Table of Contents

Rails Training

- https://github.com/sadah/rails-training
- https://sadah.github.io/rails-training/
- https://sadah.github.io/rails-training/book.pdf

Install

install

```
npm instal
```

for pdf / ebook

```
brew install Caskroom/cask/calibre
```

Usage

```
npm start # http://localhost:4000
```

Build

```
npm run build
```

Tests

```
npm test
```

Acknowledgment

I used these documents as reference. I would like to express my gratitude to these documents.

- Ruby on Rails チュートリアル：実例を使って Rails を学ぼう
- Rails Girls - Japanese
- Railsドキュメント

はじめに

Rails Training

- https://github.com/sadah/rails-training
- https://sadah.github.io/rails-training/
- https://sadah.github.io/rails-training/book.pdf

Ruby on Rails入門

- 環境構築
- Ruby on Rails基礎

Ruby on Rails 環境構築

この手順は OS X 10.8, 10.9, 10.10 を対象としています。

Xcode インストール

App Store から Xcode をインストールする。一度 Xcode を起動し、ライセンス使用許諾契約に同意する。

Command line tools インストール

Command line tools をインストールする。

```
xcode-select --install
```

Homebrew インストール

Homebrew をインストールする。

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew
```

rbenv インストール

rbenv をインストールする。

```
brew update
brew install rbenv rbenv-gem-rehash ruby-build
```

bash を利用している場合、以下のコマンドで環境変数を設定する。zshなどの場合、適宜書き換えて実行する。

```
echo 'eval "$(rbenv init -)"' >> ~/.bash_profile
echo 'export PATH="$HOME/.rbenv/shims:$PATH"' >> ~/.bash_profile
source ~/.bash_profile
```

Ruby インストール

Rubyをインストールする

```
rbenv install 2.2.3
```

デフォルトのRubyバージョンを指定する。

```
rbenv global 2.2.3
```

Ruby on Rails インストール

Ruby on Rails をインストールする。

```
gem install rails --no-ri --no-rdoc
```

確認

以下のコマンドが正しく実行できれば、Railsのインストールは正しくできています。

```
rails new sample-project
```

その他

Windows環境はサポート対象外ですが、RailsInstaller が便利だと思います。

- https://s3.amazonaws.com/railsinstaller/Windows/railsinstaller-3.1.0.exe

Ruby on Rails 基礎

Railsアプリケーションの作成

Railsアプリケーションを作成します。

```
rails new first-app
```

Railsアプリケーションは以下のような構成になっています。

```
first-app
├── Gemfile # ライブラリを管理するファイル
├── Gemfile.lock
├── README.rdoc
├── Rakefile
├── app # アプリケーションのコードが配置される
│   ├── assets
│   │   ├── images
│   │   ├── javascripts
│   │   │   └── application.js
│   │   └── stylesheets
│   │       └── application.css
│   ├── controllers
│   │   ├── application_controller.rb
│   │   └── concerns
│   ├── helpers
│   │   └── application_helper.rb
│   ├── mailers
│   ├── models
│   │   └── concerns
│   └── views
│       └── layouts
│           └── application.html.erb
├── bin
```

```
│   ├── bundle
│   ├── rails
│   ├── rake
│   ├── setup
│   └── spring
├── config # アプリケーションの設定ファイル
│   ├── application.rb
│   ├── boot.rb
│   ├── database.yml
│   ├── environment.rb
│   ├── environments
│   │   ├── development.rb
│   │   ├── production.rb
│   │   └── test.rb
│   ├── initializers
│   │   ├── assets.rb
│   │   ├── backtrace_silencers.rb
│   │   ├── cookies_serializer.rb
│   │   ├── filter_parameter_logging.rb
│   │   ├── inflections.rb
│   │   ├── mime_types.rb
│   │   ├── session_store.rb
│   │   └── wrap_parameters.rb
│   ├── locales
│   │   └── en.yml
│   ├── routes.rb
│   └── secrets.yml
├── config.ru
├── db
│   └── seeds.rb
├── lib
│   ├── assets
│   └── tasks
├── log
├── public # 静的なファイル
│   ├── 404.html
│   ├── 422.html
│   ├── 500.html
```

```
│   ├── favicon.ico
│   └── robots.txt
├── test
│   ├── controllers
│   ├── fixtures
│   ├── helpers
│   ├── integration
│   ├── mailers
│   ├── models
│   └── test_helper.rb
├── tmp
│   └── cache
│       └── assets
└── vendor
    └── assets
        ├── javascripts
        └── stylesheets
```

Railsアプリケーションを起動します。

```
cd first-app
rails server
```

ここからさきでは、first-appのディレクトリをRAILS_ROOTと呼びます。

scaffoldによるコードの自動生成

Railsではさまざまなコードを自動生成できます。scaffoldでアプリケーションの雛形を作ります。 scaffoldは一覧、詳細、追加、削除、変更のコードを自動生成します。

ここでは **bookmark** を管理する機能を作ります。RAILS__ROOTで以下のコメンドを実行します。

```
rails generate scaffold bookmark title:string description:text u
```

DBを更新して、アプリケーションを起動します。RAILS_ROOTで以下のコメンドを実行します。

```
bin/rake db:migrate
rails server
```

http://localhost:3000/bookmarksで動作確認ができます。

データの追加、削除、変更などを行ってみましょう。

scaffoldで生成されるファイル

scaffoldで生成されたファイルを確認していきます。scaffoldではこのような実行結果が出力されます。

```
% rails generate scaffold bookmark title:string description:text
      invoke  active_record
      create    db/migrate/20151129091144_create_bookmarks.rb
      create    app/models/bookmark.rb
      invoke    test_unit
      create      test/models/bookmark_test.rb
      create      test/fixtures/bookmarks.yml
      invoke  resource_route
       route    resources :bookmarks
      invoke  scaffold_controller
      create    app/controllers/bookmarks_controller.rb
      invoke    erb
      create      app/views/bookmarks
      create      app/views/bookmarks/index.html.erb
      create      app/views/bookmarks/edit.html.erb
      create      app/views/bookmarks/show.html.erb
      create      app/views/bookmarks/new.html.erb
      create      app/views/bookmarks/_form.html.erb
      invoke    test_unit
      create      test/controllers/bookmarks_controller_test.rb
      invoke    helper
      create      app/helpers/bookmarks_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/bookmarks/index.json.jbuilder
      create      app/views/bookmarks/show.json.jbuilder
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/bookmarks.coffee
      invoke    scss
      create      app/assets/stylesheets/bookmarks.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.scss
```

scaffoldでは以下のようなファイルを生成しています。

- active_record
  - テーブルの作成
  - モデルの作成
- routeの設定
- controller
  - viewの作成
  - helperの作成
  - jbuilder(jsonテンプレート)の作成
- assets
  - CoffeeScriptの作成
  - Scssの作成

実際にファイルを開いて確認して見ましょう。

migration

db/migrate/20151129091144_create_bookmarks.rb

```
class CreateBookmarks < ActiveRecord::Migration
  def change
    create_table :bookmarks do |t|
      t.string :title
      t.text :description
      t.string :url

      t.timestamps null: false
    end
  end
end
```

routes

config/routes.rb

```
Rails.application.routes.draw do
```

```
  resources :bookmarks
  # The priority is based upon order of creation: first created
  # See how all your routes lay out with "rake routes".

