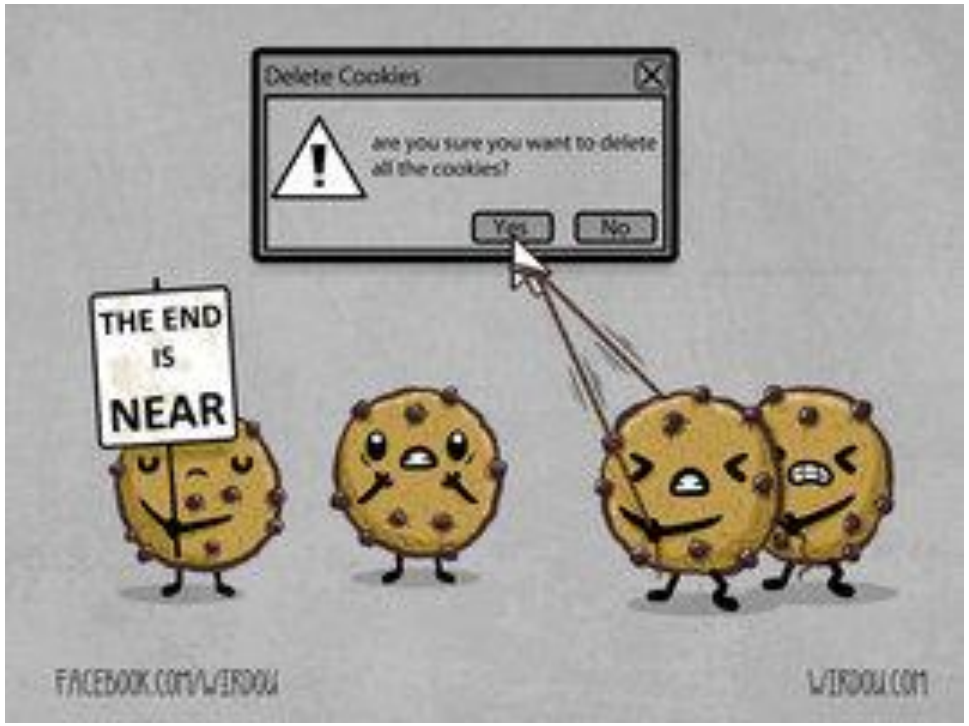
CSc 337



LECTURE 26: SESSIONS AND WRAPPING UP

What is a session?

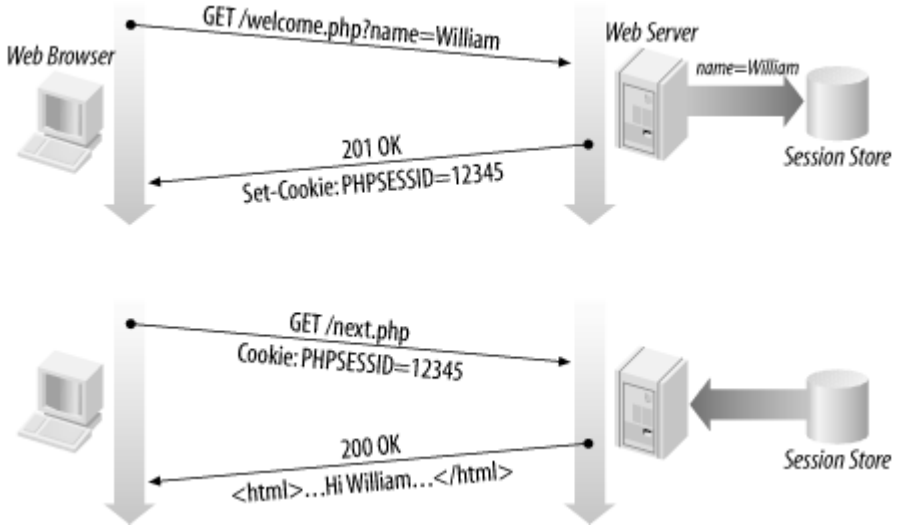
- **session**: an abstract concept to represent a series of HTTP requests and responses between a specific Web browser and server
  - HTTP doesn't support the notion of a session, but PHP does

- sessions vs. cookies:
  - a cookie is data stored on the client
  - a session's data is stored on the server (only 1 session per client)

