Developers Guide

**MEAN Framework**

The web application is based on the MEAN framework. This framework relies on several technologies to accelerate and simplify multi-tier web application development. It uses MongoDB as the database technology, Express as a web application middleware framework, AngularJS as a client side framework, and Node.js provides the server side runtime environment including the HTTP server.

MongoDB [M1] is a NoSQL database platform that is classified as a *document-oriented database*. While a traditional relational database structures data in terms of tables and rows, a document-oriented database uses *collections* of *documents*. Documents are structured, hierarchal grouping of related, primitive data values. MongoDB supports insert and selection of documents in JSON format.

Node.js [M2] is a popular JavaScript runtime for developing server side web applications. It runs on the open source V8 JavsScript engine developed by Google for use in Google Chrome and is therefore well-supported, lightweight, and efficient. Node.js includes a comprehensive API with common web application functions such as networking and non-blocking I/O, and includes an HTTP server.

Express [M3] is a framework built for Node.js that includes a number of utility functions useful for developing an HTTP server. It provides facilities for routing HTTP requests to *middleware*. Middleware allows a web server to provide common services for requests and responses. Examples include logging of all HTTP requests or parsing JSON payloads into standard JavaScript objects prior to invoking the request handler.

AngularJS [M4] is a client side JavaScript application framework. AngularJS provides a client side MVC architecture to simplify the creation of responsive web pages based on HTML, CSS, and JavaScript. The framework provides custom HTML tag attributes allowing responsive elements to be embedded in a page using a declarative programming style. This simplifies the development of rich client side logic targeted for web browsers.

The use of MongoDB, Node.js, Express, and AngularJS provides a unified development approach. Each of the technologies is based on JavaScript which allows for more code reuse and less context switching for developers as they move between server side and client side application development. JSON naturally lends itself as a data interchange format between the various technologies. It is easily serialized, transferred as a payload using HTTP, and deserialized between the client and server. MongoDB allows data to be inserted and returned directly in the JSON format.

For the web application presented in this paper, MEAN.IO [M5] was utilized. MEAN.IO is an open source, integrated full stack which includes a basic build of each of the core MEAN technologies. This allows a project to get started quickly with features such as authentication and user sign up using Passport [M6]. MEAN.IO also provides best practices and a common project hierarchy. Projects can be organized as packages to separate core application features. For example, user authentication can be handled using a user package, while data can be presented to a user using a seperate presentation package. In addition, MEAN.IO is preconfigured with several development best practices including package management services such as NPM and Bower, automated management of test and production environments using Grunt, and coding standard checks using jslint and csslint.

**Source Control**

The project is maintained using *git.* Git is an open source, distributed version control system. Git allows multiple developers to work on a project simultaneously. Feature branches can be created to maintain history of the changes required to implement various parts of an application. The feature branches can then be merged back into the master branch. The master branch for this web application is maintained using the GitHub service [M7], which provides a cloud based service to maintain git projects.

**Infrastructure**

The application was built using Ubuntu 14.04 LTS (Trusty Tahr) [M8]. This is a popular Linux distribution maintained by Canonical. This application can be deployed on any local machine running Linux or deployed in a cloud service such as Microsoft Azure or Amazon Web Services. Because the application runs on the MEAN framework which is not natively compiled, it would also be possible to set up the application on a Microsoft Windows or Apple OSX machine with minimal development effort. Multiple independent web servers can be spawned on a single multiprocessor machine, or multiple independent web servers could be utilized, provided they are all configured to access a single MongoDB database server instance.

**Software Architecture**

The web server is implemented on an internet-connected computer running Ubuntu 14.04 LTS. HTTP requests received on port 80 of the internet facing router are forwarded to port 3000 of the of the web server where the Node.js HTTP server is listening.A REST API is utilized to handle requests from clients, including user sign up and authentication, requests for interpolated pollution data, requests for triangulations of measurement sites, and uploading and viewing of user routes. A REST call is used to initiate the downloading of pollution data from the AirNow FTP server and initiate the triangulation and interpolation of the data using the SF-based reduction method. A cron job on the web server calls the required REST service functions to stream and interpolate pollution data each hour. Visualization of the pollution data is rendered on the client side by embedding a Google Maps application within the AngularJS application.

**AirNow Data**

The U.S EPA AirNow program provides real-time observed air quality information across the United States, Canada, and Mexico. AirNow receives real-time air quality observations from over 2,000 monitoring stations and collects forecasts for more than 300 cities. The AirNow program includes a web services API for accessing current and historical pollution data [M10]. However, queries to this service are generally rate limited to 500 per hour. Therefore, the web application uses the alternative FTP server method for accessing the AirNow data.

Two file products are used from the AirNow program. The first file product utilized is the Monitoring Site Information file. This file includes a number of attributes describing each monitoring site, including the site name, latitude, longitude, and types of parameters measured at the site. The second file product utilized is the Hourly Data file. This file includes one measurement per row with attributes including the site where the measurement was obtained, the measurement value, units, and time when the measurement was taken.

**Server Jobs**

The server is set up with two cron jobs. These jobs are executed at 5 after ever hour, and 30 minutes past every hour, respectively. Each job simply uses wget to call a server function through the REST API. For illustration, here are the lines from the crontab file:

5 \* \* \* \* wget -O /dev/null http://localhost:3000/filesync

30 \* \* \* \* wget -O /dev /null <http://localhost:3000/reload>

The first job calls a bash server script that syncronizes the files on the AirNow FTP server with a local cache of the files. This operation uses LFTP [M11] to simplify the function of determining which files on the server are new and should be downloaded locally.

The second job calls the reload function. This function scans the local cache of hourly AirNow data, and loads any measurements into the MongoDB database that have not already been loaded. This function also performs precomputations of spacial triangulations. This calculation is done in the background to prior to the making the associated measurement data available to the application for query. By performing computationally expense triangulation functions at data load time, user queries can be executed quickly at a later time.

**MongoDB Database Design**

The MongoDB database is set up as a single centralized database. This database is designed to include all state necessary to drive the web application, allowing several web servers and clients to access the data simultaneously. MongoDB updates each document in the database as an atomic operation assuring that access to the data is synchronized among multiple server instances.

The database includes a collection to store user data, hourly data, loaded files, user routes, and precomputed triangulations. Each of those collections is described briefly here.

*User data*

The users document stores information including the user's email, hashed password, password salt, and username. This document collection is utilized to provide authentication services and to associate specific routes uploaded by the user.

*Hourly data*

The hourlydata document stores information downloaded from the AirNow program. This document collection contains data from both file products downloaded from the AirNow FTP server (monitoring site information combined with hourly site measurement data). This data is populated by the *reload* server job.

*Loaded Files*

The loadedfiles document stores the name of a file containing one hour's worth of monitoring site information and hourly data. This collection is queried to determine which data from the AirNow FTP server have been loaded. Any unloaded files can then be processed by the server application when the *reload* job is run.

*User Routes*

The myroutes document contains a user id, date, time, and GPS coordinates describing a route the user may have traveled while recording position and time data with a mobile device. This document also stores interpolated pollution values at each coordinate/time pair so a user can estimate their exposure.

*Precomputed Triangulations*

The triangles document contains the coordinates of the corners of a triangle, date, time, pollution parameter (such as PM 2.5 or OZONE) and the measured values for each coordinate for that parameter at that time. This collection represents a delaunay triangulation that is computed when the *reload* job is run.

The following indexes should be created during the web application build process to assure acceptable performance of common queries as historical data scales in size:

db.triangles.createIndex( { valid\_date : 1, valid\_time : 1, parameter\_name : 1, triangle : "2dsphere" } )

db.triangles.createIndex( { valid\_date : 1, valid\_time : 1, parameter\_name : 1  } )

db.hourlydata.createIndex( { measurement\_key: 1, valid\_date : 1, valid\_time : 1 } )

db.hourlydata.createIndex( { valid\_date : 1, valid\_time : 1, parameter\_name : 1 } )

Of particular interest is the first index which is defined as “2dsphere”. This is a special index in MongoDB to provide support for performant geospacial queries. This index allows the web application to quickly find the containing triangle of any point that needs to be interpolated. By using precomputed triangulations and a quick point location function based on a geospatial index, a large number of query points can be interpolated in real time based on user demand.

**Node.js Application**

The server side application is structured as an MVC application. Server side routes are typically structured as REST services. For example, the following route maps a post operation to the http://domain/register URL to a controller function to create a new user:

// Setting up the users api

app.route('/register')

.post(users.create);

The users.create controller will then access the users model. The users model accesses the user document collection in the MongoDB database. If a successful registration can be performed, a view can then be used to render a page to be transferred to the client in response to the registration request.

\*this section can be greatly expanded to describe all routes defining the full REST API, controllers, models, and server side view rendering. The description of users here is representative of the software architecture\*

**Interpolation Method**

The web application uses the SF-based reduction method to visualize data. The reduction method requires that spacial dimensions and time time dimensions be treated separately. For this particular application, the method can be described simply with the following steps:

1. Measured data for the most recent hour is downloaded from the AirNow FTP site and loaded to the database

2. Since the measurement locations are typically fixed and finite, and the time resolution is desired to the nearest hour, any missing values for a specific parameter at a specific measurement site at a specific hour can be interpolated linearly and stored along with measured hourly data in the hourlydata collection

3. A delaunay triangulation can be computed for any hour/parameter combination where data has been updated. This always includes the data for the latest downloaded hour and may include data for previous hours if a time based interpolation was calculated. Triangulations are stored in the triangles collection.

4. Later, when a query is received, the web application can use a MongoDB supported geospacial query to locate the containing triangle in the triangles collection.

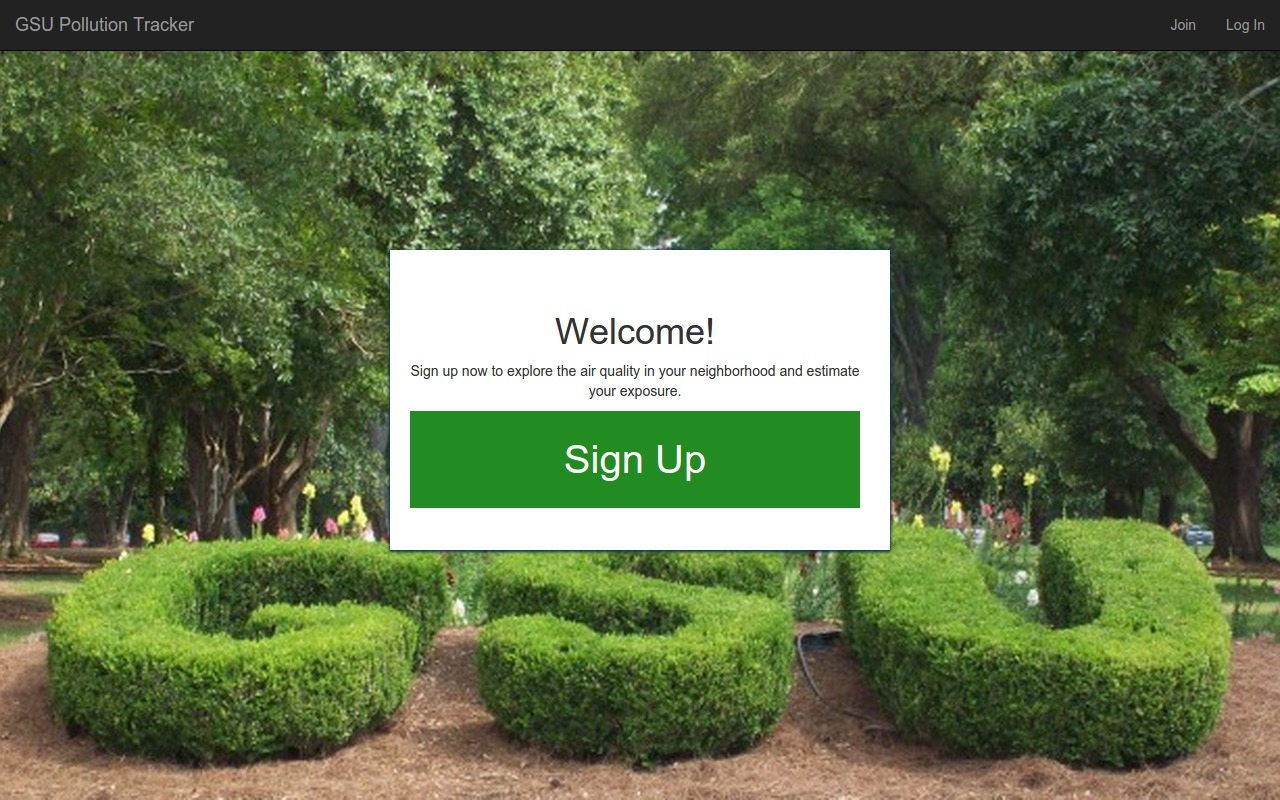
5. The shape function can be applied since the measured value (or time interpolated value) of each triangle vertex is known.