  # You can have the root of your site routed with "root"
  # root 'welcome#index'

  # Example of regular route:
  #   get 'products/:id' => 'catalog#view'

  # Example of named route that can be invoked with purchase_url
  #   get 'products/:id/purchase' => 'catalog#purchase', as: :pu

  # Example resource route (maps HTTP verbs to controller action
  #   resources :products

  # Example resource route with options:
  #   resources :products do
  #     member do
  #       get 'short'
  #       post 'toggle'
  #     end
  #
  #     collection do
  #       get 'sold'
  #     end
  #   end

  # Example resource route with sub-resources:
  #   resources :products do
  #     resources :comments, :sales
  #     resource :seller
  #   end

  # Example resource route with more complex sub-resources:
  #   resources :products do
  #     resources :comments
  #     resources :sales do
```

```
  #       get 'recent', on: :collection
  #     end
  #   end

  # Example resource route with concerns:
  #   concern :toggleable do
  #     post 'toggle'
  #   end
  #   resources :posts, concerns: :toggleable
  #   resources :photos, concerns: :toggleable

  # Example resource route within a namespace:
  #   namespace :admin do
  #     # Directs /admin/products/* to Admin::ProductsController
  #     # (app/controllers/admin/products_controller.rb)
  #     resources :products
  #   end
end
```

`rake routes` で定義されいているroutesを確認できます。

```
% rake routes
       Prefix Verb   URI Pattern                    Controller#Ac
    bookmarks GET    /bookmarks(.:format)           bookmarks#ind
              POST   /bookmarks(.:format)           bookmarks#cre
 new_bookmark GET    /bookmarks/new(.:format)       bookmarks#new
edit_bookmark GET    /bookmarks/:id/edit(.:format)  bookmarks#edi
     bookmark GET    /bookmarks/:id(.:format)       bookmarks#sho
              PATCH  /bookmarks/:id(.:format)       bookmarks#upd
              PUT    /bookmarks/:id(.:format)       bookmarks#upd
              DELETE /bookmarks/:id(.:format)       bookmarks#des
```

models

app/models/bookmark.rb

```
class Bookmark < ActiveRecord::Base
end
```

controllers

app/controllers/bookmarks_controller.rb

```
class BookmarksController < ApplicationController
  before_action :set_bookmark, only: [:show, :edit, :update, :de

  # GET /bookmarks
  # GET /bookmarks.json
  def index
    @bookmarks = Bookmark.all
  end

  # GET /bookmarks/1
  # GET /bookmarks/1.json
  def show
  end

  # GET /bookmarks/new
  def new
    @bookmark = Bookmark.new
  end

  # GET /bookmarks/1/edit
  def edit
  end

  # POST /bookmarks
  # POST /bookmarks.json
  def create
    @bookmark = Bookmark.new(bookmark_params)
```

```
    respond_to do |format|
      if @bookmark.save
        format.html { redirect_to @bookmark, notice: 'Bookmark w
        format.json { render :show, status: :created, location:
      else
        format.html { render :new }
        format.json { render json: @bookmark.errors, status: :ur
      end
    end
  end

  # PATCH/PUT /bookmarks/1
  # PATCH/PUT /bookmarks/1.json
  def update
    respond_to do |format|
      if @bookmark.update(bookmark_params)
        format.html { redirect_to @bookmark, notice: 'Bookmark w
        format.json { render :show, status: :ok, location: @book
      else
        format.html { render :edit }
        format.json { render json: @bookmark.errors, status: :ur
      end
    end
  end

  # DELETE /bookmarks/1
  # DELETE /bookmarks/1.json
  def destroy
    @bookmark.destroy
    respond_to do |format|
      format.html { redirect_to bookmarks_url, notice: 'Bookmark
      format.json { head :no_content }
    end
  end

  private
    # Use callbacks to share common setup or constraints betweer
```

```
    def set_bookmark
      @bookmark = Bookmark.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow
    def bookmark_params
      params.require(:bookmark).permit(:title, :description, :ur
    end
end
```

helpers

app/helpers/bookmarks_helper.rb

```
module BookmarksHelper
end
```

views

app/views/bookmarks/index.html.erb

```
<p id="notice"><%= notice %></p>

<h1>Listing Bookmarks</h1>

<table>
  <thead>
    <tr>
      <th>Title</th>
      <th>Description</th>
      <th>Url</th>
      <th colspan="3"></th>
    </tr>
  </thead>

  <tbody>
    <% @bookmarks.each do |bookmark| %>
      <tr>
        <td><%= bookmark.title %></td>
        <td><%= bookmark.description %></td>
        <td><%= bookmark.url %></td>
        <td><%= link_to 'Show', bookmark %></td>
        <td><%= link_to 'Edit', edit_bookmark_path(bookmark) %><
        <td><%= link_to 'Destroy', bookmark, method: :delete, da
      </tr>
    <% end %>
  </tbody>
</table>

<br>

<%= link_to 'New Bookmark', new_bookmark_path %>
```

app/views/bookmarks/_form.html.erb

```
<%= form_for(@bookmark) do |f| %>
  <% if @bookmark.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@bookmark.errors.count, "error") %> proh

      <ul>
      <% @bookmark.errors.full_messages.each do |message| %>
        <li><%= message %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :description %><br>
    <%= f.text_area :description %>
  </div>
  <div class="field">
    <%= f.label :url %><br>
    <%= f.text_field :url %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

app/views/bookmarks/index.json.jbuilder

```
json.array!(@bookmarks) do |bookmark|
  json.extract! bookmark, :id, :title, :description, :url
  json.url bookmark_url(bookmark, format: :json)
end
```

console

`rails console` で対話的に操作できます。RAILS_ROOTで以下のコメンドを実行します。

```
% rails console
Loading development environment (Rails 4.2.4)
irb(main):001:0>
```

`Bookmark.last` で最後に登録したBookmarkを取得します。

```
irb(main):001:0> Bookmark.last
  Bookmark Load (0.2ms)  SELECT  "bookmarks".* FROM "bookmarks"
=> #<Bookmark id: 1, title: "sadah", description: "My portfolio
irb(main):002:0>
```

このようにデータを作成することもできます。

```
irb(main):002:0> bookmark = Bookmark.create(title: "test", descr
   (4.0ms)  begin transaction
  SQL (5.1ms)  INSERT INTO "bookmarks" ("title", "description",
   (1.0ms)  commit transaction
=> #<Bookmark id: 2, title: "test", description: "test bookmark"
irb(main):003:0>
```

Active Recordのクエリについてはこちらがわかりやすいです。

- Active Record クエリインターフェイス | Rails ガイド

debug

pryはirb(標準のRubyコンソール)に代わる、パワフルなコンソールです。

- pry/pry

Gemfileにpryのgemを追加します。

```
group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get
  gem 'byebug'
  gem 'pry'
  gem 'pry-doc'
  gem 'pry-byebug'
  gem 'pry-rails'
  gem 'awesome_print'
end
```

bundle installで、gemをインストールします。

```
bundle install
```

Railsサーバが起動済みであれば、一度停止してください。

Railsサーバを起動すると、pryが有効になります。

```
rails s
```

それぞれのgemはこのような機能を持ちます。

gem	
pry	pry本体
pry-doc	ドキュメントやメソッドを表示する
pry-byebug	デバッグをできるようにする
pry-rails	Railsのコンソールをpryにする
awesome_print	データ構造などをわかりやすく表示する

