Process Book

d3tiknom

Sam Olds, Jesus Zarate
12/2/17
CS 6630

Basic Info

Project Title	d3tiknom
Website	jesuszarate.com/d3tiknom
Github Repository	github.com/jesuszarate/d3tiknom

Name	*E-mail Address*	*UID*
Sam Olds	samolds@yahoo.com	u0742533
Jesus Zarate	jesus.zarate@utah.edu	u0816141

Background and Motivation

Sam currently works for Vivint, building a new REST service backend for a mobile app. The system is in it's infancy and could use a good deal of profiling and monitoring to know where optimization efforts need to be focused. The system is already integrated with a monitoring library, called *monkit* (godoc.org/gopkg.in/spacemonkeygo/monkit.v2), that reports various statistics every two minutes. The data is easy to obtain and doesn't (shouldn't) require any preprocessing. In fact, it would be ideal if our project could receive a stream of data to show live statistics.

The type of of data reported by *monkit* is mostly information about the duration of any function execution, in failure and success scenarios. It collects different types of metrics about these runtimes like *average*, *min*, *max*, and *most recent*. It also collects information about the call stack of these functions, so it's possible to trace the calling path.

This all means that collecting the data will be trivial and shouldn't require any additional finagling. And a visual representation of the data would be invaluable information for the performance of this project. Sam's team currently uses *graphite* (graphiteapp.org) to represent this data. But it's very challenging to use and lacks desired functionality. There are a number of features that would make the data visualizations more valuable than what graphite currently supports. An additional benefit of this project is that it won't only benefit Sam's project; it would be useful for any project that uses *monkit*.

Project Objectives

The primary question to be answered is, "What parts of the system need to be sped up?" We would like to have a visual representation of the running time of a system as a whole, as well as all of its individual parts. This can lead to improvements in the codebase, help uncover bugs, and possibly provide indication of things like memory leaks. These sorts of things are invaluable for an industry product that is expected to have hundreds of thousands of users.

Midpoint Checkpoint TA Feedback:

After our meeting with the TA we realized we had a few more things that we needed to consider in order to make our visualization more intuitive to use. The following list are part of the things we still needed to take into consideration.

- Overview visualization in order to quickly spot trouble areas
    - We tackled this by providing a heat map with color scale that would indicate which areas were causing trouble.

- Be able to search for specific gubbins
    - We provided a tooltip that on hover over would display more information on the the gubbin, but also would allow the user to navigate to the plots on a single click.

- Ability to navigate to a specific gubbin from the overview
    - We provided a search box in order for the user to be able to search gubbins, and would display the gubbins found as well as highlight the gubbin on the navigation window.

Data

The data is reported every two minutes to a central service. The data can be downloaded from this central service as text, a CSV, or as JSON. The following data is an example of the text format:

```
env.os.fds      120.000000
env.os.proc.stat.Minflt 81155.000000
env.os.proc.stat.Cminflt        11789.000000
env.os.proc.stat.Majflt 10.000000
env.os.proc.stat.Cmajflt        6.000000
...

env.process.control     1.000000
env.process.crc 3819014369.000000
env.process.uptime      163225.292925
env.runtime.goroutines  52.000000
env.runtime.memory.Alloc        2414080.000000
...

env.rusage.Maxrss       26372.000000
...

sm/flud/csl/client.(*CSLClient).Verify.current  0.000000
sm/flud/csl/client.(*CSLClient).Verify.success  788.000000
sm/flud/csl/client.(*CSLClient).Verify.error volume missing     91.000000
sm/flud/csl/client.(*CSLClient).Verify.error dial error 1.000000
sm/flud/csl/client.(*CSLClient).Verify.panics   0.000000
sm/flud/csl/client.(*CSLClient).Verify.success times min        0.102214
sm/flud/csl/client.(*CSLClient).Verify.success times avg        1.899133
sm/flud/csl/client.(*CSLClient).Verify.success times max        8.601230
sm/flud/csl/client.(*CSLClient).Verify.success times recent     2.673128
sm/flud/csl/client.(*CSLClient).Verify.failure times min        0.682881
sm/flud/csl/client.(*CSLClient).Verify.failure times avg        3.936571
sm/flud/csl/client.(*CSLClient).Verify.failure times max        6.102318
sm/flud/csl/client.(*CSLClient).Verify.failure times recent     2.208020
sm/flud/csl/server.store.avg    710800.000000
sm/flud/csl/server.store.count  271.000000
sm/flud/csl/server.store.max    3354194.000000
sm/flud/csl/server.store.min    467.000000
sm/flud/csl/server.store.recent 1661376.000000
sm/flud/csl/server.store.sum    192626890.000000
...
```

