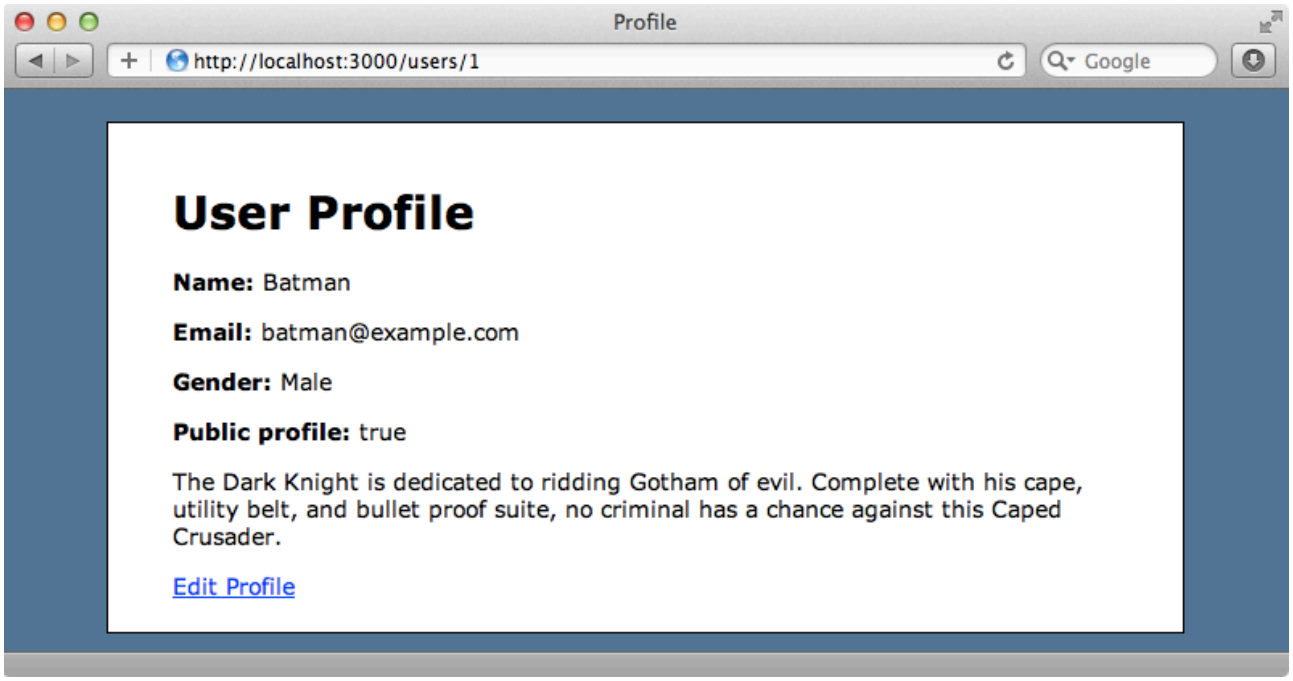
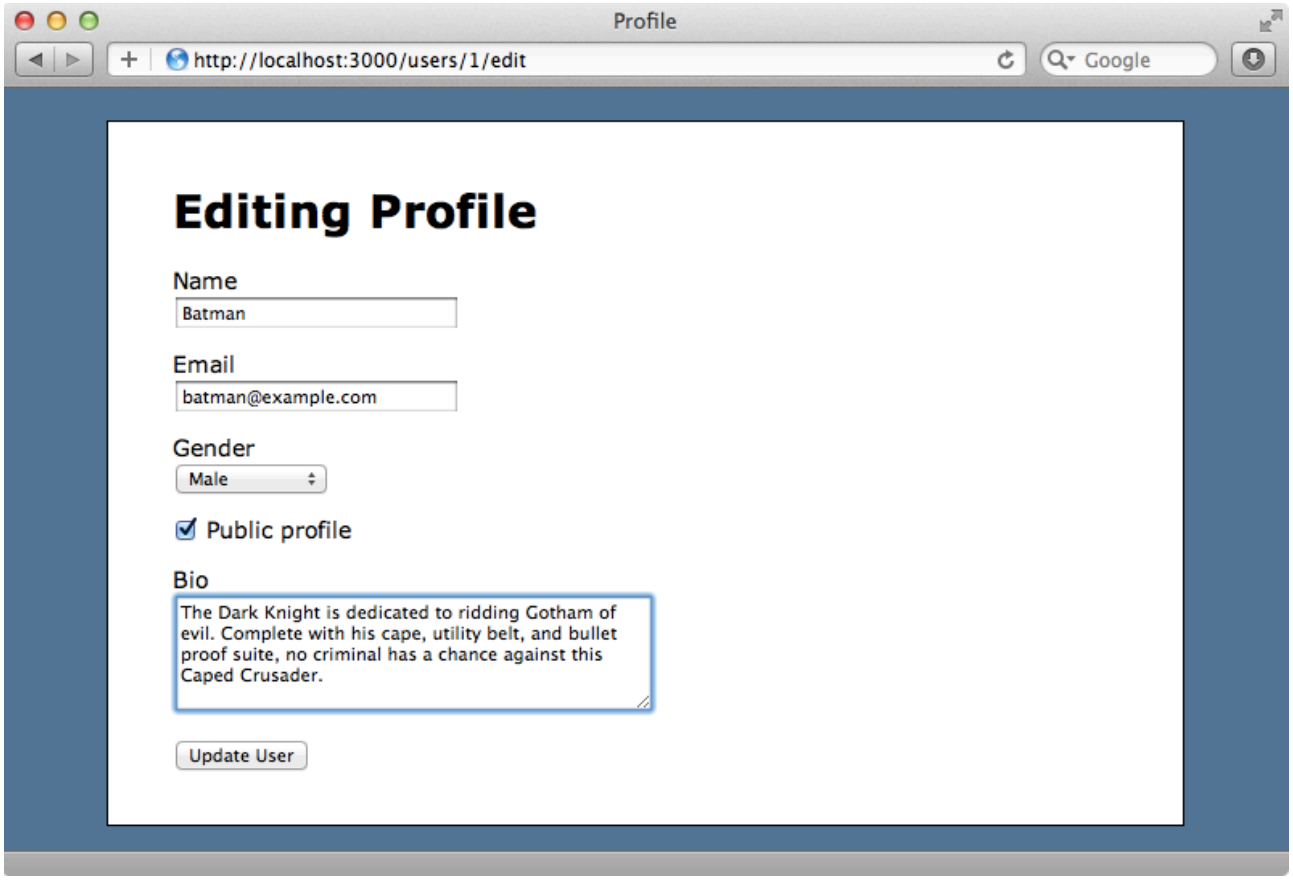
Episode 302

