Hello Rails : from Zero to CRUD

(Engineering Software as a Service §4.1)

Armando Fox

Connecting Architectural Concepts to Rails apps

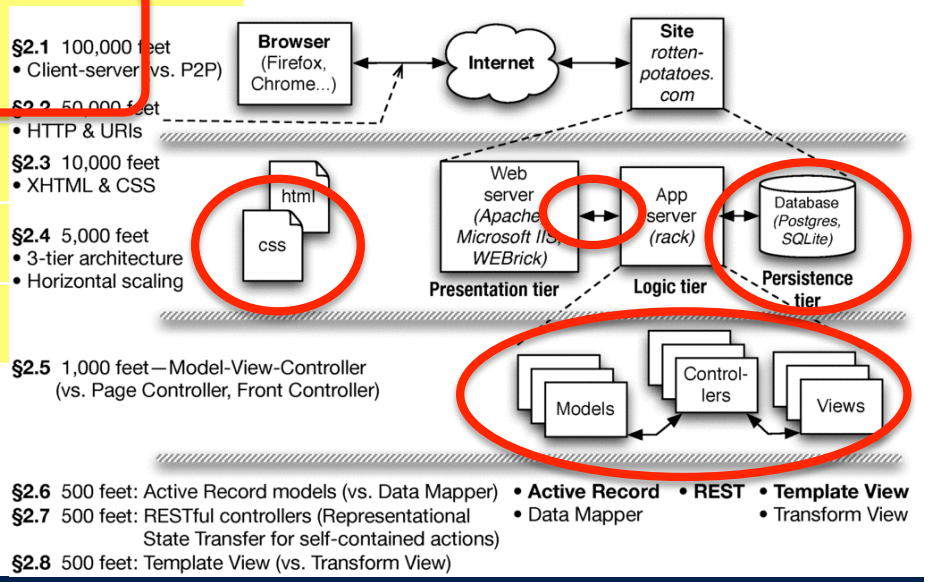
Gemfile	
Rakefile	
app	/models/, views/, controllers/
	/helpers
	/assets/stylesheets/application.cs
config	/routes.rb
	/database.yml
db	/development.sqlite3, test.sqlite3
	/migrate/
log	/development.log, test.log

Rails Models Store Data in Relational Databases (RDBMS)

- Each type of model gets its own database *table*
  - All rows in table have identical structure
  - 1 row in table == one model instance
  - Each column stores value of an *attribute* of the model
  - Each row has unique value for *primary key* (by convention, in Rails this is an integer and is called *id*)