Railsコンソールを立ち上げると、pryが起動していることがわかります。
RAILS_ROOTで以下のコメンドを実行します。

```
% rails c
Loading development environment (Rails 4.2.4)
[1] pry(main)>
```

show-docでドキュメントが表示されます。

```
[1] pry(main)> show-doc String.nil?

From: object.c (C Method):
Owner: Kernel
Visibility: public
Signature: nil?()
Number of lines: 4

Only the object nil responds true to nil?.

   Object.new.nil?   #=> false
   nil.nil?          #=> true
```

show-methodでメソッドが表示されます。

```
[2] pry(main)> show-method String.nil?

From: object.c (C Method):
Owner: Kernel
Visibility: public
Number of lines: 5

static VALUE
rb_false(VALUE obj)
{
    return Qfalse;
}
```

pryでデバッグしていきます。gemを追加したのでrailsを再起動します。

app/controllers/bookmarks_controller.rb に `binding.pry` を追加します。

```
def index
  @bookmarks = Bookmark.all
  binding.pry
end
```

http://localhost:3000/bookmarks を表示します。railsを起動したターミナルでデバッグができます。

```
% rails s
=> Booting WEBrick
=> Rails 4.2.4 application starting in development on http://loc
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
[2015-12-04 07:43:32] INFO  WEBrick 1.3.1
[2015-12-04 07:43:32] INFO  ruby 2.2.3 (2015-08-18) [x86_64-darw
[2015-12-04 07:43:32] INFO  WEBrick::HTTPServer#start: pid=35325


Started GET "/bookmarks" for ::1 at 2015-12-04 07:43:42 +0900
  ActiveRecord::SchemaMigration Load (0.5ms)  SELECT "schema_mig
Processing by BookmarksController#index as HTML

From: /Users/sada/git/gs/first-app/app/controllers/bookmarks_cor

6: def index
7:   @bookmarks = Bookmark.all
8:   binding.pry
=> 9: end
```

インスタンス変数などを表示できます。

```
[1] pry(#<BookmarksController>)> @bookmarks
  Bookmark Load (1.2ms)  SELECT "bookmarks".* FROM "bookmarks"
[
    [0] #<Bookmark:0x007feb883cbfb0> {
                  :id => 1,
               :title => "sadah",
         :description => "My portfolio site.",
                 :url => "https://sadah.github.io",
          :created_at => Wed, 02 Dec 2015 00:26:31 UTC +00:00,
          :updated_at => Wed, 02 Dec 2015 00:26:31 UTC +00:00
    },
    [1] #<Bookmark:0x007feb883cbc90> {
                  :id => 2,
               :title => "test",
         :description => "test bookmark",
                 :url => "http://exapmle.com",
          :created_at => Thu, 03 Dec 2015 00:04:59 UTC +00:00,
          :updated_at => Thu, 03 Dec 2015 00:04:59 UTC +00:00
    }
]
```

helpと入力するとpryの使い方が表示されます。

~/.pryrc にこんな感じの設定を書いておくと便利です。

```
# http://qiita.com/Linda_pp/items/d75d7c3953faa34a1f0e
begin
  require "awesome_print"
  AwesomePrint.pry!
rescue LoadError => err
  puts "no awesome_print :("
end

# http://morizyun.github.io/blog/pry-tips-rails-ruby/
# pry-debuggerのショートカット
Pry.commands.alias_command 'c', 'continue'
Pry.commands.alias_command 's', 'step'
Pry.commands.alias_command 'n', 'next'
```

またエラーが発生した場合、エラーが表示された画面でデバッグを行うこともできます。

課題

今週の課題です。

- link_toを使ってURLをリンクにしてみましょう。
- タイトルとURLを必須項目にしましょう。

さらに頑張りたい方はbookmarkにコメントできるようにしましょう。
ざっくりとした流れはこんな感じです。

- comment(user_name, content, bookmark_id)をscaffoldで作成する
- has_many, belongs_toで関連をつける
- bookmark#showでコメントを表示する
- bookmark#showでコメントを新規登録できるようする

アプリケーション開発 1

今回からは、写真を投稿できるアプリケーションを実装していきます。

アプリケーション開発 1

Railsアプリケーションの作成

Railsアプリケーションを作成します。ターミナルで以下のコマンドを実行します。

```
rails new photolog
cd photolog
```

scaffoldでphotoに関するファイルを作成します。

image、captionというカラムをもたせます。imageには画像ファイルのURLを入れます。ターミナルで以下のコマンドを実行します。

```
rails g scaffold photo image:string caption:text
rake db:migrate
```

サーバーを起動して、正しく動作するか確認してみましょう。ターミナルで以下のコマンドを実行します。

```
rails s
```

課題

- 前回導入したPryを、このRailsアプリケーションにも導入しましょう
- image と caption を必須項目にしましょう

画像アップロード(CarrierWave)

CarrierWaveというgemを使うと、簡単に画像のアップロードが実装できます。

- carrierwaveuploader/carrierwave

エディタでGemfileを編集します。この行の後に

```
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc
```

この1行を追加します。

```
gem 'carrierwave'
```

`bundle install` を実行してgemをインストールします。ターミナルで以下のコマンドを実行します。

```
bundle install
```

画像をアップロードするためのコードを生成します。ターミナルで以下のコマンドを実行します。

```
rails generate uploader Image
```

エディタでapp/models/photo.rbを編集します。この行の後に

```
class Photo < ActiveRecord::Base
```

この行を追加します。

```
  mount_uploader :image, ImageUploader
```

画像ファイルをアップロードできるようにします。エディタで app/views/photos/_form.html.erbを編集します。

この行を

```
<%= f.text_field :image %>
```

このように変更します。

```
<%= f.file_field :image %>
```

画像を表示できるようにします。エディタで app/views/photos/show.html.erbを編集します。

この行を

```
<%= @photo.image %>
```

このように変更します。

```
<%= image_tag(@photo.image_url) if @photo.image.present? %>
```

サーバーを再起動します。サーバーすでに起動している場合は Ctrl + c で停止してください。

サーバーを起動してください。ターミナルで以下のコマンドを実行します。

```
rails s
```

サーバーが起動したら、写真のアップロードを試してみてください。

デザインの適用(Bootstrapの導入)

Bootstrapを導入して、デザインを適用していきます。

- Bootstrap · The world's most popular mobile-first and responsive front-end framework.

