Creating forecast graph in weather app

We'll use our knowledge of React.js and Plotly.js to create a plot component that will show the weather forecast.

Now that we've our Plot component ready, we just need to pass in the weather data and we are done.

The problem is that we need the data for the x-axis and the y-axis to be separate, but the data we get from the OpenWeatherMap API doesn't make that distinction. This means we need to shape our data a little bit to suit our needs. What we want is human readable dates on the x-axis, and the degrees at that time on the y-axis!

We'll shape the data into a form we can use it in and save both the raw and the formed data in our component state. Remember, this is what the raw data looks like:

```
"city": {
  "id": 2761369,
  "name": "Vienna",
  "coord": {
    "lon": 16.37208,
    "lat": 48.208488
  },
  "country": "AT",
  "population": 0,
  "sys": {
    "population": 0
  }
},
"cod": "200",
"message": 0.0046,
"cnt": 40,
"list": [ /* Hundreds of objects here */ ]
```

With the `list` array containing objects of this form:

```
{
  "dt_txt": "2016-04-09 18:00:00",

  "main": {
    "temp": 6.94,
    "temp_min": 6.4,
    "temp_max": 6.94
  },
  "weather": [
    {
      "main": "Rain",
      /* …more data here */
    }
  ],
  /* …more data here */
}
```

What we really care about is `data.list[element].dt_txt`, a human-readable timestamp, and `data.list[element].main.temp`, the temperature at that time.

We'll loop through all the weather information we have, making two arrays and fill one with the timestamps, and another array with the temperatures. *(We'll use the `push` method of arrays, which adds an element to the end of an array).*

We'll do that in the `fetchData` method, so we only recalculate the data when new one comes in and not on every render. (which would possibly mean shaping the data every second or more!) This is what happens when the data comes back in at the moment

Our `xhr` call to OpenWeatherMap will now look like this:

```
xhr({
      url: url
    }, () => (err, data) {
      var body = JSON.parse(data.body);
      var list = body.list;
      var dates = [];
      var temps = [];
      for (var i = 0; i < list.length; i++) {
        dates.push(list[i].dt_txt);
        temps.push(list[i].main.temp);
      }

      this.setState({
```

```
        data: body

      });
    });
```

We also need to add those new properties to our initial state:

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

  fetchData = (evt) => { /* … */ };

  changeLocation = (evt) => { /* … */ };

  render() { /* … */ }
}
```

Now that we have that data saved in our component state, we can render our plot by passing `this.state.dates` as the x-axis data, `this.state.temps` as the y-axis data and we'll also pass it a `type` prop of `"scatter"`!

Here's the working example (*reminder: you will need to update your OpenWeatherMap API key to make it work*):

```
import React from 'react';
import './App.css';
import xhr from 'xhr';

import Plot from './Plot.js';

const API_KEY = "82f40c24bce69950c7aa3d09e07b391b";

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

  fetchData = (evt) => {
    evt.preventDefault();

    if (!API_KEY) {
```

```
  if (!API_KEY) {
    console.log('Enter your API_KEY and the enter location');
    return;
  }

  let location = encodeURIComponent(this.state.location);
  let urlPrefix = '/cors/http://api.openweathermap.org/data/2.5/forecast?q=';
  let urlSuffix = '&APPID=' + API_KEY + '&units=metric';
  let url = urlPrefix + location + urlSuffix;

  xhr({
    url: url
  }, (err, data) => {
    if (err) {
      console.log('Error:', err);
      return;
    }
    var body = JSON.parse(data.body);
    var list = body.list;
    var dates = [];
    var temps = [];
    for (var i = 0; i < list.length; i++) {
      dates.push(list[i].dt_txt);
      temps.push(list[i].main.temp);
    }

    this.setState({
      data: body,
      dates: dates,
      temps: temps
    });
  });
};

changeLocation = (evt) => {
  this.setState({
    location: evt.target.value
  });
};

render() {
  let currentTemp = 'Specify a location';
  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>
      <p className="temp-wrapper">
        <span className="temp">{ currentTemp }</span>
        <span className="temp-symbol">°C</span>
      </p>
      <h2>Forecast</h2>
```

```
        <Plot
                xData={this.state.dates}
          yData={this.state.temps}
          type="scatter"
        />
      </div>
    );
  }
}

export default App;
```

If you tried playing with the example, you would have notiecd that there is one problem: When we change the city and refetch data, the graph doesn't update. This is the case because we're solely using the `componentDidMount` lifecycle method, which is only ever called once when the component mounts. We also need to draw the plot again when new data comes in, i.e. when the component did update! (*hinthint*)

As you might have guessed, we can use the `componentDidUpdate` lifecycle method of our `Plot` component to fix this. However, instead of copy and pasting the `Plotly.newPlot` call (which is identical) in both methods, we should factor that out into a `drawPlot` method and call `this.drawPlot` from `componentDidMount/Update`.

Here's the working example with the updated code (*reminder again: you will need to update your OpenWeatherMap API key to make it work*):

```
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
    });
  }

  componentDidMount() {
    this.drawPlot();
  }
```

```
  componentDidUpdate() {
    this.drawPlot();
  }

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


export default Plot;
```

Now try it! You'll see a beautiful 5 day weather forecast rendered like this. Normally, creating a graph like this manually would take ages, but Plotly.js makes it incredibly easy!

Plotly.js also provides many other features like handling clicks on the graphs. I think you might have guessed where this is going. We are going to show the temperature of the day/time where user clicks on the graph.