Engineering Robust Server Software
Web Protocols and Technologies
Web Protocols

- REST Principles
- HTTP
- Data: XML, HTML, JSON
- Manipulation: JavaScript
Recall: Server Big Picture

- Let's remember our view of the world
How should such a protocol be designed?
REST Principles

⦁ **Representational State Transfer:**
  - Let us derive the principles...
  - Principle 1 (easy): Client/Server architecture
Suppose two requests go to two different servers.

Why?

What does this say about protocol design?
REST Principle 2: Stateless

- Why: **balance load**
- Protocol principle: **stateless**
  - Server side code does not remember anything about previous request
  - Each request needs all information to proceed
    - But wait... servers have to have some state, right?

https://xkcd.com/869/
REST Principles

- State: only in storage tier
  - User booked a flight: goes into storage tier (not application tier)
Stateless: Implications

- Need to identify user: include in request
  - But...don't we distrust everything from client?
  - Yes! Distrust client:
    - Give session ID at login
    - Client must provide session ID with each request
    - Session ID should be hard to forge
    - How do you validate session ID?
      - Have it be a big random string that we compare to our database (session cookies) or
      - Have it be cryptographically signed (Javascript Web Tokens (JWTs))
Suppose many people want the same resource?
- Asking for it frequently
- What implication does this have?
Or Maybe…

- Many people at Duke decide to watch same video
  - What implications does this have?
  - How can we address this issue?
Or Maybe...

- Would like to cache responses
  - Reduce bandwidth + latency
  - Reduce load on servers
  - But, what difficulties?

Internet

Cache
Principle 3: Cacheability

- Responses should be cacheable
- ...Except when this creates problems
  - Explicit cache control
    - Label responses as non-cacheable
    - Label responses as expiring at a certain time
    - Provide a way to validate that response is still current
REST Principles

- Speaking of caches and load balancing..
REST Principles

- Speaking of caches and load balancing...
  - We decide to add a cache and a hardware load balancer...
 Speaking of caches and load balancing..
  We decide to add a cache and a hardware load balancer...
  And maybe some other things (e.g., IDS - Intrusion Detection System)
What should client do differently in response to changes?
Principle 4: Transparencially Layered System

- Principle 4: Transparencially Layered System
  - Client should do **nothing** differently
REST Principles

- Storage Tier: has data we want to manipulate
REST Principles

- Storage Tier: has data we want to manipulate
  - E.g., table of seats on flights + who booked them (or nobody)

<table>
<thead>
<tr>
<th>FlightNum</th>
<th>SeatNum</th>
<th>BookedBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1A</td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>1B</td>
<td>FRX345</td>
</tr>
<tr>
<td>1234</td>
<td>1C</td>
<td>JMN895</td>
</tr>
<tr>
<td>1234</td>
<td>1D</td>
<td></td>
</tr>
</tbody>
</table>

Should client know about this?
REST Principles

Should client know about this?

- No (for many reasons)
- ...but needs to be able to manipulate that resource

<table>
<thead>
<tr>
<th>FlightNum</th>
<th>SeatNum</th>
<th>BookedBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1A</td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>1B</td>
<td>FRX345</td>
</tr>
<tr>
<td>1234</td>
<td>1C</td>
<td>JMN895</td>
</tr>
<tr>
<td>1234</td>
<td>1D</td>
<td></td>
</tr>
</tbody>
</table>
Sub-principle 5.1: Manipulate Representations

- Manipulate representations of resources
  - Client gets a representation of the resource (JSON, XML, ...)
  - Works with that representation
  - And can make any appropriate changes based on what it has
    - E.g., book a seat (send back JSON, XML, etc... ) request
- How does client even know flight numbers?
- How does it refer to particular flight?

<table>
<thead>
<tr>
<th>FlightNum</th>
<th>SeatNum</th>
<th>BookedBy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1A</td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>1B</td>
<td>FRX345</td>
</tr>
<tr>
<td>1234</td>
<td>1C</td>
<td>JMN895</td>
</tr>
<tr>
<td>1234</td>
<td>1D</td>
<td></td>
</tr>
</tbody>
</table>
Principle 5: Uniform Interface

- Manipulate representations of resources
  - HTML, JSON, XML, ...
- Uniform resource identification in request
  - HTTP: /flights /flights/1234/seats
- Self-descriptive messages
  - Messages have metadata
    - e.g. HTML: MIME type (text/html, image/jpeg, application/zip, etc.)
- "Hypermedia As The Engine Of Application State"
  - Can "find" other (appropriate) resources from root
  - In HTML: hyperlinks
Principle 6 (Optional): Code on Demand

- Server can send code to client
- E.g., Can send JavaScript to client to run client-side code
HTTP and REST

- HTTP protocol obeys REST principles
  - But could make other protocols that are RESTful too
  - Speaking of HTTP...
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)
  
  GET / HTTP/1.1
  POST /home/drew HTTP/1.1
  PUT /foo/bar/xyz HTTP/1.1
  DELETE /blah/blah/blah HTTP/1.1

- 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

RFC 7231 will be your best friend on Homework 2
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
  - ...

