



Ruby on Rails

画像アップロード

ユーザ設定・ログイン機能実装

この回の到達目標

画像アップロード機能とユーザー機能・ログイン機能の実装

基本的な CRUD 機能が実装された状態からスタートして、画像アップロード機能とユーザー機能・ログイン機能の実装をすることで、より実践的なアプリケーションの制作を目指します！

※スタートは下記 URL の内容からになります。

<https://gsblogver1.herokuapp.com/>

(※ソースコードは <https://github.com/castero1219/gsblog1/tree/31deceabd2871b566c3c9faf2aa4f106fdcecfaf0> に該当します)

MVC それぞれの役割（復習）

MVC それぞれの役割を説明

■ Controller（コントローラー）

controller（コントローラー）は、司令塔的役割を果たすものです。

■ view（ビュー）

view（ビュー）はいわゆる見栄えの部分を指します。HTML に Ruby によるデータが埋められたテンプレートといったところです。

■ Model（モデル）

Model（モデル）は、データベースとのやり取りを担当する役割を担っています。

Controller と Model の関係

Controller

- ・ 出力するために必要なデータを Model にリクエスト
- ・ データの追加や変更を Model に指示



Model

テーブル

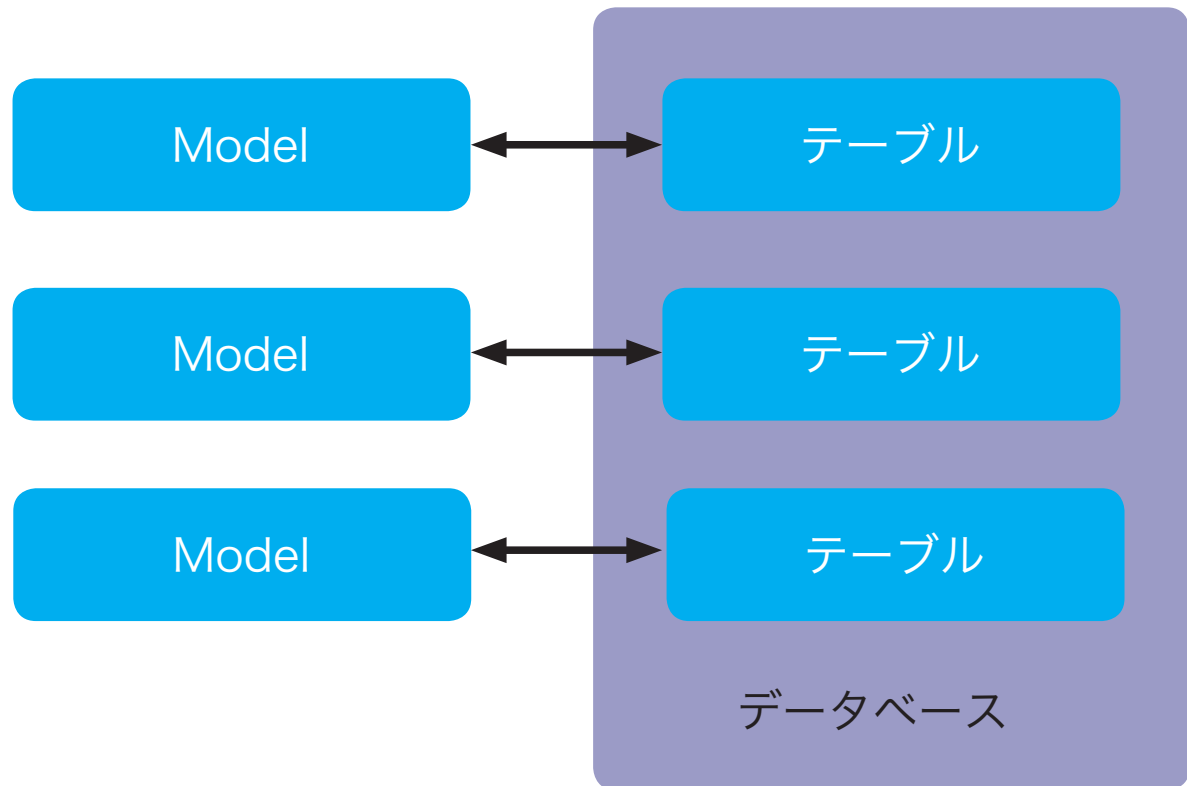
Model

テーブル

Model

テーブル

データベース





画像アップロード機能

画像アップロード機能の実装

画像アップロードに必要なことリスト

- ・ form に画像アップエリア（アップフィールド）を追加。
- ・ 画像アップロード置き場の作成
- ・ controller ファイル内に画像アップロード機能実装
- ・ strong パラメーターに image を追加（strong パラメータ自体も説明）
- ・ データベースの blogs テーブルに image カラムを追加

