

Overview

¡Gracias por tu interés en Microsoft y en nuestro equipo! Queremos ayudarte a descubrir si nuestro equipo es una buena opción para ti. Para ello, a continuación, encontrarás problemas que representan el tipo de trabajo que realizamos día tras día. Trabajamos con la tecnología web más avanzada y resolvemos problemas transformando datos del servidor para crear las mejores experiencias de usuario para nuestros clientes.

Thank You for your interest in Microsoft and our team! We are excited to help you see if our team is a good fit for you! Towards helping you figure it out, here are some problems representing the kind of work we do day in and day out. We work with the latest web technology and solve problems by transforming data from the server to create the best user experiences for our customers.

We would like you to highlight your problem-solving skills and technical depth in Web technologies by solving *all the following problems*. Please feel free to program in JavaScript or TypeScript, whichever is comfortable to you. We use React in our work, and it is one of today's most popular and versatile web stacks. Hence, we would like you to use React. However, please feel free to use any libraries as you wish and any state management library and pattern of your choice.

Problem 1

Implement a function in JavaScript or TypeScript that takes an array as input with format: [id, leftChild, rightChild], and parse this array into a binary tree data structure, with the tree node interface as follows and return parsed data as JSON.

```
interface BinTreeNode {  
  id: string | number,  
  left: BinTreeNode,  
  right: BinTreeNode  
}
```

Examples:

input = ["a", ["b"], ["c"]]

Expected Output:

```
{  
  "id": "a",  
  "left": {  
    "id": "b"  
  },  
  "right": {  
    "id": "c"  
  }  
}
```

```
}  
}
```

input = [1, [2], [3, null, [5]]]

Expected Output:

```
{  
  "id": 1,  
  "left": {  
    "id": 2  
  },  
  "right": {  
    "id": 3,  
    "left": null,  
    "right": {  
      "id": 5  
    }  
  }  
}
```

Problem 2

Now that we have done the basic parsing and transformation of data, let us use it to create a web experience.

Process the input into a tree

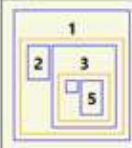
Tree source

Fetch

Tree text

```
{  
  "id": 1,  
  "left": {  
    "id": 2  
  },  
  "right": {  
    "id": 3,  
    "left": null,  
    "right": {  
      "id": 5  
    }  
  }  
}
```

Output



Let us create a Single Page Web app using React libraries and your favorite state management library with the above layout. Requirements for the web page:

- a) It allows the user to provide a file with input data like Problem 1 using the input box. When the user presses the fetch button, the app should read the data from the file and supply it as Input to the function you implemented in Problem 1. Show the Output from your function in the text area.
- b) JSON text shown in the text area should display the tree in a **visual format** in the output area. (Similar to the picture labeled as "Output").
- c) Users can now manually edit the JSON text. As the user is editing the JSON, the app may encounter parsing errors. In this case, display an error status indicator or *field validation error indicator* of your choice. As long as the input box continues to have invalid JSON, the output area should not change, and it should continue displaying the last-known-good Input. As soon as the Input becomes valid, the Output area should update and visualize the new JSON data.

Optional

You could optionally use the ready-to-run sample app skeleton we have created for you. Please download the attached sample application and extract it into a folder to get it going. You could update that code inside the sample app to solve Problem 2. You could also look at its source code to get inspiration for approaching the problem, and feel free to update it to make it better.

If you want to reuse our application, here are some pointers that could help you.

- Download the sample app from <https://aka.ms/mwtflwwebbhiringssampleapp>
- Extract the solution to an independent folder.
- It has dependencies on yarn, node, and typescript. Ensure you have them installed on your machine.
- Run `yarn && yarn start` to start the application.
- Some links would help you to understand the technologies we have used in this sample.
 - <https://reactjs.org/docs/components-and-props.html>
 - <https://reactjs.org/docs/state-and-lifecycle.html>
 - <https://reactjs.org/docs/lists-and-keys.html>
 - <https://reactjs.org/docs/thinking-in-react.html>
 - <https://mobx.js.org/intro/overview.html>
 - <https://mobx.js.org/intro/concepts.html>
 - <https://mobx.js.org/best/react.html>
 - <https://github.com/mobxjs/mobx-react#api-documentation>
 - <https://github.com/mobxjs/mobx-react#provider-and-inject>

Problem 3

In the output area shown in your solution above Problem 2, find the smallest subtree with all the deepest nodes and set its border to 2px solid green.

If a tree has only one node at depth equal to the max tree depth, then that node by itself represents the subtree that contains all the deepest nodes. If multiple nodes whose depths are equal to the max tree depth, then the solution is the smallest subtree containing all those deepest nodes.

In the accompanying README.md you make, please write down your basic assumptions and engineering tradeoffs for your interpretation of this problem and your solution.

Bonus

Be creative and add any functionality you feel will enhance the application. Please describe your enhancements and a bit of your thought process in the Readme file. For example, if you feel like you want to demonstrate your UI design instincts, you can consider improving the UI design and layout of the page to meet its purpose better and look nicer. You can also make your code scalable and performant or make it easy to distribute (and put in comments about these extra steps you undertook). If you have ideas to improve it but do not have the time to code that, add your thoughts as comments to demonstrate your thinking.

Submission

You have a couple of choices, both of which work well for us. You can upload your code to your own GitHub project or zip up your work and share it with us.

In either case, here are some requirements so that we can better understand and appreciate your work.

- Create a Readme.MD file that explains where you have your code for Problem 1. Also, please list any special steps to run your solution that displays Problem 2 and 3.
- Add any necessary comments in your code to explain your thought process.
- If you are doing a zip file, please remember to delete any node_modules folders or unnecessary files before compressing the folder for submission.
- Add a sample screenshot of your solutions for all problems and mention them in your Readme file.

Tips for ultimate success

- *Describe your code in README or doc comments.*
- *Write an enterprise-grade code keeping engineering fundamentals in mind. We want your code to be clean, well-structured with no duplications, and easy to understand.*
- *Manually test your work thoroughly. Write unit tests and describe them.*
- *Bonus answers are your opportunity to highlight your creativity, skills, and qualities – so utilize them!*