In-place Editing



Above is a user profile page from a Rails application. There's an "Edit Profile" link at the bottom of the page and clicking this link will take us to a page where we can edit our details.



Instead using a separate editing page we want to give users the ability to edit the fields directly inline on the profile page by clicking on any field. Doing this should make the field editable and hitting Enter or tabbing out of the field should save any changes made to the database.

Best In Place

There are a number of Rails plugins that can help us to add in-place editing to our application and there's a comprehensive list of them at The Ruby Toolbox[1]. The one we'll be using is called Best In Place[2], although the others are worth talking a look at. Best In Place is a fork of another project called Rest in Place[3] and what makes it worth using is the `best_in_place` helper method it offers. This makes it easy to add in-place editing fields in our Rails applications.

Best In Place has a demo page[4] and if we try it we'll see that we can edit any field by clicking it. Doing this replaces the static text with a form field appropriate for that type of data that's being edited. When we hit enter for click out of the field the changes are sent back to the server and the database is updated. Best in place supports validations so if we enter, say, an invalid email address we'll see an error message and the entered value will revert back to the last valid value. It also supports different data types and so can show a dropdown menu for editing an association or toggle between two values for a boolean field. Let's see what's involved in adding it to our profile page.

Adding Best In Place to Our Application

To add Best in Place to our app we first need to add its gem to our application's Gemfile and run `bundle`.

---

[1] https://www.ruby-toolbox.com/categories/rails_in_place_editing

[2] https://github.com/bernat/best_in_place

[3] https://github.com/janv/rest_in_place

[4] http://bipapp.heroku.com/

/Gemfile

```
source 'http://rubygems.org'

gem 'rails', '3.1.3'

gem 'sqlite3'

# Gems used only for assets and not required
# in production environments by default.
group :assets do
  gem 'sass-rails',   '~> 3.1.5'
  gem 'coffee-rails', '~> 3.1.1'
  gem 'uglifier', '>= 1.0.3'
end

gem 'jquery-rails'
gem 'best_in_place'
```

The gem includes two JavaScript files that we'll need to include in our application, `jquery.purr`, a jQuery plugin, and `best_in_place`. Since this is a Rails 3.1 app we can add them to the JavaScript manifest file.

