Advanced JS Developer Interview

Please answer the following 5 questions to the best of your ability. One question exists per page. If the question involves a language or framework you're unfamiliar with (e.g. Java, Facebook React), note that and move on to the next question. The entire interview should take about an 1 hour.

1. **Describe the tools you use for compiling, testing and debugging your your JavaScript code.**

Webpack (alternatively **Browserify** or **RequireJS**) for module compiling.

Jest (alternatively **Jasmine**) for client-side Unit Testing.

Chrome Developer Tools for debugging.

Mocha for server-side testing of JavaScript.

Gulp(alternatively **Grunt**) for automation of routine tasks.

Npm/Bower for package management.

2. **Fix this object. It contains at least 2 mistakes.**

```
function AverageAggregation() {
   this.count = 0,
   this.total = 0;


   this.add = function(value) {
       if (_.isArray(value)) {
           _.each(value, function(value) {
               this.total += value;
               this.count++;
           });
       }

       this.total += value;
       this.count++;
   };


   this.result = function() {
       return this.total / this.count;
   };
}
```

Fixed

```
var _ = require('underscore');

function AverageAggregation() {
  this.count = 0,
  this.total = 0;

  var parent = this;

  this.add = function(value) {
    if (_.isArray(value)) {
      _.each(value, function(iter) {
        parent.total += iter;
        parent.count++;
      });
    }
    if (_.isNumber(value)) {
      parent.total += value;
      parent.count++;
    }
  };

  this.result = function() {
     return this.total / this.count;
  };
}

var averageAggregation = new AverageAggregation;
```

```
averageAggregation.add([1, 2, 3, 4]);
averageAggregation.add(7);
console.log(averageAggregation.result()); // => 3.4
```

3. **You want to allow third-party developers to write their own data parsers. Given the following loadData function, write code that provides a value for the 'parser' variable. Feel free to add other functions and objects—you are not restricted to doing it all inline. Ensure that developers using your code can easily register new parser types without updating your code.**

```
function loadData(url, parserType, options) {
   var parser = ????

   return parser.loadData(url);
}

loadData('mydata.csv', 'csv', { delimiter: ','});
loadData('mydata.json', 'json');
```

Answer:

```
function loadData(url, parserType, options) {
  var parser = {
    parser_type: parserType,
    parser_options: options,
    loadData: function(url) {
      return url;
    }
  };

  return parser.loadData(url);
}

var psr1 = loadData('mydata.csv', 'csv', { delimiter: ','});
var psr2 = loadData('mydata.json', 'json');

console.log(psr1);
console.log(psr2);
```

4. **Port the following Java code to JavaScript. The variable prefixes represent scope. Feel free to remove these and use normal JavaScript naming conventions and formatting. Use underscore, jQuery or lodash functions or define other functions if you need to.**

```
/**
 * Builds a hierarchy starting at the given parent.
 * Returns an array of the non-leaf nodes in the hierarchy.
 */
public DefaultTreeDataNode[] buildHierarchy(TreeDataModel pModel, TreeDataNode pParent, String
pKeyField, String pParentKeyField) {
    OrderedHashtable lNodes = new OrderedHashtable();

    // Grab all the parents current children before we start rearranging them
    for (int i = 0; i < pParent.getChildren().length; i++) {
        TreeDataNode lChild = (TreeDataNode) pParent.getChildren()[i];
        Object       lKey   = pModel.getValue(lChild, pKeyField);

        // If the key is null, make one up since the parent key is the one that really
        // matters here. Nodes with null parent keys are children of root, so the node
        // couldn't have any children anyway.
        if (lKey == null) {
            lKey = new Object();
        }

        if (lNodes.containsKey(lKey)) {
            throw new RuntimeException("Node with non-unique key: " + lKey);
        } else {
            lNodes.put(lKey, lChild);
        }
    }

    // Return just the values of the lNodes hashtable
    Object[] lNodeArray = (Object[]) CollectionsUtil.values(lNodes, new Object[0]);
    Vector   lParents   = new Vector();

    for (int i = lNodeArray.length - 1; i >= 0; i--) {
        TreeDataNode lNode      = (TreeDataNode) lNodeArray[i];
        Object       lParentKey = pModel.getValue(lNode, pParentKeyField);
        Object       lKey       = pModel.getValue(lNode, pKeyField);

        if (lParentKey != null && lNodes.containsKey(lParentKey))
        {
            TreeDataNode lParent = (TreeDataNode) lNodes.get(lParentKey);

            pModel.insertNode(lNode, lParent);

            if (!lParents.contains(lParent)) {
                lParents.addElement(lParent);
            }
        } else {
            pModel.insertNode(lNode, pParent);
        }
```

```
    }

    return (TreeDataNode[]) lParents.toArray(new TreeDataNode[0]);
```

