```
var express = require('express');
var bodyParser = require('body-parser');
var fs = require('fs');
const { Pool, Client} = require('pg');

app = express();
app.use(express.static("public"));  //serves all static files on the localhost

//use the public folder bc you do not want to serve the app.js and package.json static files to users
app.use(bodyParser.json());

var login_id = null;
var maxen = null;

const pool = new Pool({
  user: 'signout_admin',
  host: 'localhost',
  database: 'signoutdb',
  password: 'signout_admin',
  port: 5431,
});


app.get('/', (req, res)=>{
  res.sendFile(__dirname + '/public/views/intro.html');
});

app.get('/login', (req, res)=>{
  res.sendFile(__dirname + '/public/views/login.html');
});

app.get('/login/authenticate', (req, res)=>{
  if (req){
    var u_input = req.query.username;
    var p_input = req.query.pwd;
    //authenticate
    pool.query("SELECT id FROM users WHERE (username='" + u_input + "') AND (password='" + p_input + "')", (err, response) => {
      //if the username and password is not correct, the select statement will
      //return an empty array []. Err will never run unless the sql is wrong
      //recognize [] as incorrect login information. if incorrect uname+pwd, output.id will throw cannot read property 'id' of undefined
```

```
        if (err) {
          console.log(err);
        }
        if (response) {
          var output = response.rows[0]; //res.rows is an array
          try{
            var id_number = output.id; //NOTE: admin's id number is
            NULL so id_number = null for admin
            login_id = id_number;
            console.log('Logged in as ' + id_number);
            if (id_number == null){
              res.json({'id':'admin'}) // don't send ok. send admin if
              admin and id if student
            }
            else{
              res.json({'id':login_id});
            }

          }
          catch(error){
            console.log('incorrect credentials');
            res.json({'id':'ic'}) //ic = incorrect credentials
          }

        }
      });
    }

});

app.get('/get-id', (req, res)=>{
  res.json({'id':login_id});
});

app.get('/student', (req, res)=>{
  res.sendFile(__dirname + '/public/views/student.html');
});

app.get('/admin', (req, res)=>{
  res.sendFile(__dirname + '/public/views/admin.html')
});

app.get('/log', (req, res)=>{
  res.sendFile(__dirname + '/public/views/log.html')
});

app.get('/analytics', (req, res)=>{
```

```
app.get('/analytics', (req, res)=>{
  res.sendFile(__dirname + '/public/views/analytics.html')
});

app.get('/record', (req, res)=>{
  res.sendFile(__dirname + '/public/views/record.html')
});

app.get('/workspace', (req, res)=>{
  res.sendFile(__dirname + '/public/views/workspace.html')
});


app.get('/log/ajax-get', (req, res)=>{
  console.log('Get Request encountered');

  var promise = new Promise(function(){ //promise to do something
  later --> better version of a callback (cannot have multiple
  callbacks for one function)
    pool.query('SELECT * FROM records', (err, response)=>{
      if (err){
        console.log(err)
      }
      if (response){
        var output = response.rows;
        res.send(output);
      }
    })
  });
});

app.get('/max-entry', (req, res)=>{

  console.log('max entry number get request encountered');
  pool.query('SELECT MAX(entry) FROM records;', (err, response)=>{
    if (err){
      console.log(err);
    }
    if (response){
      res.json(response.rows[0]);
    }
  })

});

app.get('/location-data', (req, res)=>{
  //bathroom, main office, nurse, , library other
```

```
  pool.query("SELECT entry FROM records WHERE(location =
  'Bathroom');", (err, response)=>{
    if (err){console.log(err);}
    if (response){
      pool.query("SELECT entry FROM records WHERE(location = 'Main
      Office');", (err, r2)=>{
        if (err){console.log(err);}
        if (r2){
          pool.query("SELECT entry FROM records WHERE(location =
          'Nurse');", (err, r3)=>{
            if (err){console.log(err);}
            if (response){
              pool.query("SELECT * FROM records;", (err, r4)=>{
                if (err){console.log(err);}
                if (r4){
                  res.send({Bathroom:response.rows.length,
                  Main_Office:r2.rows.length, Nurse:r3.rows.length,
