Lab 3

Due Sep 29, 2022 by 11:59pm **Points** 100 **Submitting** a file upload **Available** after Sep 22, 2022 at 2am

CS-546 Lab 3

The purpose of this lab is to familiarize yourself with asynchronous programming in JavaScript, as well as using modules from the Node.js Package Manager (**npm (https://www.npmjs.com/)**).

For this lab, you **must** use the `async/await` keywords (not Promises). You will also be using `axios` **(https://github.com/axios/axios)** , which is a HTTP client for Node.js; you can install it with `npm i axios`.

In addition, you must have error checking for the arguments of all your functions. If an argument fails error checking, you should throw a string describing which argument was wrong, and what went wrong.

You will be creating three `.js` files: `people.js`, `companies.js` and `app.js`.

You can download the starter template here: **lab3_stub.zip (https://sit.instructure.com/courses/61549/files/10261566?wrap=1)** **(https://sit.instructure.com/courses/61549/files/10261566/download?download_frd=1)** **PLEASE NOTE: THE STUB DOES NOT INCLUDE THE PACKAGE.JSON FILE. YOU WILL NEED TO CREATE IT! DO NOT FORGET TO ADD THE START COMMAND. DO NOT ADD ANY OTHER FILE OR FOLDER APART FROM PACKAGE.JSON FILE.**

Note: Remember that the order of the keys in the objects does not matter so `{firstName: "Patrick", lastName: "Hill"}` is the same as: `{lastName: "Hill", firstName: "Patrick"}`

Network JSON Data

You will be downloading JSON files from the following GitHub Gists:

- **people.json (https://gist.githubusercontent.com/graffixnyc/448017f5cb43e0d590adb744e676f4b5/raw/495e09557914db5d2f40141aaef60113eb19bb41/people.json)**
- **companies.json (Links to an external site.) (https://gist.githubusercontent.com/graffixnyc/90b56a2abf10cfd88b2310b4a0ae3381/raw/f43962e103672e15f8ec2d5e19106e9d134e33c6/companies.json)**

For every function you write, you will download the necessary JSONs with `axios`. DO NOT just save the data into a local file, you MUST use Axios to get the data. Here is an example of how to do so:

```
async function getPeople(){
  const { data } = await axios.get('https://.../people.json')
  return data // this will be the array of people objects
}
```

Instead of making the request in every single function, remember that code reuse is key, so if you see you are making the same axios request in all of your functions, it's best to put it in a function like noted above and then calling that function in all the functions that need to get the data from whichever json file you're working with. Always do this when you see you are doing the same thing over and over again in multiple different places. It's much easier to maintain. Say if the URL of the file ever changes, then you only need to change it in one place, not 10 different places.

people.js

This file will export the following functions:

getPersonById(id)

This will return the person object for the specified id within the `people.json` array. Note: The `id` is case sensitive.

You must check:

- That the `id` parameter exists and is of proper type (string). If not, throw an error.
- If the id exists and is in the proper type but the `id` is not found in the array of people, throw a 'person not found' error.
- if the `id` parameter is just empty spaces, throw an error.

```
await getPersonById("fa36544d-bf92-4ed6-aa84-7085c6cb0440");
\\ Returns:
{id:"fa36544d-bf92-4ed6-aa84-7085c6cb0440", first_name:"Archambault", last_name:"Forestall", email:"aforestall0@usnews.com", phone_number:"702-503-4409", address:"07322 Sugar Avenue", city:"Las Vegas", state:"Nevada", postal_code:"89140", company_id:"ed37ae87-f461-42d2-bf24-8631aad856de", department:"Services", job_title:"Social Worker"}

await getPersonById(-1); \\ Throws Error
await getPersonById(1001); \\ Throws Error
```

```
await getPersonById();\\ Throws Error
await getPersonById('7989fa5e-5617-43f7-a931-46036f9dbcff');\\ Throws person not found Error
```

sameJobTitle(jobTitle)

For this function, you will return an array of people `objects` who have the same job title from `people.json`. You must return at least two people, so if there are not 2 or more people that have the same job title for the `jobTitle` provided you will throw an error.

