Introduction to Web Frameworks
Confusion, or frustration?

• Office hours starting tomorrow!
• PIAZZA
• Ask us for help!
TODAY

• Get ready to write some code!
• A bit more background knowledge:
  • Web frameworks
  • Git and dev. environments
• Your first framework: Node.js!
LET ME KNOW HOW YOU’RE DOING

6148clicker.meteor.com
How are websites structured?

How do you “talk” to a website?

How do you make a website?
A brief history lesson
The first web page

Tim Berners-Lee, 1989
The first web page

“... a wide area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents”

http://info.cern.ch/hypertext/WWW/TheProject.html
Documents of a markup language

... connected with hyperlinks
Popularity increases

http://www.wonder-tonic.com/geocitiesizer/
If your site for 6.148 looks like that, you should probably fix it.
Let’s look at some of the technical details
turns out many of the ideas of the ‘90s are still relevant today!
<table>
<thead>
<tr>
<th>CONTENT</th>
<th>static pages, host-generated content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>CONTENT</td>
<td>static pages, host-generated content</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>BROWSER</td>
<td>loads the page, displays static content</td>
</tr>
<tr>
<td>CONTENT</td>
<td>static pages, host-generated content</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>BROWSER</td>
<td>loads the page, displays static content</td>
</tr>
<tr>
<td>WEB SERVER</td>
<td>simple request/response model</td>
</tr>
</tbody>
</table>
Accessing a website

client (you)
Accessing a website

client (you)

server (xkcd)
Accessing a website

HTTP request: GET xkcd.com

client (you) → server (xkcd)
Accessing a website

HTTP request: GET 107.6.106.82

client (you)  ->  server (xkcd)
What’s a server?

The 10,000 foot view...
What’s a server?

The 10,000 foot view...

loop indefinitely:
  wait for incoming connection
  accept incoming connection
  handle incoming connection
  send response back
What’s a server?

The 10,000 foot view...

loop indefinitely:
wait for incoming connection
accept incoming connection
handle incoming connection
send response back

In our simple world of static files ... just find the file requested, and send its contents back!
Accessing a website

client (you)

HTTP request: GET xkcd.com

HTTP response: web content (HTML, CSS, JavaScript)

server (xkcd)
The server has no idea what it’s sending back.
It’s just text.

The client makes sense of the response.
That’s the client-server model, simplified, in a nutshell.
That’s the client-server model, simplified, in a nutshell.

strong modularity between client and server
clients served simultaneously, unaware of each other
Questions?
So how do we get from here to what we have today?
Well, what do we have today?
Well, what do we have today?
• *dynamic* web sites, “*web applications*”
• dynamic web sites, “web applications”

• user-generated content; personalization
• *dynamic* web sites, “web applications”
• user-generated content; personalization
• user *interaction* with content
• *dynamic* web sites, "web applications"

• user-generated content; personalization

• user *interaction* with content

• the web as a *platform* or *service*
A KEY BREAKTHROUGH
SERVER SIDE SCRIPTS

(connected to databases*)

*we’ll talk more about these on Friday
early Facebook, c. 2004
built on top of a “web stack” of PHP + MySQL
Recall our server...

loop indefinitely:
  wait for incoming connection
  accept incoming connection
  handle incoming connection
  send response back

handle incoming connection:
  run the program dictated by the request
  return the response of the program
Server-side scripts

What is this program?
Server-side scripts

What is this program?

if user is logged in

print "<p>hello, $user_name</p>"

else

print "<p>Login here:</p>"
Server-side scripts

Most important thing to remember:

The output of your program is something that the client can understand.
Server-side scripts

Most important thing to remember:

The *output* of your program is something that the client can understand:

**HTML, CSS, Javascript!**
Hello, web 2.0!

What is it? No one really knows.
Hello, web 2.0!

The proliferation of “web apps” characterized by:

- dynamic content
- personalization
- user-generated content
- the role of “social media” and “big data”
sounds a bit like 6.148, huh?
<table>
<thead>
<tr>
<th></th>
<th>early web</th>
<th>modern web-app</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CONTENT</strong></td>
<td>static pages, host-generated content</td>
<td></td>
</tr>
<tr>
<td><strong>BROWSER</strong></td>
<td>loads the page, displays static content</td>
<td></td>
</tr>
<tr>
<td><strong>WEB SERVER</strong></td>
<td>simple request/response model</td>
<td></td>
</tr>
<tr>
<td></td>
<td>early web</td>
<td>modern web-app</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><strong>CONTENT</strong></td>
<td>static pages, host-generated content</td>
<td>dynamic server-generated pages, user content</td>
</tr>
<tr>
<td><strong>BROWSER</strong></td>
<td>loads the page, displays static content</td>
<td></td>
</tr>
<tr>
<td><strong>WEB SERVER</strong></td>
<td>simple request/response model</td>
<td></td>
</tr>
<tr>
<td>CONTENT</td>
<td>early web</td>
<td>modern web-app</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>----------------</td>
</tr>
<tr>
<td>CONTENT</td>
<td>static pages, host-generated content</td>
<td>dynamic server-generated pages, user content</td>
</tr>
<tr>
<td>BROWSER</td>
<td>loads the page, displays static content</td>
<td>provides a platform for user interaction</td>
</tr>
<tr>
<td>WEB SERVER</td>
<td>simple request/response model</td>
<td></td>
</tr>
<tr>
<td></td>
<td>early web</td>
<td>modern web-app</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td><strong>CONTENT</strong></td>
<td>static pages, host-generated content</td>
<td>dynamic server-generated pages, user content</td>
</tr>
<tr>
<td><strong>BROWSER</strong></td>
<td>loads the page, displays static content</td>
<td>provides a platform for user interaction</td>
</tr>
<tr>
<td><strong>WEB SERVER</strong></td>
<td>simple request/response model</td>
<td>scripts, frameworks, databases</td>
</tr>
<tr>
<td>“web stack”</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>early web</td>
<td>modern web-app</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td><strong>CONTENT</strong></td>
<td>static pages, host-generated content</td>
<td>dynamic server-generated pages, user content</td>
</tr>
<tr>
<td><strong>BROWSER</strong></td>
<td>loads the page, displays static content</td>
<td>provides a platform for user interaction</td>
</tr>
<tr>
<td><strong>WEB SERVER</strong></td>
<td>simple request/response model</td>
<td><strong>scripts, frameworks, databases</strong></td>
</tr>
<tr>
<td>&quot;web stack&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questions?
The nitty-gritty details of the server

HTTP request: GET xkcd.com

HTTP response: web content (HTML, CSS, JavaScript)

How do the server and client communicate?
HTTP

• Hypertext transfer protocol
• Request and response
  • Headers and body
Request/response

- Open up Chrome and Chrome Inspector
- Right click, select “Inspect Element”
- Click on the “Network” tab
- Open your favorite website
Response Headers

- activity-id: "tlactivity-id: 25d1d799-32b3-4c08-91ec-298afca5b07c-->
- Cache-Control: private, max-age=0, proxy-revalidate, no-store, no-cache
- Connection: keep-alive
- Content-Encoding: gzip
- Content-Language: en-US
- Content-Length: 26209
- Content-Type: text/html; charset=UTF-8
- Date: Tue, 06 Jan 2015 06:13:39 GMT
- Expires: Thu, 12 Jun 2014 00:33:32 GMT
- K3s: "tlK3s:21-->
- P3P: policyref="/w3c/p3p.xml", CP="CAO DSP IND COR ADM CONo CUR CUSi D
to"
- Pragma: no-cache
- RTSS: 1
- Server: openresty/1.5.11.1
- Set-Cookie: SSLB=1; path=/; domain=.expedia.com
- Set-Cookie: SSID1=BwBpzB2oAAAAAAAsfatUgccMFRJ9q1QBAAAAAAAAAEn2rVAAK
  rVAAE2wMAAUhfAAASfatUAQDOAwAB5VoAABJ9q1QBA0kDAAFKYQAEn2rVAAE8QMAAZFiA
  NWIAABJ9q1QBA; path=/; domain=.expedia.com; expires=Wed, 06-Jan-2016
- Set-Cookie: SSSC1=1.610110763819511169.1|974.23269:987.24392:989.244
  .25130:1006.25141:1009.25233; path=/; domain=.expedia.com
- Set-Cookie: SSRT1=En2rVAIBAQ; path=/; domain=.expedia.com; expires=Wed
- Set-Cookie: ipsnf3=v.3|us|1|506|cambridge; Domain=.expedia.com; Path=/
- Set-Cookie: MC1=GUID=d291e5c3a5b403d8073358ef85d55ed; Domain=.expedia
  com
- Set-Cookie: JSESSION=7f063d98-b1b4-488f-a6e2-5d139b5d0de1; Domain=.expedi
  a.com
- Set-Cookie: tpid=v.1,1; Domain=.expedia.com; Expires=Sat, 17-Jan-2015
- Set-Cookie: iEAPID=0,; Domain=.expedia.com; Path=/
- Set-Cookie: linfo=v.4,|0|0|255|1|0|||1033|0|0||0|0|0|0|-1|-1; Domain=
  .expedia.com
- Set-Cookie: ipsnf3=v.3|us|0|506|cambridge; Domain=.expedia.com; Path=/
- Set-Cookie: SSPV1=EMAAAAAAAAAAAAAAAAAAAAAAEAAAAA; path=/; domain=
  .expedia.com
- Vary: Host,Accept-Encoding
- X-UA-Compatible: IE=Edge
Now we’ve gotten a single client-server round trip.

