Assignment #3 Documentation

Yuqi Zhou

23 March

Contents

1 divvy_stations_status

1) Created new procedure function notify_event() to notify server the updating data.

```
cursor.execute("""CREATE OR REPLACE FUNCTION notify_event() ...
    RETURNS TRIGGER AS $$
  DECLARE
    record RECORD;
    payload JSON;
  BEGIN
    IF (TG_OP = 'DELETE') THEN
      record = OLD;
    ELSE
      record = NEW;
    END IF;
    payload = json_build_object('data', row_to_json(record));

    PERFORM pg_notify('events', payload::text);

    RETURN NULL;
  END;
$$ LANGUAGE plpgsql;""")
```

2) Created new trigger divvy_update on table divvy_stations_logs and for each row of newly inserted data, execute procedure notify_event().

```
 cursor.execute("""CREATE TRIGGER divvy_update
  AFTER INSERT ON divvy_stations_logs
  FOR EACH ROW EXECUTE PROCEDURE notify_event();""")
```

2 server.js

1) Created new post request and added new query function to query last seven days data from the database. Return station_selected_info as the response.

```
router.route('/stations/selected_seven_day').post((req, res) => {
    var str = JSON.stringify(req.body, null, 4);
    for (var i = 0,len = stations_found.length; i < len; i++) {

        if ( stations_found[i].id === req.body.id ) { // strict ...
            equality test

```

```
            station_selected = stations_found[i];

            break;
        }
    }

    const query1 = {
        // give the query a unique name
        name: 'fetch-divvy-selected-station_seven_day',
        text: ' SELECT * FROM divvy_stations_logs WHERE id = $1 ...
            AND lastcommunicationtime >= (NOW() - INTERVAL \'168 ...
            hours\' ) order by lastcommunicationtime',
        values: [station_selected.id]
    }

    find_selected_stations_from_divvy(query3).then(function ...
        (response) {
        var hits = response;
        res.json({'station_selected_info': 'Added successfully'});
    });

});
```

2) Query last seven days information of a specific divvy station and push data into array station_selected_info.

```
async function find_selected_stations_from_divvy(query) {
    const response = await pgClient.query(query);
    station_selected_info = [];

    for (i = 0; i < response.rows.length; i++) {
        plainTextDateTime =  ...
            moment(response.rows[i].lastcommunicationtime).format('YYYY-MM-DD, ...
            h:mm:ss a');

        var station = {
            "id": response.rows[i].id,
            "stationName": response.rows[i].stationname,
            "availableBikes": response.rows[i].availablebikes,
            "availableDocks": response.rows[i].availabledocks,
            "is_renting": response.rows[i].is_renting,
            "lastCommunicationTime": plainTextDateTime,
            "latitude": response.rows[i].latitude,
            "longitude": response.rows[i].longitude,
            "status": response.rows[i].status,
            "totalDocks": response.rows[i].totaldocks
        };

        station_selected_info.push(station);

    }

}
```

3) Listen on the notification from the database and added socket.io to help push new coming data to subscribed clients.

```
const http = require('http')
var server = app.listen(4000, () => console.log('Express server ...
    running on port 4000'));
var io = require('socket.io').listen(server);
pgClient.query('LISTEN events')

pgClient.on('notification', (msg) => {
        var obj = JSON.parse(msg.payload);  // msg.payload string
        plainTextDateTime =  ...
            moment(obj.data.lastcommunicationtime).format('YYYY-MM-DD, ...
            h:mm:ss a');
        var station = {
            "id": obj.data.id,
            "stationName": obj.data.stationname,
            "availableBikes": obj.data.availablebikes,
            "availableDocks": obj.data.availabledocks,
            "is_renting": obj.data.is_renting,
            "lastCommunicationTime": plainTextDateTime,
            "latitude": obj.data.latitude,
            "longitude": obj.data.longitude,
            "status": obj.data.status,
            "totalDocks": obj.data.totaldocks };
        io.sockets.emit(obj.data.id, station);
 })

io.on('connection', function (socket) {
    console.log('a user connected');
  });
```

3 list-of-stations

1). Added Line Chart buttons after every record of stations.

```
<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)">Line ...
              Chart</button>
        </td>
      </ng-container>
```

2). Added the "Back" button. Go back to /list_of_places page.

```
<button mat-raised-button color="primary" ...
    routerLink="/list_of_places">
        Back
             
        <mat-icon>search</mat-icon>
      </button>
```

3). 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);
```

```
10    }
```

2). Get data updates from the server

```
1   socket = socketio(this.uri);
2    getUpdates(thisID) {
3
4      let divvySub = new Subject<Station>();
5      let divvySubObservable = from(divvySub);
6      this.socket.on(thisID, (stationStatus: Station) => {
7        console.log(stationStatus)
8        divvySub.next(stationStatus);
9      });
10
11     return divvySubObservable;
12   }
```

3). Unsubscribe from socket.io

```
1    socketOff() {
2      this.socket.removeAllListeners();
3    }
```

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.

```
1  ngOnInit() {
2      this.placesService
3        .getSelectedStation()
4        .subscribe((data: Station[]) => {
5          this.thisID = data[0].id
6          this.stationDataSevenDay = data
7          this.updatedStationDataSevenDay = data
8          this.updateDataOneDay = this.getWithinOneDayData()
```

```
9           this.updateDataOneHour = this.getWithinOneHourData()
10
11          let UpdateObservable = ...
                this.placesService.getUpdates(this.thisID);
12          UpdateObservable.subscribe((latestStatus: Station) => {
13            this.updateDataSevenDay = ...
                  this.stationDataSevenDay.concat([latestStatus])
14            this.updateDataOneDay = ...
                  this.stationDataOneDay.concat([latestStatus])
15            this.updateDataOneHour = ...
                  this.stationDataOneHour.concat([latestStatus])
16          });
17        });
18    }
```

2). Destroy: Unsubscribe.