/app/assets/javascripts/application.js

```
//= require jquery
//= require jquery_ujs
//= require jquery.purr
//= require best_in_place
//= require_tree .
```

We need to define the fields that will be editable and we'll do this in the `users` CoffeeScript file. All we have to do is call `best_in_place()` on any elements we want to be editable. We'll apply it to elements with the class `best_in_place` which is what Best In Place uses internally.

/app/assets/javascripts/users.js.coffee

```
jQuery ->
  $('.best_in_place').best_in_place()
```

Now that we have Best In Place set up we can start to apply it to our User Profile page. Here's how the page's template currently looks:

/app/views/users/show.html.erb

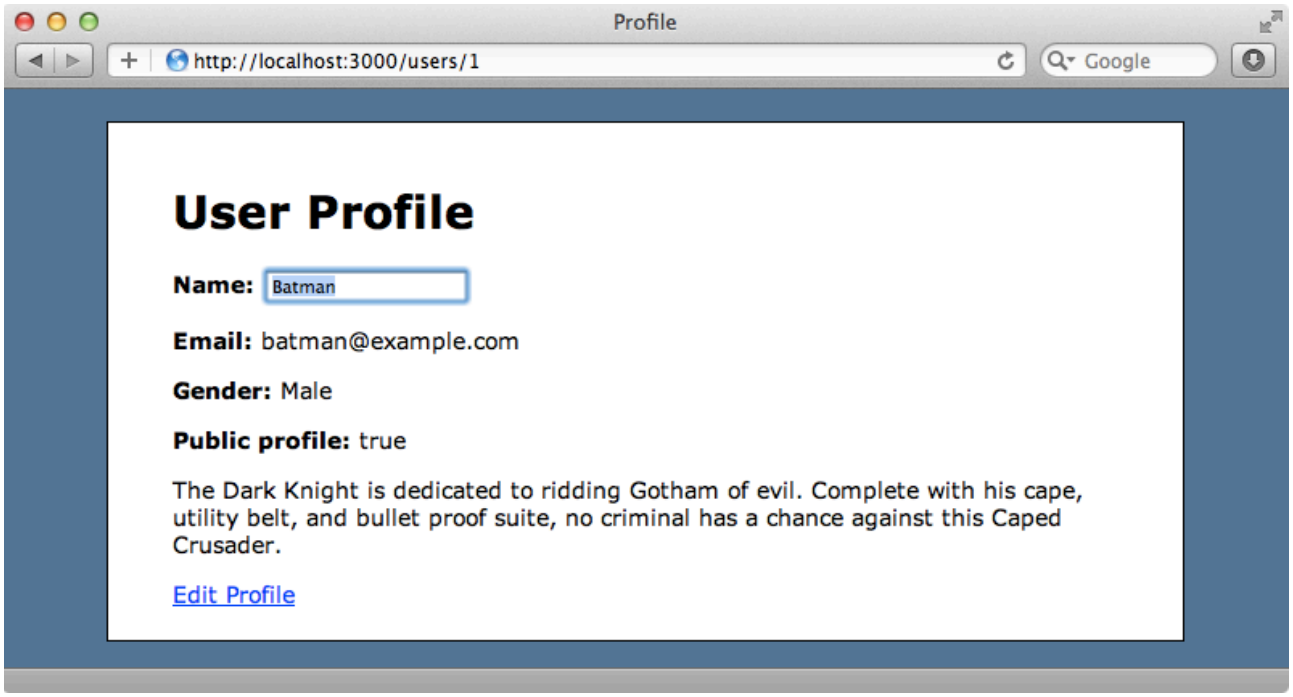
```
<h1>User Profile</h1>

<p>
  <b>Name:</b>
  <%= @user.name %>
</p>
<p>
  <b>Email:</b>
  <%= @user.email %>
</p>
<p>
  <b>Gender:</b>
  <%= @user.gender %>
</p>
<p>
  <b>Public profile:</b>
  <%= @user.public_profile %>
</p>
<p>
  <%= @user.bio %>
</p>

<%= link_to 'Edit Profile', edit_user_path(@user) %>
```

The template simply outputs the value for each field. We'll add Best In Place's helper method to the `name` and `email` fields. This method takes two arguments, the object we're viewing and a symbol for the field we want to edit.