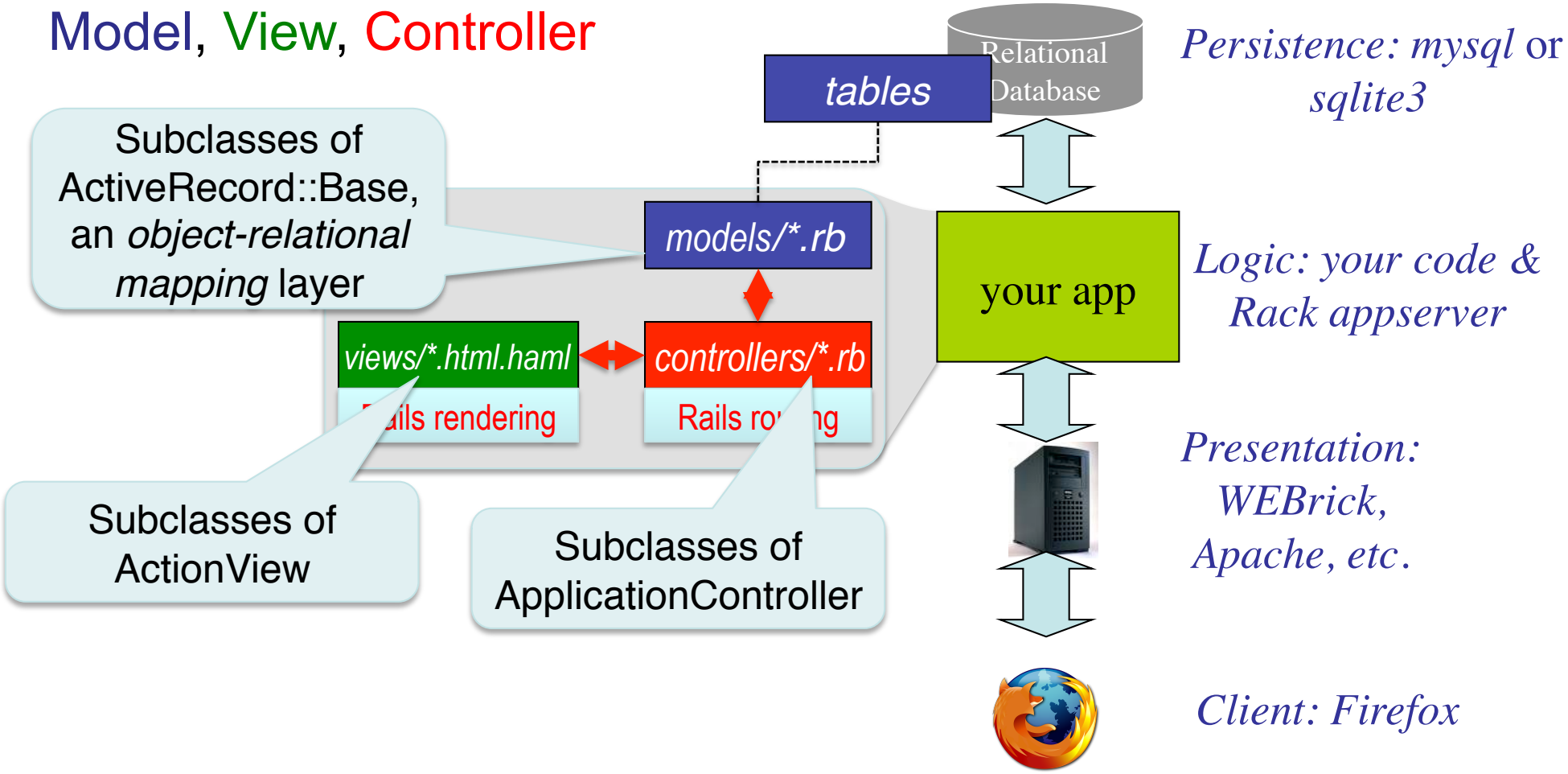
id	rating	title	release_date
2	G	Gone With the Wind	1939-12-15
11	PG	Casablanca	1942-11-26
...	...	...	...
35	PG	Star Wars	1977-05-25

