Engineering Robust Server Software
Web Protocols and Technologies
Today: Web Protocols/Technologies

• "The Web" is full of
  • Many browsers (Chrome, Firefox, …)
  • Different server "stacks"

• Yet they all work together..
  • Everything speaks the same language

• Let's delve into that
The Life of a Web Request

- I enter a URL in my browser...
The Life of a Web Request

- Browser sends an HTTP "GET" request to the server
  - Which is running a web server daemon, listening on port 80
HTTP Request Basics

- HTTP Requests have a "verb" and a URI (and then a version number)
  
<table>
<thead>
<tr>
<th>Method</th>
<th>URI</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>POST</td>
<td>/home/drew</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>PUT</td>
<td>/foo/bar/xyz</td>
<td>HTTP/1.1</td>
</tr>
<tr>
<td>DELETE</td>
<td>/blah/blah/blah</td>
<td>HTTP/1.1</td>
</tr>
</tbody>
</table>

- Read about HTTP "verbs" (aka methods):
  - [https://tools.ietf.org/html/rfc7231#section-4.3](https://tools.ietf.org/html/rfc7231#section-4.3)
  - Most common for web browsers: GET + POST
    - Others useful for web-based APIs

The Life of a Web Request

HTTP/1.1 200 OK
Date: Tue, 17 Jan 2017 02:08:36 GMT
Server: Apache/2.2.15 (Scientific Linux)
Etag: "1484618676-0"
Content-Language: en
Cache-Control: public, max-age=3600
Last-Modified: Tue, 17 Jan 2017 02:04:36 GMT
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Content-Type: text/html; charset=utf-8

- Server responds (in this case: 200 OK)
- With headers and data
  - The data (in this case) is HTML—could be anything (JSON, XML, image,...)
HTTP Responses

- Responses come with response code
  - 1xx = informational
  - 2xx = successful
  - 3xx = redirection
  - 4xx = error
  - ...
  - [link](https://tools.ietf.org/html/rfc7231#section-6)
- Headers, give meta-data about response
  - E.g, content length, encoding,…
- Also, (if appropriate), the data
HTTP: Stateless Protocol

- Http is Stateless:
  - Each request is separate from all the others
  - Want session information? Include in request/responses

https://xkcd.com/869/
Hypertext Markup Language:

- Not a programming language (does not execute things)
- Marks up content (describes how to format it)
Fancier Page?

• Most common fancier things:
  • `<a href="http://foo.bar.com/xyz/blah.html">link text</a>`
  • `<div> ... </div>`
  • `<p> ... </p>`
  • `<h1>...</h1>  <h2>...</h2> etc`
  • `<ul> <li> thing1 </li> <li> thing 2 </li> ... </ul>`
  • `<ol> <li> thing1 </li> <li> thing 2 </li> ... </ol>`
  • `<img src="cats.png">`
Elements can have Attributes

- `<a href="http://foo.bar.com/xyz/blah.html">link text</a>`
- `<img src="cats.png">`
- A few interesting ones:
  - `class`: for use with CSS
  - `name`: for use with forms
  - `id`: for use with JavaScript (also CSS)

HTML Forms

- Often we want to submit data to the server
  - E.g., when the user presses a "submit" button
- Use HTML "forms"
  - Use <form> tag to enclose the inputs for the form
    - Has attributes of where to send data, whether to GET or POST
  - Put input elements (and others) inside:
    - <textarea>, <select>, <button>, <input>, …
  - Give each input a name attribute
    - Will be how you identify which data is which on the server
Cascading Style Sheets

Even if we put more stuff on our page, it doesn't look nice
With CSS...

