Day 15: Web Applications

suggested reading:
Agile Web Development with Rails (3e), by Ruby, Thomas, & Heinemeier Hansson

but I may be biased... (^_^)
HOMEWORK
What a Tangled Web We Weave

• In the beginning... static web pages
• Dynamic pages add interactivity
  – Forms
  – Banner ads
  – JavaScript
  – Web applications
  – Web 2.0

• Code runs on server, browser, both
Web Architecture

Browser → HTTP Request → Web Server → Application Code

HTTP Response

Internet

Database

(as needed)
HTTP Request

http://foo.com/page?name1=val1&n2=2

- Path: /page
- Parameters (from URL, form, etc.):
  - name1 => “val1”
  - n2 => “2”
- Protocol version (1.0, 1.1)
- Stuff (req. method, headers, …)
- Server may add more to “request”
HTTP Response

- Status (OK or type of failure)
- Headers (type, length, date, etc.)
- Payload
  - HTML
  - image file
  - CSS
  - PDF
  - etc.
HTML

<h1>My Cat</h1>
<p>Picture of <em>my cat</em>:</p>
<img src="cat.jpg">

My Cat

Picture of my cat:
Dynamic Web Page

• Change HTML based on input
• Client-side (JavaScript, Flash, ...):
  – Code loads along with page
  – Server not required after page load
• Server-side
  – Code runs on server only
  – Changes HTML before page is sent
  – Does not affect page after it’s loaded
Web Architecture

Browser → HTTP Request → Web Server → Application Code → Database

Internet

HTTP Response → Browser

(as needed)
**CGI** (Common Gateway Interface)

- Protocol between web server & app.
- Handles requests for dynamic pages
- Somewhat dated (but see FastCGI)
- Details:
  - Input via environment and stdin
  - Output via stdout
  - Output is: HTTP headers, blank, HTML
- Very quickly, gets tedious to code
Better Web Apps

• Use libraries to make things easy(er)
• Package request headers & params
• Make HTML generation easier
• Allow mixed HTML & code

• Examples:
  – Perl CGI
  – PHP
  – ASP
use CGI;
my $query = new CGI;
my $name = $query->param('name');
print $query->header,
    $query->start_html('hello'),
    $query->h1("Hello, $name"),
    $query->end_html;

• OK for short code, but how do we organize longer applications?
Model-View-Controller (MVC)

• General code pattern
• Originally for GUI apps
• Helps organize code logically
Model

- Domain logic
- Domain data
- Data persistence (usu., database)

```perl
my $greeting = $salutation + $name;
my $expiration = today() + 14;
if (record_found()) {
    $total = record_total();
} else {
    $total = -1;
}
```
View

• Renders model element(s)
• Creates user interface

```perl
print $q->h3($greeting);
my $disp_tot =
  ($total >= 0 ? $total : 'n/a');
print $q->p("Total: $disp_tot");
print $q->p("Expires ".
  strftime(''%M/%D/%Y'', $expiration));
```
View Templates

• Simplify views using mostly HTML

```html
... 
<h1><%= @page_title %></h1>
<p>Hello, <%= @name %>!</p>
<% if @friends.size > 0 -%>
<p>
Friends logged in:
  <%= @friends.join(', ', ) %>
</p>
<% end -%>
```
Controller

- Handles incoming requests
- Routes to correct code
- May initiate model changes
- May pick correct view

```php
if ($q->param('update')) {
    update_totals();
}
render('summary');
```
Web Application Frameworks

• Next step up from a library
• Provides:
  – library functions
  – code organization scheme (e.g., MVC)
  – template system
  – database connectivity
  – object-relational mapping
  – INTEGRATION!
• Mason, Rails, Django, & dozens more
Live Rails Example