

WEB+DBシステム(応用編)



第12回(2016年12月14日)
人気投票サイトの制作(1/3)

設計の基本方針

- ※ 作った部分が目に見えて、試しながら(ある程度、達成感を感じながら)進める。
(実際の仕事であっても、全く見えないと疲労が倍加する)
- ※ メッセージは、基本は英語として多国語化の一つの言語として日本語を入れて行く。
- ※ 画面の修飾は後回しにする。
(WEB Designについては、他に参考資料が多いので、ロジックを作り上げる部分に力点を置く。)
但し、力作であれば加点の対象とします。
- ※ 言うまでもなく、gitを使ってバックアップを取りながら進める。

これまでに作った部分

ショッピングサイトをまず作成した。

これをベースにして、商品の「人気投票」機能を組み込んでみる。

作りたいシステムのイメージ

私の場合には、「好きな野菜の人気投票」です。

(しつこいようですが、皆さんは各自のイメージで、
〇×の人気投票で読み替えて作ってください。)

出来上がり形を、画面から考えてみる。

※ こんなことを画面に出したい、というものを
考えて、機能をイメージする。

人気投票画面(こんな感じ)

好きな野菜に投票しよう！

あなたの投票権番号：12345



トマトに一票



現在は、大根に投票済みです。



カボチャに一票

こんな感じの画面
イメージから、設計
を始めて見る。

機能を考える。

「投票権」ごとに一票とするために、「ログイン認証」をしない人でも、「投票権番号」入力をしてもらい、その番号の投票記録を残したい。
(気が変わるかも知れない)

- ということは、「投票権番号入力画面」が必要。
- 「発行済」の「投票権番号」の管理画面も必要。
- 投票画面や、結果閲覧画面で、「投票権番号」が入力済みかどうか、確認が必要。

ランキング画面(これが見たい)

野菜人気Best10!

第1位



トマト 234票

第2位



カボチャ 123票

欲を出すと大変なので、
今回作るのはここまで
にする。

機能を考える。

投票結果が「投票権」に関連して保存されているなら、それを集計すれば順位がわかる。

順位がわかったら、順位順にベスト10を表示すれば良いので、ここでは集計のロジックだけ考えれば良い。

一般に、「格納」されている情報を表示させる方が、「格納」すべき情報を検証し、入力させる部分に比べて、構造は単純になる。

作るべき主な画面のリストアップ

投票権番号の登録

→ ticketとして、scaffoldする。

野菜の登録画面

商品画像を活用する

投票画面

ランキング画面

作るべきモデル(テーブル)

野菜(商品): Merchandise(作成済み)

ユーザ: User(作成済み)

今回は、ログインユーザだけ管理画面を操作させる。

投票権: Ticket

投票権(1票)ごとに、投票内容を記録する。

つまり、このテーブルに「投票結果」を保存する。

作るべき機能

投票権番号を入力済みかどうか確認し、入力済みでなければ投票権番号入力画面にリダイレクトする。

投票権番号の発行画面は、「管理者権限」のある人にものみ操作させる。(発展課題)

投票ボタンをクリックした時に、記録する。

ランキングの問い合わせがあった際に集計して、ランキング順位を作成する。

開発手順(1)

作るもののイメージが固まったら、手順を決める。

投票権管理(投票権番号の発行や管理)
できれば、「一括発行」も作ってみる。

人気投票画面に野菜(商品)を表示し、投票を受け付ける。

開発手順(2)

人気投票画面で「投票権番号」の入力済みを確認

ユーザの、投票権番号の入力画面

人気投票画面に「投票ボタン」を作る。

人気投票画面で「投票」を記録する。(2日目/年内最終でここまで)

投票結果を集計する。

ランキング画面に表示する。(年明け:最終日)