**AngularJS Client Application**

The AngularJS client application can be accessed by using a web browser such as Mozilla Firefox or Google Chrome. This client side web application provides a convenient interface to the server side REST API to render interpolated pollution data using services provided by Google Maps. It allows a user to sign up for an account, browse current and historical pollution data, adjust pollution visualization parameters (radius of influence, max parameter value, triangulations, and measurement site details), and upload/view personal routes.

**Welcome Screen**

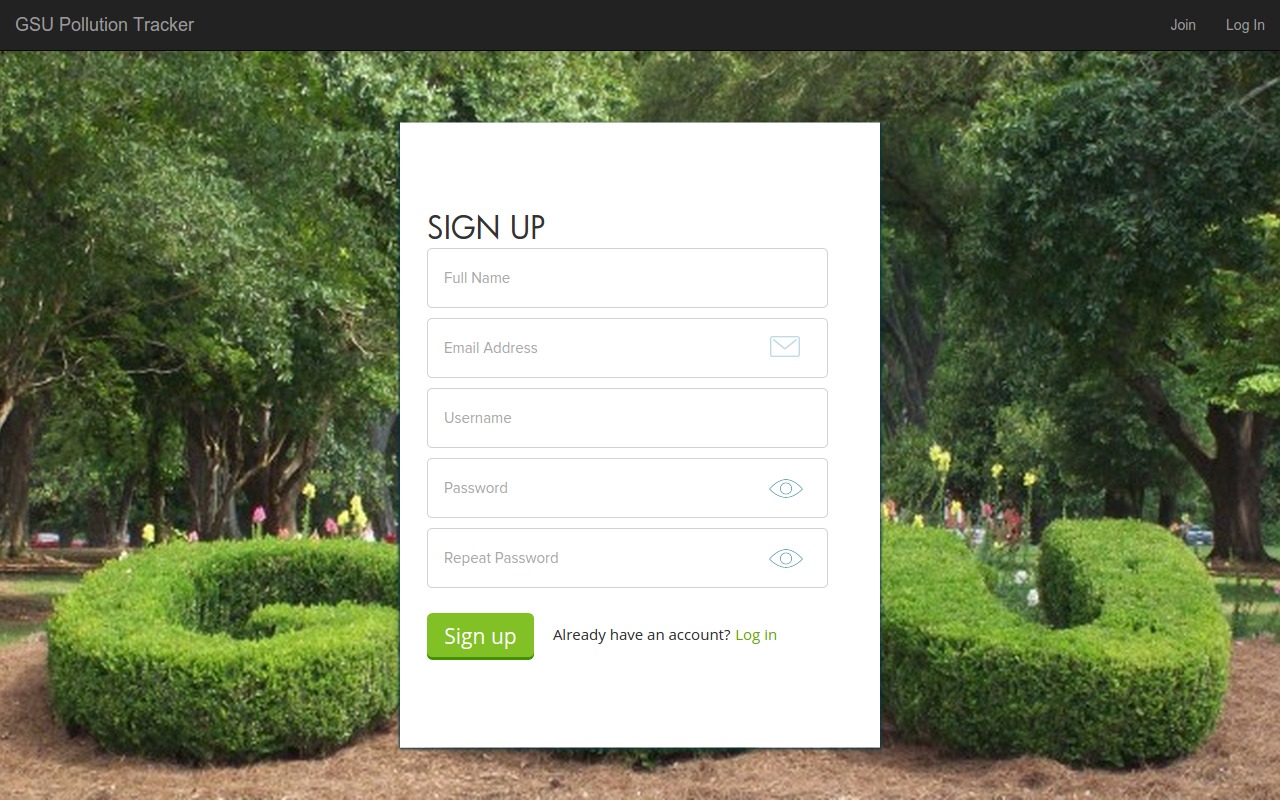
The following figure depicts the welcome screen/landing page when the website URL is entered into a browser. From this screen, the user can sign up using the prominently displayed button or log in if they already have an accout.



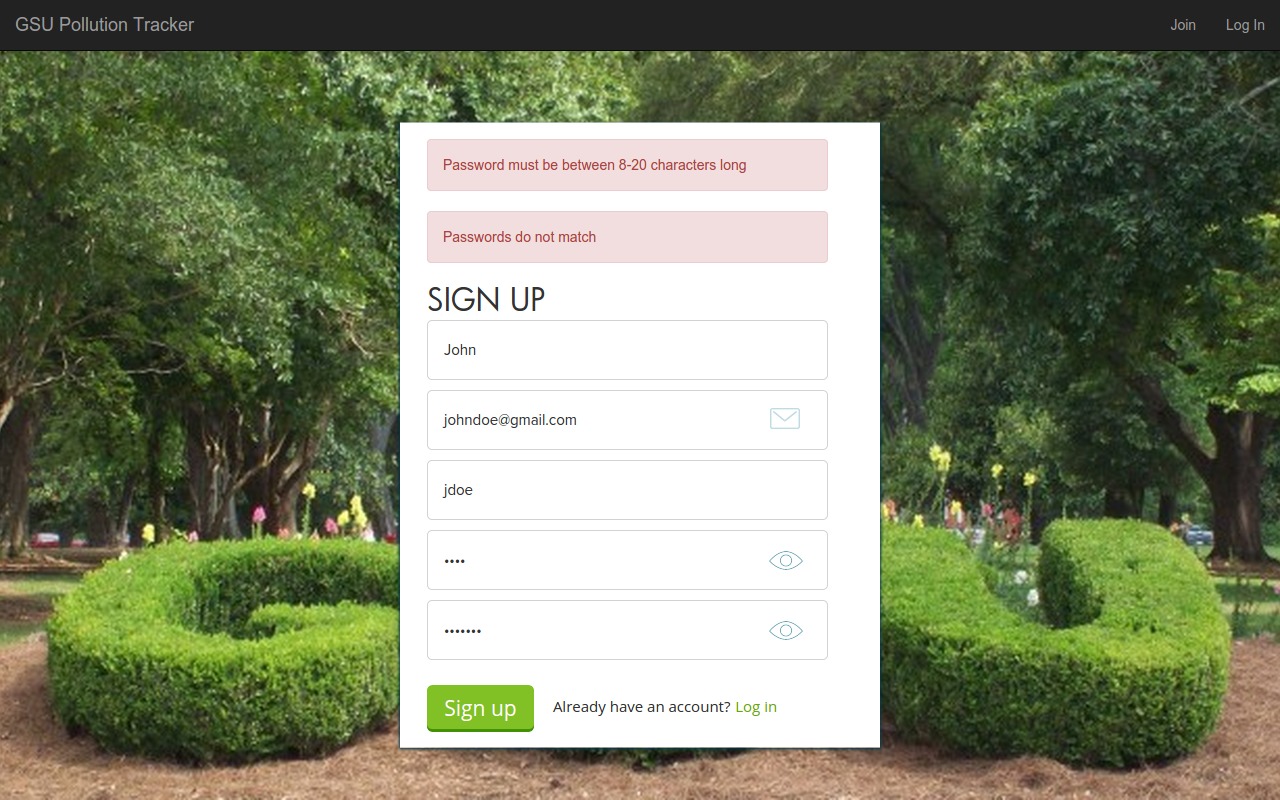
**User Sign Up**

When the user attempts to sign up, they must fill out a simple form including the following basic information:

* Name
* Email Address
* Username
* Password

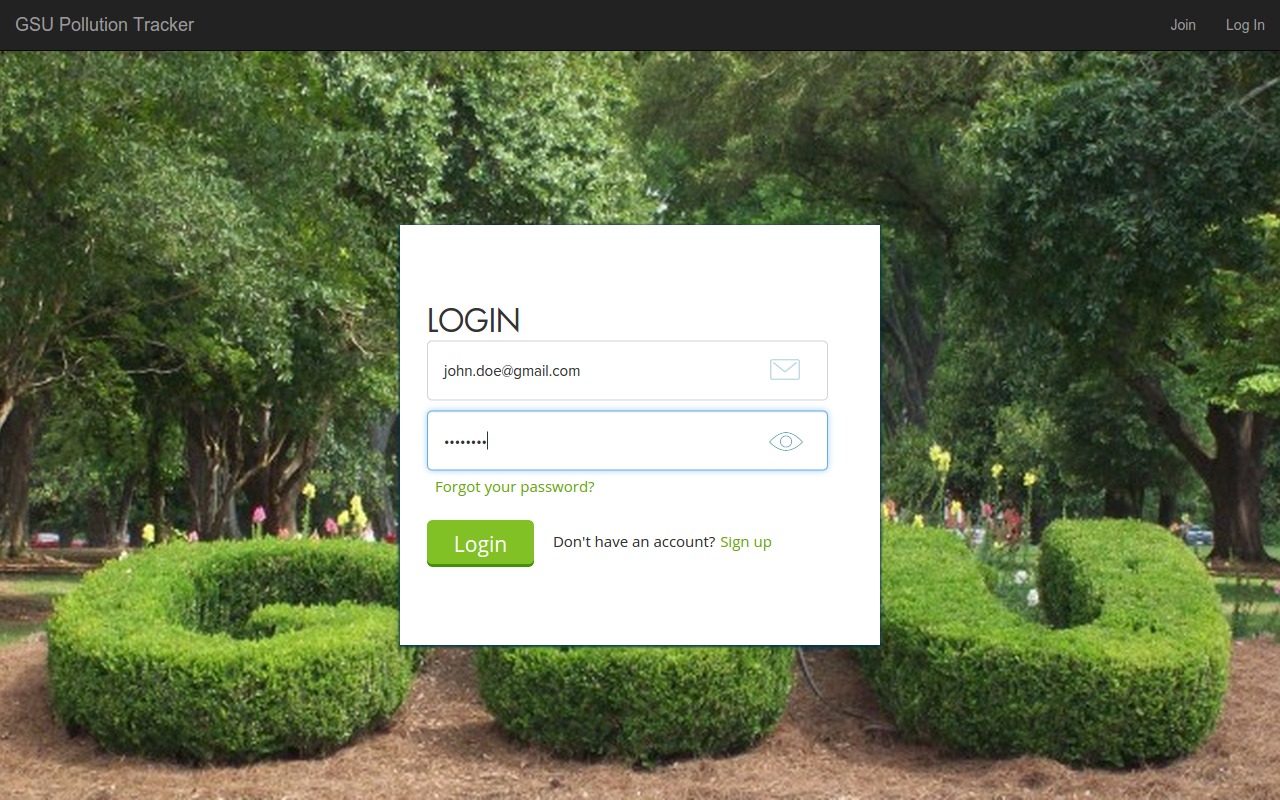


The form includes validation for required fields, input format, and password complexity rules. Any field that does not pass validation will prompt the user with an alert message rendered in the form. This includes client side validation for basics email formatting and required fields as well as server side validation to check for email addresses or user names already in use:



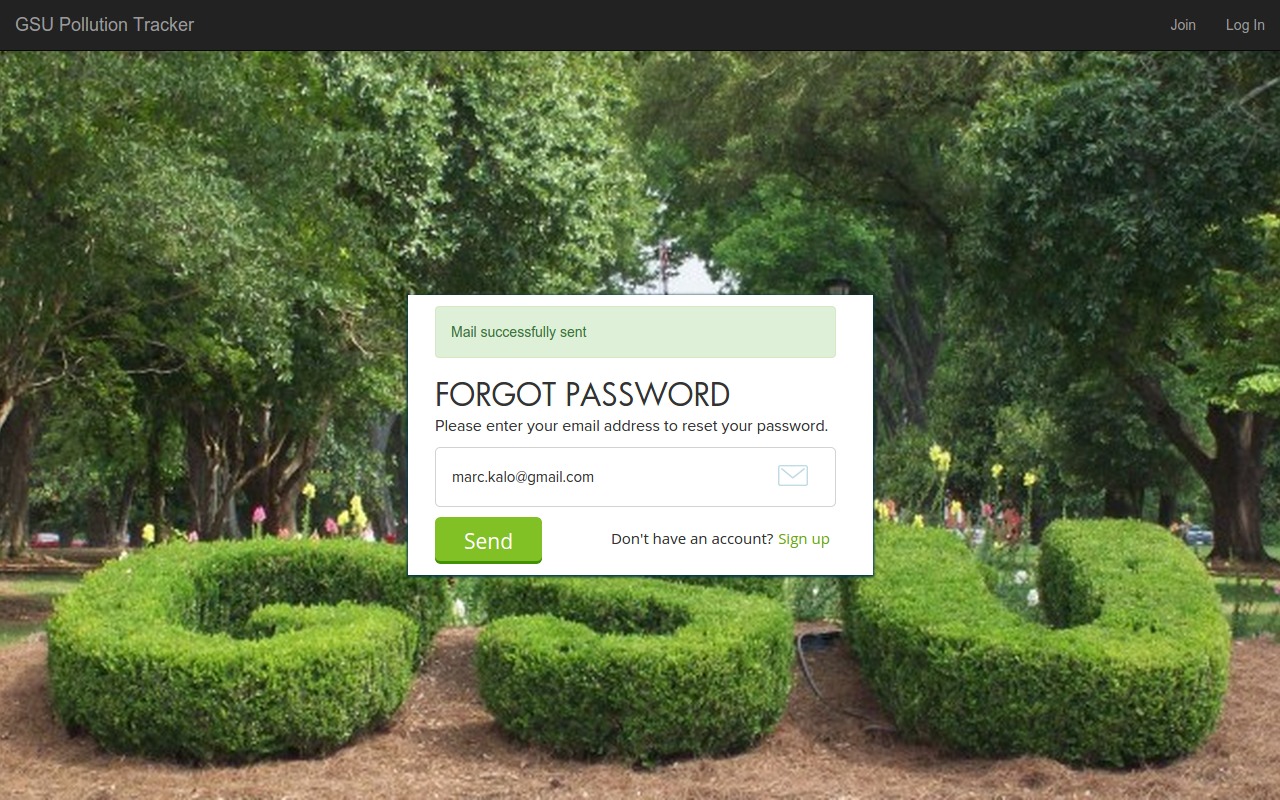
**User Login**

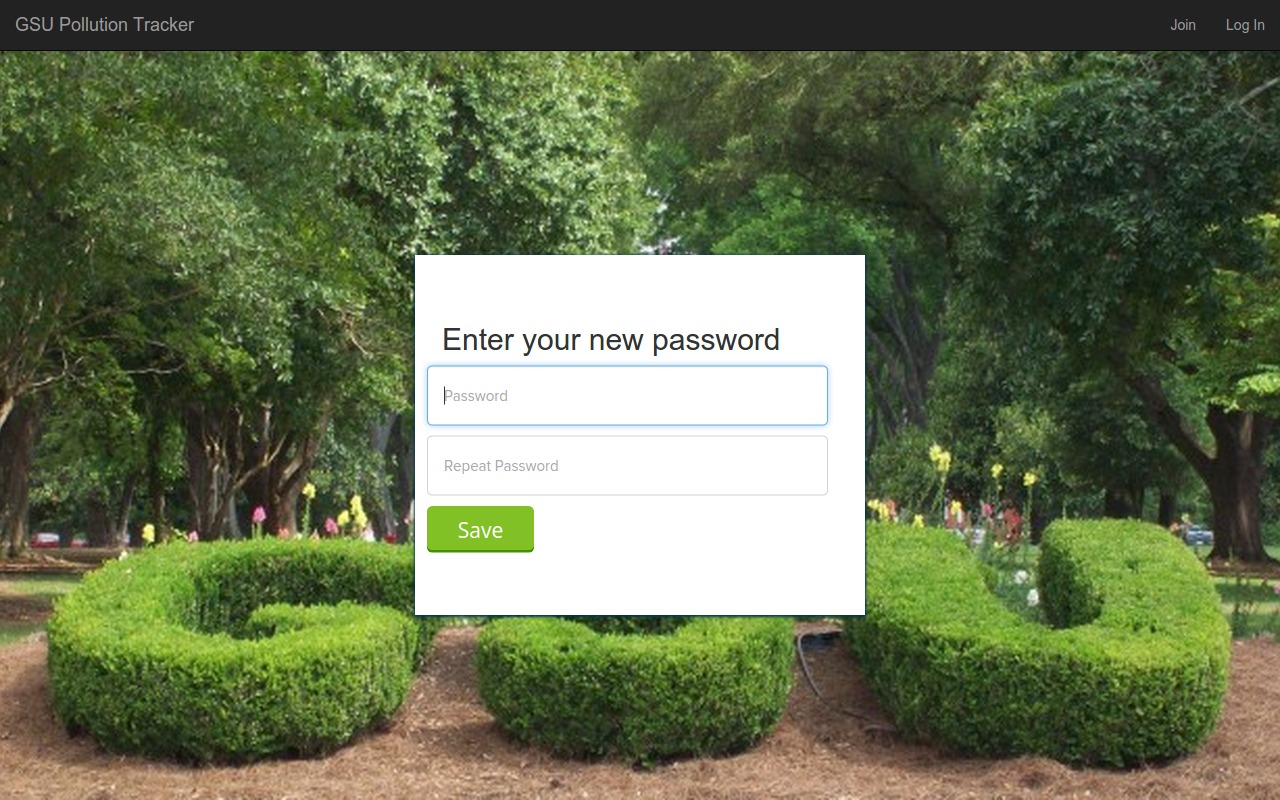
If a user already has an account, they can simply sign in using their email address and password:

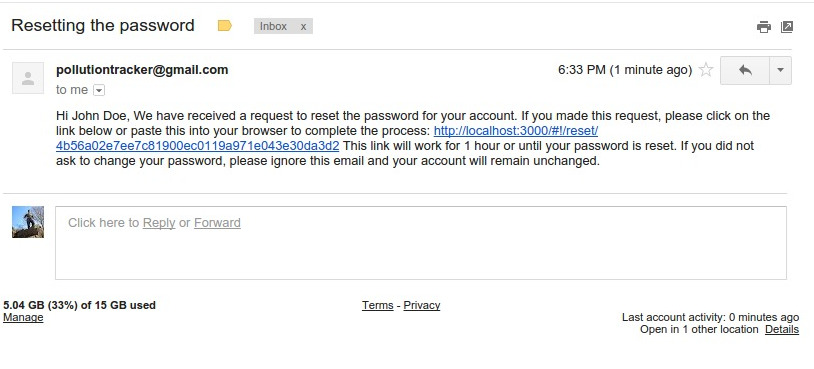


**Forget Password**

If a user forgets their password, the can use the “Forgot Password” link to email a secure, temporary link that the user can use to reset their password.

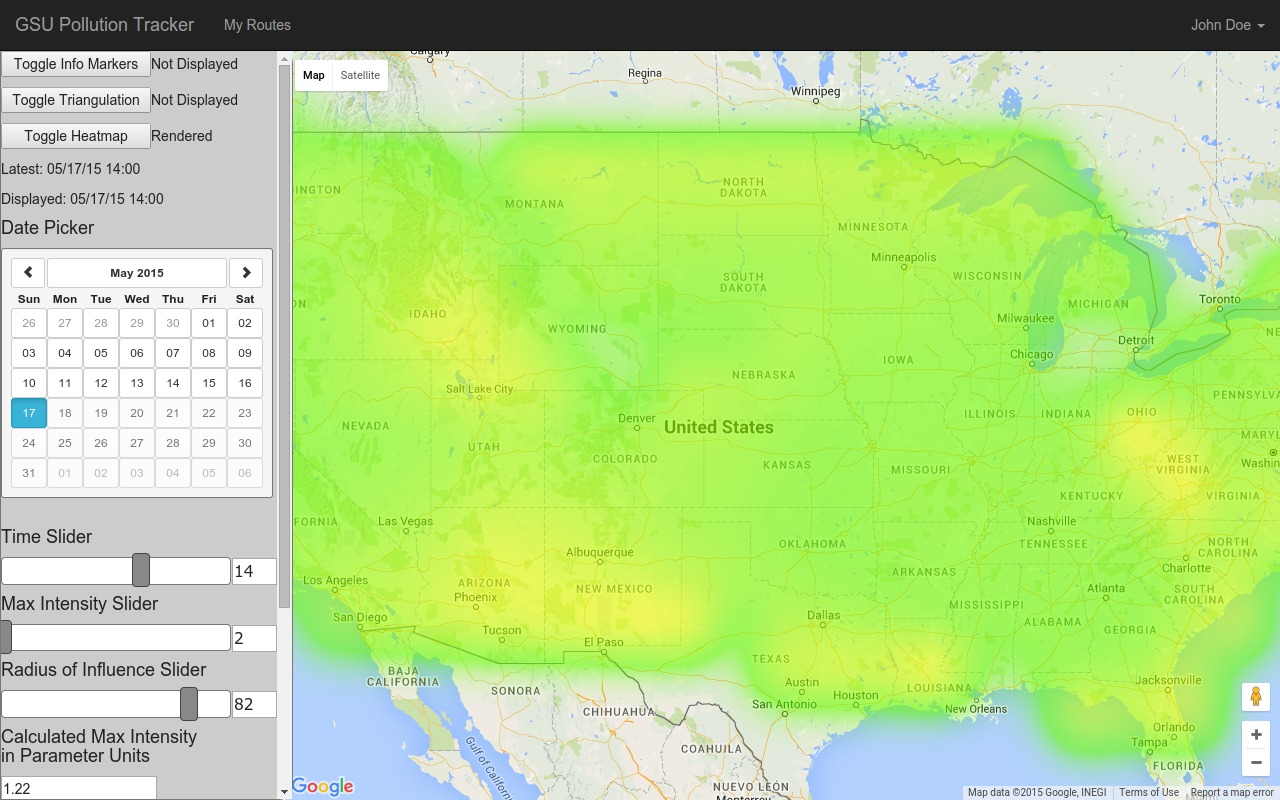




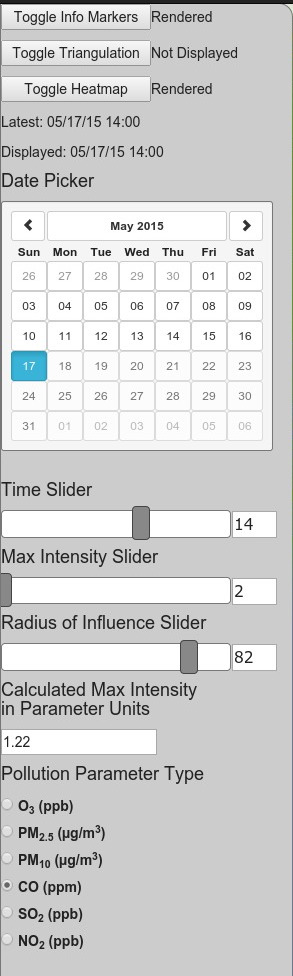


**Map Overview**

After successful log in, the user will see the following screen. The screen includes an options menu on the left and an embedded Google Maps applicaiton on the right. The Google Maps application is the main panel used for visualization of pollution data. When the user changes visualization options in the options menu, such as selecting the pollution parameter type, date, time, or visualization redering parameters, the data in the Google Maps application will be updated automatically and responsively rendered.



