COMPSCI 326 - Web Programming
Networking & HTTP

join on the Slack #q-and-a channel as well as Zoom
Remember, you can ask questions of your teammates on your group Slack!
please **turn on your webcam** if you can
mute at all times *when you aren't asking a question*
(https://docs.google.com/document/d/1QryZVXBdxqnw0QMyfaZb8mtWlLteSXcVvQcnoL7u0zo/edit?usp=sharing)

Today: Networking & HTTP

resources:

- https://developer.mozilla.org/en-US/docs/Web/HTTP
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Other stuff:

- Last time: in-class exercise

Networking

- IP addresses: example = 128.119.240.84 (IPv4 = www.cics.umass.edu)
- google.com → DNS "directory name service" → 172.217.10.110, ...
- Very basic networking (focusing on TCP/IP but also UDP: "socket")
  - messages are sent as "packets" - chunks of ~1K
  - packets have a destination address and a return address, a destination and return **port number,** plus a sequence number, and then their actual data
  - routed to and from the recipient
  - re-assembled in order by the recipient
  - packet receipt is always ACKnowledged via an ACK message
  - packet delivery is "reliable"
    - not so for UDP, used for video streaming
    - faster, OK if datagrams are "dropped"

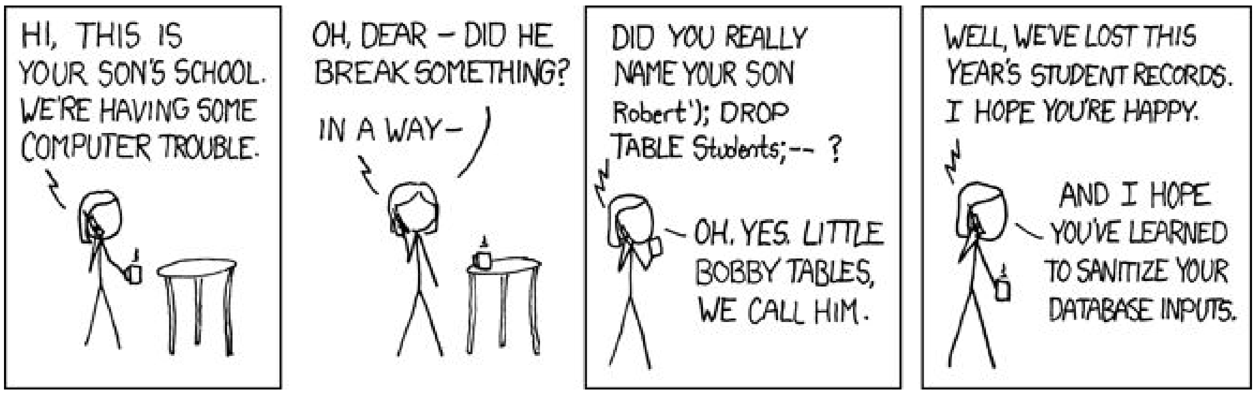
HTTP/HTTPS

- HTTP and HTTPS are "protocols" built on top of TCP/IP
  - basic HTTP details how a client can talk to a server
  - in short, client sends URLs (like you type into the browser bar) with possibly other info to the server
  - server returns HTML or other data, and possibly an error (e.g., 404)

`404 Not Found`

The server can not find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 to hide the existence of a resource from an unauthorized client. **This response code is probably the most famous one due to its frequent occurrence on the web.**

- important: HTTP itself is **stateless** - it always behaves the same way given the same input!
    - if you want to save state, you need to store data either on the client side (localStorage, cookies) or the server side (on a database connected to the web server), or both
- basic HTTP request: **pages**
    - usually connected to directories, returns HTML page
- not really enough - needed support for user requests
- **GET** - encodes info as an addition to the URL
    - http://somefckingsite.com?name=Fred&age=99
    - http://localhost:**8080**?first=Test&last=Me
      http://somefckingdb.com?query="SELECT%20%20FROM%20empDB where emp.salary > 10000 and userid=1044 OR TRUE
        - SQL injection attack


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

    - Twitter links, FB links   fbclid=....
- This is a bit brittle and also limited -- need to "escape" characters like spaces (into %20)
    - Makes it easy to do URL hacking
    - Limited length
- **POST** - sent along as a separate message, handled by HTTP

Web Servers

- nginx, Apache, httpd

- Implement HTTP/s protocol and more
- **routing** so you can map directories to arbitrary places
- access control so you can limit who can see what pages
- **authentication** to handle logins

Basic Web Server

- built-in module in node.js - **http**
- Uses template strings: [Template literals (Template strings) - JavaScript | MDN]

```
const http = require('http');
const url = require('url');

function basicServer(req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'}); // we're sending HTML
    res.write('Hello World!<BR>'); //write a response to the client
    const requestedURL = req.url;
    res.write(`URL is <B>${requestedURL+"foo"}</B><BR>`); // template string
(backquote)
    const parsedURL = url.parse(requestedURL, true);
    const query = parsedURL.query;
    const pathname = parsedURL.pathname;
    res.write(`path is <B>${JSON.stringify(pathname)}</B><BR>`);
    res.write(`query components are <B>${JSON.stringify(query)}</B><BR>`);
    if (Number.parseInt(query.age) >= 21) {
      res.write('DRINK UP!<BR>');
    } else {
      res.write('STAY SOBER KIDS!<BR>');
    }
    res.end(); //end the response
}

// Start the server
http.createServer(basicServer).listen(8080); //the server object listens on port 8080
```