

Temario

Unidad 2

a) Autenticación con Devise

Temas: Agregar autenticación con Devise. Leyendo la documentación. Realizando una integración acorde a la documentación. Devise y métodos helpers: `current_user`, `user_signed_in?`.

b) Redireccionando en función de si un usuario ha ingresado

Temas: Redireccionando en función de si un usuario ha ingresado: helper: `authenticate_user!`. Asociando recursos al usuario que ha iniciado sesión.

c) Personalizar las vistas de devise

Temas: Agregar campos al formulario; Strong params con devise. Recuperación de las claves. Pruebas unitarias automatizadas de acceso con Devise.

d) Usuarios y roles

Temas: Limitar accesos y funcionalidades al rol del usuario.

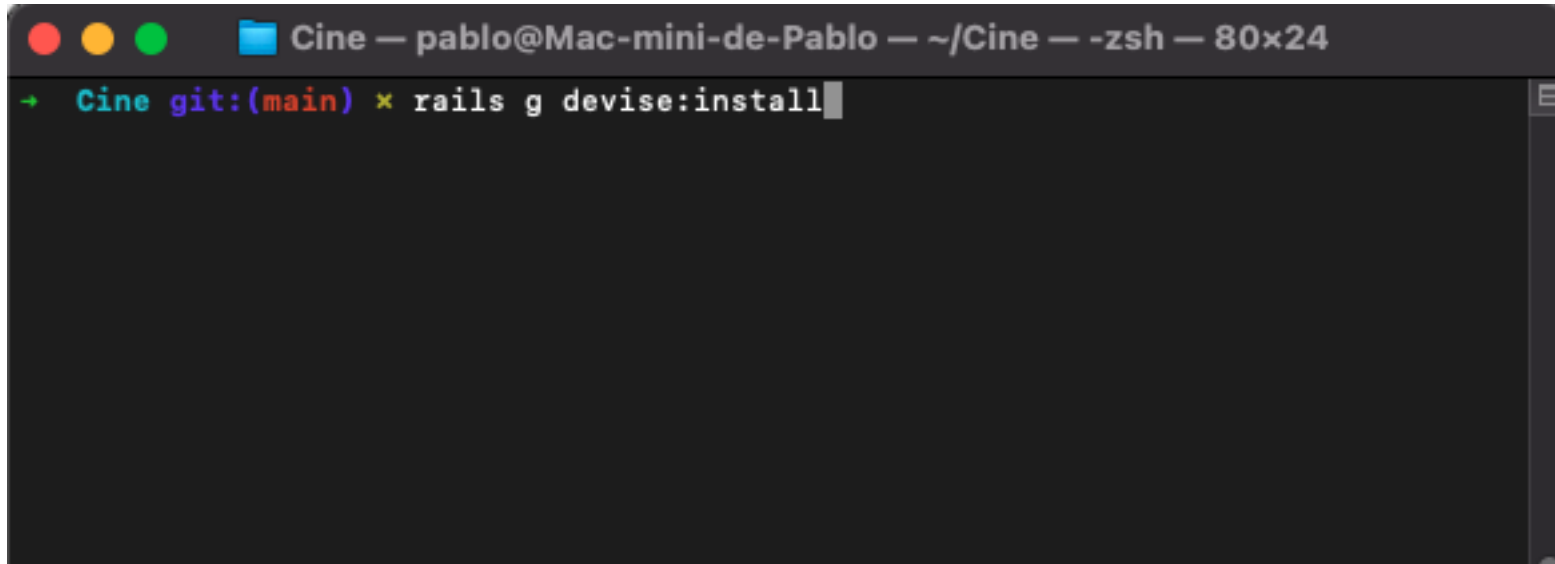
Unidad 2

Autenticación con Devise

Autenticación con devise

Devise es una **gema flexible de autenticación**, fuertemente valorado en la industria, con el podemos crear registros para cada cosa. Y editar los modelos, pasarle **Strong Params**, editar las **vistas**, agregarle **sanitizers**, editar las **rutras de devise**, **generar los controllers de devise y manipularlo**, en fin, una infinidad de cosas maravillosas se pueden hacer con esta gema, **muchas páginas las usan hoy en día, en producción.**

Generando con devise



```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine — zsh — 80x24
→ Cine git:(main) ✖ rails g devise:install
```

Preparamos todo con el generador devise, al finalizar nos mostrará un dialogo

Agregando configuración de devise a development

```
config > environments >  development.rb
```

```
6 # In the development environment your application's code is reloaded any time
7 # it changes. This slows down response time but is perfect for development
8 # since you don't have to restart the web server when you make code changes.
9
10 config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
11
12
```

Esto nos asegura que en el ambiente de desarrollo la dirección en el cual vamos a iniciar la aplicación para usar devise, en este caso es **localhost** con el puerto **3000**

En producción solo se requeriría con el el símbolo **:host**

Añadiendo root_path a routes

```
config > routes.rb
1  Rails.application.routes.draw do
2    root to: "home#index"
3  end
4
```

Muy importante saber que para que funcione tu aplicación y redireccione a esa ruta debemos crear un **controlador** llamado **Home** con la **acción index** o **método index**. Para esto simplemente debes usar el siguiente comando:

rails generate controller Home index una vez hecho este procedimiento lo editas a **root to: "home#index"** como en el ejemplo

Mensajes flash, Notice y Alert

```
app > views > layouts > application.html.erb
12
13 <body>
14   <div id="notice_wrapper">
15     <% if notice %>
16       <p class="notice"><%= notice %></p>
17     <% elsif alert %>
18       <p class="alert"><%= alert %></p>
19     <% end %>
20   </div>
21   <%= yield %>
22 </body>
23 </html>
24
```

Devise nos genera unas variables de mensajes **flash** con un corto tiempo de vida, llamado **Notice y Alert** con ello podemos Decirle al usuario que una acción o más se ha creado exitosamente o lo contrario

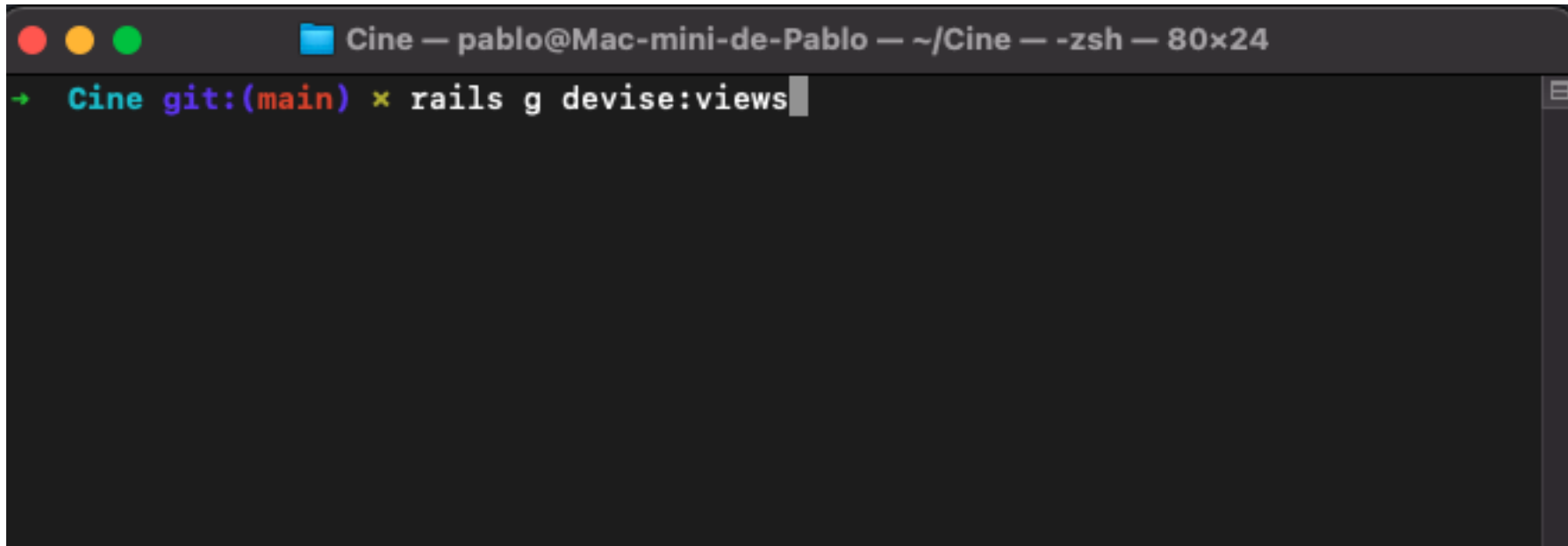
Generando el modelo User con devise



```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine — -zsh — 79x18
+ Cine git:(main) ✖ rails generate devise User
```