投票権番号の発行機能を、管理者権限のあるログインユーザに限定する。

何かを作るときは・・・

卒研もそうですが・・・

そのプロジェクトごとに、開発のための「ノート」を一冊用意すると良い。

Rspecが「ノート」の代わりになるかも知れませんが未確認なので・・・。

「こんな機能があったらいい」とか、「ここの動作がおかしい」など、気付いた点をノートに書き留めておく。

機能の追加は「思いつき」ではなく「影響範囲」を熟考して行う。
(ノートに整理して行く)

PCをノート代わりにしている人は、プロジェクト用のメモファイルを作る。(IDEで代用)

今日の作業(その1)

投票権管理(投票権番号の発行や管理)

設計イメージから、以下のように考えた。

(皆さんは、自分のイメージで決めて下さい。)

Class名はTicketとする。

データとして

(1) 投票権番号[number : integer, 3桁]

(2) 投票内容 [vote : integer]

を持たせる。

投票権番号の制約

今回は、以下のようにする。

「主」となる番号は、3桁(テーブルに保存)とする。

100~999

これに、チェックコードを付加する。

チェックコードでは、bit rotation, EXORなどの演算で、元のコードから類推しにくいものを作成する。

ユーザには、「6桁」の投票権番号が渡るものとする。

例: 100 - 951, 101 - 208(適当ですが…)

見破られにくいチェックコードの合成方法については、各自「暗号論」などで調べて下さい。
この授業の守備範囲外とします。但し、優れたものは加点対象とします。

チェックコード部分は、毎回計算で求める。

始める前に、まずバックアップ

現状を保存しておきます。

rails3work/ecocar(プロジェクトのルート)で

`git add -A`

`git commit -m '第13回授業開始時'`

などを入力して、バックアップをとっておきます。

このあとやってしまった失敗をなかったことにするには

`git checkout .`

で戻す。

```
[root@cisnote ecocar]# git add -A
[root@cisnote ecocar]# git commit -m '第8回作業前'
[master 91f6321] 第8回作業前
10 files changed, 97 insertions(+), 4 deletions(-)
create mode 100644 .rspec
create mode 100644 spec/controllers/cars_controller_spec.rb
create mode 100644 spec/models/car_spec.rb
create mode 100644 spec/spec_helper.rb
create mode 100644 spec/views/cars/index.html.erb_spec.rb
[root@cisnote ecocar]# git log
```

投票権のScaffold

以下のコマンドを、**一行で**実行します。

```
rails g scaffold ticket number:integer  
vote:integer
```

Scaffold.scssは上書きしない。次はmigration。

```
rake db:migrate
```

```
[root@cisnote ecocar]# rails g scaffold ticket number:integer vote:integer  
active_record  
create db/migrate/20121114153726_create_tickets.rb  
create app/models/ticket.rb  
rspec  
create spec/models/ticket_spec.rb  
resource_route  
route resources :tickets  
scaffold_controller  
[root@cisnote vegetable-market]# rake db:migrate  
== 20151204083810 CreateTickets: migrating =====  
-- create_table(:tickets)  
-> 0.0026s  
== 20151204083810 CreateTickets: migrated (0.0027s) =====  
  
[root@cisnote vegetable-market]# █
```

引き続き、投票画面に行きます。

ここから、投票画面の作成に移ります。

特定のモデルと直結していない、votesというcontrollerを作り、投票画面をindexと、vote画面を生成します。

`rails g controller votes index vote`
と入力します。

rails g controller votes index vote

```
[root@cisnote vegetable-market]# rails g controller votes index vote
  create  app/controllers/votes_controller.rb
  route  get 'votes/vote'
  route  get 'votes/index'
  invoke erb
  create  app/views/votes
  create  app/views/votes/index.html.erb
  create  app/views/votes/vote.html.erb
  invoke rspec
  create  spec/controllers/votes_controller_spec.rb
  create  spec/views/votes
  create  spec/views/votes/index.html.erb_spec.rb
  create  spec/views/votes/vote.html.erb_spec.rb
  invoke helper
  create  app/helpers/votes_helper.rb
  invoke rspec
  create  spec/helpers/votes_helper_spec.rb
  invoke assets
  invoke coffee
  create  app/assets/javascripts/votes.coffee
  invoke scss
  create  app/assets/stylesheets/votes.scss
[root@cisnote vegetable-market]# █
```

