Net ID:_____ **Name:**_____

CSCI-UA.0480-007 – Applied Internet Technology

Midterm Exam

October 26th, 2016

Instructor: Joseph Versoza

Keep this test booklet closed until the class is prompted to begin the exam

- Computers, calculators, phones, textbooks or notebooks are **not allowed** during the exam
- Please turn off your phone to avoid disrupting others during the exam
- Some questions allow you to make assumptions about what's already written or to use **shorthand for html by omitting close tag**, so read the instructions carefully
- Tear off the last page – it can be used as scrap/scratch paper and as a reference
- Choose to skip one of the last 4 questions (7 through 10) by marking it as DO NOT GRADE

1. Write out the full **HTTP interaction** involved in the following scenario. (11 points)

   a) A **user enters** the following **URL in the URL bar** of their **browser**: . http://pizza4.ru/yes/plz/
   b) The browser ends up with a **cookie** called `visited`, that has the value `true`
   c) The browser ends up displaying this:

← the html for the page is *exactly*:
```
<h1>Pizza!</h1>
<img src="/img/za.gif">
```

← the contents of the image can be denoted in your answer below as:
`[image data]`

In the tables below, write out the HTTP requests and responses (there are 4 total) that transpired:

- Specify "**request**" or "**response**" in the **first row**, and write the **entire HTTP** request or response **below** it
- Use **`\r\n`** to denote an explicit newline in your requests and responses
- Write the **requests** and **response** in the **order that they will *likely* occur** starting off with the initial request from the browser (note that the table columns are numbered)
- Assume version HTTP/1.1
- Only add the headers that are necessary for the interaction specified above (based on our homework and class material)

Request or Response?	1 **Request**	2 **Response**
Entire Source of Request or Response	**(+1) GET /yes/plz HTTP/1.1**	**(+1) HTTP/1.1 200 OK** **(+1) Content-Type: text/html** **(+0.5) Set-Cookie:visited=true** **(+1) Pizza! ** **<img src="/img/za.gif">**

Request or Response?	3 **Request**	4 **Response**
Entire Source of Request or Response	**(+1) GET /img/slice.jpg HTTP/1.1** **(+0.5) Cookie: visited=true**	**(+1) HTTP/1.1 200 OK** **(+1) Content-Type: image/gif** **(+1) [image data]**

2. Give a short, one-sentence answer to the following questions about cookies and session management. (3 points)

   a) What does the `HttpOnly` option prevent (be as specific as possible – avoid using "prevents hacking", "is more secure", etc. without detail)?

To prevent JavaScript (specifically 3rd party) from reading cookies via document.cookie

   b) How does the `Secure` option affect the client's (browser's) behavior when dealing with cookies (be as specific as possible – avoid using "prevents hacking", "is more secure", etc. without detail)?

Only sends cookies over secure connection (SSL/TLS)

   c) How does the server tell a client that it *should* delete cookies (be as specific as possible; include any references necessary for HTTP requests/responses)?

Send back a max-age or expires option in the Set-Cookie header

3. **Read the code** in the left column. **Answer the questions** about the code in the right columns. Show your work if possible (for partial credit) (20 points)

Code	Question #1	Question #2

`name = 'databae';`
`function playerInfo(includeScore) {`
`  console.log(this.name);`
`  if(includeScore) {`
`  console.log(this.score);`
`  }`
`}`
`playerInfo(false);`
`var player = {name: 'amirite42'};`
`var info = new playerInfo(false);`
`playerInfo.call(player, true);`
	a) What is the output of this program (error or nothing are possible, explain why if error)?
	
	databae
	undefined
	amirite42
	undefined
		b) In *one line*, use a method that you can call on the function, `playerInfo`, to create a new function, **`newFn`**, where `playerInfo`'s `this` is `player`, and `includeScore` is always true (the resulting function has no parameters).
		
		`playerInfo.bind(player,true);`

`var makeCounter = function(start) {`
`  var count = start || 0;`
`  return {`
`  inc: function() {`
`  count += 2;`
`  },`
`  print: function() {`
`  this.inc();`
`  console.log(count);`
`  }`
`  };`
`};`
`var counter = makeCounter(8);`
`for(var i = 0;i < 4; i++) {`
`  counter.print();`
`}`
	c) What is the output of this program (error or nothing are possible, explain why if error)?
	
