
  </div>
  <div class="field">
    <%= label_tag :password %><br />
    <%= password_field_tag :password %>
  </div>
  <div class="actions"><%= submit_tag "Log In" %>
<% end %>
```

As our `SessionsController` doesn't have a model behind it we use `form_tag` for this form. It POSTs to the `sessions_path` which will trigger the `SessionsController`'s `create` action and we'll write that action next.

`/app/controllers/sessions_controller.rb`

```
def create
  user = User.find_by_email(params[:email])
  if user && user.authenticate(params[:password])
    session[:user_id] = user.id
    redirect_to root_url, notice: "Logged in!"
  else
    flash.now.alert = "Email or password is invalid."
  end
end
```

When the form is submitted we try to find a user by the email address that was entered on the form. If we find one we use `authenticate` to check that the entered password is correct. This is a method that `has_secure_password` provides and will return a boolean value. If the password matches we store the user's id in a session variable and redirect to the home page. If either the username or password are incorrect the form is shown again with an error message. We use `flash.now` here as we need the message to be displayed immediately, not after a redirect.

To finish this feature we just need to add a link next to the "Sign Up" link.

`/app/views/layouts/application.html.erb`

```
<div id="user_header">
  <%= link_to "Sign Up", new_user_path %> or
  <%= link_to "Log In", new_session_path %>
</div>
```

If we reload the home page now we'll see the new "Log In" link and if we click it we'll be taken to our new form. If we use the email address and password we entered when we signed up we'll be able to log in to the site.



Even though we've logged into the site now the "Log In" link still shows. It would be better if this changed to show that the user is currently logged in. To do this we need a way to fetch the currently logged-in user record and we'll do this in the `ApplicationController` so that it's available in all controllers.

`/app/controllers/application_controller.rb`

```
class ApplicationController < ActionController::Base
  protect_from_forgery
end
```

```

private
def current_user
  @current_user ||= User.find(session[:user_id]) if session[:user_id]
end
helper_method :current_user
end

```

This current user will look for a User based on the session's user id that was stored when they logged in, if that session variable exists. This method will probably be called many times per request and so it's a good idea to cache it in an instance variable as we have done here. We're going to need to access this method in the views, too, so we've used `helper_method` to make it a helper method. We can use this now to change the "Sign Up" and "Log In" links when the user is logged in and display a message telling the user that they're logged in and showing their email address.

/app/views/layouts/application.html.erb

```

<div id="user_header">
  <% if current_user %>
    Logged in as <%= current_user.email %>.
  <% else %>
    <%= link_to "Sign Up", new_user_path %> or
    <%= link_to "Log In", new_session_path %>
  <% end %>
</div>

```

As we're logged in reloading the page now will display the email address we signed up with. If your authentication setup uses usernames instead you could display that here just as easily.



Logging Out

While we can log in now there's no way to log out. We need a new link that shows when we're signed in and we'll need a new action in the SessionsController to handle that behaviour.

/app/controllers/sessions_controller.rb

```

def destroy
  session[:user_id] = nil
  redirect_to root_url, notice: "Logged out!"
end

```

This action is pretty simple. We just clear out the `user_id` session variable and redirect back to the home page. We could clear the entire session by calling `reset_session` instead, but clearing the `user_id` is enough to sign the user out. Now in

the layout file we can add a “Log Out” link. Pointing it to the `destroy` action can be a little bit tricky because the `session_path` helper method expects an `id` to be passed in. We’ll just pass “current” in but we could pass in anything as the controller doesn’t read this parameter. We also need to set the method to `delete` so that the destroy action is triggered.

`/app/views/layouts/application.html.erb`

```
<div id="user_header">
  <% if current_user %>
    Logged in as <%= current_user.email %>.
    <%= link_to "Log Out", session_path("current"), method: "delete" %>
  <% else %>
    <%= link_to "Sign Up", new_user_path %> or
    <%= link_to "Log In", new_session_path %>
  <% end %>
</div>
```

Reloading the page will now show us the “Log Out” link and when we click it we’ll be logged out.



Better Routes

We could make the “Sign Up” and “Log In” URLs a little prettier. The login form is currently at `/sessions/new` but it would better to have `/login` point to this. We can add a three custom routes to improve the URLs for signing in and logging in and out.

`/config/routes.rb`

`Blog::Application.routes.draw do`

```
  get 'signup', to: 'users#new', as: 'signup'
  get 'login', to: 'sessions#new', as: 'login'
  get 'logout', to: 'sessions#destroy', as: 'logout'

  resources :sessions
  resources :users

  root to: 'articles#index'
  resources :articles
end
```

One thing to note about the `logout` path is that we’re using `get`. We might consider using `delete` instead as technically this action does change user state but as it doesn’t do any permanent damage we’ll leave this as `get` here. Now we can update our layout file to use these new routes.


```
/app/views/layouts/application.html.erb
```

```
<div id="container">
  <div id="user_header">
    <% if current_user %>
      Logged in as <%= current_user.email %>.
      <%= link_to "Log Out", logout_path %>
    <% else %>
      <%= link_to "Sign Up", signup_path %> or
      <%= link_to "Log In", login_path %>
    <% end %>
  </div>
```

Automatically Logging In New Users

One loose end that we've left untied is that when the user signs up they aren't automatically logged in. The user record is created but the new user has to enter their details again to sign in. This is easy to fix by modifying the UsersController's create action and setting the user_id session variable when the new user is saved.