- sessions are often built on top of cookies:
  - the only data the client stores is a cookie holding a unique **session ID**
  - on each page request, the client sends its session ID cookie, and the server uses this to find and retrieve the client's session data

How sessions are established

- client's browser makes an initial request to the server
- server notes client's IP address/browser, stores some local session data, and sends a **session ID** back to client (as a cookie)
- client sends that same session ID (cookie) back to server on future requests
- server uses session ID cookie to retrieve its data for the client's session later (like a ticket given at a coat-check room)



Cookies vs. sessions

- **duration:** sessions live on until the user logs out or closes the browser; cookies can live that long, or until a given fixed timeout (persistent)
- **data storage location:** sessions store data on the server (other than a session ID cookie); cookies store data on the user's browser
- **security:** sessions are hard for malicious users to tamper with or remove; cookies are easy
- **privacy:** sessions protect private information from being seen by other users of your computer; cookies do not

Using Sessions

You will need to install client-sessions in order to use sessions in NodeJS:

```
npm install client-sessions
```

In order to use cookie-parser in your code you will need to include the following lines:

```
var session = require('client-sessions');
app.use(session({
  cookieName: 'session',
  secret: 'random_string_goes_here',
  duration: 30 * 60 * 1000,
  activeDuration: 5 * 60 * 1000,
}));
```

Sessions in NodeJS

Check if a session exists with the following code:

```
if(req.session) { … }
```

Halt the current session, getting rid of all session data:

```
req.session.reset()
```

To access or set session data use `req.session`.variableName:

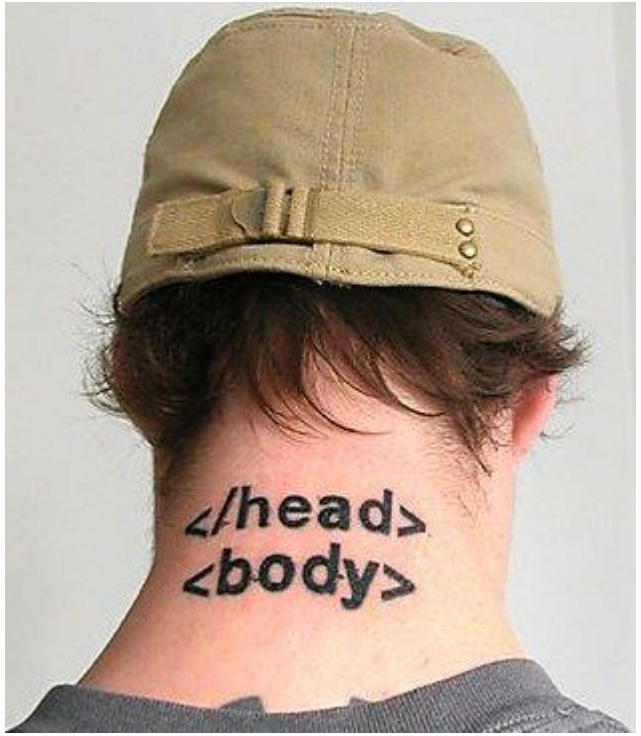
```
var name = req.session.name;
req.session.email = name + '@email.arizona.edu';
```

Session timeout

- because HTTP is stateless, it is hard for the server to know when a user has finished a session
- ideally, user explicitly logs out, but many users don't
- client deletes session cookies when browser closes
- server automatically cleans up old sessions after a period of time
    - old session data consumes resources and may present a security risk
    - you can explicitly delete a session by calling `session.reset()`

Congratulations!

- You now know more than the average web developer. (Really!)
- You know an introductory amount about many topics/languages
- You're capable of teaching yourself more...



The #1 question I hear about web programming: How do I stay up to date??

There are so many changing technologies…

- How do I know which ones to use?
- How do I learn about new libraries?
- Won't everything I learn be obsolete in 2 months?!?!

Q: How do I stay up to date??

A: This is the wrong question to ask.
Staying "up to date" is not that important.

Tech doesn't fundamentally change very often or very fast.

Most new web technology makes your life easier **but is not necessary.**

Everything* you want to do can already be done with the web technology available not just today, but 10 years ago.

*You know, within reason

Fundamentals don't change

Tech *doesn't* change that quickly

- Much of Facebook is still written in PHP
- Most of Google is written in Java and C++
- You will not (and should not) totally rewrite your codebase every year
- Tons of parallel problems, patterns, etc across tech

The real question to ask

Also: **Many new libraries are bad.**

- Literally anyone can post a library on npm - there is no
- Most libraries on npm are therefore garbage
- Even popular libraries can be poorly written.

