CS1320
Creating Modern Web and Mobile Applications

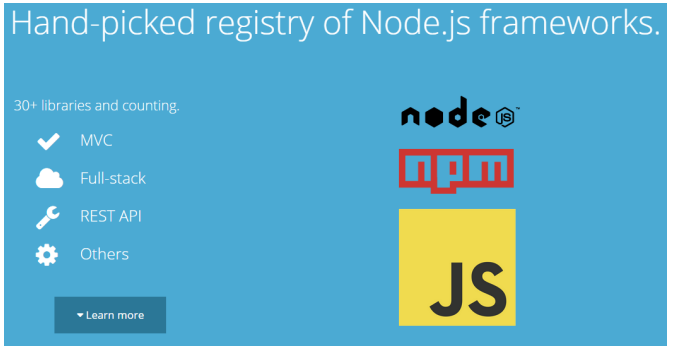
Lecture 13:

Node.JS

Objective

- Create a simple-to-code, powerful web server
  - Designed to scale
  - Designed for modern web applications (e.g. reactive, RESTful)
- Node.JS
  - Reactive, event-based
    - Events for both push and pull based web pages
    - Events for what the server needs to do
  - Using JavaScript

Node.JS



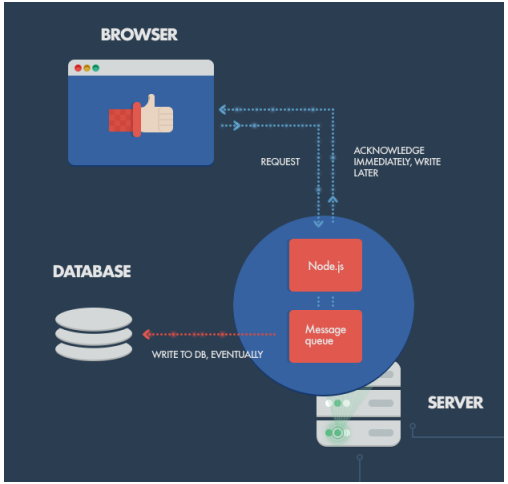
- Skeleton for creating Web Servers
  - Plug-ins provide framework-like functionality
- Frameworks (downloadable modules; micro-frameworks)
  - Express + express extensions: routing
    - URL decoding, sessions, cookies, error handling, internationalization, …
  - Templating
    - Handlebars (mustache), jade, …
  - Email, Login validation
  - Databases
  - Many others

Node.JS Features

- JavaScript (compiled for performance)
- Modularity (avoid namespace clutter)
  - Different from new JS modules
  - Can use new JS modules (depends on node version)
- URL-Based Dispatch
- Event-based (Reactive)
- Powerful Libraries
- Templating
- Designed for modern (RESTful) applications

Simple Node.JS Example

- Remember the first lab
  - Database of CDs
  - Wanted to query the database and return CDs
- Lets see what it takes to implement the back end



Simple Node.JS Server: DEMO I

- Start by showing html pages
- Show node1
  - package.json, npm
  - server.js
  - using express, static display
- Run node1
  - Access localhost:7774/index.html

Node.JS Handling Form Data

- We want to handle the query submit button
  - This will pass back the user's text as a post
- The next step is to modify the server to handle this
  - Detecting the post
  - Handling it

Node.JS Server Demo II

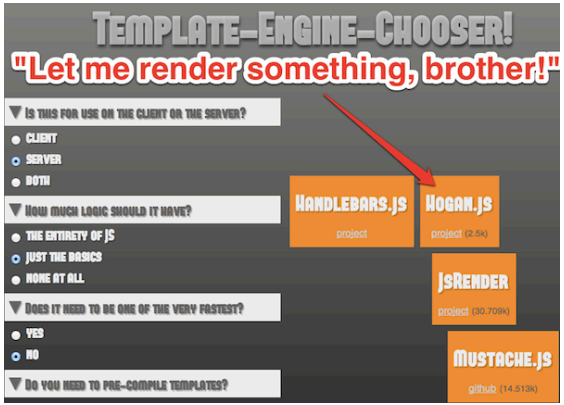
- Show package.json
- Show HTML for the form
- Show server.js additions
  - var query = require("./query.js")
  - var bodyparser = require("body-parser") [npm install]
  - app.use(bodyparser)…
  - app.post(query.handleQuery)
- Look at query.js
  - handleQuery … handleQuery1
  - Note that this doesn't do anything
- Look at database.js