JavaScript:

```
/**
 * Builds a hierarchy starting at the given parent.
 * Returns an array of the non-leaf nodes in the hierarchy.
 */

function buildHierarchy(Model, Parent, KeyField, ParentKeyField) {
  var lNodes = new OrderedHashtable();

  // Grab all the parents current children before we start rearranging them
  for (var i = 0; i < Parent.getChildren().length; i++) {
    var lChild = Parent.getChildren()[i];
    var lKey = Model.getValue(KeyField);

    if (lKey === null) {
      lKey = new Object();
    }

    if (lKey in lNodes) {
      throw "Node with non-unique key: " + lKey;
    } else {
      lNodes[lKey] = lChild;
    }
  }

  // Return just the values of the lNodes hashtable
  var lNodeArray = lNodes.map(function(item) { return item.name });
  var lParents = new Array();

  for (var i = 0; i < lNodeArray.length - 1; i >= 0; i--) {
    var lNode = lNodeArray[i];
    var lParentKey = Model.getValue(lNode, ParentKeyField);
    var lKey = Model.getValue(lNode, KeyField);
    var lParent = null;

    if (lParentKey != null && lNodes.containsKey(lParentKey)) {
      lParent = lNodes.get(lParentKey);
      Model.insertNode(lNode, lParent);
      if (!lParents.contains(lParent)) {
        lParents.push(lParent);
      }
    } else {
      Model.insertNode(lNode, Parent);
    }
  }

  return lParents;
 }
```

5. **Write a Facebook React class that can be used with the code below to generate a chart that looks like:**

Also define an ‘onSelect’ event that fires when an element is clicked on that correctly identifies the index of the bar that was clicked on.

```
<script type="text/jsx">
/**
 * @jsx React.DOM
 */
function onSelect(i) {
  console.log("Selected Bar # " + i);
}

React.renderComponent(
  <BarChart values={[3, 6, 12, 4, 8]}
            width="500"
            height="300"
            onSelect={onSelect} />,


  document.getElementById('example')
);
</script>
```

BarChart component:

```
/** @jsx React.DOM */


var React = require('react');
var d3 = require('d3');
var _ = require('underscore');


var Chart = React.createClass({
  render: function() {
    return (
      <svg width={this.props.width} height={this.props.height}>{this.props.children}</svg>
    );
  }
});


var Bar = React.createClass({
  getDefaultProps: function() {
    return {
      width: 0,
      height: 0,
      offset: 0
    }
  },


  onSelect: function(event) {
    alert(Math.round(this.props.height/8.33));
  },


  render: function() {
```

```
    return (
      <rect onClick={this.onSelect} fill={this.props.color}
        width={this.props.width} height={this.props.height}
        x={this.props.offset} y={this.props.availableHeight - this.props.height} transform = "rotate(90 0 0)
translate(0 -100)"/>
    );
  }
});


var DataSeries = React.createClass({
  getDefaultProps: function() {
    return {
      title: '',
      data: []
    }
  },


  render: function() {
    var props = this.props;


    var yScale = d3.scale.linear()
      .domain([0, d3.max(this.props.data)])
      .range([0, this.props.height]);


    var xScale = d3.scale.ordinal()
      .domain(d3.range(this.props.data.length))
      .rangeRoundBands([0, this.props.width], 0.05);


    var bars = _.map(this.props.data, function(point, i) {
      return (
        <Bar
          height={yScale(point)}
          width={xScale.rangeBand()}
          offset={xScale(i)}
          availableHeight={props.height}
          color={props.color}
```

```
          key={i} />
      )
    });

    return (
      <g>{bars}</g>
    );
  }
});

var BarChart = React.createClass({
  render: function() {
    return (
      <Chart width={this.props.width} height={this.props.height}>
        <DataSeries data={[ 3, 6, 12, 4, 8 ]} width={this.props.width} height={this.props.height} color="gray" />
      </Chart>
    );
  }
});

React.renderComponent(
  <BarChart width={100} height={100} />,
  document.getElementById('container')
);
```