



OBJECT ORIENTED WEB PROGRAMMING USING RUBY

Day 7: 31/May/2012

CRUD of Database

Today's Goal (From syllabus)

- ❑ By testing each of the database operations: CRUD, both via the Web screen and through SQL commands, we deepen the knowledge on SQL.
- ❑ It was my translation, and I guess, sometimes automatic translator may generate the better translation than a man.

Mail Address Regular Expression

Solved! Thanks to you, folks!

I have received mail message, that

```
"^([a-zA-Z0-9_%.+]+)@([a-zA-Z0-9.-]+?)(\\.[a-zA-Z0-9_-]*)$"
worked fine. Yes, it actually worked.
```

He mentioned that, "iku+o@hosei%example.jp" was not rejected, so he doubted that '\.' must have been recognized as \.

I do not think it was the fault of expression.

It makes sense. Inside '[]', '.' does not have the meaning of any character. But if '\.' recognized as \., it could be matched to '%' or '@'.

Who parses ‘\’?

In information from the student, he mentioned that he had found the related article below;

<http://jp.rubyist.net/magazine/?0019-BundledLibraries>

(Japanese)

The problem seems to be “Somebody had eaten ‘\.’
And until we try, we can hardly know how many backslashes we should write. The conclusion is that we should perform tests in the ‘product’ environment.

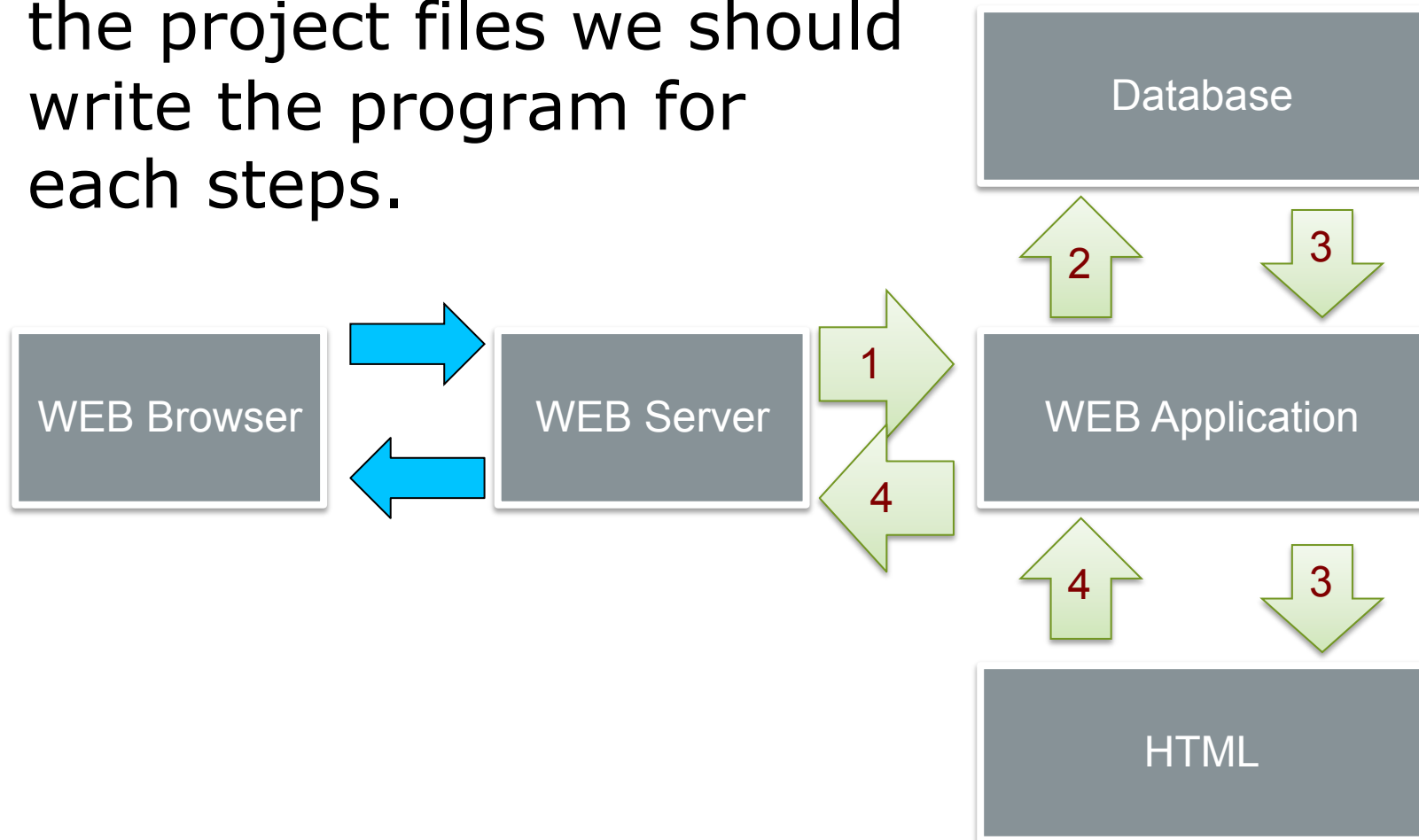
Any way, I appreciate very much.

