InsideSherpa

JPMorgan Chase Software Engineering Virtual Experience

Task 3 - software engineering task : code changes

Module 3 - Display data visually for traders

Disclaimer

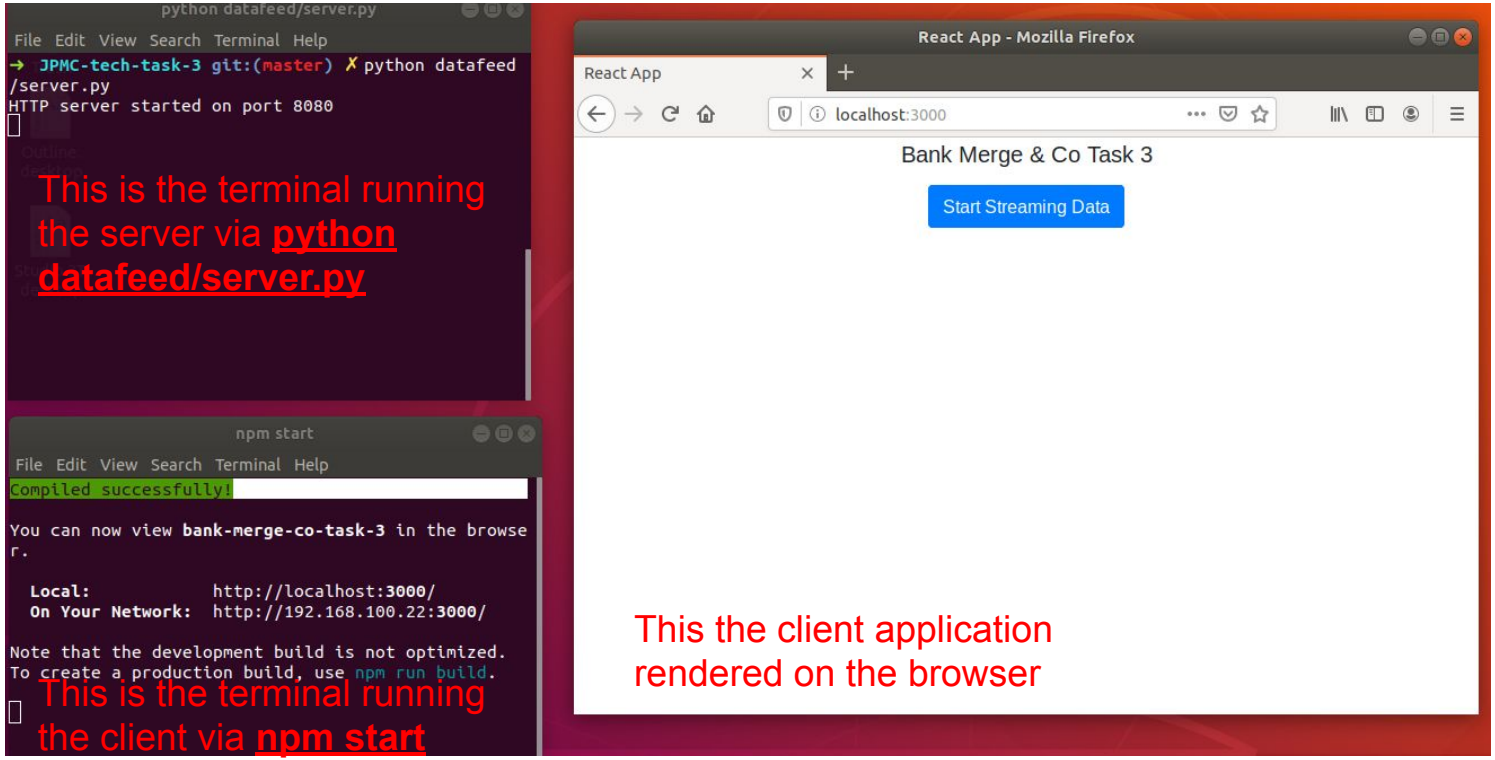
- This guide is only for those who did the setup locally on their machines.