Con este comando generaremos el **modelo User** de devise, el cual nos generará la migración, el modelo, los test y la ruta. Ustedes pueden ponerle Visitant o Person, etc. En este caso se llamará **User**.

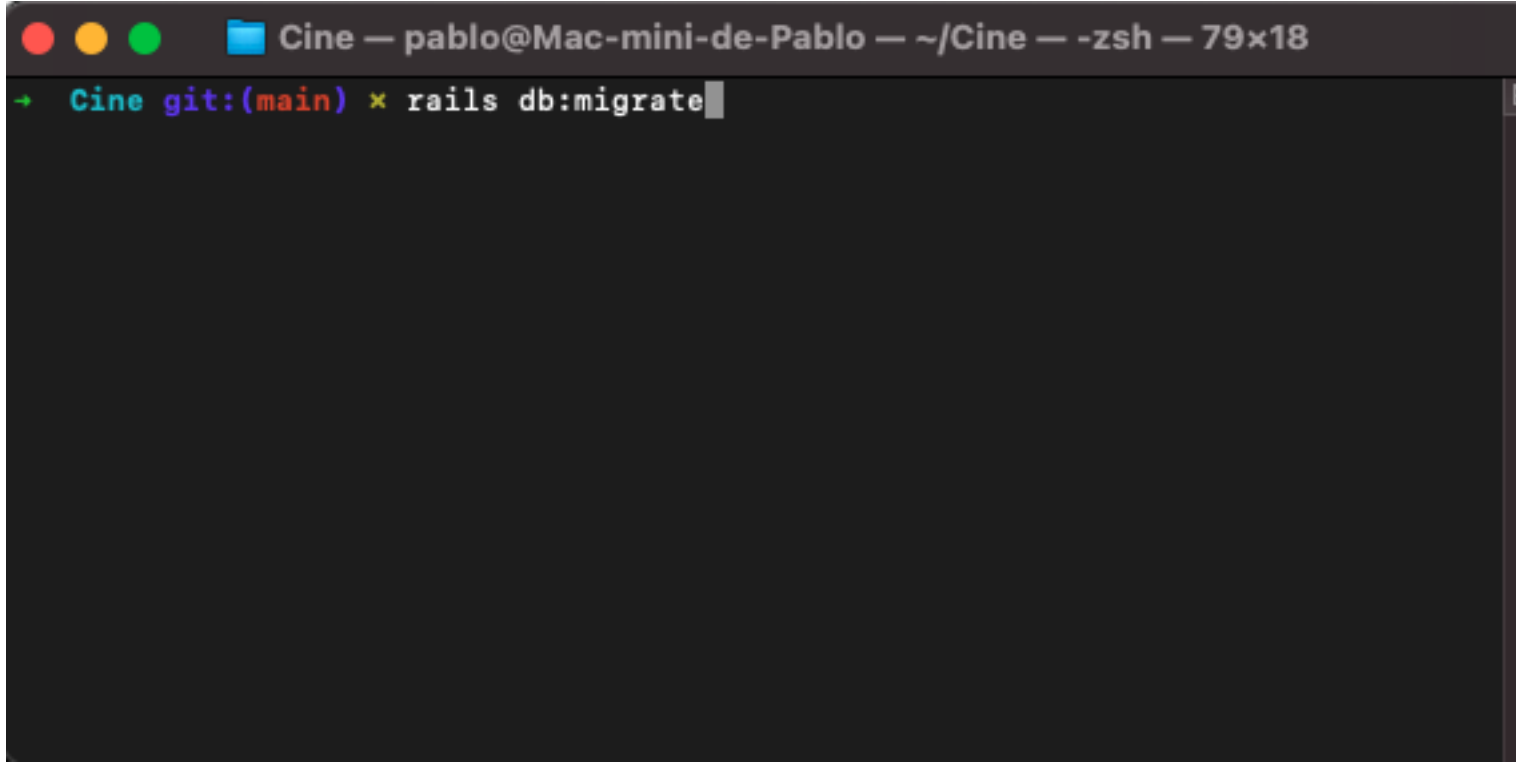
Generando las vistas con devise



```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine — -zsh — 80x24
→ Cine git:(main) ✖ rails g devise:views
```

Con este comando generaremos **las vistas** con devise, este nos proveerá vistas de errores, confirmación, registro, etc.

Migrar modelo User generado por devise



```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine — -zsh — 79x18
→ Cine git:(main) ✖ rails db:migrate
```

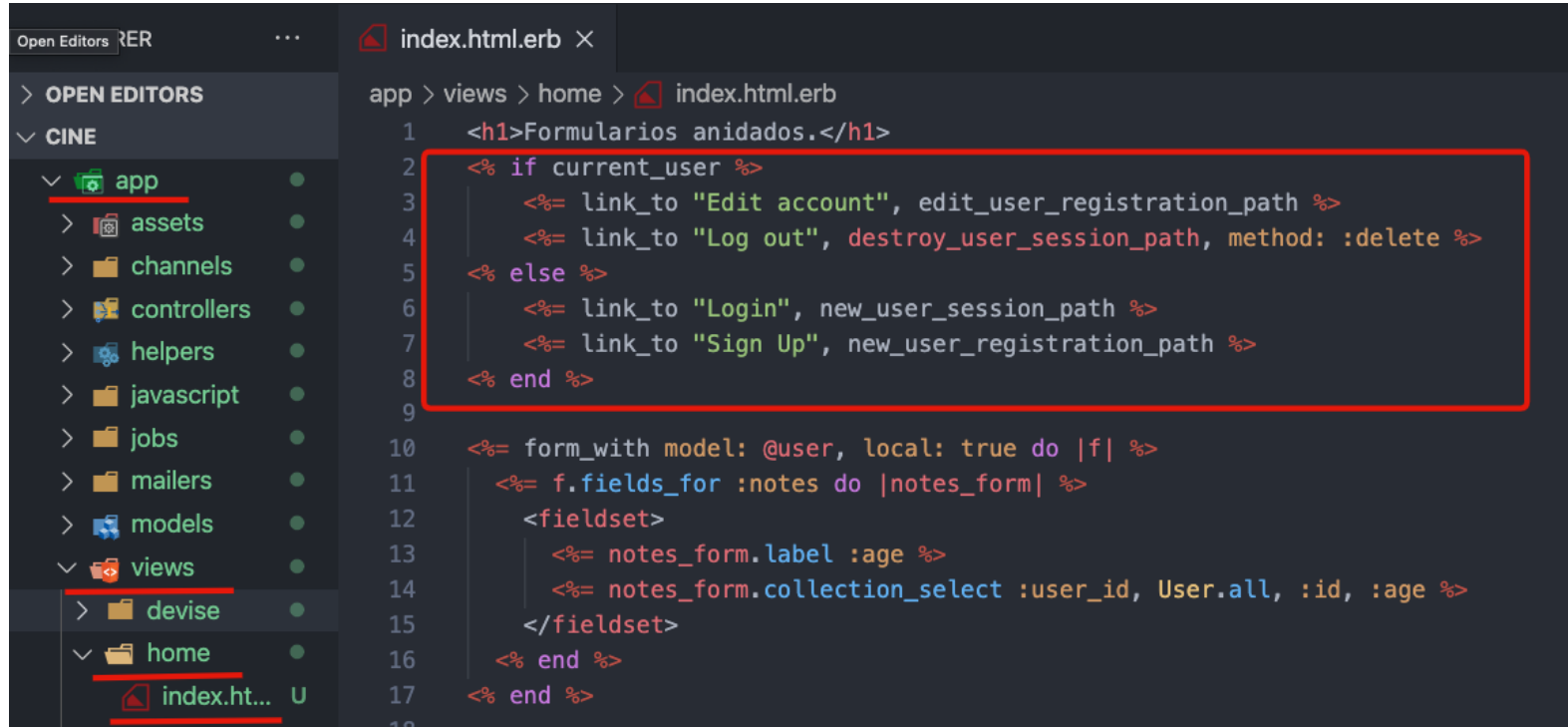
Cada vez que creamos algún modelo o insertemos alguna migración, debemos migrarlo. esto les arrojará en blanco unos mensajes