So the real question to ask:

- **How do I distinguish good web technology from bad web technology?**

Choosing good tools

Either:

- You have enough knowledge to be able to decide whether a tool or technology is beneficial

Choosing good tools

Roll over image to zoom in

Wilson

Wilson Tour Slam Lite Tennis Racquet

19 customer reviews | 5 answered questions

List Price: $30.00

Price: $19.99 Prime

You Save: $10.01 (33%)

In Stock.

Want it Tuesday, June 6? Order within **11 hrs 34 mins** and choose On checkout. Details

Ships from and sold by Amazon.com. Gift-wrap available.

Color: **Blue/Black**

Size:

Grip Size: 4 3/8

- Volcanic frame technology for power and stability
- 110" head, 27.25" length
- 11.5 oz strung weight(326g)
- 4 3/8 inch grip size

Or:

- You don't have enough knowledge to tell the difference
- Therefore it doesn't really matter
- And you should choose the simplest / cheapest thing that other people say is good

Choosing good tools

If you keep getting better at tennis, someday you'll look back at your first racquet and think

- "OMG how was I using this terrible racquet" or,
- "Lol I had a $300 racquet and had no idea how to use it", or
- "Huh, that cheap one was actually pretty good"

But the ability to choose good tools takes expertise and experience that you don't have as a beginner.

And sometimes there's just a bit of personal preference

General advice

Don't be afraid or intimidated by new technology.

When you confront a new web thing, like a library or framework, one of two things will happen:

1. You will be excited by it, and you will want to use it.
2. You will not be excited by it, and you can safely ignore it.

Simpler is always better.

- ALWAYS delete code if you can
- ALWAYS remove a library if you can
- ALWAYS remove a framework if you can

What next?

CSC 337 is a fundamentals course, meaning we covered the critical stuff, but we just scratched the surface.

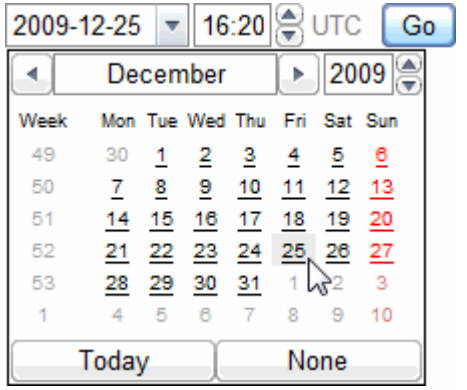
Helpful CS classes

Recommended CS classes:

- Databases
- As many systems classes as you can take
  - Networking
  - Operating Systems
- Compilers
- Programming languages

Learn more about HTML5

- semantic **tags**: `nav`, `aside`, `header`, `footer`, `section`, `aside`, `article`
- **Forms** 2.0: sliders, search bars, color/number/email/url/date/time, placeholders, ...



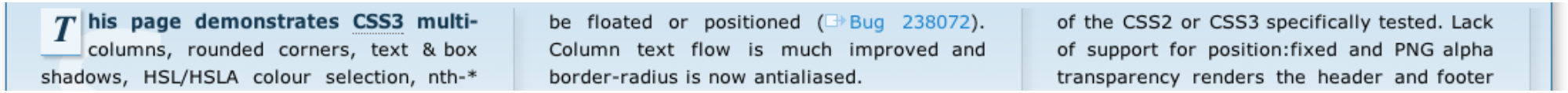
- **`audio`** and **`video`** tags for embedding multimedia

```
<video src="video.ogv" width="425" height="350"></video>
```

- **`canvas`** tag for drawing 2D shapes in HTML/JS (like `DrawingPanel`) (ref 1, 2, 3)

Learn more about CSS3

- Flex and grid layouts
- new **selectors**: `nth-child`, `inline-block`, `:not`, `+`
- ability to **embed fonts** in a page *(yay)*
- easy built-in support for **multi-column layouts**



- transparency/**opacity**, color **gradients**, **shadows**

- **animations** and transitions
- affine **transformations** (scaling, rotation, perspective)

Learn more about JavaScript

- Support older browsers with BableJS
  - Write code with the latest features in JavaScript
  - Support older browsers without having to rewrite anything