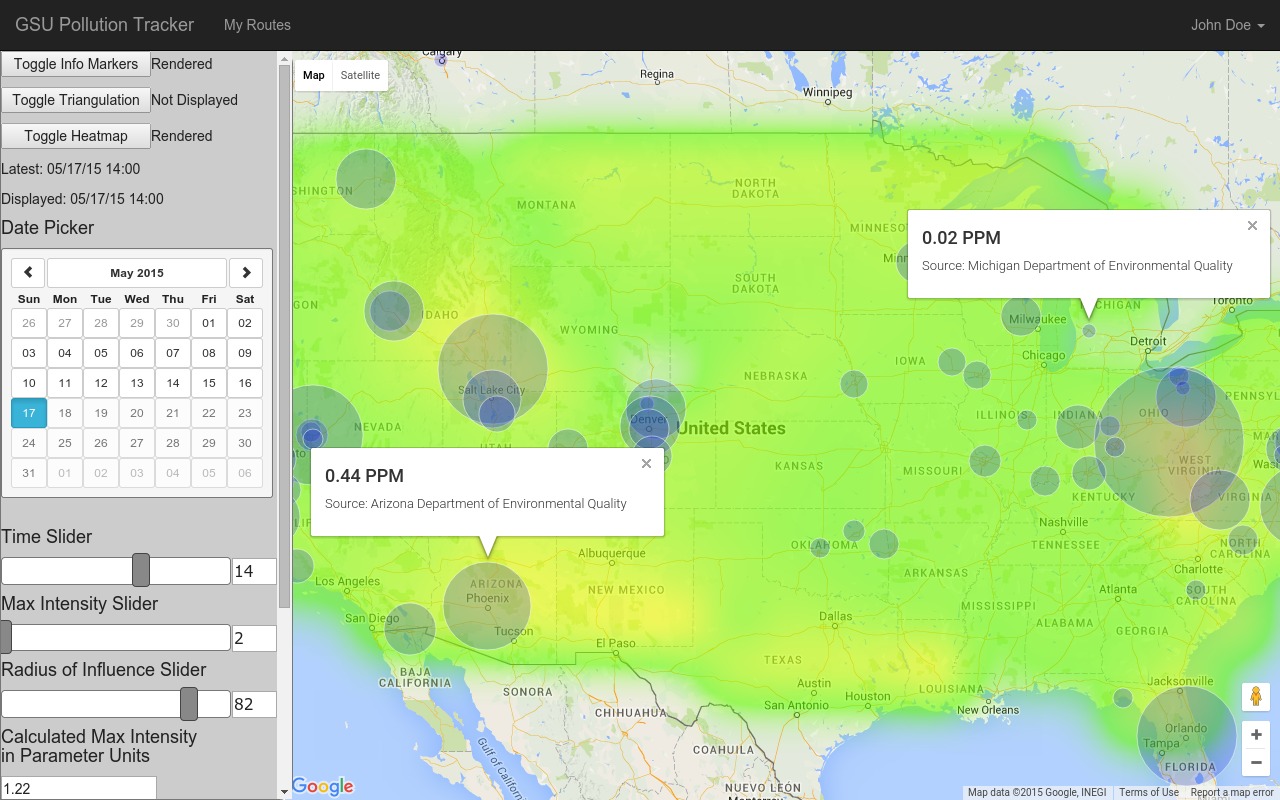
**Options Menu**

The options menu on the left of the map overview responsively controls the map visualization. Each option is discussed here along with screenshots illustrating the function.

**Toggle Info Markers**



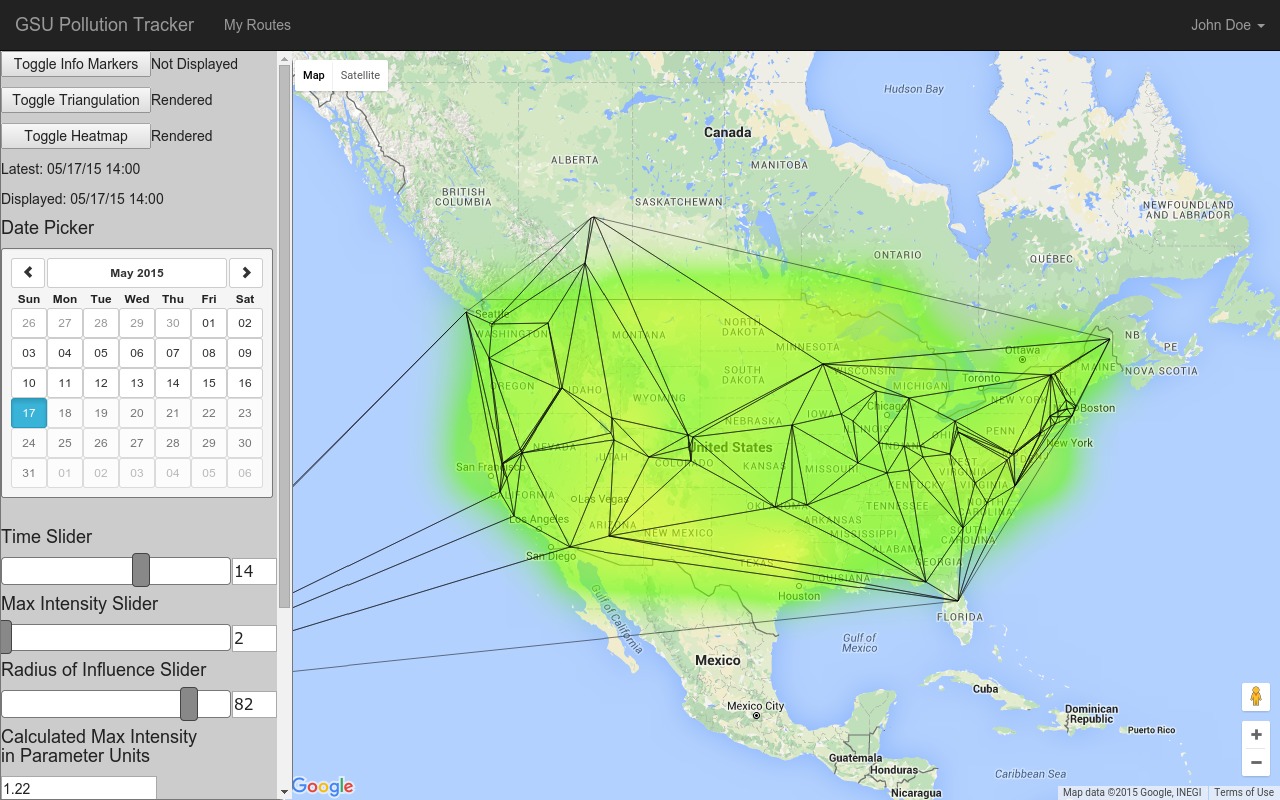
Info Markers are off by default. Clicking the “Toggle Info Markers” button will render markers on the screen at the location of each measured pollution value. The radius of the marker is larger where a larger reading is found, allowing a user to quickly visualize larger versus smaller readings. Clicking on an info marker will show a tag with the name of the institution which supplies the measurement as well as a numerical value of the measurement and unit.



**Toggle Triangulations**



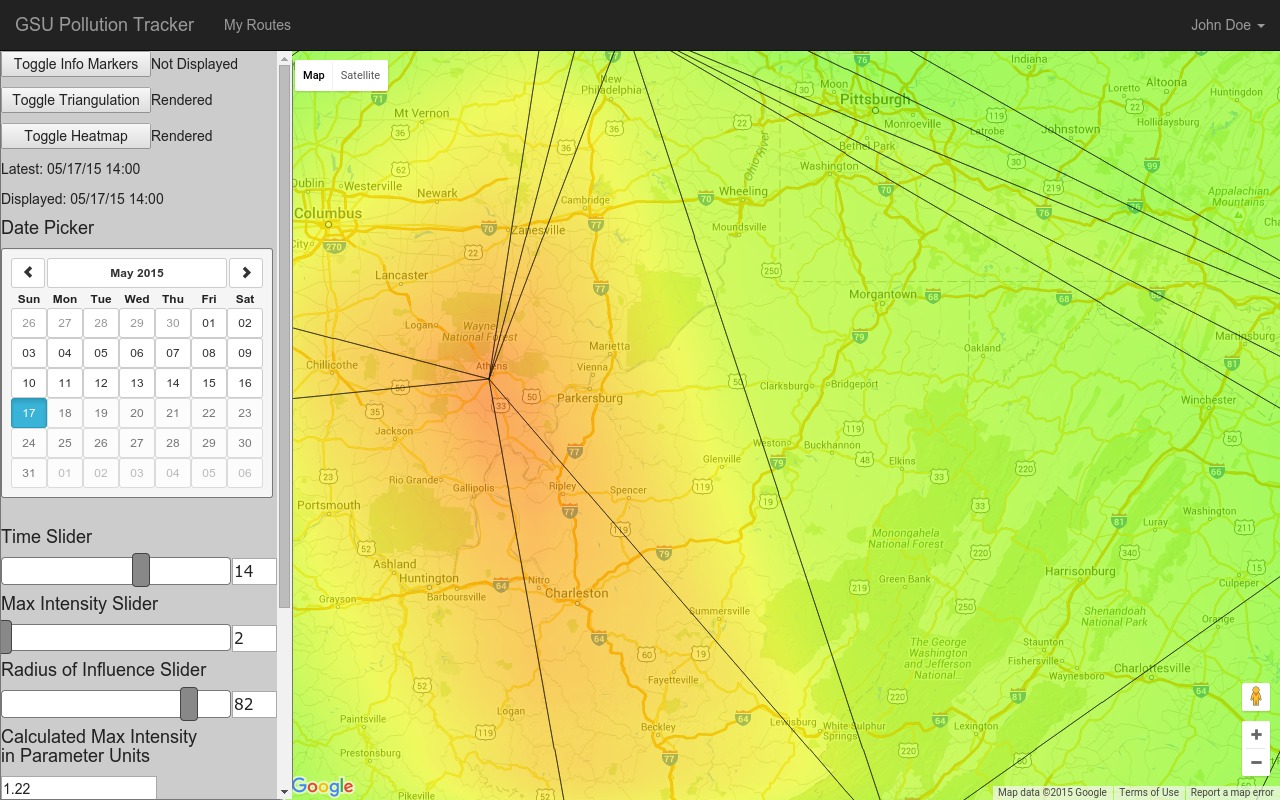
The triangulation visualization is off by default. Clicking the “Toggle Triangulation” button will render a visualization of the 2D delaunay triangulation which is stored in the database and utilized when interpolating the value of pollution parameters using the reduction method.



**Toggle Heatmap**



The heatmap is on by default. The heatmap can be toggled off by clicking the “Toggle Heatmap” button. The heatmap visualization is described in detail in the later section titled “Heatmap Visualization Technique”.

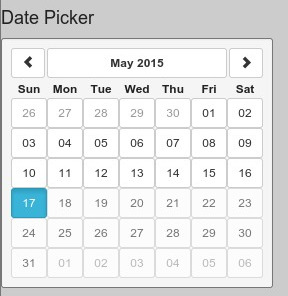


**Data Status**



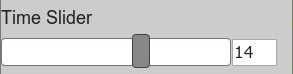
The data status section simply shows the date and time of the latest data available in the database and the date and time of the currently visualized data. This is shown in GMT time. The latest data will be automatically downloaded to the server assuming the Server Jobs have been properly configured. The displayed data can be adjusted using the Date Picker and the Time Slider.

**Date Picker**



The date picker allows a user to select the date where he wants to view the pollution data. The date picker is a dynamic widget that allows the user to browse days within a month, months within a year, and years. The days/months/years with valid data that can be visualized will be selectable. Days/months/years where no data is available will be grayed out. When a user selects a new date, the visualization will be automatically updated. The data status will update to show the selected date as “Displayed”.

