

ECS 89H

Beginning Web Programming

Practice Final – Spring 2016

1. There are a number of things the browser can do that the server cannot, and also a number of things the server can do that the browser cannot. Label each Javascript line below as most likely browser code, server code, or generic code that could be in either the browser or server.

- `console.log(textString);`
- `request.send(null);`
- `pList = d3.selectAll("p");`
- `response.end();`
- `this.idNum = id;`
- `node.appendChild(left);`
- `request (requestString, function (error, head, body) console.log(body));`
- `exports.makeElement = makeElement;`
- `server.listen(8080);`

Wowed by our Primary Dashboard project, the New York Times hires us to make a General Election Web graphic, in which we will display predictions for the general presidential election in November. The predictions will come from an API at the Kooks Political Report Website (I made this API up, it does not really exist). In the general election, each State elects delegates to a national Electoral College (it is a Convention, it is just called a College), and then the Electoral College elects the President.

2. Every Monday the Kooks Political Report Website issues new predictions of the vote in each state. It has an API that works like this: you send it the URL `www.kooks.com/api?states=all`, and it sends back a JSON file that starts like this:

```
[ {"AL": {"D":43, "R":57}},  
  {"AK": { "D":47, "R":53}},  
  ...  
]
```

The JSON file contains one line per state, with the predicted Democratic percent of the vote (“D”), and the predicted Republican percent of the vote (“R”).

Rather than contact Kooks every time we need this data, we will download it and keep it in a database on the NY Times server. So we need to write a node program, `refreshDB.js`, that we will run every Monday: it should do an API request, get the new data, and replace the old data in the database with the new data.

On the following pages, we have a basic structure for this program, with code in the functions missing. Please fill them in. Give careful thought to what function calls what other function.

```

// refreshDB.js - use to keep database \texttt{voteData.db} up-to-date
var http = require('http');
var request = require('request');
var sqlite3 = require("sqlite3")
requestString = "http://www.kooks.com/api?states=all";

main()

// the main function - runs when the program starts
function main() {

}

// This is the node request function that gets the API data.
// Please fill in missing contents!
function getAPIdata() {
    request (requestString, function (error, header, body) {

    })

}

// database error callback function, used below
function dbErrorFunc(error) {
    if(error)console.log(error);
}

// This function puts data into the database
// Check out the "CREATE" command for the database schema
// Be sure to use "INSERT OR REPLACE" instead of "INSERT"
function fillDB (kooksJSON) {

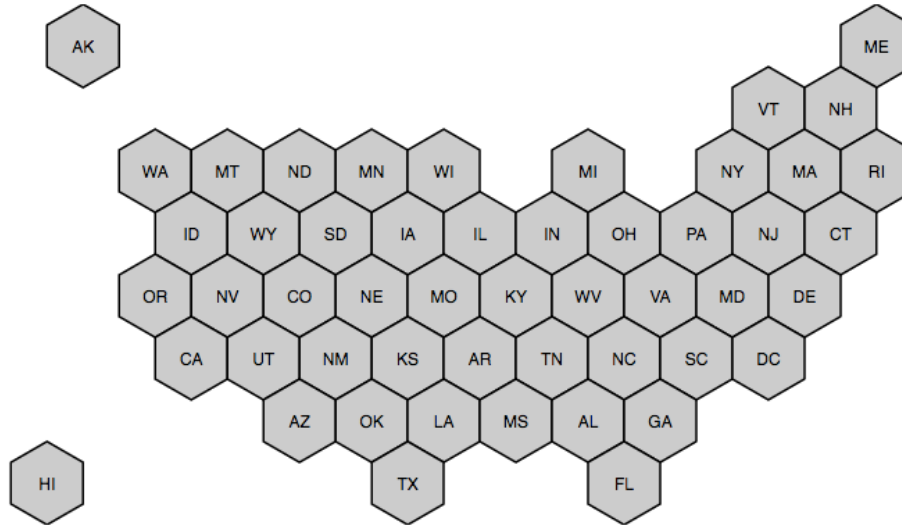
    db = new sqlite3.Database("voteData.db");
    db.serialize( function() {
        // check for table in database
        db.get("SELECT name FROM sqlite_master WHERE type='table' AND name='Votes'",
            function (error,table) {
                // create table if not there
                if (table == undefined)
                    db.run("CREATE TABLE Votes (state TEXT, dem INT, rep INT)", dbErrorFunc);
            });
        // now either insert (if not there) or replace (if there) each row of the database

    })

} // end serialize
}

```

General Election Prediction



The main display of our general election Web graphic will consist of a hex-map of the United States, in which each state is replaced with a hexagon, as in the picture above. This kind of abstract map visualization lets the user find different states using their geographic location, but makes them all the same size so we can see all the data. We plan to color each state depending on whether the Kooks API predicted the state will vote Republican (red) or Democratic (blue). We will color states that are exactly 50-50 purple.