Outputting Templated HTML

- Templating library
  - Handlebars (mustache)
- Template files
  - layouts: overall view of a page
  - bodys: contents of the page
- Insertions
  - {{ var }} : replace with var
  - {{#var}} … {{fld}} … {{/var}} : iterate over array var, fld in i^{th} element
    - Use as if when var is empty or false
  - Other conditions are possible (else, ifnot, functions)

Mustache Templates

- Mustache knows nothing about the template file
  - {{ … }} can occur arbitrarily
- Can generate html text in the parameters
  - parms = { sub : " Substitute … " }
  - text {{ sub }}
- Can generate JavaScript in the parameters
  - parms = { map : ' "MyMap" ' };
  - var mapname = {{ map }};
- Complex structures (objects) can be passed too
  - Using json
  - Using {{{ var }}}



Node.JS Server DEMO III

- Show template directory and structure
  - Show main layout
  - Show template for results page (based on html for that page)
- Show server.js
  - Setup for handlebars
- Show query.js
  - Setup for handlebars
  - Output
- Run server and show the result

Outputting Individual CD Information

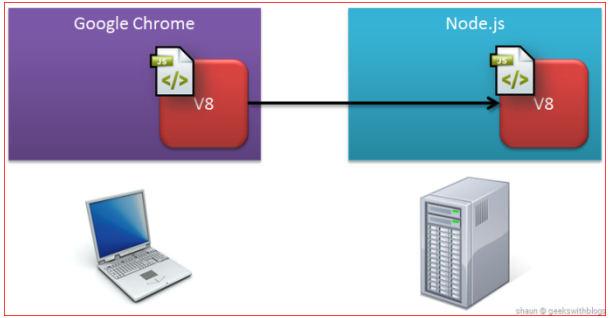
- Clicking on a CD should provide additional information
  - http://localhost:7776/showdisk/<id>
- We need to decipher the URL and provide the page for the CD
- URL-Based Dispatch
  - showdisk is a command
  - <id> is the argument to the command
- Express makes this easy
  - app.get('/showdisk/:diskid',query.handleShow)

Node.JS Server DEMO IV

- Show single template (based on html)
- Show server.js
  - URL additions
- Show query.js: handleShow
  - Note partial computation of rdata and passing
  - Note multiple queries and continuations
- Run server
  - Try some examples

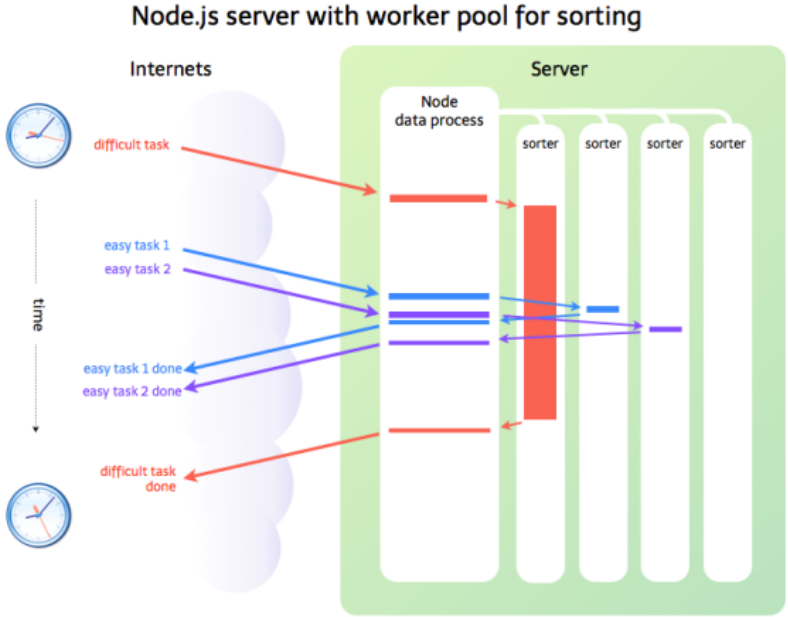
Events in Node.JS

- Recall our server game
  - Multiple people help speed up the service
  - Multitasking can speed up the service
- How to achieve multitasking?
  - Multiple threads
    - This is what apache, nginx, tomcat, … do
    - Threaded coding can be very complex
    - JavaScript does not support threads
  - Multiple servers
    - Need to ensure same user gets the same server
    - Supported by nginx directly
    - Supported by various front ends for apache
    - Supported by a node.js plug-in
  - Multitask without threads