Helpers `current_user` y `user_signed_in?`

Los **Helpers** a los que vas a tener acceso en las **vistas** son los siguientes:

Método	Descripción
<code>user_signed_in?</code>	Retorna <code>true</code> si el usuario está autenticado, <code>false</code> de lo contrario
<code>current_user</code>	Retorna el usuario que está autenticado o <code>nil</code>

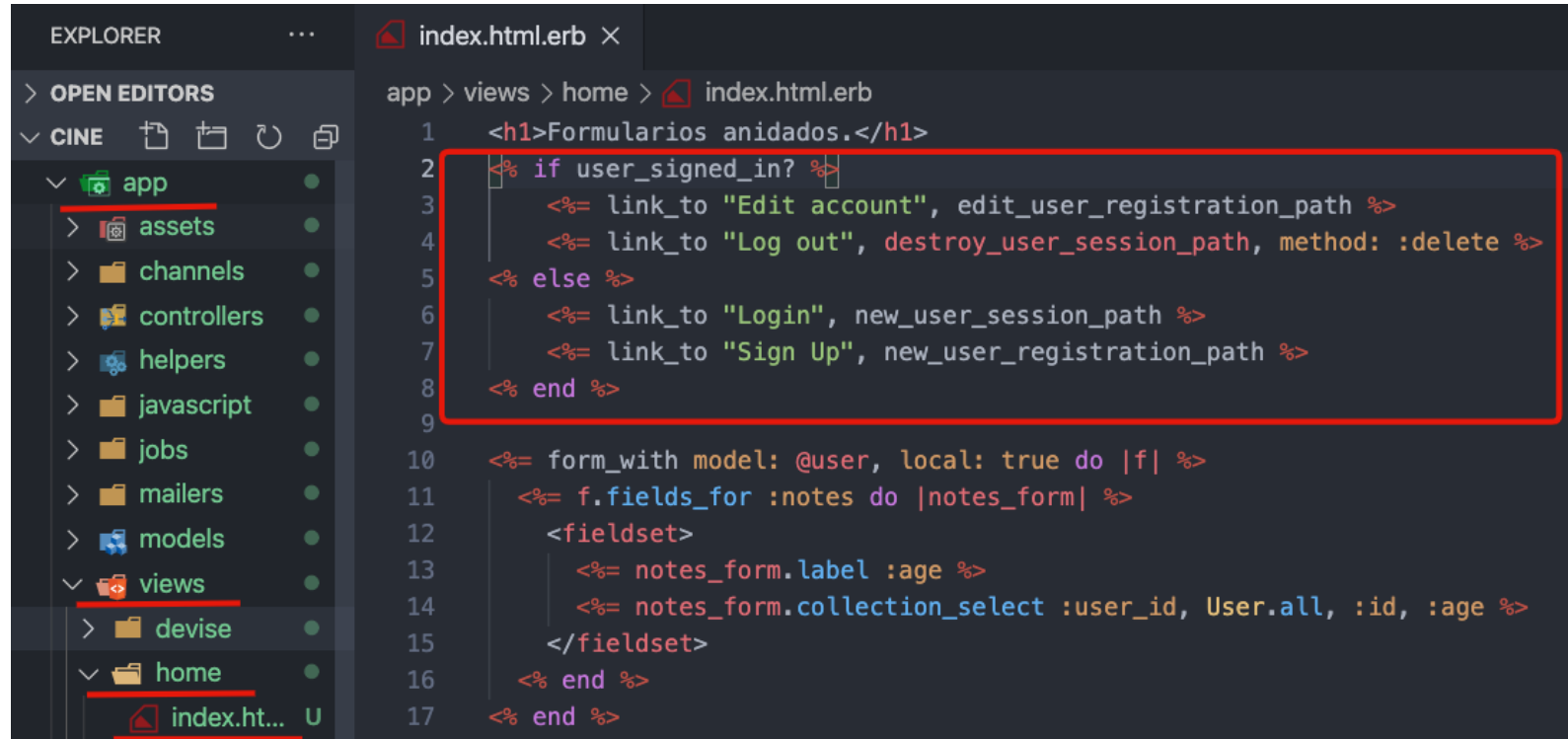
Probando current_user en la vista index



```
app > views > home > index.html.erb
1  <h1>Formularios anidados.</h1>
2  <% if current_user %>
3    <%= link_to "Edit account", edit_user_registration_path %>
4    <%= link_to "Log out", destroy_user_session_path, method: :delete %>
5  <% else %>
6    <%= link_to "Login", new_user_session_path %>
7    <%= link_to "Sign Up", new_user_registration_path %>
8  <% end %>
9
10 <%= form_with model: @user, local: true do |f| %>
11   <%= f.fields_for :notes do |notes_form| %>
12     <fieldset>
13       <%= notes_form.label :age %>
14       <%= notes_form.collection_select :user_id, User.all, :id, :age %>
15     </fieldset>
16   <% end %>
17 <% end %>
18
```

Podemos usar `user_signed_in?` en un bloque if con el helper `link_to` y con los path que corresponde. en este caso para editar la cuenta y salirnos de la sesión destruyendolo con el método `delete`.

Probando user_signed_in? en la vista index



```
1 <h1>Formularios anidados.</h1>
2 <% if user_signed_in? %>
3   <%= link_to "Edit account", edit_user_registration_path %>
4   <%= link_to "Log out", destroy_user_session_path, method: :delete %>
5 <% else %>
6   <%= link_to "Login", new_user_session_path %>
7   <%= link_to "Sign Up", new_user_registration_path %>
8 <% end %>
9
10 <%= form_with model: @user, local: true do |f| %>
11   <%= f.fields_for :notes do |notes_form| %>
12     <fieldset>
13       <%= notes_form.label :age %>
14       <%= notes_form.collection_select :user_id, User.all, :id, :age %>
15     </fieldset>
16   <% end %>
17 <% end %>
```

Podemos usar **user_signed_in?** en un **bloque if** con el **helper link_to** y con los **path** que corresponde. en este caso para editar la cuenta y **salirnos de la sesión destruyendolo** con el **método delete**.

