

Description:

Since this lab is longer than usual, it is assigned 20 points, and its due date will be in two weeks, instead of one. Also, since this is one more lab than the plan in the syllabus, this lab will replace the 2nd quiz.

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 8. So make sure you have finished Lab 8, before you proceed.** This lab is loosely based on the Rails book. If you have a question, feel free to ask the instructor or the assistant. Note: this lab assignment is designed as an individual work. 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.

You have been working on creating a system that shows events in the local venues. In this lab, you will add the last feature of the system, user reviews, and wrap up the project.

Lab Instructions:

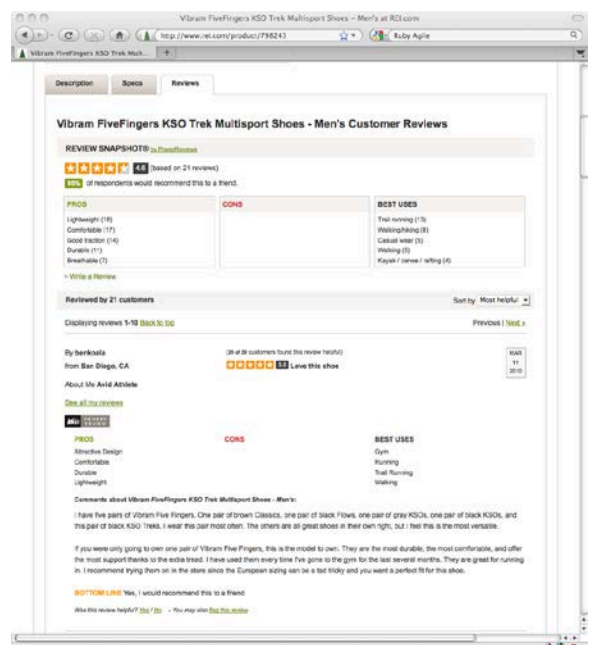
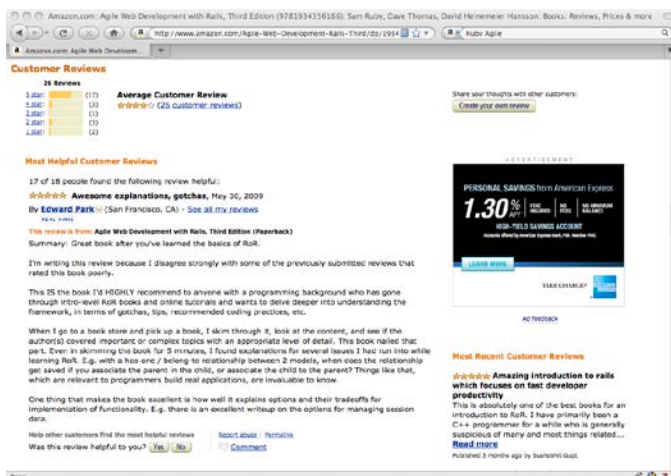
1. Basic Setup

- 1.1. As usual, launch the terminal application, and go to the app directory that you are working on.
- 1.2. If the Rails server is not running, run the server in the directory.
- 1.3. Launch Firefox and make sure Rails is running properly.

2. Adding Review Model

The goal of this lab is to add a review feature like the customer reviews at many online stores.

Below are a couple of user reviews seen at REI and Amazon.com:



Although there are differences in details, the main functionalities seen in both web sites are the same: 1) a user can post a rating of scale from 1 to 5 and a review about a product; 2) the reviews and ratings are listed under the product; 3) the average rating is calculated and also displayed under the product; and 4) visitors can also see all the reviews from one person.

