

MODUL 151

TEIL 4: INSTAGRAM

NOTIFICATION USER PROFIL PAGE

Ralph Maurer

# Inhaltsverzeichnis AB151-04

## Modul 133: Instagram mit Rails bauen

Teil 4: Instagram User Profile Page

[Inhaltsverzeichnis AB151-04 1](#_Toc19502)

[Modul 133: Instagram mit Rails bauen 1](#_Toc19503)

[Flash in Rails 2](#_Toc19504)

[Fancy Flash in Rails mit toastr 3](#_Toc19505)

[Aufgabe: Error Rails flash bei falscher Benutzerregistration 5](#_Toc19506)

[User Profile Page 6](#_Toc19507)

[Model User erweitern 6](#_Toc19508)

[routes.rb 6](#_Toc19509)

[users\_controller.rb 7](#_Toc19510)

[show-View mit Bootstrap Modal 8](#_Toc19511)

[Gravatar für User Photo in show-View 8](#_Toc19512)

[link\_to Profile Page 14](#_Toc19513)

[Edit User Page 15](#_Toc19514)

[Auftrag: Quicknote AB151-04 18](#_Toc19515)

# Flash in Rails

Rails bietet die Möglichkeit temporäre Variablen, normalerweise Meldungen (notification) zwischen Interaktionen (vor allem Actions), weiterzugeben. Alles was Sie im Flash ablegen, wird der nächsten Aktion übergegeben und dann wieder gelöscht. Dies ist eine grossartige Möglichkeit, Benachrichtigungen und Warnungen auszuführen, z. B. von «gem devise».

Nachdem Sie sich in unserer Rails App angemeldet haben, erhalten Sie eine Meldung mit dem Text "Signed in successfully." und verschwindet, wenn Sie zu einer anderen View wechseln oder eine den Browserrefresh durchführen. Nachrichten können unterschiedliche Typen haben:

› success

› notice

› alert

› error

Jede notification besteht aus einem Typen und einer Textnachricht.

In der Datei app/views/layouts/application.html.erb werden Flash wie folgt ausgeben:

<p class="notice"><%= notice %></p>

<p class="alert"><%= alert %></p>

Diese Schreibweise ist gleich wie

<p class="notice"><%= flash[:notice] %></p>

<p class="alert"><%= flash[:alert] %></p>

oder auch

<p class="notice"><%= flash['notice'] %></p> <p class="alert"><%= flash['alert'] %></p>

Schauen Sie sich folgende Videos an:

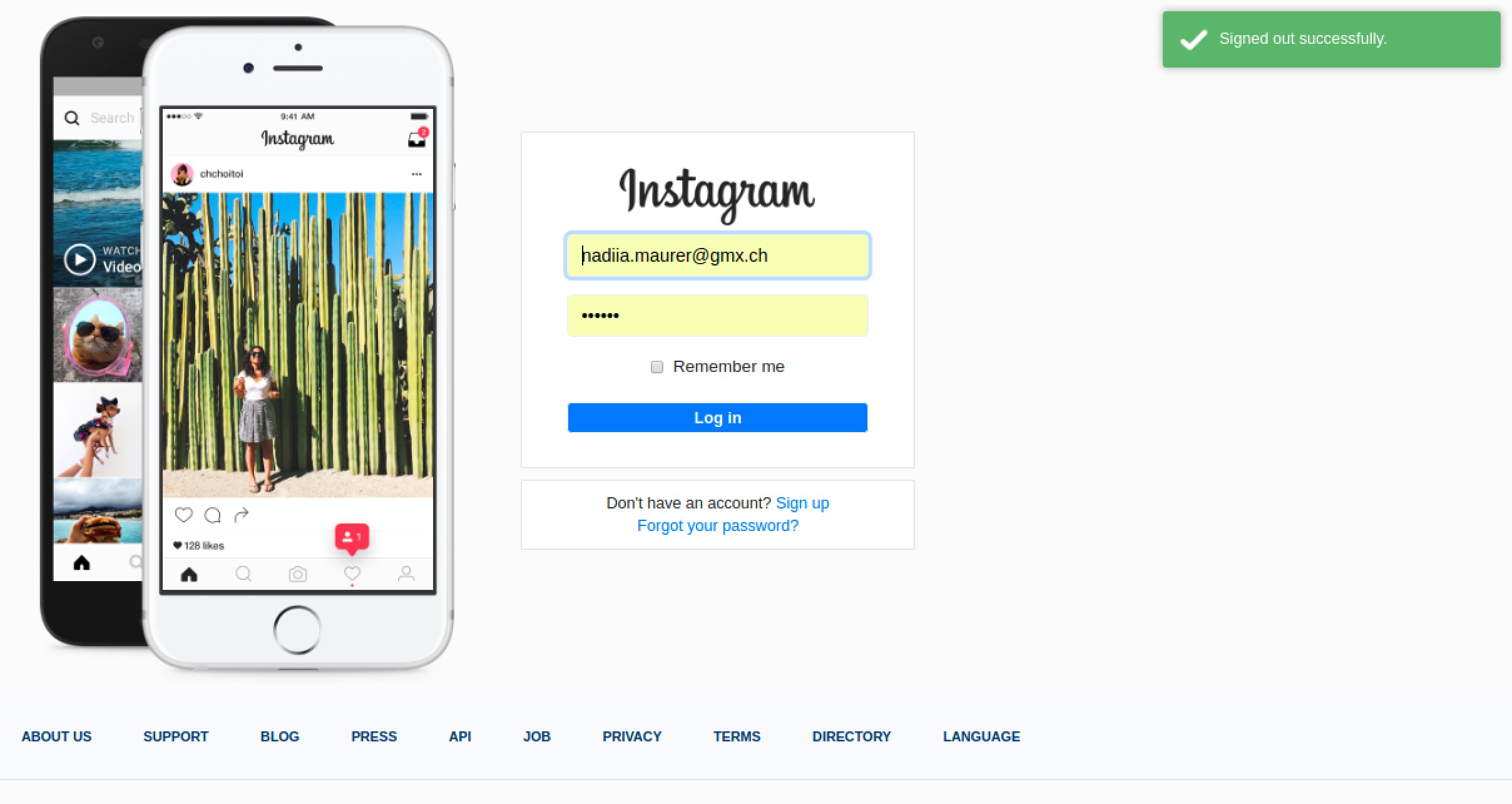
sh-modules/iet-151/03\_Arbeitsblaetter/AB151-04\_flash\_1.mp4 und AB151-04\_flash\_2.mp4.

# Fancy Flash in Rails mit toastr

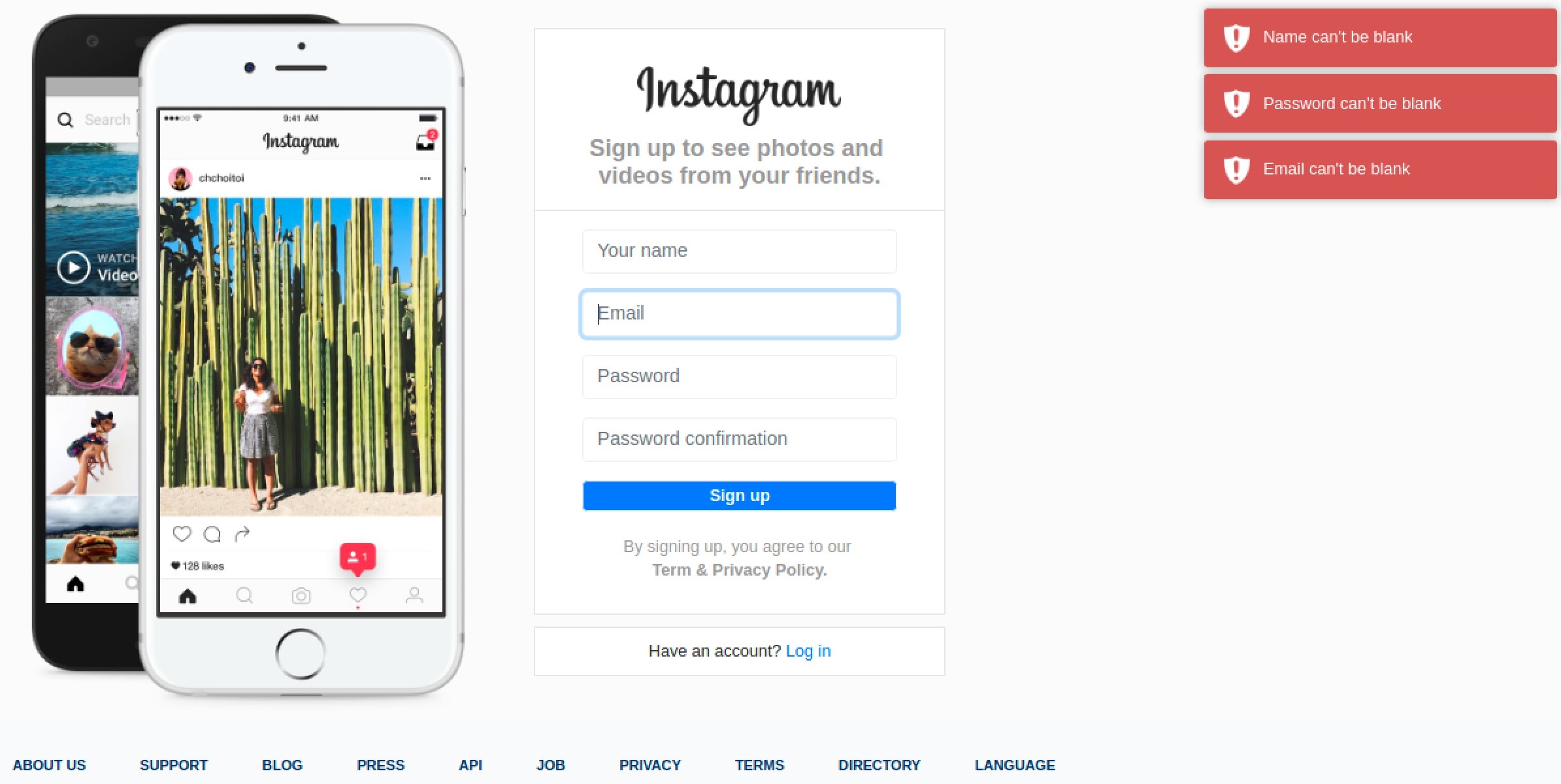
Hier können Sie sich in die «gem toastr» vertiefen:

https://github.com/tylergannon/toastr-rails toastr ist eine coole gem um flash notification in Rails zu gestalten:

Statt eine einfache Textmessage kommen die Nachrichten je nach Typ in einer entsprechenden Farbe oben rechts. Hier das Beispiel beim ausloggen:



oder hier bei einer leeren Benutzerregistration



Installieren Sie toastr:

› Im gemfile

gem 'toastr-rails' eintragen

› In app/assets/javascripts/application.js

//= require toastr eintragen

› In app/assets/stylesheets/application.scss

@import "toastr";

eintragen

Führen Sie im Terminal den Befehl bundle aus und starten Sie den Server neu.

Schreiben Sie in die Datei app/views/layouts/application.html.erb folgenden Code und beschreiben Sie was passiert:

<script type="text/javascript"> toastr.success("Wow, a success")

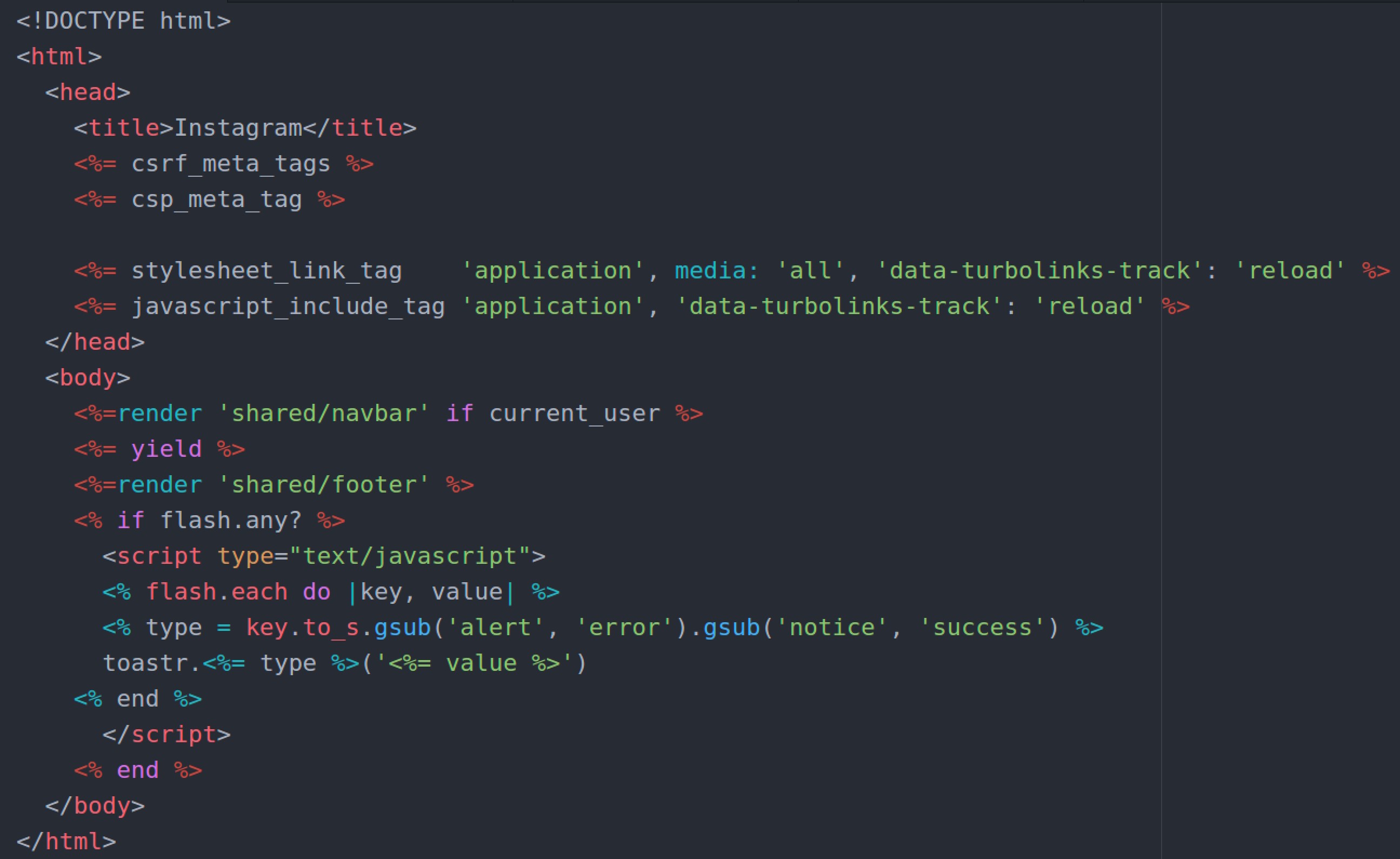
</script>

Da wir nie wissen, ob es mehrere Rails Flash Notification nach einer Action gibt, können wir diese iterieren und z.B. alle ausgeben:

|  |
| --- |
| <script type="text/javascript"> <% flash.each do |key, value| %  toastr.<%= type %>('<%= value %>')  <% end %>  </script> |

Key ist succes, alert, notice oder error. Value beinhaltet den Text der Rails flash.

Passen Sie app/views/layouts/application.html.erb wie folgt an:



## Aufgabe: Error Rails flash bei falscher Benutzerregistration

Erstellen Sie eine Partial-View app/views/shared/\_devisemes.html.erb, die bei falscher Benutzerregistration in der richtigen View gerendert wird und nur Error-flashs ausgibt. Es gibt mehrere und einige sind zu erwähnen:

› Name can’t be blank

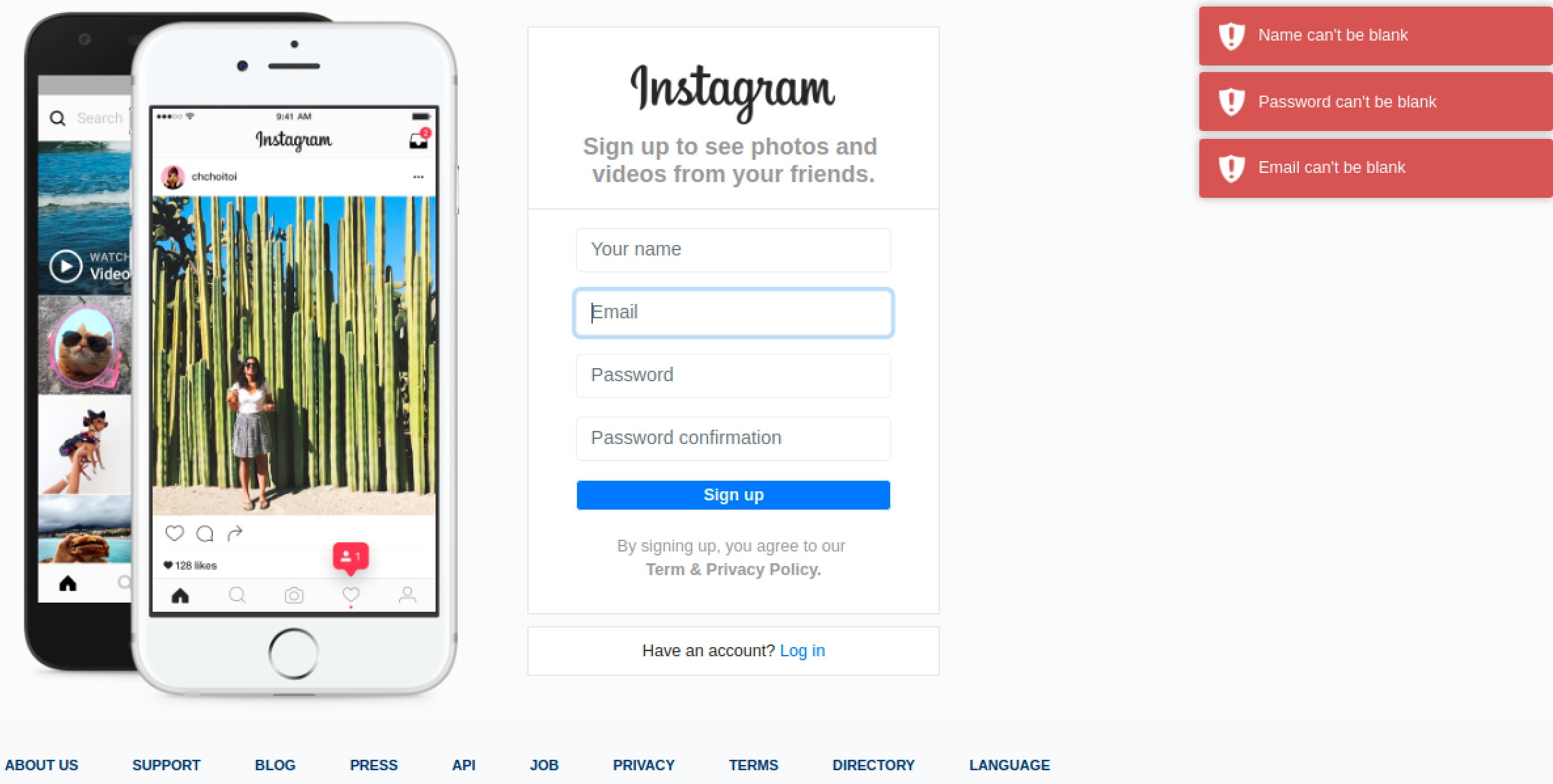
› Email can’t be blank

› Password can’t be blank

› Password confirmation doesn’t match

› Password is too short

› …



# User Profile Page

## Model User erweitern

Erstellen sie eine Migration AddFieldsToUser, die der Tabelle users folgende Attribute hinzufügt:

:users, :website, :string

:users, :bio, :text

:users, :provider, :string

:users, :uid, :string

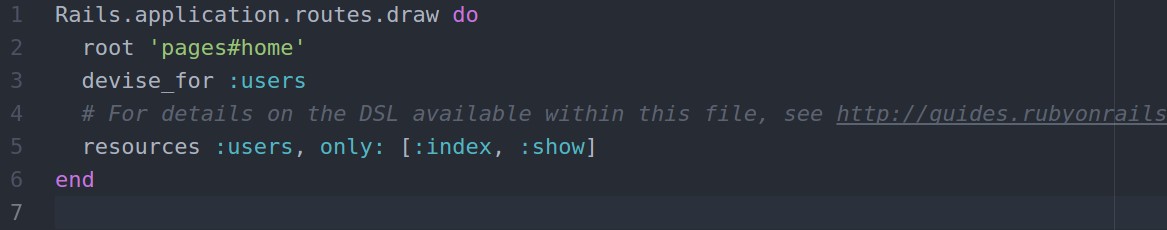
:users, :image, :string

Notieren Sie das Migrationsdatei und die notwendigen Kommandos für die Migration:

|  |
| --- |
|  |
|  |
|  |
| Rails generate migration |
| class AddFieldsToUser < ActiveRecord::Migration[5.2]  def change  add\_column :users, :website, :string  add\_column :users, :bio, :text  add\_column :users, :provider, :string  add\_column :users, :uid, :string  add\_column :users, :image, :string  end  end |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

## routes.rb

In der Datei routes.rb müssen wir nun einen folgenden Eintrag erstellen:

 resources :users ist ein Shortcut für alle http-Verben und umfasst z.B. folgendes:

GET /users /show

GET /users

PATCH/PUT /users

DELETE /users

POST /users

Mit der Einschränkung :only: [:index, :show] grenzen wir den Inhalt von app/views/users/\*.html.erb auf die Views index und show. index ist zwar noch nicht erstellt aber wird folgen.

Das Rails Routing wird ausführlich unter https://guides.rubyonrails.org/routing.html erklärt. Das HowTo ist auch unter sh-modules/iet-151/03\_Arbeitsblaetter/AB15104\_routing.pdf zu finden.

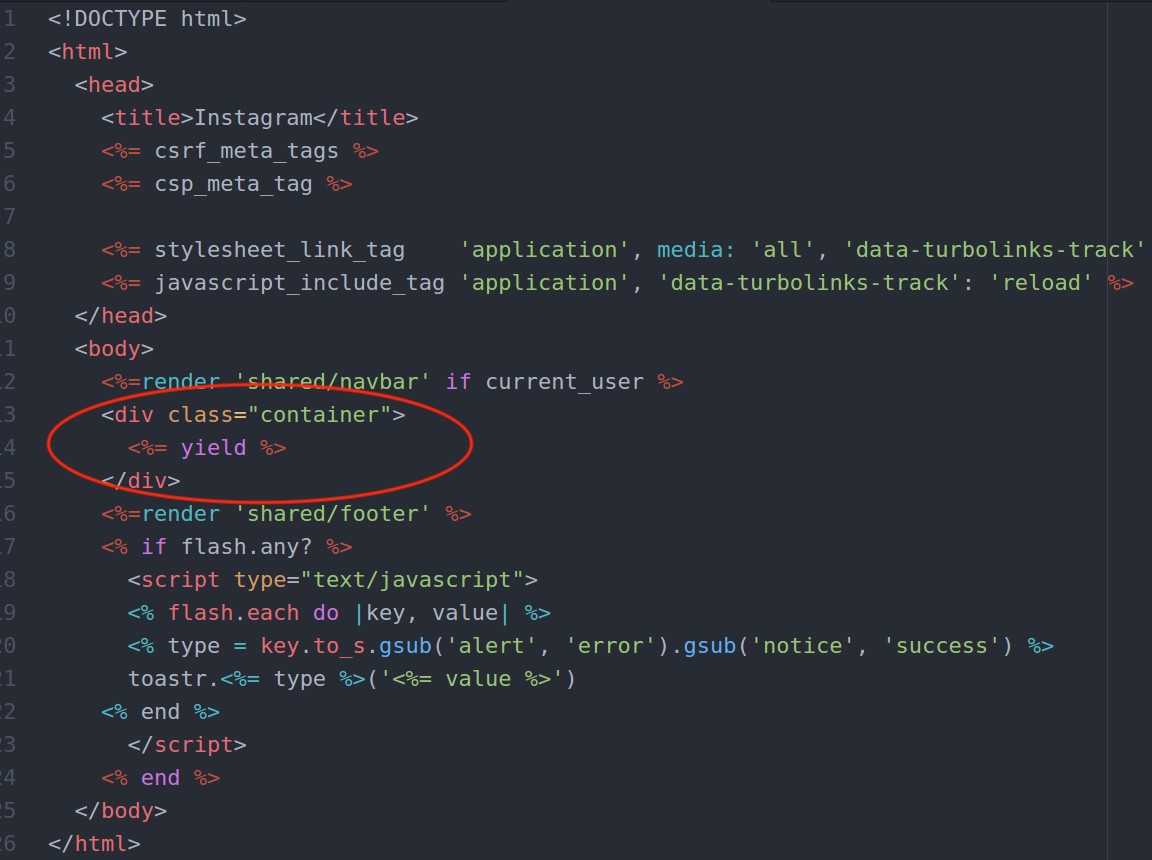
## users\_controller.rb

Erstellen Sie nun manuell (Datei neu) einen Controller mit Namen

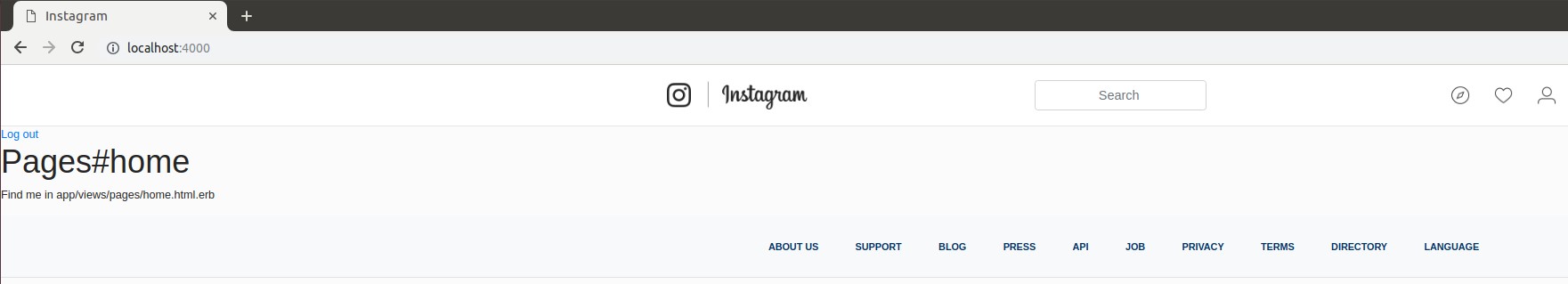
app/controllers/user\_controller.rb und erfassen eine show-Methode welche einen bestimmten User mittels Attribut id findet und die Variable @user speichert und der showView zur Verfügung stellt. Notieren Sie die show-Methode:

|  |
| --- |
| class UsersController < ApplicationController  def show  @user = User.find(params[:id])  end  end |
|  |
|  |
|  |
|  |