You must check:

- That the `jobTitle` parameter exists and is of proper type (string). If not, throw an error.
- If there are not at least two people that have the same job title the `jobTitle` provided , you will throw an error.
- if `jobTitle` is just empty spaces, throw an error.
- The `jobTitle` parameter must be case in-sensitive i.e. `sameJobTitle("Help Desk Operator")` should return the same results as passing `sameJobTitle("HELP DESK OPERATOR")`

```
await sameJobTitle("Help Desk Operator");
\\ Returns:
[
{id:"71b58028-2d43-447a-9911-925d15fc5936", first_name:"Dionis", last_name:"Morson", email:"dmorsonw@newsvine.com", phone_number:"813-647-2585", ad
dress:"98753 Surrey Way", city:"Tampa", state:"Florida", postal_code:"33647", company_id:"216602a1-032f-4a0c-8e01-84e32d3d9e26", department:"Busine
ss Development", job_title:"Help Desk Operator"},
{id:"d7fdb4b4-e5d8-46be-831e-cdd3966d9da7", first_name:"Jillana", last_name:"Defries", email:"jdefriesa7@reference.com", phone_number:"570-774-058
8", address:"50 Veith Avenue", city:"Wilkes Barre", state:"Pennsylvania", postal_code:"18763", company_id:"bc50e7ff-8a3f-42a8-a99e-2fe686d0923f", d
epartment:"Training", job_title:"Help Desk Operator"},
{id:"5773c99d-655b-46b6-9655-80406cabffd0", first_name:"Barrett", last_name:"Bachs", email:"bbachsq6@parallels.com", phone_number:"516-387-4592", a
ddress:"078 Lindbergh Place", city: "Port Washington", state:"New York", postal_code:"11054", company_id:"da1c6c44-c35a-4d10-a9e6-1c816c99e0e5", de
partment:"Business Development", job_title:"Help Desk Operator"}
]

await sameJobTitle(); \\ Throws Error
await sameJobTitle("farmer"); \\ Throws Error since there are not two people with that job title
await sameJobTitle(123); \\ Throws Error
await sameJobTitle(["Help Desk Operator"]); \\ Throws Error
await sameJobTitle(true); \\ Throws Error
```

getPostalCodes(city, state)

This function will take in the city and state and it will return an array of all the postal_codes for that city and state in the data, You will sort the returned array from lowest numbered postal_code to highest numbered postal_code. The same postal_code **may** appear in the data more than once. You will return ALL for that city and state, even if it appears multiple times in the data.

Note: In the returned data the postal_code is a string, notice the elements in the example below are numbers. Your function must return them as such

- That the `city` and `state` parameters exists and are of proper type (strings). If not, throw an error.
- if `city` or `state` are just empty spaces, throw an error.
- The `city` and `state` parameters must be case in-sensitive i.e. `getPostalCodes("Austin", "Texas")` should return the same results as passing `getPostalCodes("AUSTIN", "TEXAS")`
- If there are no postal_codes for a given `city` and `state`, throw an error

```
await getPostalCodes("Salt Lake City", "Utah"); \\ Returns: [84130, 84135, 84145]
await getPostalCodes(); \\ Throws Error
await getPostalCodes(13, 25); \\ Throws Error
await getPostalCodes("Bayside", "New York"); \\ Throws Error: There are no postal_codes for the given city and state combination
```

sameCityAndState(city, state)

This function will take in the city and state and it will return an array of strings with all the people who live in that city in that state. You will show each person's first and last name as one string for each person as shown in the output below. You will sort the names in the array alphabetically by **last name** You must return at least two people, so if there are not 2 or more people that live in the same city and state you will throw an error.

You must check:

- That the `city` and `state` parameters exists and are of proper type (strings). If not, throw an error.
- If there are not at least two people that live in the city and state provided, you will throw an error.
- if `city` or `state` are just empty spaces, throw an error.
- The `city` and `state` parameters must be case in-sensitive i.e. `sameCityAndState("Austin", "Texas")` should return the same results as passing `sameCityAndState("AUSTIN", "TEXAS")`

```
await sameCityAndState("Salt Lake City", "Utah"); \\ Returns: ['Vonnie Faichney', 'Townie Sandey',  'Eolande Slafford']
await sameCityAndState(); \\ Throws Error
await sameCityAndState("    " , "      "); \\ Throws Error
await sameCityAndState(2, 29); \\ Throws Error
await sameCityAndState("Bayside", "New York"); \\ Throws Error: there are not two people who live in the same city and state
```

companies.js

This file will export the following three functions:

listEmployees(companyName)