Frameworks:

- AngularJS
- Backbone.js
- Bootstrap
- Ember.js
- ReactJS
- Vue.js
- Flask (backend)
- Ruby on Rails (backend)
- Django (backend)

How do I use a Framework? Just try it out

General advice:

- Go to the official website
- Use the official website's tutorials
  - Like, actually follow along; don't just skim the docs
- Then **build a small app of your own** on the framework
  - The only way to "learn" a framework is to build something using it, beyond just following a tutorial
  - Suggestion: Choose something you could build in 24 hours using the tech you already know

Most well-known frameworks have tutorials, excellent documentation, strong developer communities, etc.

Publishing to the web

Protecting web resources

- **Please don't post your HW solutions on the web unprotected!**
  (we want to be able to assign some of these programs again)
- posting resources with a shared password:
  - create files named `.htaccess` and `.htpasswd` with proper contents and put them in your HW root folder on UA
  - doesn't require a UA NetID
  - can give password to friends / family / employers

Example htaccess, htpasswd files

`.htaccess`:

```
AuthUserFile full/path/to/.htpasswd
AuthName "Restricted Access"
AuthType Basic
require user username
```

`.htpasswd`:

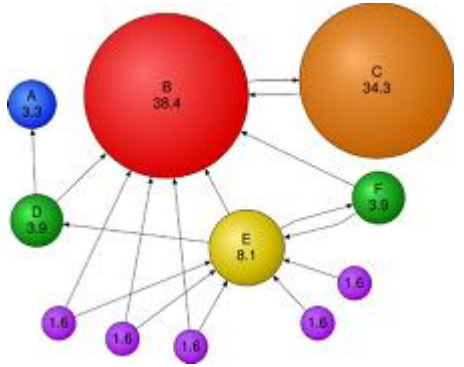
```
username:encryptedPassword
```

- place these files in the top folder of the content you want to protect
- htpasswd generator (gives you the text to copy-paste into your `.htpasswd` file)
- or from *nix terminal: `htpasswd -c .htpasswd` *username*

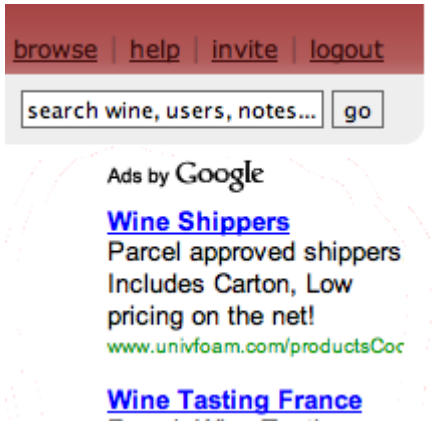
Taking a web site "live"

- buy your own domain name (e.g. allisonobourn.com): DreamHost
- learn about (stalk) your users with Google Analytics

Top search queries	Average top position
1. cellarspot	1
2. henri boillot	11
3. bossi siena	1
4. les charmes chardonnay	2



- improve your page / improve PageRank: SEO, meta tags
- make a few bucks with ads: Google AdSense
- get the word out: Google AdWords, Webmaster Tools



Publishing web pages

Domain name registration:

- Reserves a custom URL: myawesomesite.com
- But doesn't usually include web hosting; all you own is the name.

Web hosting:

- Provides a location on the internet to upload files
- Usually with some crummy URL, like http://bucket.s3-website-us-west-2.amazonaws.com/

Domain name registration and web hosting are sometimes provided by the same company, but not always.

Publishing static web pages

You can register your own domain name through many companies:

- Google Domains: Only domain name registration
- Amazon S3: Only web hosting
- Dreamhost: Domain name **and** web hosting options

Domain name registration is usually ~$12/year
Web hosting is usually ~$10/month

- Amazon S3 is **significantly** cheaper (virtually free for low-traffic websites) but more complicated to set up

Publishing server-side code

If you want to host both a frontend and a backend, you may want a web host that allows you to configure a server.

There are an immense number of options, with different levels of configuration. Here are some:

- Heroku: Super easy to use, but offers less control. Also a lot more expensive.
- AWS: Cheap, lots of options, but more complicated
- Google Cloud: Basically the Target brand of AWS: Cheaper than AWS; as complex as AWS; fewer products than AWS