Entendiendo la funcionalidad final de `current_user` y `user_signed_in?`

El método `current_user` devuelve el usuario que **ha iniciado sesión actual**, mientras que `user_signed_in?` se utiliza **para verificar si algún usuario ha iniciado sesión y devuelve verdadero o falso**. **if `user_signed_in?` es falso**, entonces el método `current_user` devolverá `nil`.

Podemos usar los 2 helpers de devise sin ningún problema en cualquier situación, siempre y cuando teniendo en consideración cuando este devuelve `nil`, verdadero o falso.

Por ejemplo en el registro o inicio de sesión index sin registrarse o iniciando sesión, ocurrirá lo mismo, al final no es tan diferente aun que `current_user` se puede usar para otras cosas como `current_user.username` que devuelve el user actual de la persona que inicio sesión.

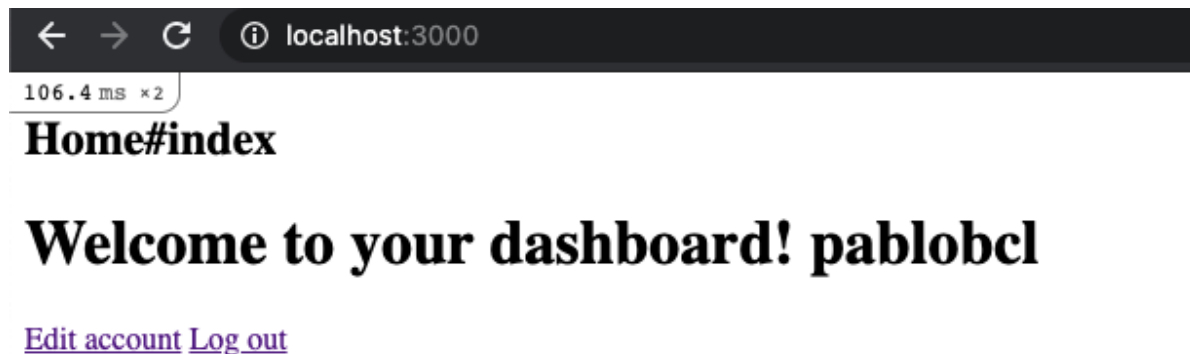
Asociando el path de edición a la vista index

Asociamos con `user_signed_in?` y con un `link_to`, para poder mostrar “Edit account” en un bloque if

```
<h1>Welcome to your dashboard!</h1>
<div>
  <% if user_signed_in? %>
    <%= link_to "Edit account", edit_user_registration_path %>
    <%= link_to "Log out", destroy_user_session_path, method: :delete %>
  <% else %>
    <%= link_to "Login", new_user_session_path %>
    <%= link_to "Sign Up", new_user_registration_path %>
  <% end %>
</div>
```

Nota: La vista `index` se encuentra en `app/views/home/index.html.erb`

Finalizando y corroborando la edición de usuario



A screenshot of a web browser at localhost:3000/cine/edit. The address bar shows 'localhost:3000/cine/edit' and a loading time of 171.2 ms. The page title is 'Edit User'. The form contains the following fields and labels:

- Email:
- Password (leave blank if you don't want to change it):
- 6 characters minimum Password confirmation:
- Current password (we need your current password to confirm your changes):
- Username:
- Age:
- Country:

Below the form is an 'Update' button. Underneath the form, there is a section titled 'Cancel my account' with the text 'Unhappy?' and a 'Cancel my account' button. At the bottom of the form area is a 'Back' link.

Nota: Omitir los campos Username, Age y Country, se verá posteriormente y en el docx.

Unidad 2

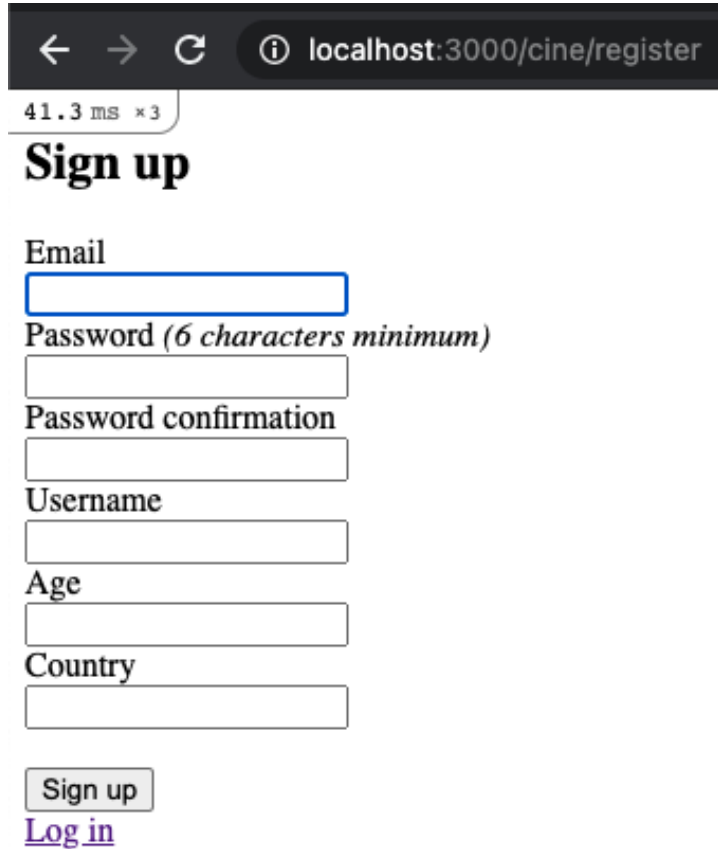
Redireccionando en función de si un usuario ha ingresado

Usando el Helper de devise authenticate_user!

```
app > controllers > users_controller.rb
~/Cine/app • Contains emphasized items
1 class UsersController < ApplicationController
2   before_action :authenticate_user!
3
4   def index
5     @user = User.all
6   end
7
```

Para usar bien nuestro modelo **User** y la gema **devise**, así como los mensajes flash, es recomendable usar un **before_action**. Solo en esta ocasión, el cual nos permite ejecutar antes que todo lo demás descrito, es decir antes de la acción o método.

Crear cuenta e iniciar sesión



← → ↻ ⓘ localhost:3000/cine/register

41.3 ms × 3

Sign up

Email

Password (6 characters minimum)