/app/views/users/show.html.erb

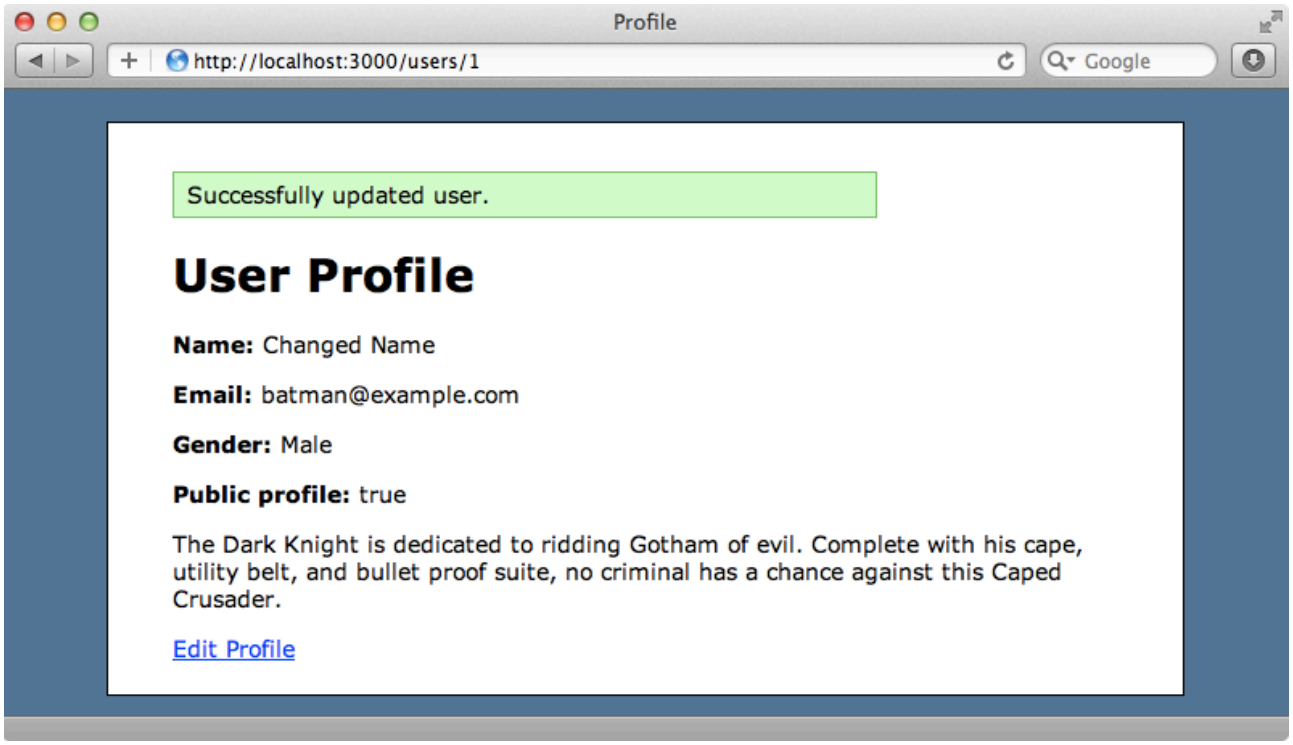
```
<p>
  <b>Name:</b>
  <%= best_in_place @user, :name %>
</p>
<p>
  <b>Email:</b>
  <%= best_in_place @user, :email %>
</p>
```

When we load the user profile page now and click on either the `name` or `email` field that field becomes editable.



Storing Changes

When we make a change a field then click away from it the field reverts to its previous value, which isn't what we want to happen. If we then reload the page, though, we'll see a message telling us that the user has been updated and the page shows the changed value.



This is happening because Best In Place uses JSON to send updates back to the server but our `UsersController` isn't set to respond to JSON requests correctly. This controller uses the usual seven RESTful action, which Best In Place expects, and when we change value in the `name` field the `update` action is triggered. The correct parameters are sent to the action but it doesn't respond to JSON requests as it should.

To fix this we could add a `respond_to` block in `update` or, as this is a Rails 3 application, use `respond_with` which will handle everything for us. This expects a `respond_to` call at the top of the controller so we'll need to add that as well.

