CS169.1x Lecture 7: Rails continued

Fall 2012

Outline

- When things go wrong: Debugging (§4.5)
- Finishing CRUD (§4.8)
- Fallacies, Pitfalls, and Perspectives on SaaS-on-Rails (§4.9–4.11)
- Introduction to BDD and User Stories (§5.1)
- SMART User Stories (§5.2)
- Introducing Cucumber (§5.3)
- Running Cucumber and Introducing Capybara (§5.4)
- Lo-Fi UI Sketches & Storyboards (§5.5)– If time

Debugging SaaS can be tricky

- "Terminal" (STDERR) not always available
- Errors early in flow may manifest much later

  URI➔route➔controller➔model➔view➔render
- Error may be hard to localize/reproduce if affects only some users, routes, etc.

What	Dev?	Prd?
Printing to terminal ("printf debugging")	✔	
Logging	✔	✔
Interactive debugging	✔	

RASP

- ***Debugging is a fact of life: 4 Tips***

1. **R**ead the error message. Really read it.
2. **A**sk a colleague an *informed* question.
3. **S**earch using StackOverflow, a search engine, etc.
   - Especially for errors involving specific versions of gems, OS, ...
4. **P**ost on StackOverflow, class forums, etc.
   - Others are as busy as you. Help them help you by providing *minimal but complete* information

Reading Ruby error messages

- The *backtrace* shows you the call stack (where you came from) at the stop point
  - (demo)
- A very common message:
  `undefined method 'foo' for nil:NilClass`
- Often, it means an assignment silently failed and you didn't error check:

```
@m = Movie.find_by_id(id)  # could be nil
```

Instrumentation (a/k/a "Printing the values of things")

- In views:
  - `debug(@movie)`
  - `@movie.inspect`
- In the log, usually from controller method:

  `logger.debug(@movie.inspect)`
- Don't just use `puts` or `printf`! It has nowhere to go when in production.

Search: Use the Internet to answer questions

- Google it
  - "How do I format a date in Ruby?"
  - "How do I add Rails routes beyond CRUD?"
- Check the documentation
  - api.rubyonrails.org, complete searchable Rails docs
  - ruby-doc.org, complete searchable Ruby docs (including standard libraries)
- Check StackOverflow

If you use `puts` or `printf` to print debugging messages in a production app:

- ☐ Your app will raise an exception and grind to a halt
- ☐ Your app will continue, but the messages will be lost forever
- ☐ Your app will continue, and the messages will go into the log file
- ☐ The SaaS gods will strike you down in a fit of rage

Edit/Update pair is analogous to New/Create pair

- What's the same?
  - 1st action retrieves form, 2nd action submits it
  - "submit" uses redirect (to `show` action for movie) rather than rendering its own view
- What's different?
  - Form should appear with *existing* values filled in: retrieve existing Movie first
  - Form action uses `PUT` rather than `POST`

http://pastebin.com/VV8ekFcn

http://pastebin.com/0drjjxGa

Helper method	URI returned	RESTful Route and action	
movie_path(m)	/movies/1	PUT /movies/:id	update
movie_path(m)	/movies/1	DELETE /movies/:id	destroy

http://pastebin.com/VV8ekFcn

```
def edit
   @movie = Movie.find params[:id]
 end
 def update
   @movie = Movie.find params[:id]
   @movie.update_attributes!(params[:movie])
   flash[:notice] = "#{@movie.title} was successfully
  updated."
   redirect_to movie_path(@movie)
 end
```

http://pastebin.com/0drjjxGa

```
%h1 Edit Existing Movie
= form_tag movie_path(@movie), :method => :put
  do
  = label :movie, :title, 'Title'
  = text_field :movie, 'title'
  -# ...same as new.html.haml!
  -# Soon we will see a way to DRY it out.
  -# ....


  = submit_tag 'Update Movie Info'
```

Destroy is easy

- Remember, destroy is an *instance* method
  - Find the movie first...then destroy it
  - Send user back to Index

```
def destroy
  @movie = Movie.find(params[:id])
  @movie.destroy
  flash[:notice] =
       "Movie '#{@movie.title}' deleted."
  redirect_to movies_path
end
```

If you set an instance variable in a controller method, its value will be retained for how long?

- ☐ This request and all subsequent requests
- ☐ Only this request and the next request
- ☐ Only this request—once the view is rendered, the variable is reset to nil
- ☐ All of the above will work

Pitfall: Fat controllers & views

- Really easy to fall into "fat controllers" trap
  - Controller is first place touched in your code
  - Temptation: start coding in controller method
- Fat views
  - "All I need is this for-loop."
  - "....and this extra code to sort the list of movies differently."
  - "...and this conditional, in case user is not logged in."
- No! Let controller & model do the work.

Designing for Service-Oriented Architecture

- A benefit of *thin* controllers & views: easy to retarget your app to SOA
- Typically, SOA calls will expect XML or JSON (JavaScript Object Notation, looks like nested hashes) as result
- A trivial controller change accomplishes this

http://pastebin.com/bT16LhJ4

http://pastebin.com/bT16LhJ4

```
def create
  @movie = Movie.find params[:id]
  @movie.update_attributes!(params[:movie])
  respond_to do |client_wants|
    client_wants.html {  redirect_to
    movie_path(@movie)  } # as before
    client_wants.xml  {  render :xml => @movie.to_xml    }
  end
end
```

Summary

- Rails encourages you to put real code in models, keep controllers/views thin
  - Reward: easier SOA integration
- Rails encourages convention over configuration and DRY
  - Reward: less code ➔ fewer bugs
- Debugging can be tricky for SaaS
  - Use logging, interactive debugger
  - Next: Behavior-Driven Development to help *reduce bugs* in the first place