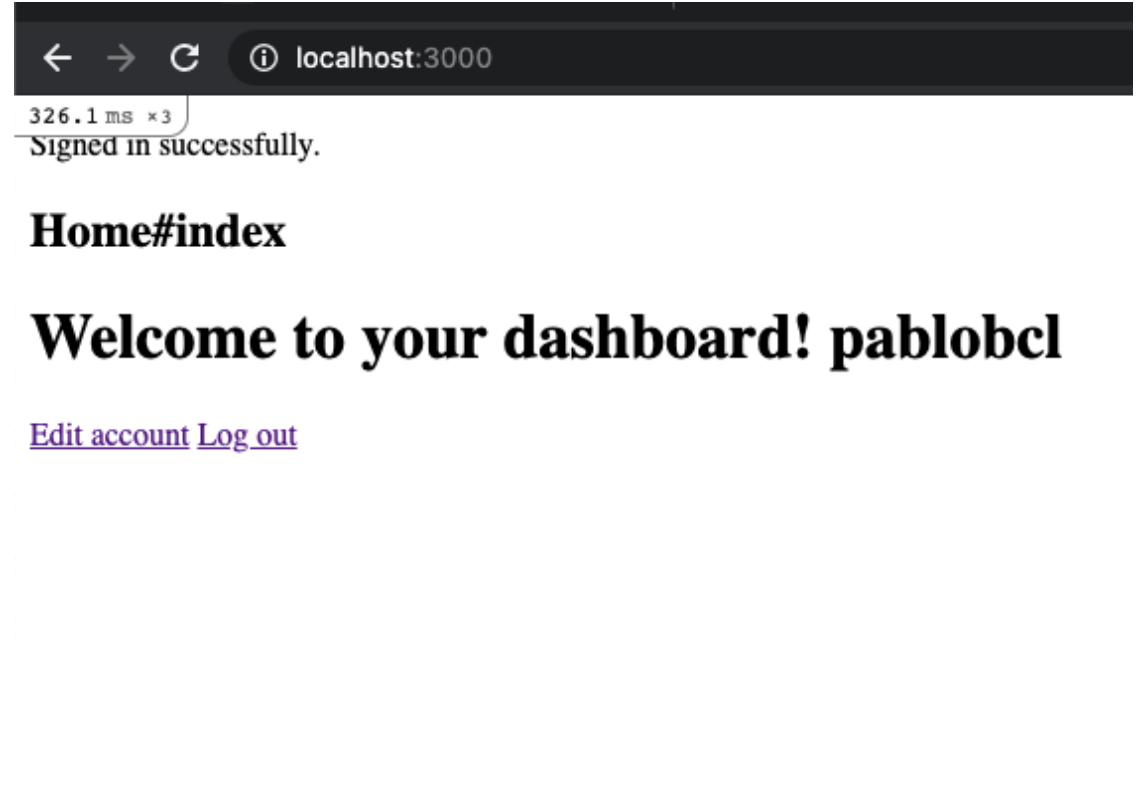
Password confirmation

Username

Age

Country

[Log in](#)



← → ↻ ⓘ localhost:3000

326.1 ms × 3

Signed in successfully.

Home#index

Welcome to your dashboard! pablo bcl

[Edit account](#) [Log out](#)

Finalmente podemos crear e iniciar sesión en nuestro actual formulario HTML5 con la ruta **localhost:3000** en tu navegador web.

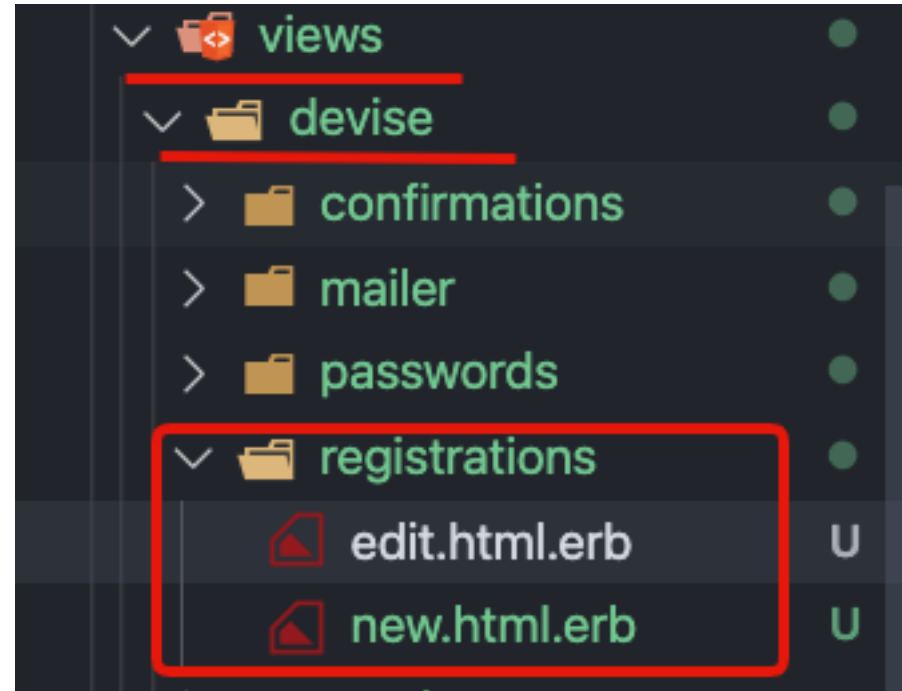
Unidad 2

Personalizar las vistas de devise

Añadir nuevos campos al formulario de las vistas y usaremos strong params

Para esta unidad añadiremos todo los parámetros de los modelos anteriormente mediante formularios también le diremos **a devise** que vamos a usar los **sanitizers** para poder crearlo dentro de las **parciales y vistas**. usando **strong params** permitiremos solo los que hemos definido protegiendo así lo más delicado y asegurarnos ante un posible **ataque CSRF**

Implementar los cambios en las vistas de registros



Nos dirigimos primero a **app/views/devise/registrations/edit.html.erb**
luego a **new.html.erb**

Implementar los cambios en las vista de edición

```
app > views > devise > registrations > edit.html.erb
34 <div class="field">
35   <%= f.label :username %><br />
36   <%= f.text_field :username, autofocus: true %>
37 </div>
38
39 <div class="field">
40   <%= f.label :age %><br />
41   <%= f.number_field :age, autofocus: true %>
42 </div>
43
44 <div class="field">
45   <%= f.label :country %><br />
46   <%= f.collection_select :country, User.all, :id, :country, prompt: "Select your country" %>
47 </div>
```

Añadimos nuevos campos `:username`, `:age`, `:country`.

Dentro de un **div** con los **labels**, **tex_fields**, etc.

Implementar los cambios en las vistas de registros

```
app > views > devise > registrations > new.html.erb
23
24 <div class="field">
25   <%= f.label :username %><br />
26   <%= f.text_field :username, autocomplete: "username"%>
27 </div>
28
29 <div class="field">
30   <%= f.label :age %><br />
31   <%= f.text_field :age, autocomplete: "age"%>
32 </div>
33
34 <div class="field">
35   <%= f.label :country %><br />
36   <%= collection_select(resource, :user_id, User.all, :id, :country,
37     :prompt => 'Please select a country') %>
38 </div>
```

Añadiremos otro campo llamado country, country usa un **collection_select** para mostrar el país el cual se registro el usuario, con un **prompt** que despliega con el mensaje **“Select your country”**

Implementar los cambios en las vistas de registros

```
app > views > devise > registrations > new.html.erb
23
24 <div class="field">
25   <%= f.label :username %><br />
26   <%= f.text_field :username, autofocus: true %>
27 </div>
28
29 <div class="field">
30   <%= f.label :age %><br />
31   <%= f.number_field :age, autofocus: true %>
32 </div>
33
34 <div class="field">
35   <%= f.label :country %><br />
36   <%= f.text_field :country, autofocus: true %>
37 </div>
```

Añadimos los campos posteriores pero esta vez hicimos un pequeño cambio field **:country como text_field**, mucho cuidado con la **diferencia**, en este caso Vamos a ponerlo como campo texto para crear el país junto al nombre de usuario y la edad.