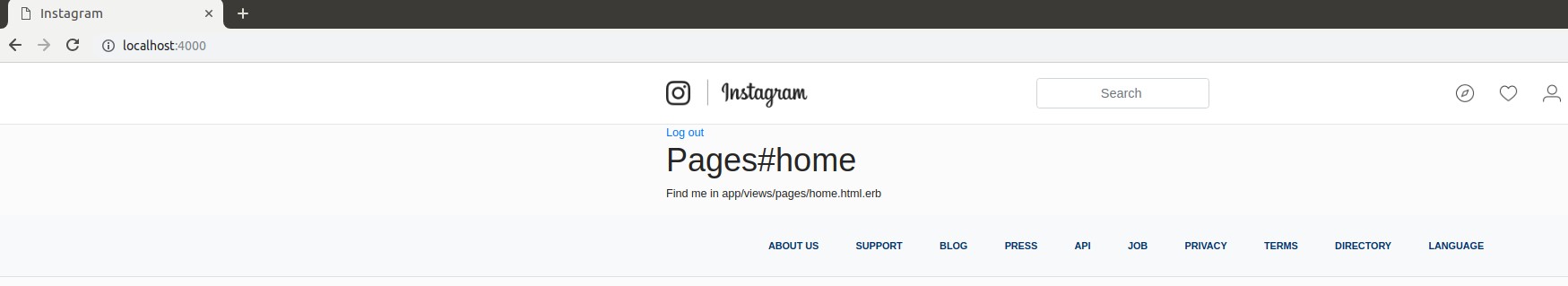
Da unserer View noch ein wenig unordentlich aussieht, wrappen wir den Inhalt von yield in app/views/layouts/application.html.rb in einen Container:



Ohne container:



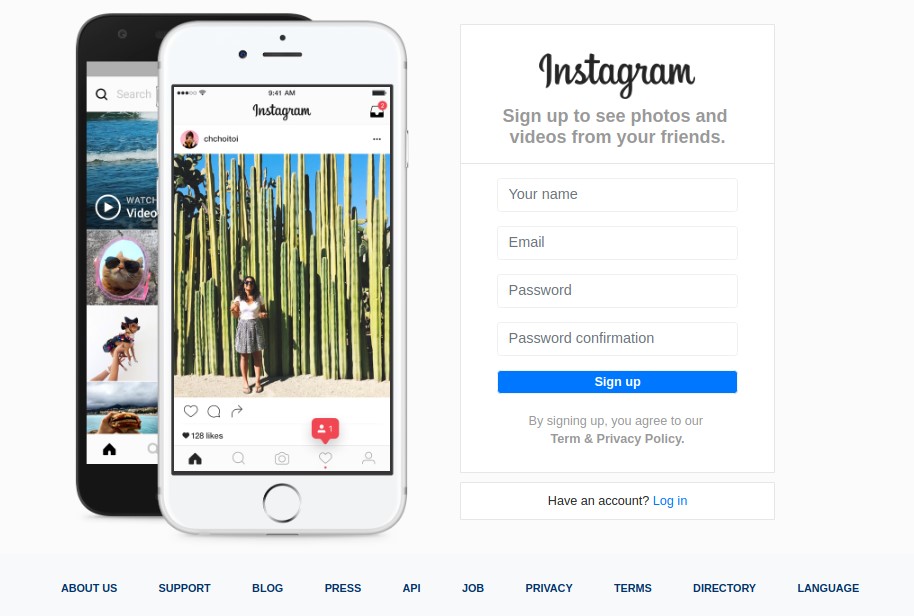
Mit container:



## show-View mit Bootstrap Modal

Erstellen Sie eine manuelle View app/views/users/show.html.erb. In dieser View nutzen wir Bootstrap Modal. Mit Modal können wir fancy lightboxes, notifications oder Dialogboxen bauen. Alle Informationen zu Bootstrap Modal finden Sie unter: https://getbootstrap.com/docs/4.0/components/modal/

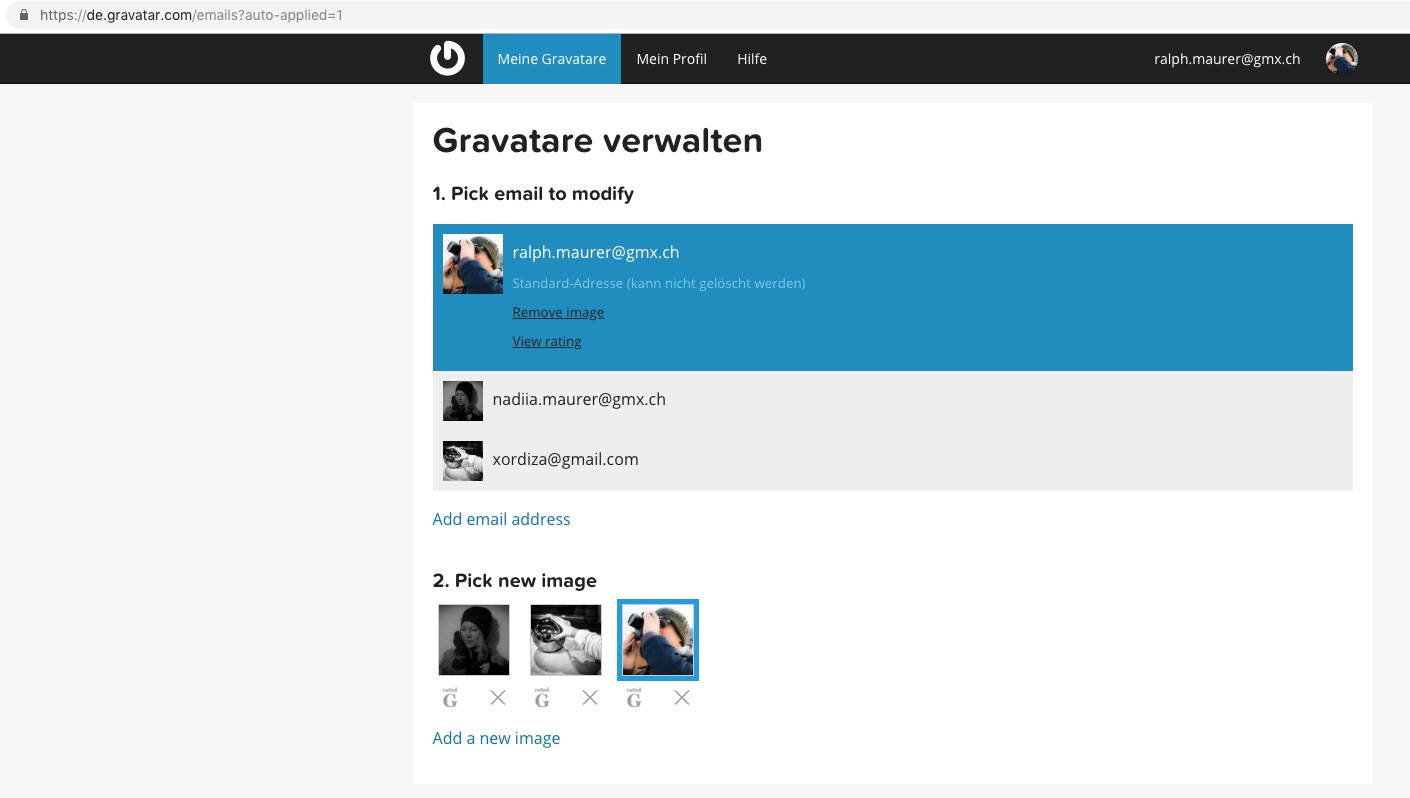
Stellen Sie nun sicher, dass mindestens zwei besser drei Benutzer in unserer Insta-App erfasst sind:



Die Benutzer müssen wir nun mit einem Profilbild versehen und dazu nutzen wir den Dienst von Gravatar.