レイアウトファイルの編集

エディタで app/views/layouts/application.html.erb を編集します。この行の後に

```
<title>Photolog</title>
```

この2行を追加します。

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/boo
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/boo
```

この行の前後を

```
<%= yield %>
```

このように変更します。

```
<div class="container">
  <%= yield %>
</div>
```

この行の前に

```
</body>
```

この行を追加します

```
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/
```

show.html.erb の編集

エディタで `app/views/photos/show.html.erb` を編集します。

この行以降の内容を

```
<p id="notice"><%= notice %></p>
```

この内容に変更します。

```
<div class="row">
  <div class="col-lg-12">
    <%= image_tag(@photo.image_url) if @photo.image.present? %>
    <div>
      <h4><%= @photo.caption %></h4>
      <p>
        <%= link_to 'Edit', edit_photo_path(@photo) %>
        <%= link_to 'Destroy', @photo, method: :delete, data: {
      </p>
    </div>
  </div>
</div>
```

課題

- `app/views/photos/index.html.erb` でも画像ファイルを表示しましょう
- `app/views/photos/index.html.erb` でも表示を整えてみましょう。
- 画像のサイズを指定しましょう

さらに頑張りたい人はレスポンシブウェブデザインに挑戦しましょう。

- Media Queriesを使って、レスポンシブウェブデザインを適用しましょう
- 画面が一定サイズより小さい場合は、画像が横幅いっぱいに表示されるようにしましょう

画面が大きい場合はこのように表示され

画面が小さい場合はこのように表示されるようにしてみましょう。

photolog

Goodpatch 4th Anniversary!!

アプリケーション開発 2

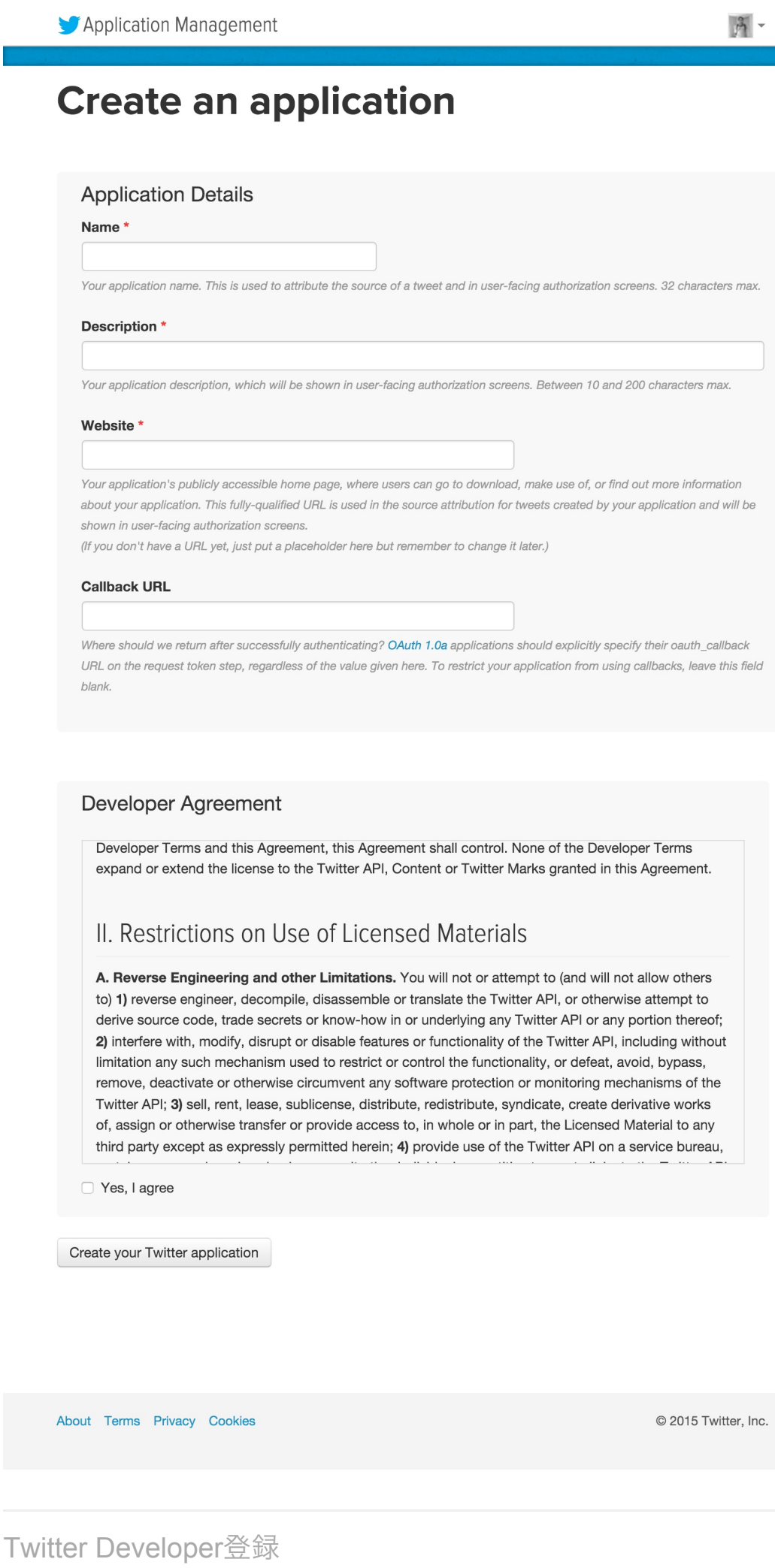
前回からは、引き続き写真を投稿できるアプリケーションを実装していきます。今回はTwitterログイン機能を実装します。

Twitter Developer登録

Twitter認証のログインを行うためアプリケーション登録を済ませておいてください。

Twitterにログインした状態で https://apps.twitter.com/ にアクセスします。

「Create New App」ボタンでアプリケーション登録を行います。


Application Management
Create an application
Application Details
Name *
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.
Description *
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.
Website *
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)
Callback URL
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.
Developer Agreement
Developer Terms and this Agreement, this Agreement shall control. None of the Developer Terms expand or extend the license to the Twitter API, Content or Twitter Marks granted in this Agreement.
II. Restrictions on Use of Licensed Materials
A. Reverse Engineering and other Limitations. You will not or attempt to (and will not allow others to) 1) reverse engineer, decompile, disassemble or translate the Twitter API, or otherwise attempt to derive source code, trade secrets or know-how in or underlying any Twitter API or any portion thereof; 2) interfere with, modify, disrupt or disable features or functionality of the Twitter API, including without limitation any such mechanism used to restrict or control the functionality, or defeat, avoid, bypass, remove, deactivate or otherwise circumvent any software protection or monitoring mechanisms of the Twitter API; 3) sell, rent, lease, sublicense, distribute, redistribute, syndicate, create derivative works of, assign or otherwise transfer or provide access to, in whole or in part, the Licensed Material to any third party except as expressly permitted herein; 4) provide use of the Twitter API on a service bureau,
Yes, I agree
Create your Twitter application
About Terms Privacy Cookies
© 2015 Twitter, Inc.

Application Detailsで、以下の内容を登録します。

- Name
- Description
- Website
- Callback URL

Twitter全体で、Nameは重複不可となっています。Descriptionは適当な文章で問題ありません。

Website / Callback URL には http://127.0.0.1:3000 を入力してください。

すべて入力したら利用規約に同意し、アプリケーションを作成してください。

認証機能(Devise)

ログインやユーザ管理の機能を持つDeviseと、Twitter認証の機能を持つOmniAuth Twitterを利用して、Twitter認証の機能を実装します。

実装の流れは以下のようになります。

1.DeviseとOmniAuth Twitterの導入 2.Userモデルの作成 3.認証処理の実装 4.Twitter認証の設定

Deviseの導入

DeviseとOmniAuth Twitterのgemをインストールします。

- plataformatec/devise
- arunagw/omniauth-twitter

Gemfileをエディタで開き、この行の後に

```
gem 'carrierwave'
```

この行を追加します。

```
gem 'devise'
gem 'omniauth-twitter'
```

gemをインストールします。RAILS_ROOTでbundle installを実行します。

```
bundle install
```

Deviseのファイルを生成します。

```
rails g devise:install
```

実行するとこのようなメッセージが表示されます。

```
% rails g devise:install
      create  config/initializers/devise.rb
      create  config/locales/devise.en.yml
===============================================================

Some setup you must do manually if you haven't yet:

  1. Ensure you have defined default url options in your environ
     is an example of default_url_options appropriate for a deve
     in config/environments/development.rb:

       config.action_mailer.default_url_options = { host: 'local

     In production, :host should be set to the actual host of yo

  2. Ensure you have defined root_url to *something* in your con
     For example:

       root to: "home#index"

  3. Ensure you have flash messages in app/views/layouts/applica
     For example:

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

  4. If you are deploying on Heroku with Rails 3.2 only, you may

       config.assets.initialize_on_precompile = false

     On config/application.rb forcing your application to not ac
     or load models when precompiling your assets.

