Introduction

This is a test task designed to evaluate your ability within JavaScript, Node.JS & React.JS. We also want to gauge your ability to research, autonomy with respect to dealing with new technical challenges in order to complete the task below.

Prerequisite Software

You will need to install some software to get you started however you can use what you think is best. Items marked with a * are required.

- **Operating System*: Windows / Linux (Ubuntu) / iOS
- **Javascript Runtime:* Node.JS 8.11.3 - https://nodejs.org/dist/v8.11.3/node-v8.11.3-x64.msi (any higher version is fine if you like. Do not use v10 and above)
- **Package Manager:* NPM https://www.npmjs.com/get-npm or Yarn https://yarnpkg.com/en/ (you must install node first in both cases)
- *Source Control:* Git (https://git-scm.com/downloads) or whatever you use to manage source code on Github/Bitbucket
- *IDE:* Atom - https://atom.io/ (you can use what you are most comfortable with i.e. Sublime Text)

Initial Setup

After unzipping the attached zip file the first step is to add these files to a GitHub repository. If you don't have a GitHub account it is free to create one. Below are the steps:

The following steps assume you are going to be using the command line. If you are not comfortable you can use a visual alternative such as `GitKracken - https://www.gitkraken.com/`

On your local machine:

1. Navigate to this folder containing the `README.md` and open a terminal within this folder.
2. Type `git init`.
3. Type `git add -A` to add all of the relevant files.
4. Type `git commit -a -m "initial commit"`.

On GitHub:

You've now got a local git repository. The next step is to connect it to GitHub. Do the following below:

1. Go to `github.com`.
2. Log in to your account.
3. Click the new repository button in the top-right. You'll have an option there to initialize the repository with a README file, but do not as we already have one.
4. Click the `Create repository` button.
5. Now, follow the second set of instructions, `Push an existing repository…`

After completing this step you should be good to go!

Recommended Reading & References

We recommend you do some reading before attempting this task if you are unsure on anything above. This is not mandatory however this task will assume certain knowledge so its important you are aware of certain commands or are familiar with JS libraries like React.JS. You of course can use any resource you like.

1. Javascript:
   `https://www.codecademy.com/learn/introduction-to-javascript`
2. React:
   `https://www.codecademy.com/learn/react-101`
   `https://www.codecademy.com/learn/react-102`
3. NPM/Yarn:
   NPM: `https://docs.npmjs.com/`
   Yarn: `https://yarnpkg.com/en/docs`
4. Github
   `https://guides.github.com/activities/hello-world/`

Installing

To install the application execute one of the below commands:

```
npm install
```

or

```
yarn install
```

Yarn is quicker than NPM but you can use what you want.

Building The Application

While you are completing this task its better to have a auto reloading server while you are working. This allows you to instantly see changes you've made OR when you break something.

To run the web server execute:

```
npm run start
```

or

```
yarn start
```

... and navigate to http://localhost:8080

Exercises

These exercises will increase with difficulty and complexity as you go on. The important thing is to understand what the web application is doing. This is party to gauge how well you can dive into the inner workings of an application to understand what's going on in order to solve a problem.

You are free to use any NPM modules you think will help you or you can solve using plain JavaScript or a mix of the two. We would like to see you use ES6 where applicable, i.e. using `let` or `const` where appropriate instead of `var`.

It would be preferable if you followed the functional programming paradigm however this isn't strictly required to complete this exercise.

Functional Programming Reading: https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0

Please separate your commits by exercise. e.g. commit "exercise1 fixed issue one" for exercise 1 and so on. You can of course choose your own naming convention...

Exercise 1 - Top menu navigation not working

Open the following files:

File 1: `src/App.jsx`
File 2: `src/Functions.js`

Currently the navigation is not working. The navigation is controlled via the function `handleOnPageNavigation` passed to the prop `onClick` on the `<Menu/>` component within [File 1]. Without changing the following line:

```
onClick={item => handleOnPageNavigation.call(this, item)}
```

Find out why navigation is not working and fix the issue. You will not need to install any helper libraries for this.

HINT: You will need to find out what the method call() is and how it behaves.

Exercise 2 - String manipulation

Open the following file:

File 1: `src/Functions.js`

The navigation options are currently shown in camelCase i.e. `task1`. Using the existing function `formatCamelCaseString` edit the contents of the functions to allow it to convert any `camelCase` text into `Sentence case` with space separating numbers and strings. So an input string `task1` would be changed to `Task 1` and so on.

HINT: You can do this using a library or you can write your own function to make the conversion.

Exercise 3 - Breadcrumb and navigation text

Open the following file:

File 1: `src/App.jsx`

Breadcrumbs (or breadcrumb trail) is a secondary navigation system that shows a user's location in a site or web app. Currently the breadcrumb is only showing `Home`, it should also show what page the current user is on. Have a look at [File 1] and find out how to get the breadcrumb to also show the current page.