- *Schema:* Collection of all tables and their structure



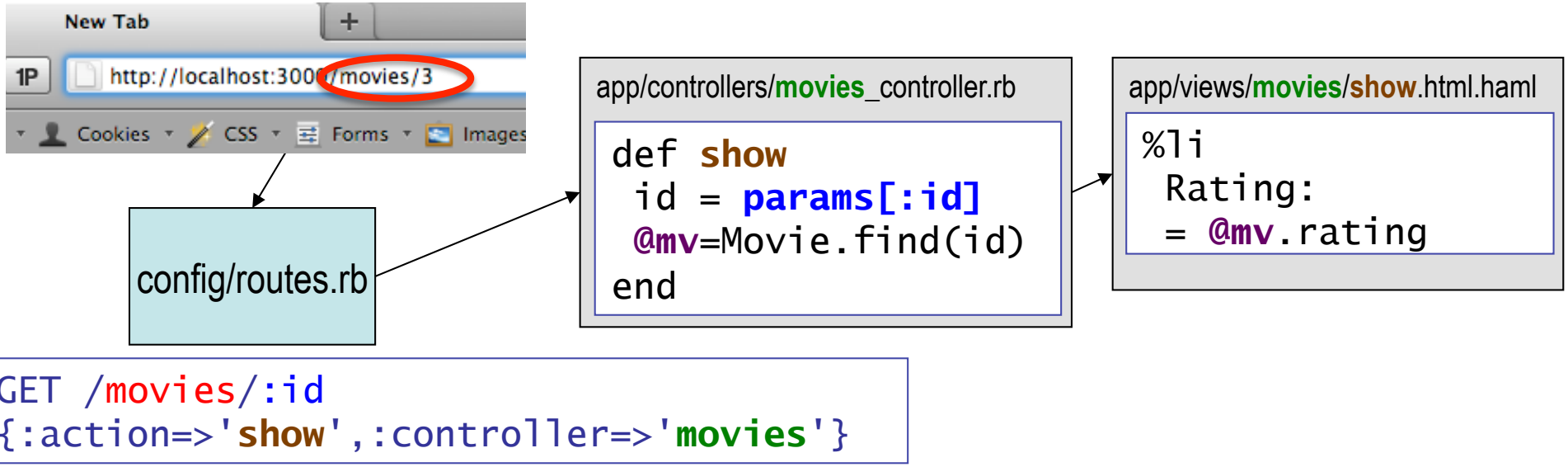
Rails as an MVC Framework

Model, View, Controller



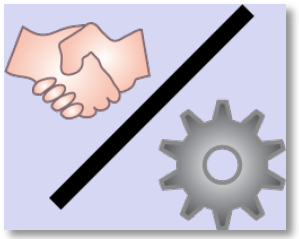
A trip through a Rails app

1. Routes (in `routes.rb`) map incoming URL's to *controller actions* and extract any optional *parameters*
   - Route's "wildcard" parameters (eg `:id`), plus any stuff after "`?`" in URL, are put into `params[]` hash accessible in controller actions
2. Controller actions set *instance variables,* visible to *views*
   - Subdirs and filenames of `views/` match controllers & action names
3. Controller action eventually *renders* a view



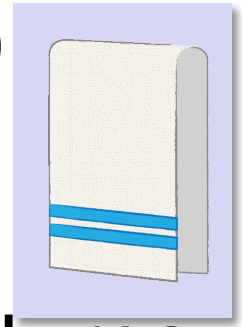
Rails Philosophy

- *Convention over configuration*
  - If naming follows certain conventions, no need for config files



`MoviesController#show` in `movies_controller.rb`
➔ `views/movies/show.html.haml`

- Don't Repeat Yourself (DRY)
  - mechanisms to extract common functionality



- Both rely heavily on Ruby features:
  - introspection and metaprogramming
  - blocks (closures)
  - modules (mix-ins)

Why must every interaction with a SaaS app eventually cause something to be rendered?

- ☐ Because of convention over configuration
- ☐ Because HTTP is a request-reply protocol
- ☐ Because Model-View-Controller implies that every action renders its own View
- ☐ All of the above

When things go wrong: Debugging

(Engineering Software as a Service §4.5)

Armando Fox

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.***
- **R**ead the error message. Really read it.
- **A**sk a colleague an *informed* question.
- **S**earch using StackOverflow, a search engine, etc.
  - Especially for errors involving specific versions of gems, OS, etc.
- **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