**Time Slider**



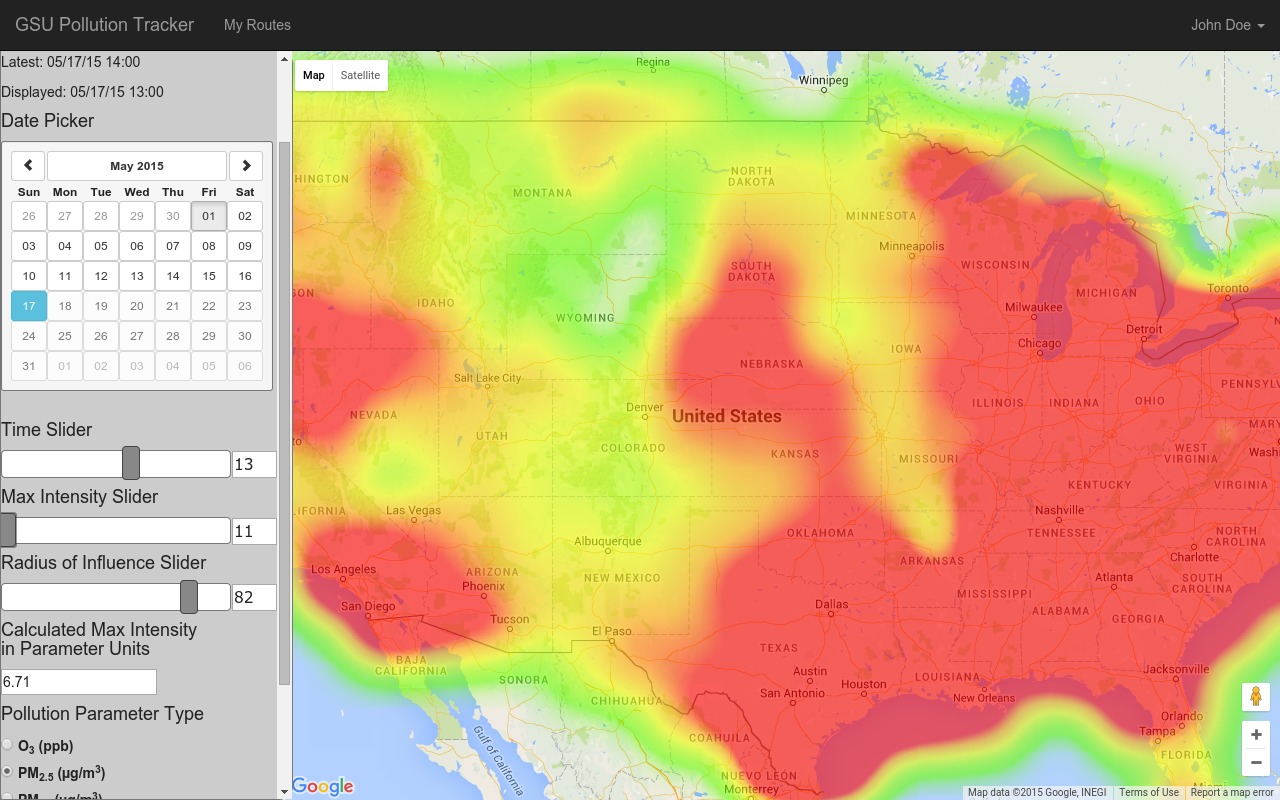
The time slider allows a user to select the time when he wants to view the pollution data. The values are 0 to 23 using a 24 hour clock. If no data is available for the selected hour on the selected day, an error message will be displayed above the time slider. When a user selects a new time, the visualization will be automatically updated. The data status will update to show the selected time as “Displayed”.

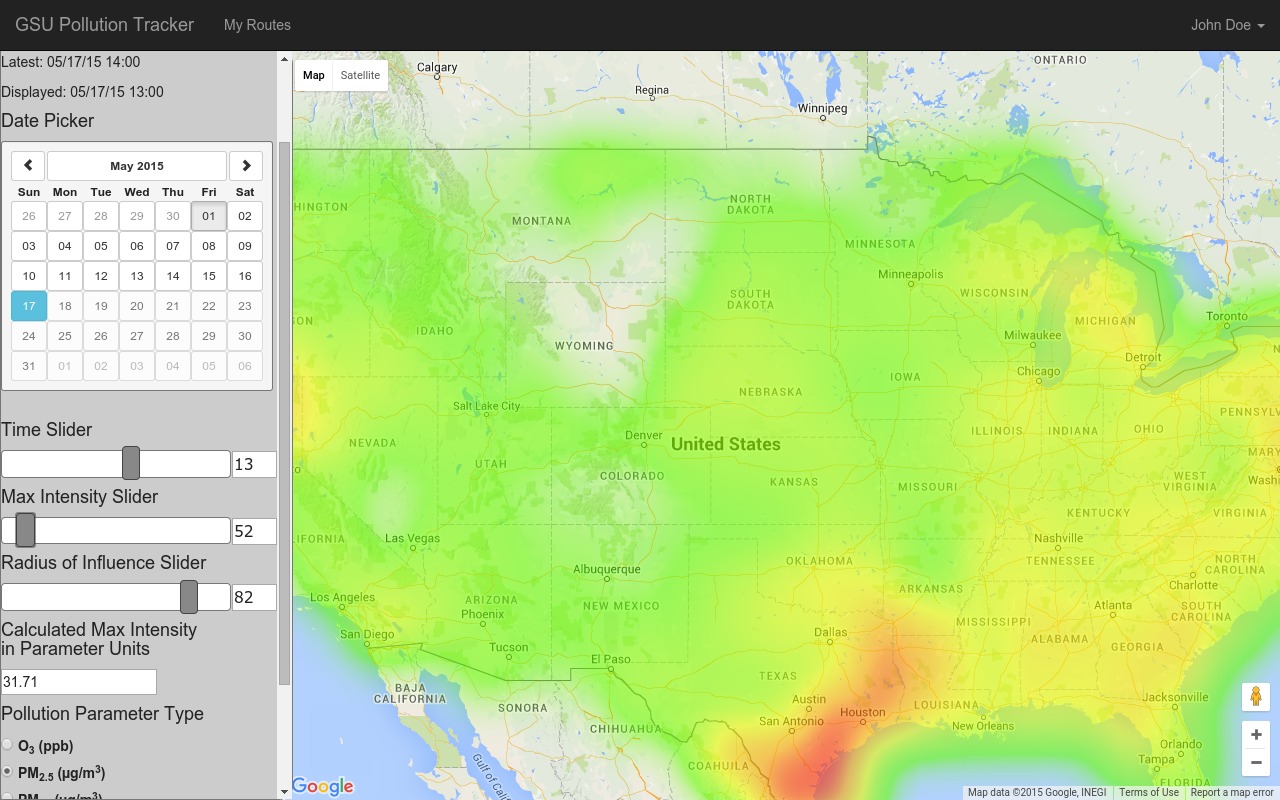
**Max Intensity Slider**

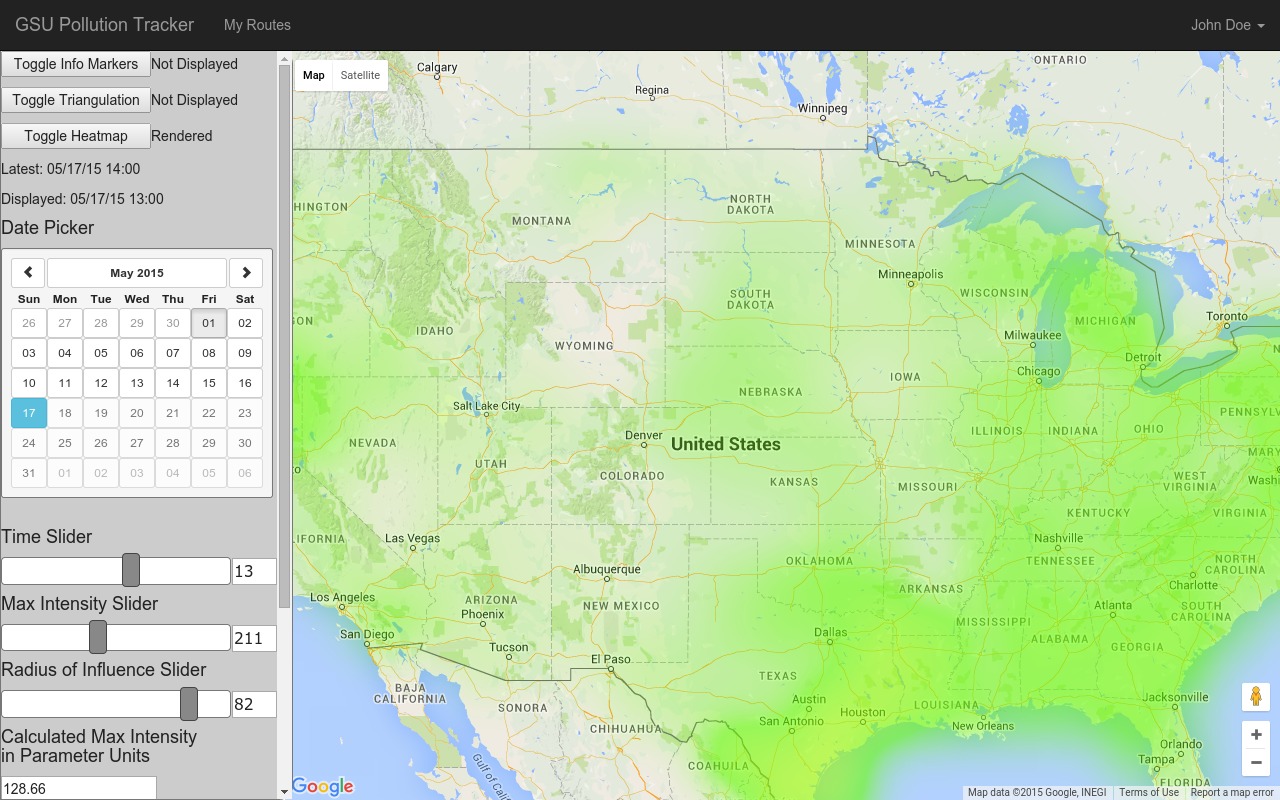


The max intensity slider is a visualization parameter that is passed to the Google Heatmap API. The heat map will render a gradient such that light green is a low value, yellow is a medium value, and red is a high value. The max intensity slider defines the maximum value that will be displayed on the map. Anything readings at or above the max intensity slider will be shown in red. Anything between 0 and the max intensity will be rendered on a linearly increasing scale. To illustrate this effect, the following three screenshots show the same data with low, medium, and high max intensity values. A value that is too low or too high may wash out useful visual gradients. A medium value can show contrast between areas but may cause the data to be misinterpreted. For example, if values across the country are relatively low then showing a certain area as red may mislead inhabitants of that area.

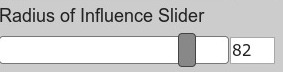
Max Intensity Low



Max Intensity Medium

Max Intensity High

**Radius of Influence Slider**

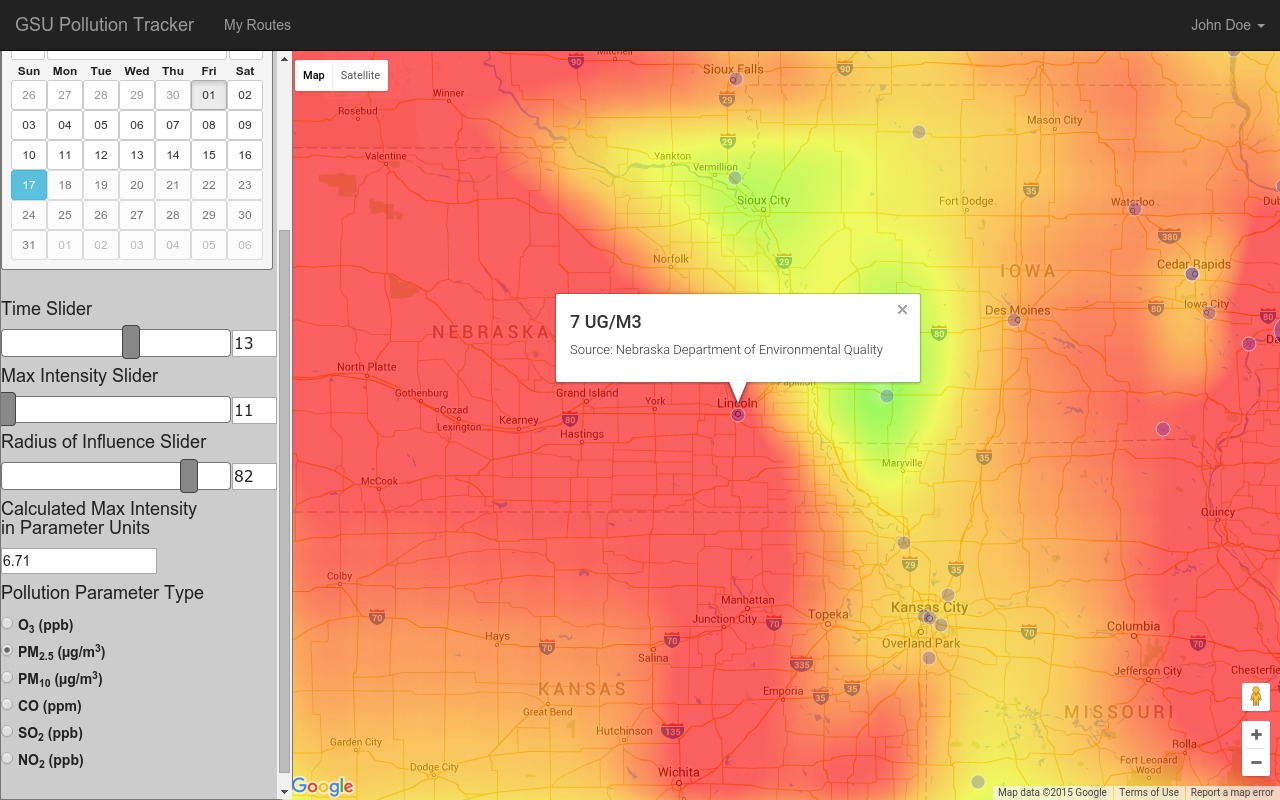


The radius of influence slider is described in more detail in under the section on Visualization technique. This slider behaves similar to other widgets in the options menu and will adjust the visualization in real time. To understand how the parameter affects the visualization, read the section on Visualization technique.

