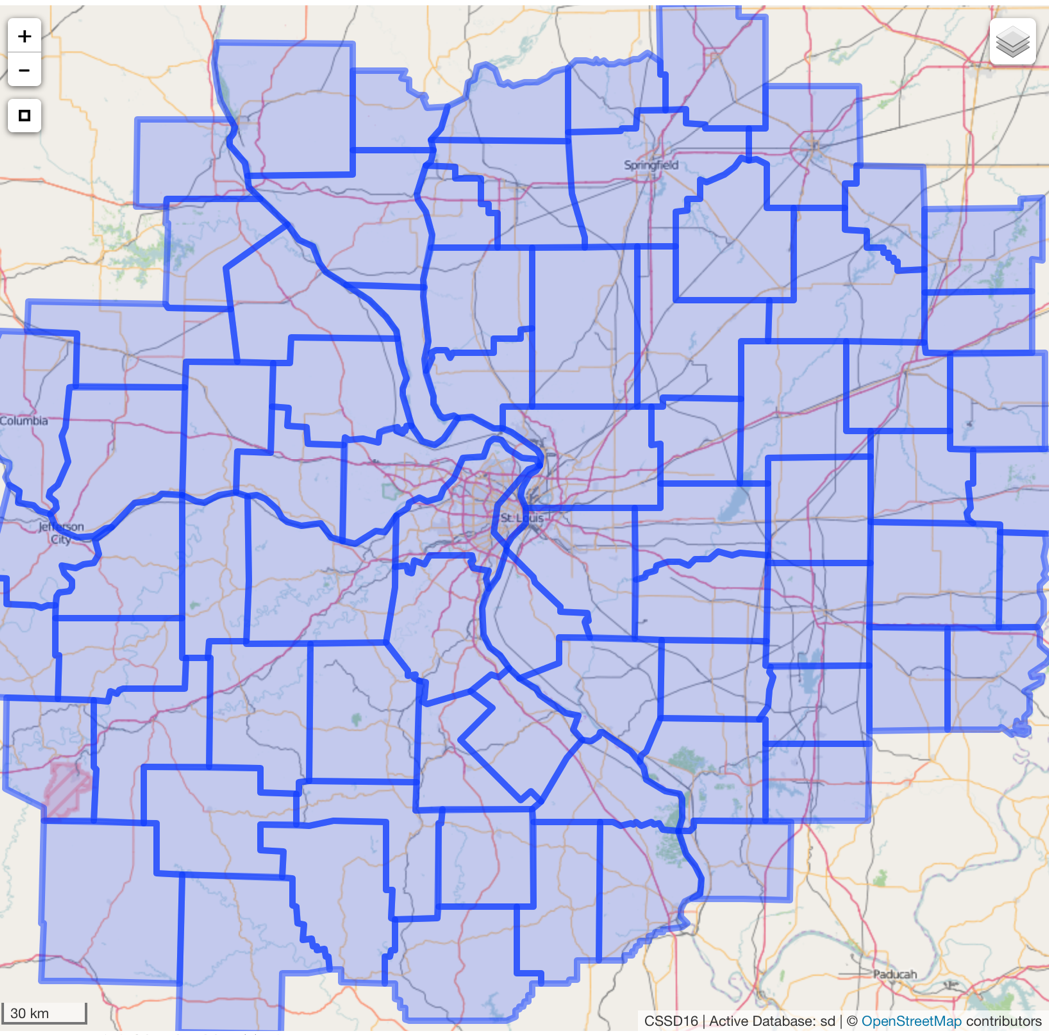
Ben Carleton  
SD Design 2

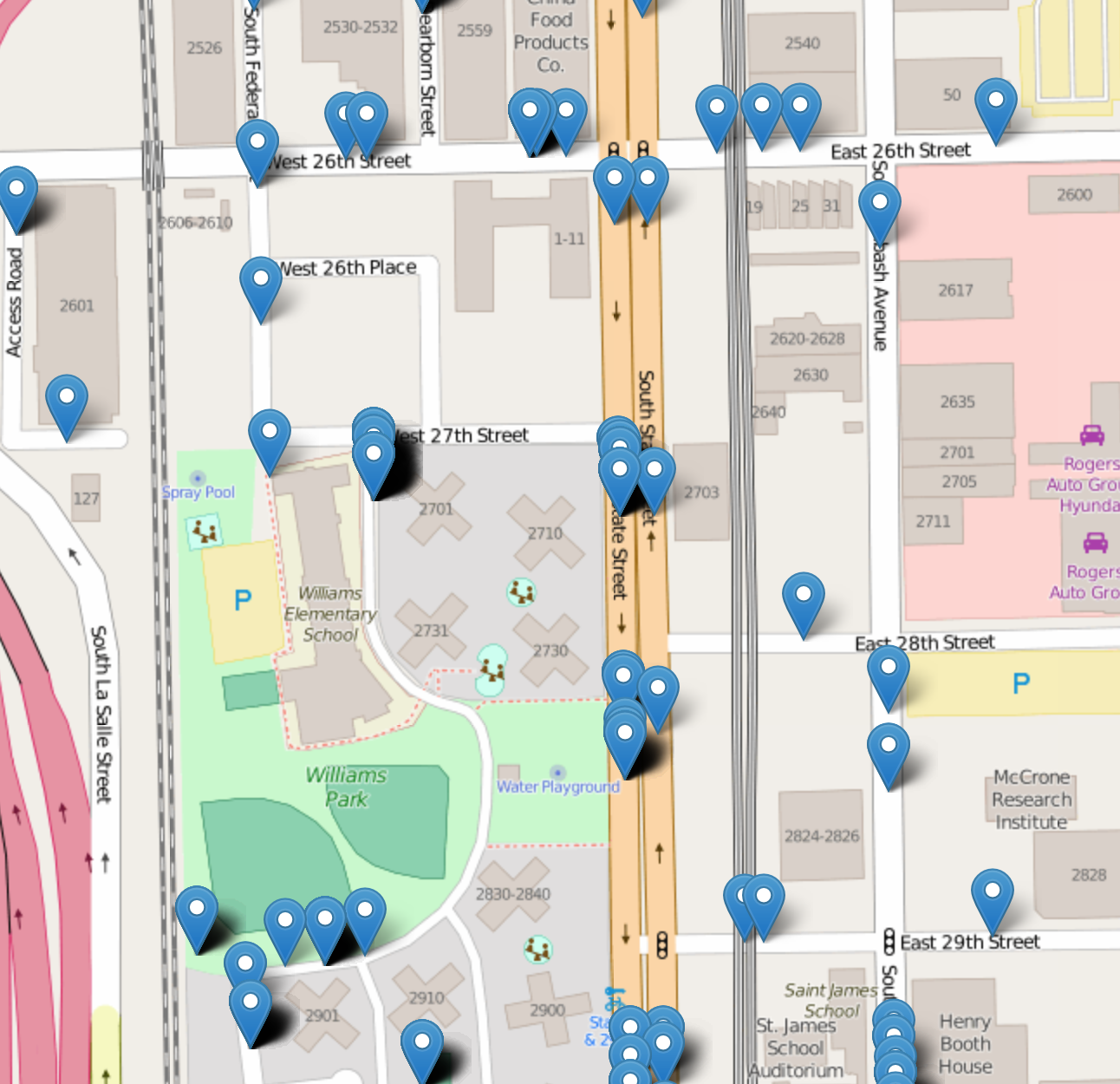
**Graphical User Interface - Overview**

The user interface will consist primarily of the mapping interface and a table-based reporting interface. The mapping interface will display a number of layers on an OpenStreetMap base layer, including individual incidents and the results of analysis methods. The map, which will offer full pan and zoom controls, will dynamically redraw the incident objects as well as the analysis layers in response to changes in the map’s active bounding box. The user will be able to interact with the incident and analysis layers to view the details of a given object, if available. The map will also include a layer controller which will allow the user to control what is being displayed on the map.

