CIS*2750 Assignment 3 - Stub

For this assignment, and web development in general you will need to exercise your "google-fu", your skill in using a search engine to find answers to problems.

You do not need to do much backend coding at all for Module 1. app.js should just contain hard coded response values.

Installation of Node App

1. Install

```
# From the root of the directory
npm install
```

2. Running Server

```
# PORT is your personally given port number, e.g. 1234
npm run dev PORT
# Server will be accessible at http://localhost:PORT
```

Directory Structure

```
# This contains the Backend Node Server, with our Web Application and API
app.js

# These are the package configuration files for npm to install dependencies
package.json
package-lock.json

# This is the Frontend HTML file that you see when you visit the document root
public/index.html

# This is the Frontend browser JavaScript file
public/index.js

# This is the Frontend Custom Style Sheet file
public/style.css

# This is the directory for uploaded .vcf files
upload/

# This is the directory where you put all your C parser code
parser/
```

You will need to add functionality to app.js, index.html, index.js and, if you wish, style.css.

Components

Public Files, HTML, CSS, JavaScript

- These make up the "frontend", files that are directly accessed by users. They can see the source code
- To view these files from your browser, right click on something on the webpage and click "inspect element" to open your browsers "dev tools"
- Common tabs in the dev tools are:
  - Console: Displays errors, you can run JavaScript, see console.log() output
  - Inspector: See the current state of HTML on the webpage. HTML can be manipulated by JavaScript, it can update often. You can see the "raw" HTML returned from your server by right clicking the webpage and clicking "View Page Source"
  - Network: This Tab is important. It shows all your network requests, (fetching HTML, CSS, JavaScript files, and AJAX requests)
    - When you "refresh" the page, most browsers do a "soft" refresh, meaning that some assets are fetched from your "browser cache" and not directly updated. To "hard" refresh type Ctrl+Shift+R or Ctrl+Shift+F5

https://developer.mozilla.org/en-US/docs/Tools

HTML

```
# Doctype of the file, this doctype specifies HTML5
<!DOCTYPE html>

# Language English, useful for Accessibility
<html lang="en">
# The head contains metadata and script loading
<head>
# Here is some metadata about the file, you can change those values to match your own
    <title>CHANGE ME</title>
    <meta name="description" content="CHANGE ME">
    <meta name="keywords" content="CHANGE ME">
    <meta name="author" content="CHANGE ME">
    <meta charset="utf-8">

# This is useful for viewing on non-standard screens
    <meta name="viewport" content="width=device-width, initial-scale=1">

# Here we are using a Content-Distribution-Network (CDN) to get quick loading
# bootstrap, jQuery libraries. Note you need internet access to load these
# (assuming they're not cached)
    <!-- Load jquery, bootstrap -->
```

```
    <script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLN
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstr
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integr

# Here we're loading our custom Style
    <!-- Custom CSS -->
    <link rel="stylesheet" type="text/css" href="/style.css">
    <!-- Custom CSS End -->
</head>
# The body contains all of our elements, divs, tables, forms...
<body>
# A link for verifying that file download functionaity works
    <h3>Download test</h3>
    If testCalSimpleUTC.ics exists in the uploads/ directory on the server, then clicking or
    <a href="/uploads/testCalSimpleUTC.ics">testCalSimpleUTC.ics</a>
    <hr>


    <!-- Leave me at the bottom of body -->
    <script src="/index.js"></script>
</body>
</html>
```

CSS Bootstrap 4 style guide (example, forms): https://getbootstrap.com/docs/4.0/components/forms/

```
/* I've added Style to align our content in the center of the page */
html {
    width: 60%;
    height: 100%;
    min-height: 100%;
    margin: 0 auto;
    font-size: 16px;
    background-color: #FFFFFF;
}
body {
    height: 100%;
    min-height: 100%;
    line-height: 1.75;
    background-color: #FFFFFF;
}

a:link:hover {
    -webkit-transition: all 0.3s ease;
    -moz-transition: all 0.3s ease;
    transition: all 0.3s ease;
```

```
    text-decoration: underline;
    cursor: pointer;
    cursor: hand;
}
input[type=submit], button {
    cursor: pointer;
    cursor: hand;
}

/*
 * These are media queries, resizing the page for small devices
 * https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries
 * When changing font-size use "rem" units, relative to root font-size
 *   font-size: 1.5rem is 1.5X the root element (html) font-size
 *   16 * 1.5 = 24
 * When you resize your browser you'll notice everything gets smaller
 */
@media(max-width: 900px) {
    html {
        font-size: 14px !important;
        width: 80% !important;
    }
}
@media(max-width: 480px) {
    html {
        font-size: 12px !important;
        width: 90% !important;
    }
}
@media(max-width: 320px) {
    html {
        font-size: 12px !important;
        width: 100% !important;
    }
}
```