/app/controllers/users_controller.rb

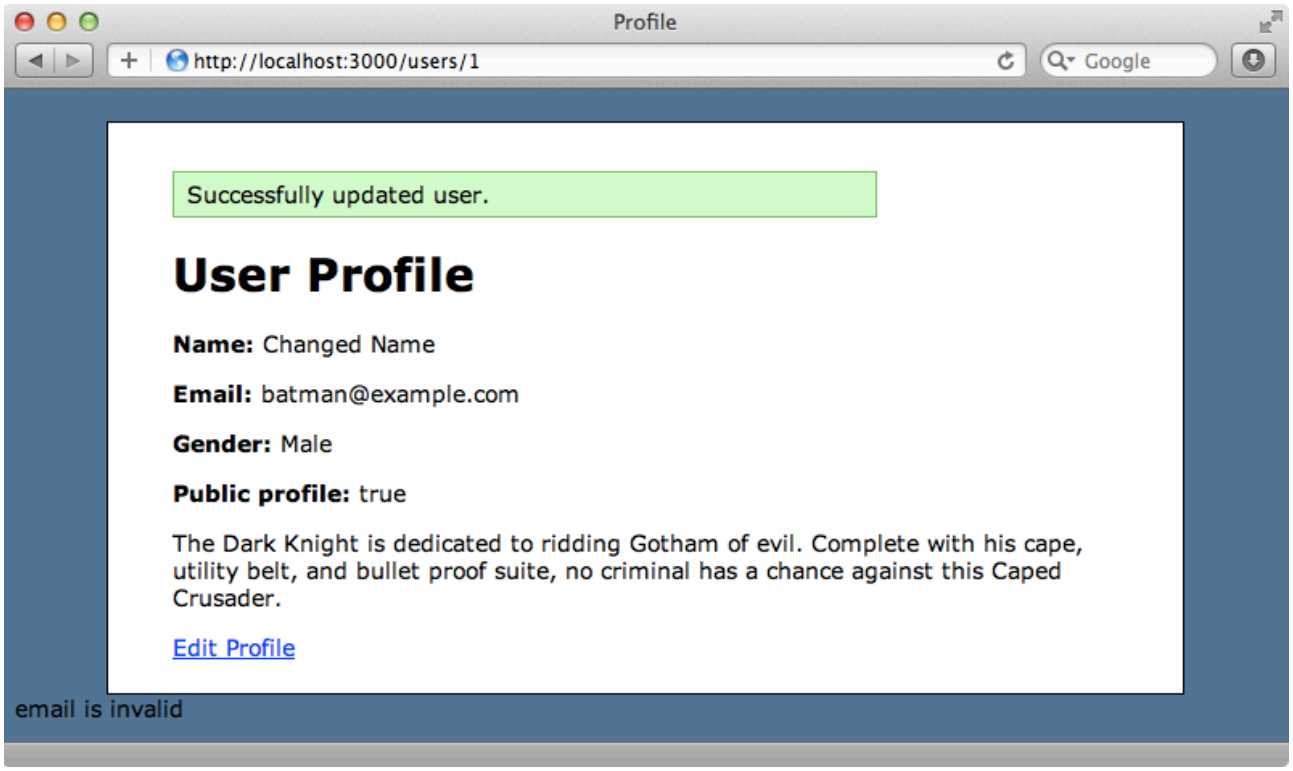
```
class UsersController < ApplicationController
  respond_to :html, :json

  # Other actions omitted.

  def update
    @user = User.find(params[:id])
    @user.update_attributes(params[:user])
    respond_with @user
  end

end
```

If we need to customize this behaviour further we can use a full `respond_to` block in the `update` action and there's an example of how to do this in Best in Place's README[5].

Now that our controller responds to JSON the in-place functionality works correctly. Any changes we make are added to the database and the UI updates as we expect.

Validations also work, too. If we enter an invalid email address an error message will show, although the by default it isn't particularly visible.