Implementando el sanitizer

app > controllers > application_controller.rb

```
1  class ApplicationController < ActionController::Base
2    before_action :configure_permitted_parameters, if: :devise_controller?
3
4    protected
5
6    def configure_permitted_parameters
7      devise_parameter_sanitizer.permit(:sign_up, keys: [:username, :age, :country, :email, :password])
8      devise_parameter_sanitizer.permit(:account_update, keys: [:username, :age, :country, :email, :current_password])
9    end
10 end
11
```

Vamos a application_controller y le pasamos un before_action con el callback :configure_permitted_parameters

Dentro del callback hay dos sanitizers que permite añadir y editar en la vista de sign_up las llaves :username, :age, :country, etc. Al igual que en account_update pero donde actualizaremos los datos de las llaves :username, :age, etc. Todo estará guardado Y seguro a la vez.

Strong params


Para usar **strong param** debemos ir al controlador padre, en este caso el **users_controller** luego escribir lo siguiente:

```
29
30   private
31   def user_params
32     params.require(:user).permit(:username, :age, :country)
33   end
34 end
35
```

Usando el **uso reservado private**, más el strong params y los atributos de movies, luego puedes usarlo en el método create por ejemplo.

Validaciones y accepts_nested_attributes_for

Lo siguiente que vamos a hacer es **validar :username, :age, :country** en nuestro modelo User

```
app > models >  user.rb
1  class User < ApplicationRecord
2    has_many :movies
3
4    accepts_nested_attributes_for :movies
5
6    validates :username, presence: true
7    validates :age, presence: true, length: { maximum: 2}
8    validates :country, presence: true
9    # Include default devise modules. Others available are:
10   # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
11   devise :database_authenticatable, :registerable,
12          :recoverable, :rememberable, :validatable
13 end
```

A :age le vamos a colocar un **length máximo de 2 dígitos**, por último vamos a aceptar los atributos nested para :movies.

users_controller y su adaptación

```
app > controllers > users_controller.rb
1  class UsersController < ApplicationController
2    before_action :authenticate_user!
3
4    def index
5      @user = User.all
6    end
7
8    def show
9      @user = User.find(params[:id])
10   end
11
12   def new
13     @user = User.new
14   end
15
16   def edit
17     @user = User.find(params[:id])
18   end
19
20   def create
21     @user = User.new(user_params)
22
23     if @user.save
24       redirect_to @user
25     else
26       render :new, status: :unprocessable_entity
27     end
28   end
29
30   private
31   def user_params
32     params.require(:user).permit(:username, :age, :country)
33   end
34 end
```

Atributos en las vistas

Los atributos se pueden escribir en las vistas y mostrarlas en la web, se puede iterar de varias formas, ciclos, times, bloques if, etc. Podemos agregar lo siguiente para poder mostrar al iniciar sesión nuestro nombre creado en el **form**. Usaremos **current_user.username**, también podemos usar **current_user.age**, **current_user.country**

current_user, usará el usuario actual de la sesión con el atributo **username**. y este lo imprimirá.

```
app > views > users > index.html.erb
1  <h2>Home#index</h2>
2  |  <h1>Welcome to your dashboard! <%= current_user.username %></h1>
3  <div>
4  |  <% if user_signed_in? %>
5  |  |  <%= link_to "Edit account", edit_user_registration_path %>
6  |  |  <%= link_to "Log out", destroy_user_session_path, method: :delete %>
7  |  <% else %>
8  |  |  <%= link_to "Login", new_user_session_path %>
9  |  |  <%= link_to "Sign Up", new_user_registration_path %>
10 |  <% end %>
11 </div>
```

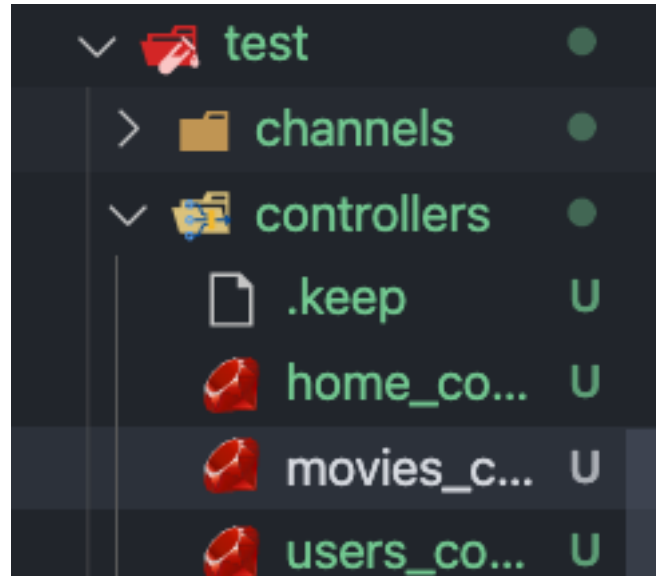
Qué significa pruebas unitarias

Las pruebas unitarias permiten al programador **refactorizar** el código o actualizar las bibliotecas del sistema en una fecha posterior y asegurarse de que el **módulo aún funciona correctamente**. El procedimiento consiste en escribir casos de prueba para todas las funciones y métodos, de modo que siempre que un cambio **cause una falla**, se pueda **identificar rápidamente**.

Pruebas unitarias con acceso devise

Usaremos de prueba los controladores en la **carpeta controllers** de test también usaremos los **modelos y fixtures**.


Usaremos **users_controller_test**




Nota: esta carpeta de test se puede encontrar en la raíz de tu proyecto en test/controllers

Pruebas unitarias users_controller_test con acceso devise

Al abrir **users_controller_test** nos encontraremos con varios test, borramos todo y empezaremos desde 0. En este caso vamos a escribir lo siguiente:

```
test > controllers >  users_controller_test.rb
1  require "test_helper"
2
3  class UsersControllerTest < ActionDispatch::IntegrationTest
4    test "should get index" do
5      get users_url
6      assert_response :success
7    end
8  end
9
```

Pruebas unitarias users_controller_test con acceso devise

```
test > controllers >  users_controller_test.rb
1  require "test_helper"
2
3  class UsersControllerTest < ActionDispatch::IntegrationTest
4    test "should get index" do
5      get users_url
6      assert_response :success
7    end
8  end
9
```

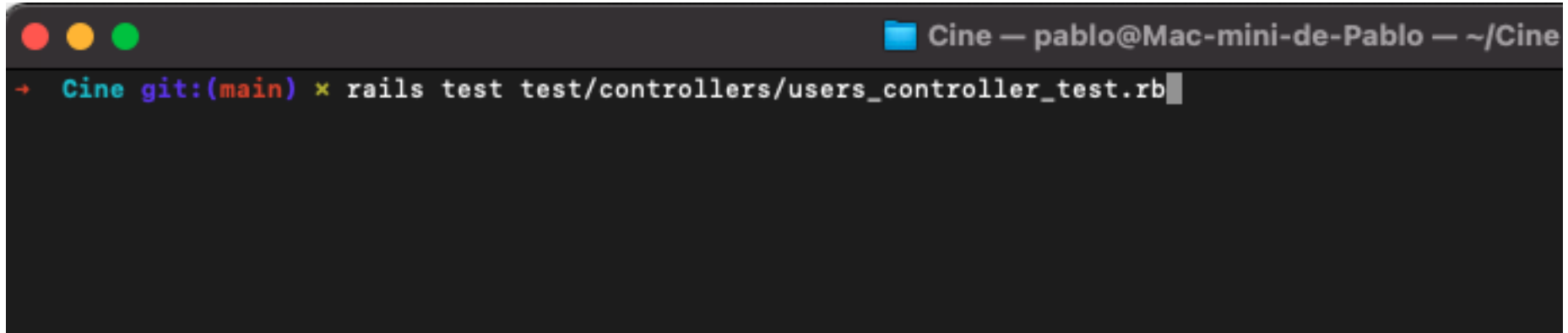
Usamos el método **test** y escribimos **"should get index"** ← esto quiere decir: **debería obtener respuesta de index?**, iniciamos un **bloque do**

Escribimos **get users_url** ← esto quiere decir: usamos el método petición **GET** y **users_url** que sería la **ruta /users/sign_in**

Finalmente escribimos **assert_response :success** ← esto quiere decir: verifica que la respuesta a una petición tenga el código de respuesta y **:success** que sería como respuesta **200** osea, pasó **exitosamente** y cerramos con el bloque con **end**

Probando tests de users_controller_test

Para correr la prueba test de users_controller escribimos en la consola lo siguiente:



```
Cine — pablo@Mac-mini-de-Pablo — ~/Cine
→ Cine git:(main) ✖ rails test test/controllers/users_controller_test.rb
```

Probando tests de users_controller_test

Corrimos la prueba pero nos dio error, esto era de esperarse en una prueba unitaria en este caso obtuvimos una error de **petición 302**, esto estuvo mal dado a la petición y a la vez bien porque era lo que esperábamos de una prueba.