  5. You can copy Devise views (for customization) to your app b

       rails g devise:views
```

```
==========================================================
```

メッセージにしたがって、必要な部分の対応を行います。

今回は利用しませんが、Deviseの機能を使うとユーザ作成や認証時にメールを送信できます。この機能を利用するための設定を追加します。

config/environments/development.rbをエディタで開きます。この行の後に

```
# Don't care if the mailer can't send.
config.action_mailer.raise_delivery_errors = false
```

この行を追加します。

```
config.action_mailer.default_url_options = { host: 'localhost',
```

root_url(開発環境の場合は http://localhost:3000)にアクセスした時に表示されるページを指定します。今回は /photos にリダイレクトさせます。

config/routes.rbをエディタで開きます。この行のあとに

```
Rails.application.routes.draw do
```

この行を追加します。

```
  root to: redirect('/photos')
```

認証した時のメッセージを表示できるようにします。

app/views/layouts/application.html.erbをエディタで開きます。この行のあとに

```
<div class="container">
```

この行を追加します。

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

メッセージが重複して表示されてしまうため、不要になったコードを削除します。

app/views/photos/index.html.erb をエディタで開きます。この行を削除します。

```
<p id="notice"><%= notice %></p>
```

4と５の対応はいまは必要ないのでなにもしません。

Userモデルの作成

認証で利用するUserモデルを自動生成します。

```
rails g devise User
```

コードが自動生成されます。

```
rails g devise User
      invoke  active_record
      create    db/migrate/20151216140212_devise_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
      insert    app/models/user.rb
       route  devise_for :users
```

マイグレーションファイルはこのような内容になっています。Deviseにはアカウントロックなどの機能があり、それらに必要なカラムが定義されています。

db/migrate/20151216140212_devise_create_users.rb

```
class DeviseCreateUsers < ActiveRecord::Migration
  def change
    create_table(:users) do |t|
      ## Database authenticatable
      t.string :email,              null: false, default: ""
      t.string :encrypted_password, null: false, default: ""

      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at

      ## Rememberable
      t.datetime :remember_created_at

      ## Trackable
      t.integer  :sign_in_count, default: 0, null: false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.string   :current_sign_in_ip
      t.string   :last_sign_in_ip
```

```
      ## Confirmable
      # t.string   :confirmation_token
      # t.datetime :confirmed_at
      # t.datetime :confirmation_sent_at
      # t.string   :unconfirmed_email # Only if using reconfirma

      ## Lockable
      # t.integer  :failed_attempts, default: 0, null: false # C
      # t.string   :unlock_token # Only if unlock strategy is :e
      # t.datetime :locked_at


      t.timestamps null: false
    end

    add_index :users, :email,                unique: true
    add_index :users, :reset_password_token, unique: true
    # add_index :users, :confirmation_token,   unique: true
    # add_index :users, :unlock_token,         unique: true
  end
end
```

Userモデルはこのようなコードになっています。Deviseで利用できるモジュールが定義されています。

```
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
end
```

Twitter認証で利用するカラムを追加します。

カラム	説明
provider	認証で使うサービス。OmniAuthTwitterで利用します
uid	ユーザーのID。OmniAuthTwitterで利用します
user_name	ユーザ名
avatar_url	アイコン画像のURL

マイグレーションファイルを自動生成します。

```
rails g migration add_columns_to_users provider uid user_name av
```

マイグレーションファイルが自動生成されました。

```
rails g migration add_columns_to_users provider uid user_name av
      invoke  active_record
      create    db/migrate/20151216235406_add_columns_to_users.r
```

マイグレーションファイルはこのような内容になっています。

db/migrate/20151216235406_add_columns_to_users.rb

```
class AddColumnsToUsers < ActiveRecord::Migration
  def change
    add_column :users, :provider, :string
    add_column :users, :uid, :string
    add_column :users, :user_name, :string
    add_column :users, :avatar_url, :string
  end
end
```

マイグレーションを実行します。

```
rake db:migrate
```

このような実行結果が表示されます。

```
rake db:migrate
== 20151217223032 AddColumnsToUsers: migrating ================
-- add_column(:users, :provider, :string)
   -> 0.0007s
-- add_column(:users, :uid, :string)
   -> 0.0003s
-- add_column(:users, :user_name, :string)
   -> 0.0004s
-- add_column(:users, :avatar_url, :string)
   -> 0.0003s
== 20151217223032 AddColumnsToUsers: migrated (0.0019s) ========
```

Twitter認証の設定

https://apps.twitter.com/ で、作成したアプリケーションの情報を表示します。「Keys and Access Tokens」のタブをクリックすると以下の情報が表示されます。

- Consumer Key (API Key)
- Consumer Secret (API Secret)

config/secrets.ymlにTwitter認証で利用するKeyを追加します。
development: の前に設定を追加します。

```
default_twitter: &default_twitter
  twitter_api_key: (API Keyの値)
  twitter_api_secret: (API Secretの値)
```

development: と test: の設定のあとにTwitter認証の情報を追加します。

```
development:
  secret_key_base: 9cbxxxxx617b21163a7d31f1280e6973a62ea0a21a98e
  <<: *default_twitter

test:
  secret_key_base: 4fbxxxxx4de9c47525a3365728fca18fa5e4401aeff04
  <<: *default_twitter
```

本番環境では環境変数からtwitter認証の情報を取得するようにします。この行のあとに

```
production:
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

この行を追加します。

```
  twitter_api_key: <%= ENV["TWITTER_API_KEY"] %>
  twitter_api_secret: <%= ENV["TWITTER_API_SECRET"] %>
```

config/initializers/devise.rbで、Twitter認証で利用するKeyを設定します。

最後のendの前に3行追加します。

```
  config.omniauth :twitter,
    Rails.application.secrets.twitter_api_key,
    Rails.application.secrets.twitter_api_secret
```

app/models/user.rb に OmniAuth を利用する設定を追加します。この行の最後に

```
devise :database_authenticatable, :registerable,
       :recoverable, :rememberable, :trackable, :validatable
```

:omniauthable を追加します。

```
devise :database_authenticatable, :registerable,
       :recoverable, :rememberable, :trackable, :validatable, :o
```

認証処理の実装

Twitter認証は以下のような流れで行います。

1. ログインリンクをクリックする
2. Twitterの認証画面にリダイレクトする
3. Twitterと連携する
4. Twitterからのコールバックでアプリケーションにリダイレクトされる
5. Twitterからアクセストークンを使ってユーザーの認証 or ユーザーの登録を行う

2-4はDeviseの設定を行うことで実現できるため、実装する必要はありません。1と5の実装を行います。

コールバック用コントローラーの実装

最初にTwitterからのコールバックを受け取るコントローラーを自動生成します。

```
rails g controller users/omniauth_callbacks
```