- 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
@m.title   # will fail: 'undefined method'
```

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

Use `rails console`

- Like *irb*, but loads Rails + your app code
- But context is still not quite right for "peeking into" controllers and views
  - Controllers rely on environment prepared by presentation tier
  - Views rely on context set up by controllers
- Big guns: ruby-debug (demo shortly)

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

Models: ActiveRecord Basics

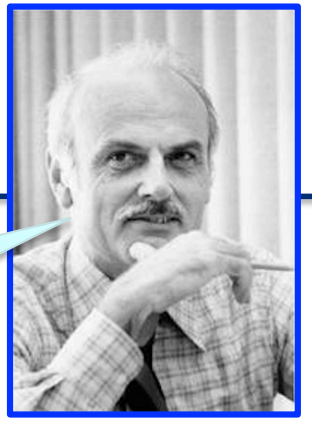
(Engineering Software as a Service §4.3)

Armando Fox

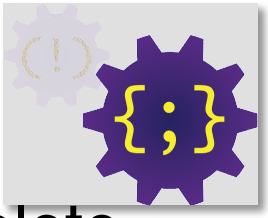
How can language features simplify design & implementation of design patterns?

In this case, *Active Record,* which "bridges the gap" between in-memory Ruby objects & their stored representation in a database.

CRUD in SQL



- Structured Query Language (SQL) is the query language used by RDBMS's
- Rails *generates* SQL statements at runtime, based on your Ruby code
- 4 basic operations on a table row:
  Create, **R**ead, **U**pdate attributes, **D**elete

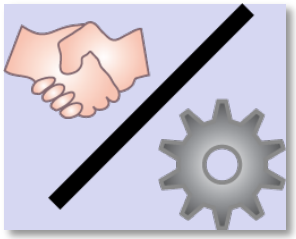


```
INSERT INTO users
   (username, email, birthdate)
   VALUES ("fox", "Fox@cs.berkeley.edu", "1968-05-12"),
      "patterson", "pattrsn@cs.berkeley.edu", "????")
SELECT * FROM users
   WHERE (birthdate BETWEEN "1987-01-01" AND "2000-01-01")
UPDATE users
   SET email = "armandofox@gmail.com"
   WHERE username="fox"
DELETE FROM users WHERE id=1
```

The Ruby side of a model

- Subclassing from `ActiveRecord::Base`
  - "connects" a model to the database
  - provides CRUD operations on the model

http://pastebin.com/ruu5y0D8

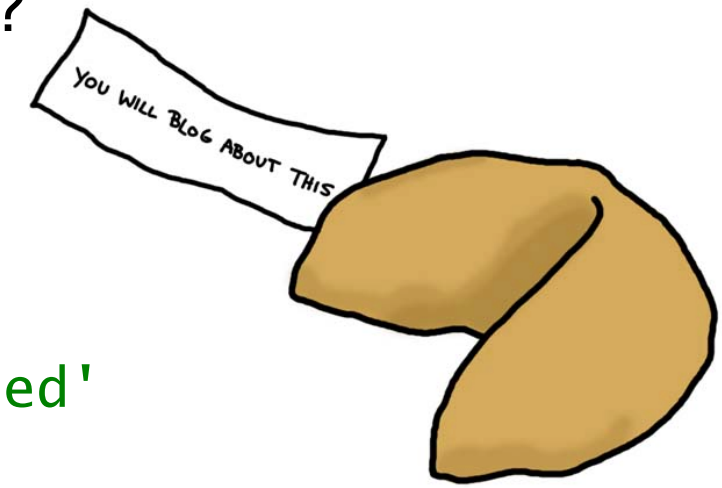


- Database table name derived from model's name: `Movie`→`movies`
- Database table column names are getters & setters for model attributes
- *Observe: the getters and setters do not simply modify instance variables!*

Creating: new ≠ save

- Must call `save` or `save!` on an AR model instance to actually save changes to DB
  - '!' version throws exception if operation fails
  - `create` just combines `new` and `save`
- Once created, object acquires a primary key (`id` column in every AR model table)
  - if `x.id` is `nil` or `x.new_record?` is true, `x` has never been saved
  - These behaviors inherited from `ActiveRecord::Base`—not true of Ruby objects in general

Assume table `fortune_cookies`
has column `fortune_text`
Which of these instance methods of
`FortuneCookie < ActiveRecord::Base`
will **not** return a silly fortune (if any)?



- ☐
```
def silly_fortune_1
  @fortune_text + 'in bed'
end
```
- ☐
```
def silly_fortune_2
  self.fortune_text + 'in bed'
end
```
- ☐
```
def silly_fortune_3
  fortune_text + 'in bed'
end
```
- ☐ They will all return a silly fortune

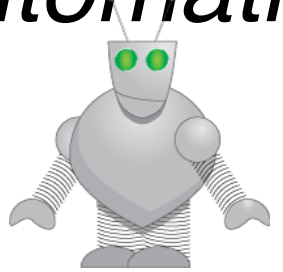
Databases & Migrations

(Engineering Software as a Service §4.2)

Armando Fox

Your customer data is golden!

- How do we avoid messing it up when experimenting/developing new features?
- How do we track and manage *schema changes* ?
- …the answer to both is *automation!*

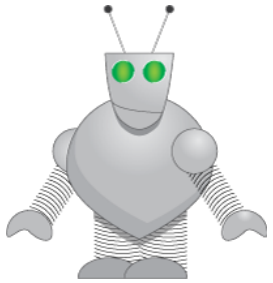


Multiple environments, multiple databases

- Rails solution: development, production and test *environments* each have own DB
  - Different DB types appropriate for each!
- How to make *changes* to DB, since will have to repeat changes on production DB?
- Rails solution: *migration*—script describing changes, portable across DB types

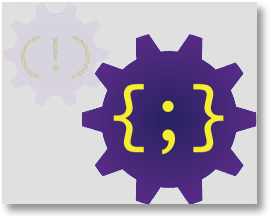
Migration Advantages

- Can identify each migration, and know which one(s) applied and when
  - Many migrations can be created to be *reversible*
- Can manage with version control
- *Automated == reliably repeatable*



- Theme: *don't do it—automate it*
  - *specify* what to do, create tools to automate

Meet a Code Generator



