CSCI 1320
Creating Modern Web Applications

Lecture 12: Web Application Architectures
Web Applications

- Are distributed systems
  - Some work is done in the front end
  - Some work is done in the back end
  - Some work is done in servers or databases
- Different web applications allocate the work differently
  - Server-side heavy: banner, blogs, ...
  - Client-side heavy: gmail, google docs
- What should be done where
  - Responsiveness; Performance
  - Access to and security of code and data
  - Amount of communications needed
  - Where the data is actually needed; what is done with the data
Server-Side Application
Server-Side Application

- The default browser-server model
  - Browser sends a HTTP request
  - The HTTP response is used to replace the current page
- Various technologies can support this model
  - Using PHP, JSP, Servlets to generate new HTML page
    - Based on properties passed in the URL query
  - Using Node.JS with a templating engine
  - Front-end JavaScript only used for interactive features (i.e. pull downs, validation)
Server-Side Pros/Cons

- Templating lets you write HTML directly for the most part
  - Easier to change templates than actual code
- Don’t have to send lots of code over the web
  - The code can be kept private
- Server code is generally synchronous, straight-forward
- Data isn’t directly accessible to users
- Not as interactive, responsive
- Requires more compute power on server
  - Less on the clients
- Works naturally with assistive devices
Client Side Application

Select an item to buy

Create running subtotal

Get credit card and shipping info

Error check data

Email order to administrator

Web Server

Hardware/Software

Running CGI Script.

Client Using Web Browser

AngularJS by Google

React JS
Client-Side Application

- Most of the work is done on the page
  - Using JavaScript
  - As with Angular, React, ...

- Front end still needs to get/send information
  - To the server, database, back end, application
  - To actually get work done
  - To get additional information
  - To ensure information is permanent
  - To save status in case of refresh, return to page

- Page update done in JavaScript
  - Based on information retrieved
  - JavaScript handles formatting, updating, etc. the page
Client-Side Pros and Cons

- JavaScript isn’t the nicest language
  - Especially if you have to write lots of code
  - Front end JavaScript isn’t modular
- JavaScript isn’t the most efficient language
  - Today’s browser provide efficient implementations
- Responses are asynchronous
- Might need to send large amount of data
  - To cover all possible interactions
  - But data can be sent on demand
- Your base code is public; base data is available
- Normal navigation can be difficult
- Interface can be highly interactive, responsive
- Working with assistive devices can be tricky
Find Your CDs

CD Search:

Find Your CDs

Title

CD#1 title and major artist
* Track 1
* Track 2

Artist

CD#2 title and major artist
* Track 1
* Track 2

Track

CD#3 title and major artist
* Track 1
* Track 2

CD#4 title and major artist
* Track 1

Find Your CDs

CD TITLE

ARTIST

Description

TRACK Title

Artist

Length

Description

TRACK Title

Artist

Length

Description
CQ Query Tasks

- Primary Tasks
  - Initial Search For CDs
  - Look at the details of a specific CD
  - Refine initial search by title, artist, track, genre; sort results

- Should these be done client-side or server-side?
  - A: All server side
  - B: Initial search server side, rest client side
  - C: Refinement & detail client side, rest server side
  - D: Detail page client side, rest server side
  - E: All client side
AJAX

- **Asynchronous JavaScript And XML**
  - JavaScript is used to send an XML request to the server
    - Using a particular URL
    - Expecting XML output as a response
  - When the response comes back, JavaScript runs again
    - Interprets that output
    - Changes the DOM to update the page
- **JSON is often used today rather than XML**
- **JavaScript libraries provide support for this**
  - Setting up request; handling response
  - XML, JSON encoding and decoding
XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ARTICLES SYSTEM "D:\Projects\Clients\XML\Contents\Temp\articledtd.dtd">
<!xml-stylesheet type="text/xsl" href="D:\xsltohtml.xsl" ?>
<ARTICLES>
  <ARTICLE>
    <ARTICLEDATA>
      <TITLE>XML Demystified</TITLE>
      <AUTHOR>Jaidev</AUTHOR>
    </ARTICLEDATA>
  </ARTICLE>
  <ARTICLE>
    <ARTICLEDATA>
      <TITLE>XSLT Demystified</TITLE>
      <AUTHOR>X S Cel Tea</AUTHOR>
    </ARTICLEDATA>
  </ARTICLE>
  <ARTICLE>
    <ARTICLEDATA>
      <TITLE>C# Demystified</TITLE>
      <AUTHOR>Aleksey N</AUTHOR>
    </ARTICLEDATA>
  </ARTICLE>
</ARTICLES>