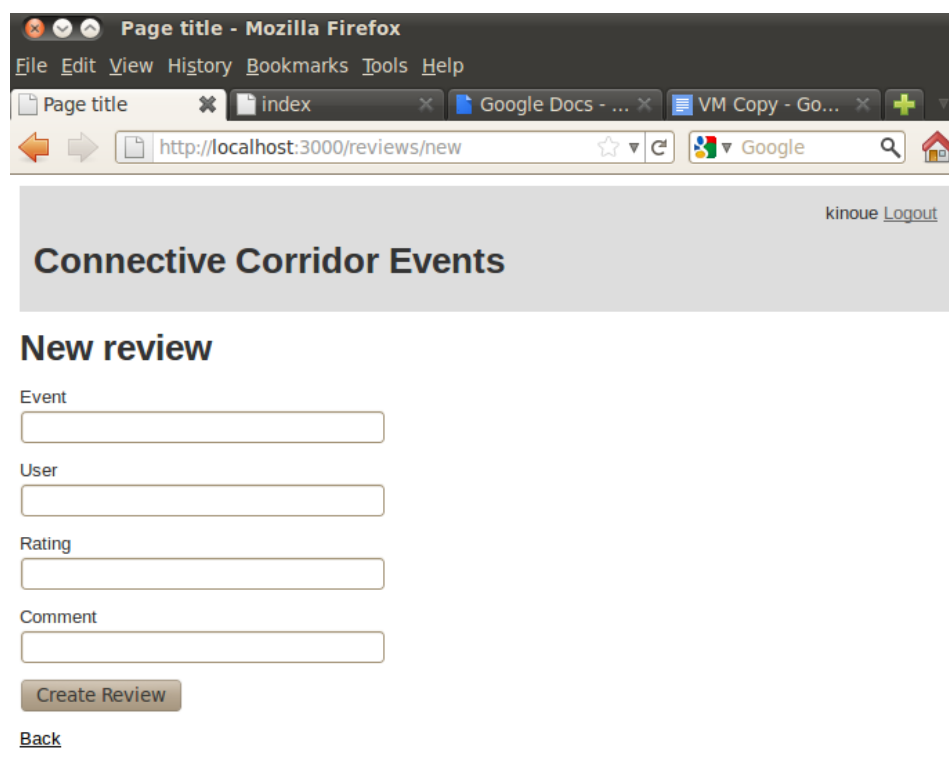
We would like to do the same, except that in our case, the reviews will be about the events in the database.

2.1. The first thing we need, as always, is to create a model for the reviews by scaffolding. Each review consists of a rating and a comment, and since each review is posted by a user, about an event, each review should hold a user ID and an event ID, too. In other words, following should be included as the attributes of the new model, review:

- event_id: integer
- user_id: integer
- rating: integer
- comment: string

2.2. Go ahead and do the scaffolding. Once it's done, run `rake db:migrate` as usual.

2.3. Check with FireFox and see if actions are working i.e., check out <http://localhost:3000/reviews/new>. Below is what it should look like:



The screenshot shows a Mozilla Firefox browser window with the address bar at `http://localhost:3000/reviews/new`. The page title is "Page title - Mozilla Firefox". The browser's menu bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The address bar shows the URL and a search icon. The page content features a header with "kinoue Logout" and "Connective Corridor Events". Below this is a "New review" section with four text input fields labeled "Event", "User", "Rating", and "Comment". A "Create Review" button is positioned below the "Comment" field. A "Back" link is located at the bottom left of the form area.

Scaffolding is nice, there is a limit as to what it can do. In this case, we have four problems (well, you may find more...):

- 1) The rating should be limited, say, from 1 to 5, so we probably want a dropdown list.
- 2) Event should to be chosen from a dropdown list, too, so that the user doesn't need to type it in.
- 3) The text field for the comment should be a bit bigger.
- 4) And lastly, we probably don't want the user to specify the user ID. Since we know who the user is (it's in the session variable, right?), we should use the information.

So let's fix all these problems...

- 2.4. ... But before doing so, at this point, Rails does not even know that `user_id` and `event_id` are meant to be foreign keys for a user and an event. So let's change that first. Open the review model file, and add lines to declare the two relationships: 1) between users and reviews and 2) between events and reviews. Remember, each relation has to be declared using one of the three: `has_many`, `belongs_to`, and `has_and_belongs_to_many`. You need to choose which one to use. Each user can post as many reviews as they want, but each review must be posted by one user; similarly, each event can be reviewed as many as times as users want, but each review must be about one event.
- 2.5. Once you edit the review model, modify the user and venue models to declare the same relationship (but from the opposite side). Again, you will be using one of the three methods listed above.
- 2.6. Now, let's move onto tackling the problems. Open the form file for the review model (`app/views/reviews/_form.html.erb`). First, we want to change the text field for the rating to a dropdown list. This can be done by... oh, wait, you should know how to do it by now! You just need to replace a `f.text_field` call with a `f.select` call. You have used the same technique twice before: once for selecting a venue on the form for events, and another time for selecting a role for the user form.