このようなファイルが自動生成されます。

```
rails g controller users/omniauth_callbacks
      create  app/controllers/users/omniauth_callbacks_controlle
      invoke  erb
      create    app/views/users/omniauth_callbacks
      invoke  test_unit
      create    test/controllers/users/omniauth_callbacks_contro
      invoke  helper
      create    app/helpers/users/omniauth_callbacks_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/users/omniauth_callback
      invoke    scss
      create      app/assets/stylesheets/users/omniauth_callback
```

コントローラーにコールバックを受け取るアクションを実装します。

app/controllers/users/omniauth_callbacks_controller.rb をエディタで開きます。編集前はこのような内容になっています。

```
class Users::OmniauthCallbacksController < ApplicationController
end
```

このように書き換えます。

```
class Users::OmniauthCallbacksController < Devise::OmniauthCallb

  def twitter
    callback_from :twitter
  end

  private

  def callback_from(provider)
    provider = provider.to_s

    @user = User.find_or_create_from_oauth(request.env['omniauth

    if @user.persisted?
      flash[:notice] = I18n.t('devise.omniauth_callbacks.success
      session[:user_id] = @user.id
      sign_in_and_redirect @user, event: :authentication
    else
      session["devise.#{provider}_data"] = request.env['omniauth
      redirect_to new_user_registration_url
    end
  end

end
```

ApplicationControllerではなくDevise::OmniauthCallbacksControllerを継承するようにします。

```
class Users::OmniauthCallbacksController < Devise::OmniauthCallb
```

プロバイダー名(サービス)に対応したアクションが必要なため、twitterというアクションを実装しています。

```
def twitter
  callback_from :twitter
end
```

実際の処理はcallback_fromメソッドが担当します。

ユーザーがすでに存在する場合はDBから取得し、存在しない場合は新規作成します。このメソッドはあとでmodelに実装します。
request.env['omniauth.auth']にはTwitterからのアクセストークンが格納されています。

```
@user = User.find_or_create_from_oauth(request.env['omniauth.aut
```

@user.persisted?でモデルが保存済みか確認します。

```
if @user.persisted?
```

正しく保存されている場合は、表示するメッセージを設定し、Deviseの機能を使ってリダイレクトします。

ユーザー情報の検索と保存

Userモデルでユーザー情報の検索と保存をできるようにします。

app/models/user.rbをエディタで開きます。以下のように書き換えます。

```
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable,

  def self.find_or_create_from_oauth(auth)
    User.find_or_create_by(provider: auth.provider, uid: auth.ui
      user.user_name   = auth.info.nickname
      user.avatar_url = auth.info.image
      user.email      = User.dummy_email(auth)
      user.password   = Devise.friendly_token[0, 20]
    end
  end

  private

  def self.dummy_email(auth)
    "#{auth.uid}-#{auth.provider}@example.com"
  end

end
```

ユーザーがすでに存在する場合はDBから取得し、存在しない場合は新規作成します。 find_or_create_byメソッドはActiveRecordの機能です。ユーザーが存在しなかった場合は、Twitterからのアクセストークンの情報を使ってユーザーを新規作成しています。

```
def self.find_or_create_from_oauth(auth)
  User.find_or_create_by(provider: auth.provider, uid: auth.uid)
    user.user_name   = auth.info.nickname
    user.avatar_url = auth.info.image
    user.email      = User.dummy_email(auth)
    user.password   = Devise.friendly_token[0, 20]
  end
end
```

emailは今回利用ないため、ダミーデータを登録しています。ダミーデータを作成するメソッドを実装します。

```
def self.dummy_email(auth)
  "#{auth.uid}-#{auth.provider}@example.com"
end
```

Deviseが利用するURLを定義します。

config/routes.rbをエディターで開ます。この行を

```
devise_for :users
```

このように書き換えます。

```
devise_for :users, controllers: { omniauth_callbacks: 'users/omr
```

ログイン、ログアウトを行うためのリンクを追加します。

app/views/layouts/application.html.erbをエディタで開きます。この行のあとに

```
<div class="container">
```

これらのコードを追加します。

```
<% if user_signed_in? %>
  <%= link_to 'logout', destroy_user_session_path, method: :dele
<% else %>
  <%= link_to 'Twitter login', user_omniauth_authorize_path(:twi
<% end %>
```

user_signed_in?はDeviseの機能で、ログインしているかを確認できます。

```
<% if user_signed_in? %>
```

サーバーを起動し、ログインできるか確認します。

```
rails s
```

ログイン後にユーザーが作成されているか確認します。サーバーを起動したターミナルとは別のターミナルで、Railsコンソールを起動します。

```
rails c
```

コンソールで最後のユーザを取得します。

```
User.last
```

このような情報が保存されていました。

```
[1] pry(main)> User.last
  User Load (0.5ms)  SELECT  "users".* FROM "users"  ORDER BY "u
#<User:0x007fe6020122b0> {
                        :id => 1,
                     :email => "1111111-twitter@example.com",
         :encrypted_password => "xxxxx",
       :reset_password_token => nil,
    :reset_password_sent_at => nil,
       :remember_created_at => nil,
             :sign_in_count => 2,
          :current_sign_in_at => Thu, 17 Dec 2015 22:53:46 UTC +00
             :last_sign_in_at => Thu, 17 Dec 2015 22:53:09 UTC +00
          :current_sign_in_ip => "::1",
             :last_sign_in_ip => "::1",
                  :created_at => Thu, 17 Dec 2015 22:53:09 UTC +00
                  :updated_at => Thu, 17 Dec 2015 22:53:46 UTC +00
                    :provider => "twitter",
                         :uid => "1111111",
                   :user_name => "sada_h",
                  :avatar_url => "http://pbs.twimg.com/profile_ima
```

課題

- ログイン後にユーザー名とユーザーのアイコンが表示出来るようにしましょう。
- Facebook認証を実装しましょう
- photoとuserを紐付けましょう
- ログインしていないとphotoの新規登録/編集/削除ができないようにしましょう。

アプリケーション開発 4

前回からは、引き続き写真を投稿できるアプリケーションを実装していきます。

- アプリケーション開発 3
  - ユーザー情報の表示
  - 認証機能(Devise/Facebook)
  - UserとPhotoの関連付け
  - ログインチェック
  - コメント機能
  - 課題

ユーザー情報の表示

ログイン後にユーザー名とユーザーのアイコンが表示出来るようにします。

app/views/layouts/application.html.erbをエディタで開きます。この行のあとに

```
<% if user_signed_in? %>
```

これらのコードを追加します。

```
    <%= image_tag current_user.avatar_url %>
    <%= current_user.user_name %>
```

認証機能(Devise/Facebook)

Facebookログイン機能を実装します。

Facebook Developer登録

Facebookログインを行うため、アプリケーション登録を済ませておいてください。

Facebookにログインした状態で https://developers.facebook.com にアクセスします。

My Appsから「Add a New App」ボタンでアプリケーション登録を行います。プラットフォームにはウェブサイトを選択します。

App ID、カテゴリを設定し、アプリケーションを登録します。Site URLには http://localhost:3000 を設定します。

画面をリロードし、My Appsから作成したアプリケーションを選択すると、ダッシュボードが表示されます。ここにApp IDとApp Secretがあり、これらを利用してFacebookログインを行います。