Web Programming Basics

1. Receive REQUESTS from browsers,
2. Handle parameters in the specified method, and access to the database,
3. Retrieve information from the database, and render it to html file.
4. Send RESPONSE to browsers.

Rough Sketch of the Flow

Today we learn what part in the project files we should write the program for each steps.



routing

Type `rake routes,` in the project directory.
It should show the method names for html requests.

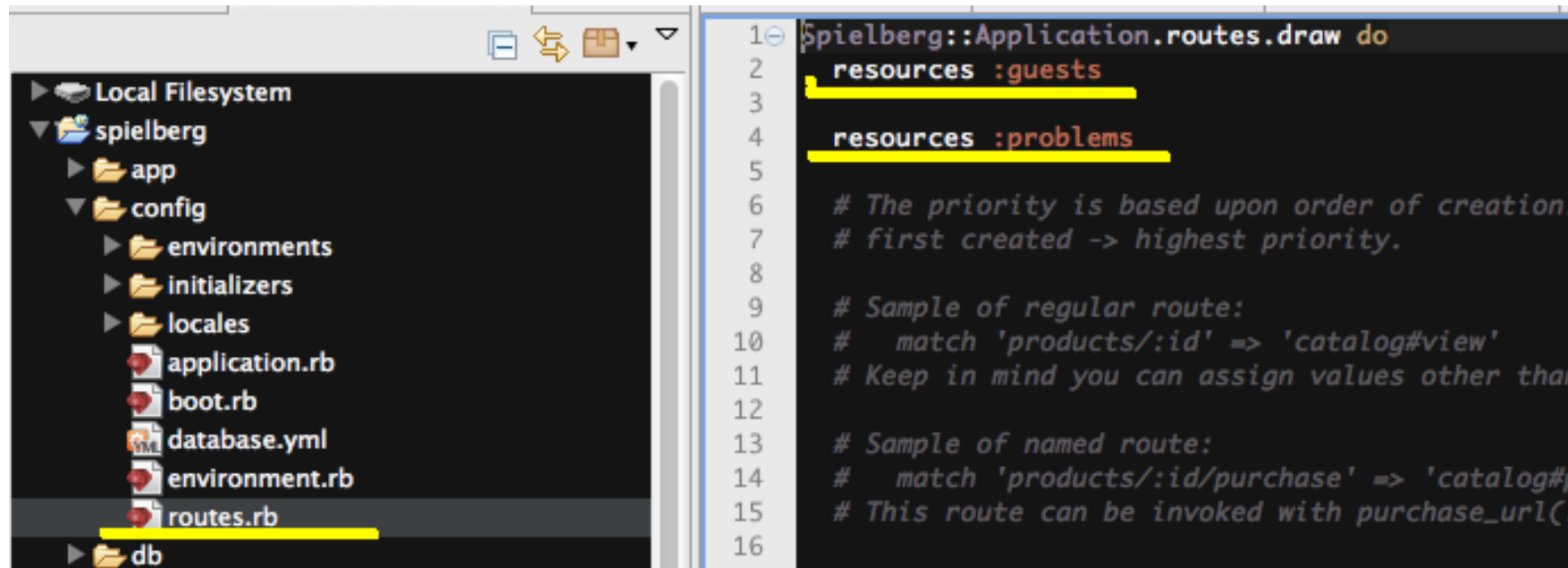
```
kobayashi-ikuo-no-MacBook:spielberg kobayashi$ rake routes
  guests GET    /guests(.:format)      guests#index
          POST   /guests(.:format)      guests#create
  new_guest GET    /guests/new(.:format)  guests#new
  edit_guest GET    /guests/:id/edit(.:format) guests#edit
   guest GET    /guests/:id(.:format)  guests#show
          PUT    /guests/:id(.:format)  guests#update
          DELETE /guests/:id(.:format)  guests#destroy
  problems GET    /problems(.:format)    problems#index
          POST   /problems(.:format)    problems#create
  new_problem GET   /problems/new(.:format) problems#new
  edit_problem GET   /problems/:id/edit(.:format) problems#edit
   problem GET   /problems/:id(.:format) problems#show
          PUT    /problems/:id(.:format) problems#update
          DELETE /problems/:id(.:format) problems#destroy
kobayashi-ikuo-no-MacBook:spielberg kobayashi$ █
```