Just remember, the `select` method takes two parameters: the attribute name, and a two-dimensional array (i.e. an array of arrays). The two-dimensional array lists items to be included in the dropdown menu. Each array in the array (pair) consists of a string to be displayed and a value to be returned if the string is selected. In the case above, if a user selects "Excellent", 5 should be the selected value, etc. Below are the suggested label-and-value pairs:

Excellent: 5
Good: 4
Fair: 3
Bad: 2
Disappointing: 1

(You are welcomed to come up with your own labels.) At this point, you can save the file and reload Firefox. You should see the dropdown is correctly displayed.

The screenshot shows a Mozilla Firefox browser window with the address `http://localhost:3000/reviews/new`. The page title is "Page title - Mozilla Firefox". The browser's tab bar shows several open tabs: "Page title", "index", "Google Docs - ...", and "VM Copy - Go...". The page content includes a header with "kinoue Logout" and a main heading "Connective Corridor Events". Below this is a section titled "New review". The form contains three input fields: "Event", "User", and "Rating". The "Rating" field is a dropdown menu with "Excellent" selected. The dropdown menu is open, showing the following options: "Excellent", "Good", "Fair", "Bad", and "Disappointing". Below the form is a "Back" link.

2.7. Next, we need to change the text field for the event ID to a dropdown list. We will use the same select method that we used above, but it is a bit tricky, because we need to construct the two-dimensional array dynamically, because the list of all events depends on what are in the database. Since it's tricky, here is how to do it. First, identify the following line:

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

Then, change it to the following two lines:

```
<% event_names_and_ids = Event.all.map { |e| [e.name, e.id] } %>
<%= f.select :event_id, event_names_and_ids %>
```

This is the exactly same technique we used for the event form. The first line is constructing a two-dimensional array we need for the select method. It first finds all the events in the database using the `all` method, and then uses the `map` method to change it to an array that we need. The `map` method is a very powerful function, very similar to the `each` method that we are familiar with, but instead of simply executing a code block to each element in a given array, it will construct another array, using the return value from the execution of the code block. (Does it make sense to you...? It is hard to explain, but once you get the idea, you know you got it!) The code “`map { |e| [e.name, e.id] }`” is saying “Take the array (of events), and for each event, construct a two-element array that consists of the event name and the event id, and create a new array of arrays.”

The second line shouldn't need any explanation. It is calling the `select` method specifying `event_id` as the attribute and the two-dimensional array as the options.

(Once again, at this point, you can save the file and reload Firefox. You should see the dropdown is correctly displayed.)

2.8. As for the user ID, there are two issues: 1) we don't want to display `user_id` and 2) we don't want users to be able to modify `user_id` on the form. The simplest way to achieve these is to use a hidden field. Identify the following lines:

```
<%= f.label:user_id %> <br />
<%= f.text_field :user_id %>
```

Then change it to the following line:

```
<%= f.hidden_field :user_id, :value => session[:user_id] %>
```

This will simply assign `user_id` from the session variable without displaying it. Perfect.

(Once again, at this point, you can save the file and reload Firefox. You should see the input field for the user is now gone.)

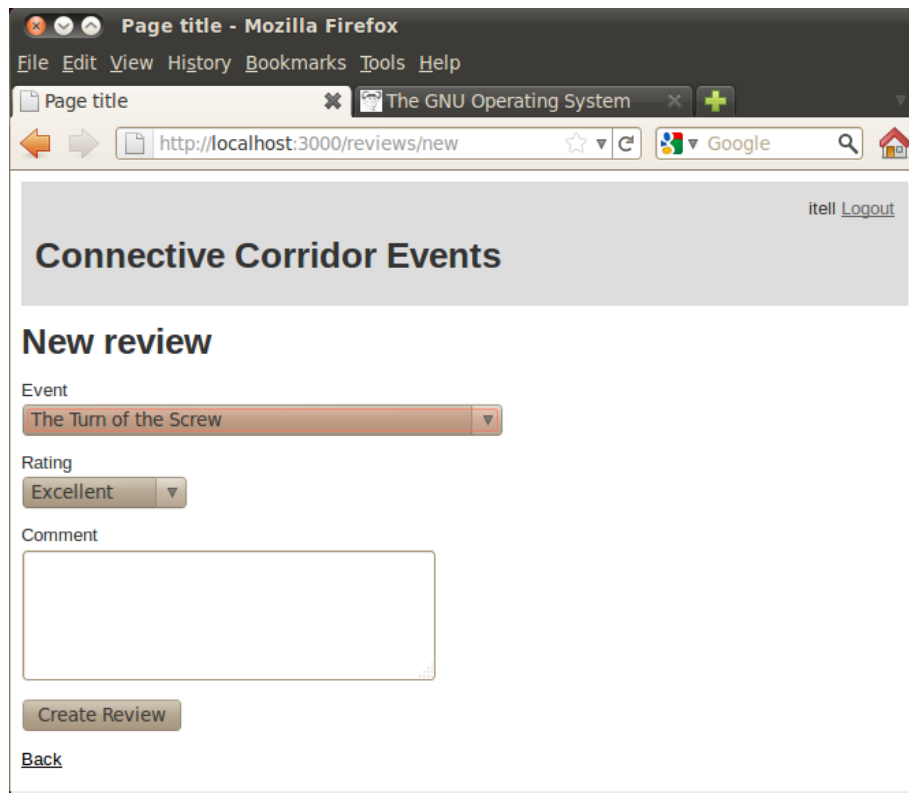
2.9. And lastly let's change the text field to a text area. Simply, change the following line :