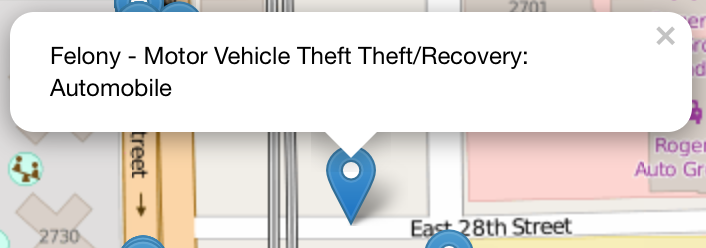
Mapping UI screenshots:



*Figure 1: the mapping UI zoomed out, displaying a placeholder analysis result on the county level*

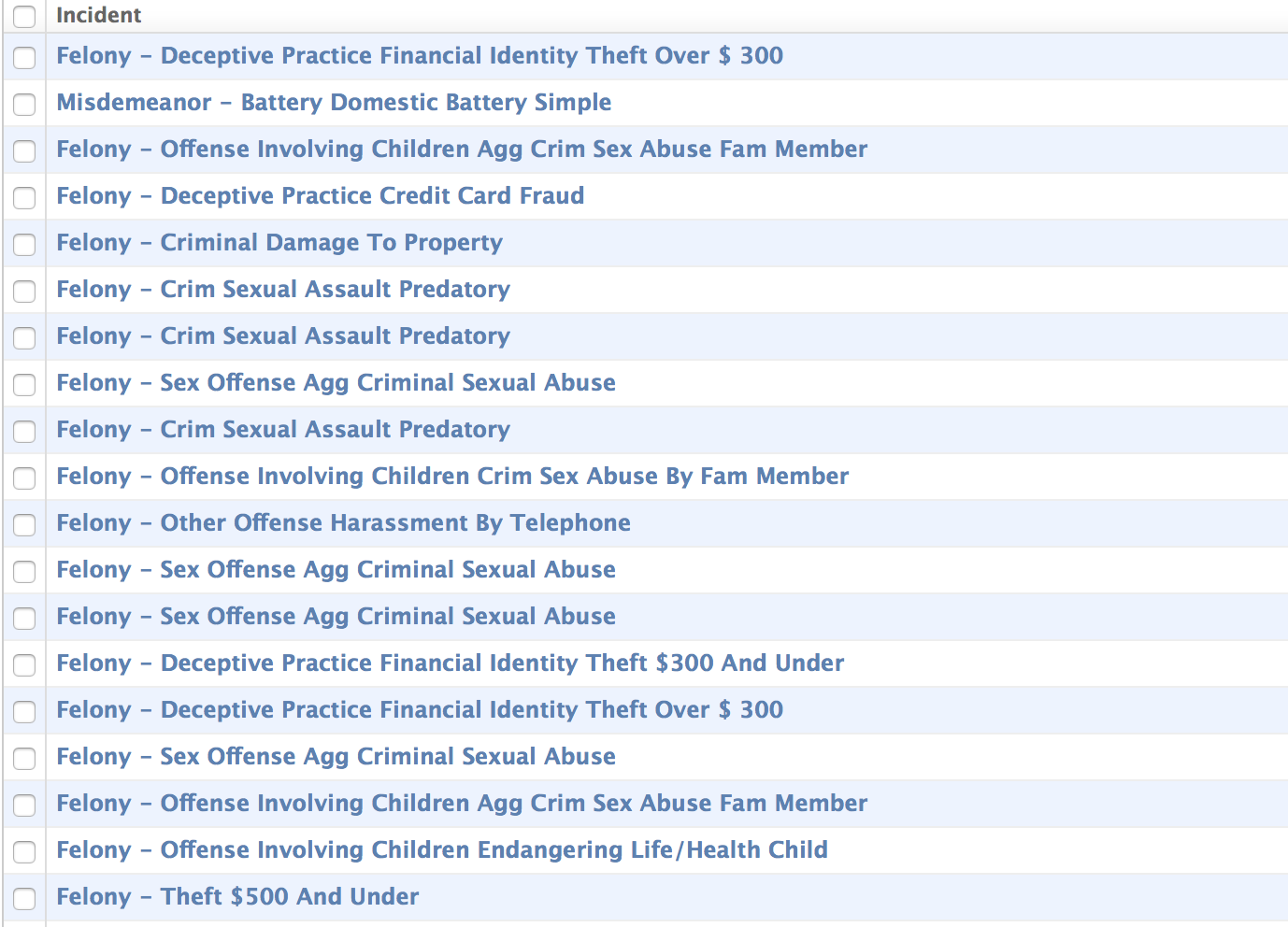
**

*Figure 2: The map zoomed into street level, displaying individual incidents.*

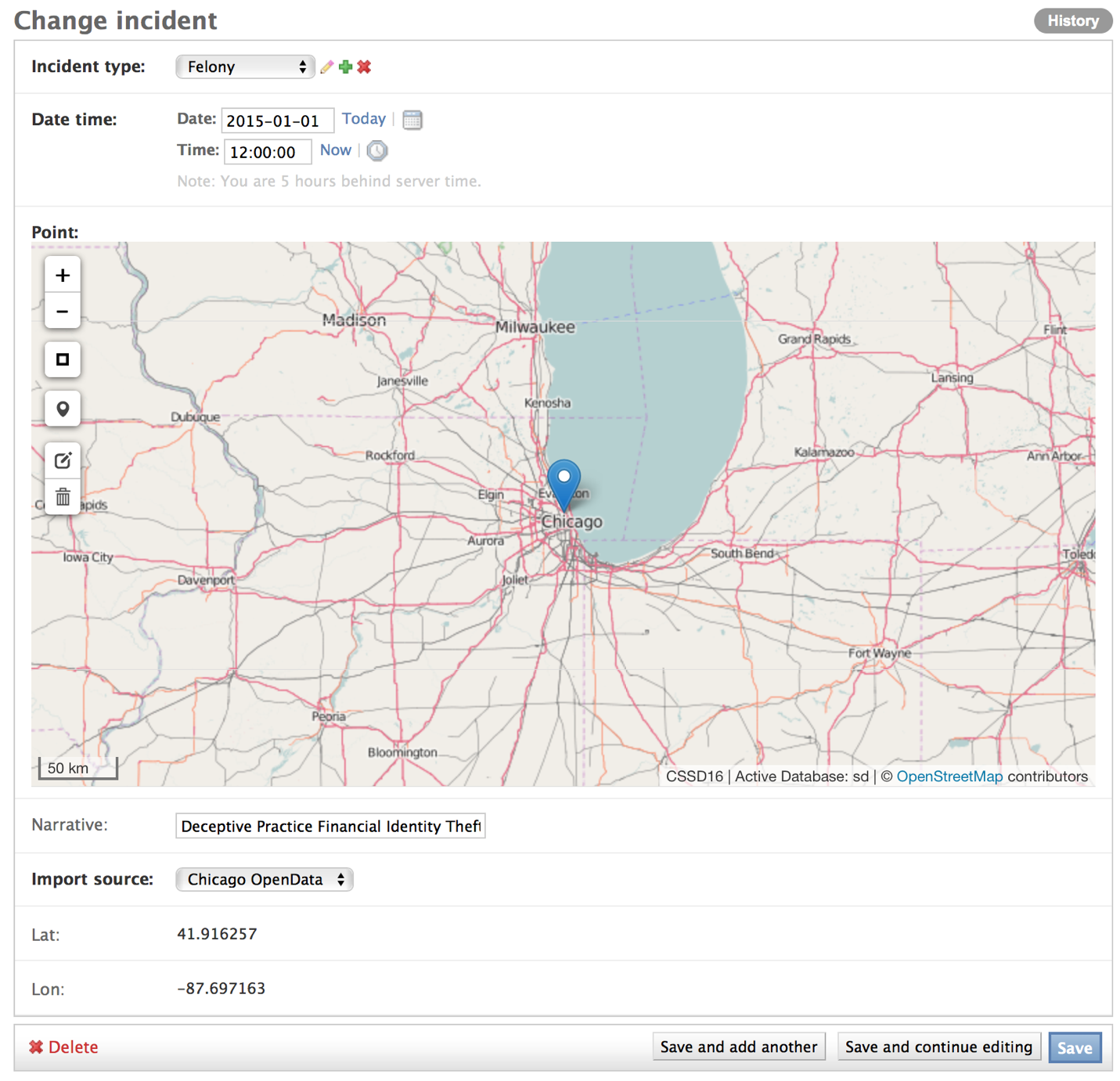
**

*Figure 3: an individual incident displayed in detail*

The text-based reporting interface will allow the user to view and modify (depending on permissions) incidents, crime types, user accounts, and other system properties. This will be a combination of custom-developed web pages and the built-in administration site provided by the Django web framework.



*Figure 4: a subset of the Incident list*

****

*Figure 5: The incident form. Note the OpenStreetMap point selector, which allows the user to drop a pin on the map and have the system calculate the latitude and longitude programmatically.*

**Major Components**

The frontend and backend of the application will be developed in component-oriented manner, to allow for easy modifications to a given component while minimizing the risk of cross-component interaction resulting in issues. The largest components on the backend are the data import module and the analysis operations module. The data import module is designed to allow for data to be imported from a variety of sources, including