```
rails generate migration CreateMovies
```

http://pastebin.com/VYwbc5fq

- Note, this just *creates* the migration. We haven't *applied* it.
- Apply migration to development:
  ```
  rake db:migrate
  ```
- Apply migration to production:
  ```
  heroku rake db:migrate
  ```
- Applying migration also records in DB itself which migrations have been applied

Rails Cookery #1

- Augmenting app functionality ==
  adding models, views, controller actions

To *add a new model* to a Rails app:

  – (or change/add attributes of an existing model)

1. Create a migration describing the changes:
   `rails generate migration` (gives you boilerplate)
2. Apply the migration: `rake db:migrate`
3. If new model, create model file
   `app/models/`*model*`.rb`
4. Update test DB schema: `rake db:test:prepare`

Based on what you've seen of Rails, what kind of object is *likely* being yielded in the migration code:

```
def up
  create_table 'movies' do |t|
    t.datetime 'release_date' ...
  end
end
```

- ☐ An object representing a database
- ☐ An object representing an instance of a model
- ☐ An object representing a table
- ☐ Give me a break, it could be anything

Models: Finding, Updating, Deleting

(Engineering Software as a Service §4.3)

Armando Fox

Read: finding things in DB

- class method `where` selects objects based on attributes

```
Movie.where("rating='PG'")
Movie.where('release_date < :cutoff and
      rating = :rating',
      :rating => 'PG', :cutoff => 1.year.ago)
Movie.where("rating=#{rating}") # BAD IDEA!
```

- Can be chained together efficiently

```
kiddie = Movie.where("rating='G'")

old_kids_films =
  kiddie.where("release_date < ?",30.years.ago)
```

Read: `find_*`

- find by id:

```
Movie.find(3) #exception if not found
Movie.find_by_id(3) # nil if not found
```

- dynamic attribute-based finders:

```
Movie.find_all_by_rating('PG')
Movie.find_by_rating('PG')
Movie.find_by_rating!('PG')
```

Update: 2 ways

- Let `m=Movie.find_by_title('The Help')`
- Modify attributes, then save object

```
m.release_date='2011-Aug-10'
m.save!
```

- Update attributes on existing object

