Assignment #4 Documentation

Yuqi Zhou

13 Aprile

Contents

1 logstash.conf

1) Strip whitespace from field ”city”.

```
filter {
  mutate { remove_field => [ "@timestamp", "@version", ...
      "executionTime", "port", "host" ] }
  mutate { strip => [ "city" ]  }
 }
```

2 server.js

1) Created new query commands to query seven days data from index ’divvy_stations_logs’. The divvy station is specified by station’s id. Field ’lastCommunicationTime’ is mapped to type text, so the array ’lastSevenDayDateInString[]’ and the query_string command are used to extract last seven days data from the index.

```
esClient.search({
     index: 'divvy_stations_logs',
     scroll: '10s',
     body: {
        "query": {
         "bool": {
           "filter": [
            { "term": { "id": id } },
                {
                    "query_string": {
                    "default_field": "lastCommunicationTime", ...
                        "query": lastSevenDayDateInString[0] + " ...
                        OR "
                    + lastSevenDayDateInString[1] + " OR " + ...
                        lastSevenDayDateInString[2] + " OR " + ...
                        lastSevenDayDateInString[3] +
                    " OR " + lastSevenDayDateInString[4] + " OR ...
                        " + lastSevenDayDateInString[5] + " OR " ...
                        + lastSevenDayDateInString[6]
                    + " OR " + lastSevenDayDateInString[7]
                        }
                    }
                ]
            }
        }
    }
}
```

2) ElasticSearch query command can query no more than 10000 size data every time, so scroll is used to query index in a loop until the size of data is equal to response.hits.total.

```
,function getMoreUntilDone(error, response) {
            if (error) {
                return reject(error);
            }
            // collect all the records
            response.hits.hits.forEach((hit) => {
                plainTextDateTime = ...
                    moment(hit._source.lastCommunicationTime)
                .format('YYYY/MM/DD, h:mm:ss a');

                var station = {
                    "id": hit._source.id,
                    "stationName": hit._source.stationName,
                    "availableBikes": hit._source.availableBikes,
                    "availableDocks": hit._source.availableDocks,
                    "is_renting": hit._source.is_renting,
                    "lastCommunicationTime": plainTextDateTime,
                    "latitude": hit._source.latitude,
                    "longitude": hit._source.longitude,
                    "status": hit._source.status,
                    "totalDocks": hit._source.totalDocks
                };
                var thisDateEnd = new ...
                    Date(station.lastCommunicationTime.valueOf());
                //if (withinSevenDays(thisDateBegin, ...
                    thisDateEnd)) {
                station_selected_info.push(station);
                //}

            });

            if (response.hits.total !== ...
                station_selected_info.length) {
                // now we can call scroll over and over
                esClient.scroll({
                    scrollId: response._scroll_id,
                    scroll: '10s'
                }, getMoreUntilDone);

            } else {
                console.log('all done');
                return resolve(station_selected_info);

            }
        });
    });

}
```

3) There is no trigger function in ElasticSearch, so setInterval function is used

to query index every 1 minute to keep data up to date. Here, the last one day's data will be extracted from the index and the updates within one hour will be sent to the front end. In the front end, the repeated data will be removed and the newest update will be displayed on the line chart.

```
setInterval(async function () {
    var uniqueIds = stationIdsToMonitor.filter(removeDuplicateIds);
    for (let i = 0; i < uniqueIds.length; i++) {
        var thisDateBegin = new Date()
        esClient.search({
            index: 'divvy_stations_logs',
            scroll: '10s',
            body: {
                "query": {
                    "bool": {
                        "filter": [{ "term": { "id": ...
                            uniqueIds[i] } },
                        {
                            "query_string": {
                                "default_field": ...
                                    "lastCommunicationTime", ...
                                    "query": ...
                                    thisDateBegin.toISOString()
                                .split('T')[0].toString()
                            }
                        }]
                    }
                }
            }
        }, function getMoreUntilDone(error, response) {
            if (error) {
                return reject(error);
            }
            // collect all the records
            response.hits.hits.forEach((hit) => {
                plainTextDateTime = ...
                    moment(hit._source.lastCommunicationTime)
                .format('YYYY/MM/DD, h:mm:ss a');

                var station = {
                    "id": hit._source.id,
                    "stationName": hit._source.stationName,
                    "availableBikes": hit._source.availableBikes,
                    "availableDocks": hit._source.availableDocks,
                    "is_renting": hit._source.is_renting,
                    "lastCommunicationTime": plainTextDateTime,
                    "latitude": hit._source.latitude,
                    "longitude": hit._source.longitude,
                    "status": hit._source.status,
                    "totalDocks": hit._source.totalDocks
                };
                var thisDateEnd = new ...
                    Date(station.lastCommunicationTime.valueOf());
                if (withinOneHour(thisDateBegin, thisDateEnd)) {
                    io.sockets.emit(uniqueIds[i], station);
                }
            });

            if (response.hits.total !== ...
                station_selected_info.length) {
                // now we can call scroll over and over
                esClient.scroll({
                    scrollId: response._scroll_id,
                    scroll: '10s'
                }, getMoreUntilDone);

            } else {
                console.log('all done');
            }
        });
    }
}, the_interval);
```

