IST488/688 Social Web Technologies, Spring 2010
Lab 6.
Instructor: Keisuke Inoue

Description:

In this lab, you will continue to work on Ruby on Rails, a web application framework based on Ruby. **Your work in this lab will be a continuation of Lab 5. So make sure you have finished Lab 5, before you proceed.** The information you will need is this lab is covered in Thomas, Chapter 5. If you have a question, feel free to ask the instructor or the assistant.

Note: this lab assignment is designed as an individual work. To maximize your learning experience, you are encouraged to work on it independently. You may ask questions to instructors and/or friends (but keep it quiet) and look up resources, but you are not allowed to copy-and-paste any resources other than you created.

Lab Instructions:

1. Basic Setup

1.1. First, login to the Linux system through itellv.ischool.syr.edu, launch a terminal application, and go to the RoR app directory that you worked on last week (i.e. type "cd apps/cc").

1.2. Now, go to the rails_apps directory, and type in "rails cc". There should be a lot of messages printed out on the terminal. Once the directory is created go to the cc directory.

1.3. Remember the development patterns that we will go through. This is exactly what we are going to do today.

1. **Run a server script.**
2. **Run scripts to modify your web application.**
3. **Edit automatically generated files or create additional files.**
4. **Make sure your web application is running as you expected on a web browser.**

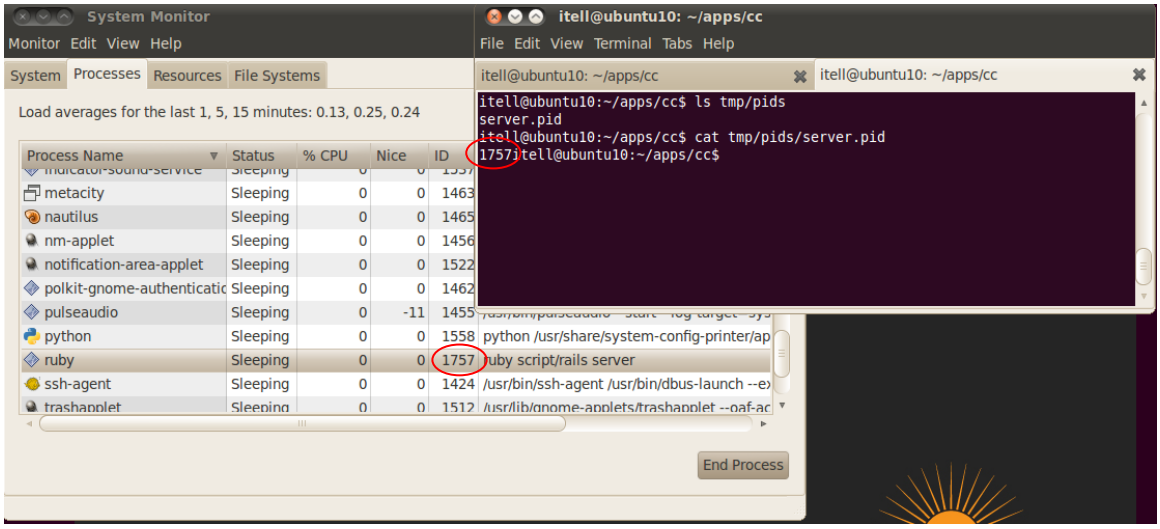
2. Stopping a server

First, let's just practice stopping the server. There are basically three ways to stop the server process.

2.1. If you still have the terminal window you used to run the server last time, go to the terminal and type in Ctrl-c. (Ctrl-C means terminating the current process on Unix systems.) That's the easiest, but this is obviously only available if you run the sever from a terminal.

2.2. Another way is to use more UNIX commands (, which you all love). First, type "ls tmp/pids". This directory contains files that the process ids of server processes (if there are any running). If you see a file called "server.pid", it means the server is running. Type "cat tmp/server.pid" to display the content of the file. That is the process ID of the server process. Now you can terminate the process by typing "kill -9 <process id>". (But wait! You don't have to do it now!)

2.3. The third way is to use GUIs. Run a program "System" → "Administration" → "System Monitor". Click the "Processes" panel. Select "Edit" → "Preference" and enable the "command line" view option. You should see the command line process "ruby script/rails server". You can terminate the process from by right-click the process and choose "Kill Process".



