How To: Allow users to sign in using their username or email address

Edit New Page

Katherine G. Pe edited this page 15 days ago · 12 revisions

Allow users to Sign In using their username or email address

For this example, we will assume your model is called `User`

Create a username field in the `users` table

Create a migration:

```
rails generate migration add_username_to_users username:string:uniq
```

Run the migration:

```
rake db:migrate
```

Rails 3:

Modify the `User` model and add username, email, password, password confirmation and remember me to `attr_accessible`

```
attr_accessible :username, :email, :password, :password_confirmation, :reme
```

Rails 4:

Modify `application_controller.rb` and add username, email, password, password confirmation and remember me to `configure_permitted_parameters`

```
class ApplicationController < ActionController::Base
  before_action :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer.for(:sign_up) { |u| u.permit(:username, :email,
    devise_parameter_sanitizer.for(:sign_in) { |u| u.permit(:login, :username,
    devise_parameter_sanitizer.for(:account_update) { |u| u.permit(:username,
  end
end
```

> see also "strong parameters"

Create a login virtual attribute in the `User` model

Add login as an attr_accessor

```
# Virtual attribute for authenticating by either username or email
# This is in addition to a real persisted field like 'username'
attr_accessor :login
```

Rails 3: Also add login to attr_accessible

Clone this wiki locally

```
https://github.com/platafc
```

```
attr_accessible :login
```

Rails 4: same as the first one

```
attr_accessor :login
```

or if you will use this variable somwhere else in the code:

```
def login=(login)
  @login = login
end

def login
  @login || self.username || self.email
end
```

Tell Devise to use :username in the authentication_keys

Modify config/initializers/devise.rb to have:

```
 config.authentication_keys = [ :login ]
```

If you are using multiple models with Devise, it is best to set the authentication_keys on the model itself if the keys may differ:

```
devise :database_authenticatable, :registerable,
       :recoverable, :rememberable, :trackable,
       :validatable, :authentication_keys => [:username]
```

Overwrite Devise's `find_for_database_authentication` method in `User` model

Because we want to change the behavior of the login action, we have to overwrite the `find_for_database_authentication` method. The method's stack works like this: `find_for_database_authentication` calls `find_for_authentication` which calls `find_first_by_auth_conditions`. Overriding the `find_for_database_authentication` method allows you to edit database authentication; overriding `find_for_authentication` allows you to redefine authentication at a specific point (such as token, LDAP or database). Finally, if you override the `find_first_by_auth_conditions` method, you can customize finder methods (such as authentication, account unlocking or password recovery).

For ActiveRecord:

MySQL users: the use of the SQL `lower` function below is most likely unnecessary and will cause any index have on the `email` column to be ignored.

```
# app/models/user.rb

    def self.find_for_database_authentication(warden_conditions)
      conditions = warden_conditions.dup
      if login = conditions.delete(:login)
        where(conditions).where(["lower(username) = :value OR lower(email) = :
      else
        where(conditions).first
      end
    end
```

Be sure to add case **insensitivity** to your validations on `:username`:

```
# app/models/user.rb

validates :username,
  :uniqueness => {
    :case_sensitive => false
  },
  :format => { ... } # etc.
```

Alternatively, change the find conditions like so:

```
  # when allowing distinct User records with, e.g., "username" and "UserName".
  where(conditions).where(["username = :value OR lower(email) = lower(:value)"
```

For Mongoid:

Note: This code for Mongoid does some small things differently than the ActiveRecord code above. Would be great if someone could port the complete functionality of the ActiveRecord code over to Mongoid [basically you need to port the 'where(conditions)']. It is not required but will allow greater flexibility.

```
  field :email

  def self.find_first_by_auth_conditions(warden_conditions)
    conditions = warden_conditions.dup
    if login = conditions.delete(:login)
      self.any_of({ :username =>  /^#{Regexp.escape(login)}$/i }, { :email =>
    else
      super
    end
  end
```