HINT: Find out where the current 'page' is stored.

Exercise 4 - Generating data and displaying it within the table

Open the following file:

File 1: `src/PageOne.jsx`

This task is about allowing a user to input text into a text box. The application will then display into an auto generating table how often a single alphabetic character appears in the textbox. Currently its not working and some functions will have to be written to get it working. We recommend each subsection be completed before moving onto the next.

Please read the following page `https://ant.design/components/table/` and understand how the `<Table/>` component works. This is because we are going to be creating functions to generate data in compliance with the props schema which can be found here: `https://ant.design/components/table/#Table`

For this exercise you be relying on the web console of your choice of browser. Its recommended to write your functions incrementally, viewing the results in the console as you go instead of writing it all at once. For more information please see or find an equivalent resource:

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

Exercise 4.1 - Getting the number of occurrences of letters in text

The function `getLetterCount` has one parameter `text`. This function should take a string input, remove all non-alphabetical characters (including numbers) and return an object with each letter as a key and the number of occurrences of said letter as a value. So see the example below:

```
const exampleText = 'Lorem ipsum dolor sit amet';
const letterCount = getLetterCount(exampleText);
console.log(letterCount);

/*
The desired output:
{
  L: 1,
  o: 3,
  r: 2,
  e: 2,
  m: 3,
  i: 2,
  p: 1,
  s: 2,
  u: 1,
  d: 1,
  l: 1,
  t: 2,
  a: 1
}
*/
```

HINT: It would help to break this down into steps i.e. removing non-alphanumeric characters, changing the string into a form where you can count individual letters and so on...

Exercise 4.2 - Building the column definition

The function `getColumns` takes two parameters `text, increment`. Using the previous function (or by other means), you must find a way to generate an array of objects, with each object representing a column. Each object should look like this:

```
{
  title: `${[lowerBound]} - ${[upperBound]}`,
  dataIndex: `${[lowerBound]} - ${[upperBound]}`
}
```

The `title` refers to what will be the label of the column displayed in the table. The `dataIndex` refers to what property within a row object this column will use when rendering the table. Please refer to this link if you need further clarification: `https://ant.design/components/table/`

This syntax `'${[lowerBound]} - ${[upperBound]}'` represents the lower and upper bounds of the letter count as defined by the `increment` i.e. `20 - 30` with an `increment` of `10`. The object within the array with the highest lower bound must not exceed the letter with the highest count of occurrences. Put another way if the letter `m` has a count of `15` and the `increment` is `10` you should have three object in the array like so:

```
[
  { title: 'Letter', dataIndex: 'name' },
  { title: '0 - 10', dataIndex: '0 - 10' },
  { title: '10 - 20', dataIndex: '10 - 20' }
]
```

So for this example:

```
const exampleText = 'Lorem ipsum dolor sit amet';
const columns = getColumns(exampleText, 1);
console.log(columns);

/*
The desired output:
[
  { title: 'Letter', dataIndex: 'name' },
  { title: '0 - 1', dataIndex: '0 - 1' },
  { title: '1 - 2', dataIndex: '1 - 2' },
  { title: '2 - 3', dataIndex: '2 - 3' }
]
*/
```

You must ensure this object `{ title: 'Letter', dataIndex: 'name' }` is the first element. The `dataIndex` doesn't strictly have to be in is format `'${[lowerBound]} - ${[upperBound]}'` but you may need to parse it later depending on how you approach the problem.

HINT: Using a while() loop may be appropriate here, although they are many ways to approach this.

Exercise 4.3 - Building the row template

We now need a function which will make a template for us to create a `row` within a table for each letter. Its recommended to have a working `getColumns` function otherwise you will need to determine the shape of the row object some other way. What I mean by this is using the example above, if your generated column array looks like this:

```
[
  { title: 'Letter', dataIndex: 'name' },
  { title: '0 - 1', dataIndex: '0 - 1' },
  { title: '1 - 2', dataIndex: '1 - 2' },
  { title: '2 - 3', dataIndex: '2 - 3' }
]
*/
```

Your row template should look like:

```
{
  'name': void 0,
  '0 - 1': '-',
  '1 - 2': '-',
  '2 - 3': '-'
}
```

Just a note, `void 0` always means `undefined`.

It would be easier to know what your column array will look like before you do this, however you can approach it another way if you like.

We will be using a `-` to indicate when the occurrence of a letter does not fall into the range shown by the column. A `X` will be used to indicate when the number of occurrences fall within the range indicated by that column. For now we just want a template, we can build the row for each letter in another function abit later.

So there is a function called `getRowTemplate` which takes two parameters `text, increment`. From this you need to generate a template for each row. So for this example:

```
const exampleText = 'Lorem ipsum dolor sit amet';
const rowTemplate = getRowTemplate(exampleText, 1);
console.log(rowTemplate);

/*
The desired output:
{
  'name': void 0,
  '0 - 1': '-',
  '1 - 2': '-',
  '2 - 3': '-'
}
*/
```

HINT: As mentioned before without knowing what your column array looks like your function will be more complicated than it needs to be. Using the getColumns() function you should be able to do this in one line (or so...).