OmniAuth Facebookの導入

OmniAuth Facebookのgemをインストールします。

- mkdynamic/omniauth-facebook

Gemfileをエディタで開き、この行の後に

```
gem 'omniauth-twitter'
```

この行を追加します。

```
gem 'omniauth-facebook'
```

gemをインストールします。RAILS_ROOTでbundle installを実行します。

```
bundle install
```

Facebook認証の設定

https://developers.facebook.com で、ダッシュボードから作成したアプリケーションの情報を表示します。

- App ID
- App Secret

config/secrets.ymlにFacebook認証で利用するKeyを追加します。Twitterの設定の後にFacebookの設定を追加します。これらの行のあとに

```
default_twitter: &default_twitter
  twitter_api_key: (API Keyの値)
  twitter_api_secret: (API Secretの値)
```

これらのコードを追加します。

```
default_facebook: &default_facebook
  facebook_api_key: (App IDの値)
  facebook_api_secret: (App Secretの値)
```

development: と test: のTwitter設定の後にFacebook認証の情報を追加します。

```
development:
  secret_key_base: 9cbxxxxx617b21163a7d31f1280e6973a62ea0a21a98e
  <<: *default_twitter
  <<: *default_facebook

test:
  secret_key_base: 4fbxxxxx4de9c47525a3365728fca18fa5e4401aeff04
  <<: *default_twitter
  <<: *default_facebook
```

本番環境では環境変数からFacebook認証の情報を取得するようにします。この行のあとに

```
twitter_api_secret: <%= ENV["FACEBOOK_API_SECRET"] %>
```

この行を追加します。

```
facebook_api_key: <%= ENV["FACEBOOK_API_KEY"] %>
facebook_api_secret: <%= ENV["FACEBOOK_API_SECRET"] %>
```

config/initializers/devise.rbで、Facebook認証で利用するKeyを設定します。この行のあとに

```
  Rails.application.secrets.twitter_api_secret
```

これらのコードを追加します。

```
  config.omniauth :facebook,
    Rails.application.secrets.facebook_api_key,
    Rails.application.secrets.facebook_api_secret
```

Facebook認証処理の実装

app/controllers/users/omniauth_callbacks_controller.rb をエディタで開きます。これらの行のあとに

```
def twitter
  callback_from :twitter
end
```

これらのコードを追加します。

```
def facebook
  callback_from :facebook
end
```

app/models/user.rbをエディタで開きます。self.find_or_create_from_oauth メソッドを以下のように書き換えます。

```
def self.find_or_create_from_oauth(auth)
  User.find_or_create_by(provider: auth.provider, uid: auth.uid)
    user.user_name  = auth.info.nickname || auth.info.name
    user.avatar_url = auth.info.image
    user.email      = User.dummy_email(auth)
    user.password   = Devise.friendly_token[0, 20]
  end
end
```

ログインを行うためのリンクを追加します。

app/views/layouts/application.html.erbをエディタで開きます。この行のあとに

```
<%= link_to 'Twitter login', user_omniauth_authorize_path(:twitt
```

この行を追加します。

```
<%= link_to 'facebook login', user_omniauth_authorize_path(:face
```

サーバーを起動し、ログインできるか確認します。

```
rails s
```

ログイン後にユーザーが作成されているか確認します。サーバーを起動したターミナルとは別のターミナルで、Railsコンソールを起動します。

```
rails c
```

コンソールで最後のユーザを取得します。

```
User.last
```

UserとPhotoの関連付け

ユーザーと写真の関連付けを行います。すでに存在するモデルに対して変更を行い、関連付けができるようにしていきます。

カラム追加

photosテーブルにuser_idというカラムを追加します。

```
rails g migration add_column_to_photos user_id
```

DBに変更を反映させます。

```
rake db:migrate
```

UserとPhotoの関連付け

UserとPhotoの関連付けを行います。

app/models/user.rbをエディタで開きます。この行のあとに

```
devise :database_authenticatable, :registerable,
       :recoverable, :rememberable, :trackable, :validatable, :c
```

この行を追加します。

```
has_many :photos
```

app/models/photo.rbをエディタで開きます。この行のあとに

```
mount_uploader :image, ImageUploader
```

この行を追加します。

```
belongs_to :user
```

app/controllers/photos_controller.rbをエディタで開きます。この行を

```
def new
  @photo = Photo.new
end
```

このように変更します。

```
def new
  @photo = current_user.photos.build
end
```

この行を

```
def photo_params
  params.require(:photo).permit(:image, :caption)
end
```

このように変更します。

```
def photo_params
  params.require(:photo).permit(:image, :caption, :user_id)
end
```

Photoはidから検索していました。ユーザーに関連付くデータを取得するようにします。このコードを

```
def set_photo
  @photo = Photo.find(params[:id])
end
```

このように変更します。

```
def set_current_user_photo
  @photo = current_user.photos.find(params[:id])
end
```

この行を

```
before_action :set_photo, only: [:show, :edit, :update, :destroy
```

このように変更します。

```
before_action :set_current_user_photo, only: [:edit, :update, :d
```

showではユーザーに紐付かないデータを表示しても問題ないため、このコードを

```
def show
end
```

このように変更します。

```
def show
  @photo = Photo.find(params[:id])
end
```

app/views/photos/_form.html.erbをエディタで開きます。この行のあとに

```
<div class="field">
  <%= f.label :caption %><br>
  <%= f.text_area :caption %>
</div>
```

この行を追加します。

```
<%= f.hidden_field :user_id %>
```

サーバーを起動し、ログインできるか確認します。すでにサーバーが起動している場合は、再起動してください。

```
rails s
```

ログイン後に新しい写真を登録し、関連付けられているか確認します。
サーバーを起動したターミナルとは別のターミナルで、Railsコンソールを起動します。

```
rails c
```

最後の写真のユーザーを確認します。

```
Photo.last.user
```

このようにユーザーの情報が表示されていれば、UserとPhotoの関連付けができています。

```
[1] pry(main)> Photo.last.user
  Photo Load (0.2ms)  SELECT  "photos".* FROM "photos"  ORDER BY
  User Load (0.2ms)  SELECT  "users".* FROM "users" WHERE "users
#<User:0x007f8821a5e460> {
                        :id => 1,
                     :email => "0000000-twitter@example.com",
        :encrypted_password => "xxxx",
      :reset_password_token => nil,
    :reset_password_sent_at => nil,
       :remember_created_at => nil,
             :sign_in_count => 5,
        :current_sign_in_at => Tue, 05 Jan 2016 22:34:41 UTC +00
           :last_sign_in_at => Tue, 05 Jan 2016 00:28:43 UTC +00
        :current_sign_in_ip => "::1",
           :last_sign_in_ip => "::1",
                :created_at => Sat, 19 Dec 2015 04:38:22 UTC +00
                :updated_at => Tue, 05 Jan 2016 22:34:41 UTC +00
                  :provider => "twitter",
                       :uid => "0000000",
                 :user_name => "xxxxx",
                :avatar_url => "http://pbs.twimg.com/profile_ima
}
```

この状態では、ログインせずに写真を登録しようとするとエラーが発生します。次はログインチェックを実装します。

これまでに作成したデータは、UserとPhotoが関連付けられていないため削除します。Railsコンソールで以下のコードを実行すると、すべてのPhotoが削除できます。

```
Photo.all.each(&:destroy)
```