```
/app/controllers/users_controller.rb
```

```
def create
  @user = User.new(params[:user])
  if @user.save
    session[:user_id] = @user.id
    redirect_to root_url, notice: "Thank you for signing up!"
  else
    render "new"
  end
end
```

If we find that there's duplication in the logic between the SessionsController's logging in behaviour and this behaviour we can abstract this out into some kind of controller method which can be shared between them. Here it's OK to have this small amount of duplication. If we sign up as a new user now we'll be signed in straight away.



Basic Authorization

A common requirement when dealing with authentication is to make sure that they're logged in before giving them access to a specific page. Our application shows a number of articles and each article's page has an "Edit" link which takes us to a page where we can edit that article. We want our application to only allow logged-in users to edit articles. Since this is such common behaviour we'll define it in the ApplicationController so that we can use it anywhere.

```
/app/controllers/application_controller.rb
```

```
def authorize
  redirect_to login_url, alert: "Not authorized" if current_user.nil?
end
```

This method will check for a current user and redirect to the login page if it fails to find one with an alert telling them that they aren't authorized. We can now use this method as a `before_filter` on any controller action we want to protect, such as the `ArticlesController`'s `edit` and `update` actions.

```
/app/controllers/articles_controller.rb
```

```
class ArticlesController < ApplicationController

  before_filter :authorize, only: [:edit, :update]

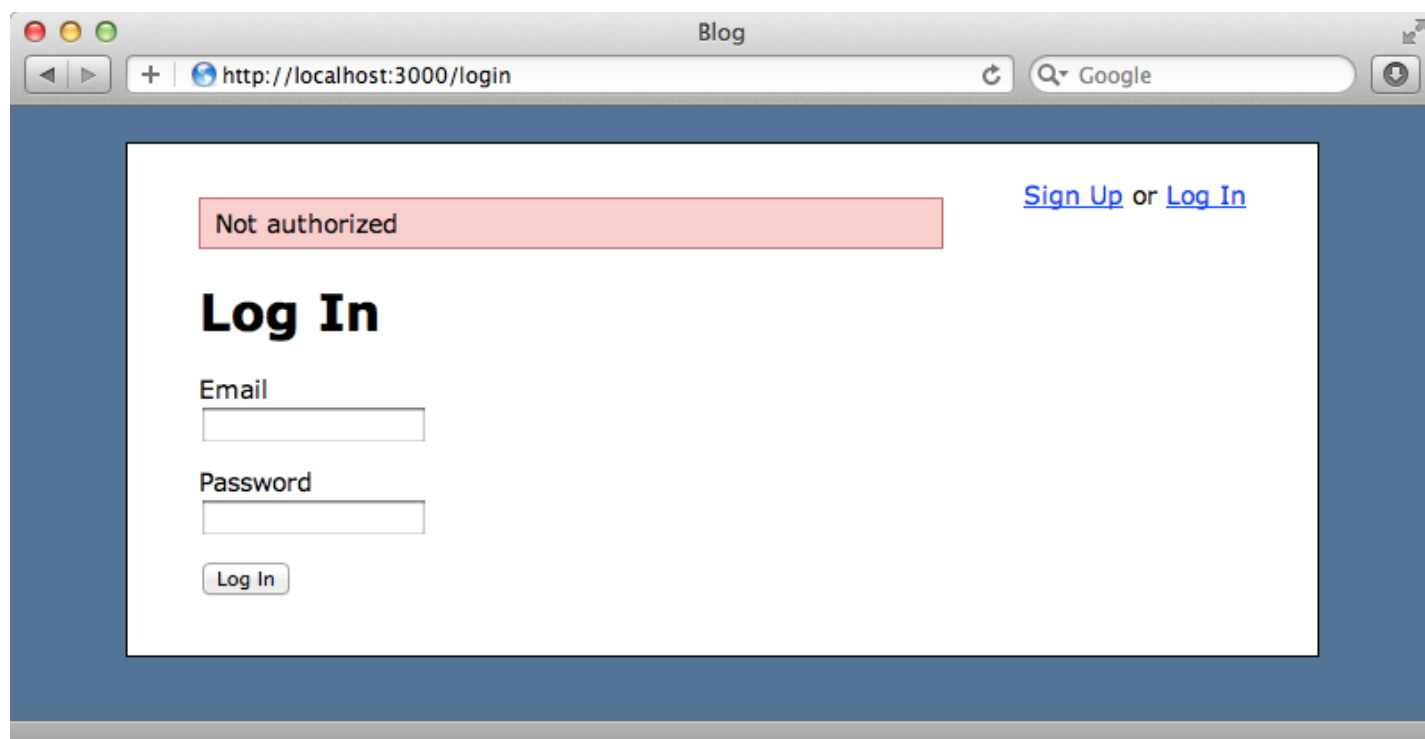
  def index
    @articles = Article.all
  end

  def show
    @article = Article.find(params[:id])
  end

  def edit
    @article = Article.find(params[:id])
  end

  def update
    @article = Article.find(params[:id])
    if @article.update_attributes(params[:article])
      redirect_to @article, notice: "Article has been updated."
    else
      render "edit"
    end
  end
end
```

We can still edit an article now when we're logged in but if we log out and try again we'll be redirected.



You could improve this user experience depending on how your application works, but the basic functionality is here. If you have more complex authorization logic than this you could use a gem like [CanCan](#) to help with this.

Now we have a complete authentication solution built from scratch and there are a variety of ways we could add on to this.

For example in UsersController we might want a profile page or a way to update a user's information. We could do this easily by adding show or edit actions to this controller. If we want to store the user's login in a permanent cookie instead of a temporary session take a look at [episode 274](#) which shows how to add "Remember Me" functionality and also a way for users to reset their password.

©2012 RailsCasts - [Privacy Policy](#) - Hosted by [Linode](#)