Browser JavaScript How to $.ajax: https://stackoverflow.com/a/22964077/5698848

```
// We're using jQuery library
// document.ready just means this JS runs when the document element (body) is loaded
// Put all onload AJAX calls here, and event listeners
$(document).ready(function() {

// This is an Asynchronous JavaScript Request (AJAX) using jQuery
// We can use this to query our API endpoints
    // On page-load AJAX Example
```

```
    $.ajax({
        type: 'get',            //Request type
        dataType: 'json',       //Data type - we will use JSON for almost everything
        url: '/someendpoint',   //The server endpoint we are connecting to
        success: function (data) {
            /*  Do something with returned object
            Note that what we get is an object, not a string,
            so we do not need to parse it on the server.
            JavaScript really does handle JSONs seamlessly
            */
            $('#blah').html("On page load, Received string '"+JSON.stringify(data)+"' from s
            //We write the object to the console to show that the request was successful
            console.log(data);
        },
        fail: function(error) {
            // Non-200 return, do something with error
            console.log(error);
        }
    });
// When you "submit" an HTML form, the default action is to redirect to another page
// This overrides it and allows us to make an AJAX request and edit data on the page
    $('#someform').submit(function(e){
        //Dumy output, to show that the form callback is working
        $('#blah').html("Callback from the form");
        e.preventDefault();
        $.ajax({});
    });
});
```

HTTP Web Server

`https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server`

We are using NodeJS runtime to create a simple web server

Web Application Framework (Web Application, API)

- This is the Backend, users cannot see this code

We are using ExpressJS framework for our API and routing: https://expressjs.com/en/starter/hello-world.html

An Application Programming Interface (API) is essentially just an interface, we're using to serve our set of routes for the client browser JavaScript to interact using HTTP protocol to access Backend functionality.

We're creating a RESTful API: https://restfulapi.net/

HTTP Methods to consider:

- GET: read data (nothing changes)
- POST: create data
- PUT: update data
- DELETE: delete data

In Express it's very simple to create a single "route". A route is just an endpoint you can access from your JavaScript

Here's an example in app.js:

```
// GET method route at http://localhost/
app.get('/', function (req, res) {
  res.send('GET request to the homepage')
})

// POST method route at http://localhost/create/
app.post('/create', function (req, res) {
  res.send('POST request to the /create')
})
```

More information here: https://expressjs.com/en/guide/routing.html

```
// You can also have "dynamic" route parameters that evaluate to values passed
// Assuming we hit /get/files
app.get('/get/:name', function(req , res){
  console.log(req.params.name);
  res.send({});
});
// "files"
```

- The first parameter to `app.get()` is the "route"
- The second is our *callback* function which has "request" and "response" parameters

Request contains the data sent from the Frontend JavaScript

```
// req.params can have a JSON (JavaScript native object) with the following dictionary of va
req.params = { "userId": "34", "bookId": "8989" }

// We can access each by
req.params['userId'] = "34"
```

Response is what we send back to the client after they make an AJAX call

```
// We're sending back the same JSON object that we recieved
res.send({
  userId: "34",
  bookId: "8989"
});
```

Note that `res.send()` should take a JavaScript Native Object, NOT a string *with* JSON in it

You can parse a JSON string into a native object:

```
let variableWithJSONString = '["list", "of", "array", "items"]';
const nativeObject = JSON.parse(variableWithJSONString);
# ["list", "of", "array", "items"]
```

The stub provides `app.get()` and `app.port()` routes for handling .vcf file upload/doanload requests from the browser. All .vcf files must be placed ino the uploads/ directory.

NodeJS Libraries

```
// Strict Mode
'use strict'

// This gives us direct access to C functions from our library
// Not needed in Module 1
const ffi = require('ffi-napi');

// Express App library
const express = require("express");
const app     = express();

// Path utility library
const path    = require("path");

// File Upload library
// Not needed in module 1
const fileUpload = require('express-fileupload');
app.use(fileUpload());

// File reading and manipulating library
const fs = require('fs');

// Minimization, this is to obfuscate our JavaScript
// Obfuscation and Minimization are ways to reduce payload size
// And to get scripts to clients quicker because of the smaller size
const JavaScriptObfuscator = require('javascript-obfuscator');
```

Package.json and Lock file

- You don't need to touch these files or add any external libraries, I've added the only ones you need
- Most popular programming languages have some package management system, PHP has Composer, Ruby has RubyGems, Python has Pip...

- Our package manager allows us to install libraries by simply typing `npm install PACKAGE_NAME --save` and then `npm install` to fetch the latest version
- The packages are installed to `node_modules/` due to the essence of a package manager, we don't need to include `node_modules/` in source control, because anyone can type `npm install`

How does everything work together?

1. Install the dependencies (only need to do this once) and spin up our node server
    - Note: We're using "nodemon" (instead of say `node run dev`) because it hot-reloads app.js whenever it's changed
2. View at http://localhost:PORT
3. The HTML is loaded when you visit the page and see forms, tables, content
4. The CSS is also loaded, and you'll see the page has style
5. The JavaScript file is loaded (index.js) and will run a bunch of "on load" AJAX calls to populate dropdowns, change elements
6. When buttons are clicked, more AJAX calls are made to the backend, that recieve a response update the HTML
7. An AJAX call is made from your browser, it makes an HTTP (GET, POST...) call to our web server
8. The app.js web server receives the request with the route, and request data (JSON, url parameters, files...)
9. Express looks for the route you defined, then runs the callback function you provided
10. Our callback function (for this module) should just return a hard coded JSON response
11. The AJAX call gets a response back from our server (either a 200 OK or maybe an error like a 404 not found) and either calls the "success" callback function or the "fail" function. If the success is called, "data" contains the returned JSON, and we can use it to update elements, e.g.`$('#elementId').html('<div>' + data["somekey"] + '</div>');` where there is a "div" somewhere with the "id" "elementId".