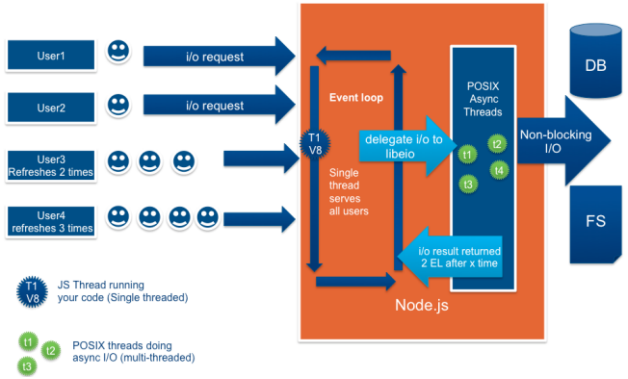
Events in Node.JS

- What does the web server spend its time doin
  - Listening for requests
  - Reading/writing from the network and files
  - Accessing a database or outside server
  - Not much time is spent doing computation
- These tasks run elsewhere
  - Done in the operating system
  - Done in database system or application server
  - Done in background threads in node.js (not javascript)
  - The web server proper spends its time waiting
- Rather than waiting, use non-blocking I/O
  - Start the I/O and let someone else run
  - When I/O finishes, the server is notified and it processes the result
  - Multiple I/O operations can be pending at once
  - Other operations can be treated as I/O

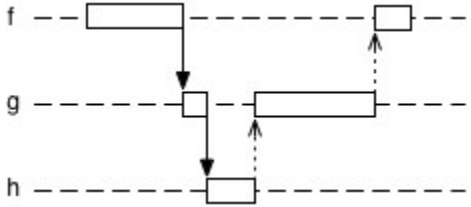


Events and Event Handlers

- Recall how JavaScript works in the browser
  - JavaScript registers for events (onXXX='function()')
  - When something happens, JavaScript is invoked to change the DOM
  - The browser continues execution when JS returns
    - And the change is effected
- Node.JS takes this approach
  - Start an operation via a function call
    - Operation defines a set of events tagged by name
    - Register callbacks (functions) for events of interest
    - Return control to Node.JS
    - This is when the operation actually begins
  - Node.JS will run the operation in background
    - Invoke your callback functions as needed



Functions and Continuations



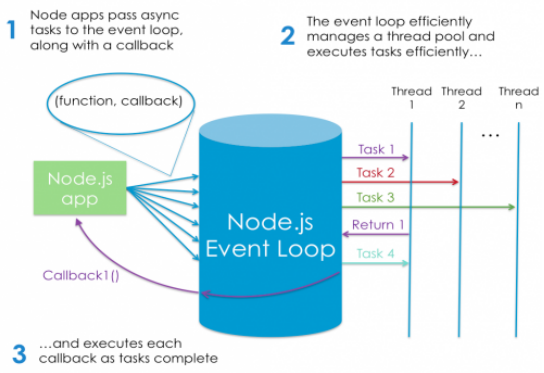
- Callbacks are functions in JavaScript
  - Arguments determined by the event
- Functions in JavaScript can be defined in-line

```
db.query("…",[…],function (e1,d1) { hQ2(req,res,e1,d1); } );
db.query("…",[…], (e1,d1) => { hQ2(req,res,e1,d1); } );
```

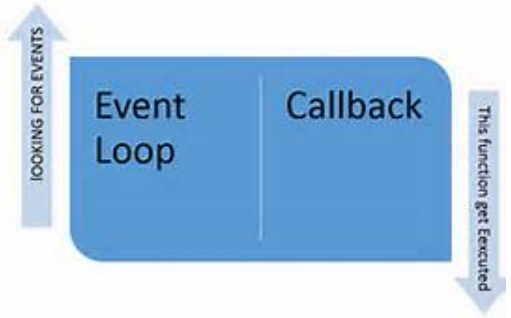
  - When a function is defined this way
    - It can access variables/parameters of the outer function
  - This is effectively a ***continuation***
    - I.e. the inner function defines how execution should continue
    - When the specific event occurs
- Coding practice
  - **Do as multiple functions (very simple in-line function calling next)**
  - **Or use Promises with functions defined separately (not nested)**