app/controllers/votes_controller.rb

空のメソッド、indexとvoteが生成されています。

Indexに、

```
@merchandises = Merchandise.all
```

を追加します。

```
class VotesController < ApplicationController
  def index
    @merchandises = Merchandise.all
  end

  def vote
  end
end
```

```
1 class VotesController < ApplicationController
2   def index
3     @merchandises = Merchandise.all
4   end
5
6   def vote
7   end
8 end
9
```

views/votes/index.html.erb

これが、投票のメイン画面です。

Controllerから、@merchandisesを受け取り、イテレータで全項目を表示します。

それぞれの野菜ごとに、ボタンを追加します。

views/votes/index.html.erb

```
<h1>Vegetables Popularity Vote</h1>
<table>
<% @merchandises.each do |vegetable| %>
  <tr>
    <td><%= vegetable.name %></td>
    <td><%= image_tag url_for({:action => 'photo',
      :controller => 'merchandises',
      :id=> vegetable.id,
      :filename => vegetable.file_name}),
      :alt => vegetable.file_name %></td>
    <td>
      <%= form_tag 'vote' do %>
        <%= hidden_field_tag :vegetable_id, vegetable.id %>
        <%= tag :input, {:type=>'hidden', :name=>'ticket',
          :value => 'number' } %>
        <%= submit_tag 'Vote', :name=>'vote' %>
      <% end %>
    </td>
  </tr>
<% end %>
</table>
```

votes/index.html.erb画面

```
1 <h1>Vegetables Popularity Vote</h1>
2 <table>
3 <%= @merchandises.each do |vegetable| %>
4   <tr>
5     <td><%= vegetable.name %></td>
6     <td><%= image_tag url_for({:action => 'photo',
7                           :controller => 'merchandises',
8                           :id=> vegetable.id,
9                           :filename => vegetable.file_name}),
10                        :alt => vegetable.file_name %></td>
11   <td>
12     <%= form_tag 'vote' do %>
13       <%= hidden_field_tag :vegetable_id, vegetable.id %>
14       <%= tag :input, {:type=>'hidden', :name=>'ticket',
15                   :value => @ticket.number } %>
16       <%= submit_tag 'Vote', :name=>'vote' %>
17     <%= end %>
18   </td>
19 </tr>
20 <%= end %>
21 </table>
22
```