- CSS lets us change how the browser **styles** the HTML
  - Positioning, colors, shapes, font sizes,…
CSS Basics

- Can re-style any occurrence of a tag (e.g., body, h1…)

```css
body {
  background: #001A57;
}

h1 {
  text-align: center;
  color: #001A57;
}
```
CSS Basics

```
div.container {
    border: 1px solid gray;
    background: #E5E5E5;
    margin: auto;
    min-width: 350px;
    max-width: 600px;
}

div.box {
    border: 1px solid gray;
    margin: auto;
    padding: 15px 2px;
}
```

- Can re-style a tag by class

```
<div class="container">
    <h1><strong>Please Login</strong></h1>
    <div class="box">
    ```
CSS Basics

```
.label {
    font-size: 20px;
    color: #001A57;
}
```

- Can re-style by class (can use with any tag)
CSS Basics: Include External Stylesheet

- Generally want to load CSS from another file (on server)
  - Lets you easily use same style for many pages (same look + feel)
  - Lets you easily change style of all pages at once

```html
<html>
  <head>
    <title>Another Page</title>
    <link type="text/css" rel="stylesheet" href="style.css"/>
  </head>
</html>
```
CSS: Can Do Fancier Things

• Reformat button when hovered over
  • With :hover
Fancier CSS

- Our button from this page
- Several properties to make
  - Nice curved corners
  - Large, centered text
  - Centered in parent area
- `.btn:hover`
  - Changes colors on hover

```css
.btn {
  border-radius: 6px;
  background-color: #001A57;
  border: 1pt solid #666666;
  color: white;
  padding: 8px 20px;
  text-align: center;
  text-decoration: none;
  font-size: 16px;
  margin: 0 auto;
  display: block;
}

.btn:hover {
  background-color: #607AB7;
  border: 1pt solid #001A57;
  color: #001A57;
}
```
More Fancy CSS?

- Much more you can do with CSS
  - We aren't going to be too picky about fancy looking sites
    - (not a UI/UX class)
  - More interested in server side
  - …but you should be able to make it look nicer than black + white
Ok, but... It Still Doesn't Do Anything..

- HTML + CSS: can make a nice looking page
- Won't "do" anything.
  - Could send data to server with **forms**, load a whole new page
  - This is how everything worked in the mid 1990s…
- Modern webpages are interactive, do things with no reload
  - Use JavaScript (actual programming language)
JavaScript Example: A Page With Some JS

- Here is the body of a page. Has:
  - A table (with only a header row)
  - A button (whose onClick is some JavaScript—calls a function not shown here)
• The JavaScript that goes with it:

```html
<title>A Page of Counters</title>
<script>
  var counter=0;
  function addCounter() {
    var elt = document.getElementById("counters");
    elt.innerHTML = elt.innerHTML + "<tr><td> " + counter + " </td> <td>  " +
    new Date().toLocaleString() + "</td></tr>";
    counter++;
  }
</script>
</head>

• Press button->add counter / current date to table
To understand what happened, you need to know about the **DOM**

- **Document Object Model**: API for HTML + XML documents
  - Language agnostic (same API in JavaScript, C, Java, Python,...)
- Think of HTML as describing a **tree** of objects
Document Object Model

- DOM specifies ways to manipulate the tree
  - Find elements meeting some criteria
  - Get children of a particular element
  - Modify an element
  - Create an element

```html
<html>
  <head>
    <title>A Page of Counters</title>
    <script>
      var counter = 0; ...
    </script>
  </head>
  <body>
    <table id="counters">
      <tr>
        <!-- A Page of Counters...
```
<head>
<title>A Page of Counters</title>
<script>
    var counter=0;
    function addCounter() {
        var elt = document.getElementById("counters");
        elt.innerHTML = elt.innerHTML + "<tr><td> " + counter + " </td><td> " + new Date().toLocaleString() + "</td></tr>";
        counter++;
    }
</script>
</head>
```html
<body>
  <table id="counters">
    <tr>
      <th>Count</th>
      <th>Time</th>
    </tr>
  </table>
  <button onClick="addCounter()">Add Counter</button>
</body>
```
<head>
  <title>A Page of Counters</title>
  <script>
    var counter=0;
    function addCounter() {
      var elt = document.getElementById("counters");
      elt.innerHTML = elt.innerHTML + "<tr><td> " + counter + " </td> <td> " + new Date().toLocaleString() + "</td></tr>";
      counter++;
    }
  </script>
</head>
Accomplish Same Task w/o Reparsing

```javascript
    var counter=0;
    function addCounter() {
        var elt = document.getElementById("counters");
        var tr = document.createElement("tr");
        var td1 = document.createElement("td");
        var td2 = document.createElement("td");
        td1.textContent = counter;
        td2.textContent = new Date().toLocaleString();
        tr.appendChild(td1);
        tr.appendChild(td2);
        elt.appendChild(tr);
        counter++;
    }
```
More JavaScript

