Module 6 – Node.js and Socket.IO

- Module 6 Contains 2 components
  - Individual Assignment
    - Due on Wednesday
  - Group Assignment
    - We are skipping the group assignment

- Individual Assignment due on Monday (July 30th)

- Read the WIKI before starting

- Portions of today’s slides came from
  - Craig Hecock
  - Iván Loire
  - Roberto Peon
  - Charles Wang

Preview of Creative Project – Due 8/1/18

- Come up with an idea utilizing skills learned from previous modules
  - Should be comparable to work of previous modules
  - You may work alone or in a group on this module

- Push to Bitbucket the project description along with a grading rubric after it has been approved by a TA
  - Rubric is due Monday July 30th by 5:30 PM
  - The project is worth 100 points, 5 of which come from submitting this rubric to Bitbucket
    - You are allowed to assign up to 20 points for a creative portion
    - Create a file named gradingRubric.md inside your Bitbucket repo
503S students will complete a performance evaluation of their web application

The evaluation will include two experiments that measure critical resources used by the app
  - For example, if my app uses Apache and MySQL
    - How many pages/sec can Apache serve?
    - How many reads/sec and writes/sec can my database perform?
  - The wiki provides a list of potential experiments

A performance evaluation proposal is due on July 30th (along with your Creative Project Description)
  - List the experiments you plan to perform and why these are important for your app

Students will submit a written document explaining the experiments and the results by 11:59 PM on Friday August 3rd

Create a file inside your Performance Evaluation Repo named proposalExperiments.md explaining your experiments
What is Node.js?

- A JavaScript runtime environment running Google Chrome’s V8 engine
  - a.k.a. a server-side solution for JS
  - Compiles JS, making it really fast

- Runs over the command line

- Designed for high concurrency
  - Without threads or new processes

- Never blocks, not even for I/O

Concurrency: The Event Loop

- Instead of threads Node uses an event loop with a stack

- Alleviates overhead of context switching
Non-blocking I/O

• Servers do nothing but I/O
  – Scripts waiting on I/O requests degrades performance

• To avoid blocking, Node makes use of the event driven nature of JS by attaching callbacks to I/O requests

• Scripts waiting on I/O waste no space because they get popped off the stack when their non-I/O related code finishes executing

I/O Example

```php
<?php
$result = mysql_query('SELECT * FROM ...');
while($r = mysql_fetch_array($result)){
    // Do something
}
// Wait for query processing to finish...
?>

<script type="text/javascript">
mysql.query('SELECT * FROM ...', function (err, result, fields){
    // Do something
});

// Don't wait, just continue executing
</script>
Traditional (blocking) Threaded Model

Node.js (non-blocking) Event Loop
Threads VS Event-driven

<table>
<thead>
<tr>
<th>Threads</th>
<th>Asynchronous Event-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock application / request with listener-workers threads</td>
<td>only one thread, which repeatedly fetches an event</td>
</tr>
<tr>
<td>Using incoming-request model</td>
<td>Using queue and then processes it</td>
</tr>
<tr>
<td>multithreaded server might block the request which might involve multiple events</td>
<td>manually saves state and then goes on to process the next event</td>
</tr>
<tr>
<td>Using context switching</td>
<td>no contention and no context switches</td>
</tr>
<tr>
<td>Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock</td>
<td>Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments</td>
</tr>
</tbody>
</table>

Node.js VS Apache

<table>
<thead>
<tr>
<th>Platform</th>
<th>Number of request per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP ( via Apache)</td>
<td>318,727</td>
</tr>
<tr>
<td>Static ( via Apache )</td>
<td>296,651</td>
</tr>
<tr>
<td>Node.js</td>
<td>556,930</td>
</tr>
</tbody>
</table>
Node.js Hello World (hello.js)

```javascript
var http = require('http');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello, world! ');
}).listen(80);
```

Node.js Examples
Module 6 Individual Demo

Socket.IO
Web Sockets

- Transport protocol for web communication
  - Alternative to the XMLHttpRequest object used for AJAX
- Provides Bi-directional, full duplex communication channel
- Designed for implementation in web browsers and servers
- Extremely low overhead for payload, just two bytes
- API standardized by W3C

HTTP Overhead (for each request)

"hello, my name is Chrome, encoding UTF-8... I would like a web page please."

- TCP handshake
- HTTP Headers (request)
- HTTP Headers (response)
Do faster connections help?

Page Load Time (PLT) is our measure of latency.

Note the diminishing returns as bandwidth increases.

Do closer server help?

Now we vary the Round Trip Time (RTT) for a fixed bandwidth.

Reducing RTT, always helps reduce Page Load Time (PLT).
Web Sockets

- Help reduce the number of Round Trip Times (RTTs)

TCP handshake (just first request)

Browser

Server

data + 2 byte overhead

data + 2 byte overhead

http://caniuse.com/#feat=websockets
Socket.IO

- **Web sockets for all**
  - Web sockets are not supported in all browsers

- **Socket IO supports a variety of transports**
  - HTML 5 WebSocket
  - Flash Socket
  - AJAX Long Polling
  - Forever Iframe
Getting Started with Socket.IO

• Socket.IO: http://socket.io

• Install Socket.IO with npm:
  – npm install socket.io

Node example using Socket.IO:
var app = require('http').createServer(callback);
var io = require('socket.io').listen(app);
app.listen(8080);

io.sockets.on('connection', function (socket) {
  socket.emit('Hello from server', { hello: 'world' });

  socket.on('Reply from client', function (data) {
    console.log(data);
  });
});
Socket.IO Client (index.html)

```html
<script src="/socket.io/socket.io.js"></script>

<script>
var socket = io.connect('http://localhost');
socket.on('Hello from server', function (data) {
  console.log(data);
  socket.emit('Reply from client', { hello: 'from client' });
});
</script>
```

Simple Socket.IO Demo
Simple Chat Server Demo