Routesの修正

投票のメイン画面を切り換えます。

config/routes.rb

で、自動的に追加されているルーティングを、切り換えます。Get `votes/vote`をPostにします。

get `votes` => `votes#index`

post `votes/vote`

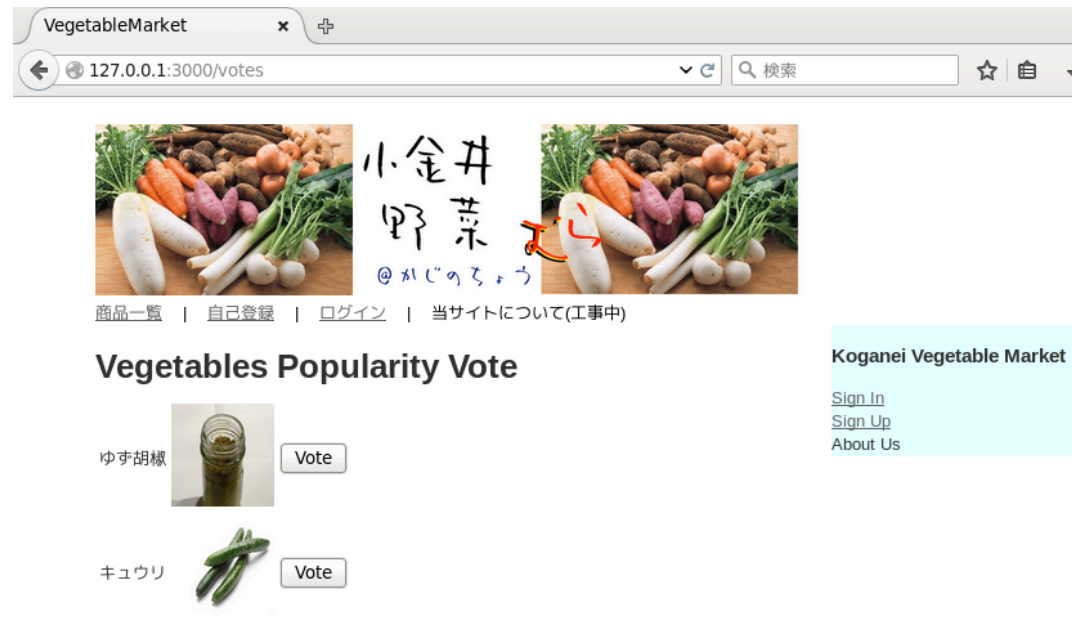
```
1 Ecocar::Application.routes.draw do
2
3   get "vote" => "vote#index"
4   post "vote/vote"
5
6   resources :photos
```

人気投票画面の表示

ここまでの修正で、

<http://127.0.0.1:3000/votes>

にアクセスすると、右の
ような画面になる
はずです。



次は、投票ボタンの処理

投票ボタンまで組み込みましたが、まだ、処理をしていません。

うまく、人気投票の画面ができたなら、ここでバックアップを取って下さい。

```
git add -A
```

```
git commit -m `人気投票画面作成`
```

投票権番号入力

Votesのcontrollerでは、login要求をしていません。

発行済の「投票権」(Tickets)を、loginで受け渡しすると、scaffoldしたticketsの方にルーティングされてしまいます。

そこで、投票権番号の受け渡し用に、むき出しのパラメータを使うか、または、専用のClassを使うことにします。

入力画面用モデルの作成

他にも設計方法があります(form_tagを使う)が、ここでは、入力画面専用、モデルを作成します。

入力画面の form_for は、引数に Class を取るため、Rails の支援を十分に得て、構造を理解するためには、このためのモデル追加がいいかなと、判断しました。(テーブルも作りますが、入力用 Class として使います。)

```
Vote クラス、 number:integer  
                security:integer
```

を作成します。

```
rails g model vote number:integer security:integer
```

投票番号入力(ログイン)用モデル

以下の通り生成し、作成しました。

```
[root@cisnote vegetable-market]# rails g model vote number:integer security:integer
invoke  active_record
create  db/migrate/20151205015708_create_votes.rb
create  app/models/vote.rb
invoke  rspec
create  spec/models/vote_spec.rb
invoke  factory_girl
create  spec/factories/votes.rb

[root@cisnote vegetable-market]# rake db:migrate
== 20151205015708 CreateVotes: migrating =====
-- create_table(:votes)
   -> 0.0015s
== 20151205015708 CreateVotes: migrated (0.0015s) =====

[root@cisnote vegetable-market]#
```

投票権入力用controllerの生成

```
rails g controller votes login
```

ログイン用のメソッドを生成しますが、

votes_controller.rbは既に生成されています。

Conflictが発生するために、votes_controller.rbについては overwriteを skipして、viewsの画面だけ生成するようにします。

