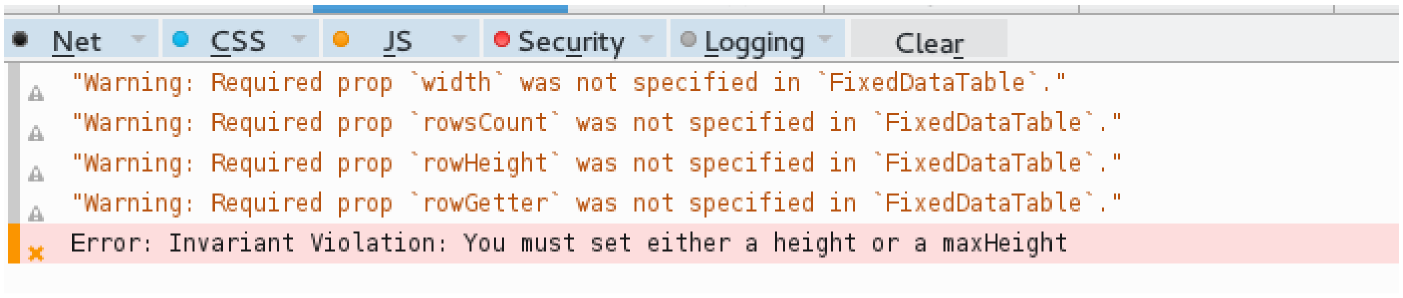
FixedDataTable React.js tutorial

29 January 2015

Recently, Facebook has released FixedDataTable , a library to build data tables with React. I tried it out to see how it stacks up against the alternatives. For reference, I have used DataTables, one of the most popular JavaScript table libraries.

Setting up the table

Creating the table is simple. You just compose and render the `Table` React component. You pass the data to the table through a `rowGetter` prop. `rowGetter` should be a function that takes a row number and returns a row. You are free to interpret the number as you wish, but the most natural thing is as an array index. The `Table` component will not render unless you also specify the table height in pixels with a `height` or `maxHeight` prop.



```
"Warning: Required prop `width` was not specified in `FixedDataTable`."
"Warning: Required prop `rowsCount` was not specified in `FixedDataTable`."
"Warning: Required prop `rowHeight` was not specified in `FixedDataTable`."
"Warning: Required prop `rowGetter` was not specified in `FixedDataTable`."
Error: Invariant Violation: You must set either a height or a maxHeight
```

.

Scrollbars appear if the table becomes taller than you specified. To configure the columns, you nest `Column` elements inside the `Table` element. Each `Column` takes a `dataKey` prop that specifies which part of the data belongs to that column. `dataKey` can be an array index or an object key.

```
React.render(
    <Table
      height={100}
      width={300}
      rowsCount={rows.length}
      rowHeight={50}
      headerHeight={40}
      rowGetter={function(rowIndex) {return rows[rowIndex]; }}>
      <Column
        dataKey="recipe"
        width={100}
        label="Recipe"/>
      <Column
        dataKey="size"
        width={100}
        label="Size"/>

       <Column  dataKey="date"
         width={100}
         label="Date"/>
    </Table>,
    document.getElementById('fixedDataTable')
);
```

Rendering

FixedDataTable renders a combination of nested `<div>`s. If you require `<table>` markup for accessibility, this library is not for you.

The table comes with default CSS in the `dist/` directory of the Github repository (https://github.com/facebook/fixed-data-table).

You can specify a custom renderer per column. A typical case would be rendering dates in a specific format:

```
<Column dataKey="date"
        width={100}
        label="Date"
        cellRenderer={function(cellData) {return new
Intl.DateTimeFormat().format(cellData); }}
```

Features

The table provides few features out of the box, but offers customization hooks instead. Setup is not as instantaneous as DataTables. While custom data rendering seems well catered for, implementing row sorting by clicking on the headers is not completely trivial.

Layout

There is no automated layout. Instead, you can supply your own logic to adapt row height according to its content. For that, you pass a function in a `rowHeightGetter` prop. It looks like the idea would be to let the application code update the table `width` and `height` props as needed.

If the table is wider than the sum of the column widths, the extra space will be distributed among the columns according to the value of the `flexGrow` prop for the `Column` element. For example, here FixedDataTable would add twice as much extra width to the second column as to each other column:

```
<Table width={400}>
  <Column width={100} flexGrow={1}/>
  <Column width={100} flexGrow={2}/>
  <Column width={100} flexGrow={1}/>
</Table>
```

This is inspired by a CSS property in the Flexbox specification.

Sorting

As mentioned, there is no way to specify a click handler for the column header, so you need a custom header renderer.

You will also need a wrapping component to hold the rows state and trigger a re-render when the rows are sorted. Look at the sample code to see how I did it.

```
<Column dataKey="recipe"
             width={100}
             flexGrow={1}
             headerRenderer={}
```

Filtering

There is no built-in filter, but you could combine the table with a custom component that updates the rows to display, like how I implemented sorting.

Conclusion

While this table does not come with many features out-of-the-box, it feels like you can customize its behaviour effectively. I like how it takes advantage of React's props system to handle the layout, as I have had bad experiences with automated layout calculations. If you already know that you desire sorting and filtering, you might want to take a look at less bare-bones components (Griddle).

More content like this?

Take a look at the forthcoming React book!

Master React

MASTER
REACT
Single-page Web Applications

Related content

- Null event properties in a React callback
- React analog clock
- Integrate jQuery UI autocomplete and React
- React without JSX, part 2
- Automate code linting with webpack and JSHint