	10
	12
	14
	16
		d) What would that output of these two lines be if they were added to the end of the code in the first column?
		
		`console.log(typeof makeCounter);`
		`console.log(typeof counter);`
		
		function
		object

`function extractMediaType(f) {`
`  var i = f.lastIndexOf('.');`
`  var ext = f.substring(i + 1);`
`  var type;`
`  if(ext === 'css') {`
`  type = 'text';`
`  } else if (ext === 'jpg'){`
`  type = 'image';`
`  }`
`  console.log(format(type, ext));`
 	
`  var format = function (t, s) {`
`  return t + '/' + s;`
`  };`
`}`
`extractMediaType('wavey.jpg');`
	e) What is the output of this program (error or nothing are possible, explain why if error)?
	
	error, format isn't a function (declaration is hoisted, but definition is not)
		f) Rewrite the code that sets the variable, `type`, (based on an extension) so that it **doesn't** use an if statement, switch/case statement, or ternary operator:
		
		`var MediaTypes = {`
		`  css:'text',`
		`  jpg:'image'`
		`};`
		
		`type = mediaTypes[ext];`

`var arr1 = [1, 2, 3];`
 	
`Array.prototype.last = function() {`
`  if(this.length > 0) {`
`  var i = this.length - 1;`
`  return this[i];`
`  }`
`};`
`console.log(arr1.last());`
 	
`var arr2 = [];`
`var r = arr2.hasOwnProperty('last');`
`console.log(r);`
`console.log(arr2.last());`
	g) What is the output of this program (error or nothing are possible, explain why if error)?
	
	3
	false
	undefined
		h) What is the **[[prototype]]** of `Array.prototype.last`? What is the **[[prototype]] of the [[prototype]]** of `Array.prototype.last`?
		
		Function.prototype
		Object.prototype

`function verbosify(fn) {`
`  return function(arg) {`
`  var res = fn(arg);`
`  console.log((typeof arg), arg);`
`  console.log((typeof res), res);`
`  return res;`
`  };`
`}`
`var newFn = verbosify(parseInt);`
`var n = newFn('11');`
`console.log(n);`
	i) What is the output of this program (error or nothing are possible, explain why if error)?
	
	string 11
	number 11
	11
		j) Modify the function, `verbosify`, in the left column so that newFn will behave like this:
		`newFn('101', 2) // outputs 5`
		Do this by rewriting line 3 only (ignore the lines directly above and below it)
		
		`var res = fn.apply(null, arguments)`

4. What are the two HTTP request methods that we've used to send data to the server? When a form is submitted with either method, specify where in the *actual* HTTP request the data from the form elements is located. On the server side, what object / property in the Express framework contains that data? Lastly, when would you use one request method over the other? (4 points)

Method Name	Where in HTTP Request is It Located?	Name of Obj /Prop in Express API
GET	Query string of url	req.query
POST	Body of http request	req.body

When would we use one request method over the other?	GET for reading resources, POST for creating … or GET when you want "bookmarkable" links / allow for url hacking

5. Answer the following database related questions. (3 points)

   a) Aside from a database, name **three** other potential mechanisms for persistent data storage (solutions that persist data only while the application are running are also acceptable):

   the "cloud", local file system, in-memory, store data on the client via cookies or "local storage"

   b) In the context of MongoDB, describe the following: **document**, **collection** and **database**

   document is composed of key value pairs, collection is a group of documents, database is a group of collections

6. Answer questions about the server below. (5 points)

```
var net = require('net');
var server =  net.createServer(function(sock) {
    sock.on('data', function(data) {
        var line = data + '';
        var lineParts = line.split(' '); // space
        var command = line[0];
        var op1 = +lineParts[1];
        var op2 = +lineParts[2];
        if(command == '*') {
            sock.write('ANSWER ' + (op1 * op2));
        } else if (command == '+') {
            sock.write('ANSWER ' + (op1 + op2));
        } else {
            sock.write('ERROR: unsupported operation');
        }
    });
});
console.log('READY 4 U');
server.listen(3000);
```

   a) What is printed to the screen when you first start this server (error or no output are possible)?