## Gravatar für User Photo in show-View

Gravatar ist ein Dienst, der Emailadressen mit Userbildern assoziiert. Registrieren Sie sich auf https://de.gravatar.com/ und erfassen Sie die zwei Email Adressen der erfassten Benutzer in unserem Instagram. Fügen Sie geeignete Userbilder zu den Emailadressen hinzu. Gravatar verlangt, dass Sie die Profilbilder nach Rudeness raten. Verzichten Sie auf Obszonitäten oder andere unpassende Bildinhalte.



Damit wir das Bild aus Gravatar anzeigen können, müssen wir einen Applicationhelper in Rails schreiben. Öffnen Sie herzu app/helpers/application\_helper.rb und erstellen Sie folgende Methode:

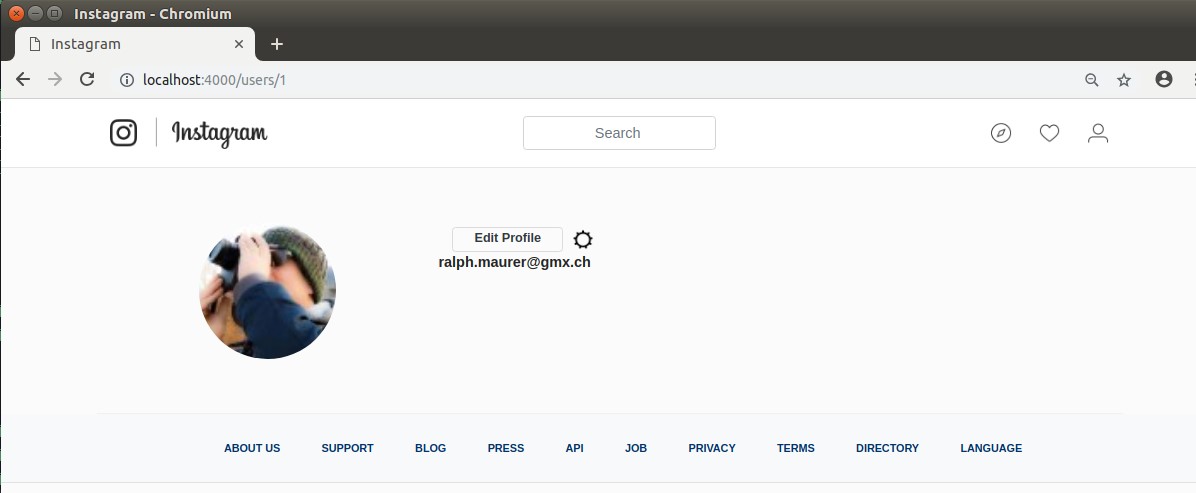


Erklären Sie in groben Zügen, was Sie geschrieben haben:

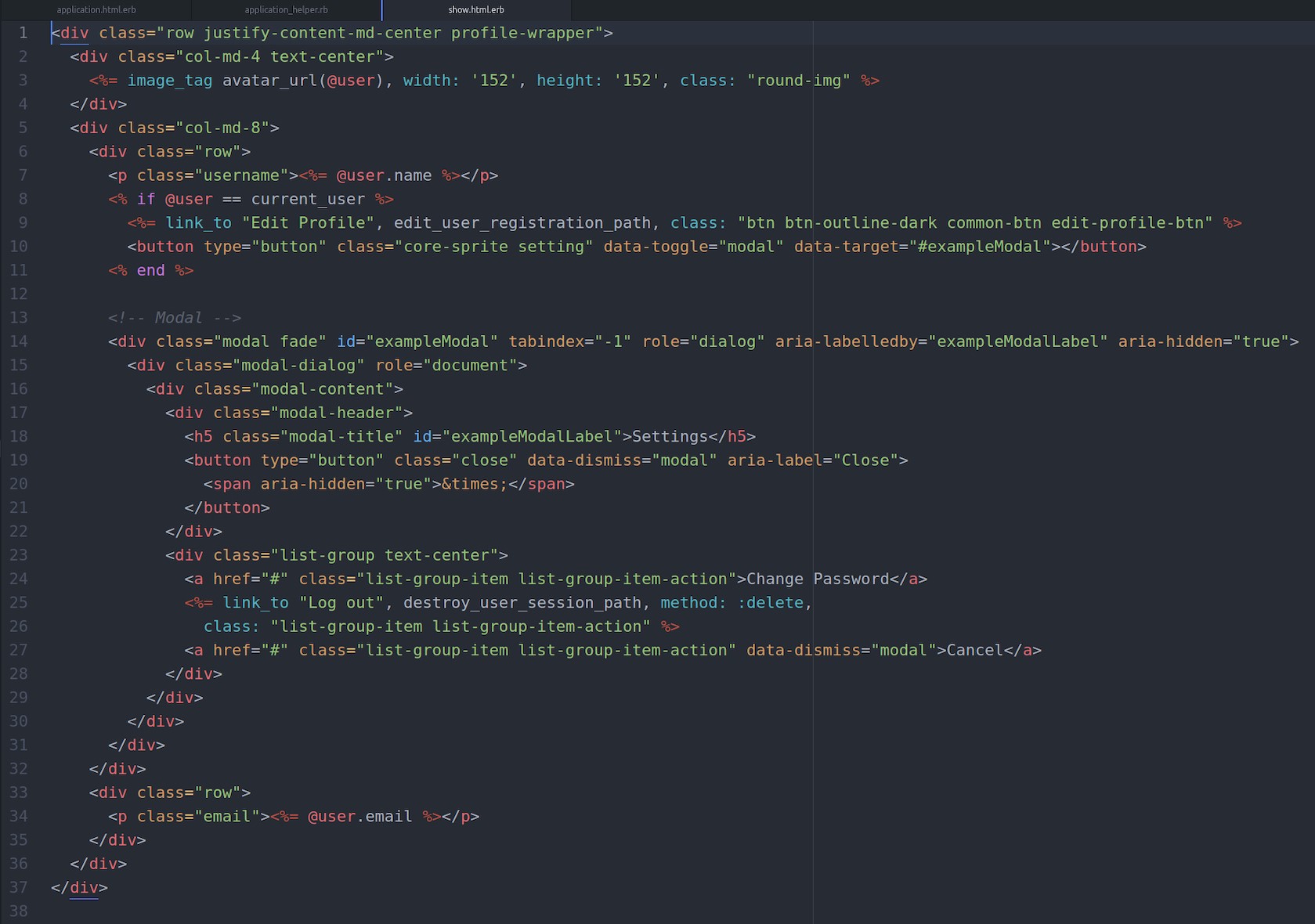
|  |
| --- |
|  |
|  |
|  |
|  |
|  |

Als nächstes muss einiges an HTML und ERB geschrieben werden.

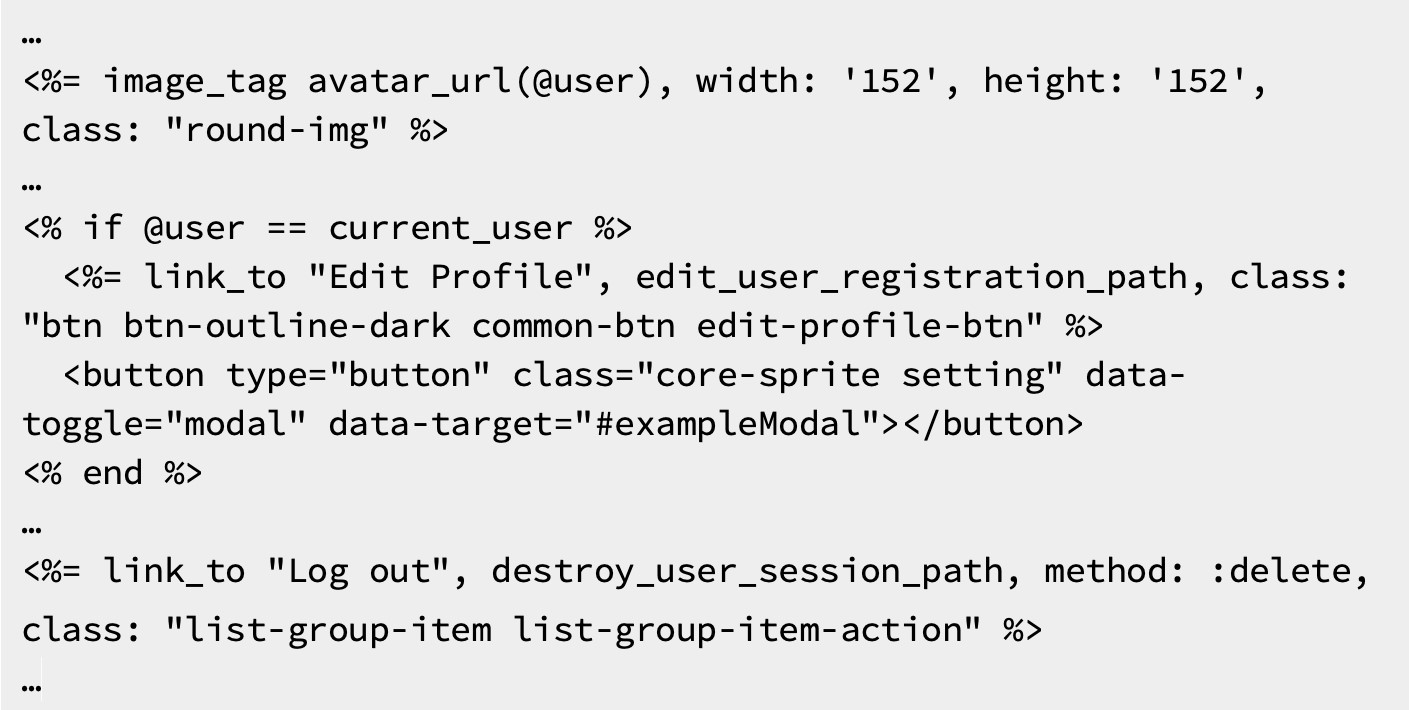
Ziel ist es, das die show-View wie folgt aussieht:



Beachten Sie das der Bereich Modal aus https://getbootstrap.com/docs/4.0/components/modal/ kopiert wurde und die CSS-Klassen leicht angepasst wurden.



Erklären Sie kurz folgende Codeausschnitte:



|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Wo und wie können Sie Informationen zu dem 2. Parameter von link\_to finden? Im Folgenden ist der 2. Parameter

Seite 11/18 edit\_user\_registration\_path und destroy\_user\_session\_path

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| …  <% if @user == current\_user %>   |  | | --- | | edit\_user\_registration\_path |   <%= link\_to "Edit Profile", , class:  "btn btn-outline-dark common-btn edit-profile-btn" %> <button type="button" class="core-sprite setting" datatoggle="modal" data-target="#exampleModal"></button>  <% end %>  …   |  | | --- | | destroy\_user\_session\_path |   <%= link\_to "Log out", , method: :delete, class: "list-group-item list-group-item-action" %> | | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

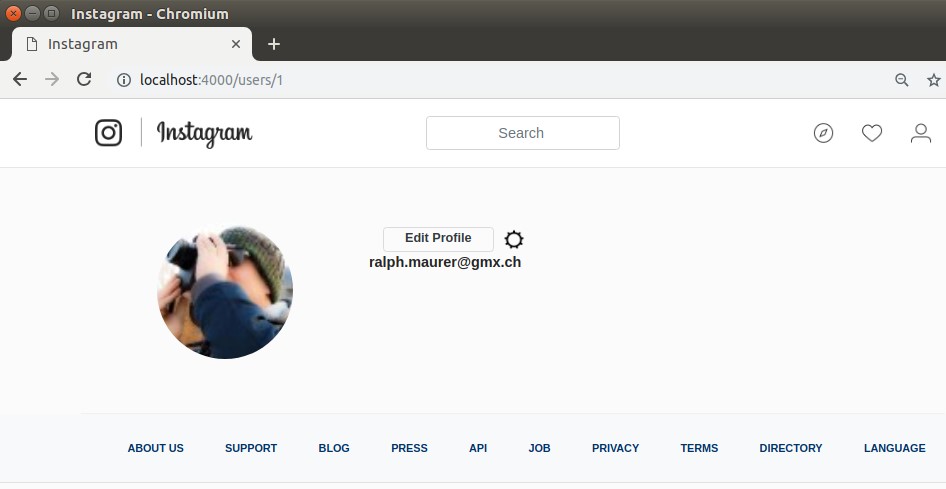
Wir bauen die benötigten SCSS-Klassen in

app/assets/stylesheets/applications.scss

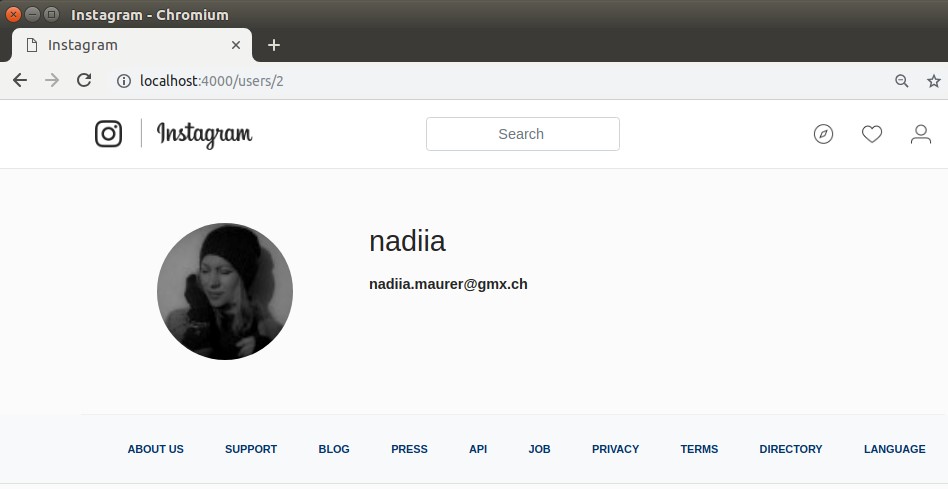


Wenn alles fehlerfrei gemacht wurde, sollten Sie nun folgendes Resultat mit den erfassten Benutzern prüfen können.

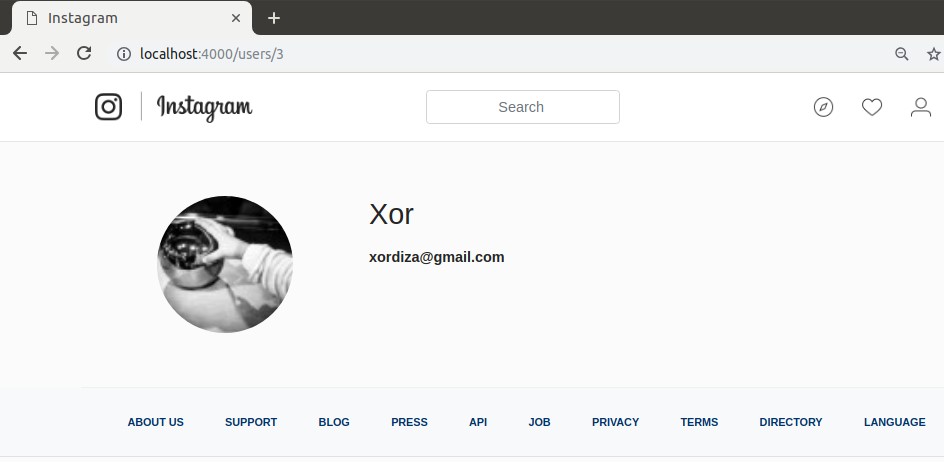
1. Benutzer (eingeloggt): http://localhost:4000/users/1