Prerequisite

- Set up should have been done. This means, your server and client applications should have been running with no problems without introducing any changes to the code yet. You can verify this if you get a similar result to any of the following slides that include a picture of the server and client app running together

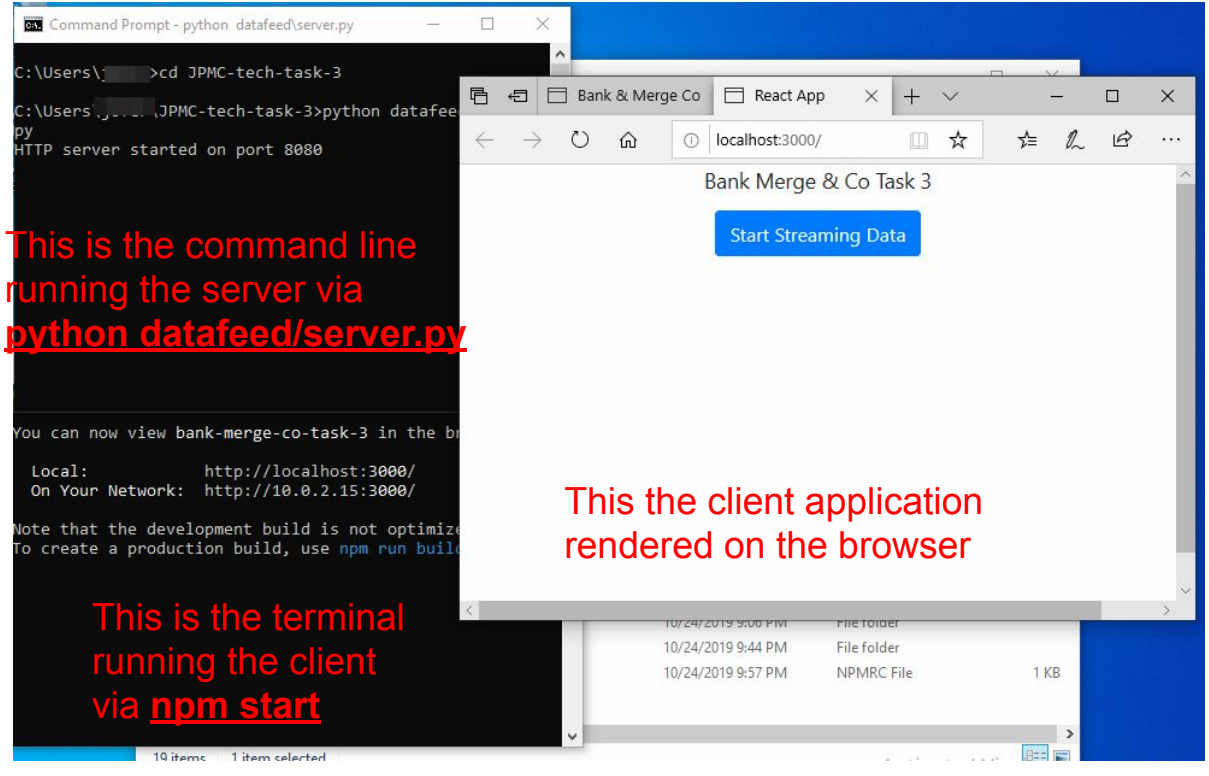
Prerequisite

Mac OS / Linux OS

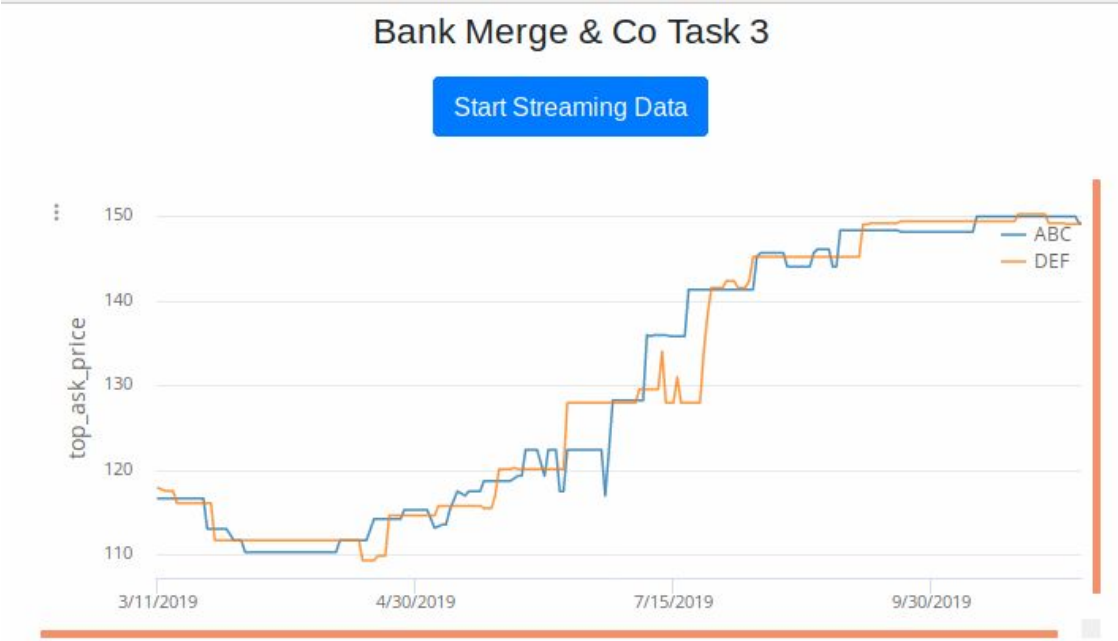


Prerequisite

Windows OS



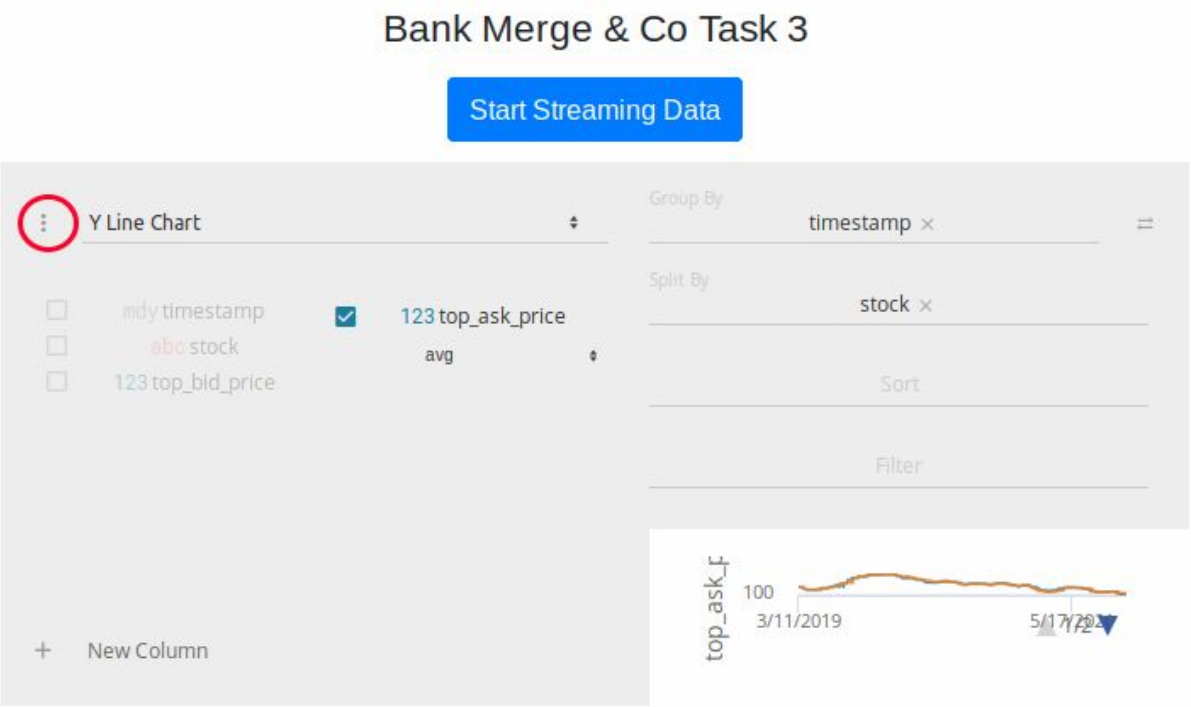
Observe Initial State Of Client App in Browser