   READY 4 U'

   b) **What client** would you use to **connect to this server**, and what is **printed** to the screen on the **server** side when you use this client to **connect** to it?

   curl or net module, no output

   c) What **data** would the **client receive** (that is, what would be **printed** on the **client** side) if the **client sends**: `* 4 4` to the server (error or no output are possible)?

   16

   d) As the client, what data could you send to the server so that the response that you get back is `Error: unsupported operation`

   A 1 2

   e) What **data** would the **client receive** (that is, what would be **printed** on the **client** side) if the **client sends**: `+ 4 4 4` to the server (error or no output are possible)?

   8

Choose 3 out of the next 4 questions to answer. Mark the question you want to skip by writing DO NOT GRADE at the top of the question.

7. You have a directory called **`static`** in your Express project, and it contains html and css files. You'd like to implement a feature that lets you serve the files in that `static` folder without specifying explicit paths for each file. (18 points)

   Here's what your app will do:

   a) send the contents of files in the `static` folder back as http response bodies based on the url that was requested
   b) for example requesting http://your.site/baz.html would read the contents of `static/baz.html` and send it back as part of the response without having to define a route handler for `/baz.html`
   c) if the path specified in the url does not match a file in your static folder, then let the rest of your application handle the request (for example, if the original request was for '`/foo`', but `foo` doesn't exist in the `static` folder, your app may still be able to handle that path through a regular route handler like `app.get('/foo', …)`
   d) to implement this:
      - for every incoming request, regardless of path...
      - try to find the resource that the request is asking for in your app's static folder
      - you can assume that if the path doesn't have an extension of html or css, that the file is not in `static`
      - if the file exists, serve up the file's content (which can be html or css)
      - make sure the browser knows how to handle the contents of the request body!
      - if there's **any issue opening the file** (for example, if it doesn't exist)
      - just **allow the request to continue being handled** by the *rest* of your application
      - you **cannot use `express-static`** (this will essentially replicate the functionality of express static) or **`res.sendFile`**
      - **hints**:
        - you'll probably need to figure out the extension of the file... when you extract the extension, make sure that the dot (.) is not included, and also make sure to handle the case where there's no extension!
        - you'll need to use the `fs` module to open a file; this should be familiar from homework...
        - make sure you check the error object; it will let you know if there was an error reading the file:
        - fs module usage summary / refresher:

```
var fs = require('fs');
fs.readFile(fileName, callbackFunction);
// callbackFunction has two parameters:
// err – undefined/null if read worked, an object if there was an error
// for example, if they file were not found err would be an object
// data - the contents of the file on a successful read
```

Write code that does this below. **Assume that all the Express setup requirements are already taken care of (require express, etc.). Just add the code to implement this feature below**.

```
var fs = require('fs');
var extMediaType = {
    html:'text/html',
    css:'text/css',
    txt:'text/plain'
};

app.use(function(req, res, next) {
    var path = './static' + req.path;
    var ext;
    if(req.path.lastIndexOf('.') !== -1) {
        ext = req.path.substring(req.path.lastIndexOf('.') + 1);
    } else {
        next();
    }

    fs.readFile(path, function(err, data){
        if(err) {
            next();
        } else {
            res.set('Content-Type',extMediaType[ext] );
            res.send(data);
        }
    });
});
```

8. Write code for the following... create two higher order functions, **`pluckWith`** and **`any`** (parts a and b), and use `reduce` to count values in an Array (part c). (10 points)

   a) **`pluckWith`**`(prop)` – **returns a function** that takes a list of objects and picks out a value from each object that corresponds to the property name specified by the argument, `prop`. The result its a list of values. For example:

```
var teams = [
    {team:'celtics', city:'boston'},
    {team:'pistons', city:'detroit'},
    {team:'kings', city:'sacramento'},
]

// creates a new function that picks out the team name from every object in a list of
// team objects... and gives back the team names as a list
var pluckTeamName = pluckWith('team');

// creates a new function that picks out the city from every object in a list of
// team objects and gives back the cities as a list
var pluckCity = pluckWith('city');

// our new functions can take a list of objects as an argument; let's call them on teams:
console.log(pluckTeamName(teams)); // prints out ['celtics', 'pistons', 'kings']
console.log(pluckCity(teams));     // prints out [''boston', 'detroit', 'sacramento']
```