<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
    <qanda>
      !-- Note: We need to add more questions later.-->
  </qanda>
</quiz>
```
{  "Rail Booking": {  "reservation": {  "ref_no": "1234567",  "time_stamp": "2016-06-24T14:26:59.125",  "confirmed": true  },  "train": {  "date": "07/04/2016",  "time": "09:30",  "from": "New York",  "to": "Chicago",  "seat": "57B"  },  "passenger": {  "name": "John Smith"  },  "price": 1234.25,  "comments": ["Lunch & dinner incl.", "Have a nice day!"]  }
}
XMLHttpRequest (using jQuery)

- Syntax
  ```javascript
  var req = $.ajax({
    method: "POST",
    url: "/url/...",
    data: { data to send },
    success: function(data,sts) { ... },
    error: function(msg,sts,err) { ... }
  });
  ```
- Request gets sent when JavaScript returns
- Other parameters and events are available
RESTful APIs
RESTful APIs

- Use HTTP methods explicitly
  - POST, GET, PUT, DELETE, ...
- Are stateless
  - Each request includes all the necessary information
- Expose directory structure-like URLs
  - Use the URL to encode the operation and the data
- Transfer XML, JSON or both
URL Encodings

• Suppose we create a chat application
  • POST /chats with { text: "...", user: "...", title: "..." } => id 01
  • GET /chats/01
  • PUT /chats/01 with { text: "...", user: "..." }
  • DELETE /chats/01

• Can also encode commands
  • GET /command/subcommand/...
  • POST /chats/01/delete

• Can have nested ids
  • GET /command/id/what/id/...
Web Sockets

- AJAX model is client-initiated (pull model)
- Some applications are server-initiated
  - Only want notification when things change
- Web sockets allow this approach
  - Establish a 2-way connection between client and server
  - Send messages from client to server or server to client
  - Messages result in events that trigger code execution
- Handling messages
  - On-events in the client
  - Node.JS events in the server (Socket.IO)
  - Similar support for PHP, Servlets, ...
Web Sockets

[XHR polling] GET /poll

[SSE] GET /sse

[WebSocket] GET /ws

HTTP protocol  EventSource protocol  WebSocket protocol

---

CS132 Lecture 11: Web Application Architectures
Socket.IO Server Code

var socket = require('socket.io')

function start() {
  ...
  app.get(...) ...
  var server = app.listen(port);
  var sio = socket.listen(server);
  sio.socket.on('connection',
                 socketConnect);
}

function socketConnect(s) {
  s.on('usercmd1',
        function(data) { uc1(s,data); });
  s.on('usercmd2',
        function(data) { ... });
  s.on('disconnect',
        function(socket) { ... });
}

function uc1(s,data) {
  s.emit('cmd',{ result: 'xxx' });
}
Socket.IO Client Code

<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('http://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my: 'data' });
  });
</script>
Web Applications and HTTP

• The web application assumes it knows the user
  • One request follows another
  • Common shopping cart for the user
  • Look up information based on the user
  • Server needs to know who the user is
    • Even if they haven’t logged in

• HTTP is stateless
  • Each request is independent of previous requests
  • Requests come on different sockets at different times

• This disparity is addressed using **sessions**
Question

Which is not true about sessions in a web application?

A. Sessions represent a connected series of user actions
B. Sessions must have a fixed start
C. Sessions must have a fixed end
D. Sessions can include a variety of different types of information
E. Sessions can be supported by cookies or URL query or post data
What is a Session

• A mechanism for maintaining state
  • For the particular user and the particular web app
  • Within the server
  • Somewhat independent of the browser
• This notion of state is called a session
  • Information for a particular user
  • Information for a particular application
  • Information for a particular use of the application
Sessions

• Represent a connected series of user actions
  • Example: log-in, select-items to purchase, check-out, log-out
  • Example: select source/destination cities, dates; request schedules; select flights; login; purchase tickets
• Needs to have a fixed start
  • Might have a fixed end (log-out request)
  • More likely, time-out if unused; exit when browser closes
What Information is Preserved

• What did you think of in pre-lecture homework?
  • For what application?
  • Between pages
  • Between runs
Session Properties

- What information needs to be kept with the session
  - Depends on the application
- Sample information
  - User id if one exists
  - Host, last-used time
  - Shopping cart
    - Associated with user id?
    - How to handle log in afterwards
  - Input values for forms (to be (re)filled automatically)
  - Previous searches or history
  - Site customization values
Tracking Sessions

• Should the CLIENT track the session
  • If you don’t browse off the page, these can be kept in html
    • Hidden fields, separate DOM tree, etc.
  • But if you replace the page, they disappear
  • Also, if there are multiple pages up, what is used
• HTML 5 Local storage
  • Key-value pairs for the same domain
  • Settable and gettable from JavaScript
  • Works if the information is local & HTML5 is available
Tracking Sessions

• Should the SERVER track the session
  • Maintain as part of state for user
  • But need to send/get it from the browser
    • Server needs to tell the browser the state for new pages
    • Browser needs to tell the server the state for all requests
  • What happens if there are multiple pages displayed
  • What happens with back and forward buttons
• Client and Server both track the session
  • Using cookies
Tracking Sessions
Cookies

- Cookies are a general mechanism
  - For conveying information between browser and server
  - Name-value pairs associated with a particular URL
    - Can have multiple pairs
  - Sent automatically as part of the HTTP header
    - With any request to that particular URL
- Can be set either by server or browser
  - Server: header on a page can request a cookie set
  - Browser: JavaScript functions to define cookies
Cookie Properties

- Name and the value associated with that name
- Maximum age
  - When the cookie should be ignored/removed by browser
  - 0 means when the browser closes
- Domain/port and path
  - When to include the cookie in a HTTP request
  - Domains can be as specific as desired
  - cs.brown.edu, taiga.cs.brown.edu, taiga.cs.brown.edu/myapp
- If you need security, use HTTPS
  - Cookies can be restricted to only work with HTTPS
Cookie Management

- Libraries in server to manage cookies
  - Call to add/set a cookie (language-dependent)
  - Call to read cookie value
  - Added to headers on output pages
  - Used to extract session ids
Session Identifiers

• How much information needs to be conveyed
  • We’ve talked about lots of things, some can be large
  • Really only need one piece of data
    • Use this as an index to a table (or database) on the server
    • Table holds all the information related to the session
  • This is the **session ID**

• Tracking Session Ids
  • Ensure validity (no spoofing; only server-generated IDs)
  • Ensure it is coming from same machine
  • Time out if not used for certain amount of time
Session Management

- Use built in session-support
  - For your server
  - Call to begin/enter session
    - Automatically looks at cookies or url
    - Validates the session
    - Makes session data available
  - Call to terminate session
- Can store arbitrary information with session
Cookies, Sessions and Express