だいたいこれくらいあれば OK。

今回は特に gem を使用しないでできる方法でやりますが、
carrierwave という gem を使ってアップロード機能を実装する方法も有名です。



form に画像アップロードエリアを追加

file_field ヘルパー = input type = "file" に相当するものを使用

画像をアップロードする際は Input type = "file" というものを PHP の授業でも活用していたと思います。Ruby on Rails の Form 系ヘルパーにも、当然 input type = "file" に相当するものがあります。

■書き方

```
<%= f.file_field :image ,class:"form-parts"%>
```

:image に相当する部分 . . . name 属性に該当

class に相当する部分 . . . class や id を任意で設定することができます。

アップされたときにプレビューを表示

プレビューがあるとアップされているかどうか判断しやすい◎

きちんと画像がアップされているときだけ、プレビューを表示させることができます。
image_tag ヘルパーというものを使うと、img タグに相当するものが表示できます。

```
<dt class="form-title">アイキャッチ画像</dt>
<dd class="form-item">
  <% if @blog.image %>
    <div class="thumb">
      <%= image_tag "/uploads/blog/#{@blog.image}" %>
    </div>
  <% end %>
  <%= f.file_field :image ,class:"form-parts"%>
</dd>
```


アップロード画像の置き場を作成

Ruby on Rails の画像の保管場所についての基礎を知っておこう◎

赤丸で囲まれた部分は、アップロードされた画像の置き場を示しています。
今回は uploads/blog フォルダ内にアップロードするという意味になります。

Ruby on Rails の画像置き場はどうなっているのでしょうか！？
ここで基礎を知っておきましょう！

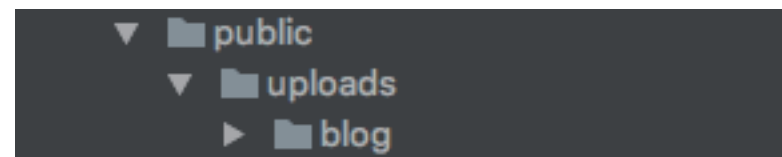
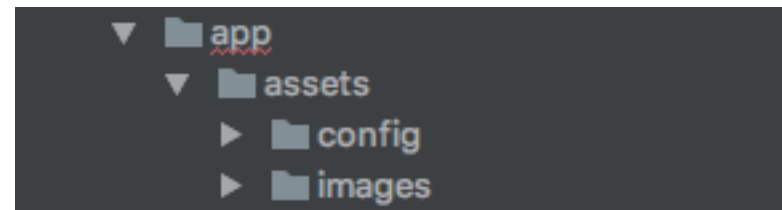
```
<dt class="form-title">アイキャッチ画像</dt>
<dd class="form-item">
  <% if @blog.image %>
    <div class="thumb">
      <%= image_tag "/uploads/blog/#{@blog.image}" %>
    </div>
  <% end %>
  <%= f.file_field :image ,class:"form-parts"%>
</dd>
```

public と assets

ユーザーがアップロードしたものは public に置かれるのが一般的

画像置き場は public フォルダ内と assets フォルダ内にあります。

どちらにおいても画像自体は表示されますが、
ユーザーがアップロードするような画像は public フォルダに置かれるのが一般的です。



public フォルダ内にある画像のパスは "/" から始めます。

(実例) /uploads/blog/image.png → public/uploads/blog 内にある image.png 参照

controller ファイル内にアップロード機能実装

まずストロングパラメーターの解説から

セキュリティ対策を何もしていないと、悪意あるユーザーから元々 form に割り当てられていない user_id などを勝手に改変してデータを改ざんする、なんていうこともできてしまいます。

これを防ぐために、form に割り当てられている内容のみが新規データ作成・更新対象であるという設定をするのが**ストロングパラメーター**だと思ってください。

言葉で説明しても伝わりずらいかなのでキャプチャで説明します・・・

controller ファイル内にアップロード機能実装

タイトル

アイキャッチ画像

ファイルを選択 選択されていません

内容

新規作成

```
private
def blog_params
  params.require(:blog).permit(:title,:content,:image,:image_cache)
end
```

フォームに入力する項目

- title
- image
- content
- (image_cache)



title,image,content,image cache 以外の更新は受け付けない！

それ以外を改ざんしようとしてもダメ！

(新規作成・更新時に使用するので一つのメソッドにまとめます。メソッド自体が外から改変されないように private をつけます)

データベースに image カラムを追加

データベースのカラムの追加も簡単にできる◎

元々のファイルは、画像をアップする機能がついていませんでしたので、データベースにも画像を格納するためのカラムは用意されていませんでした。

Ruby on Rails では、データベースのカラムの追加・変更も割と優しい手順で行うことが可能です。

追加変更手順：migration して rake db:migrate

migration はデータベースのテーブルやカラムの構造を変更できる仕組み

```
gsblog-ver3 tatsuyakosuge$ rails generate migration add_image_to_blogs image:string
```