Asynchronous Operations

- Node.JS libraries define asynchronous operations
  - File open, read
  - Network read
  - Database queries and updates
  - Web sockets
- Common combined operations also defin
  - Streams: copy from one place to another
    - From the file system to the network
    - All handled in background



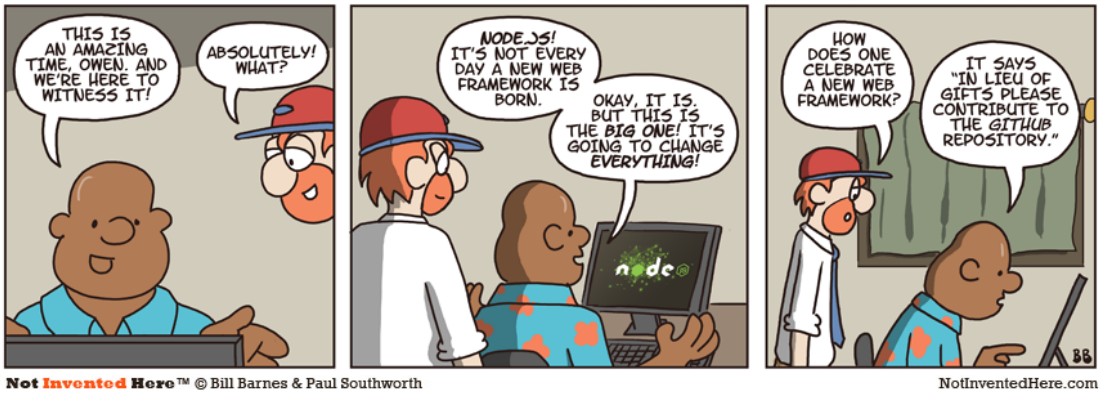
Node.JS Event Example



- Request comes in
  - JavaScript code creates database query based on parameters
  - Starts query and registers continuation
- When query completes (done asynchronously)
  - Template parameters computed from database result.
  - Template file is opened and a new continuation is provided
- When file is ready to read (done asynchronously)
  - A stream from the file to the client is established
  - The file is templated and output is output asynchronously

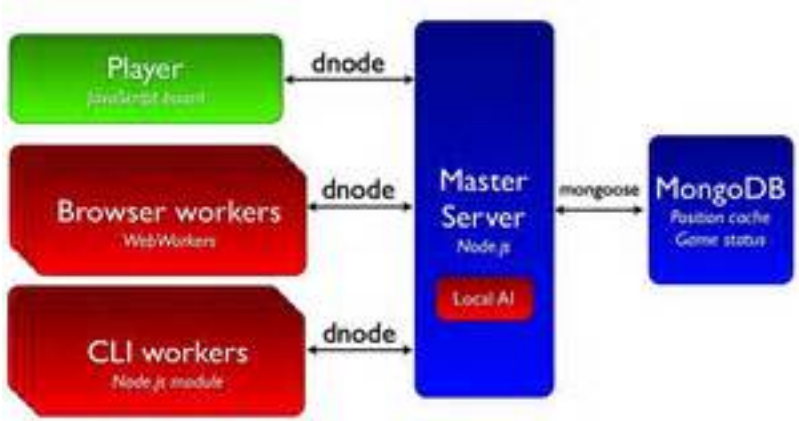
Node.JS Modules

- Synchronous
  - URL decoding
  - File path manipulations
  - Assertions, debugging, read-eval-print loop
  - OS queries
  - Utilities
- Plus external module