This is how the initial state of the client app looks like when you click the blue “**Start Streaming Data”** button

It’s pretty much like the state of task 2 when you’ve finished it. You have two stocks displayed and their top_ask_price changes being tracked through a timeline

Observe Initial State Of Client App in Browser



If you clicked on the 3-dotted button on the upper left corner of the graph you'll see something like image on this slide.

This tells you that the graph is configurable.

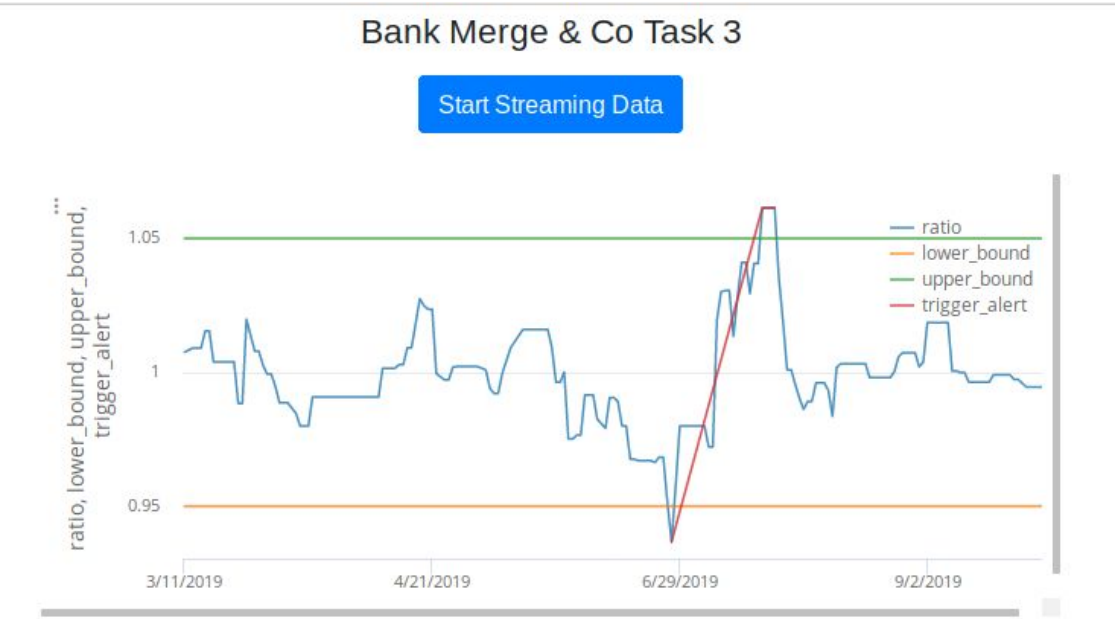
You should know this too by now if you've finished task 2 beforehand

Objectives

- There are two things we have to achieve here to complete this task
  - (1) We now want to make this graph more useful to traders by making it track the **ratio** between two stocks over time and NOT the two stocks' top_ask_price over time.
  - (2) As mentioned before, traders want to monitor the ratio of two stocks against a historical correlation with **upper and lower thresholds/bounds**. This can help them determine a trading opportunity.That said, we also want to make this graph plot those upper and lower thresholds and show when they get crossed by the ratio of the stock

Objectives

- In the end we want to achieve a graph that looks something like this



Objectives

- To achieve this we have to change (2) files: **src/Graph.tsx** and **src/DataManipulator.ts**

- Don’t worry we’ll walk you through how to get these things done

Making changes in `**Graph.tsx**`

- To accomplish our first objective, we must first make changes to the `**Graph.tsx**` file. Recall, this is the file that takes care of how the Graph component of our App will be rendered and react to the state changes that occur within the App.