[5] https://github.com/bernat/best_in_place/blob/master/README.md

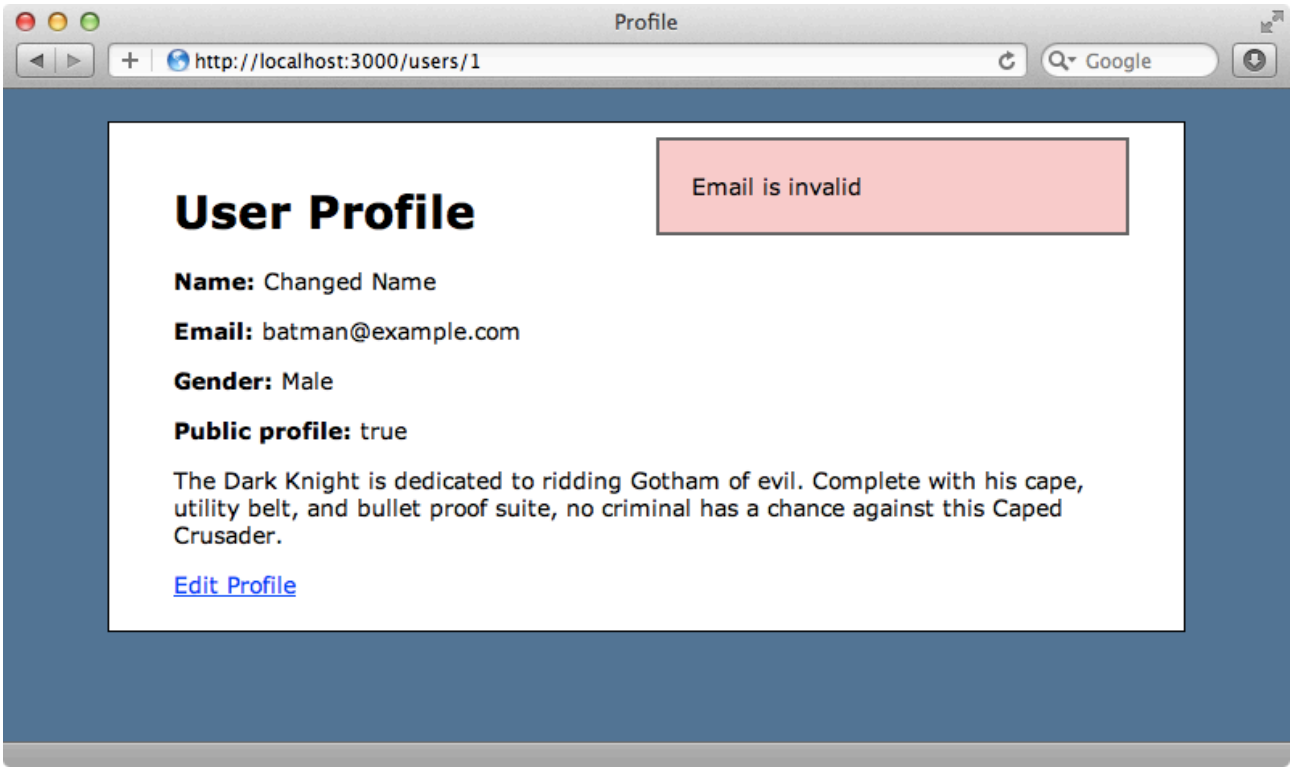


We can pretty this error message up with some CSS.

/app/assets/stylesheets/users.css.scss

```
.purr {
  position: fixed;
  top: 30px;
  right: 100px;
  width: 250px;
  padding: 20px;
  background-color: #FCC;
  border: solid 2px #666;
  &:first-letter { text-transform: uppercase };
}
```

All of the styles are in a `purr` class which is what Best in Place uses. When we reload the page now and enter a bad email address again we get a prettier error message.



Handling Other Types of Data

We've only made the `name` and `email` fields editable so far, now we'll look at the other attributes on the user page. The `bio` field contains a long piece of text but by default `best_in_place` will only show a single line text box. We can change this behaviour by specifying the `type` attribute.

/app/views/users/show.html.erb

```
<%= best_in_place @user, :bio, type: :textarea %>
```

The `public_profile` field is boolean so we should use the `checkbox` type and pass in the values that should be displayed in a `:collection` option.

/app/views/users/show.html.erb

```
  <%= best_in_place @user, :public_profile, type: :checkbox,
collection: %w[No Yes] %>
```

Clicking on the current public profile option now will toggle between these two options. The `gender` option can be treated in a similar way, but here we use `:select` instead of `:checkbox` and pass in a two-dimensional array of options.

/app/views/users/show.html.erb

```
  <%= best_in_place @user, :gender, type: :select, :collection
[["Male", "Male"], ["Female", "Female"], ["", "Unspecified"]] %>
```

This option will show a dropdown menu when we click the current gender.

That's is for this episode. We now have a fully-featured in-place editing option for each field in our User Profile page. There may be times when we want to do something a little more complicated, for instance displaying options for an association or maybe a formatted price field where the value displayed may be different from the one in the edit field. Unfortunately this is a little difficult to do with the current version of Best In Place. For these situations it's still best to fall back to a 'traditional' edit form or write your own solution from scratch.