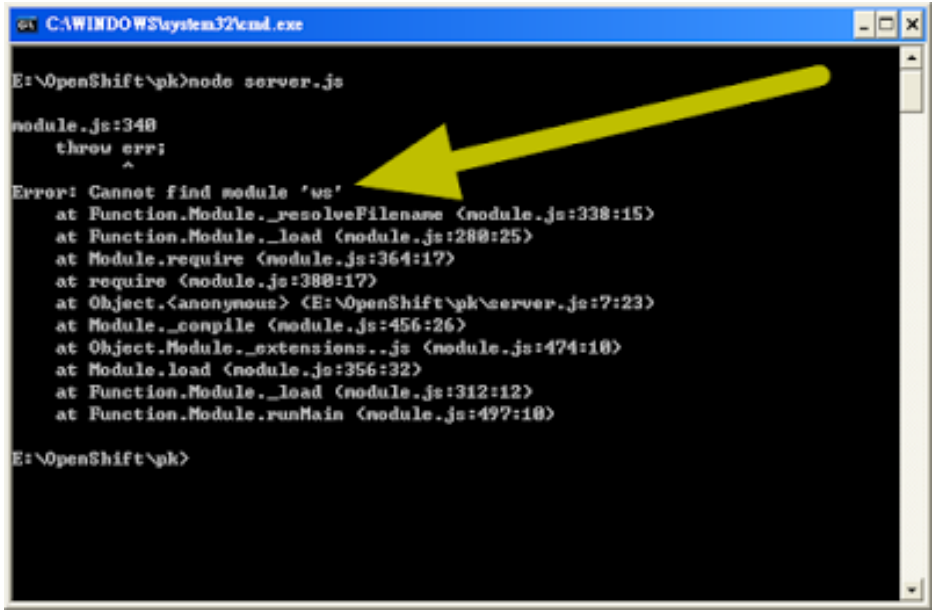
Node.JS Modules

- Asynchronous (event-based)
  - File I/O
  - External processes and code (C/C++)
  - HTTP, HTTPS
  - Crypto, TLS/SSL
  - Database access (SQL/MANGO)
  - Timers
  - Web sockets
- Plus external modules



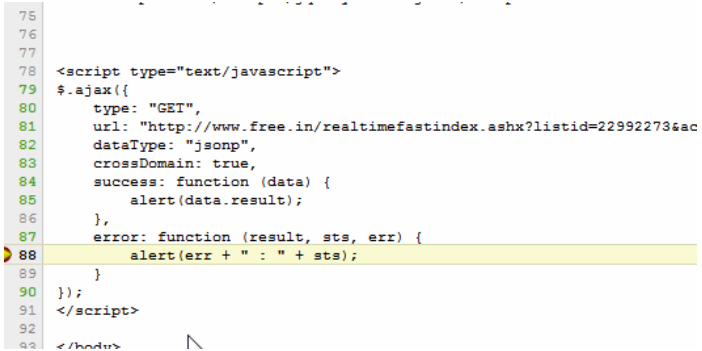
Node.JS Weaknesses

- Documentation
- Coding errors
- Error Recovery
- Scalability

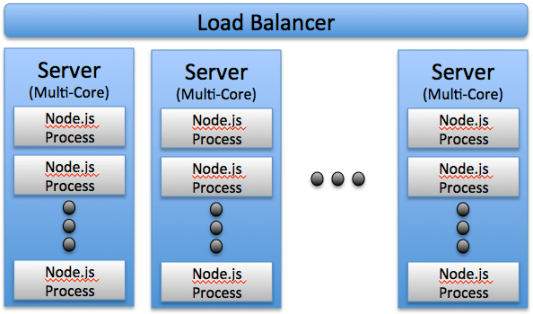


Node.JS Error Recovery

- Node.JS (your server) will halt:
  - At start up if the JavaScript doesn't compile
  - At run time if there are any run time errors
- Is this the desired behavior?
- Exceptions, try … catch
  - Doesn't work that well with asynchronous calls
  - What do you do with an exception?
  - Promise.catch
- Domains
  - Provide a more general mechanism
  - Still require considerable coding
- Add error checking code at each stage
- Try to anticipate errors as much as possible
- Express has some error handling modules



Scaling Node.JS



- Requires running multiple Node.JS server
  - On the same machine (multiple cores)
  - On separate machines (cluster)
- And sending requests based on incoming IP address
- Can be done using NginX or other front end
- Can be done using Node.JS
  - There's a module for that