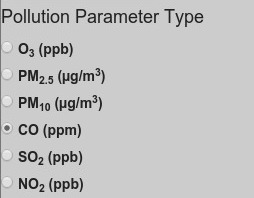
**Calculated Max Intensity in Parameter Units**



The “calculated max intensity in parameter units” is a text box that is updated based on the value of the Max Intensity Slider and the Radius of Influence slider. This box attempts to show the user the value of the pollution parameter which will show on the screen as the red color. It will typically show you the measurement you can expect see on the border of a red area. This value can be highly inaccurate if the max intensity is too low or the radius of influence is too high. It can be used as a guide but should not be trusted without validating that the settings maxes sense by sampling some values in the currently visualization. The followng example shows how the calculated max based on the sliders aligns with a measured point on the edge of a red area.



**Pollution Parameter Type**



The application allows a user to visualize 6 pollution parameters. The parameters are:

* O3 (ppb)
* PM2.5 (µg/m3)
* PM10 (µg/m3)
* CO (ppm)
* SO2 (ppb)
* NO2 (ppb)

Selecting the radio button for a parameter will automatically reload the visualization with the data related to that parameter.

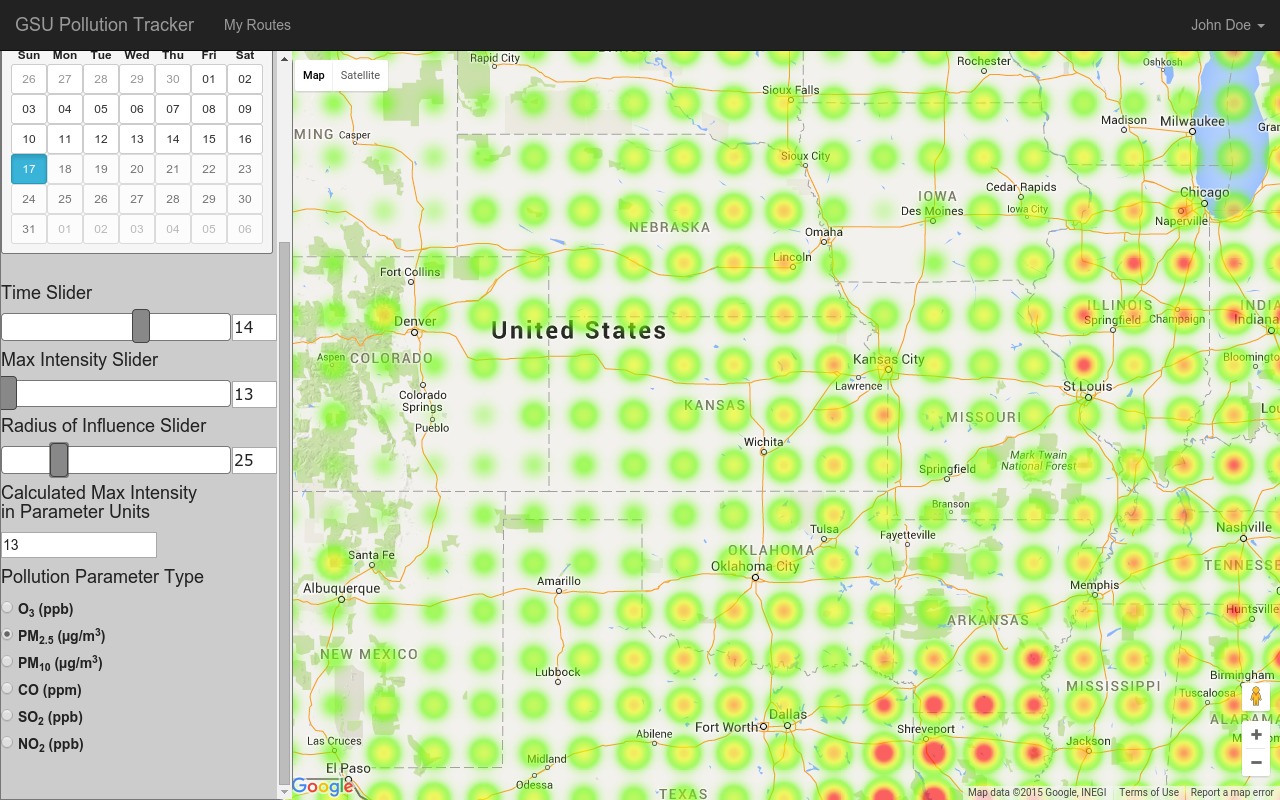
**Visualization Technique**

The heatmap is generated using the Google Heatmap API. By design, the google heatmap is a “density based” heatmap. This means that the color of the heatmap gradient is based only on the number of points that were uploaded in a particular region. Google Heatmap can also be rendered as a “weighted” heatmap. The weighted heatmap will allow a weight to be assigned to each point. The weight acts as a multiplier on the number of points in the that location. For example, if a weight of 5 is assigned, then the color of the gradient will render as if 5 points were clustered at that spot.

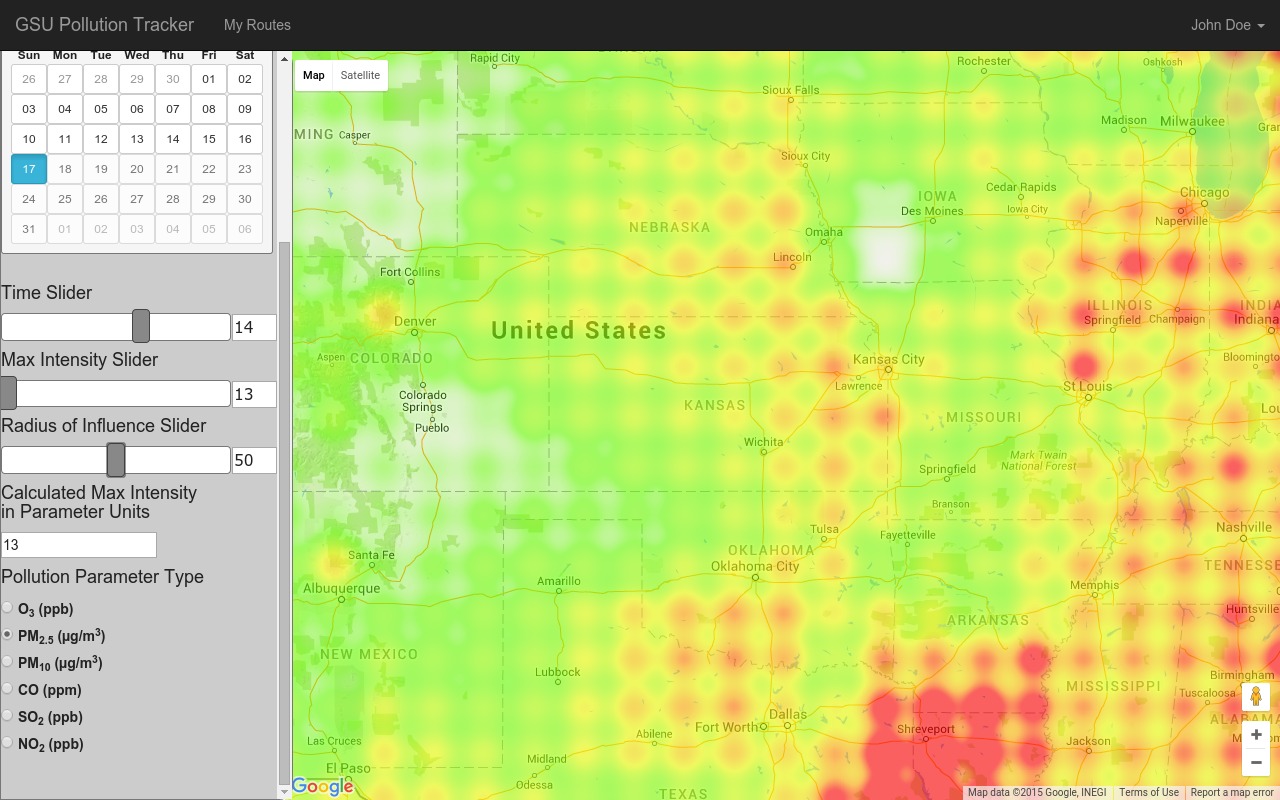
For visualizing pollution data, it is more appropriate to use a “value based” heatmap. A density based heatmap will show a higher value where more measurement locations are clustered. For example, cities will always show colors in the red because many measurement sites are typically available in a city, whereas rural areas will always display as green where few measurements are available. However,

a value based heatmap can be approximated using a density based heatmap along shape function interpolation.

To accomplish this, a regular grid of points is created across the map. The pollution value at each point is then interpolated. The interpolated value is then used to weight each point on the grid. We can visualize this grid by reducing the “Radius of Influence” slider to a small value. This will cause the Google Heatmap API to render the point in an unconnected way so we can see exactly which points were interpolated:



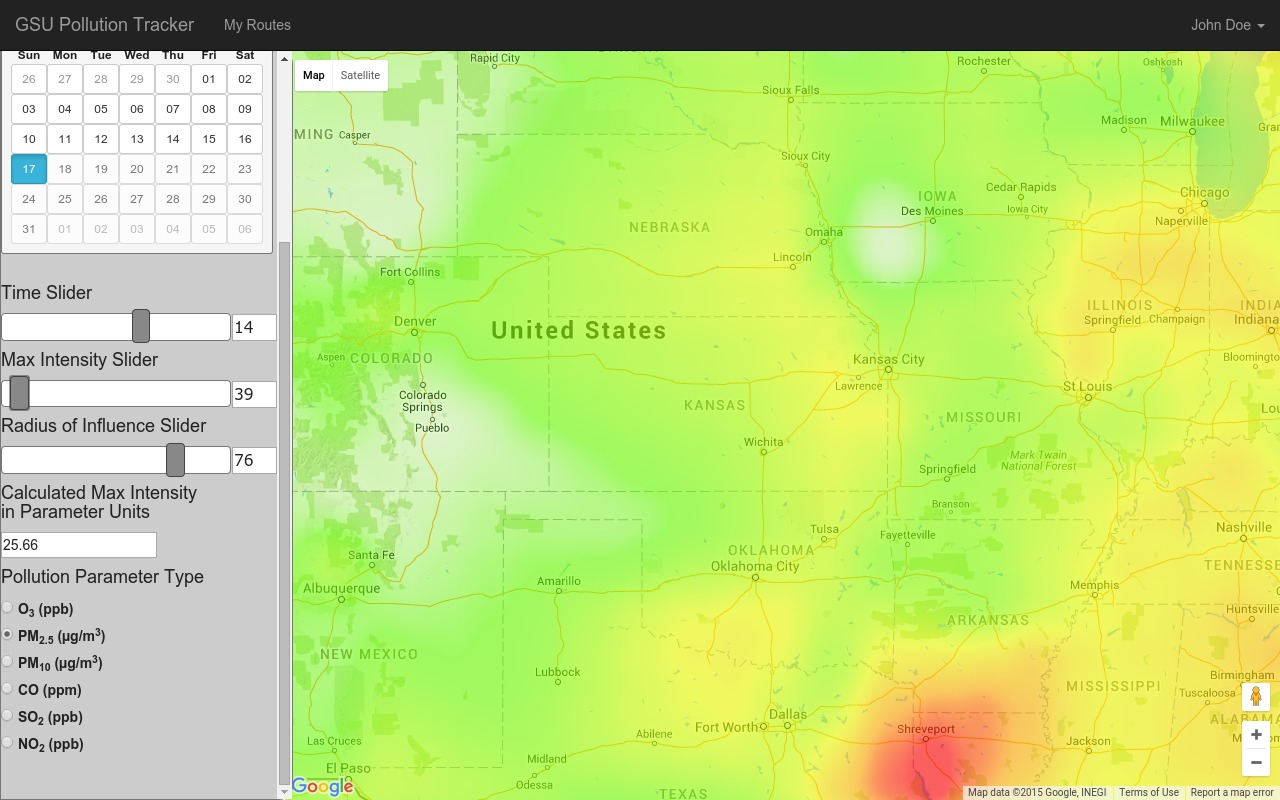
The grid is created such that the points have a 50 pixel wide spacing. When the Radius of Influence is increased to 50 (pixels) we can see that the points start to become connected, but are still rendered as distinct circles:



As the radius of influence is increased further, the visualization becomes more smooth, but it also inflates the influence of higher valued measurements:



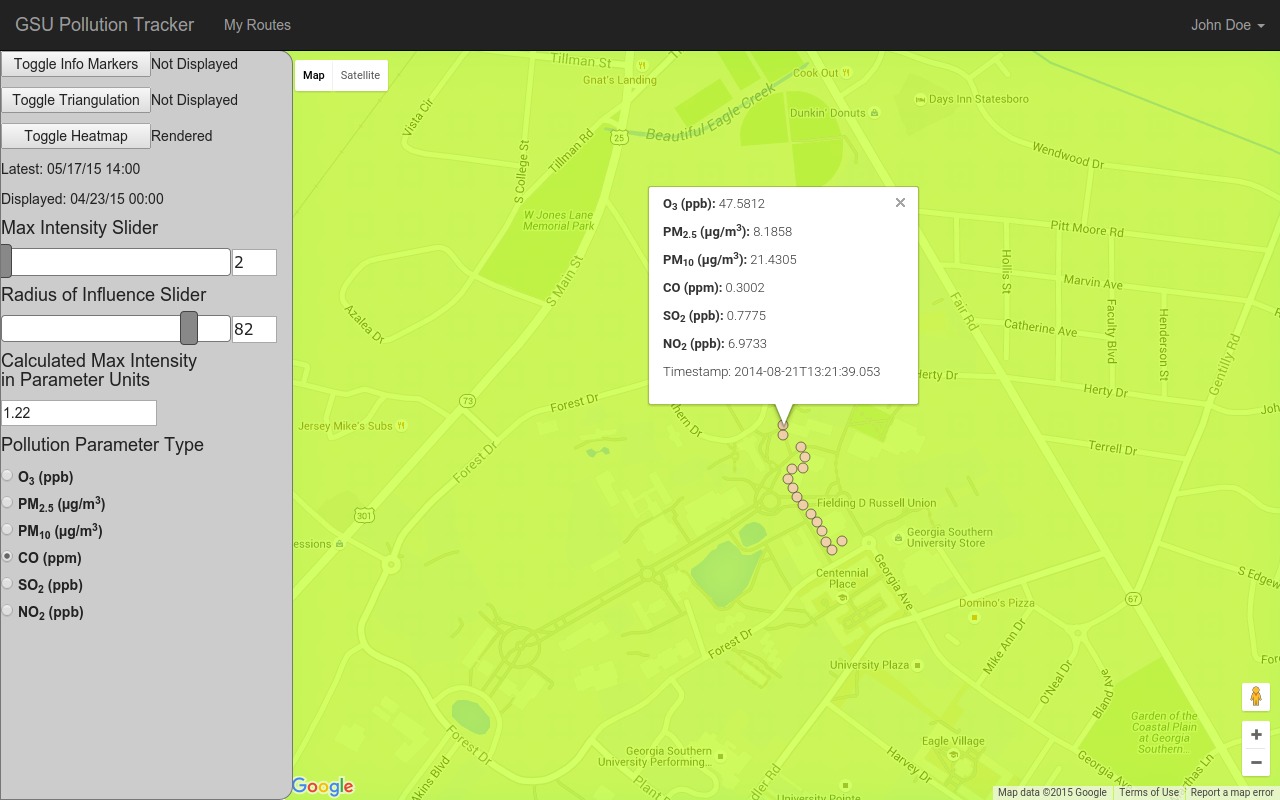
To correct this, the maximum intensity can also be increased:



The “Calculated Max Intensity in Parameter Units” estimates the maximum intensity value that will be rendered on the map by taking into account these two visualization adjustments.

**User Route Tracking**

The application additionally allows a user to upload a route, which is a collection of gps measurements over time. This data can be collected from an application using a GPS enabled device such as a smartphone. The User Route API section shows the interface to be used to upload GPS data. After the data is uploaded, the user can view their route by selecting the “My Routes” button on the toolbar and selecting the route that was created. The route is then visualized on the map. The map markers can be selected to show the interpolated value of the pollution parameters at the time and place of the GPS data that was uploaded.



**User Route API**

The following REST interface can be used by a smartphone application to upload data to the user account.

To create a new route, authenticate with the server by POSTING a message to  
   http://airquality.cloudapp.net/login  
  
   {"email":"myemail@gmail.com","password":"XXXxxXXX"}

If login is successful, then  
   GET http://airquality.cloudapp.net/loggedin  
should return your user information.  
  
Next, POST the following JSON format to  
   <http://airquality.cloudapp.net/myroutes/create>

The HTTP header for this request must contain:

Content-Type: application/json; charset=utf-8

This example shows two points but any number of points can be uploaded in this format.

{  
   "classifiedmode": [  
       "W",  
       "W"  
   ],  
   "timestamp": [  
       "2014-08-21T13:20:44.039",  
       "2014-08-21T13:21:39.053"  
   ],  
   "points": {  
       "type": "MultiPoint",  
       "coordinates": [  
           [  
               32.4263451,  
               -81.7817563  
           ],  
           [  
               32.426154,  
               -81.7817446  
           ]  
       ]  
   }  
}

The server will respond with the JSON of a complete object stored in the MongoDB  
   complete with "created" timestamp and "user" fields (user automatically populated  
   based on the user that was authenticated)

**Deployment**

This section will discuss deployment details. Cloud deployment is ideal for this application. The application can run on a single server which can be deployed for development, test, and demo purposes. Since this is designed as a web application, it is important to be able to scale resources fluidly when the site is deployed in a production environment. Additionally, since the server functions as a data warehouse, its storage needs will continue to grow over time. Cloud deployment services such as Amazon Web Services and Microsoft Azure allow incremental processing power to be added to the application on-demand.

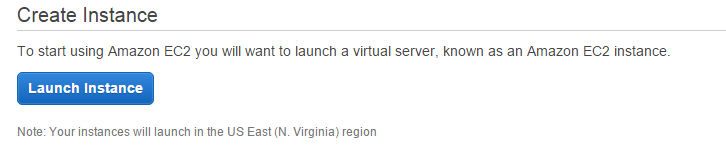
Early in the development process, Microsoft Azure was used to create an Ubuntu 14.04 LTS instance. This worked fine for the application which started out as environment-agnostic – however, it was discussed that an academic grant could be available to host the application in Amazon Web Services (AWS). Therefore, an effort was made to relaunch the server using AWS.

**Cloud Deployment**

First, a server needs to be provisioned in AWS. The AWS offering known as EC2 (Elastic Cloud Compute) can be used to provision a virtual machine. AWS can be accessed at:

<https://aws.amazon.com/>

After signing up for AWS and logging into the Management Console, you can access the EC2 dashboard. From the EC2 Dashboard, click “Launch Instance” to get started:



It is important to note in which region you are launching your instance. If you log into a second region, you will not be able to see any instances you have running in the first region.

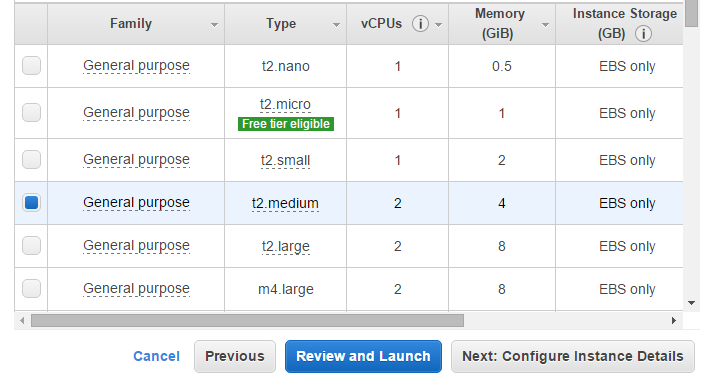
Two options are available to deploy a new server. One option is to deploy a standard Ubuntu AMI and perform the build process. This is explained first to give an overview of the build process. A more streamlined deployment is also available and is explained in a later section. The streamlined deployment uses a publicly available AMI that is already set up with all of the necessary software and offers an accelerated method for deploying new server instances.

**Manual Deployment Process**

Select the following Amazon Machine Image (Ubuntu 14.04 LTS 64-bit):



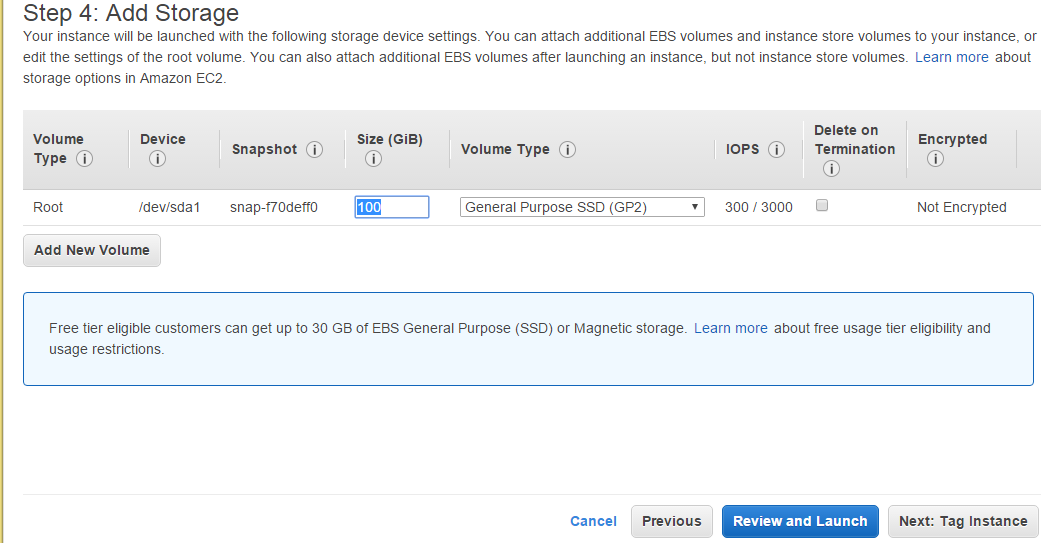
For a single server build, a t2.medium instance should be sufficient to run the application for development, test, and demo purposes. This instance includes 2 virtual CPUs and 4GB of memory. Select the appropriately sized instance, then click “Next: Configure Instance Details”.



There default settings should be appropriate for a single server deployment. You may want to consider creating a new Virtual Private Cloud (VPC) at this point. A VPC will allow several server instances to act as if they are connected to the same physical network. Network traffic between instances on the same VPC are more efficient and incur fewer data charges. Since we are walking through a single server deployment, just click “Next: Add Storage”.

Next, we should attach a disk to the instance. A default AMI does not have a persistent disk. A disk needs to be provisioned using Amazon Elastic Block Storage (EBS) primarily to store the MongoDB data. A 100GB disk should be sufficient to allow the server to run for several weeks on a development machine. A scalable architecture is discussed later in this section which will allow the storage space to grow as needed over time without interruption to the server.