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

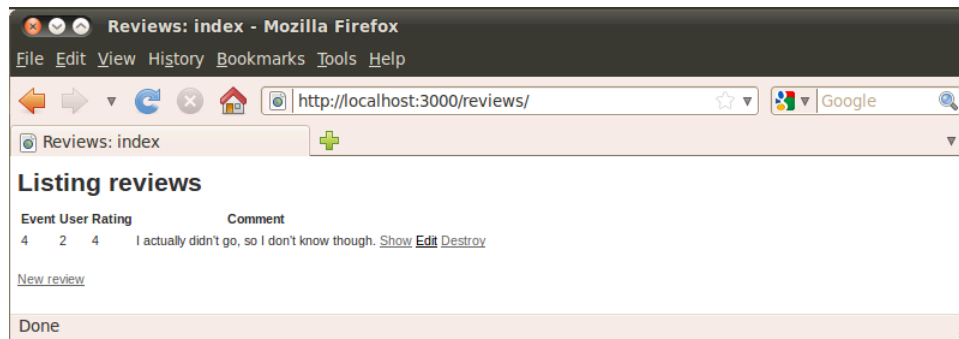
to this:

```
<%= f.text_area :comment, :rows => 5 %>
```

Now, save the file and reload the Firefox. You should see the following screenshot:



2.10. Now that you created a perfect (well, sort of) form for submitting a review, let's try it out. Create one review for whatever the event and see if it actually works. Here is what the review index page should look like after one review was created:



Notice that this page has at least a couple of problems:

- 1) The event's name is not displayed.
- 2) The user's name is not displayed.

By now, you must know the routines... go ahead and fix them!

3. Resetting Data

Before we proceed, let's reset all the data for venues, events, users, and reviews, so that all of you have the same data. First, download the three files with the following commands:

```
wget http://sanka.syr.edu/files/reset_venues.rb
wget http://sanka.syr.edu/files/reset_events.rb
wget http://sanka.syr.edu/files/reset_users_and_venues.rb
```

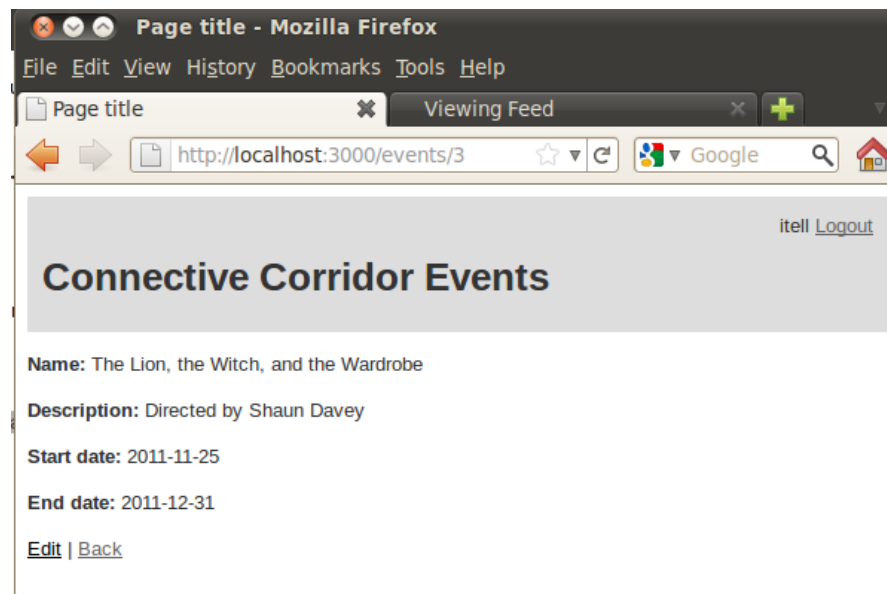
Then run the commands in order:

```
ruby reset_venues.rb
ruby reset_events.rb
ruby reset_users_and_venues.rb
```

These commands will delete all the data and re-add them. Notice that the last script will change the user information as well. Now, the admin user is “itell” and the password is “SU2orange!”. The scripts add more users with the user role. You can see those users’ information in the script.