For this function, you will return an `object` for the `companyName` provided. You will return an object that has the company data for the companyName provided, and you will also add an employees key/property to the return object that has an array of all the names ("firstName lastName") of people who work for that company, you will look up that `company_id` in `people.json.` `company_id` in people will be a company's `id` from `companies.js`. If there are no employees for the given `companyName` then just return an empty array. You will sort the employee names in the array alphabetically by **last name**

You must check:

- That `companyName` parameter exists and is of the proper type (string). If not, throw an error.
- You must check to make sure the `companyName` parameter is not just empty spaces: If it is, throw an error.
- If the company cannot be found in `companies.json` for the supplied `companyName` parameter, then throw an error.

Note: If a company has no employees, you will return the company object as is with an empty array for the `employees` key (shown below).

```
listEmployees("Yost, Harris and Cormier") Would return:
{id:"fb90892a-f7b9-4687-b497-d3b4606faddf", name:"Yost, Harris and Cormier", street_address:"71055 Sunbrook Circle", city:"Austin", state:"TX", pos
tal_code: "78715", industry:"Apparel" , employees: ["Jenda Rubens"]}

listEmployees("Kemmer-Mohr") Would return:
{id:"74f11ba3-7253-4146-b5a8-2f7139fe50bf", name:"Kemmer-Mohr", street_address:"534 Lyons Drive", city:"Cincinnati", state:"OH", postal_code: "4599
9", industry:"Industrial Machinery/Components", employees:['Janessa Arpino', 'Antoni Bottjer']}

listEmployees("Will-Harvey") Would return:
{id:"746d3cfe-c7b0-4927-ab0b-ecfaf1ef53f8", name:"Will-Harvey", street_address:"818 Russell Court", city:"Jackson", state :"MS", postal_code: "3929
6", industry:"Major Banks", employees: []}

await listEmployees('foobar') // Throws Error: No company name with foobar
await listEmployees(123) // Throws Error
```

sameIndustry(industry)

Given the `industry` provided, you will find all the companies from `companies.json` that are in the same industry and return an array of the objects

You must check:

- That `industry` parameter exists and is of the proper type (string). If not, throw an error.
- You must check to make sure the `industry` parameter is not just empty spaces: If it is, throw an error.
- If the industry cannot be found in `companies.json` for the supplied `industry` parameter, then throw an error.

```
await sameIndustry('Auto Parts:O.E.M.');
\\ Returns:

[

{id:"b0d53628-9e28-4aed-8559-b105296baf03", name:"Haag, Oberbrunner and Bins", street_address:"810 Butternut Point", city:"Hampton", state":"VA", p
ostal_code: "23668", industry:"Auto Parts:O.E.M."},

{id:"ddd9d6ec-035c-4809-9978-5117f39376b0", name:"Hayes-Barton", street_address:"27 Montana Lane", city:"Kansas City", state:"MO", postal_code: "64
187", industry:"Auto Parts:O.E.M."},

{id:"fbcae17b-481f-411b-8351-92ac66f1e3a1", name:"Schuster-Lang", street_address:"71599 Marquette Court", city:"Chicago", state:"IL", postal_code:
"60604", industry:"Auto Parts:O.E.M."},

{id:"29ac19a4-999b-4354-bc52-2ef03798c02a", name:"Tillman and Sons", street_address:"6 Hollow Ridge Trail", city:"Charleston", state:"WV", postal_c
ode: "25389", industry:"Auto Parts:O.E.M."},

{id:"b7a487a9-87a8-4c1c-a84d-ad1ba35fb52a", name:"Mertz, Blanda and Hills", street_address:"67926 Mockingbird Alley", city:"Huntington", state:"W
V", postal_code: "25770", industry:"Auto Parts:O.E.M."},

{id:"44f8ea72-24ec-44fd-b57e-8fc8053c127a", name:"Lubowitz Group", street_address:"42 Porter Hill", city:"Melbourne", state:"FL", postal_code: "329
19", industry:"Auto Parts:O.E.M."},

{id:"1ec4dade-fd59-472f-b44e-74910a5828f6", name:"Schimmel-Hickle", street_address:"67350 Derek Road", city:"Jacksonville", state:"FL", postal_cod
e: "32277", industry:"Auto Parts:O.E.M."}

]

await sameIndustry(43); \\ Throws Error
await sameIndustry(' '); \\ Throws error
await sameIndustry('Foobar Industry'); \\ Throws error No companies in that industry
await totalShares(); \\ Throws Error
```

getCompanyById(id)