SQL-based databases, web services, and static datasets such as comma-separated values and plain text files. Thanks to the simplicity of the Django object-relational mapper (ORM) API, once the data has been transformed from the source into a Python object, the data can be stored in the application’s Postgres database with a very simple loop. However, the process of transforming the data from the source to Python objects is going to be what required the highest amount of effort. This is primarily because no two data sources are exactly the same, and the application will need to take into account the specific pitfalls of each source. For example, we would not be able to read a 60 GB text file straight into memory to process, instead, we need to do so in smaller chunks.

The data analysis component is also going to require substantial development work to implement successfully. Again, this is going to be developed in a component-oriented manner so new analysis operations can be dropped in with minimal reconfiguration of the rest of the system. Each analysis method will be defined as a separate class, and the cron job that will run to execute the analysis methods will be capable of auto-detecting new classes and seamlessly executing them on the next run.

On the frontend, the design challenges primarily have to do with filtering the data that is returned from the backend, so that the client is only loading and displaying the minimum amount of required data for the current map. In testing, this has proven critical as early iterations of the UI simply read in the entire database, rendering the map unusable after several thousand rows were imported into the database. While I have managed to partially solve this problem for the initial load of the map, I continue to see performance problems when the map is panned or zoomed after the initial page load, because of the high number of AJAX calls submitted with each movement of the map. I have a few ideas on how to correct this issue and will work on implementing them over time.

**External API’s**

As of right now, we are using external API’s to provide the base layers on the mapping UI and to provide geocoding services to translate street addresses into latitude and longitude. The OpenStreetMap base layer is being provided directly by the OpenStreetMap project because OSM is very liberal with usage limits. Originally, the maps used the MapBox base layer but I found the OpenStreetMap layer to be superior both in the usage limits and in the level of detail it provides. Depending on feedback from GW Police, I may need to change the base layer if they don’t like it for some reason, but I don’t foresee this being an issue and even if it turns out to be, the new base layer will still come from OSM in all likelihood.