Veamos mas a detalle que es lo que sucedió realmente.

```
# Running:
F
Failure:
UsersControllerTest#test_should_get_index [/Users/pablo/Cine/test/controllers/users_controller_test.rb:6]:
Expected response to be a <2XX: success>, but was a <302: Found> redirect to <http://www.example.com/users/sign_in>
Response body: <html><body>You are being <a href="http://www.example.com/users/sign_in">redirected</a>.</body></html>

rails test test/controllers/users_controller_test.rb:4

Finished in 0.527748s, 1.8948 runs/s, 1.8948 assertions/s.
1 runs, 1 assertions, 1 failures, 0 errors, 0 skips
```

Probando tests de users_controller_test

```
# Running:

F

Failure:
UsersControllerTest#test_should_get_index [/Users/pablo/Cine/test/controllers/users_controller_test.rb:6]:
Expected response to be a <2XX: success>, but was a <302: Found> redirect to <http://www.example.com/users/sign_in>
Response body: <html><body>You are being <a href="http://www.example.com/users/sign_in">redirected</a>.</body></html>

rails test test/controllers/users_controller_test.rb:4

Finished in 0.527748s, 1.8948 runs/s, 1.8948 assertions/s.
1 runs, 1 assertions, 1 failures, 0 errors, 0 skips
```

Hay 3 soluciones para esto, modificar los 3 archivos de la carpeta test:

- **test/test_helper.rb**
- **test/controllers/users_controller_test.rb**
- **test/fixtures/users.yml**

Añadiendo los modulos de devise, devise y warden en test_helper

Incluimos los **modulos** de devise debajo de **fixtures :all, Devise::Tests** con la **integración de los Helpers** e incluimos **Warden::Test** con los **Helpers** dentro del archivo **test_helper**

```
include Devise::Test::IntegrationHelpers
include Warden::Test::Helpers
```

Creando método log_in

Creamos un método **log_in** con argumento **username**, iniciamos con **if integration_test?** da verdadero, pasa a **login_as username** el cual tiene el **alcance** de **:username** nuestro parámetro. **de lo contrario**, si pasa a **falso** pasa a **sign_in username**.

```
11  def log_in(username)
12    if integration_test?
13      login_as(username, scope: :username)
14    else
15      sign_in(username)
16    end
17  end
18 end
19
```

Incluyendo y configurando un bloque GET y POST con devise y :user1 de fixtures

Incluimos **devise test** con **include Devise::Test::IntegrationHelpers** y luego hacemos un **setup** y abrimos el bloque con **do**

```
3  class UsersControllerTest < ActionDispatch::IntegrationTest
4    include Devise::Test::IntegrationHelpers
5
6    setup do
7      get "/users/sign_in"
8      sign_in users(:user1)
9      post user_session_url
10   end
```

Esto significa que, iniciamos una configuración y que haga una petición de respuesta con **GET** a **"/users/sign_in"**

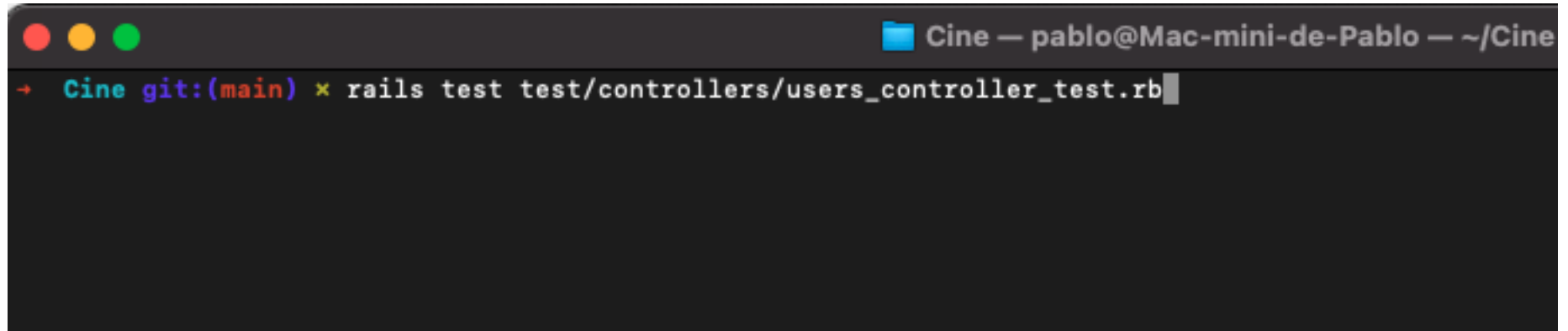
Añadiendo un Fixtures user1 en formato YML

```
test > fixtures > yml users.yml
1  # Read about fixtures at https://a
2  user1:
3    id: 1
4    username: "pablobcl"
5    age: 23
6    country: "Chile"
7    email: "pablo@pablo.cl"
8
```

Para finalizar y probar de este exitosamente y a la vez tampoco, nos vamos a **test/fixtures/users.yml**
Dentro de ese archivo **YML** crearnos un user1, en este caso el mío cuando nos registramos con el formulario
Un **user1** con **id: 1**, **username: "pablobcl"**, **age: 23**, **country: "Chile"** e **email: pablo@pablo.cl**, también podemos
Colocar el **password digest**, osea el **password tuyo encriptado**, todo lo que hayas hecho se coloca acá para que
Ese **user1** de **Fixtures YML**, pase a **:user1** un **hash** que vamos a usar en la **prueba unitaria con devise**.