```
1   ngOnDestroy() {
2       if (this.subscription) {
3         this.subscription.unsubscribe();
4       }
5       this.placesService.soketOff();
6     }
```

3). Get last one hour data.

```
1   getWithinOneHourData() {
2       var stationDataOneHour: Station[] = [];
3       var dateBegin = new Date();
4       for (let i = this.stationDataSevenDay.length - 1; i >= 0; ...
            i--) {
5         var dateEnd = new ...
              Date(this.stationDataSevenDay[i].lastCommunicationTime.valueOf());
6         var dateDiff = dateBegin.getTime() - dateEnd.getTime();
7         var leave1 = dateDiff % (24 * 3600 * 1000);
8         var hours = Math.ceil(leave1 / (3600 * 1000));
9         if (hours <= 1) {
10          stationDataOneHour = ...
                [this.stationDataSevenDay[i]].concat(stationDataOneHour);
11        } else {
12          break;
13        }
14      }
15      return stationDataOneHour;
16    }
```

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.splice(i + 1, ...
            this.stationDataOneDay.length);
      } else {
        break;
      }
    }
    this.stationDataOneDay = updateDateArray;
    return updateDateArray;
  }
```

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++) {
```

```
5         let temp: moveAverage = {} as any;
6         this.averageHourNumber = this.averageHourNumber + ...
              this.stationData[i].availableDocks.valueOf();
7         temp.availableDocks = Number(this.averageHourNumber / (i + ...
              1));
8         temp.lastCommunicationTime = ...
              this.stationData[i].lastCommunicationTime;
9         this.moveAverage.push(temp);
10      }
11    }
```

6). Calculate the 24 hours moving average.

```
1   setMoveAverageDay() {
2      this.averageDayNumber = 0;
3      this.moveAverageDay = [];
4      for (let i = 0; i < this.stationDataDay.length; i++) {
5        let temp: moveAverage = {} as any;
6        this.averageDayNumber = this.averageDayNumber + ...
             this.stationDataDay[i].availableDocks.valueOf();
7        temp.availableDocks = Number(this.averageDayNumber / (i + 1));
8        temp.lastCommunicationTime = ...
             this.stationDataDay[i].lastCommunicationTime;
9        this.moveAverageDay.push(temp);
10     }
11   }
```

7). Draw the line chart.

```
1   drawChart() {
2      this.drawAxis();
3      this.drawLine();
4    }
```

8). Update the real-time line chart.

```
1    updateChart() {
2      var body = d3.select('body').transition();
3      body.selectAll(".d-inline-block")
4        .style("opacity", 1);
5
6      this.x.domain(d3Array.extent(this.stationData, (d) => new ...
           Date(d.lastCommunicationTime.valueOf())));
7      this.y.domain([0, d3Array.max(this.stationData, (d) => ...
           Number(d.availableDocks.valueOf())) + 5]);
8
9      /* this.line.x((d: any) => this.x(new ...
           Date(d.lastCommunicationTime.valueOf())))
```

```
10        .y((d: any) => this.y(d.availableDocks.valueOf())) */
11
12      var svg = d3.select('svg').transition();
13      svg.selectAll(".line")
14        .duration(750)
15        .attr("d", this.line(this.stationData))
16
17      svg.selectAll(".line1")
18        .style("opacity", 0)
19
20      svg.selectAll(".line2")
21        .style("opacity", 0)
22
23      svg.select(".axis.axis--x") // change the x axis
24        .duration(750)
25        .call(this.xAxis);
26
27      svg.select(".axis.axis--y") // change the y axis
28        .duration(750)
29        .call(this.yAxis);
30    }
```

9). Update the SMA line chart.

```
1  updateChartSMA() {
2      var maximum : any;
3      var body = d3.select('body').transition();
4      body.selectAll(".d-inline-block")
5        .style("opacity", 0);
6
7      /* this.line.x((d: any) => this.x(new ...
           Date(d.lastCommunicationTime.valueOf())))
8        .y((d: any) => this.y(d.availableDocks.valueOf())) */
9      this.cutMoveAverageDay = ...
           this.moveAverageDay.slice(-this.stationData.length)
10
11     maximum = this.maxData(d3Array.max(this.cutMoveAverageDay, ...
           (d) => Number(d.availableDocks.valueOf()))
12             ,d3Array.max(this.moveAverage, (d) => ...
                   Number(d.availableDocks.valueOf()))
13             ,d3Array.max(this.stationData, (d) => ...
                   Number(d.availableDocks.valueOf())))
14     this.x.domain(d3Array.extent(this.stationData, (d) => new ...
           Date(d.lastCommunicationTime.valueOf())));
15
16     this.y.domain([0, Number(maximum) + 5]);
17
18
19     var svg = d3.select('svg').transition();
20
21     svg.selectAll(".line")
22       .duration(750)
23       .attr("d", this.line(this.stationData));
24
```

```
25      svg.selectAll(".line1")
26        .attr("d", this.line1(this.moveAverage))
27        .style("opacity", 1);
28
29      svg.selectAll(".line2")
30        .attr("d", this.line2(this.cutMoveAverageDay))
31        .style("opacity", 1);
32
33      svg.select(".axis.axis--x") // change the x axis
34        .duration(750)
35        .call(this.xAxis);
36
37      svg.select(".axis.axis--y") // change the y axis
38        .duration(750)
39        .call(this.yAxis);
40    }
```