Two ways to identify the server process

3. Starting a server

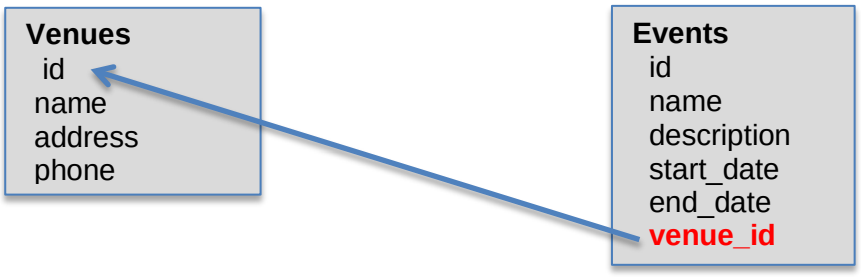
Now that the server is stopped, let's restart the server. There are two ways to start the process.

3.1. The first way is to type "script/rails server". This is the way you learned last time.

3.2. The second way is to use the "-d" option i.e. to type "script/rails server –d". This option detach the server process from the terminal, so you can use the terminal after starting the server. This would be the way to run the server, once the system is in production, but for our purpose, I personally recommend the first way, so that you can see the error and log messages from the server on the terminal.

In either case, at this point, make sure your server is running!

4. Setting up database for one-to-many relation



Last week, we did "scaffolding" using two models: Venues and Events. A problem of the models was that they were not related. This week, we are going to fix it.

4.1. During the lecture, we learned that RoR has a simple, but powerful database management mechanism called "migration". We are going to use it. First, type in the following:

```
script/rails generate migration add_venue_id_to_events venue_id:integer
```

This command line will create a migration script that adds a venue_id field the Event model. Open up the migration script just created (must be something like db/migrate/20111006XXXXXXXX_add_venue_id_to_events.rb). It should look like as follows:

```
Class AddVenueIdToEvents < ActiveRecord::Migration
 def self.up
    add_column :events, :venue_id, :integer
 end

 def self.down
    remove_column :events, :venue_id
 end
end
```

You don't need to understand what is going on exactly, but just understand that this file defines a class with two methods: 1) self.up, which is adding an integer column, named venued_id, to the Events table; and 2) self.down, removing the the venue_id column from the Events table.

4.2. Now you are know what the migration script will do, let's run it. You could simply run "rake db:migration" here (which would run only the script we just created), but we kind-of want a fresh start, since the Events table now has a new column. So let's redo the database creation. First, type in the following command:

```
rake db:rollback STEP=2
```

This will revert the two migration steps that we have done so far. Namely, creating the Events and Venue tables. (This means calling the self.down methods of the two migration files.) Now, type in the following command:

```
rake db:migrate
```

This command will once again, put the database back to the current stage of the development (calling the self.up method of the three migration files).

5. Establishing a one-to-many relationship in models

Now that the database is setup for the one-to-many relationship, you have to tell the model classes that they are related.

5.1. Now open the model file of the event model, app/models/event.rb, on your favorite editor. The file contains a very simple definition of the Event class, which inherits ActiveRecord::Base class. As follows:

```
class Event < ActiveRecord::Base
end
```

5.2. Now, add a line as follows:

```
class Event < ActiveRecord::Base
  belongs_to :venue
end
```

The new line indicates that each event is associated with one venue.

5.3. Similarly, open the model file of the venue model, app/models/venue.rb, on the editor, and add a line so that it looks like as follows:

```
class Venue < ActiveRecord::Base
```

```
  has_many :events
end
```

Be careful with the asymmetry! This new line in venue.rb indicates that each venue is associated with many events.

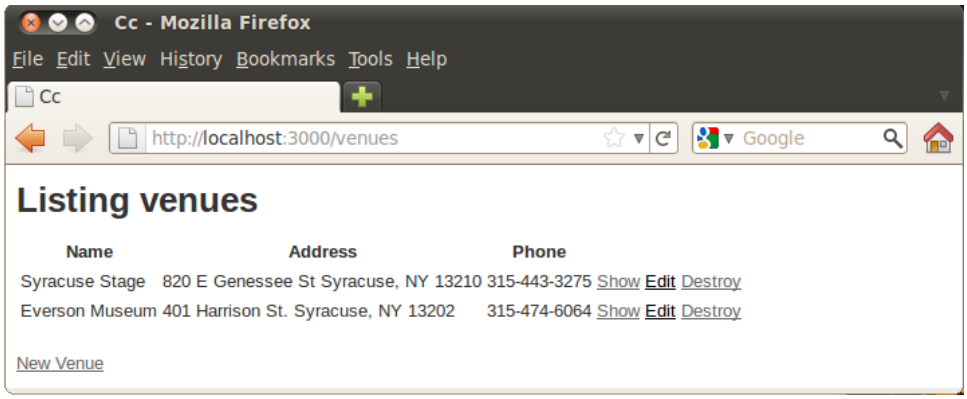
6. Venues

6.1. Now go to Firefox and go to: http://localhost:3000/venues. Create two venues with the following information:

Name	Address	Phone
Syracuse Stage	820 E. Genesee St.. Syracuse, NY 13210	(315) 443-4008
Everson Museum	401 Harrison St. Syracuse, NY 13202	(315) 474-6064

The actual information is not important for the lab, so if you would like, you can just put the names and skip the addresses and phone numbers.