ログインチェック

各ページでログインのチェックを行い、ログインしている場合のみ画像のアップロードをできるようにします。 また他人のアップロードした画像の編集、削除をできないようにします。

ログインのチェックは、ViewとControllerの両方で行います。

Controllerのログインチェック

app/controllers/photos_controller.rb をエディタで開きます。

ログインチェックを行うメソッドを実装します。ログイン指定なかった場合は、メッセージを表示しindexにリダイレクトするようにします。この行のあとに

```
private
```

これらのコードを追加します。

```
def login_check
  unless user_signed_in?
    flash[:alert] = "ログインしてください"
    redirect_to root_path
  end
end
```

この行のあとに

```
class PhotosController < ApplicationController
```

この行を追加します。login_checkメソッドを :new, :edit, :update, :destroy のアクションの前に実行し、ログインチェックを行います。

```
before_action :login_check, only: [:new, :edit, :update, :destro
```

サーバーを起動します。すでにサーバーが起動している場合は、再起動してください。

```
rails s
```

ログインしていない状態で画像の編集や追加をしようとすると、メッセージが表示されindexにリダイレクトされることを確認してください。

Viewのログインチェック

Viewを修正します。

app/views/photos/index.html.erb をエディタで開きます。この行の前後を変更します。

```
<p><%= link_to 'New Photo', new_photo_path %></p>
```

このように変更します。

```
<% if user_signed_in? %>
  <p><%= link_to 'New Photo', new_photo_path %></p>
<% end %>
```

app/views/photos/show.html.erb をエディタで開きます。この行の前後を変更します。

```
<%= link_to 'Edit', edit_photo_path(@photo) %>
<%= link_to 'Destroy', @photo, method: :delete, data: { confirm:
```

このように変更します。ログイン済みで、写真のユーザーとログインユーザーが一致している場合は、リンクを表示します。

```
<% if user_signed_in? && @photo.user == current_user %>
  <%= link_to 'Edit', edit_photo_path(@photo) %>
  <%= link_to 'Destroy', @photo, method: :delete, data: { confir
<% end %>
```

`@photo.user` で、PhotoからUserを取得しています。このままだとViewからDBに対するクエリーが発行されてしまうため、Controllerで合わせて取得するようにします。

app/controllers/photos_controller.rb をエディタで開きます。このコードを

```
def show
  @photo = Photo.find(params[:id])
end
```

このように変更します。

```
def show
  @photo = Photo.includes(:user).find(params[:id])
end
```

コメント機能

アプリケーションにコメント機能を追加します。ログインしているユーザーがコメントできるようにします。

scaffold

コメントのscaffoldを自動生成します。コメントはUserとPhotoに関連付けます。コンソールで以下のコマンドを実行します。

```
rails g scaffold comment user_id:integer photo_id:integer body:t
```

マイグレーションを行います。コンソールで以下のコマンドを実行します。

```
rake db:migrate
```

CommentとUserとPhotoの関連付け

app/models/user.rb をエディタで開きます。この行のあとに

```
has_many :photos
```

この行を追加します。

```
has_many :comments
```

app/models/photo.rb をエディタで開きます。この行のあとに

```
belongs_to :user
```

この行を追加します。

```
has_many :comments
```

app/models/comment.rb をエディタで開きます。この行のあとに

```
class Comment < ActiveRecord::Base
```

この行を追加します。

```
  belongs_to :user
  belongs_to :photo
```

サーバーを起動したターミナルとは別のターミナルで、Railsコンソールを起動します。

```
rails c
```

PhotoとCommentが関連付けられているか確認します。まだデータがないので何も取得できませんが、エラーなどが発生しないことを確認します。

```
Photo.last.comments
```

UserとCommentが関連付けられているか確認します。まだデータがないので何も取得できませんが、エラーなどが発生しないことを確認します。

```
User.last.comments
```

Controllerの修正

app/controllers/photos_controller.rb をエディタで開きます。showアクションを修正して、コメントを表示できるようにします。このコードを

```
def show
  @photo = Photo.includes(:user).find(params[:id])
end
```

このように変更してください。

```
def show
  @photo = Photo.includes(:user).find(params[:id])
  @comments = @photo.comments.includes(:user).all
  @comment  = @photo.comments.build(user_id: current_user.id) if
end
```

`@comments` にPhotoに関連したコメントを格納します。user_nameの表示を行うため、コメントに関連したUserも取得しておきます。

```
@comments = @photo.comments.includes(:user).all
```

`@comment` はコメントを新規作成するためのインスタンスです。

```
@comment  = @photo.comments.build(user_id: current_user.id) if c
```

Photoに関連させるため `@photo.comments.build` としてインスタンスを生成します。 buildの引数にuser_idを渡して、Userとも関連付けします。ログインしていない場合はコメントできないため、@commentは不要です。

Viewの修正

app/views/photos/show.html.erb をエディタで開きます。このコードのあとに

```
  <% end %>
</div>
```

このコードを追加します。

```
<div>
  <% @comments.each do |comment| %>
    <div>
      <strong><%= comment.user.user_name %></strong>
      <br />
      <p><%= comment.body %></p>
      <% if user_signed_in? && comment.user == current_user %>
        <p><%= link_to 'Delete', comment_path(comment), method:
      <% end %>
    </div>
  <% end %>
  <% if user_signed_in? %>
    <%= render 'comments/form' %>
  <% end %>
</div>
```

コメントしたユーザーのuser_nameは、commentの関連から取得します。

```
<strong><%= comment.user.user_name %></strong>
```

コメントしたユーザーのみ、コメントの削除ができるようにします。

```
<% if user_signed_in? && comment.user == current_user %>
  <p><%= link_to 'Delete', comment_path(comment), method: :delet
<% end %>
```

ログインしていないと、コメントできないようにします。

```
<% if user_signed_in? %>
  <%= render 'comments/form' %>
<% end %>
```

app/views/comments/_form.html.erb をエディタで開きます。これらのコードを削除し

```
<div class="field">
  <%= f.label :user_id %><br>
  <%= f.number_field :user_id %>
</div>
<div class="field">
  <%= f.label :photo_id %><br>
  <%= f.number_field :photo_id %>
</div>
```

これらのコードを追加します。

```
<%= f.hidden_field :user_id %>
<%= f.hidden_field :photo_id %>
```

user_idやphoto_idは表示する必要がないので、hiddenフィールドとして保持します。

サーバーを起動し、コメントできるか確認します。すでにサーバーが起動している場合は、再起動してください。

```
rails s
```

Lesson 4 課題

コメント機能を完成させましょう。

- コメントのControllerでログインチェックを行いましょう。
- コメントしたら、コメントした写真のページを表示しましょう。
- コメントの一覧画面など、不要なView/Action/Routeを削除しましょう。

総合課題

これまでの授業を通じて、基本的なRailsアプリケーションの実装方法を学びました。

総合課題では、これまでに作ってきたアプリケーションをより本格的なものにするか、新しいアプリケーションを作ってみてください。

例えばこのような機能があると、より本格的なアプリケーションになると思います。

- ページ遷移なしにコメントできるようにする
- Fav/Like機能を実装する
- ページ遷移なしにFav/Likeできるようにする
- ユーザーをフォローできるようにする
- ユーザー情報の編集ページを作る
- 退会機能を実装する
- TwiterとFacebookで同じアカウントを使えるようにする
- デザインをあてる
- Herokuなどに公開する

頑張って総合課題に取り組んでください。