投票権入力画面の生成

```
[root@cisnote vegetable-market]# rails g controller votes login
  conflict  app/controllers/votes_controller.rb
  Overwrite /root/Documents/rails4work/vegetable-market/app/controllers/votes_controller.rb? (enter "h" for help) [Ynaqdh] n
  skip     app/controllers/votes_controller.rb
  route    get 'votes/login'
  invoke   erb
  exist    app/views/votes
  create   app/views/votes/login.html.erb
  invoke   rspec
  conflict  spec/controllers/votes_controller_spec.rb
  Overwrite /root/Documents/rails4work/vegetable-market/spec/controllers/votes_controller_spec.rb? (enter "h" for help) [Ynaqdh] n
  skip     spec/controllers/votes_controller_spec.rb
  exist    spec/views/votes
  create   spec/views/votes/login.html.erb_spec.rb
  invoke   helper
  identical app/helpers/votes_helper.rb
  invoke   rspec
  identical spec/helpers/votes_helper_spec.rb
  invoke   assets
  invoke   coffee
  identical app/assets/javascripts/votes.coffee
  invoke   scss
  identical app/assets/stylesheets/votes.scss
[root@cisnote vegetable-market]#
```


ログイン画面の確認

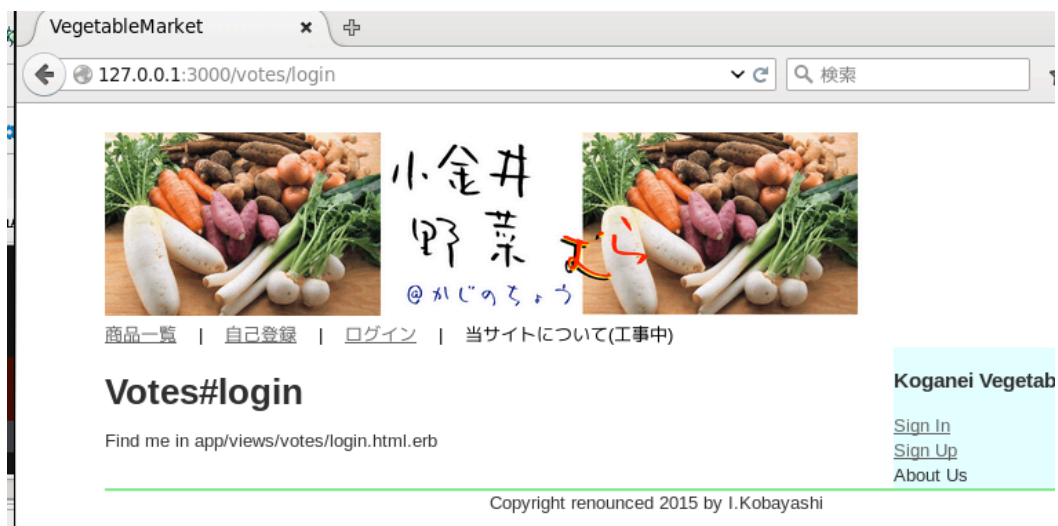
`app/views/votes/login.html.erb`

が作成されていることを確認し、メッセージを修正して
いきます。この画面で、投票権番号を入力させます。

テストランのpathは、

`http://127.0.0.1:3000/votes/login`

です



Scaffoldとの違い

rails g model と、rails g controllerの組み合わせと

rails g scaffoldとはどう違うか？

scaffoldしたモデルは、データメンテナンス用の画面が一式生成される。(ただ、 unnecessaryな画面も多い。)

個別に生成したコントローラは、controllerのメソッドと対応する画面とが、その都度、その部分だけ生成される。

Masterデータにはscaffoldが楽だが、scaffoldだとトランザクションデータには、重すぎる気がします。

投票権入力の考え方

Voteクラスのインスタンスで、投票権番号を入力させます。このインスタンスデータは、ログイン時のデータ入力の確認にのみ使います。

投票権番号とセキュリティコードをデータとして持つので、このクラスのインスタンスを受け取り、Ticketクラスのデータと照合します。

この番号が有効だったら、Ticketsクラスからデータを取得し、vote画面で制御に使います。

app/views/votes/login.html.erb

```
<h1>Input your Vote Number</h1>
```

```
<%= form_for @vote, :url =>
  { :action => 'check',
    :method => 'post' } do |f| %>
```

