# Exercise

CSC359 Applications for the Web

24 April 2023

Suppose that we want to create an application that presents us with a table that looks like this:

| **Book** | **Author** |
|---|---|
| I, Robot | Isaac Asimov |
| Einstein's Dreams | Alan Lightman |

We want to use React to write HTML that looks like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>Favorite Books</title>
  <meta charset="utf-8" />
</head>
<body>
  <table>
    <thead>
      <tr> <th>Books</th> <th>Authors</th> </tr>
    </thead>
    <tbody>
      <tr>
        <td>I, Robot</td>
        <td>Isaac Asimov</td>
      </tr>
      <tr>
        <td>Einstein's Dreams</td>
        <td>Alan Lightman</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

Do you see how we used HTML to describe the structure of the table?

- An opening <table> tag and a closing </table> tag enclose the definition of the table.
- The table has a head.
  - The head contains labels for the table's columns.
  - An opening <thead> tag a closing </thead> ("table head") tag enclose the head.
  - An opening <tr> ("table row") tag and a closing </tr> tag enclose the list of labels.
  - An opening <th> ("table head item") tag and a closing </th> tag enclose each label.
- The table has a body.
  - An opening <body> tag and a closing </body> tag enclose the body.
  - An opening <tr> ("table row") tag and a closing </tr> tag enclose each row in the body.
  - An opening <td> ("table data") tag and a closing </td> tag enclose each item in each row.

We might begin with JavaScript statements that labels of the columns and the contents of the rows in arrays:

```
const headers = ['Books', 'Authors'];

const data = [
  ['I, Robot', 'Isaac Asimov'],
  ['Einstein's Dreams', 'Alan Lightman']
];
```

Let's first write JavaScript that will produce the HTML that describe the table's head.

We need code that changes this...

```
['Books', 'Authors'];
```

. . . to this . . .

```
[ '<th>Books</th>', '<th>Authors</th>' ]
```

Here's one way of doing that:

```
const result = [];
for (const label of headers) {
  result.push('<th>{label}</th>');
}
```

In React, we use JSX in place of HTML. (The JSX looks very much like HTML.)

Reach let's us leave out the quotation marks around '<th>label</th>'.

We can get the headers array from the props object of our component.

Let's also replace the identifier **label** with title .

Lines 20–23 in the following React application do the job.

## 03.01.table-th-loop.html

```
 1  <!DOCTYPE html>
 2  <html>
 3    <head>
 4      <title>Table</title>
 5      <meta charset="utf-8" />
 6      <link rel="stylesheet"
 7            type="text/css"
 8            href="03.table.css" />
 9    </head>
10    <body>
11      <div id="app">
12        <!-- my app renders here -->
13      </div>
14      <script src="react/react.js"></script>
15      <script src="react/react-dom.js"></script>
16      <script src="react/babel.js"></script>
17      <script type="text/babel">
18        class Excel extends React.Component {
19          render() {
20            const headers = [];
21            for (const title of this.props.headers) {
```

```
              headers.push(<th>{title}</th>);
            }
            return (
              <table>
                <thead>
                  <tr>{headers}</tr>
                </thead>
              </table>
            );
          }
        }

        const headers = ['Book', 'Author',
            'Language', 'Published', 'Sales'];

        ReactDOM.render(
          <Excel headers={headers} />,
          document.getElementById('app'),
        );
      </script>
  </body>
</html>
```

Let's improve the previous program in two ways.

- We will replace the **for** loop with a call to map().
- We will give each item in the head a unique identifying integer key.

Do a little experiment in Node to prove to yourself that this...

```
const fab4 = ['John', 'Paul', 'George', 'Ringo'];

const band = [];
for( const beatle of fab4 ) {
    band.push( '<th>' + beatle + '</th>' );
} // for
console.log( band )
```

and this...

```
const fab4 = ['John', 'Paul', 'George', 'Ringo'];

const group = fab4.map( beatle =>
    '<th>' + beatle + '</th>' );
console.log( group )
```

. . . produce the same result.

Then see how we can use the map() function to also number the members of the band.

```
const fab4 = ['John', 'Paul', 'George', 'Ringo'];

const group = fab4.map( (beatle, index) =>
    '<th key = ' + index + '>' + beatle + '</th>' );
console.log( group )
```

This next version of the application is like the previous version except for changes in lines 24–27.

## 03.04.table-th-map-key.html

```
 1  <!DOCTYPE html>
 2  <html>
 3    <head>
 4      <title>Table</title>
 5      <meta charset="utf-8" />
 6      <link rel="stylesheet"
 7            type="text/css"
 8            href="03.table.css" />
 9    </head>
10    <body>
11      <div id="app">
12        <!-- my app renders here -->
13      </div>
14      <script src="react/react.js"></script>
15      <script src="react/react-dom.js"></script>
16      <script src="react/babel.js"></script>
17      <script type="text/babel">
18        class Excel extends React.Component {
19          render() {
20            return (
21              <table>
22                <thead>
23                  <tr>
24                    {this.props.headers.map((title, idx)
25                      => { return (
26                          <th key={idx}>{title}</th>;
27                    )})}
28                  </tr>
29                </thead>
```

```
            </table>
          );
        }
      }

      const headers = ['Book', 'Author',
          'Language', 'Published', 'Sales'];

      ReactDOM.render(
        <Excel headers={headers} />,
        document.getElementById('app'),
      );
    </script>
  </body>
</html>
```

We have code that produces the head of the table. Now we want to create the body of the table.