```javascript
var session = require('express-session');
var cookieparser = require('cookie-parser');
...
app.use(cookieparser("KEY"));
app.use(session { secret : "KEY", store: new RedisStore(), ...}));
app.use(sessionManager);
...
function sessionManager(req,res,next) {
  if (req.session.uuid == null) {
    req.session.uuid = <unique id>
    req.session.save();
  }
  next()
}
```
Next Time

- Node.JS lab
- Homework: Prelab for Node.JS
CDQuery (Again)

Find Your CDs

CD Search: 

Find Your CDs

<table>
<thead>
<tr>
<th>Title</th>
<th>CD#1 title and major artist</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
<tr>
<td></td>
<td>* Track 2</td>
</tr>
<tr>
<td>Artist</td>
<td>CD#2 title and major artist</td>
</tr>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
<tr>
<td></td>
<td>* Track 2</td>
</tr>
<tr>
<td>Track</td>
<td>CD#3 title and major artist</td>
</tr>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
<tr>
<td></td>
<td>CD#4 title and major artist</td>
</tr>
<tr>
<td></td>
<td>* Track 1</td>
</tr>
</tbody>
</table>

Find Your CDs

CD TITLE
ARTIST
Description

TRACK Title
Artist
Length
Description

2/21/18

CS132 Lecture 11: Web Application Architectures
XMLHttpRequest

var req = new XMLHttpRequest();
req.onreadystatechange = function () {
    if (req.readyState == req.DONE) {
        if (req.status == 200) << Handle returned data req.responseText>>
        else << Handle error >>
    }
};
req.open("POST","/url/...");
req.setRequestHeader("Content-type", "application/json");
req.send(<data to send>);
Session Management

1. Session setup
   - Select a temporary session ID.
   - Set up a new session on the target server.
   - Extract the session ID from this session.

2. Session fixation
   - Keep the session alive by sending periodic requests.

3. Session entrance
   - Wait for the user to log in.
   - Enter the session.

Figure 11: Web Application Architectures
Sessions in URLs

- Putting sessions Ids in URLs is not a good idea
  - Especially if the URL is public (GET rather than POST)

- Problems
  - GET requests may be logged; server logs now contain private information
  - Copy and paste of URLs can confuse the server
  - Server might use the passed in session id, allowing attacker to steal information

- Solution: use cookies
  - But what if cookies aren’t enabled?
Session Tracking Mechanisms

- Encode the session id in the URL
  - All requests from the browser are URLs
  - The ID can be part of each request
    - http://...?SID=xxxxxxxxxxxxxxxxxxxxx&…

- How to get this into the URLs on the page
  - If requests come from forms, add a hidden field
  - Requests for new pages, replace the URL on generation
  - How to get all URLs on the page
  - Problems?