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

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

```
<div class="actions">
  <%= f.submit "Enter" %>
</div>
<% end %>
```

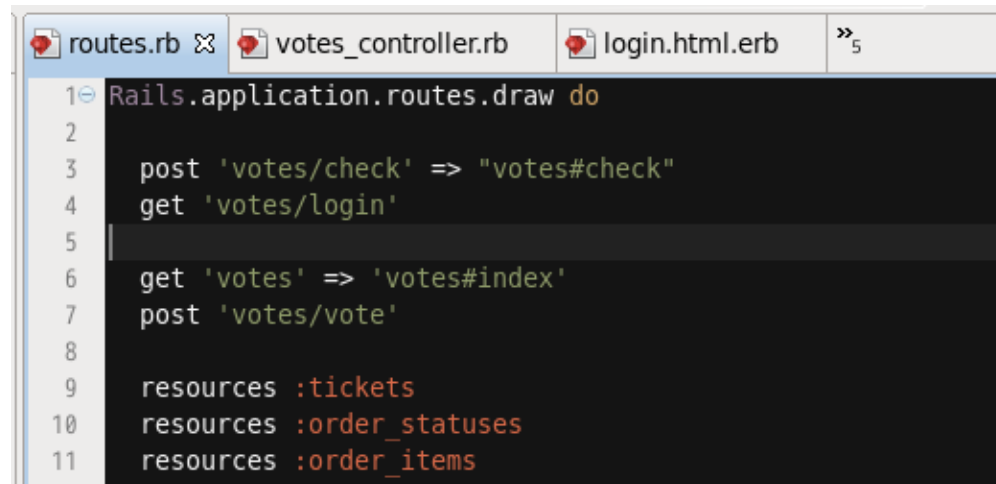
Actionとしてcheckを指定しているため、この部分のcontrollerを書くまでは、テストランできません。

config/routes.rbの修正

Login画面から、checkメソッドに入力データを渡すため、checkをpostとして追加します。

```
post "votes/check" => "votes#check"  
get "votes/login"
```

を追加します。自動生成分との競合に注意



```
routes.rb ⌵ votes_controller.rb login.html.erb »  
1 Rails.application.routes.draw do  
2  
3   post 'votes/check' => "votes#check"  
4   get 'votes/login'  
5  
6   get 'votes' => 'votes#index'  
7   post 'votes/vote'  
8  
9   resources :tickets  
10  resources :order_statuses  
11  resources :order_items  
12
```

app/controllers/ votes_controller.rb(1)

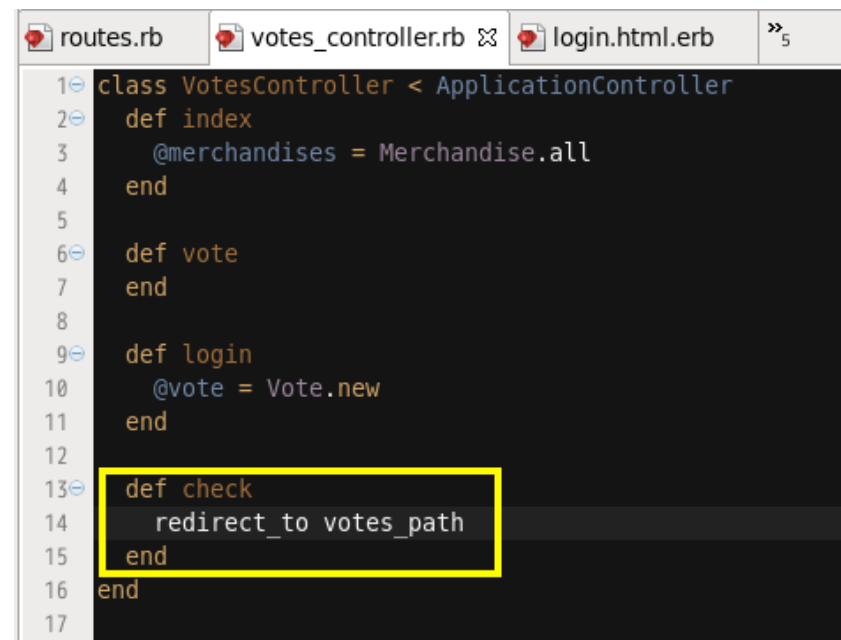
checkメソッドと loginメソッドを追加します。
checkメソッドは、テンプレートを持ちません。

```
def check  
end
```

