Handling clicks on our forecast graph

We'll learn how to handle clicks on a Plotly.js graph and how to update our weather app in React.js to show the temperature of the day/time where the user has clicked.

WE'LL COVER THE FOLLOWING

Let's add one more feature to our weather application. When clicking on a specific point of our graph, we want to show the user in text the temperature at that date!

Setting up the click handler

The first thing we need to do is add an event listener to our graph. Thankfully, Plotly gives us a handy `plotly_click` event to listen to, like so:

```
// Called when a plot inside the DOM element with the id "someID" is click
ed
document.getElementById('someID').on('plotly_click', function(data) {
  /* …do something here with the data… */
});
```

The nice thing about `plotly_click` is that it doesn't pass you the event, it passes you a very useful `data` object. We care about two particular properties of that `data` object:

```
{
  "points": [{
    "x": "2016-07-29 03",
    "y": 17.4,
```

```
    /* …more data here… */


  }]
}
```

These tell us which date was clicked on and what the relevant temperature was, exactly what we want! We'll pass a function down to the `Plot` component called `onPlotClick` that will get called when the `plotly_click` event is fired, i.e. when a point on our forecast is clicked on.

Let's start off by binding that event listener in our `Plot` component. In our `drawPlot` method bind the `plotly_click` event to `this.props.onPlotClick`!

```
// Plot.js

class Plot extends React.Component {
  drawPlot = () => {
    Plotly.newPlot( /* … */ );
    document.getElementById('plot').on('plotly_click', this.props.onPlotCl
ick);
  };

  componentDidMount() { /* … */ }
  componentDidUpdate() { /* … */ }
  render() { /* … */ }
}
```

Perfect, but running this will not work since we don't pass an `onPropClick` prop to `Plot`. Let's jump to our `App` component and change that. First, we pass an `onPlotClick` prop to our `Plot` component calling our `App` component's (currently missing) `this.onPropClick` method:

```
// App.js

class App extends React.Component {
  state = { /* … */ };
  fetchData = (evt) => { /* … */ };
  changeLocation = (evt) => { /* … */ };

  render() {
    /* … */
    return (
      { /* … */ }
```

```
      <Plot
        xData={this.state.dates}

        yData={this.state.temps}
        onPlotClick={this.onPlotClick}
        type="scatter"
      />
      { /* … */ }
    );
  }
}
```

Then we add a first version of the `onPlotClick` method to our `App` component where we only log out the passed `data`:

```
import React from 'react';

class Plot extends React.Component {
  drawPlot = () => {
    Plotly.newPlot('plot', [{
      x: this.props.xData,
      y: this.props.yData,
      type: this.props.type
    }], {
      margin: {
        t: 0, r: 0, l: 30
      },
      xaxis: {
        gridcolor: 'transparent'
      }
    }, {
      displayModeBar: false
    });

    document.getElementById('plot').on(
      'plotly_click', this.props.onPlotClick);
  }

  componentDidMount() {
    this.drawPlot();
  }

  componentDidUpdate() {
    this.drawPlot();
  }

  render() {
    return (
      <div id="plot"></div>
    );
  }
}


export default Plot;
```

Now try opening your application, select a city and, when the forecast has rendered, click on a specific data point in the plot. If you see an object logged in your console containing an array called `points`, you're golden!

Updating our app state to store click data

Instead of logging the data, we now want to save that data in our state. Let's add a new object to our initial state called `selected`, which contains a `date` and a `temp` field. The date field will be an empty string by default, and the temp `null`:

```
class App extends React.Component {
  state = {
    location: '',
    data: {},
    dates: [],
    temps: [],
    selected: {
      date: '',
      temp: null
    }
  };

    /* ... Rest of the component ... */
  }
}
```

Now, when our `onPlotClick` method is called we'll set the `selected.date` to `data.points[0].x`, and the the `selected.temp` to `data.points[0].x`:

```
onPlotClick = (data) => {
    if (data.points) {
      this.setState({
        selected: {
          date: data.points[0].x,
          temp: data.points[0].y
        }
      });
    }
  };
```

Now that we have the necessary data in our state, we need to do something with it! Let's render some text saying "The current temperature on some-date is some-temperature°C!" if we have a date selected, and otherwise show the current date. We thus need to adapt the `render` method of our `App` component to include that. We check if `this.state.selected.temp` exists (i.e. isn't `null`, the default value), and if it does we render the text with `this.state.selected`:

```
    var currentTemp = 'not loaded yet';
    if (this.state.data.list) {
      currentTemp = this.state.data.list[0].main.temp;
    }
    return (
      <div>
        <h1>Weather</h1>
        <form onSubmit={this.fetchData}>
          <label>I want to know the weather for
            <input
              placeholder={"City, Country"}
              type="text"
              value={this.state.location}
              onChange={this.changeLocation}
            />
          </label>
        </form>
        {(this.state.data.list) ? (
          <div className="wrapper">
            {/* Render the current temperature if no specific date is sele
cted */}
            <p className="temp-wrapper">
              <span className="temp">
                { this.state.selected.temp ? this.state.selected.temp : cu
rrentTemp }
              </span>
              <span className="temp-symbol">°C</span>
              <span className="temp-date">
                { this.state.selected.temp ? this.state.selected.dat
e : ''}
              </span>
            </p>
            <h2>Forecast</h2>
            <Plot
```

```
              xData={this.state.dates}
              yData={this.state.temps}

              onPlotClick={this.onPlotClick}
              type="scatter"
            />
          </div>
        ) : null}

      </div>
    );
```

There is one small user experience improvement we could do. When switching to a new city, the text persists because `this.state.selected.temp` still references the old data—in reality want to show the current temperature though!

To fix this, we set `selected` back to the default values in our `fetchData` method when the request has returned data:

```
fetchData = (evt) => {
    /* ... Rest of the code */
  self.setState({
        data: body,
        dates: dates,
        temps: temps,
        selected: {
          date: '',
          temp: n
        }
      }}
```

Let's look at our app again.

```
import React from 'react';

class Plot extends React.Component {
  drawPlot = () => {
    Plotly.newPlot('plot', [{
      x: this.props.xData,
      y: this.props.yData,
      type: this.props.type
    }], {
      margin: {
        t: 0, r: 0, l: 30
      },
      xaxis: {
        gridcolor: 'transparent'
```

```
      }
    }, {
      displayModeBar: false
    });

    document.getElementById('plot').on(
      'plotly_click', this.props.onPlotClick);
  }

  componentDidMount() {
    this.drawPlot();
  }

  componentDidUpdate() {
    this.drawPlot();
  }

  render() {
    return (
      <div id="plot"></div>
    );
  }
}


export default Plot;
```

Summary

We've created a new `Plot` component, shaped the data we get from the OpenWeatherMap API to suit our needs and used Plotly.js to render a beautiful and interactive 5 day weather forecast!

Additional Material

- Official plotly.js docs
- OpenWeatherMap API
- JavaScript Graphing Library Comparison