4. Listing Reviews

Now that we have some review data, let’s list them under the event show page. Currently, the event show page looks like something like this:



We want to add two things: 1) the list of all the reviews, and 2) average ratings.

4.1. First, open the show viewer file. What we want to do is to list the reviews that are associated with the event. (And we know that an array of reviews is now an attribute of event objects.) We can pretty much copy-and-paste the index viewer file for the reviews, but we are going to use Rails’ feature called, *partial*. Partial is a template in HTML that can be used to render part of a page. It is like a for loop (or the each method) for HTML. What you need to do, is to specify a partial file, and a collection (array, in our case) as a parameter. Insert the following line in the file, near the bottom, but above the link_to tags.

```
<% if @event.reviews.count > 0 %>
  <p>
    <b>Reviews</b> <br />
    <table>
      <tr>
        <th>User</th> <th>Rating</th> <th>Comment</th>
      </tr>
      <%= render :partial => 'review', :collection => @event.reviews %>
    </table>
  <% else %>
    No reviews found.
  <% end %>
</p>
<% end %>
```

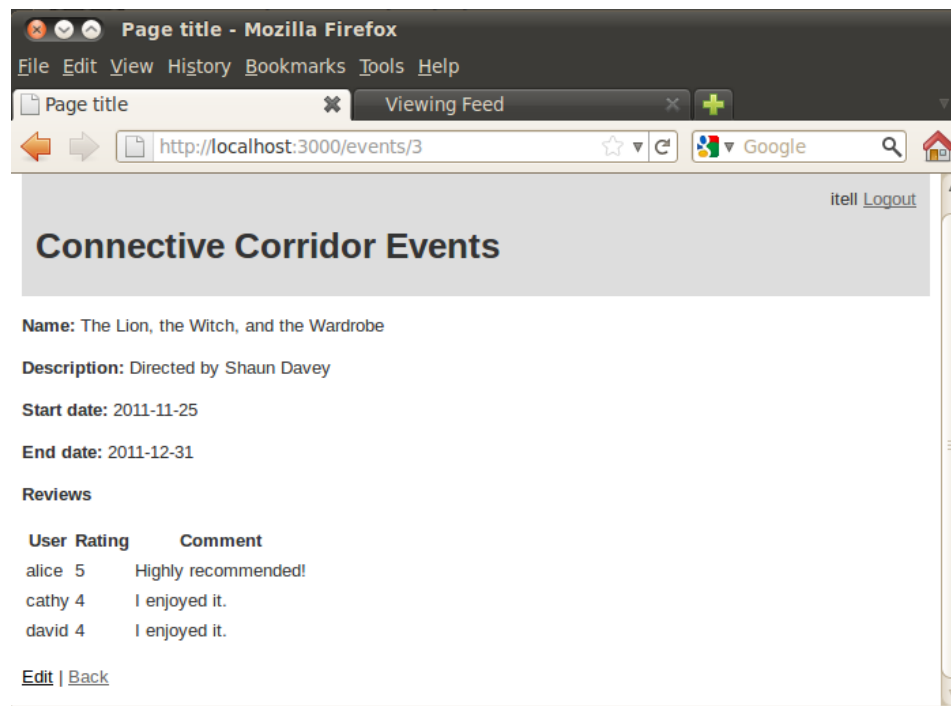
This code is using a table, but if you are comfortable with CSS, you are free to use `<div>` instead of all the table-related tags.

What it is doing here is to render each review as a table row, which is specified in the separate file, which you will create in the next step. By specifying 'review' for the `:partial` parameter, you are telling Rails two things: 1) the template file is named `"_review.html.erb"` and 2) each review will be assigned to a local variable "review". (Sounds simple and reasonable, right?)