3). The withinOneHour() function is used to extract the last one hour's updates.

```
function withinOneHour(dateBegin, dateEnd) {
    var dateDiff = dateBegin.getTime() − dateEnd.getTime();
    var leave1 = dateDiff % (24 * 3600 * 1000);
    var hours = Math.ceil(leave1 / (3600 * 1000));
    if (hours ≤ 1) {
        return true;
    } else {
        return false;
    }
}
```

3 list-of-stations

1). Added Line Chart buttons after every record of stations and changed the icon.

```
   <ng−container matColumnDef="lineChart">
        <th mat−header−cell *matHeaderCellDef ...
            class="mat−column−right"></th>
        <td mat−cell *matCellDef="let element" ...
            class="mat−column−right">
          <button mat−raised−button color="primary" ...
              (click)="findSelectedStations(element.id)">Real−Time ...
              Chart
            &nbsp; &nbsp; &nbsp;
            <mat−icon>bar_chart</mat−icon>
          </button>
        </td>
      </ng-container>
```

2). Added the "Home" button on the Far-Left. Go back to /find page.

```
<button mat-raised-button color="primary" routerLink="/find">
      Home
      &nbsp; &nbsp; &nbsp;
      <mat-icon>home</mat-icon>
    </button>
```

3). Added the "Back" button on the Far-Right. Go back to /list_of_places page.

```
<button mat-raised-button color="primary" ...
    routerLink="/list_of_places" id="back-button">
      Back
      &nbsp; &nbsp; &nbsp;
      <mat-icon>arrow_back</mat-icon>
    </button>
#back-button {
    float: right;
    margin-right: 10px
}
```

4). Get seven days data from the service: places.service.

```
findSelectedStations(id) {

    for (var i = 0,len = this.stations.length; i < len; i++) {

      if ( this.stations[i].id === id ) { // strict equality test

          var station_selected =  this.stations[i];

          break;
      }
    }


    this.placesService.findSelectedStationsSevenDay(id)
    .subscribe(() => {
      this.router.navigate(['/line_chart_divvy']);
    });

  }
```

4 places.services.ts

1). Get seven days data from the server.

```
findSelectedStationsSevenDay(id) {
    const find_selected_stations = {
      id: id
    };

    //console.log("find selected stations",id)
    var str = JSON.stringify(find_selected_stations, null, 2);

    return ...
        this.http.post(`${this.uri}/stations/selected_seven_day`, ...
        find_selected_stations, httpOptions);
  }
```

2). Get data updates from the server

```
 socket = socketio(this.uri);
  getUpdates(thisID) {

    let divvySub = new Subject<Station>();
    let divvySubObservable = from(divvySub);
    this.socket.on(thisID, (stationStatus: Station) => {
      console.log(stationStatus)
      divvySub.next(stationStatus);
    });

    return divvySubObservable;
  }
```

3). Unsubscribe from socket.io

```
  socketOff() {
    this.socket.removeAllListeners();
  }
```

5 line-chart-divvy.component

1) In ngOnInit() function, gets initial data from place service(this.placesService .getSelectedStation()) and assigns the data to the updatedStationDataSevenDay

array. Get the last one hour data and last one day data from stationDataSevenDay and assigns the data to array updatedStationDataOneHour and array updatedStationDataOneDay respectively. The child component line-chart-divvy-child.component.ts will get the update of the input variable stationDataSevenDay, stationDataOneDay and stationDataOneHour and calls ngOnChanges() to update the line chart. Client is subscribed to listen the data updating from the server. A new function stationDataSevenDayDuplicate(latestStatus) is added to remove repeated data.

```
ngOnInit() {
    this.placesService
      .getSelectedStation()
      .subscribe((data: Station[]) => {
        this.thisID = data[0].id;
        this.stationDataSevenDay = data;
        this.updatedStationDataSevenDay = data;
        this.updateDataOneDay = this.getWithinOneDayData();
        this.updateDataOneHour = this.getWithinOneHourData();
        let UpdateObservable = ...
            this.placesService.getUpdates(this.thisID);
        UpdateObservable.subscribe((latestStatus: Station) => {
          if (!this.stationDataSevenDayDuplicate(latestStatus)) {
            console.log("latestStatus:", latestStatus)
            this.updateDataSevenDay = ...
                this.stationDataSevenDay.concat([latestStatus]);
            this.updateDataOneDay = ...
                this.stationDataOneDay.concat([latestStatus]);
            this.updateDataOneHour = ...
                this.stationDataOneHour.concat([latestStatus]);
          }
        });
      });
  }
```