- We’re not starting from a static graph anymore and we’re basically jumping off from where we’ve finished in task 2. So the changes we’ll be making here aren’t going to be as much. What we want to do here is to have one main line tracking the ratio of two stocks and be able to plot upper and lower bounds too.

Making changes in `**Graph.tsx**`

- To do this, we need to modify `**componentDidMount**` method. Recall, the **componentDidMount()** method runs after the component output has been rendered to the DOM. If you want to learn more about it and other lifecycle methods/parts of react components, read more here.

- In this method, we first have to modify the **schema** object as that will dictate how we'll be able to configure the Perspective table view of our graph. In the next slide we'll show how we want to change this exactly

Making changes in `**Graph.tsx**`

```
const schema = {
  stock: 'string',
  top_ask_price: 'float',
  top_bid_price: 'float',
  timestamp: 'date',
};
```

Before

```
const schema = {
  price_abc: 'float',
  price_def: 'float',
  ratio: 'float',
  timestamp: 'date',
  upper_bound: 'float',
  lower_bound: 'float',
  trigger_alert: 'float',
};
```

After

- Notice the changes we've made to schema:
  - Since we don't want to distinguish between two stocks now, but instead want to track their ratios, we made sure to add the **ratio** field. Since we also wanted to track **upper_bound**, **lower_bound,** and the moment when these bounds are crossed i.e. **trigger_alert**, we had to add those fields too.
  - Finally, the reason we added **price_abc** and **price_def** is just because these were necessary to get the **ratio** as you will see later**.** We won't be configuring the graph to show them anyway.
  - Of course since we're tracking all of this with respect to time, **timestamp** is going to be there

Making changes in `**Graph.tsx**`

- Next, to configure our graph you will need to modify/add more attributes to the element. We've done this before in task 2 so it should be slightly more familiar to you now. (if you forgot you can reread the doc on [Perspective configurations particularly on the table.view configurations]()) The change should look something like:

```
  elem.load(this.table);
  elem.setAttribute('view', 'y_line');
  elem.setAttribute('column-pivots', '["stock"]');
  elem.setAttribute('row-pivots', '["timestamp"]');
  elem.setAttribute('columns', '["top_ask_price"]');
  elem.setAttribute('aggregates', JSON.stringify({
    stock: 'distinctcount',
    top_ask_price: 'avg',
    top_bid_price: 'avg',
    timestamp: 'distinct count',
  }));
}
```

Before

```
  elem.load(this.table);
  elem.setAttribute('view', 'y_line');
  elem.setAttribute('row-pivots', '["timestamp"]');
  elem.setAttribute('columns', '["ratio", "lower_bound", "upper_bound", "trigger_alert"]');
  elem.setAttribute('aggregates', JSON.stringify({
    price_abc: 'avg',
    price_def: 'avg',
    ratio: 'avg',
    timestamp: 'distinct count',
    upper_bound: 'avg',
    lower_bound: 'avg',
    trigger_alert: 'avg',
  }));
}
```

After

Making changes in `**Graph.tsx**`

- ‘**view**’ is the the kind of graph we wanted to visualize the data as. Initially, this is already set to **y_line**. This is the type of graph we want so we’re good here.

- **‘column-pivots’** used to exist and was what allowed us to distinguish / split stock ABC with DEF back in task 2. We removed this because we’re concerned about the ratios between two stocks and not their separate prices

- ‘**row-pivots’** takes care of our x-axis. This allows us to map each datapoint based on the timestamp it has. Without this, the x-axis is blank. So this field and its value remains

Making changes in `**Graph.tsx**`

- ‘**columns’** is what will allow us to only focus on a particular part of a datapoint’s data along the y-axis. Without this, the graph will plot all the fields and values of each datapoint and it will be a lot of noise. For this case, we want to track **ratio, lower_bound, upper_bound** and **trigger_alert**.