- 4.2. Now, on the text editor, create a new file, `app/views/events/_review.html.erb` and insert the following lines:

```
<tr>
  <td> <%= review.user.name %> </td>
  <td> <%= review.rating %> </td>
  <td> <%= review.comment %> </td>
</tr>
```

(Once again, if you want go with `<div>` tags, feel free to do so.) The lines shouldn't need any explanation. Save the file, reload Firefox and see if it's working.



5. Displaying Average

The design is pretty horrible so far, but we will focus on the functionality and keep moving forward for now. The other thing we wanted was to display the average rating. So let's get it done.

- 5.1. On the text editor, open the event model file (`app/models/event.rb`). We need to define a method that tells you the average rating of the event. Insert the following method definition in the Event class definition.

```
def average_rating
  if reviews.count > 0
    sum = 0.0
    reviews.each do |r|
      sum += r.rating
    end
  end
end
```

```

    end
    sum / reviews.count
  else
    0.0
  end
end
end

```

The method simply returns the average rating, except when there are no reviews yet, in which case it returns nil.

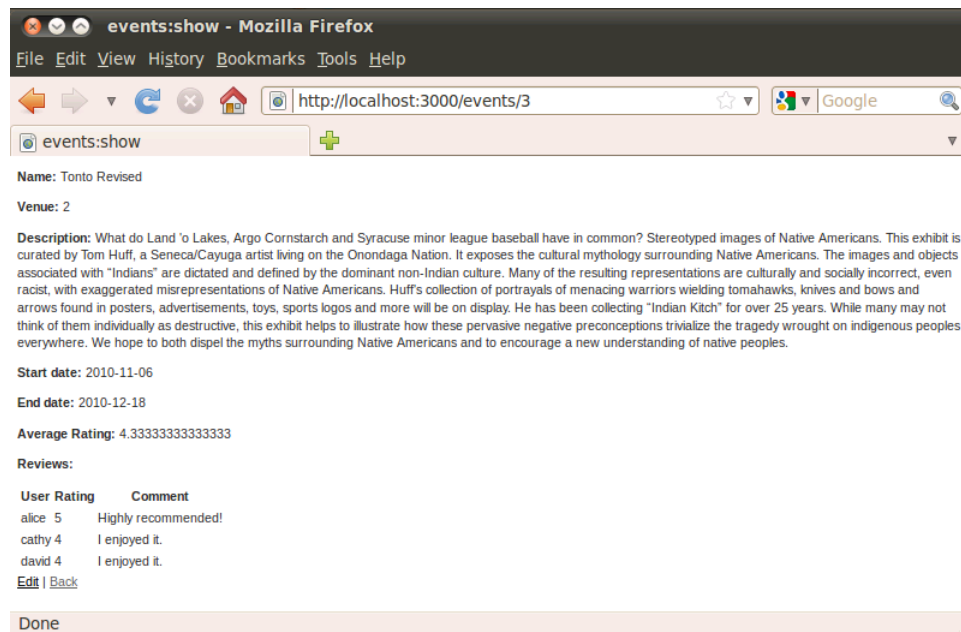
Then, go back to the show viewer file and insert the following lines below the line `<% if @event.reviews.count > 0 %>`:

```

<p>
  <b>Average Rating:</b>
  <%= @event.average_rating %>
</p>

```

Save the files and reload Firefox. You should see something like below:



6. Listing events based average ratings

Now that we have all the ratings, let's add a new page that lists the events based on the ratings. Let's call the action "top_rated". Given Rails' URL mapping, the page should be accessed by: `http://localhost:3000/events/top_rated`.

6.1. On the text editor, open the Events controller file (`app/controllers/events_controller.rb`). Add a definition of a new method, named, `popular`, as follows. (You need to type less, if you start from copy-and-pasting the `index` method defined in the class:

```

def top_rated
  Event.all.each do |event|
    if event.average_rating >= 3.0
      @events.push(event)
    end
  end
end

```



```

end

@events.sort_by!{|e| e.average_rating }.reverse!
respond_to do |format|
  format.html # index.html.erb
  format.xml { render :xml => @events }
end
end

```

The first part of the method is creating an array of event objects whose average rating is higher or equal to 3.0. Then, in the array is sorted by the average rating, and then reversed (so that it will go from high to low ratings). The syntax of the `sort_by!` method might be confusing for you, but it is similar as the `each` method, so I hope you get the idea.

