Web Applications & Ruby on Rails

SWEN 250

Personal Software Engineering
How Websites Work

• Browsers send HTTP requests to a Server
• Servers send back HTTP responses
• HTTP requests & responses are newline-delimited strings with:
  • Queries, parameters, body
  • Metadata (e.g. headers)
• Main requests: GET and POST.

• Thus: *Web servers are just fancy string processors*
Example HTTP Requests

• GET requests are when you click a link
  (the *vast* majority of HTTP requests are GET)
  
  http://www.example.com?p1=arg&p2=arg2

• May have query parameters in the URL:

GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.se.rit.edu/~swen-250
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

• POST requests are when you request data
  be *modify* data on the server
A Typical Web App Flow

1. User types in a URL to a website, ourexample.com
   *i.e. HTTP GET request to ourexample.com*

2. Server responds with an HTTP response and HTML data.

3. Browser parses the HTML, page starts loading
   *i.e. Browser parses HTTP response, renders HTML, forms new GET requests for images*  
   (<img src=url>), CSS, Javascript, advertisements, and all the other assets – rendering those as they come in.

4. User sees login screen
   *i.e. an HTML form,*
   <form action="/login">
   Username: <input type="text" name="username">
   Password: <input type="password" name="password">
   </form>

5. User enters password and hits “Login”
   *i.e. An HTTP POST request goes to ourexample.com, with page "/login" and parameters “username” and “password”.

6. User sees their home page
   *i.e. webapp checks password, determines which page to render next*
But how does the server work?

• A web server will need to
  • Know what to do at each page
  • Get and store data in a persistent way
  • Handle concurrency
  • Track user flow from one request to another (i.e. HTTP by itself is *stateless*)
  • etc. etc. etc.

• Web *applications* are typically built within *frameworks* that do most of the HTTP, Database, and even HTML work for you
  • Ruby on Rails, Django, Express.js, Struts, Play, Spring, etc.
Most WebApps Use Model-View-Controller

HTTP requests → internet → Router → Controller → Views

Controller renders returns

Queries to Models

Database

Use sparingly!!
MVC Purposes

• Models handle *persistence*
  • Handle queries, storage, CRUD (Create, Read, Update, Delete), transactions, data integrity

• Views handle *presentation*
  • Arranging outputs in HTML, CSS, Js, responsive design,
  • A view can also be simply an API call accessible via HTTP requests (e.g. REST)

• Controllers handle *logic* (aka “business logic”)
  • Routing, behavior, authentication, which queries to call
  • Glues everything together
MVC with Ruby on Rails

- **Models**: ActiveRecord
  - Object-relational mapper
  - Construct SQL queries based on Ruby methods
  - e.g. a Thought object represents a row in the thoughts table.

- **Views**: ERB templates
  - Embed Ruby into HTML
  - Look for the `<% %>` and `<%= %>`

- **Controllers**: Ruby Classes
  - Pages → Methods
  - Render views: respond_to

- **Routing**: routes.rb
Key Rails facts for SWEN 250

• We are running on *Windows, not* nitron.
• Install your app via the z: drive in your folder so you can still commit.
• Use PuTTY for Git, use Windows cmd for rails server
• You must do this in the lab. If you choose to install Rails on your home machine, you are on your own.