6.2. Now go to Firefox and go to back to: http://localhost:3000/venues. You should see something like below:



For our purpose, we would like to see the ids of the venues. So, let's modify the viewer file for this page. Open app/views/venues/index.html.erb on the text editor.

6.3. As you can see, the file is basically an HTML file, but you see some Ruby codes embedded here and there. The rules are pretty simple:

- You can insert Ruby's string expression between <=% and %>. And the string will appear in the HTML.
- You can insert any Ruby expressions <% and %> but they won't appear in the HTML file.

If you look at the HTML, you can see, it is a simple HTML table. The trick is it uses the each method of the array, @venues, to the rows of the table. Now, let's Add a column to the table which shows the venue ID.

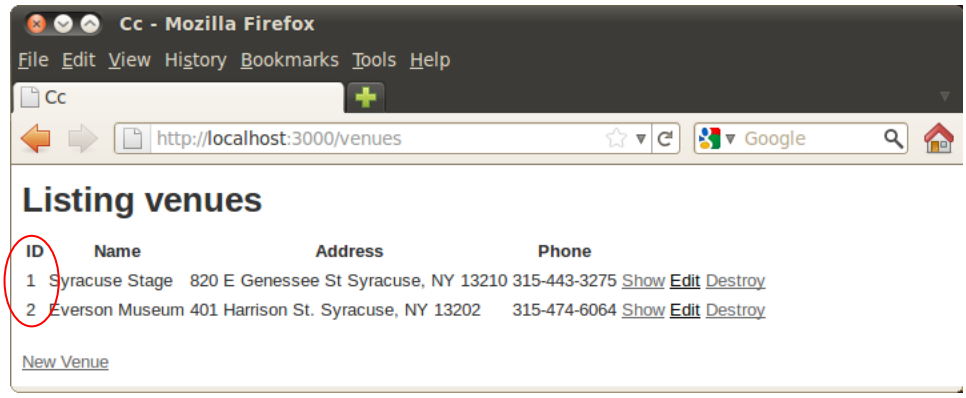
6.4. First, insert the following line above the line that says "`<th>Name</th>`".

```
<th> ID</th>
```

6.5. Similarly, insert the following line above the line that says "`<td><%= venue.name %></td>`".

```
<td><%= venue.id %></td>
```

6.6. Once it's done, save the files and go back to the browser and refresh the page. Now you should see the IDs of the venues.



Yes, the difference is very subtle…

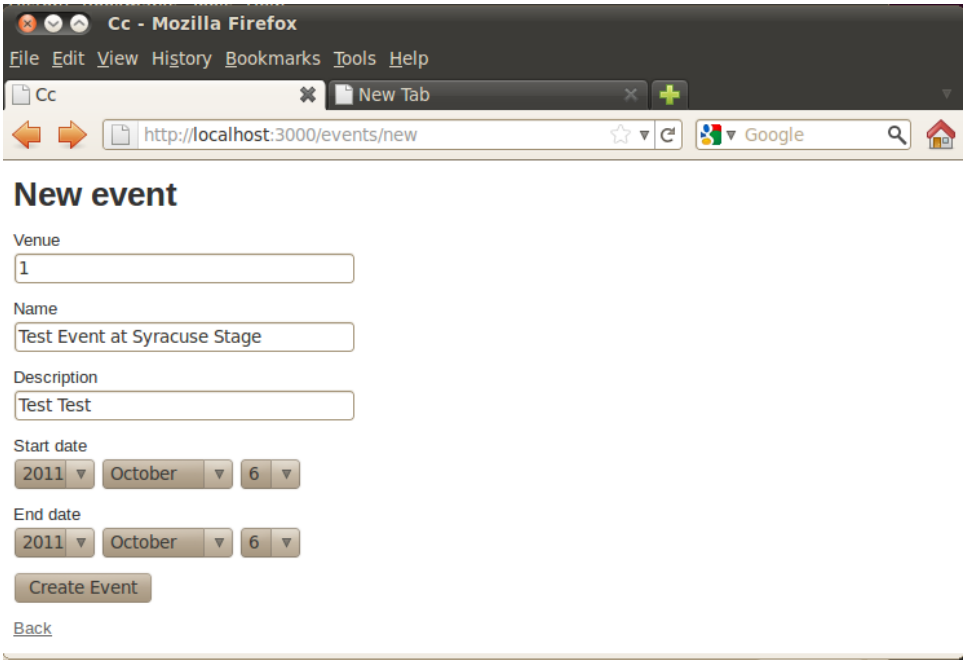
7. Events

7.1. Now go to: http://localhost:3000/events/new. This is where the app allows uses to create a new event. A problem is that you can not specify the venue_id on this page. So let's change that. Open app/views/events/_form.html.erb on the text editor.