Exercise 4.4 - Building a row for each letter

Now we have most of our functions in place we now need to convert the result of function `getLetterCount` into an array of rows. The function `getLetterRow` takes three parameters `letter, count, template`. From this you should be able to generate a row for the current letter.

As mentioned before we will be using a `-` to indicate when a letters occurrences does not fall into the range shown by the column and `X` will be used when it does.

So given these inputs you should get this output:

```
const letterRow = getLetterRow('L', 57, {
  name: '-',
  '0 - 10': '-',
  '10 - 20': '-',
  '20 - 30': '-',
  '30 - 40': '-',
  '40 - 50': '-',
  '50 - 60': '-',
  '60 - 70': '-',
  '70 - 80': '-',
  '80 - 90': '-',
  '90 - 100': '-',
  '100 - 110': '-',
  '110 - 120': '-',
  '120 - 130': '-',
  '130 - 140': '-',
  '140 - 150': '-'
});
console.log(letterRow);

/*
The desired output:
{
  name: 'L',
  '0 - 10': '-',
  '10 - 20': '-',
  '20 - 30': '-',
  '30 - 40': '-',
  '40 - 50': '-',
  '50 - 60': 'X',
  '60 - 70': '-',
  '70 - 80': '-',
  '80 - 90': '-',
  '90 - 100': '-',
  '100 - 110': '-',
  '110 - 120': '-',
  '120 - 130': '-',
  '130 - 140': '-',
  '140 - 150': '-',
  key: 'L' // Note you need a unique key for each row value, it can be anything.
}
*/
```

HINT: If you have your getRowTemplate() function working as intended you only need to assign the name property and determine which range the count of occurrences of that letter appears in and assign this property. Using Object.assign() or equivalent may be useful here.

Exercise 4.5 - Building the data source for the table

At this stage we should have everything we need to get the page working as intended. There is a function called `getDataSource` which takes two parameters `text, increment`. Using the functions `getLetterCount`, `getLetterRow` and `getRowTemplate` (or via other means) build your data source array using this function. See this example:

```
const exampleText = 'Lorem ipsum dolor sit amet';
const dataSource = getDataSource(exampleText, 1);
console.log(dataSource);

/*
The desired output:
[
  { name: 'L', '0 - 1': 'X', '1 - 2': '-', key: 'L' },
  { name: 'o', '0 - 1': 'X', '1 - 2': '-', key: 'o' },
  { name: 'r', '0 - 1': 'X', '1 - 2': '-', key: 'r' },
  { name: 'e', '0 - 1': 'X', '1 - 2': '-', key: 'e' },
  { name: 'm', '0 - 1': 'X', '1 - 2': '-', key: 'm' }
]
*/
```

If you get this working you should now be able to enter text within the text box and get a count by letter in the table!

Exercise 5 - Proficiency with React

Open the following file:

File 1: `src/PageTwo.jsx`
File 2: `src/App.jsx`

This exercise is designed to test how good you are with React.JS, building user interfaces and handling API requests. For this task you will be completely free to approach this exercise however you want and add any number of properties to the state.

If you intend to use and update the state from [File 2] you must use the `updateState` function which is passed to [File 1] as the second parameter.

The Star Wars API is the world's first quantified and programmatically-accessible data source for all the data from the Star Wars canon universe! Using `antDesign` components (https://ant.design/docs/react/introduce) and any NPM libraries you want to use, you objective is to build a *stateless* react page using the star wars API (https://swapi.co/). Your page should do the following:

- Show a summary list of `people` in the star wars universe
- Allow you to click on a `person` and view detailed information about them

The only limitations is you must export your final page as a stateless component. The parameters must also remain the same:

```
export default (state, updateState) => (
  <div className="page-two">
    <p>Your page goes here</p>
  </div>
);
```

If you want to know more information on stateless components follow this link: `https://medium.com/@joshblack/stateless-components-in-react-0-14-f9798f8b992d`.

You can increase / decrease the complexity of the data shown as you like (i.e. feel free to go further than the instructions above). Please read how to use the API via this documentation (https://swapi.co/documentation). There are no limitations here, the point is to see how you approach the task, how creative you can be and the solution you come out with.

You can make as many files as you like if you want to organise your code abit better. Feel free to poke around `src/PageOne.jsx` to get an idea of how you may start.

Conclusion

Its not necessary to complete all exercises within this document, the purpose is to allow us to gauge your technical ability so don't worry if you don't get it all done within the allotted timeframe. When you are done:

1. Commit any remaining changes and files
2. Push them to your repository on GitHub
3. Send your repo link back to us along with any comments you may have

Good luck!