- 6.2. Now that the action is defined, you need a viewer for the action. For now, we just need pretty much the same output as the Event index file. We can copy it, but we can use the partial here, too. On the text editor create a viewer file for the popular action. If you are so inclined you can start from copying the index file.

```
cp app/views/events/index.html.erb app/views/events/top_rated.html.erb
```

Then edit the file so that it will use a new partial template:

```

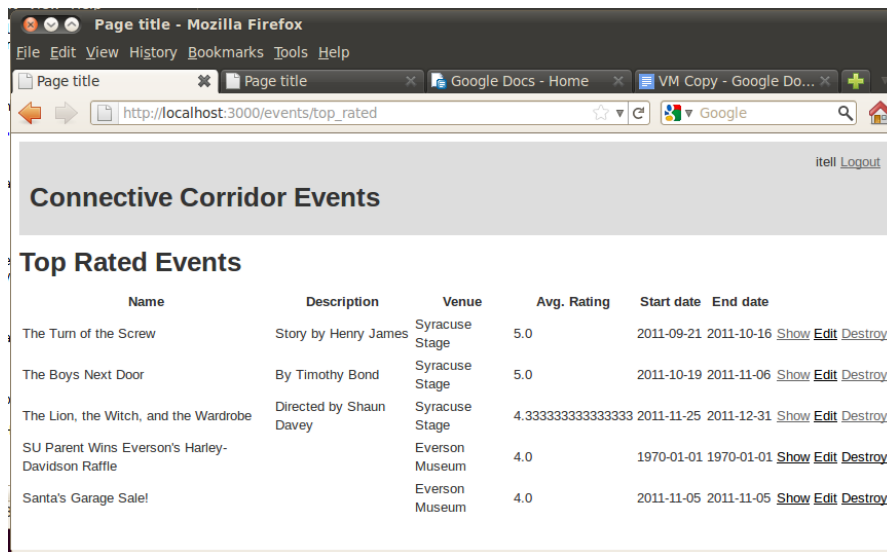
<h1>Top Rated Events </h1>
<table>
  <tr>
    <th>Name</th>
    <th>Description</th>
    <th>Venue</th>
    <th>Avg. Rating</th>
    <th>Start date</th>
    <th>End date</th>
  </tr>
  <%= render :partial => 'event', :collection => @events %>
</table>

```

- 6.3. Then, create the partial template file, i.e. `app/views/events/_event.html.erb`. You should know how it should look like. If you have no idea, check out the step 4.1 and 4.2.
- 6.4. Open the file named `config/routes.rb`, and insert the following line right above the line that says “resources :events”:

```
get "events/top_rated"
```

- 6.5. Now go back to Firefox and load the page `http://localhost:3000/events/top_rated`. You should the screen as below:



7. Adding “Current Events” page

Let's add one more page, which lists all the events that are currently going on. Let's say the page will be at <http://localhost:3000/events/current>. The current method should look like this:

```
def current
  @events = Event.where(["start_date <= ? and end_date >= ?", Date.today,
    Date.today])
  @events.sort!{ |x| x.start_date }

  respond_to do |format|
    format.html # index.html.erb
    format.xml { render :xml => @events }
  end
end
```

It's similar to the top_rated method that we defined earlier, but slightly simpler. Notice that the where method call is limiting the events to the ones that are currently going on.

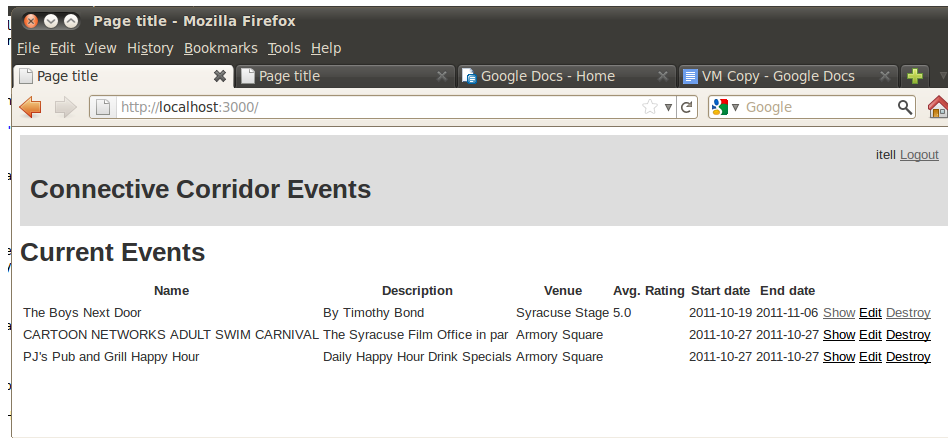
Now, follow the similar steps as Step 6 above and complete work on the rest.