- Headers, give meta-data about response
  - E.g, content length, encoding,...

- Also, (if appropriate), the data
So What Do We Transfer?

- Could transfer pretty much anything over HTTP
  - HTML
  - CSS
  - XML
  - JSON
  - Text
  - Images
  - Videos
  - ....
So What Do We Transfer?

- Could transfer pretty much anything over HTTP
  - HTML - describes content
  - CSS
  - XML
  - JSON
  - Text
  - Images
  - Videos
  - ....
So What Do We Transfer?

- Could transfer pretty much anything over HTTP
  - HTML - describes content
  - CSS - describes styling
  - XML
  - JSON
  - Text
  - Images
  - Videos
  - ....
So What Do We Transfer?

- Could transfer pretty much anything over HTTP
  - HTML - describes content
  - CSS - describes styling
  - XML - good for APIs
  - JSON - good for APIs
  - Text
  - Images
  - Videos
  - ....
Web Technologies

- **HTML**
  - Content
    - Tree structured data
      - Server code will generate from data (Probably use templates)

- **CSS**
  - Style
    - How to draw elements
    - Can use library (e.g. bootstrap)

- **JavaScript**
  - Code: Manipulate HTML
    - Alter tree (DOM)

- **Note**: we are NOT focusing on front-end stuff
  - This is not a UI/UX class
  - Strongly encouraged to make things look nice (show off your work)
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)

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...)
● Can re-style a tag by class

```css
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;
}
```

```html
<div class="container">
    <h1> Please Login </h1>
    <div class="box">
```

Duke University
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
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)
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)
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
<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>
JavaScript Example: Revisited

```html
<body>
<table id="counters">
<tr>
<th>Count</th>
<th>Time</th>
</tr>
</table>

<button onClick="addCounter()">Add Counter</button>
</body>
```
JavaScript Example Revisited

```html
<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>
```
<script>
    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++;
    }
</script>
More JavaScript

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

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

```javascript
var pt = { x: 3, y: 4, moveLeft: function() { this.x -- ; } };
```

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"] }
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)
...but How to Interact With Server?

- 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
AJAX Basics

```javascript
function someJSFun() {
    //whatever code...
    var xhttp = new XMLHttpRequest();
}
```

This is the object to contact the server and get a response...

Note: XML was invented before JSON and other stuff, so it’s called an XMLHttpRequest even if we’re not doing XML.
function someJSFun() {
    //whatever code...
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        //some other code in here...
    };
}

Set its onreadystatechange to be notified when stuff happens
function someJSFun() {
  //whatever code...
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    //some other code in here...
  };
}

Yes, you can write one function inside another.
JavaScript has **lexical scope**.
This makes a **closure**.
function someJSFun() {
  //whatever code...
  var xyz = something;
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    ...xyz...
  };
}
function someJSFun() {
    //whatever code...
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        //some other code in here...
    }
    xhttp.open("GET", "/api/foo/bar/42", true);
}

.open() specifies where to connect:
HTTP Request Method
URL to request
Asynchronous (usually true)
function someJSFun() {
    //whatever code...
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        //some other code in here...
    };
    xhttp.open("GET", "/api/foo/bar/42", true);
    xhttp.send();
}

.send() makes the actual request.

Will make callback to our function when state changes
AJAX Basics

```javascript
xhr.onreadystatechange = function() {
}
```

Now let us look inside our ready state change callback
AJAX Basics

```javascript
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
    }
};

Typically inspect this.readyState first

this is our XMLHttpRequest

readyState: 0—4. 4 is Done
AJAX Basics

```javascript
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && xhr.status == 200) {

    }
};
```

May also want to inspect

`xhr.status` (HTML response status)

200 = OK
Once we have our response, generally want to use:

```javascript
this.responseText
```

which has the text we received.
AJAX Basics

```javascript
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4 && xhr.status == 200) {
    var resp = JSON.parse(xhr.responseText);
  }
};
```

If our response is JSON, can use JSON.parse to turn into JavaScript object!
Wrap Up

- Today:
  - REST: protocol principles
  - Super quick intro to HTML/CSS/JavaScript/JSON/XML
    - Not main focus of this class, but you will need
  - AJAX: ties to previous ideas!