- As a programming language:
  - First class functions
  - Dynamically typed
  - Has Objects
  - C-/Java- like syntax (mostly)

- See:
In JavaScript, you write down objects like this:

- `var pt = { x : 3, y: 4, moveLeft: function() { this.x = 0; } };`
- i.e., A comma separated sequence of **field: value**
- Note that methods are just fields whose values are functions!

JavaScript Object Notation (JSON) is a common data format:

- Can't put function values in
- Only string, number, true, false, arrays, objects, null
- Arrays are written with [], objects with {}  
- Field names are quoted: `{ "x" : 3, "y" : 4, "colors": [ "orange", "pink"] }`
More JavaScript: Later

- JavaScript can also contact the server
  - Get a response (later), and then do something with it
  - Server can send responses that are not HTML
    - Could send JSON, or XML -> easy to parse
    - JS on client can take data, show in appropriate way
- AJAX: Asynchronous JavaScript And XML
  - We'll talk about this later when we start into server side web code
XML

• Similar looking to HTML (tags, attributes, nesting)
  • No predefined tags: make any tags with any meaning you want
  • Stricter /more uniform rules (all tags must be closed)
XML

- Why XML?
  - Extensible
  - Human readable
  - Ubiquitous: parsers for it in most languages
    - DOM: similar to HTML (but different)
- C++: xerces
  - You'll use later
- Other XML tools
  - E.g., XSLT (not going to use/cover, but you might find useful sometime)
Web APIs

- Many websites provide APIs
  - Programmatic ways to interact with websites (possibly used by your own JS)
  - Usually by HTTP: GET, POST, PUT, DELETE, …

- Request format?
  - Conform to HTTP format
  - Use URI to specify what to query/update/etc
    - GET /api/courses/DEPT/NUM
  - Include information in POST data
    - Format? JSON, XML, …

- Response? Easy to parse (JSON, XML, …)
REST APIs

• Representational State Transfer (REST, or RESTful API)
  • Commonly used design principles
  • Ask many people: "Uses HTTP protocol"
  • 
REST APIs

- Representational State Transfer (REST, or RESTful API)
  - Client-server
  - Stateless (request contains all info needed, does not rely on previous)
  - Cacheable (can control cache-ability)
  - Uniform Interface (identify resources, manipulate representations)
    - E.g., Request by URI, manipulate textual representation of data
  - Layerable Systems (transparent to client—can't really tell)
  - [Maybe] Code on Demand
    - E.g., JavaScript
APIs

• Why do people think "HTTP API"
  • Because HTTP follows the REST rules

• Why do people like HTTP-based APIs?
  • Much work done for you
    • Web-servers parse incoming requests
    • Web-browsers parse incoming responses
    • Both "speak" the same language for errors
    • Can transfer any data/ any meta-data over HTTP
    • Can name resources in useful [and easy to read] ways
Wrap Up

• Today:
  • HTML, CSS, JavaScript, JSON, XML
  • Super-quick intro: not main content of class
  • But useful technologies to know/use: web big example in servers
  • References to more learning

• Next time:
  • Dive into servers: UNIX Daemons

• Homework 1:
  • Start? Http request/response parsing…
  • Coming soon: Unix Daemons (this class), networking (650)