8. Final Touch: Making the default root page

This is the final touch. Currently, our top page (<http://localhost:3000/>) shows a simple place holder public/index.html. In order to replace it with the newly created “Current Events” page, remove the file, and add the following line in the config/routes.rb file. (It can be anywhere in the do block of the file. If you are not sure, put it in the bottom, but right before the line that says end.

```
root :to => "events#current"
```

Phew! That's it. Now access to <http://localhost:3000/>. It should look like below:



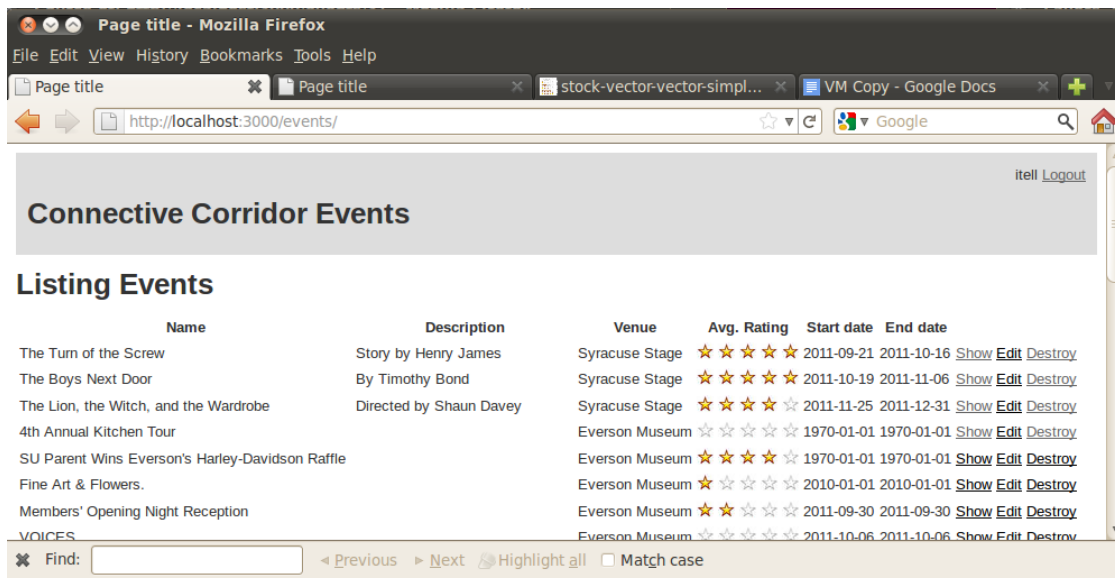
9. Bonus Step 1 (5 points)

Currently, the ratings are all displayed by the number. You can modify the .erb file (_event.file.erb, especially), so that it will show the rating graphically. There are many ways to do it, but easiest way is to use the `times` loop of the number (integer) object and display as many as star images as the rating. Optionally, you can display as many gray stars as 5 minus the rating. In order to display images, you can use a method `image_tag`. For example, `image_tag("star.png")` will display the image file `public/images/star.png`.

You can download the files that I used by the following commands:

```
wget http://sanka.syr.edu/files/star.png
wget http://sanka.syr.edu/files/empty_star.png
```

You are more than welcomed to use different images. It should look like something like following:



10. Bonus Step 2 (5 points)

Currently, there are no menu bars for navigating through the web site, so it is almost impossible to use the system if you don't know how Ruby on Rails works. Fix this situation by adding a navigation bar in the template file (`application.html.erb`). (If you have no idea where to start, use <http://sanka.syr.edu/swt/> as a reference... I am not a CSS/HTML person, so don't laugh at me though.) The menu bar should provide menus (hyperlinks)

for the major feature of the system (e.g. current events, top rated events, all the events, venues, etc.) You are welcomed to add even more actions, too.

11. Submission

1. On your terminal window, in the cc directory, type in the following commands (this will only update your repository):
`git commit -am 'Lab 9'`
2. Then, go to the apps directory (e.g. type in 'cd ..').
3. Type in the following command:
`tar cvfz lab9.tgz cc`
4. Now, launch Firefox on your virtual machine.
5. Go to: <http://sanka.syr.edu/swt/> on Firefox, login, and submit the file that was created in the previous step.