■書き方 (db/migrate フォルダに migration ファイルが生成)

```
rails generate migration add_image_to_blogs image:string
```

赤部分・・・migration ファイル名になる。命名の仕方には慣例あり。
(add_ 追加するカラム名 _to_ テーブル名)

緑・・・追加するカラムの名前とデータの型

■ migration ファイルを生成したら、rake db:migrate

migration の内容を実際にファイルに反映させるのが rake db:migrate だと思ってください◎

```
rake db:migrate
```



ユーザー機能追加

scaffold でさくっとユーザー機能を実装しよう

CRUD 機能をさくっと実装してくれる scaffold 機能

```
rails generate scaffold user name:string email:string
```

■書き方

```
rails generate scaffold user name:string email:string
```

赤部分・・・モデル名

緑・・・テーブルの構造

scaffold は、イメージ的には generate controller と generate model の合わせ技です。
前回もやりました resources :users など用いられているので、routes.rb をチェック！

scaffold で使用する view の見た目を整える

新規ユーザー登録

Name

Email

一覧へ戻る

登録・更新

※参考 URL

<https://gsblogver2.herokuapp.com/>

(スタートの状態＋画像アップロードをした状態です)

(※ソースコードは <https://github.com/castero1219/gsblog1>)。



ログイン機能

devise という gem でさくっとログイン機能実装

いとも簡単にログイン機能が実装できてしまう優れたもの

Ruby on Rails では、gem を活用することで、ログイン機能をさくっと実装することができます。ログイン機能を実装できるもので代表的な gem は、devise と言われるものです。

■ログイン機能実装の流れリスト

- gemfile に gem 'devise' を追記
- bundle install で gem の読み込み
- サーバの再起動
- rails generate devise:install
- rails generate devise でログイン機能をもつモデルクラスを実装。
- モデル・migration ファイルの調整 (今回は email が重複するので migration ファイルから消去) (devise の機能群を必要に応じたものに修正)
- migration ファイルができていることを確認したら、rake db:migrate

ログインしないと閲覧できないページを作成

before_action :authenticate_user! でさくっと実装可能

ログイン機能を実装するということは「ログインしないと閲覧できないページ」を作ることも自然と伴います。今回であれば、blogs 関連、user 関連全てが対象でいいのではないかと思います。

■書き方

before_action :authenticate_user! をログインしないと閲覧できないページが存在するアクションファイルの冒頭に追記するだけ。

:authenticate_user! は devise 自体が実装してくれているアクションなので、特に自分で書く必要はありません。

```
class BlogsController < ApplicationController
  before_action :authenticate_user!
  before_action :set_blog, only: [:show, :edit, :update, :destroy]
```

devise の注意

rails generate devise:views しないとログイン関連の view を触れない

rails generate devise:view をしないと、ログイン関連の view ファイルをいじることができません。
よく触るであろうファイルは以下になります。

※サインアップ画面

app/views/devise/ragistrations/new.html.erb

※ログイン画面

app/views/devise/sessions/new.html.erb

ログイン関連各画面の URL

基本的なメールと

※サインアップ画面 URL


http://localhost:3000/users/sign_up

※ログイン画面 URL

http://localhost:3000/users/sign_in

モデルファイルの修正

rake db:migrate する前に、余計な記述を省く！

```
class User < ApplicationRecord
  
  has_many :blogs
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable, :validatable
end
```


- database_authenticatable・・・ログイン機能
- registerable・・・ユーザー登録機能
- validatable・・・ユーザー登録時のバリデーション機能

(devise だけでソーシャルログインも可能！)

(今回はあくまで上記のようにするだけで、やりたいことに応じて中身は変更すべきです)

モデルファイルの修正

rake db:migrate する前に、余計な記述を省く！

```
class User < ApplicationRecord
  
  has_many :blogs
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable, :validatable
end
```

- database_authenticatable・・・ログイン機能
- registerable・・・ユーザー登録機能
- validatable・・・ユーザー登録時のバリデーション機能

(devise だけでソーシャルログインも可能！)

(今回はあくまで上記のようにするだけで、やりたいことに応じて中身は変更すべきです)

migration ファイルの修正

rake db:migrate する前に、余計な記述を省く！

```
class AddDeviseToUsers < ActiveRecord::Migration[5.0]
  def self.up
    change_table :users do |t|
      ## Database authenticatable
      t.string :encrypted_password, null: false, default: ""
    end
  end
end
```

devise のデフォルト仕様で email が出力されますが、こちらを削除します！
(赤丸で囲われたあたりに書かれているかと思います)

■最後までできたら

rake db:migrate そして完了！