CSCI UA.0480 – Applied Internet Technology

Final Exam Practice Questions Set 2

1. Answer these questions about salt!

   a) In the context of storing passwords after a user registers, when is a salt added?

   After a password is retrieved, but before the password is hashed.

   b) How is the salt used to authenticate a user when a user is attempting to login by sending a username and password?

   The salt is appended to the incoming password... the result is hashed … and compared to the hash in the database

2. If two CSS rules target the same elements, how do you determine which rule will be applied? A high level description is adequate.

   The rule with the more specific selector is applied. Specificity is determined by concatenating the counts of types of selectors and using the resulting numbers for comparison.

   (the actual algorithm is... something like concat(num id selectors, num class selectors, num element selectors))

3. Describe at least 2 disadvantages when using AJAX polling to simulate real time communication in web applications?

   a) **not really real-time – depends on how often you poll / how long polling interval is**

   b) **server has to handle large volume of requests (every client will poll!)**

   c) **client side performance will also be degraded because client will have to continuously issue requests**

4. Describe 2 situations where you should not (or at least be very careful!) using arrow functions:

   a) **as callbacks / event handlers in addEventListener**

   b) **to create constructors**

   c) **to create methods**

5. One example of cross site request forgery is a script running on a malicious site that issues a POST request to another site that the user may be logged in to. Because the user is already logged in, the malicious script has access to the user's authenticated session! Describe the technique(s) / mechanism(s) used to prevent this.

   a) **the web application will require token as part of POST data to verify that request is coming from form (rather than script)**

   b) **the web browser will not allow requests from scripts from one origin to read resources from another origin (SOP)**

6. Answer the questions in the 2nd and 3rd columns about the code in the 1st column:

<!-- html -->
<section>
  automaton cyborg
  robot ai
</section>
 machine
 	
// javascript
const s = 'section > p'
const res = document.querySelector(s);
console.log(res.textContent);
console.log(res.innerHTML)
	What is the output of the 1st column assuming the html and JavaScript provided?
	
	robotai
	
	** robot ai**
		Write a css rule that makes the first div under section "disappear".
		
		**section > div {
		display: none;
		}**

function f(val) {
console.log(val);
return new Promise((fulfill, reject) => {
console.log('acoustics');
fulfill('bandit');
console.log('cotton');
});
}
 	
const result1 = f('desert');
const result2 = result1.then(f);
const result3 = result2.then(console.log);
	What is the output of the code?
	
	**desert
	acoustics
	cotton
	bandit
	acoustics
	cotton
	bandit**
		What is result3 in the last line (type and value)?
		
		It's an object, specifically a Promise, that fulfills immediately, with the value undefined (because console.log doesn't return anything, it gives back undefined, a non-Promise value)

const fs = require('fs');
 	
function Reader(fn, prefix) {
this.fn = fn;
this.prefix = prefix;
console.log('fn is:', this.fn);
console.log('prefix is:', this.prefix);
}
 	
Reader.prototype.print = function() {
fs.readFile(this.fn, 'utf8',
function(err, data) {
if(!err) {
console.log(this.prefix);
console.log(data);
}
}
);
};
 	
const r = new Reader('colors.txt', 'RAINBOW');
r.print();
	What is the output of the code? The data parameter in the readFile callback will contain the contents of the file read as a regular string. Assume that colors.txt exists, and it contains this data:
	
	red
	orange
	yellow
	
	**fn is colors.txt
	prefix is RAINBOW
	undefined
	red
	orange
	yellow**
		Using the same definition for the Reader constructor. What would the output be if the only line of code after the constructor were:
		
		Reader();
		
		**fn is: undefined
		prefix is: undefined**

<!-- markup -->
 	
 - A
 - B
 - C
 - D
 	
// javascript
var lis = document.getElementsByTagName('li');
for(let i = 0; i <lis.length; i++) {
lis[i].parentNode.removeChild(lis[i]);
}
	What will the markup look like after the script in the 1st column is run on the markup in the 1st column?
	
	**
	 - B
	 - D
	  **
		Fix the code so that all of the list items are removed (but the outer ul remains present):
		
		**var ul = document.
		getElementsByTagName('ul')[0];
		
		while(ul.firstChild) {
		ul.removeChild(ul.firstChild);
		}**

const glue = {
sep: 'x',
join(arr) {
return arr.reduce((acc, cur) => {
return acc + cur + this.sep;
}, '');
}
};
console.log(glue.join(['foo', 'bar', 'baz']));
	What is the output of the code?
	
	fooxbarxbazx
		Rewrite the callback to arr.reduce so that it uses bind instead of an arrow function.
		