```
m.update_attributes
    :release_date => '2011-Aug-10'
```

- Transactional: either all attributes are updated, or none are

Deleting is straightforward

- Note! destroy is an *instance* method

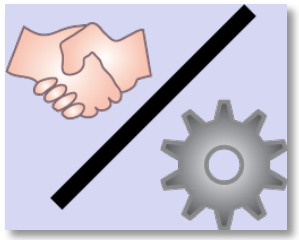
```
m = Movie.find_by_name('The Help')
m.destroy
```

- There's also `delete`, which doesn't trigger *lifecycle callbacks* we'll discuss later (so, avoid it)
- Once an AR object is destroyed, you can access *but not modify* in-memory object

```
m.title = 'Help'  # FAILS
```

Summary: ActiveRecord intro

- Subclassing from `ActiveRecord::Base` "connects" a model to database
  - **C** (`save`/`create`), **R** (`where`, `find`), **U** (`update_attributes`), **D** (`destroy`)
- Convention over configuration maps:
  - model name to DB table name
  - getters/setters to DB table columns
- Object in memory ≠ row in database!
  - `save` must be used to persist
  - `destroy` doesn't destroy in-memory copy



Suppose we've done

```
movie = Movie.where("title='Amelie'")
```

Then *another app* changes the movie's title *in the database table* directly. Just after that instant, the value of `movie`:

- ☐ will be updated automatically because an ActiveRecord model "connects" your app to the database
- ☐ will be updated automatically because of ActiveRecord's use of metaprogramming
- ☐ will *not* be updated automatically, but can be updated manually by re-executing

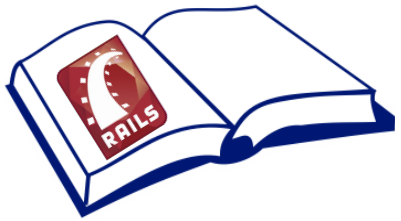
  ```
  movie = Movie.where("title='Amelie'")
  ```

- ☐ may be undefined or implementation-dependent

Controllers & Views

(Engineering Software as a Service §4.4)

Armando Fox



Rails Cookery #2

- To *add a new action* to a Rails app

1. Create *route* in `config/routes.rb` if needed
2. Add the *action* (method) in the appropriate `app/controllers/*_controller.rb`
3. Ensure there is something for the action to *render* in `app/views/`*model*`/`*action*`.html.haml`

- We'll do Show action & view (book walks through Index action & view)

MVC responsibilities

- *Model:* methods to get/manipulate data

```
Movie.where(...), Movie.find(...)
```

- *Controller:* get data from Model, make available to View

```
def show
  @movie = Movie.find(params[:id])
end
```

Instance variables set in Controller available in View

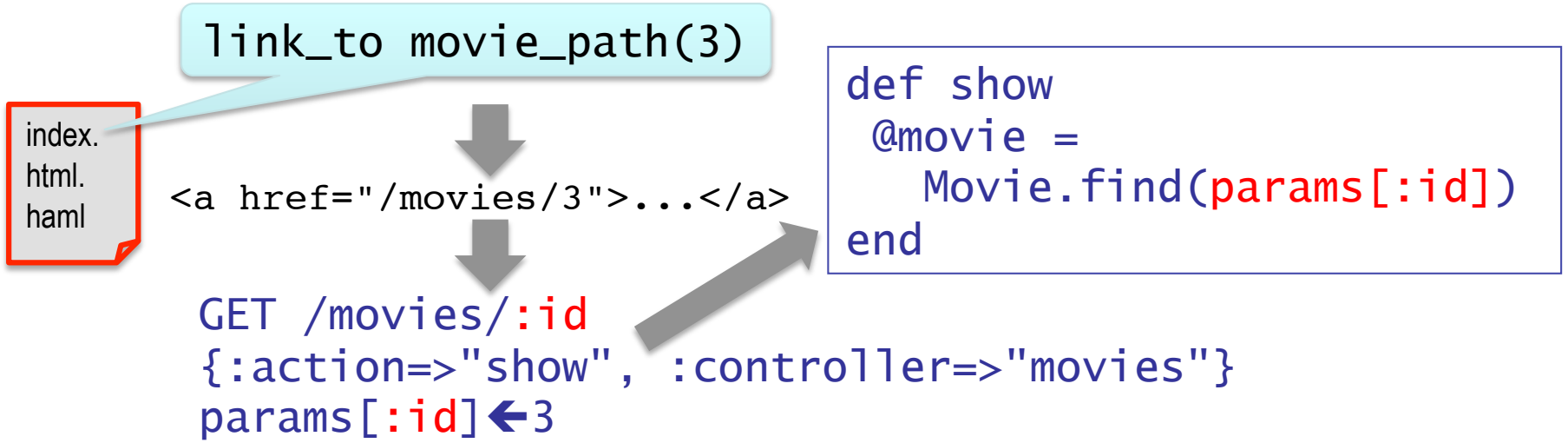
Absent other info, Rails will look for app/views/movies/show.html.haml

- *View:* display data, allow user interaction
  - Show details of a movie (description, rating)
- But…
  - What else can user do from this page?
  - How does user get to this page?

http://pastebin.com/kZCB3uNj

How we got here: URI helpers

Helper method	URI returned	RESTful Route and action	
movies_path	/movies	GET /movies	index
movies_path	/movies	POST /movies	create
new_movie_path	/movies/new	GET /movies/new	new
edit_movie_path(m)	/movies/1/edit	GET /movies/:id/edit	edit
movie_path(m)	/movies/1	GET /movies/:id	show
movie_path(m)	/movies/1	PUT /movies/:id	update
movie_path(m)	/movies/1	DELETE /movies/:id	destroy



What else can we do?

- How about letting user return to movie list?
- RESTful URI helper to the rescue again:
- `movies_path` with no arguments links to Index action