To get this routing table,

We should specify the following two lines in config/routes.rb. (automatically generated)

`resources :guests`

`resources :problems`



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the project structure, with the `routes.rb` file highlighted. The code editor shows the following content:

```
1 Spielberg::Application.routes.draw do
2   resources :guests
3
4   resources :problems
5
6   # The priority is based upon order of creation
7   # first created -> highest priority.
8
9   # Sample of regular route:
10  #   match 'products/:id' => 'catalog#view'
11  # Keep in mind you can assign values other than
12
13  # Sample of named route:
14  #   match 'products/:id/purchase' => 'catalog#
15  # This route can be invoked with purchase_url(
16
```


HTML form tag

HTML files containing form tags may send a request to server, according to the form element's attribute.

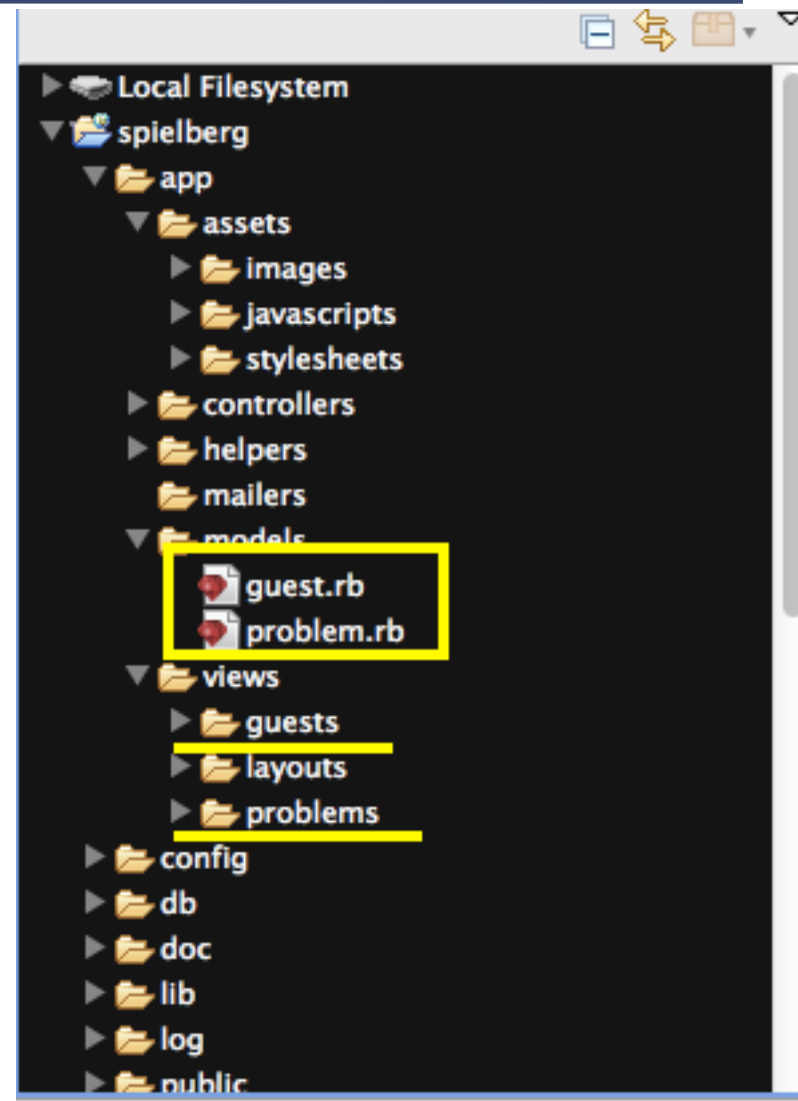
Method attribute specifies "method," and Action attribute specified "program."

How many programs do we run?

We scaffold two tables;
guests and **problems**.

So we have two programs;
guests and problems.