		**// in join
		function(acc, cur) {
		return acc + cur + this.sep;
		}
		return arr.reduce(f.bind(this),
		'');**

7. Create a React component, Adder, that displays 2 numbers and the sum of the 2 numbers. Both numbers start at 0. Every time a number is clicked on, it is incremented based on an attribute, called inc, that the component is rendered with. For example, if the component were rendered as: <Adder inc='1'>, then the following interactions can take place:

Initial Page:	Clicking on the top Number:	After clicking 3 and 5 times:
Num: 0 Num: 0 Sum: 0	Num: 1 Num: 0 Sum: 1	Num: 3 Num: 5 Sum: 8

Assume the following components are already present. Use es6 classes or createClass to implement the parent Adder component:

```
class NumBox extends React.Component {
    render() {
        return <div onClick={this.props.onClick}>Num:
{this.props.num}</div>;
    }
}
```

```
class Sum extends React.Component {
  render() {
    return <div>Sum: {this.props.sum}</div>;
  }
}
```

```
class Adder extends React.Component {
  constructor() {
    super();
    this.state = {
      boxes: [0, 0]
    };
  }

  handleClick(i) {
    const newBoxes = this.state.boxes.slice();
    newBoxes[i] += (+this.props.inc || 1);
    this.setState({boxes: newBoxes});
  }

  render() {
    const boxes = [];
    for(let i = 0; i < this.state.boxes.length; i++) {
      boxes.push(<NumBox onClick={() => {this.handleClick(i)}} num={this.state.boxes[i]}/>);
    }
    const sum = this.state.boxes.reduce((sum, n) => { return sum + n;}, 0);
    return <div>{boxes}<Sum sum={sum} /></div>;
  }
}
```

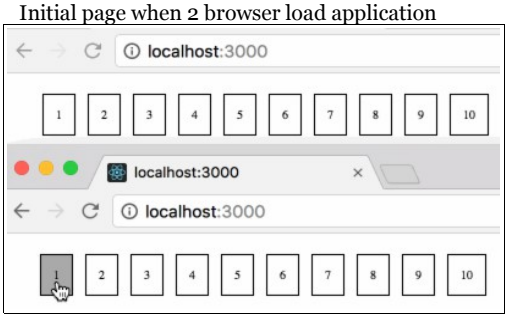
8. Create a realtime, multi-user web application that displays 10 numbers in a single row:
   a) if a user clicks on one of the numbers, it disappears from the user's screen as well as any other users' screens in real time
   b) if a new user loads the page, the page will only show the remaining numbers (the numbers that haven't been clicked on yet)
   c) write out the (1) server code and (2) client code (including DOM manipulation to create elements)
   d) elements must be created programmatically, though assume that the following code is already present for you:

```
// ON THE SERVER
const express = require('express');
const app = express();
const server =
require('http').Server(app);
const io = require('socket.io')(server);
app.use(express.static('public'));

// TODO: fill out server code

server.listen(3000);
```

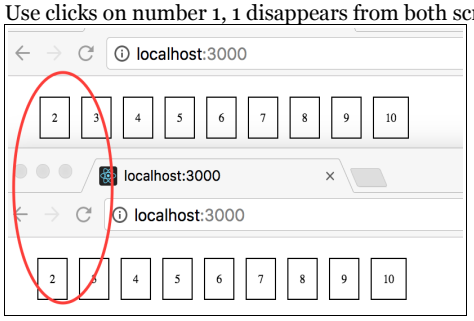
```
// ON THE CLIENT
// assume that span elements are styled with borders
// assume that there is no markup (only html, head, body)
// /socket.io/socket.io.js is already included

const socket = io();
document.addEventListener('DOMContentLoaded', function()) {

  // TODO: fill out client side code

}
```

   e) see example interaction below:

Initial page when 2 browser load application



Use clicks on number 1, 1 disappears from both screens




```
// server code
const numbers = [];
for(let i = 1; i < 11; i++) {
    numbers.push(i);
}

io.on('connect', (socket) => {
    socket.emit('init', numbers);
    socket.on('remove', (n) => {
        numbers.splice(numbers.indexOf(n), 1);
        socket.broadcast.emit('remove', n);
    });
});

// client code
socket.on('init', (numbers) => {
    numbers.forEach((n) => {
        let div = document.createElement('span');
        div.classList.add('val' + n);
        div.textContent = n;
        document.body.appendChild(div);
        div.addEventListener('click', function() {
            socket.emit('remove', +this.textContent);
            this.parentNode.removeChild(this);
        });
    });
});


socket.on('remove', (n) => {
   const div = document.querySelector('.val' + n);
   if(div) {
        div.parentNode.removeChild(div);
   }
});
```