1. Benutzer: http://localhost:4000/users/2

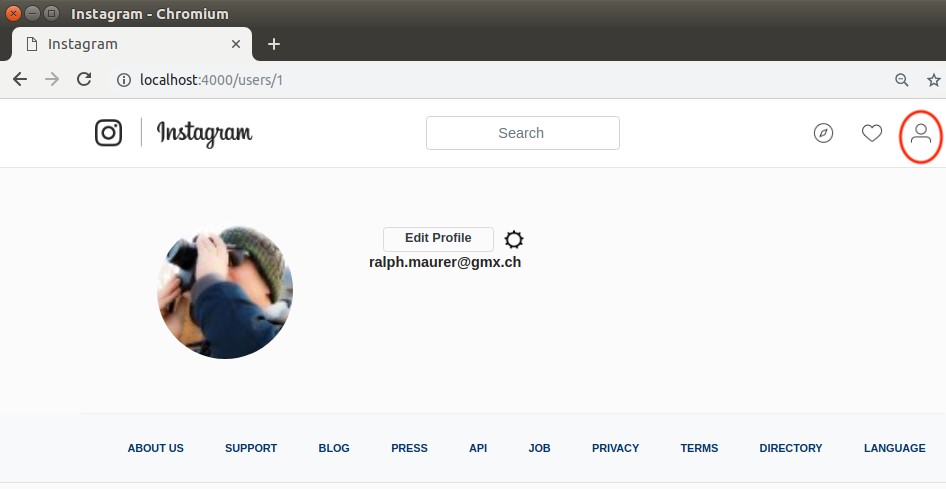


1. Benutzer: http://localhost:4000/users/3

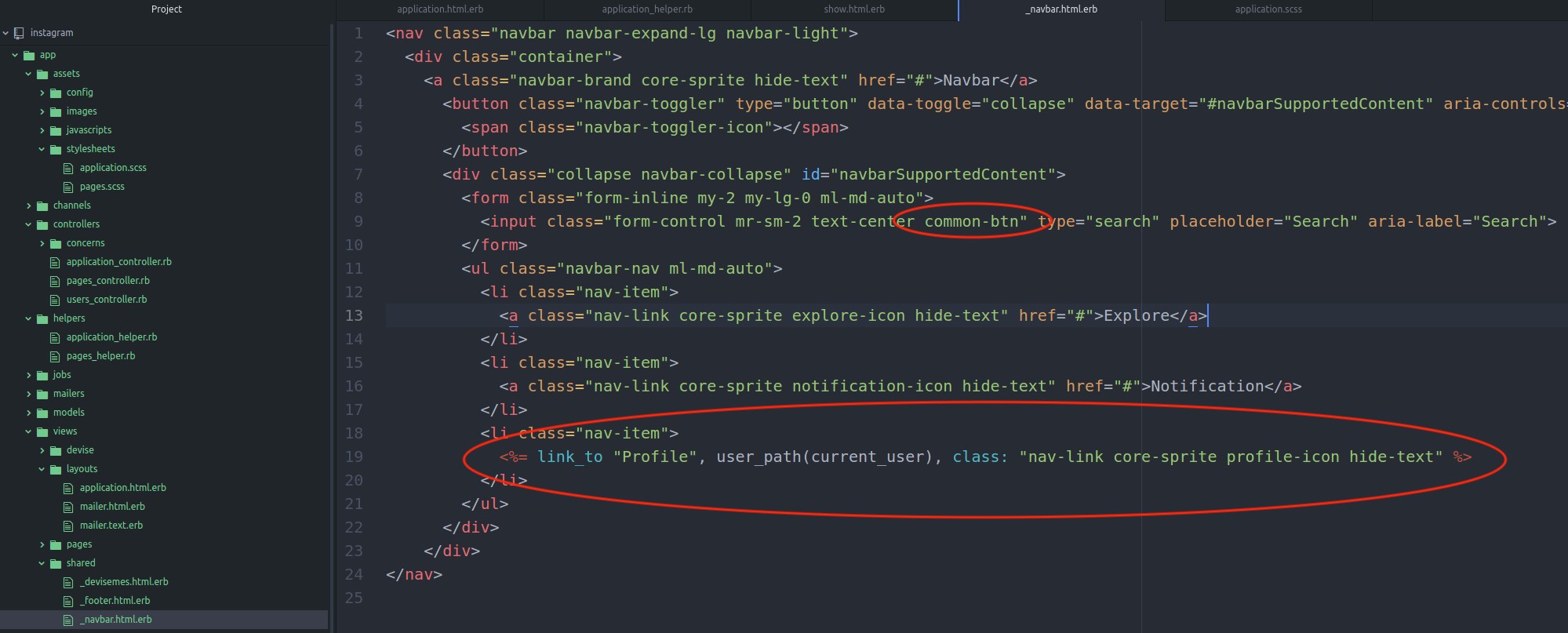


## link\_to Profile Page

Als nächstes wollen wir, dass der Link Profile Page in der Navbar funktioniert:

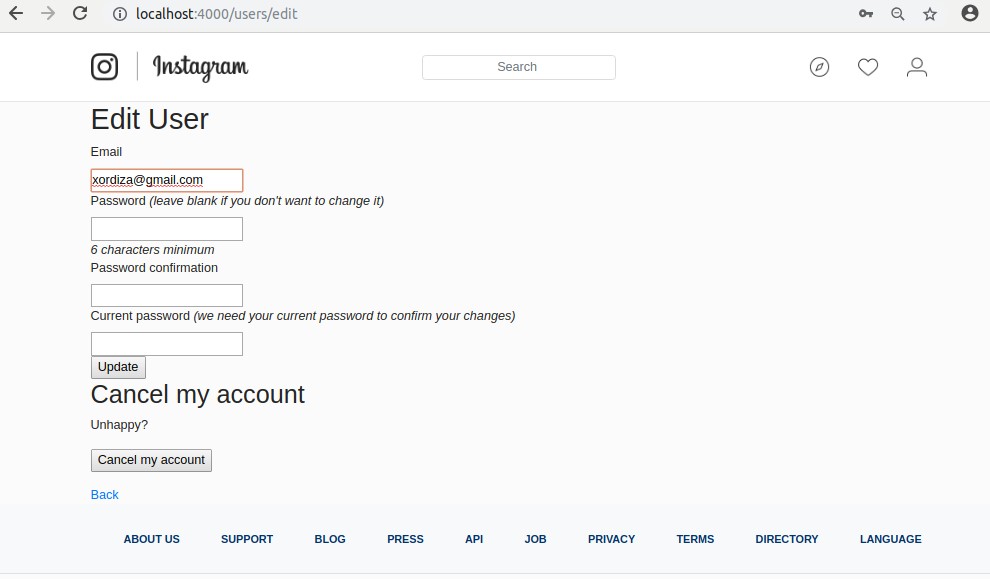


Die Datei -navbar.html.erb wird wie folgt angepasst:



## Edit User Page

Ein User-Profile soll einfach veränderbar sein. Devise hat uns eine Vorlage unter app/views/devise/registrations/edit.html.erb geliefert:



Als erstes ergänzen Sie den application\_controller.rb wie folgt:

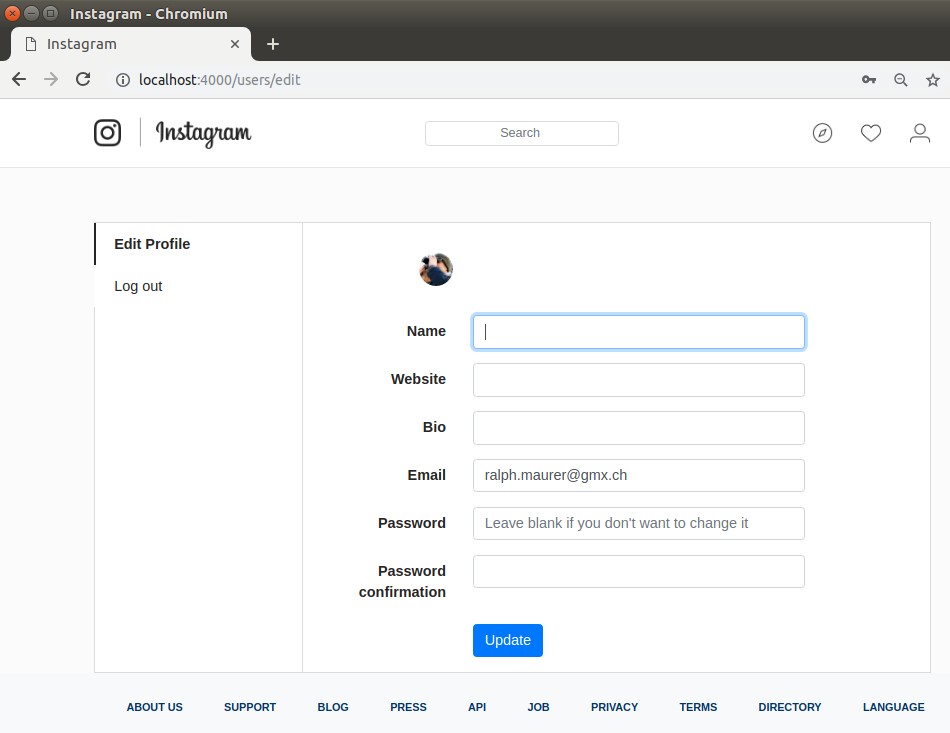
|  |  |  |  |
| --- | --- | --- | --- |
| devise\_parameter\_sanitizer.permit(:account\_update, keys: [:name, | |  | | --- | | :bio, :website | | ]) |

Ergänzen Sie app/views/devise/registrations/edit.html.erb und app/assets/stylesheets/application.scss

Die Vorlagen finden Sie unter sh-modules/iet-151/03\_Arbeitsblaetter/AB15104\_edit.html.rb sowie:

|  |
| --- |
| .edit-profile-wrapper { margin: 60px 0 0;    .left-col { border: 1px solid #dbdbdb; padding: 0;  background-color: white;    .list-group-item { border: none; color: #262626; border-radius: 0; font-size: 16px;  border-left: 2px solid transparent;  }    .list-group-item:first-child { font-weight: bold; border-left-color: #262626;  } |
| .list-group-item:not(:first-child):hover { border-left-color: #dbdbdb;  }  }    .right-col {  border: 1px solid #dbdbdb; border-left: none; background-color: white;    .username-img { margin: 32px 0;    .col-sm-9 { font-size: 20px; font-weight: 400; line-height: 20px;  }  }    .form-group { font-size: 16px;  }  }  } |