Of course, we can add new
program manually to the
projects, but by
combining those
programs, we extend the
system.



Top most HTML file is

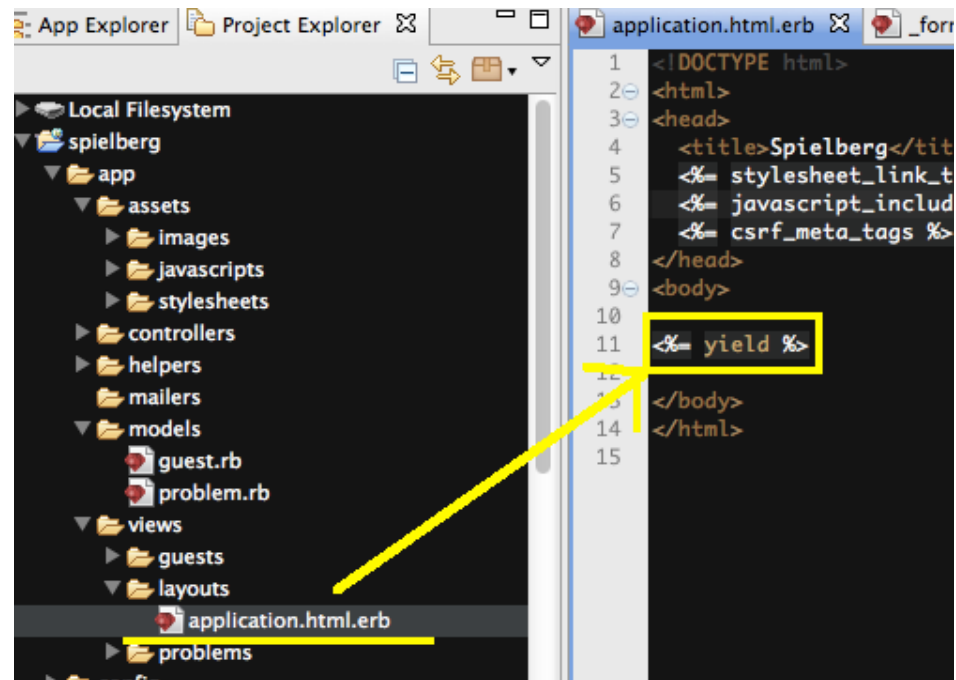
layouts/application.html.erb.

In `<%= yield %>` part,

views/guests/foo.html.erb for 'guests' program, or

views/problems/foo.html.erb for 'problems' program is embedded.

Index.html.erb is a rather simple file, so let us see `new.html.erb`.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Spielberg</title>
5   <%= stylesheet_link_tag :application, :media => :all %>
6   <%= javascript_include_tag :application %>
7   <%= csrf_meta_tags %>
8 </head>
9 <body>
10
11 <%= yield %>
12
13 </body>
14 </html>
15
```

New.html.erb to _form.html.erb

File 'new.html.erb' is simple.
It has only three lines.

```
1 <h1>New guest</h1>
2
3 <%= render 'form' %>
4
5 <%= link_to 'Back', guests_path %>
6
```

In `<%= render 'form' %>` part,
`_form.html.erb` is embedded.

_form.html.erb

```
1 <%= form_for(@guest) do |f| %>
2   <%= if @guest.errors.any? %>
3     <div id="error_explanation">
4       <h2><%= pluralize(@guest.errors.count, "error") %> prohibited this guest from being saved:</h2>
5
6       <ul>
7         <%= @guest.errors.full_messages.each do |msg| %>
8           <li><%= msg %></li>
9         <%= end %>
10      </ul>
11    </div>
12    <%= end %>
13
14    <div class="field">
15      <%= f.label (t :guest_login) %><br />
16      <%= f.text_field :login %>
17    </div>
18    <div class="field">
19      <%= f.label (t :age) %><br />
20      <%= f.number_field :age %>
21    </div>
22    <div class="field">
23      <%= f.label (t :sex) %><br />
24      <%= f.number_field :sex %>
25    </div>
26    <div class="actions">
27      <%= f.submit %>
28    </div>
29  <%= end %>
30
```

do |f|, f contains 'form' parts

Following methods are available for rails

Form block;

check_box, convert_to_model, email_field
fields_for, file_field, form_for, hidden_field
label, number_field, password_field,
phone_field, radio_button, range_field
search_field, telephone_field,
text_area, text_field, url_field

See:

<http://api.rubyonrails.org/classes/ActionView/Helpers/FormHelper.html>

in detail

Generated html code for form tag