Next Time

- Web Application Architectures
  - Including cookies, sessions, web sockets, …

Objective

- Create a web server using JavaScript
  - This has been done before: Server-side JavaScript
  - Runs like PHP in the Web Server
  - Didn't really catch on
    - Not what JavaScript was designed for
    - JavaScript was slow (and not as comprehensive) back then
- JavaScript Problems
  - Performance
  - Naming in large systems
  - Single threaded

Example Node.JS application

- Setting up Node.JS
  - Define a package.json file
    - Specifies what external modules are needed
    - These can be automatically downloaded and used
  - npm install
- Look at twitter/nodeweb structure and code

```
{
  "name": "server",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test
specified\" && exit 1"
  },
  "dependencies": {
    "connect": "1.9.2"
  },
  "author": "Joe LeBlanc/ modified by
Mark DuBois",
  "license": "BSD"
}
```

Simple Node.JS Server

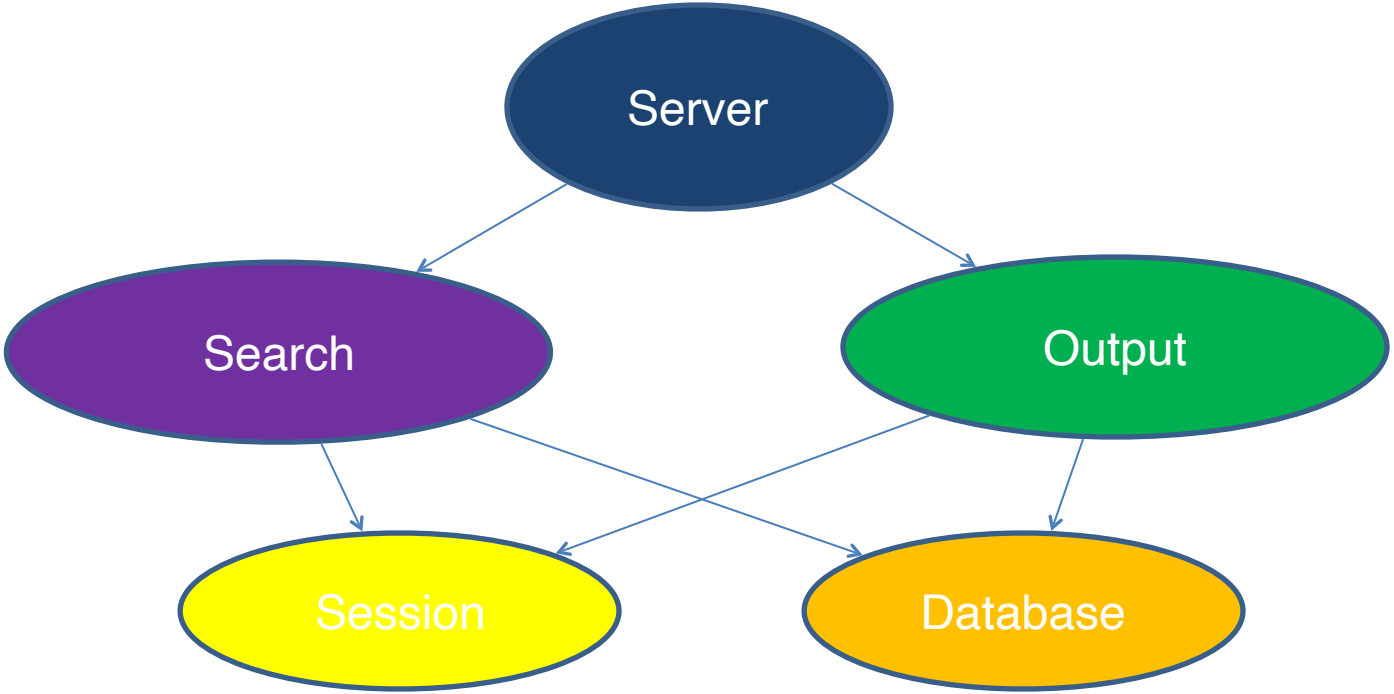
```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200,
        {'Content-Type':'text/plain'});
    res.end('Hello World\n');
}).listen(8888, '127.0.0.1');
console.log('Server running');
```

```
1  var http = require('http');
2
3  http.createServer(function (
   request, response) {
4      response.writeHead(200, {
       'Content-Type':
       'text/html'});
5      response.end(
       '<html><head><title>Node.JS
       </title></head><body>Home,
       URL was: ' + request.url +
        '<p>Powered by
       Node.js</p></body></html>'
       );
6  }).listen(3000, 'localhost');
7
8  console.log('Node.js Server
   running at
   http://localhost:3000/');
```