2). Destroy: Unsubscribe.

```
ngOnDestroy() {
    this.placesService.removeRegisteredIdOnDestory(this.thisID)
    .subscribe(() => {
      console.log("removed")
    });
    this.placesService.socketOff();
  }
```

3). Get last one hour data.

```
getWithinOneHourData() {
    var stationDataOneHour: Station[] = [];
    var dateBegin = new Date();
    for (let i = this.stationDataSevenDay.length − 1; i ≥ 0; ...
        i−−) {
      var dateEnd = new ...
          Date(this.stationDataSevenDay[i].lastCommunicationTime
      .valueOf());
      var dateDiff = dateBegin.getTime() − dateEnd.getTime();
      var leave1 = dateDiff % (24 * 3600 * 1000);
      var hours = Math.ceil(leave1 / (3600 * 1000));
      if (hours ≤ 1) {
        stationDataOneHour = [this.stationDataSevenDay[i]]
        .concat(stationDataOneHour);
      } else {
        break;
      }
    }
    return stationDataOneHour;
  }
```

4). Get last twenty four hours data.

```
getWithinOneDayData() {
    var stationDataOneDay: Station[] = [];
    var dateBegin = new Date();
    for (let i = this.stationDataSevenDay.length − 1; i ≥ 0; ...
        i−−) {
      var dateEnd = new ...
          Date(this.stationDataSevenDay[i].lastCommunicationTime
      .valueOf());
      var dateDiff = dateBegin.getTime() − dateEnd.getTime();
      var dayDiff = Math.ceil(dateDiff / (24 * 3600 * 1000));
      if (dayDiff ≤ 1) {
        stationDataOneDay = ...
            [this.stationDataSevenDay[i]].concat(stationDataOneDay);
      } else {
        break;
      }
    }
    return stationDataOneDay;
  }
```

5). Update and remove old data.

```
updateStationDataWithinOneDay() {
    var updateDateArray: Station[] = this.stationDataOneDay;
    var dateBegin = new Date();
    for (let i = 0; i < this.stationDataOneDay.length; i++) {
      var dateEnd = new ...
          Date(this.stationDataOneDay[i].lastCommunicationTime
      .valueOf());
      var dateDiff = dateBegin.getTime() − dateEnd.getTime();
      var dayDiff = Math.ceil(dateDiff / (24 * 3600 * 1000));
      if (dayDiff > 1) {
        updateDateArray = this.stationDataOneDay.slice(i + 1, ...
            this.stationDataOneDay.length);
      } else {
        break;
      }
    }
    this.stationDataOneDay = updateDateArray;
    return updateDateArray;
  }
```

6). Remove repeated data.

```
stationDataSevenDayDuplicate(stationData) {
    return this.stationDataSevenDay.some(station_selected_info ...
        => station_selected_info.id == stationData.id
      && station_selected_info.stationName == ...
          stationData.stationName
      && station_selected_info.availableDocks == ...
          stationData.availableDocks
      && station_selected_info.lastCommunicationTime == ...
          stationData.lastCommunicationTime)
  }
```

6 line-chart-divvy-child.component

1) Update one hour data and the line chart.

```
subscribeIntervalOneHour() {
    this.timeTitle = 'One Hour'
    this.whichScale = 0
    this.stationData = this.stationDataOneHour
    this.stationSelected$ = of(this.stationData)
    this.updateChart()
  }
```

2) Update one day data and the line chart

```
subscribeIntervalOneDay() {
    this.timeTitle = 'Twenty Four Hours'
    this.whichScale = 1
    this.stationData = this.stationDataOneDay
    this.stationSelected$ = of(this.stationData)
    this.updateChart()
```

```
  }
```

3) Update seven days data and the line chart.

```
subscribeIntervalSevenDay() {
   this.timeTitle = 'Seven Days'
   this.whichScale = 2
   this.stationData = this.stationDataSevenDay
   console.log(this.stationDataSevenDay)
   this.stationSelected$ = of(this.stationData)
   this.updateChart()
 }
```