```
<form accept-charset="UTF-8" action="/guests" class="new_guest" ↓
id="new_guest" method="post"> ↓
  <div style="margin:0;padding:0;display:inline"> ↓
    <input name="utf8" type="hidden" value="&#x2713;" /> ↓
    <input name="authenticity_token" type="hidden" value="Jt2zSy/6c2U/AEE4YY2ttGQMRTlxG+GgcP5lj8mB4/M=" /> ↓
  </div> ↓
  ↓
  <div class="field"> ↓
    <label for="guest_ログインID">ログインid</label><br /> ↓
    <input id="guest_login" name="guest[login]" size="30" type="text" /> ↓
  </div> ↓
  <div class="field"> ↓
    <label for="guest_年齢">年齢</label><br /> ↓
    <input id="guest_age" name="guest[age]" type="number" /> ↓
  </div> ↓
  <div class="field"> ↓
    <label for="guest_性別">性別</label><br /> ↓
    <input id="guest_sex" name="guest[sex]" type="number" /> ↓
  </div> ↓
  <div class="actions"> ↓
    <input name="commit" type="submit" value="登録する" /> ↓
  </div> ↓
</form> ↓
↓
```

Methods for form tags

When we write rails methods for form elements, e.g.

```
<%= f.number_field :age %> line 20 in _form.html.erb
```

This statement is rendered to

```
<input id="guest_age" name="guest[age]" type="number" />
```

in html file on the client browser, as the 'form_for' iterator part is for 'guest';

```
<%= form_for(@guest) do |f| %> line 1 in _form.html.erb
```


When we submit page,

we can see the following console message, when we click the [Register](submit) button. Here POST method is sent to `"/guests"` program.

```
Started POST "/guests" for 127.0.0.1 at Tue May [REDACTED]:22:25 +0900 2012
Processing by GuestsController#create as HTML
  Parameters: {"commit"=>"登録する", "utf8"=>"✓", "guest"=>{"age"=>"73", "login"=>"who_am_i@example.jp", "sex"=>"1"}, "authenticity_token"=>"Jt2zSy/6c2U/AEE4YY2ttGQMRTLxG+GgcP5lj8mB4/M="}
    (0.1ms) begin transaction
    SQL (33.6ms) INSERT INTO "guests" ("age", "created_at", "login", "sex", "updated_at") VALUES (?, ?, ?, ?, ?) [{"age", 73}, ["created_at", Mon, 28 May 2012 23:22:25 UTC +00:00], ["login", "who_am_i@example.jp"], ["sex", 1], ["updated_at", Mon, 28 May 2012 23:22:25 UTC +00:00]]
    (58.3ms) commit transaction
Redirected to http://127.0.0.1:3000/guests/5
Completed 302 Found in 98ms (ActiveRecord: 91.9ms)
```

Receiving HTML REQUEST

POST methods for `/guests` is handled in `app/controllers/guest_controller.rb` with `create` method, as is seen in

`POST /guests(.:format) guests#create`

message replied to `'rake routes'` command.

```
# POST /guests
# POST /guests.json
def create
  @guest = Guest.new(params[:guest])

  respond_to do |format|
    if @guest.save
      format.html { redirect_to @guest, :notice => 'Guest was successfully created.' }
      format.json { render :json => @guest, :status => :created, :location => @guest }
    else
      format.html { render :action => "new" }
      format.json { render :json => @guest.errors, :status => :unprocessable_entity }
    end
  end
end
end
```

Guest.new(params[:guest])

In the first line of 'create' method in guest_controller.rb file, 'new' method of Guest class is called with an argument of ':guest' hash data.

But we can not see any 'new' method for `class Guest` in the file '`app/models/guest.rb`.' Here, default `new` method for ActiveRecord::Base is called.

We can use default accessors with;

```
attr_accessible :age, :login, :sex
```

Database Access

With this `Guest.new` method, the following SQL command is submitted to the database.

