**User Information App - AJAX Server**

Starting with your previous website, create a new branch to preserve the old site.  
Your site has a form on it that acts like a search bar. When someone types into the search bar, it should retrieve a list of matching users and list them by name on the same page, similar to how the search bars on airbnb.com or hipmunk.com function.

Once the user submits the search bar, it should exhibit the same behavior as the previous assignment, i.e. display a new page with the search results.

Hints:

* use [https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global\_Objects/String/indexOf (Links to an external site.)Links to an external site.](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/String/indexOf)
* you cannot send or render a response more than once per request.
* you must find a way to capture whenever the user's input changes in the search bar. This will trigger your Ajax request to your server.

**Part 0: If you're having trouble finding matching users, solve this puzzle first:**

// given an array of values, write a function that finds the index of where the value is located, and if nothing is found, returns -1.  
// example: for ['apple', 'orange', 'pineapple']  
// 'orange' returns '1'  
// 'durian' returns '-1'  
  
// now, write a function that finds all the indexes of where the value is located and returns them in an array, and if nothing is found, returns -1  
// example: ['apple', 'orange', 'orange', 'pineapple']  
// 'orange' returns [1,2]

**Remember you are only allowed to call  "response.send" once per request!**

**Part 1: Autocomplete**  
Modify your form so that every time the user enters a key, it makes an AJAX call that populates the search results.  
Do this work in a git branch called "autocomplete". Then, merge this branch into master with a pull request.  
  
**Part 2: Bandwidth optimization**  
Modify your form again so that AJAX requests happen at most once every 300 milliseconds.  
Do this work in a git branch called "bandwidth-optimization". Then, merge this branch into master with a pull request.

Hints:

* Use [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Date/now (Links to an external site.)Links to an external site. (Links to an external site.)Links to an external site.](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now)
* Test it by opening the Network tab in Chrome's Developer Tools by checking that fewer requests are going off.

**Additional Grading Criteria**

- The app must be able to be run by simply doing "npm install". That means your dependencies must be properly listed in package.json. Otherwise Thomas will have a difficult time grading everyone's applications. For example: "npm install --save express" will update your dependencies to include the express module.

example package.json file:

{  
 "name": "user-information-app",  
 "version": "1.0.0",  
 "dependencies": {  
 "body-parser": "^1.14.2",  
 "express": "^4.13.4",  
 "pug":"2.0.0"  
 }  
}

**Teacher notes**

-Extract data on every key up from an input field using JQuery

-Send this data to the relevant (new) route on the server using $.POST or $.GET (this is using AJAX). Make Sure the url in the $.POST/$.GET and the route on your server are aligned. You have to send the data in an object: data: {suggestionKey: suggestionValue}

-Receive the data on the server, check if you can access it. Try to console.log the whole req.body if you’re having difficulties

-Search for a partial instead of an exact match by modifying your already existing search user algorithm. The algorithm should also be able to deal with multiple matches

-Send the matched users back to the client by using res.json(/\*the matched users\*/)

-Let the callback function in $.POST/$.GET modify the html based on the data it got back from the server i.e. the matched users. You could do this with using the Jquery append method in a loop.