   b) **`any`**`(arr, fn)` – returns true if *any* (at least one) of the elements in the Array, `arr`, pass the test, `fn`. The test, `fn`, is a function that takes a single argument, and returns either true or false. You **cannot use a for loop or forEach loop** in this.

```
// prints out true
console.log(any(teams, function(obj) {
  return obj.team == 'pistons';
}));
```

```
// prints out false
console.log(any(teams, function(obj) {
    return obj.team == 'knicks';
}));
```

   c) Use reduce on this Array: `var nums = [41, 41, 41, 80, 80] ...`So that the result is an object with keys as the unique numbers in the Array, and values are the number of occurrences of the number in the Array: { '41': 2, '80': 1 }

```
function pluckWith(prop) {
    var newFn = function(arr) {
        return arr.map(function(ele) {
            return ele[prop];
        });
    };
    return newFn;
}


function any(arr, fn) {
    return arr.reduce(function(accum, ele) {
       return fn(ele) ? true : accum;
    }, false);
}


var nums = [41, 41, 41, 80, 80];
var res = nums.reduce(function(accum, cur) {
    accum[cur] = accum[cur] + 1 || 1;
    return accum;
}, {});
```

9. Write a constructor that makes **`Request` objects**. Unlike our homework, this object **will not parse** an HTTP Request string. Instead, its **properties will be set** through **individual arguments** passed to the constructor. Additionally, it will have the **ability to send an actual http request** and **print** out the resulting response! (18 points)

   You will do this by using the net module as a client (see the example client code in the first column below) and by examining the example usage of a `Request` object in the second column.

Client Code with net Module Example

```
var net = require('net');
var client = new net.Socket();

// connect to port and domain
client.connect(3000, 'localhost', function() {

  // this code is only executed when this
  // client successfully connects to a server
  console.log('Connected');

  // write sends data to the server connected
  // to
  client.write('Example being sent');
});

client.on('data', function(data) {
  // this code is executed when the server
  // sends
  // data back to the client
  console.log('Received: ' + data);
  client.end();
});
```

Example Usage of Request Constructor and Object

```
// create a new Request object with constructor
var req = new Request('GET', 'www.google.com', 80);

// resulting object has these properties
console.log(req.method);  // GET
console.log(req.domain);  // www.google.com
console.log(req.port);    // 80
console.log(req.path);    // /

// the properties are defined on the object itself
console.log(req.hasOwnProperty('method')); // true

// the method, send, is defined elsewhere???
console.log(req.hasOwnProperty('send'));   // false

// send the request
req.send()

// prints out response from server
// HTTP/1.1 200 OK … etc.
```

   a) Note that the resulting `Request` object has the following **properties**: **`method`**, **`domain`**, **`port`** and **`path`**
   b) These properties are set from the **constructor's parameters**: **`method`**, **`url`** and **`port`**
   c) `domain` and `path` are taken from `url` – you can assume that no protocol is included (no http://), everything before the 1st / is the `domain`, and everything after is the `path`. For example: foo.bar/baz/qux → `domain` is `foo.bar` and `path` is `/baz/qux`
   d) If the domain doesn't contain a /, assume the whole url is the `domain`, and the `path` is `/`
   e) The `Request` object's **`send`** method **connects to the server** specified by the `domain` and `port`, **sends a valid http** request (no need to add a Host header), and **prints out the response** when it receives data
   f) **Draw arrows** to the parts of **client code you want to copy** over, **copy and modify** the **lines** that you want **to change**.