Damit erhalten wire in schönes und anschauliches Resultat:

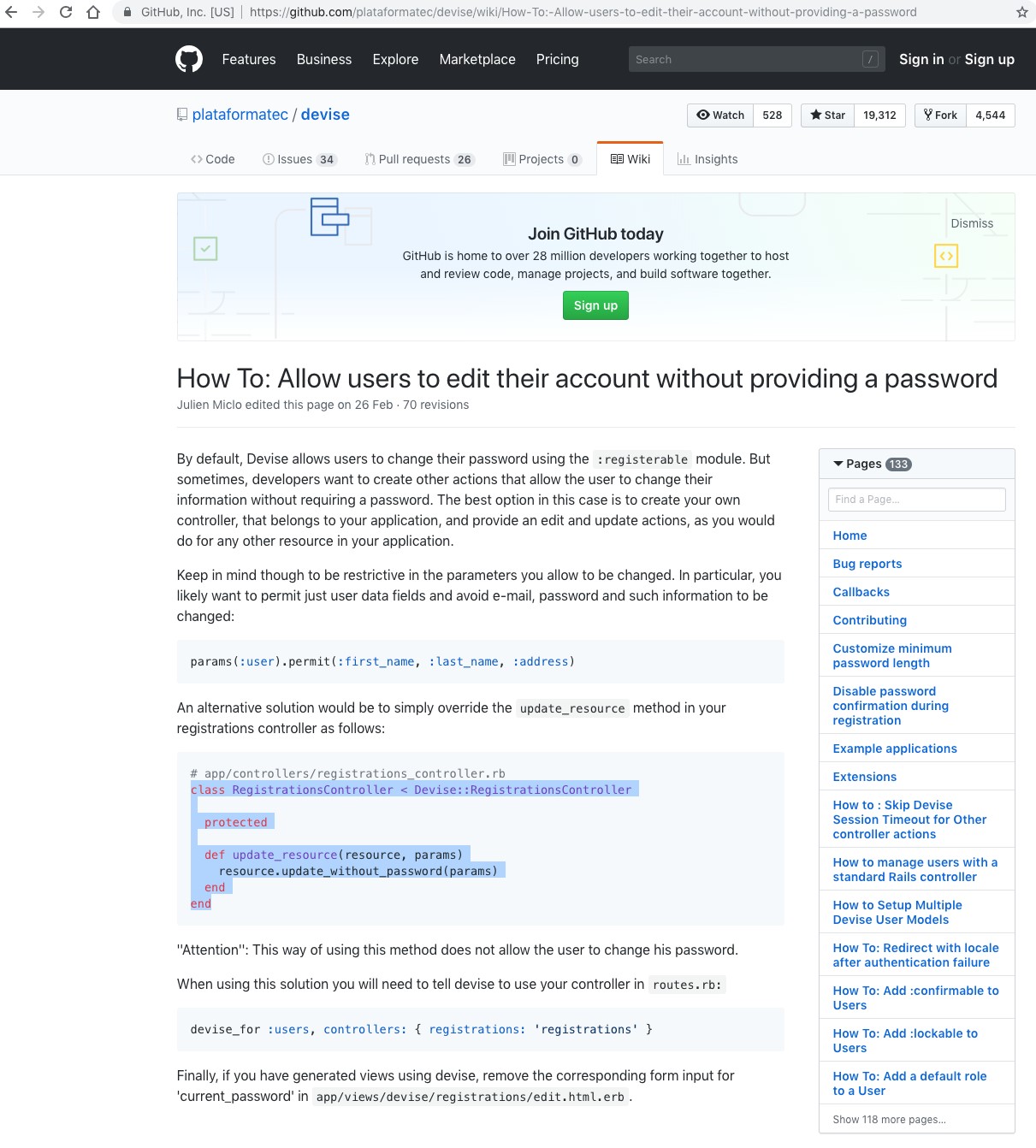


Versuchen Sie nun etwas zu verändern. Was stellen Sie fest:

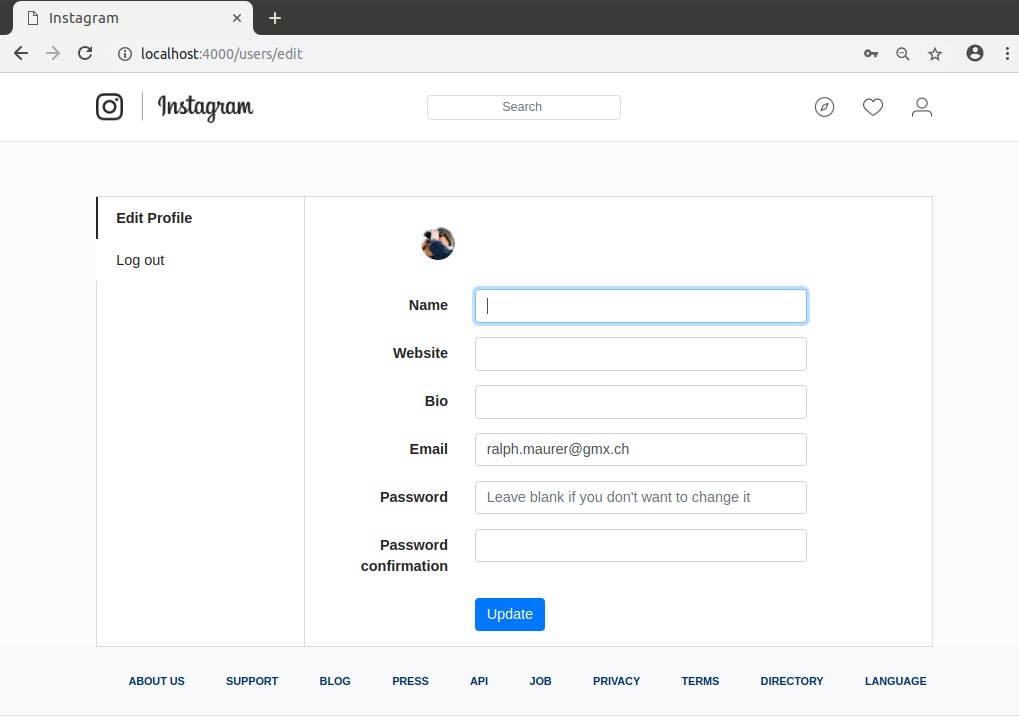
|  |
| --- |
|  |
|  |

Lösen Sie das Problem mit Hilfe von:

https://github.com/plataformatec/devise/wiki/How-To:-Allow-users-to-edit-their-accountwithout-providing-a-password



Das Editieren der Profil Page muss funktionieren 



# Auftrag: Quicknote AB151-04

Alle Aufträge des Typen Quicknote sind Bestandteil der Bewertung. Erstelle Sie eine Quicknote von max. 4 A4-Seiten und geben Sie diese in PDF Form gemäss Zeitangabe der Lehrperson ab. Angedacht sind max. 4 Lektionen seit Abgabe dieses Dokumentes.

Titel der Quicknote: *Klasse\_Name\_Vorname\_QN\_AB151-04.zip* Das Archiv beinhaltet:

› **Quichknote:** Klasse\_Name\_Vorname\_QN\_AB151-04.pdf

› **Code:** show.html.erb, application.html.erb, application.scss, application\_helper.rb, registrations\_controller.rb, users\_controller.rb, edit.html.rb, routes.rb

Zusammenfassung AB151-04

Die Quicknote soll eine kurze prägnante Zusammenfassung des Dokuments AB151-04 beinhalten und einen Überblick über die vorgestellten Techniken, Methoden und Konzepte beinhalten.

Achten Sie darauf, dass alle wesentliche Themen vollständig erwähnt und wenn möglich in Zusammenhang gebracht sind.

Beantworten Sie folgende Fragen zu Anwendungszweck und Selbstreflexion:

Anwendungszweck:

1. Wie und wo können die vorgestellten Techniken, Methoden und Konzepte in einer Rails-App angewandt werden?
2. Was sind Vorteile und was sind Nachteile?

Selbstreflexion:

1. Was habe ich gelernt?
2. Was hat mich behindert?
3. Was habe ich nicht verstanden?
4. Was kann ich beim Studium besser machen?

Abschliessende Reflexion über das Gelernte:

1. Schreiben Sie eine persönliche Schlussfolgerung über den Lerninhalt in 2-3 Sätzen.