4) Update SMA data and the line chart.

```
subscribeIntervalSMA() {
   this.timeTitle = 'SMA CHART'
   this.whichScale = 3
   this.stationData = this.stationDataOneHour
   this.stationSelected$ = of(this.stationData)
   this.stationDataOneDay = this.stationDataOneDay
   this.setMoveAverageHour()
   this.setMoveAverageDay()
   this.updateChartSMA()
 }
```

5). Calculate the one hour moving average.

```
setMoveAverageHour() {
   this.averageHourNumber = 0;
   this.moveAverage = [];
   for (let i = 0; i < this.stationData.length; i++) {
     let temp: moveAverage = {} as any;
     this.averageHourNumber = this.averageHourNumber + ...
         this.stationData[i].availableDocks.valueOf();
     temp.availableDocks = Number(this.averageHourNumber / (i + ...
         1));
     temp.lastCommunicationTime = ...
         this.stationData[i].lastCommunicationTime;
     this.moveAverage.push(temp);
   }
 }
```

6). Calculate the 24 hours moving average.

```
setMoveAverageDay() {
   this.averageDayNumber = 0;
   this.moveAverageDay = [];
   for (let i = 0; i < this.stationDataDay.length; i++) {
     let temp: moveAverage = {} as any;
     this.averageDayNumber = this.averageDayNumber + ...
         this.stationDataDay[i].availableDocks.valueOf();
     temp.availableDocks = Number(this.averageDayNumber / (i + 1));
     temp.lastCommunicationTime = ...
         this.stationDataDay[i].lastCommunicationTime;
     this.moveAverageDay.push(temp);
   }
 }
```

7). Draw the line chart.

```
drawChart() {
   this.drawAxis();
   this.drawLine();
 }
```

8). Update the real-time line chart.

```
 updateChart() {
   var body = d3.select('body').transition();
   body.selectAll(".d-inline-block")
     .style("opacity", 1);

   this.x.domain(d3Array.extent(this.stationData, (d) => new ...
       Date(d.lastCommunicationTime.valueOf())));
   this.y.domain([0, d3Array.max(this.stationData, (d) => ...
       Number(d.availableDocks.valueOf())) + 5]);

   /* this.line.x((d: any) => this.x(new ...
       Date(d.lastCommunicationTime.valueOf())))
     .y((d: any) => this.y(d.availableDocks.valueOf())) */

   var svg = d3.select('svg').transition();
   svg.selectAll(".line")
     .duration(750)
     .attr("d", this.line(this.stationData))

   svg.selectAll(".line1")
     .style("opacity", 0)

   svg.selectAll(".line2")
     .style("opacity", 0)

   svg.select(".axis.axis--x") // change the x axis
     .duration(750)
     .call(this.xAxis);
```

```

    svg.select(".axis.axis--y") // change the y axis
      .duration(750)
      .call(this.yAxis);
  }
```

9). Update the SMA line chart.

```
updateChartSMA() {
    var maximum : any;
    var body = d3.select('body').transition();
    body.selectAll(".d-inline-block")
      .style("opacity", 0);

    /* this.line.x((d: any) => this.x(new ...
        Date(d.lastCommunicationTime.valueOf())))
      .y((d: any) => this.y(d.availableDocks.valueOf())) */
    this.cutMoveAverageDay = ...
        this.moveAverageDay.slice(-this.stationData.length)

    maximum = this.maxData(d3Array.max(this.cutMoveAverageDay, ...
        (d) => Number(d.availableDocks.valueOf()))
            ,d3Array.max(this.moveAverage, (d) => ...
                Number(d.availableDocks.valueOf()))
            ,d3Array.max(this.stationData, (d) => ...
                Number(d.availableDocks.valueOf())))
    this.x.domain(d3Array.extent(this.stationData, (d) => new ...
        Date(d.lastCommunicationTime.valueOf())));

    this.y.domain([0, Number(maximum) + 5]);


    var svg = d3.select('svg').transition();

    svg.selectAll(".line")
      .duration(750)
      .attr("d", this.line(this.stationData));

    svg.selectAll(".line1")
      .attr("d", this.line1(this.moveAverage))
      .style("opacity", 1);

    svg.selectAll(".line2")
      .attr("d", this.line2(this.cutMoveAverageDay))
      .style("opacity", 1);

    svg.select(".axis.axis--x") // change the x axis
      .duration(750)
      .call(this.xAxis);

    svg.select(".axis.axis--y") // change the y axis
      .duration(750)
      .call(this.yAxis);
  }
```