Twitter Data Access

Feature	Where
Home Page	Static
Define Initial Query	Client
Initial Result Page	Server
Refine Query	Client + WebSocket
Download Results	Server
Show Map-Based Results	Server
Interact with Map	Client

Twitter Web Server Components



Node.JS Modularity

- Each file is a Module
- Files explicitly export accessible items
  - exports.handleQuery = handleQuery;
- Files that use the module, require it
  - var query = require("./query");
- Items can be accessed using the require return
  - query.handleQuery(…);

URL-Based Dispatch

- Normal URLs
  - http://host:port/path/file?args
    - Path and file determine a file in the file system
    - File is the page that is returned
    - Can be PHP (template), XML, HTML, …
- Alternative
  - Use the path and file to determine functionality
    - Call a certain routine based on path & file
    - Access a different file than the one specified

"I begged him to get a smart reader."

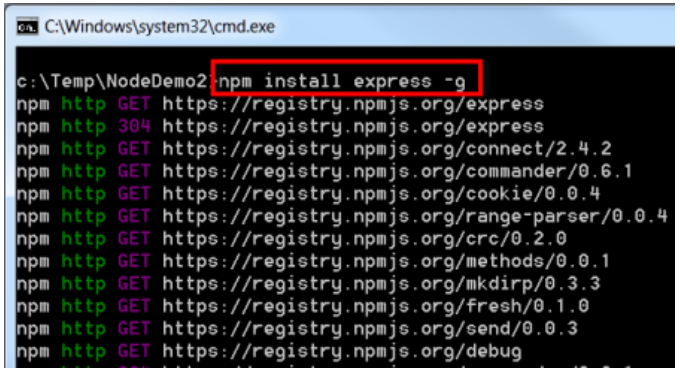
Node.JS Dispatch

- Code

```
http.createServer(function (req, res) {
    res.writeHead(200,
        {'Content-Type':'text/plain'});
    res.end('Hello World\n'); });
```

  - Function invoked for all URLs
    - Can decode the URL as it wants
    - Can then take action based on the request
- Common Actions
  - Invoke a function
  - Output a static page based on path/file
    - HTML, Image, Script, CSS

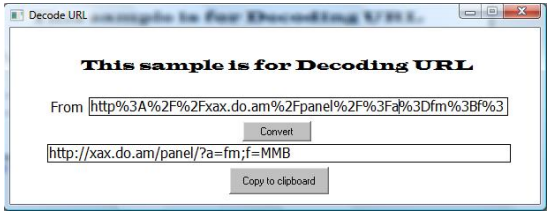
Express Dispatch



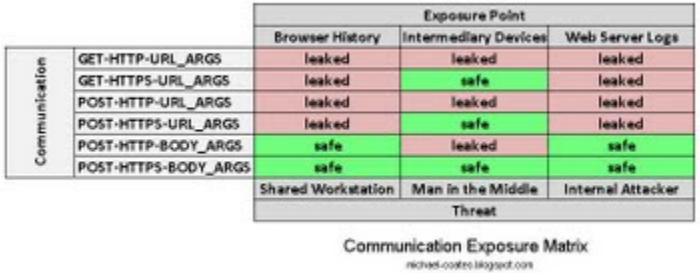
- Defines URL Processing
  - var app = express();
  - app.use("/scripts",express.static(__dirname + "/share/"));
  - app.get("/search",function(req,res) { … });
  - app.post(function(req,res,next) { … });
  - app.put("/devices/:deviceid/", function(req,res) { … });
  - app.use("/",express.static(__dirname + "/html/"));
  - Plug-ins for other functionality
    - app.use(express.logger());
    - sessions, cookies, favicon, error handling, …

Decoding URLs

- Decoding URL components
  - url = require(“url”);
  - var obj = url.parse(string);
  - obj.host, obj.query, …
- Decoding query strings
  - var querystring = require(“querystring”);
    - var q = querystring.parse(data);
    - q.field
  - var bodyparser = required(“body-parser”);
    - app.use(bodyparser.urlencoded({ extended: false }));
    - req.body.field