```
=link_to 'Back to List', movies_path
```

Helper method	URI returned	RESTful Route and action	
movies_path	/movies	GET /movies	index
movies_path	/movies	POST /movies	create
new_movie_path	/movies/new	GET /movies/new	new
edit_movie_path(m)	/movies/1/edit	GET /movies/:id/edit	edit
movie_path(m)	/movies/1	GET /movies/:id	show
movie_path(m)	/movies/1	PUT /movies/:id	update
movie_path(m)	/movies/1	DELETE /movies/:id	destroy

ESaaS, Fig. 4.7

Which statements are TRUE:

a) A route consists of **both** a URI and an HTTP method

b) A route URI **must** be generated by Rails URI helpers

c) A route URI **may** be generated by Rails URI helpers

- ☐ Only (a) is true
- ☐ Only (c) is true
- ☐ Only (a) and (b) are true
- ☐ Only (a) and (c) are true

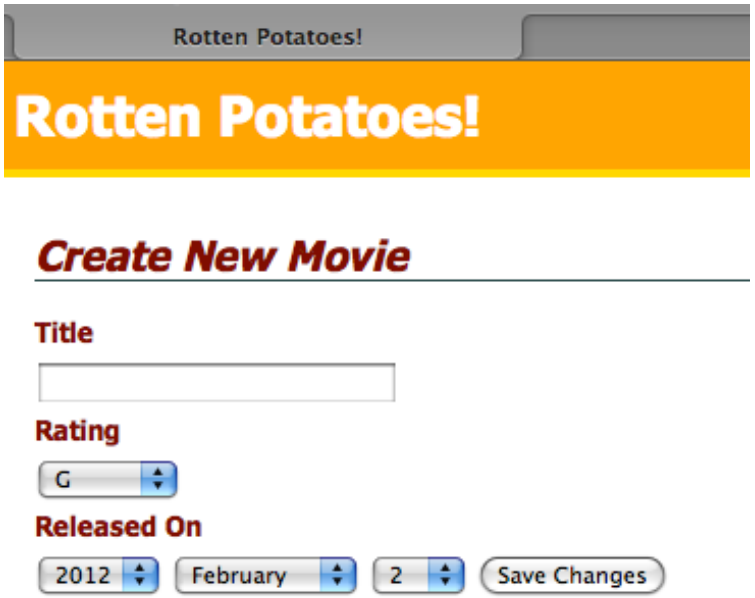
Forms

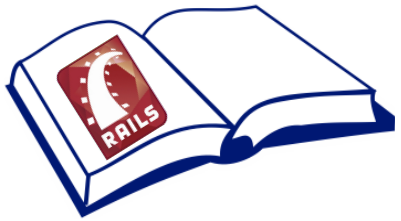
(Engineering Software as a Service §4.6)

Armando Fox

Dealing with forms

- Creating a resource usually takes 2 interactions
  - `new:` Retrieve blank form
  - `create:` Submit filled form
- How to generate/display?
- How to get values filled in by user?
- What to "return" (render)?





Rails Cookery #3

- To create a new submittable form:

1. Identify the action that serves the form itself
2. Identify the action that receives *submission*
3. Create routes, actions, views for each

- Form elements' `name` attributes will appear as keys in `params[]`
- Helpers provided for many common elements

Creating the Form

- Anatomy of a form in HTML
  - the *action* and *method* attributes (i.e., the route)
  - only *named* form inputs will be submitted
- Generating the form in Rails
  - often can use URI helper for *action,* since it's just the URI part of a route (still need *method*)
  - *form field helpers* (see api.rubyonrails.org) generate conveniently-named form inputs

http://pastebin.com/k8Y49EhE

http://pastebin.com/3dGWsSq8

Which of these would be valid for generating the form that, when submitted, would call the Create New Movie action?

- ☐
```
= form_tag movies_path do
  ... end
```
- ☐
```
%form{:action => movies_path,
      :method => :post}
```
- ☐
```
%form{:action => '/movies',
      :method => 'post'}
```
- ☐ All of the above

Redirection, the Flash and the Session

(Engineering Software as a Service §4.7)

Armando Fox

Receiving the form

- A neat trick: use debugger to inspect what's going on
