1. **Graphs**

The parts of the trend class include 3 graphs which each have 3 components associated with them

+Data

+Labels

+and a Name

-There are 3 types of graphs you can render, pie chart, bar graph, and line graph. These all share the same format and contain these three parts, but the format slightly differs by graph.

**Parts**

These two are universal across all graphs:

-Name is what you would expect, just a string inputted and saved

Labels are an array of strings representing the x axis. Ex. [‘January’, ‘February’, etc.]

Data is much trickier as there is a specific format to it, but at the end of the day, it’s still just an array.

-Data is a bit different as the format changes based on the graph.

Data is labeled as ‘datasets’ in the context of chart.js and react which does mean that you can have multiple datasets rendered at the same time. The format for a datasets is as follows for the different graphs.

**Chart**

[

{

label: '# of Votes',

data: [12, 19, 3, 5, 2, 3],

backgroundColor: [

'rgba(255, 99, 132, 0.2)',

'rgba(54, 162, 235, 0.2)',

'rgba(255, 206, 86, 0.2)',

'rgba(75, 192, 192, 0.2)',

'rgba(153, 102, 255, 0.2)',

'rgba(255, 159, 64, 0.2)',

],

borderColor: [

'rgba(255, 99, 132, 1)',

'rgba(54, 162, 235, 1)',

'rgba(255, 206, 86, 1)',

'rgba(75, 192, 192, 1)',

'rgba(153, 102, 255, 1)',

'rgba(255, 159, 64, 1)',

],

borderWidth: 1,

},

],

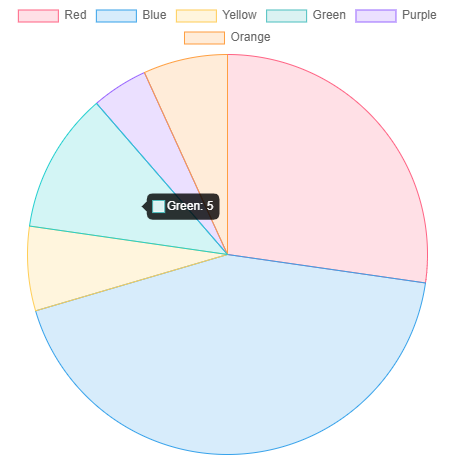
Note that this is just test data and these can easily be swapped out, but

“label” is associated with each section,

data is in the form of numbers.

Background color is in rgb format and is unessential, but feel free to color them as you see fit. Borderwidth is the distance between each section in pixels roughly.

This test dataset renders like this…



**Line**

[

{

label: 'Dataset 1',

data: labels.map(() => faker.datatype.number({ min: -1000, max: 1000 })),

borderColor: 'rgb(255, 99, 132)',

backgroundColor: 'rgba(255, 99, 132, 0.5)',

},

{

label: 'Dataset 2',

data: labels.map(() => faker.datatype.number({ min: -1000, max: 1000 })),

borderColor: 'rgb(53, 162, 235)',

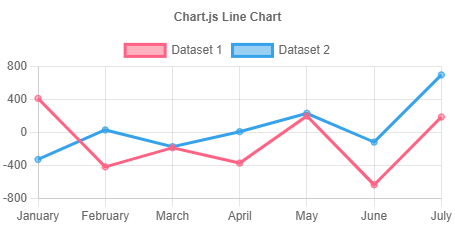
backgroundColor: 'rgba(53, 162, 235, 0.5)',

},

],

Another test dataset, but the format is very similar. In this case, you only use 1 color for each dataset and the colors once again aren’t necessary. The data is also in the same format, being an array of numbers, though this example uses a randomizer to get its data.

Here’s how it should look.



**Bar**

We can actually use the same format for line in this case, meaning we can use. datasets: [

{

label: 'Dataset 1',

data: labels.map(() => faker.datatype.number({ min: 0, max: 1000 })),

backgroundColor: 'rgba(255, 99, 132, 0.5)',

},

{

label: 'Dataset 2',

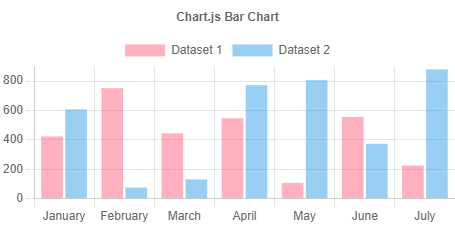
data: labels.map(() => faker.datatype.number({ min: 0, max: 1000 })),

backgroundColor: 'rgba(53, 162, 235, 0.5)',

},

],

And it will render as



The thing to keep in mind with all of these and how they are rendered is a couple of things they have in common.

1. A label per dataset
2. An array of numerical values for data
3. Background color if desired.

When using the trend class, you need to input data into the trend in the format according to its graph. Speaking of which, let’s talk about the trend class.

1. **Methods**

In the trend class, there are the 3 graphs as mentioned and not much else besides the setters and getters.

// Create functions

CreatePieChart(newlabels, newdata, newname)

CreateLineGraph(newlabels, newdata, newname)

CreateBarGraph(newlabels, newdata, newname)

// Update functions

UpdatePieChart(newpielabels, newpiedata)

UpdateLineGraph(newgraphlabels, newgraphdata)

UpdateBarGraph(newbarlabels, newbardata)

These are your setters and getters for the different graphs. Though js doesn’t have type checking, the graphs are expected to receive the formatted data as described.

In addition, you can adjust the width and height of the graph displayed. This effects all graphs and can be done in a setter and getter for each of the attributes

GetHeight(), SetHeight(newheight), GetWidth(), SetWidth(newwidth)

These functions will set the graph’s height and width in pixels for when it is rendered.

1. **Rendering**

**Normal rendering**

To render your graph, you just have to call the trend class’s render() function and input the graph you want to render

BarGraph => input ‘bargraph’

LineGraph => input ‘linegraph’

PieChart => input ‘piechart’

**Testing**

If you want to test the rendering in general, there are other inputs you can enter as well that should display the previously mentioned graphs and charts.

TestBarGraph => input ‘dummybar’

TestLineGraph => input ‘dummyline’

TestPieChart => input ‘dummypie’