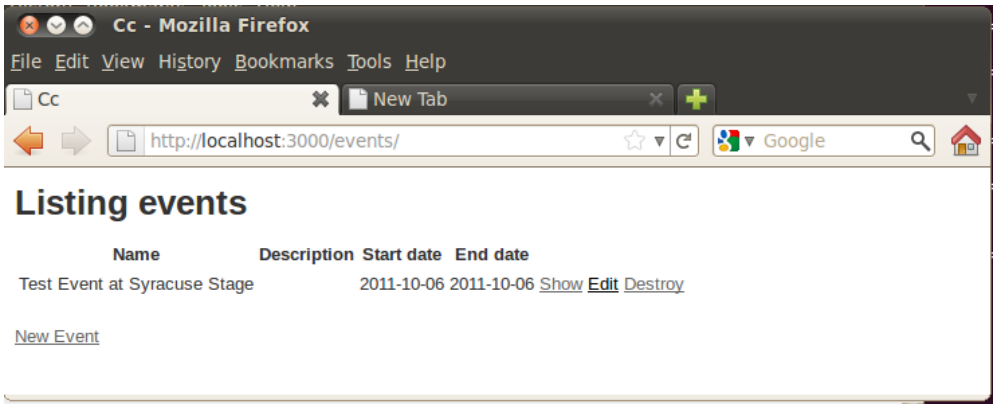
7.2. Insert a block of code (**bold** face in the excerpt below) to take input for venue_id, just above the form for the name field. Save the file.

```
…
 <div class="field">
    <%= f.label :venue_id %><br />
    <%= f.text_field :venue_id %>
 </div>

 <div class="field">
    <%= f.label :name %><br />
    <%= f.text_field :name %>
 </div>

…
```

7.3. Now go back to the browser and refresh the page. It should display the field for venue id. Now let's create an event for testing. Remember to specify the venue id that actually exists! (For our case, it should be 1 or 2.)



7.4. Once the event is created, let's checking out the event listing, i.e. http://localhost:3000/events. Currently the page does not show the event venue. So we should fix it.



The listing does not show the venue, even though the event should belong to a venue!

7.5. (By now, you should know what to do.) Open the viewer page for the event listing page (app/views/events/index.html.erb) on your text editor.

7.6. Just like what you did for the venue listing page, add a column to the table. This time, the header cell must be:

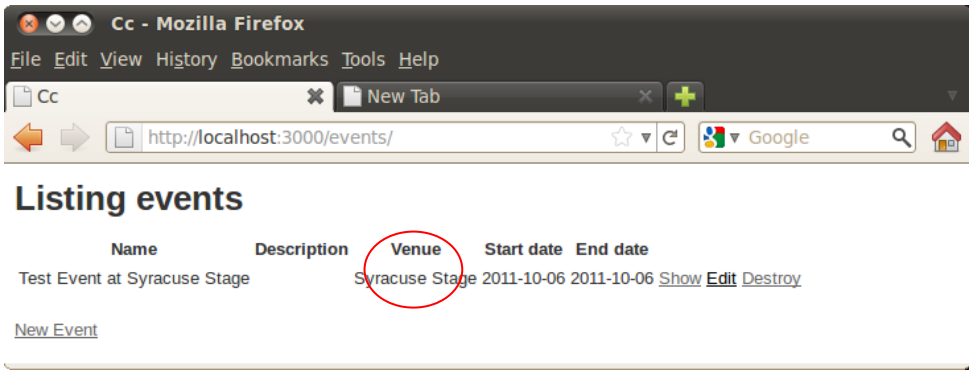
```
<th>Venue</th>
```

and the data cell must be:

```
<td><%= event.venue.name %></td>"
```

Notice that you can directory access to the venue, from the event object, and the name attribute of the venue object. This is because you specified the many-to-one relationship (the belongs-to command) in the Event model. (This is very important, so ask the instructor if you don't know what this paragraph means…) Save the file.

7.7. Go back to FireFox and reload the page, you should be able to the Venue column!



8. Submission

1. On your terminal window, go to the `apps` directory (i.e. if you are in the cc directory, type "`cc ..`").
2. Type in the following command:

   ```
   tar cvfz cc.tgz cc
   ```

   You should see a bunch of output and a new file cc.tgz created. (This is a compressed archive of the cc folder.)
3. Go to: http://sanka.syr.edu/swt/ and submit the file.