                  Other:(r4.rows.length - r2.rows.length -
                  r3.rows.length - response.rows.length)})
                }
              });
            }
          });
        }
      });
    }
  });
});

app.get('/ajax-post', (req, response)=>{
  pool.query('SELECT NOW()', (err, res) => { //replace with python
  script for dt
    if (err) {
      console.log(err);
    }
    if (res) {
      var output = res.rows;
      pool.query("INSERT INTO records VALUES(" + req.query.entry +
      "," + req.query.id + ",'" + req.query.loc + "','" +
      output[0].now + "')", (err)=>{
        if (err){console.log(err);}
      });
    }
  });
});

app.post('/student_record', (req, response)=>{
```

```
168     console.log(req.query); //location
169     console.log(login_id); //id
170     pool.query('SELECT NOW()', (err, res) => { //time
171       if (res) {
172         var output = res.rows;
173         pool.query('SELECT MAX(entry) FROM records;', (err,
  •         response)=>{
174           if (err){
175             console.log(err);
176           }
177           if (response){
178             console.log(response.rows[0].max + 1); //this is new entry
  •             val
179             pool.query("INSERT INTO records VALUES(" +
  •             (response.rows[0].max + 1) + "," + login_id + ",'" +
  •             req.query.loc + "','" + output[0].now + "')", (err)=>{
180               if (err){console.log(err);}
181             });
182           }
183         });
184
185       }
186     });
187   });
188
189   app.get('/time', (req, res)=>{
190     pool.query("SELECT time FROM records ORDER BY entry DESC LIMIT 1",
  •     (err, response)=>{
191       if (err){console.log(err);}
192       if (response){
193         //res.send only works in asynchronous--> after promise
194
195         //outputs last time
196         var prom = new Promise(function(){ //asynchronous callback.
  •         Easier if this section is synchronous
197           var time = response.rows[0].time;
198           time = time.split(' ');
199           // console.log(time); //this is the maximum time value
200           //values needed: 1 (month),2 (day),3 (year)
201           var end_date = {'m':time[1], 'd':time[2], 'y':time[3]}
202           pool.query("SELECT time FROM records", (err, time_res)=>{
203             if (err){console.log(err);}
204             if (time_res){
205               var time_list = {values: []};
206               function timesend(){
207                 // console.log(time_list);
208                 res.json(time_list);
```

```
            res.json(time_list);
          }
          for (z=0; z < (time_res.rows.length); z++){
            var check_date = time_res.rows[z].time; //assigns
            check_date to entire time string (from db)
            check_date = check_date.split(' ');
            if (check_date[1] == time[1] && check_date[3] ==
            time[3]){ //if the month and year are the same
              var zcheck = check_date[2].toString(); //convert to
              string to check 0-
              // console.log(zcheck.charAt(0)); //will be 0 (for
              08,9,etc.), 1 (11, 12, 13), or 2 (23, 24), or 3 (31,
              30)
              if (zcheck.charAt(0)=='0'){
                zcheck = zcheck.slice(1);
                // console.log(zcheck); //this works --> convert
                to int
                zcheck = Number(zcheck); //int conversion
                check_date[2] = zcheck;
              }
              else{
                zcheck = Number(zcheck);
                check_date[2] = zcheck;
              }

              if (zcheck + 5 >= time[2]){
                // console.log(check_date); //this WORKS! the last
                5 days are the only returned values
                time_list.values.push(check_date);
              }
            }
          }
          timesend();
        }
      });

    });
  }
});

app.get('/analytics/filter-log', (req, res)=>{
  console.log(req.query.filter + ' ' + req.query.type);
  pool.query("SELECT * FROM records WHERE (" + req.query.type + "='"
  + req.query.filter + "')", (err, response)=>{
    if (err){console.log(err);}
```

```
    if (response){
      console.log(response.rows);
      res.send(response.rows);
    }
  });


});

app.listen(8080, ()=>{
  console.log('SignOut is listening for web requests on TCP port
  8080.');
});

```