In particular, **watch out for the value of `this`** when using callbacks (either bind this to another name in the closure, make the variables you need available in the closure or use bind).

```
function Request(method, url, port) {
    var i = url.indexOf('/');
    this.path = '/';

    if(i == -1) {
        i = url.length;
    } else {
        this.path  = url.substring(i);
    }

    this.method = method;
    this.domain = url.substring(0, i);
    this.port = port;
}

Request.prototype.send = function() {
    var client = new net.Socket();
    var that = this;
    client.connect(this.port, this.domain, function(){
        var s = that.method + ' ' + that.path + ' HTTP/1.1\r\n\r\n';
        client.write(s);
    });
    // client.on('data') part from above
};
```

10. **Write the route handlers and complete the template for a small Express application.** The app is a collaborative poem where users add a line to a poem by submitting an adverb and verb combination. Every combination entered by every user is displayed. The last line entered is specific to each client, though. The lines entered into the poem are saved in mongodb. (18 points)

A Poem **Add to the poem:** adverb: casually verb: cook Submit	**A Poem** casually cook **Add to the poem:** adverb: verb: Submit **Last line entered by you**: casually cook	**A Poem** casually cook elatedly eat soundly sleep **Add to the poem:** adverb: verb: Submit **Last line entered by you**: soundly sleep
The path `/poem` displays all of the lines of the poem (starting with no lines), along with a form to add a new line.	Submit adds a line to the poem and takes you back to the original page (showing the new line added, as well as displaying it as the last line *you* added).	Lines can continually submitted to to construct a poem.

a) Assume that any possibly required modules and configuration is already present (body-parser, hbs, mongoose, express-sessions, layouts, etc.); **you only have to write the routes and templates**
b) Assume that you have a mongoose model, available, **`Line`**, that has **two string** fields: **`adverb`** and **`verb`**
c) Assume that in your template, you already have code that iterates through an Array of Line objects that displays the lines in the poem - **you just have to finish the form and the last line entered**
d) Assume that **no error handling is necessary**
e) When you write your templates, you can omit closing tags as a shorthand way of writing html
f) There's only one page in the entire application: `/poem`
g) You should **not** be able to refresh the page and cause a form submission to resubmit
h) The poem looks the same for all users (that is, if I add a line on my browser, it will show up when you view the page on your browser). However, the last line entered is specific to a client (that is, if I add a new line, I see the last line I entered... but if I open a different browser, last line entered will be blank)

```
app.get('/poem', function(req, res) {
  Line.find({}, function(err, lines, count) {
    res.render('index', {lines: lines, last: req.session.last});
  });
});
app.post('/poem', function(req, res) {
  var line = new Line({
      adverb: req.body.adverb,
      verb: req.body.verb
  });
  line.save(function(err, line, count){
    req.session.last = req.body.adverb + ' ' + req.body.verb;
    res.redirect('/poem');
  });
});
<form method="POST" action="">
              adverb: <input type="text" name="adverb" value="">  <br>
              verb: <input type="text" name="verb" value=""><br>
              <input type="submit"><br>
</form>
last line you entered: {{last}}
```

Name_____ **Net ID**_____

Objects / Methods Reference

Array

properties

length

methods

pop()

reverse()

sort([compareFunction])

splice(index, howMany[, element1[, …[, elementN]]])

slice(index, howMany[, element1[, …[, elementN]]])

join([separator = ','])

concat(value1[, value2[, ...[, valueN]]])

indexOf(searchElement[, fromIndex = 0])

forEach(callback[, thisArg])

map(callback[, thisArg])

filter(callback[, thisArg])

reduce(callback[, initialValue])

some(callback[, thisArg])

every(callback[, thisArg])

String

properties

length

methods

split([separator][, limit])

toUpperCase()

slice(beginSlice[, endSlice])

replace(regexp|substr, newSubStr|function[, flags])

substring(begin[, end])

indexOf(ch)

lastIndexOf(ch)

Object / Object.prototype

Object.getPrototypeOf(obj)

Object.prototype.hasOwnProperty(prop)

Mongoose Model

as a constructor

var obj = MyModel({prop1:val1, prop2:val2})

instance methods

obj.save(fn) // fn args: err, objs, count

"static" methods

MyModel.find(fn) // fn args: err, objs, count

Request Object

properties

body

headers

method

path

query

session

url

methods

get

Response Object

methods

append

end

redirect

render

send

sendFile

set

status

writeHead