  - start with `rails server --debugger`
  - insert `debugger` where you want to stop
  - details & command summary: ESaaS §4.7
- To notice: `params[:movie]` *is a hash,* because of the way we named form fields
  - Conveniently, just what `Movie.create!` wants

What view should be rendered for create action?

- Idiom: *redirect* user to a more useful page.
  - e.g., list of movies, if create successful
  - e.g., New Movie form, if unsuccessful
- Redirect triggers a *whole new HTTP* request
  - How to inform user *why* they were redirected?
- Solution: `flash[]`—quacks like a hash that *persists until end of **next** request*
  - `flash[:notice]` conventionally for information
  - `flash[:warning]` conventionally for "*errors*"

Flash & Session

- `session[]`: like a hash that persists forever
  - `reset_session` nukes the whole thing
  - `session.delete(:some_key)`, like a hash
- By default, cookies store *entire contents* of `session` & `flash`
  - Alternative: store Rails sessions in DB table
  - (Google "rails session use database table")
  - Another alternative: store sessions in a "NoSQL" storage system, like *memcached*

Ben Bitdiddle says: "You can put arbitrary objects (not just "simple" ones like ints and strings) into the `session[]`." What do you think?

- ☐ True—knock yourself out!
- ☐ True—but a bad idea!
- ☐ False, because you can't put arbitrary objects into a hash
- ☐ False, because `session[]` isn't really a hash, it just quacks like one

Finishing CRUD

(Engineering Software as a Service §4.8)

Armando Fox

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

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
- ☐ It depends on whether the instance variable was declared static

Fallacies, pitfalls, and perspectives on SaaS-on-Rails

(Engineering Software as a Service §4.9–4.11)

Armando Fox

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! That's for model, controller, helpers

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

Which steps are ALWAYS required when adding a new action 'foo' to the Movie model of a Rails app:
(a) Ensure there is a template to render in `app/views/movies/foo.html.haml` (or `.html.erb`, etc)
(b) Ensure a route exists in `config/routes.rb`
(c) Implement helper method to generate necessary route-helper URIs

- ☐ Only (a) and (b)
- ☐ Only (b)
- ☐ Only (b) and (c)
- ☐ Only (a) and (c)