- **‘aggregates’** is what will allow us to handle the cases of duplicated data we observed way back in task 2 and consolidate them as just one data point. In our case we only want to consider a data point unique if it has a timestamp. Otherwise, we will average out the all the values of the other non-unique fields these ‘similar’ datapoints before treating them as one (e.g. ratio, price_abc, …)

Making changes in `**Graph.tsx**`

- Finally, we have to make a slight update in the **componentDidUpdate** method. This method is another [component lifecycle method] that gets executed whenever the component updates, i.e. when the graph gets updated in our case. The change we want to make is on the argument we put in **[this.table.update]**. This is how it's supposed to look like after the change:

```
55    componentDidUpdate() {
56      if (this.table) {
57        this.table.update([
58          DataManipulator.generateRow(this.props.data),
59        ]);
60      }
61    }
```

There's a reason why we did this change but you'll understand it in the next couple of slides, based on our changes in **DataManipulator.ts**

Making changes in `**DataManipulator.ts**`

- To fully achieve our goal in this task, we have to make some modifications in the **DataManipulator.ts** file. This file will be responsible for processing the raw stock data we've received from the server before it throws it back to the Graph component's table to render. Initially, it's not really doing any processing hence we were able to keep the status quo from the finished product in task 2

- The first thing we have to modify in this file is the **Row** interface. If you notice, the initial setting of the **Row** interface is almost the same as the old **schema** in **Graph.tsx** before we updated it. So now, we have to update it to match the new **schema**. See next slide to better visualize the change that's supposed to happen.

Making changes in `**DataManipulator.ts**`

```
3   export interface Row {
4     stock: string,
5     top_ask_price: number,
6     timestamp: Date,
7   }
```

Before

```
4   export interface Row {
5     price_abc: number,
6     price_def: number,
7     ratio: number,
8     timestamp: Date,
9     upper_bound: number,
10    lower_bound: number,
11    trigger_alert: number | undefined,
12  }
```

After

- This change is necessary because it will be the structure of the return object of the only function of the DataManipulator class, i.e. the **generateRow** function
- It's important that the return object corresponds to the the **schema** of the table we'll be updating in the Graph component because that's the only way that we'll be able to display the right output we want.

note: Interfaces help define the values a certain entity must have. If you want to learn more about interfaces in Typescript you can read this material in your spare time

Making changes in `**DataManipulator.ts**`

- Finally, we have to update the **generateRow** function of the DataManipulator class to properly process the raw server data passed to it so that it can return the processed data which will be rendered by the Graph component's table.

- Here we can compute for **price_abc** and **price_def** properly (like what you did back in task 1). Afterwards we can also compute for **ratio** using the two computed prices, (like what you did in task 1 too). And, set **lower** and **upper** bounds, as well as **trigger_alert**. To better understand this see the expected change in the next slide

Making changes in `**DataManipulator.ts**`

```
15  export class DataManipulator {
16    static generateRow(serverRespond: ServerRespond[]): Row {
17      const priceABC = (serverRespond[0].top_ask.price + serverRespond[0].top_bid.price) / 2;
18      const priceDEF = (serverRespond[1].top_ask.price + serverRespond[1].top_bid.price) / 2;
19      const ratio = priceABC / priceDEF;
20      const upperBound = 1 + 0.05;
21      const lowerBound = 1 - 0.05;
22      return {
23        price_abc: priceABC,
24        price_def: priceDEF,
25        ratio,
26        timestamp: serverRespond[0].timestamp > serverRespond[1].timestamp ?
27          serverRespond[0].timestamp : serverRespond[1].timestamp,
28        upper_bound: upperBound,
29        lower_bound: lowerBound,
30        trigger_alert: (ratio > upperBound || ratio < lowerBound) ? ratio : undefined,
31      };
32    }
```

Feel free to change this to +/-10% of the 12 month historical average ratio

This was just for a test value

Making changes in `**DataManipulator.ts**`

- Observe how we're able to access **serverRespond** as an array where in the first element (0-index) is about stock ABC and the second element (1-index) is about stock DEF. With this, we were able to easily just plug in values to the formulas we used back in task 1 to compute for prices and ratio properly

- Also note how the return value is changed from an array of **Row objects** to just a single **Row object** This change explains why we also adjusted the argument we passed to **table.update** in Graph.tsx earlier so that consistency is preserved.