The code below also supports Mongoid but uses the where method and the OR operator to choose between username and email.

```
  # function to handle user's login via email or username
  def self.find_for_database_authentication(warden_conditions)
    conditions = warden_conditions.dup
    if login = conditions.delete(:login).downcase
      where(conditions).where('$or' => [ {:username => /^#{Regexp.escape(login
    else
      where(conditions).first
    end
  end
```

Update your views

Make sure you have the Devise views in your project so that you can customize them

Rails 3 & 4:

```
rails g devise:views
```

Rails 2:

```
script/generate devise_views
```

Modify the views

sessions/new.html.erb:

```
  -  <p><%= f.label :email %><br />
  -  <%= f.email_field :email %></p>
```

```
+  <p><%= f.label :login %><br />
+  <%= f.text_field :login %></p>
```

registrations/new.html.erb:

```
+  <p><%= f.label :username %><br />
+  <%= f.text_field :username %></p>
   <p><%= f.label :email %><br />
   <%= f.email_field :email %></p>
```

registrations/edit.html.erb

```
+  <p><%= f.label :username %><br />
+  <%= f.text_field :username %></p>
   <p><%= f.label :email %><br />
   <%= f.email_field :email %></p>
```

Manipulate the `:login` label that Rails will display

Rails 3 & 4 (config/locales/devise.en.yml)

change

```
   invalid: "Invalid email or password."
   ...
   not_found_in_database: "Invalid email or password."
```

to

```
   invalid: "Invalid login or password."
   ...
   not_found_in_database: "Invalid login or password."
```

Allow users to recover their password or confirm their account using either username or email address

This section assumes you have run through the steps in *Allow users to Sign In using their username or email*.

Configure Devise to use login as reset password or confirmation keys:

Simply modify config/initializers/devise.rb to have:

```
 config.reset_password_keys = [ :username ]
 config.confirmation_keys = [ :username ]
```

Use `find_first_by_auth_conditions` instead of `find_for_database_authentication`

Replace (in your `Users.rb`):

```
def self.find_for_database_authentication(warden_conditions)
  conditions = warden_conditions.dup
  if login = conditions.delete(:login)
    where(conditions).where(["lower(username) = :value OR lower(email) = :valu
  else
    where(conditions).first
  end
end
```

with:

```
def self.find_first_by_auth_conditions(warden_conditions)
  conditions = warden_conditions.dup
  if login = conditions.delete(:login)
    where(conditions).where(["lower(username) = :value OR lower(email) = :valu
  else
    where(conditions).first
  end
end
```

Update your views

passwords/new.html.erb:

```
  -  <p><%= f.label :email %><br />
  -  <%= f.email_field :email %></p>
  +  <p><%= f.label :username %><br />
  +  <%= f.text_field :username %></p>
```

confirmations/new.html.erb:

```
  -  <p><%= f.label :email %><br />
  -  <%= f.email_field :email %></p>
  +  <p><%= f.label :username %><br />
  +  <%= f.text_field :username %></p>
```

Gmail or me.com Style

Another way to do this is me.com and gmail style. You allow an email or the username of the email. For public facing accounts, this has more security. Rather than allow some hacker to enter a username and then just guess the password, they would have no clue what the user's email is. Just to make it easier on the user for logging in, allow a short form of their email to be used e.g "someone@domain.com" or just "someone" for short.

```
before_create :create_login

  def create_login
    email = self.email.split(/@/)
    login_taken = User.where( :login => email[0]).first
    unless login_taken
      self.login = email[0]
    else
      self.login = self.email
    end
  end

  # You might want to use the self.find_first_by_auth_conditions(warden_condit
  # instead of using this find_for_database_authentication as this one causes
  # def self.find_for_database_authentication(conditions)
  #   self.where(:login => conditions[:email]).first || self.where(:email => c
  # end
```