ただだと、何もしないでテンプレート:check.html.erbを探しますので、テンプレートを探させないために、

```
  redirect_to votes_path
```

を記述しておきます。

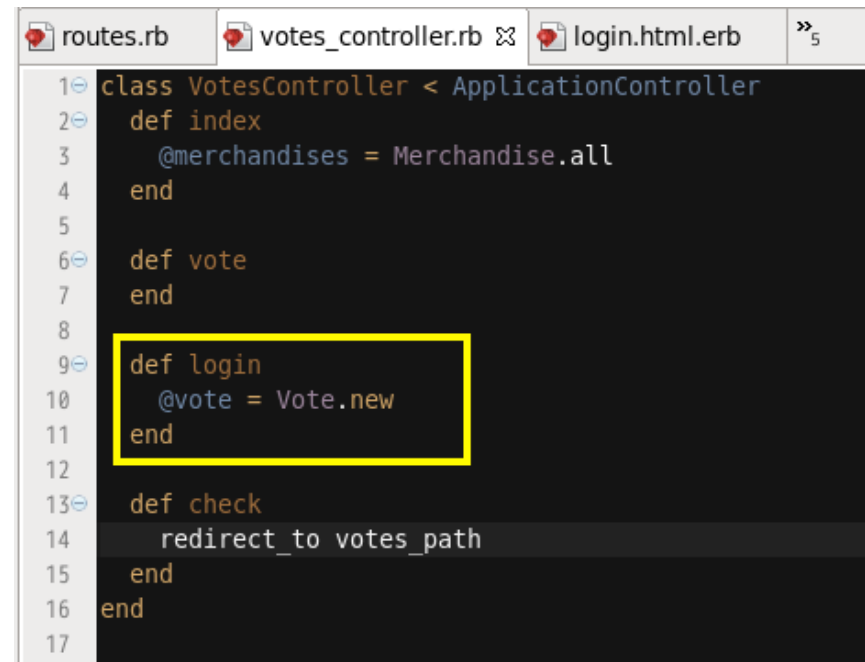


```
routes.rb | votes_controller.rb | login.html.erb | »5  
1 class VotesController < ApplicationController  
2   def index  
3     @merchandises = Merchandise.all  
4   end  
5  
6   def vote  
7   end  
8  
9   def login  
10    @vote = Vote.new  
11  end  
12  
13  def check  
14    redirect_to votes_path  
15  end  
16 end  
17
```

app/controllers/ votes_controller.rb(2)