But what if we make many requests?

Does the server remember our past?
what?
clearly, it does, somehow.
HTTP

- HTTP is *stateless*.
HTTP

• HTTP is **stateless**.

• The protocol itself doesn’t remember anything.

• We have to do other things to have the server remember.
Remembering things

client  server  database
Remembering things

client
cookies

server
sessions

database

more about these things on Friday!
Whew, that was a lot. Let’s make your life easier.
Number of Internet users in the world
Number of websites (unique domains)
Building faster

- There is a huge demand to constantly build more features faster.
MOVE
FAST AND
BREAK
THINGS
Building faster

- There is a huge demand to constantly build more features faster.

- How?
Building faster

• There is a huge demand to constantly build more features faster.

• How?

  • Reduce the amount of code written.

  • Eliminate lots of boilerplate code.
IDEA: Many web apps have similar functions! e.g:

*can you name some?*
**IDEA:** Many web apps have similar functions! e.g:

- login/logout/account management
- create, read, update, delete data fields ("CRUD")
- relations between objects (e.g. friends, followers, ...)
- standard types of user interaction
At its core, almost every web application:

- stores and manipulates data
- handles user input and requests
- displays results and feedback to the user
6.005 students, does this sound familiar?
At its core, almost every web application:

**MODEL**: stores and manipulates data

**CONTROLLER**: handles user input and requests

**VIEW**: displays results and feedback to the user
Enter web frameworks

• Typically, model-view-controller structured frameworks
  • Introduce modularity
  • Reduce complexity
  • Increase development speed
• “DRY” -- don’t repeat yourself!
Web MVC

Web Server (www.facebook.com)

Web Framework
- router/controller
- view
- model

GET www.facebook.com/photos/123
Web MVC

Web Server (www.facebook.com)

Web Framework

GET www.facebook.com/photos/123

GET /photos/123

router/controller

view

model
Router

1. Create a mapping of URL --> router/controller functions

2. Execute the function
   1. Make calls to the model if necessary
   2. Return a view (HTML) to the user
Router, node.js style

router.get('/photos', ...
router.get('/photos/:id', ...
router.post('/photos', ...
router.post('/photos/:id', ...
/* more routes */
Router, node.js style

router.get('/photos', ...)  
router.get('/photos/:id', ...) // a match!  
router.post('/photos', ...)  
router.post('/photos/:id', ...)  
/* more routes */
What's up with get, post, etc.?

- HTTP *verbs*
  - GET, POST, PUT, DELETE
What’s up with get, post, etc.?

- HTTP *verbs*
  - GET, POST, PUT, DELETE

- General rules of thumb:
  - GET: no changes, simply getting data
  - POST: create a new object
  - PUT: edit an existing object
  - DELETE
What’s up with get, post, etc.?

- HTTP verbs
  - GET, POST, PUT, DELETE
- Router matches URL + HTTP verb
- If you want to know more about HTTP verbs: [http://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services](http://en.wikipedia.org/wiki/Representational_state_transfer#Applied_to_web_services)
Our photos controller

GET /photos/123:

get the Photo object with id 123 from the Photo model

return data to the corresponding Photo view
Views & templates

- HTML template with placeholder “holes”
- Router / controller passes in an object with contents for placeholders
Views & templates

<p>Hello! <%= name %></p>

<img src=<%= imgurl %>>

{
  "name":"Charles",
  "imgurl":"...
}
Web MVC

Web Server (www.facebook.com)

Web Framework

router/controller

model

view

HTML/CSS/JS

GET www.facebook.com/photos/123

GET /photos/123
Well that was a lot.

Don’t worry if it doesn’t all make sense right now.
We’ll see more concrete examples today and tomorrow.
Lunch!

A brief primer on Git and Dev Environments
Your first web framework: Node.js!

Install things!