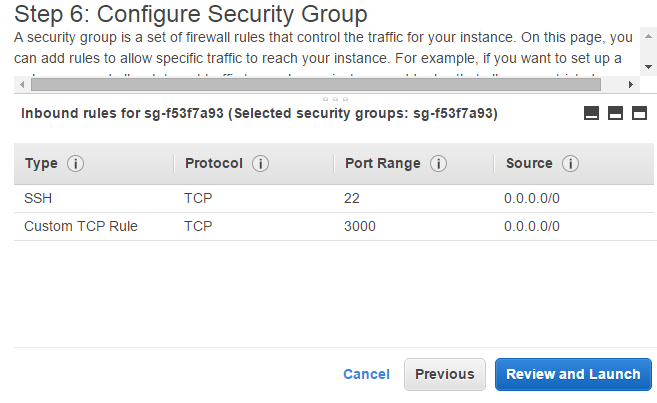
Add a new volume with the following settings. Click “Add New Volume”, then click “Next: Tag Instance”.



In the Tag Instance settings, you can add key/value pairs to describe your instance. Add key/value pair with a descriptive name, then click “Next: Configure Security Group”.

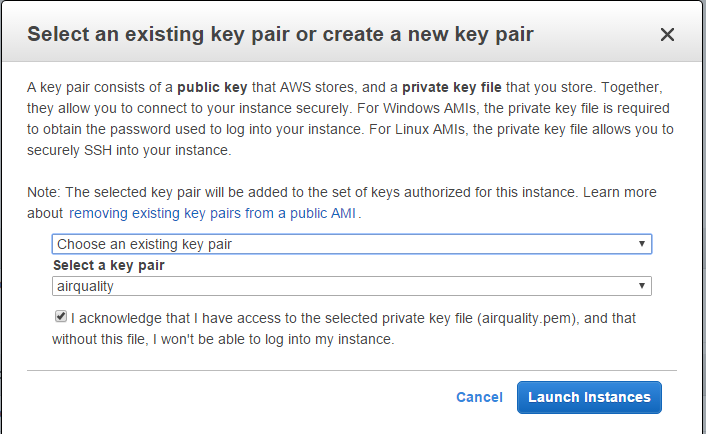


For development purposes, you can configure a security group to allow access to TCP over Port 22 for SSH connections as well as TCP over Port 3000 to test the web interface. The application will default to port 3000. Details of production deployment including routing of HTTP traffic through Port 80 is discussed later in this section.



It is highly recommended that the Source IP addresses be restricted on a development machine. When finished adding rules, click “Review and Launch”.

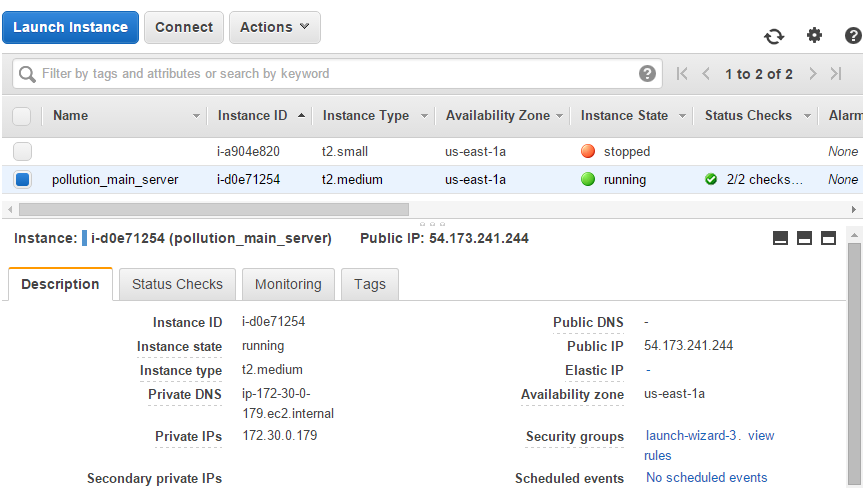
AWS will then prompt you to create a security key pair or select an existing pair. Make sure that a key pair is selected with a descriptive name, then downloaded the .PEM file if creating a new key or select your existing .PEM file. Store the .PEM file in a secure location as it is necessary to log into your created instance.



Click “Launch Instances” to continue.

If you are experimenting with instances then it is recommended to set up billing alerts at this point. AWS charges for a number of services “cafeteria style”. You only pay for what you use, but the exact cost of running a server is nontrivial to calculate as it is based on instance size, provisioned storage, and network traffic. If security settings are not restricted to your source IP address, it is likely that data charges could grow beyond expectation if someone finds tries to brute force their way into your server. At the very least, it is recommended to set an Alarm where the EstimatedCharges metric is greater than or equal to your budget for the development work.

Your instance will take a few minutes to be provisioned and to boot. You can check the status of the new instance in the EC2 Dashboard. When the instance state is “Running” you can proceed.



Note the public IP address that is shown in the status window.

Next, go to the VPC management console, select the VPC containing your server, click on Actions, select Edit DNS Hostnames and select Yes.

Using an ubuntu dash shell, you can connect to the instance using this command:

chmod 600 my.pem

ssh -X -i "my.pem" ubuntu@54.173.241.244

After successful login to the remote shell, bootstrap the install scripts and run them. The script shown below illustrates the steps. It is recommended to run these manually and inspect the logged output after each step to ensure that packages are installing as expected. Package management using npm (for node.js) and bower (for angularJS) can be tricky when working with legacy package versions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Initial Server Config

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#copy install\_script to a fresh Ubuntu installation and run it to set up the environment

wget <https://raw.githubusercontent.com/kalomj/Pollution/master/serverbuild/install_script>

#convert line endings

sed -i 's/\r//' install\_script

#test sudo

ls

#if you get an error with sudo about hostnames, login to AWS, go to the VPC console,

#select the VPC of your server, select actions, select edit dns hostames, select yes

#change permissions

chmod 755 install\_script

#run and save output to log file

./install\_script "First Last" "email@site.com" > out.log 2>&1

#restart your login window now to make sure necessary scripts are sourced.

#you may need to run the following from the location /home/$(whoami)/airquality

#to clear out packages inconsistent with the tracked project

git stash

cd /home/$(whoami)/airquality/serverbuild

#you will need to to set execute permissions

chmod 755 /home/$(whoami)/airquality/serverbuild/airnow\_setup

chmod 755 /home/$(whoami)/airquality/serverbuild/mongo\_setup\_script

chmod 755 /home/$(whoami)/airquality/serverbuild/dthp\_script

chmod 755 /home/$(whoami)/airquality/serverbuild/move\_mongo\_script

chmod 755 /home/$(whoami)/airquality/serverbuild/enable\_https

chmod 755 /home/$(whoami)/airquality/serverbuild/upgrade\_wiredtiger

#to setup lftp to download from teh airnow api, run the following (where username and password correspond to an Airnow API account)

./airnow\_setup username password

#to setup mongodb on a new disk, run:

#./mongo\_setup\_script device\_name (where device name is like sdb)

#or on an existing disk

./mongo\_setup\_script NONE device\_name (where device name is like sda1)

#to disable transparent huge pages for mongodb performance, run

./dthp\_script

#upgrade to the wired tiger mongo storage engine

./upgrade\_wiredtiger /home/$(whoami)/mongo

#to enable https on port 3001, run this script to generate a key/cert

#and set environment variables

#you may need to stop the http server and log out/log in after running this script

./enable\_https

#use the following to stop the server daemon from the location

#/home/$(whoami)/airquality

grunt forever:server:stop

#use the following to start the server daemon from the location

#/home/$(whoami)/airquality

grunt forever:server:start

#use the following to tail the log file when running as a daemon

tail -f /home/$(whoami)/airquality/logs/log.log

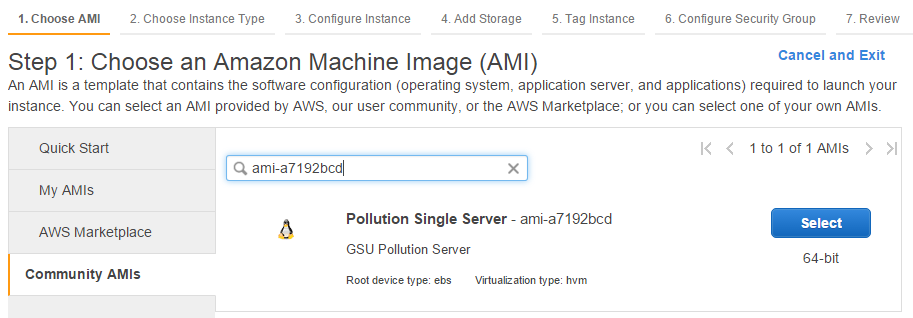
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/Initial Server Config

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Streamlined Deployment Process**

A prebuilt server is available publicly on AWS. To deploy this server, follow the instructions above to log into AWS and launch a new AMI. Instead of selecting a generic Ubuntu 14.04 LTS AMI after clicking “Launch Instance”, you will need to search the Community AMIs for the prebuilt server. Click the “Community AMIs” button in the left panel, and search for “ami-a7192bcd”.



Select this AMI and deploy as described above. However, after logging into the server, it is not necessary to bootstrap the installation scripts. Simply run the command to start the server daemon:

cd /home/$(whoami)/airquality

git pull origin master

grunt forever:server:start

To ensure the database is initially populated with data, run the following command after the server has started:

./reload

**Google Maps API License Key**

Google Maps requires an API License key to be used. A key can be obtained by logging in to the Google Developers Console at:

<https://console.developers.google.com/>

You must enable a key for “Google Maps Javascript API”.

A key will only accept requests from known referrers. You need to configure your key to accept requests from the IP address or DNS hostname of the server you have deployed.

The key needs to be injected into your project. To inject the key, open the file located at:

/home/$(whoami)/airquality/packages/custom/pollution/server/views/includes/head.html

Source your key in place of the existing key at line 22:

<script src='//maps.googleapis.com/maps/api/js?libraries=visualization,geometry&key=AIzaSyDY0QbwfRK0j0Fz6IUISTBOAmEe2yHeQPM'></script>

**Automated Gmail Notifications**

The application provides support for password resets using Gmail. To configure a Gmail account, open the file located at:

/home/$(whoami)/airquality/config/env/development.js

Edit the emailFrom and mailer properties at lines 39 through 44 to configure the desired Gmail account.

Note that if you edit the settings in place, you must configure your Gmail account settings to “Allow less secure apps”.

The application uses nodemailer to handle the email feature. You can configure the mailer to use a more secure OAuth token. There are plenty of tutorials with instructions on making this work with your Gmail account.

**HTTPS**

The application includes built in support for HTTPS. You must run the “enable\_https” script in the following location:

/home/$(whoami)/airquality/serverbuild

Note that you must configure your AWS security settings to forward port 3001 after running this script. You must also update your Google Maps API key to allow access from the HTTPS protocol.

As an alternative to native HTTPS support, you can configure an AWS load balancer as an HTTPS endpoint. The load balancer can then forward HTTP traffic within your VPC. In this arrangement, the server itself uses HTTP but the user connects using HTTPS.

**Listening Ports**

By default, the application listens on port 3000. To access the server via a web browser, note the public IP address mapped to your instance. Type the following address into the address bar:

<http://nnn.nnn.nnn.nnn:3000>

where nnn.nnn.nnn.nnn is your public IP address.

If you enable HTTPS, this listens on port 3001. The address for HTTPS would be:

<https://nnn.nnn.nnn.nnn:3001>

You can use Amazon Route 53 to assign a domain name to your instance and register the route with a DNS.

The server needs to be configured to forward port 80 for HTTP and port 443 for HTTPS to avoid needing the port qualification in the web address.

**References**

[M1]

MongoDB. https://www.mongodb.org/.

[M2]

Express. <http://expressjs.com/>.

[M3]

Node.js. https://nodejs.org/.

[M4]

AngularJS. <https://angularjs.org/>.

[M5]

MEAN.IO. http://mean.io/.

[M6]

Passport. <http://passportjs.org/>.

[M7]

Web Application GitHub Project. <https://github.com/kalomj/Pollution>.

[M8]

Ubuntu 14.04 LTS Trusty Tahr Release. <http://releases.ubuntu.com/14.04/>.

[M9]

AirNow Developer Tools. <http://airnowapi.org/>.

[M10]

AirNow Web Services Documentation. <http://airnowapi.org/webservices>.

[M11]

LFTP Documentation. <http://lftp.yar.ru/>.