```
Started POST "/guests" for 127.0.0.1 at Tue May 29 08:22:25 +0900 2012
Processing by GuestsController#create as HTML
  Parameters: {"commit"=>"登録する", "utf8"=>"✓", "guest"=>{"age"=>"73", "login"=>"who_am_i@example.jp", "sex"=>"1"}, "authenticity_token"=>"Jt2zSy/6c2U/AEE4YY2ttGOMRTlxG+GacP5li8mB4/M="}
  (0.1ms) begin transaction
  SQL (33.6ms) INSERT INTO "guests" ("age", "created_at", "login", "sex", "updated_at") VALUES (?, ?, ?, ?, ?) [{"age", 73}, {"created_at", Mon, 28 May 2012 23:22:25 UTC +00:00}, {"login", "who_am_i@example.jp"}, {"sex", 1}, {"updated_at", Mon, 28 May 2012 23:22:25 UTC +00:00}]
  (58.3ms) commit transaction
Redirected to http://127.0.0.1:3000/guests/5
Completed 302 Found in 98ms (ActiveRecord: 91.9ms)
```

When it is committed,

The following ruby expression is processed, seen in 'create' method in GuestController

```
respond_to do |format|
  if @guest.save
    format.html { redirect_to @guest, :notice => 'Guest was successfully created.' }
    format.json { render :json => @guest, :status => :created, :location => @guest }
  else
    format.html { render :action => "new" }
    format.json { render :json => @guest.errors, :status => :unprocessable_entity }
  end
end
```

If default 'save' method was successful, the result is redirected to 'show' method, because the URL of redirection is for a record(@guest), and this is converted to '/guest/id'

Redirect_to with Record

Redirect_to cause browser level redirection, i.e. 'external' redirection.

When the redirection is for

Hash - The URL will be generated by calling url_for with the options.

Record - The URL will be generated by calling url_for with the options, which will reference a named URL for that record.

See: <http://api.rubyonrails.org/classes/ActionController/Redirecting.html>

Redirect_to @guest

This can be recognized as,

`redirect_to :action => "show", :id => nn`

(where nn is @guest.id,) and it is a browser level "request" redirection.

GET /guest/:id is for guests#show

```
Started GET "/guests/5" for 127.0.0.1 at Tue May 29 08:22:25 +0900 2012
```

```
Processing by GuestsController#show as HTML
```

```
Parameters: {"id"=>"5"}
```

```
Guest Load (0.3ms) SELECT "quests".* FROM "quests" WHERE "quests"."id" = ? LIM
```

```
IT 1 [{"id", "5"}]
```

```
Rendered guests/show.html.erb within layouts/application (1.3ms)
```

```
Completed 200 OK in 20ms (Views: 17.5ms | ActiveRecord: 0.3ms)
```

'show' method in GuestController

Show method of GuestController calls 'find' method of Guest class. This method will be translated into SQL as,

Select * from guests where id=':id';

Then, the result will be handed to [show.html.erb](#).

If we see the following 'successfully created' message, it means that the data had been written to the database, and read from the database.

Guest was successfully created.

Login: who_am_i@example.jp

Age: 73

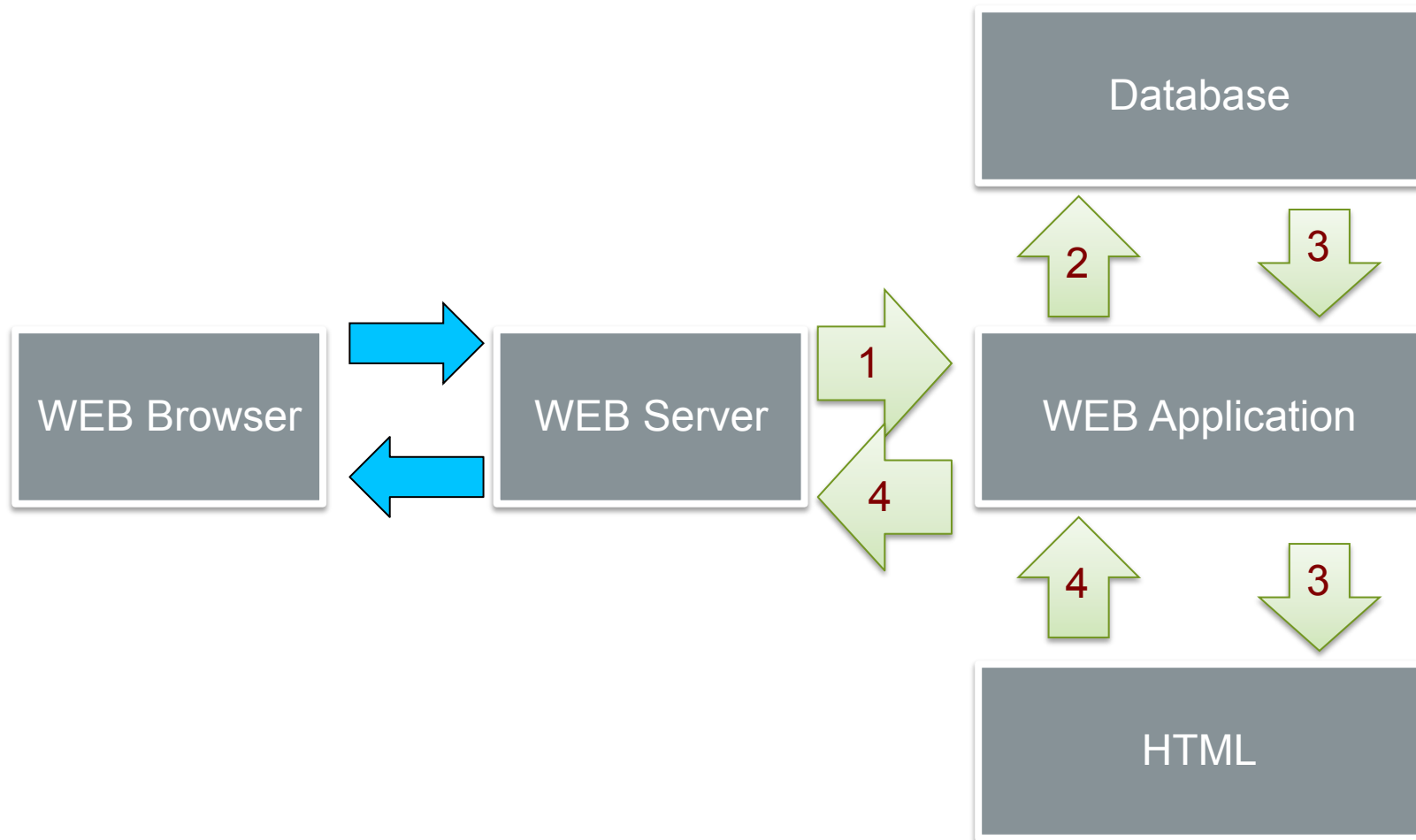
Sex: 1

[Edit](#) | [Back](#)