Probando el test unitario en devise en consola

Nuevamente corremos el test con el siguiente comando:

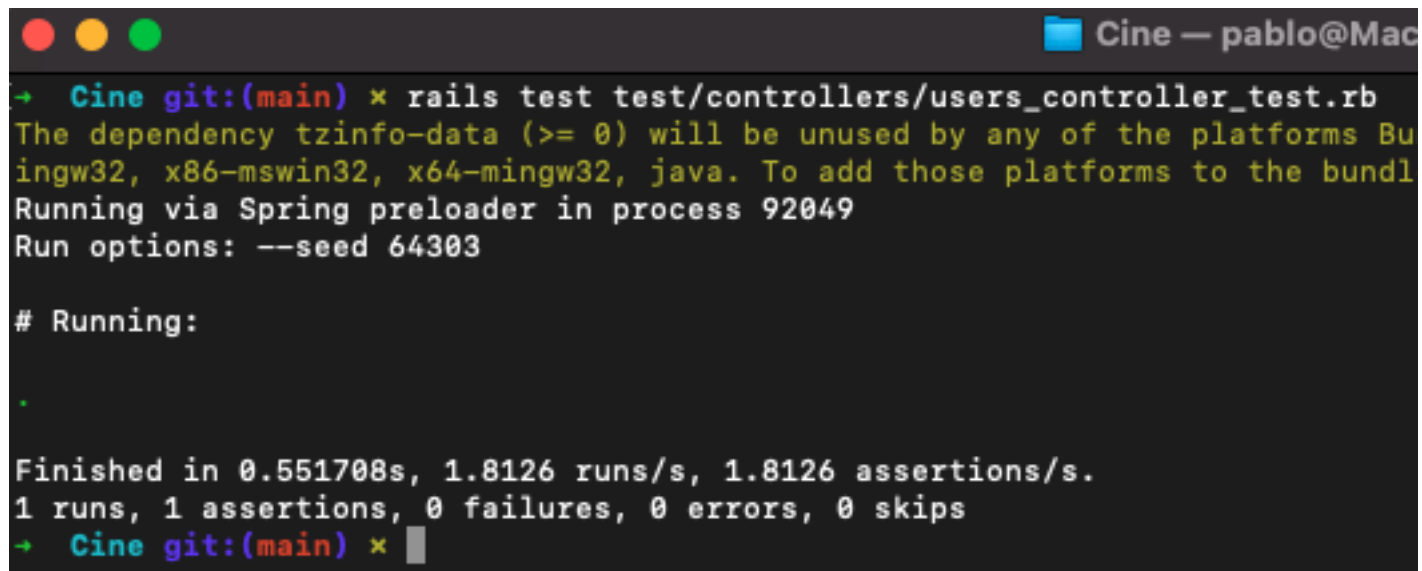
A terminal window with a dark background. The title bar shows three colored window control buttons (red, yellow, green) on the left and a folder icon followed by the text 'Cine — pablo@Mac-mini-de-Pablo — ~/Cine' on the right. The terminal content shows a prompt '→ Cine git:(main) ✖' followed by the command 'rails test test/controllers/users_controller_test.rb' and a cursor at the end of the line.

```
→ Cine git:(main) ✖ rails test test/controllers/users_controller_test.rb
```

Resultado del test unitario en devise por consola

Cuando haya **un punto en verde** este significa que pasó exitosamente la prueba

En comparación al primero que mostraba una **E** en rojo

A terminal window titled "Cine — pablo@Mac" with standard macOS window controls (red, yellow, green buttons). The terminal shows the command to run a Rails test: `→ Cine git:(main) × rails test test/controllers/users_controller_test.rb`. It displays output from Bundler about the `tzinfo-data` dependency, the Spring preloader process ID (92049), and run options (`--seed 64303`). A section labeled `# Running:` shows a single green dot `•` indicating a successful test run. The summary line reads: `Finished in 0.551708s, 1.8126 runs/s, 1.8126 assertions/s.` followed by `1 runs, 1 assertions, 0 failures, 0 errors, 0 skips`. The prompt returns to `→ Cine git:(main) ×` with a cursor.

```
→ Cine git:(main) × rails test test/controllers/users_controller_test.rb
The dependency tzinfo-data (>= 0) will be unused by any of the platforms Bu
ingw32, x86-mswin32, x64-mingw32, java. To add those platforms to the bundl
Running via Spring preloader in process 92049
Run options: --seed 64303


# Running:

•

Finished in 0.551708s, 1.8126 runs/s, 1.8126 assertions/s.
1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
→ Cine git:(main) ×
```

Resultado inverso el test unitario en devise en el controlador

Escribimos un método vacío sin definir por ejemplo:

```
app > controllers >  users_controller.rb  
1  class UsersController < ApplicationController  
2  
3    fallo_en_este_metodo  
4
```

Resultado final inverso el test unitario en devise en consola

Resultado final con un método no definido en `users_controller.rb`

```
→ Cine git:(main) × rails test test/controllers/users_controller_test.rb
The dependency tzinfo-data (>= 0) will be unused by any of the platforms B
ingw32, x86-mswin32, x64-mingw32, java. To add those platforms to the bund
Running via Spring preloader in process 92391
Run options: --seed 40936

# Running:

E

Error:
UsersControllerTest#test_should_get_index:
NameError: undefined local variable or method `fallo_en_este_metodo' for U
  app/controllers/users_controller.rb:4:in `<class:UsersController>'
  app/controllers/users_controller.rb:1:in `<main>'
  test/controllers/users_controller_test.rb:15:in `block in <class:Users

rails test test/controllers/users_controller_test.rb:14

Finished in 0.205279s, 4.8714 runs/s, 0.0000 assertions/s.
1 runs, 0 assertions, 0 failures, 1 errors, 0 skips
→ Cine git:(main) ×
```

Unidad 2

Usuarios y roles

Limitar acceso al rol User

En este escenario veremos como **limitar** de **forma super simple** los **accesos o funcionalidades**, en este caso del **rol usuario**

Haremos cambios a las **vistas, controladores y rutas** para tener una mejor experiencia y que esté todo correcto

Limitando el acceso al rol User

Para agregar los siguientes cambios a nuestro **index.html.erb** de la vista en la carpeta home, para explicar el procedimiento es simplemente cambiar el bloque de código y añadir lo siguiente:

```
app > views > home > index.html.erb
```

```
1  <h2>Home#index</h2>
2  <h1>Welcome to your dashboard! <%= current_user.username %></h1>
3  <div>
4    <% if current_user.username == "pabloski123@"%>
5      <%= link_to "Edit account", edit_user_registration_path %>
6      <%= link_to "Log out", destroy_user_session_path, method: :delete %>
7    <% else %>
8      <%= "You can't edit your account, because your name is #{current_user.username} !"%>
9      <%= link_to "Log out", destroy_user_session_path, method: :delete %>
10   <% end %>
11 </div>
```


Mejorando rutas

En el archivo **routes.rb** cambiamos **devise_for** por los siguientes parámetros:

```
config > routes.rb
1  Rails.application.routes.draw do
2
3    devise_for :users, path: 'cine', path_names: { sign_in: 'login', log_out: 'logout', sign_up: 'register' }
4
5    root to: "home#index"
6
7    resources :users do
8      resources :movies
9    end
10 end
11
```

Finalizando y mejorando el controlador

Para mejorar los controladores, en el controlador **application_controller** agregamos el **Helper before_action**

Ya no es necesario usarlo en un controlador en específico

Ya que no estaríamos usando el principio **DRY == don't repeat yourself**

```
app > controllers > application_controller.rb
~/Cine/app/controllers • Contains emphasized items
1  class ApplicationController < ActionController::Base
2      before_action :authenticate_user!
3      before_action :configure_permitted_parameters, if: :devise_controller?
4
5      private
6
7      def after_sign_out_path_for(resource_or_scope)
8          new_user_session_path
9      end
10
```

Y por último también agregamos un **callback** que se asocia a **devise** con las rutas llamado **after_sign_out_path_for**

Material complementario de la unidad

Link a video relacionado

1. <https://www.youtube.com/watch?v=pLbNeangPvU>
2. <https://www.youtube.com/watch?v=tWMIlsZYXyA>

Link a lectura complementaria

1. <https://guias.makeitreal.camp/ruby-on-rails-i/devise>
2. <https://guias.makeitreal.camp/ruby-on-rails-ii/testing-en-rails>

Link a investigación relacionada

1. <https://guias.makeitreal.camp/ruby-on-rails-ii/testing-en-rails>
2. <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>
3. <https://guides.rubyonrails.org/v6.1.0/testing.html>