loginでは、@voteをコントローラから受け取る、という記述で生成しました。

このため、loginメソッドに
`@vote = Vote.new`
と追加しておきます。

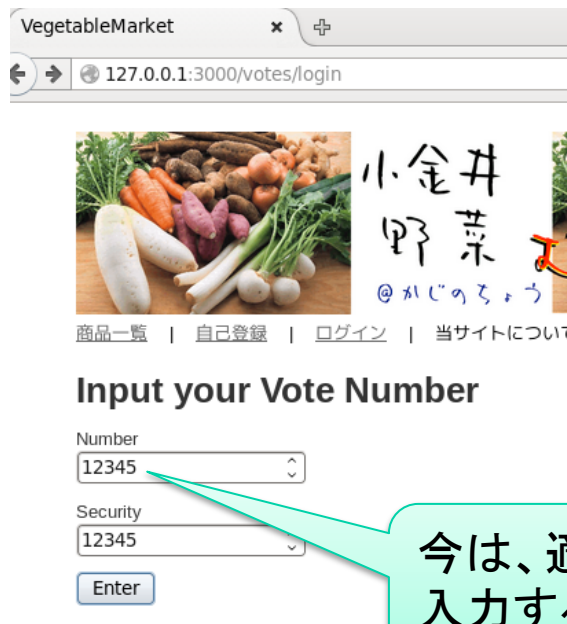


```
routes.rb | votes_controller.rb | login.html.erb | »5
1 class VotesController < ApplicationController
2   def index
3     @merchandises = Merchandise.all
4   end
5
6   def vote
7   end
8
9   def login
10    @vote = Vote.new
11  end
12
13  def check
14    redirect_to votes_path
15  end
16 end
17
```

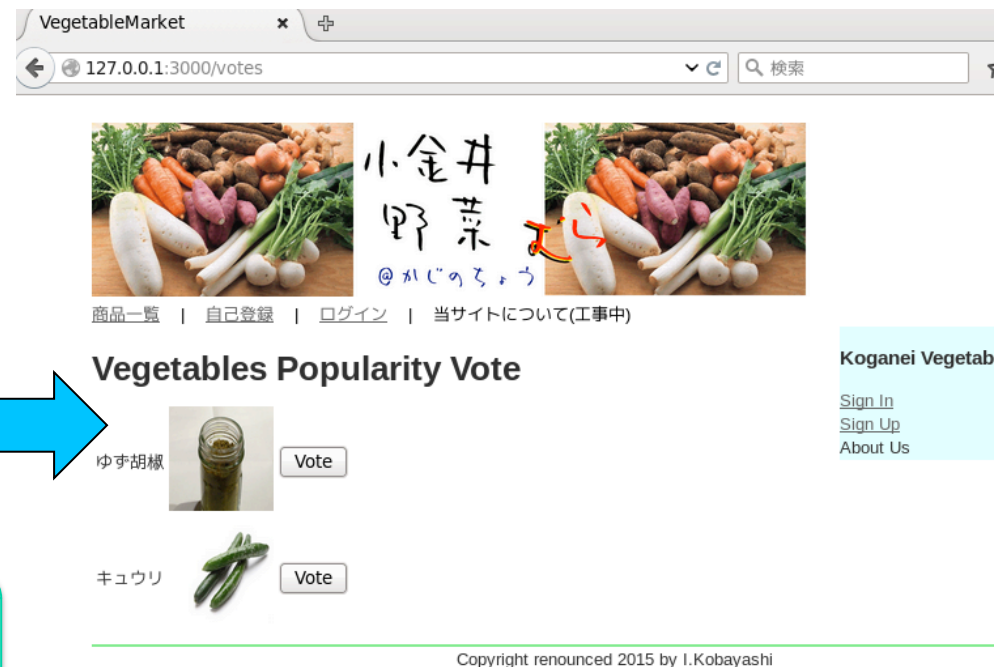
ここまでで、テストラン

<http://127.0.0.1:3000/votes/login>

で、一般ユーザの「投票権確認」を行います。
番号を入力したら、投票画面へリンクすることを確認します。



今は、適当に入力する。



実習課題

今日から話題が切り替わりました。

本日は、ここまでとします。

各自が選んだ「人気投票」の画像登録と、投票ボタンまでを作成してみてください。

今日のレポート提出はありません。

読み替えがわからなかったら、質問をして下さい。

今日の欠席課題

画像の出る人気投票画面を報告して下さい。

出席に切り換えます。

細かい説明は不要です。画面コピーをつけて下さい。