Handling POSTs



Communication		Exposure Point: Browser History	Exposure Point: Intermediary Devices	Exposure Point: Web Server Logs
	GET-HTTP-URL_ARGS	leaked	leaked	leaked
	GET-HTTPS-URL_ARGS	leaked	safe	leaked
	POST-HTTP-URL_ARGS	leaked	leaked	leaked
	POST-HTTPS-URL_ARGS	leaked	safe	leaked
	POST-HTTP-BODY_ARGS	safe	leaked	safe
	POST-HTTPS-BODY_ARGS	safe	safe	safe
		Threat: Shared Workstation	Threat: Man in the Middle	Threat: Internal Attacker

Communication Exposure Matrix

- Setup code

```
app.use("/search", function(req,res) { decoder(req,res,search.search); });
```

- Decoder Function

```
function decoder(req,res,fct) {
    var data = "";
    req.setEncoding("utf8");
    req.addListener("data",function(chunk) { data += chunk; }
    req.addListener("end",function() { fct(req,res,data); }
}
```

- Function can then use data as such
  - Or querystring.parse(data)

Mustache

Logic-less templates.

- Language-independent templating library
  - Works with Node.JS and other languages
- Template files contain the basic output
  - And places for inserting computed output
    - {{ var }}
  - Substitutions are precomputed (not done in-line)
- Additional features
  - Ability to include other files
  - Conditional regions (mini-programs)
    - For example, if var is defined …
  - Some simple control structures

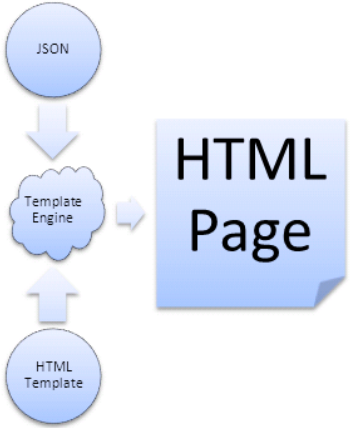
Using Mustache from Node.JS

- Using the module

```
var mu = require("mu2");
var util = required("util");
mu.root = __dirname + "/templates";
```

- Generating a page

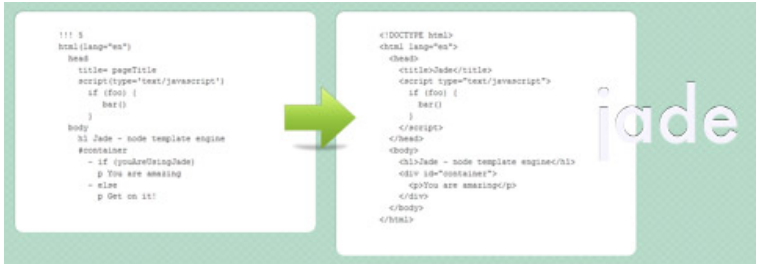
```
var parms = { x : "value", y : "value", ... }
mu.clearCache();
response.writeHead(200,{"Content-Type" : "text/html" });
var stream = mu.compileAndRender("file.html",parms)
util.pump(st,response);
```



Using Mustache with Express

- Set up
  - var exphbs = require(“express3-handlebars’);
  - var handlebars = exphbs.create({ defaultLayout: “main” });
  - app.engine(‘handlebars’,handlebars.engine);
  - app.set(‘view engine’,’handlebars’);
- Use
  - var rdata = { title: “Home Page” };
  - res.render(‘index’,rdata);

Templating



- PHP and JSP use html templates
  - With nested PHP/Java code using <? … ?>
  - These are quite useful for generating pages
    - Substituting text from computation into html output
    - Including standard portions of a page to avoid duplication
- Node.JS does not provide this
  - Different dispatch model
- Several libraries/modules are available however
  - Mustache is similar to PHP/Angular
  - Velocity, jade, …

Question

Node.JS is reactive. This means that:

A. Complex operations are handled in separate JavaScript threads
B. Complex operations are handled by background processing
C. User code works by starting operations and registering callbacks that are invoked when the operations complete or return data
D. A and C
E. B and C

Node.JS Examples

- http://conifer.cs.brown.edu:8888
- http://bdognom.cs.brown.edu:5000