CS 416
Web Programming

Ruby on RAILS
Chapter 3

Dr. Williams
Central Connecticut State University

A look ahead

- How views work in rails
- Avoiding duplication
- Automated testing

Getting started

- Create new app: `sample_app`
- Copy Gemfile from Listing 3.2 in book
- Then install

```
bundle update
bundle install --without production
bundle update
```

- Then create git repository

```
git init
git add .
git commit -m "Initial version"
git remote add origin….
git push -u origin –all
```

- Then start a new branch for this work

```
git checkout -b static-pages
```

Generate controller

- Create a controller to handle pages that don’t need data access
  - First argument the name of the controller
  - Remaining arguments the actions/views to attach

```
rails generate controller StaticPages home help
```

(note this is identical to specifying generate `static_pages`)

Push through git

```
git add .
git commit -m "Add Static Pages Controller"
```

git push -u origin static-pages

“Undo” for rails

- The undo for generation in Rails is the destroy command:

```
rails generate controller StaticPages home help
rails destroy  controller StaticPages home help

rails generate model User name:string email:string
rails destroy model User
```

Database equivalent is rollback or migrate to specific prior version

```
rails db:migrate
rails db:rollback
rails db:migrate VERSION=0
```

Rails and testing

- Designed to make regression testing very easy
- Create tests as go, at any point can rerun tests to see if broke something previously working
- With test-driven development (TDD) you know when you have completed each development goal

When to write tests – guidelines if not TDD

- Always write tests of security model
- Code that is especially likely to break
- Anytime bug is found write test to protect against it in future
- Write tests before refactoring code
- Lean against writing tests against detailed HTML structure as likely to change in the future

Tests

From test/controllers/static_pages_controller_test.rb

```
class StaticPagesControllerTest < 
ActionDispatch::IntegrationTest

  test "should get home" do
    get static_pages_home_url
    assert_response :success
  end

  test "should get help" do
    get static_pages_help_url
    assert_response :success
  end
end
```

Execute the tests:

rails db:migrate RAILS_ENV=test

`rails test` ***or*** **`rails t`**

Passes

Test first approach

- Create a new test case for *about* page in:

test/controllers/static_pages_controller_test.rb

- Sanity check of test

```
rails test
```

Fails (as expected)

Add *about* by hand

- Add route
  - *Still fails*
- Add controller action for *about*
  - *Still fails*
- Create view: *app/views/static_pages/about.html.erb*
  - ***Passes***
- Add some content for page

In Groups - Compare and contrast basics

- Online retailer has multiple product categories and associated with each of the categories are multiple products (a product can only belong to one category). There should be:
  - A page to display a list of categories
  - A page displays a list of products, and
  - A page that displays a single category showing its detail and all of its products.
- Design an MVC architecture solution for this same description in both Java and Rails including DB design.
- Create bullet point list of similarity/differences at each component.
- For the last page diagram a step-by-step flow for each architecture, pay particular attention to controller, model, view interaction for this call

Dynamic pages and refactoring commonality

- Want to add common style title for each page (home, help, and about):

"***Home* | Ruby on Rails Tutorial Sample App**“

- Follow TDD approach add test condition for each:

```
assert_select "title", "Home | Ruby on Rails
Tutorial Sample App"
```

- Clean up redundancy with test setup and variables

```
 def setup
    @base_title = "Ruby on Rails Tutorial Sample App"
 end

assert_select "title", "Home | #{@base_title}"
```

- Run test
  - Fails (as expected)

Rails layouts

- Common on most websites to have similar look and feel, common includes etc

- Layouts designed to provide shell for commonality and only page specific area needs to be handled by the view

- See *app/views/layouts/application.html.erb*
  - Key component:

  <%= yield %>

Making layouts dynamic

- Key component:

  <%= yield %>

  - Tells it to take output of view and insert it here
- Outputting specific passed elements

  <%= yield(:title) %>

  - Passing elements

  <% provide(:title, "About") %>

- Now put it together…

- **Passes**

Wrap up

- Set root route to home page

```
root 'static_pages#home'
```

- Commit

```
git add .
git commit -m "Finish static pages"
```

- Merge branch into master

```
git checkout master
git merge static-pages
git push
```

Test and push to cloud

```
rails test
git push heroku
```

In Final project groups - sketch solution

- Identify an idea for your final project – or a couple potential ideas

On the given paper (to hand in at end of class)

- Design an MVC architecture solution for your project in both Java and Rails including DB design.
- Create bullet point list of similarity/differences at each component.
- For the last page diagram a step-by-step flow for each architecture, pay particular attention to controller, model, view interaction for this call
  - Note for Rails we haven't covered some of the elements you will need so make a guess as to where it would be from a MVC perspective based on the roles of each component