This will return the Company for the specified id within the `companies.json` array. Note: The `id` is case sensitive.

You must check:

- That the `id` parameter exists and is of proper type (string). If not, throw an error.
- If the id exists and is in the proper type but the `id` is not found in the array of companies, throw a 'company not found' error.
- if the `id` parameter is just empty spaces, throw an error.

```
await getCompanyById("fb90892a-f7b9-4687-b497-d3b4606faddf");
\\ Returns:
```

```
{id:"fb90892a-f7b9-4687-b497-d3b4606faddf", name:"Yost, Harris and Cormier", street_address:"71055 Sunbrook Circle", city:"Austin", state:"TX", pos
tal_code:"78715", industry:"Apparel"}

await getCompanyById(-1); \\ Throws Error
await getCompanyById(1001); \\ Throws Error
await getCompanyById();\\ Throws Error
await getCompanyById('7989fa5e-5617-43f7-a931-46036f9dbcff');\\ Throws company not found Error
```

app.js

This file is where you will import your functions from the two other files and run test cases on your functions by calling them with various inputs. We will not use this file for grading and is only for your testing purposes to make sure:

1. Your functions in your 2 files are exporting correctly.

2. They are returning the correct output based on the input supplied (throwing errors when you're supposed to, returning the right results etc..).

Note: You will need an `async` function in your app.js file that `awaits` the calls to your function like the example below. You put all of your function calls within `main` each in its own `try/catch` block. and then you just call `main()`.

```
const people = require("./people");

async function main(){
    try{
        const peopledata = await people.getPeople();
        console.log (peopledata);
    }catch(e){
        console.log (e);
    }
}

//call main
main();
```

Requirements

1. Write each function in the specified file and export the function so that it may be used in other files.
2. Ensure to properly error check for different cases such as arguments existing and of the proper type as well as **throw (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw)** if anything is out of bounds such as invalid array index or negative numbers for different operations.
3. Submit all files (including `package.json`) in a zip with your name in the following format: `LastName_FirstName.zip`.
4. Make sure to save any npm packages you use to your `package.json.`
5. **DO NOT** submit a zip containing your `node_modules` folder.

Lab 3 Rubric

Criteria	Ratings		Pts
people.js - getPersonById Test cases used for grading will be different from assignment examples.	**8 to >0.0 pts** **1 Points/Error Handling Test Case & 2 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	8 pts
people.js - sameJobTitle Test cases used for grading will be different from assignment examples.	**12 to >0.0 pts** **1 Points/Error Handling Test Case & 2 Points/Valid Input Test Case** 4 Error Handling Test Cases & 4 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	12 pts
people.js - getPostalCodes Test cases used for grading will be different from assignment examples.	**18 to >0.0 pts** **2 Points/Error Handling Test Case & 5 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	18 pts
people.js - sameCityAndState Test cases used for grading will be different from assignment examples.	**18 to >0.0 pts** **2 Points/Error Handling Test Case & 5 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	18 pts
companies.js - listEmployees Test cases used for grading will be different from assignment examples.	**24 to >0.0 pts** **2 Points/Error Handling Test Case & 4 Points/Valid Input Test Case** 4 Error Handling Test Cases & 4 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	24 pts
companies.js - sameIndustry Test cases used for grading will be different from assignment examples.	**12 to >0.0 pts** **1 Points/Error Handling Test Case & 2 Points/Valid Input Test Case** 4 Error Handling Test Cases & 4 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	12 pts
companies.js - getCompanyById Test cases used for grading will be different from assignment examples.	**8 to >0.0 pts** **1 Points/Error Handling Test Case & 2 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	8 pts
			Total Points: 100