We have code that puts the contents of the body in a two-dimensional array. That code is in lines 57–98 of this next version of the program.

We need code that will produce JSX from that array.

Let's do another little experiment to see how we might use two calls to the map() function.

This...

```
const samples = [['a', 'b'], ['c', 'd']];

const r0 = samples.map( row => '<tr>' + row + '</tr>' );
console.log( r0 )
```

...produces this output...

```
[ '<tr>a,b</tr>', '<tr>c,d</tr>' ]
```

And this...

```
const samples = [['a', 'b'], ['c', 'd']];

const r1 = samples.map( row => '<tr>' +
    row.map( item => '<td>' + item + '</td>' ) +
    '</tr>' );
console.log( r1 )
```

...produces this output...

```
[ '<tr><td>a</td>,<td>b</td></tr>',
  '<tr><td>c</td>,<td>d</td></tr>' ]
```

Examine lines 35–41 in this next version of the program. You will find code like that with which we just experimented.

Where's the difference? The code in the program, unlike the code in our little experiment, gives each element a unique, identifying integer key. But this is just like what we did for the table's head, so there's really nothing new here.

Just one more thing: Lines 48–52 contain code that specifies the types of data that we want to put into our table. The code makes use of another library. That library is PropTypes. It adds type checking to our program. Type checking will help us guard against efforts to insert the wrong kind of data into the table. Right now the data values are specified in the source code, but later we will add a means to retrieve data from the network. We cannot be sure that data we find on the network will always conform to our specifications, so a means of automatically checking types will be helpful.

Instructions for downloading this library are on page 42 of *React Up & Running*. Line 17 of our program includes the library.

## 03.06.table-th-td-prop-types.html

```
 1 <!DOCTYPE html>
 2 <html>
 3   <head>
 4     <title>Table</title>
 5     <meta charset=”utf−8” />
 6     <link rel=”stylesheet”
 7           type=”text/css”
 8           href=”03.table.css” />
 9   </head>
10   <body>
11     <div id=”app”>
12       <!−− my app renders here −−>
13     </div>
14     <script src=”react/react.js”></script>
15     <script src=”react/react−dom.js”></script>
16     <script src=”react/babel.js”></script>
17     <script src=”react/prop−types.js”></script>
18     <script type=”text/babel”>
19       class Excel extends React.Component {
20         constructor(props) {
21           super();
```

```
      this.state = {data: props.initialData};
    }
    render() {
      return (
        <table>
          <thead>
            <tr>
              {this.props.headers.map((title, idx)
                => (<th key={idx}>{title}</th>
              ))}
            </tr>
          </thead>
          <tbody>
            {this.state.data.map((row, idx) => (
              <tr key={idx}>
                {row.map((cell, idx) => (
                  <td key={idx}>{cell}</td>
                ))}
              </tr>
            ))}
          </tbody>
        </table>
      );
    }
  }

  Excel.propTypes = {
    headers: PropTypes.arrayOf(PropTypes.string),
    initialData: PropTypes.arrayOf(
        PropTypes.arrayOf(PropTypes.string)),
  };

  const headers = ['Book', 'Author',
      'Language', 'Published', 'Sales'];

  const data = [
    [
      'A Tale of Two Cities',
      'Charles Dickens',
      'English',
      '1859',
      '200 million',
    ],
    [
      'Le Petit Prince (The Little Prince)',
      'Antoine de Saint-Exupery',
          'French',
          '1943',
          '150 million',
        ],
        [
          "Harry Potter and the Philosopher's Stone",
          'J. K. Rowling',
          'English',
          '1997',
          '120 million',
        ],
        [
          'And Then There Were None',
          'Agatha Christie',
          'English',
          '1939',
          '100 million',
        ],
        [
          'Dream of the Red Chamber',
          'Cao Xueqin',
          'Chinese',
          '1791',
          '100 million',
        ],
        [
          'The Hobbit',
          'J. R. R. Tolkien',
          'English', '1937',
          '100 million'],
      ];

      ReactDOM.render(
        <Excel headers={headers} initialData={data} />,
        document.getElementById('app'),
      );
    </script>
  </body>
</html>
```

## Next steps

In the next steps, we will add a sorting option to the application.

This is also a good time to learn how to divide our code among several files. Modularity makes development, debugging, and the sharing of code much easier.

- Write a little program that sorts an array of integers first in ascending order and then in descending order.
- Write a little program that serializes a JavaScript object and then deserializes the result.
- Learn how to use JavaScript's **import** and **export** statements.
- Place the definition of the Excel component in its own file. Define an App component that contains only an instance of the Excel component.