Making changes in `**DataManipulator.ts**`

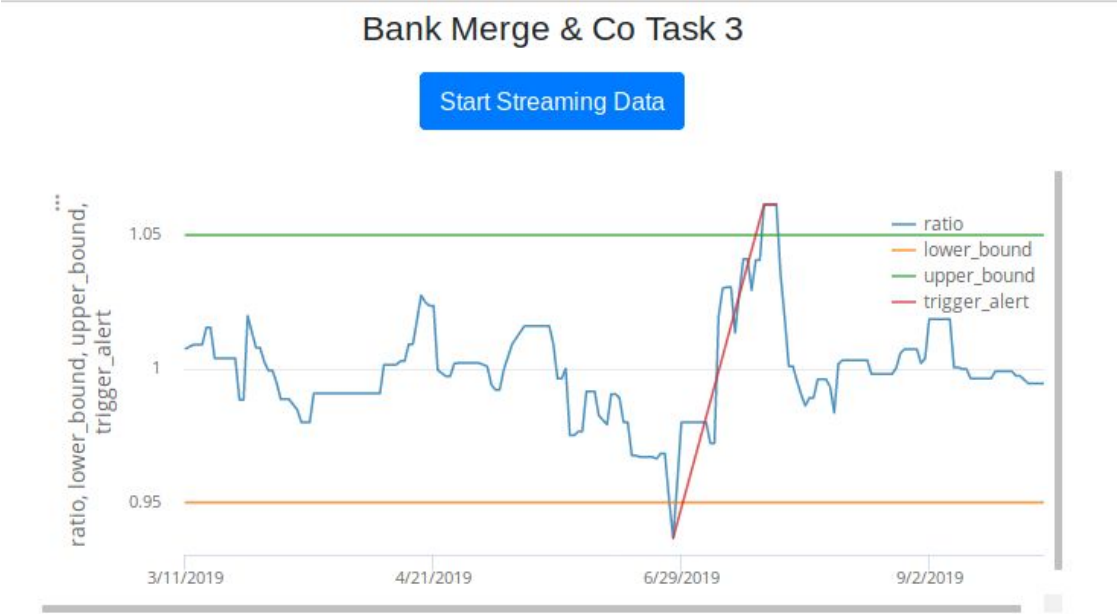
- The **upper_bound** and **lower_bound** are pretty much constant for any data point. This is how we will be able to maintain them as steady upper and lower lines in the graph. While 1.05 and 0.95 isn't really +/-10% of the 12 month historical average ratio (i.e. 1.1 and 0.99) you're free to play around with the values and see which has a more conservative alerting behavior.

- The **trigger_alert** field is pretty much just a field that has a value (e.g. the ratio) if the threshold is passed by the ratio. Otherwise if the ratio remains within the threshold, then no value/undefined will suffice.

Wrapping up

- Changes in **Graph.tsx** and **DataManipulator.ts** are done.
- By now you should've accomplished all the objectives of the task
- Feel free to poke around before completely saving everything and creating your patch file, e.g. see the different effects of changing the configurations would do to your table/graph.
- Please don't forget to leave comments in your code especially at the places where the fixes for bugs and where you piped in the data feed - this will help with other team member's understanding of your work.

End Result


Bank Merge & Co Task 3
Start Streaming Data
ratio, lower_bound, upper_bound, trigger_alert
1.05
1
0.95
3/11/2019
4/21/2019
6/29/2019
9/2/2019
ratio
lower_bound
upper_bound
trigger_alert