3. Right now the function "colorHexes" is empty. It will have to do an AJAX request to the NY Times server to get the voting data we stored in the previous question, and then use it to change the colors of the hexagons. A programmer at the NY Times agrees to have their server accept queries of the form "http://www.nytimes.com/getData?hexmapData=stateColors" and respond with a JSON file containing the colors, that looks like this:

```
{ "AK": "red", "ME":"blue", "VT":"blue", "NH":"purple", ...
```

(the order of the states does not matter). On the next page is the "makeRequest" function we handed out as part of the solution to Assignment 6.

```

function makeRequest(requestString, callbackFunction) {
    // first, make a request object
    var req = new XMLHttpRequest();
    // set up callback function
    // "onreadystatechange" method is called a bunch of times
    // as the request makes its way out to the internet and back
    req.onreadystatechange = function() {
        if (req.readyState === XMLHttpRequest.DONE) {
            // the whole process is over
            if (req.status === 200) { // status 200 means all went well!
                var resp = req.response; // response shows up in the object
                callbackFunction(resp); // call the callback function
            } else {
                console.log("Problem requesting data from server");
                console.log("Response code ", req.status);
            }
        }
    }

    var reqURL = "/query?" + requestString;
    req.open('GET', reqURL, true); // load up the request string
    req.send(null); // and send it off!
}

```

a) You'll need to make one small change - one word! - in MakeRequest, to work with the NY times server. What is it?

b) The html, css, and javascript for the gray hex map follow. The function "colorHexes" is empty. In the following space, write a version of the function colorHexes, including using MakeRequest to get the data from the server.

HTML:

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" href="hexes.css"></link>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<h1>General Election Prediction</h1>

<script src="https://d3js.org/d3.v3.min.js"></script>
<script src="https://d3js.org/queue.v1.min.js"></script>
<script src="https://d3js.org/topojson.v1.min.js"></script>

<!-- adds the SVG containing the map -->
<script src="hexes.js"></script>

</body>
</html>
```

CSS:

```
body{
  font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
  background-color: #fff;
  text-align: center;
  font-size: large;
}

svg {
  background-color: #fff;
}

path {
  stroke-linejoin: round;
  stroke-linecap: round;
  fill: #ccc;
  stroke: #000;
}

text {
  text-anchor: middle;
  font-family: sans-serif;
  font-size: 8px;
  fill: #000;
}
```

Javascript:

```
// get JSON data from static server
queue()
  .defer(d3.json, "hexUSA.json")
  .await(ready);

// global var to hold the hexmap data
var geometry;

// Called when hex data arrives
function ready(error, hexArray) {
  if (error) throw error;
  geometry = hexArray;
  drawHexMap();
  colorHexes();
}

// the function you're supposed to write
function colorHexes() {}

// the important part to read
// defines the hex map using D3
function drawHexMap() {

  var width = 435,
      height = 250;

  // Add hexagon centers to data structure
  // you can just accept that this works, it is not relevant to
  // your task.
  addCenters();

  // Add the svg to the body
  var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

  // A group of elements
  var states = svg.append("g")
    .attr("id", "usHexmap");

  for (var i=0; i<geometry.length; i++) {
    // append the hexagon itself
    states.append("path")
      .attr("d", geometry[i].hex)
      .attr("id", geometry[i].label);
    // now append an SVG text node with label
    states.append("text")
      .attr("x", geometry[i].center.x)
      .attr("y", geometry[i].center.y+3)
      .text(geometry[i].label);
  } // end for loop
}
```

```

// computes hexagon centers and adds them to the "geometry" array
// not important to study this (it's not difficult, just irrelevant)
function addCenters() { for (var i=0; i<geometry.length; i++) {
    // get center of hexagon
    var cx = 0;
    var cy = 0;
    var words = geometry[i].hex.split(" ");
    for (var j=0; j<6; j++) {
        var coords = words[j].split(",");
        cx = cx+Number(coords[0].slice(1));
        cy = cy+Number(coords[1]);
    }
    cx = cx/6;
    cy = cy/6;
    geometry[i].center = {"x": cx, "y": cy};
}
}

```

4. In our main server file, we had a line:

```
var staticFileServer = new statix.Server('./public',{cache: 0});
```

One student's server contains instead the line:

```
var staticFileServer = new statix.Server('./static',{cache: 0});
```

a) How does this affect the behavior of this student's server as opposed to the one we handed out? Is it exactly the same? Does it work at all? Does the browser have to use different URLs for some requests?

b) How does this affect the rest of the implementation of the the student's project, if at all?