```
12
13 # GET /guests/1
14 # GET /guests/1.json
15 def show
16   @guest = Guest.find(params[:id])
17
18   respond_to do |format|
19     format.html # show.html.erb
20     format.json { render :json => @guest }
21   end
22 end
23
```


Once again, this is what

we have traced.



Now let's extend PSE

For today's practice, we should extend the Problem Solving Engine, a little bit.

Add 'cause' table, and 'solution' table for PSE.

Add mutual link between 'causes-problem', and 'problem-solutions.'

Table Design for [Causes]

Essential Cause should be "Facts."

Cause

should have a field of 'fact' (text,)

a counter for pro (integer,)

a counter for con (integer,)

and a link to the solution (link.)

Scaffolding 'cause' table

Type the following command,

`rails g scaffold cause fact:text pros:integer cons:integer`

```
kobayashi-ikuo-no-MacBook:spielberg kobayashi$ rails g scaffold cause fact:text pr
os:integer cons:integer
  invoke  active_record
  create  db/migrate/20120529073626_create_causes.rb
  create  app/models/cause.rb
  invoke  test_unit
  create  test/unit/cause_test.rb
  create  test/fixtures/causes.yml
  route  resources :causes
  invoke  scaffold_controller
  create  app/controllers/causes_controller.rb
  invoke  erb
  create  app/views/causes
  create  app/views/causes/index.html.erb
  create  app/views/causes/edit.html.erb
  create  app/views/causes/show.html.erb
  create  app/views/causes/new.html.erb
  create  app/views/causes/_form.html.erb
  invoke  test_unit
  create  test/functional/causes_controller_test.rb
  invoke  helper
  create  app/helpers/causes_helper.rb
  invoke  test_unit
  create  test/unit/helpers/causes_helper_test.rb
  invoke  assets
  invoke  coffee
  create  app/assets/javascripts/causes.js.coffee
  invoke  scss
  create  app/assets/stylesheets/causes.css.scss
  invoke  scss
  identical  app/assets/stylesheets/scaffolds.css.scss
kobayashi-ikuo-no-MacBook:spielberg kobayashi$
```

Table Design for [Solution]

Solution is an "Action."

Solution

should have a field of 'action' (text,)

a counter for pro (integer,)

a counter for con (integer,)

and a link to the solution (link.)

Scaffolding 'Solutions' table

Type the following command,

`rails g scaffold solution action:text pros:integer cons:integer`

```
kobayashi-ikuo-no-MacBook:spielberg kobayashi$ rails g scaffold solution action:text pros:integer cons:integer
invoke active_record
create db/migrate/20120529074011_create_solutions.rb
create app/models/solution.rb
invoke test_unit
create test/unit/solution_test.rb
create test/fixtures/solutions.yml
route resources :solutions
invoke scaffold_controller
create app/controllers/solutions_controller.rb
invoke erb
create app/views/solutions
create app/views/solutions/index.html.erb
create app/views/solutions/edit.html.erb
create app/views/solutions/show.html.erb
create app/views/solutions/new.html.erb
create app/views/solutions/_form.html.erb
invoke test_unit
create test/functional/solutions_controller_test.rb
invoke helper
create app/helpers/solutions_helper.rb
invoke test_unit
create test/unit/helpers/solutions_helper_test.rb
invoke assets
invoke coffee
create app/assets/javascripts/solutions.js.coffee
invoke scss
create app/assets/stylesheets/solutions.css.scss
invoke scss
identical app/assets/stylesheets/scaffolds.css.scss
kobayashi-ikuo-no-MacBook:spielberg kobayashi$
```

Link from Problem to Cause

Design Concept:

Problems table and Causes table could have many to many relations. Because, one 'cause' may raises many problems, and one problem may be raised by many causes.

But, 'Solution' to solve the 'Cause' for certain problem may differ from the solution to another problem even if the cause may be the same.

So we design Problem-Cause relation as "one to many" relation.

For One to Many relation

One 'Cause' belong to only one 'Problem,'
One 'Problem' may have many 'Causes.'

To 'Cause' model, set

`belongs_to :problem`

and, to Problem model, set

`has_many :causes`

Modify Models

To modify app/models/cause.rb, add
`belongs_to :problem`

To modify app/models/problem.rb, add
`has_many :causes`

```
1 class Cause < ActiveRecord::Base
2   attr_accessible :cons, :fact, :pros
3   belongs_to :problem
4 end
5
```

```
1 class Problem < ActiveRecord::Base
2   attr_accessible :content, :title
3   has_many :causes
4 end
5
```

Add one column to 'Cause'

Let 'Cause' point one 'Problem,' add one field ':problem_id' of integer.

Because id field is automatically added by rails, and its type is 'integer.'

Add one line

`t.integer :problem_id`

In 2012MMDDHHmmSS_create
_causes.rb migration file.

```
1 class CreateCauses < ActiveRecord::Migration
2   def change
3     create_table :causes do |t|
4       t.text :fact
5       t.integer :pros
6       t.integer :cons
7
8       t.integer :problem_id
9
10      t.timestamps
11    end
12  end
13 end
14
```

When we finish adding links,

Type

`rake db:migrate`

To migrate the database.

```
kobayashi-ikuo-no-MacBook:spielberg kobayashi$ rake db:migrate
== CreateCauses: migrating =====
-- create_table(:causes)
   -> 0.0351s
== CreateCauses: migrated (0.0355s) =====

== CreateSolutions: migrating =====
-- create_table(:solutions)
   -> 0.0025s
== CreateSolutions: migrated (0.0033s) =====

kobavashi-ikuo-no-MacBook:spielberg kobavashi$ █
```

Let us see cause table

Go to db directory, and type
`sqlite3 development.sqlite3`

Then, type sqlite3 command
`.schema causes`

Now we can see what table is created in the database.

```
kobayashi-ikuo-no-MacBook:spielberg kobayashi$ cd db
kobayashi-ikuo-no-MacBook:db kobayashi$ ls
development.sqlite3  schema.rb            test.sqlite3
migrate              seeds.rb
kobayashi-ikuo-no-MacBook:db kobayashi$ sqlite3 development.sqlite3
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema causes
CREATE TABLE "causes" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "fact" text, "pros" integer, "cons" integer, "problem_id" integer, "created_at" datetime NOT NULL, "updated_at" datetime NOT NULL);
sqlite> .exit
kobayashi-ikuo-no-MacBook:db kobayashi$
```

Prepare for the Next Week

We are now designing PSE, Problem Solving Engine. In order to proceed this project, we need to bind 'guest' account to the problems table, causes table, and solutions table, to record who had written these wisdoms (and/or garbage.) To bind guests table, we introduce 'Login Authentication' next week.