Data Processing

Although we anticipated not to have to do any data processing, we ended up doing two rounds of processing, due to the fact that we needed to represent the data in a hierarchy structure, and the data that we had was flat.

The initial approach we took was to do preprocessing on the data and turn it into a hierarchical structure.

This approach however restricted us because we would have to preprocess the data every time we got new data. We wanted to be able to provide new raw data and be able to process it on the fly so that we could potentially have live data.

The final approach was to get the raw data preprocess on the fly, and that approach yielded the following structure.

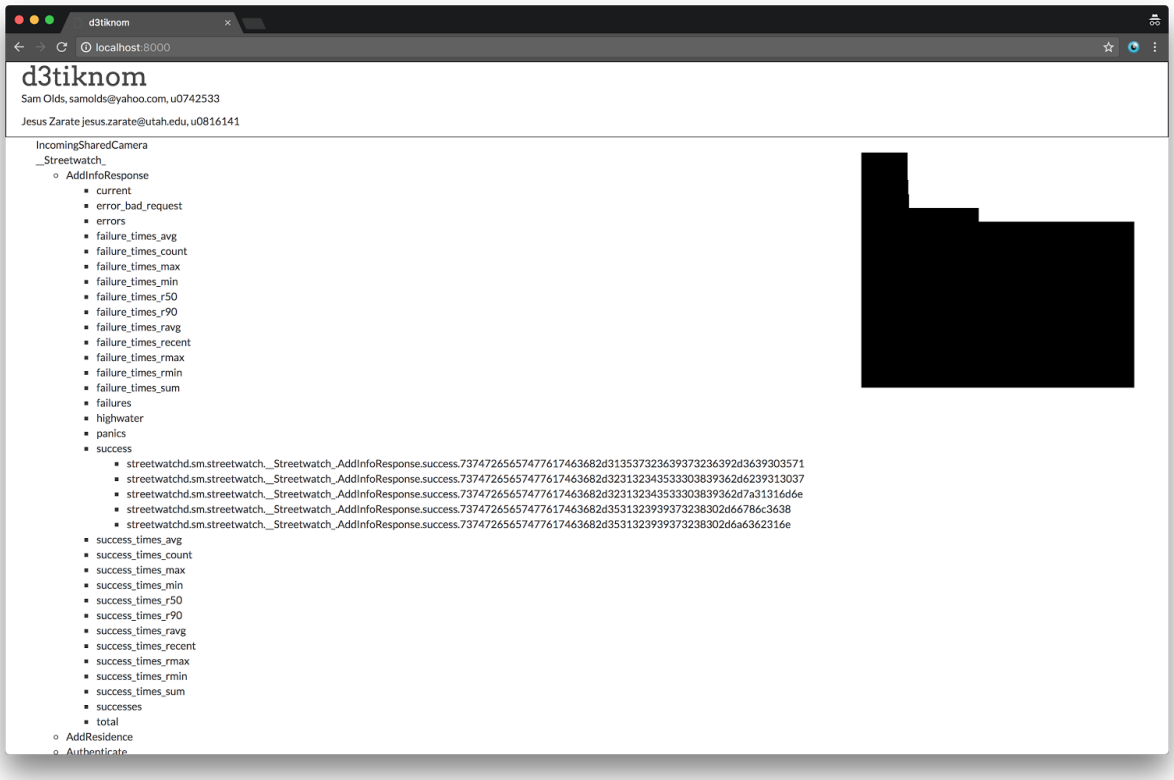
This approach allows us to easily traverse the tree like structure and display the navigation system.

Visualization Design

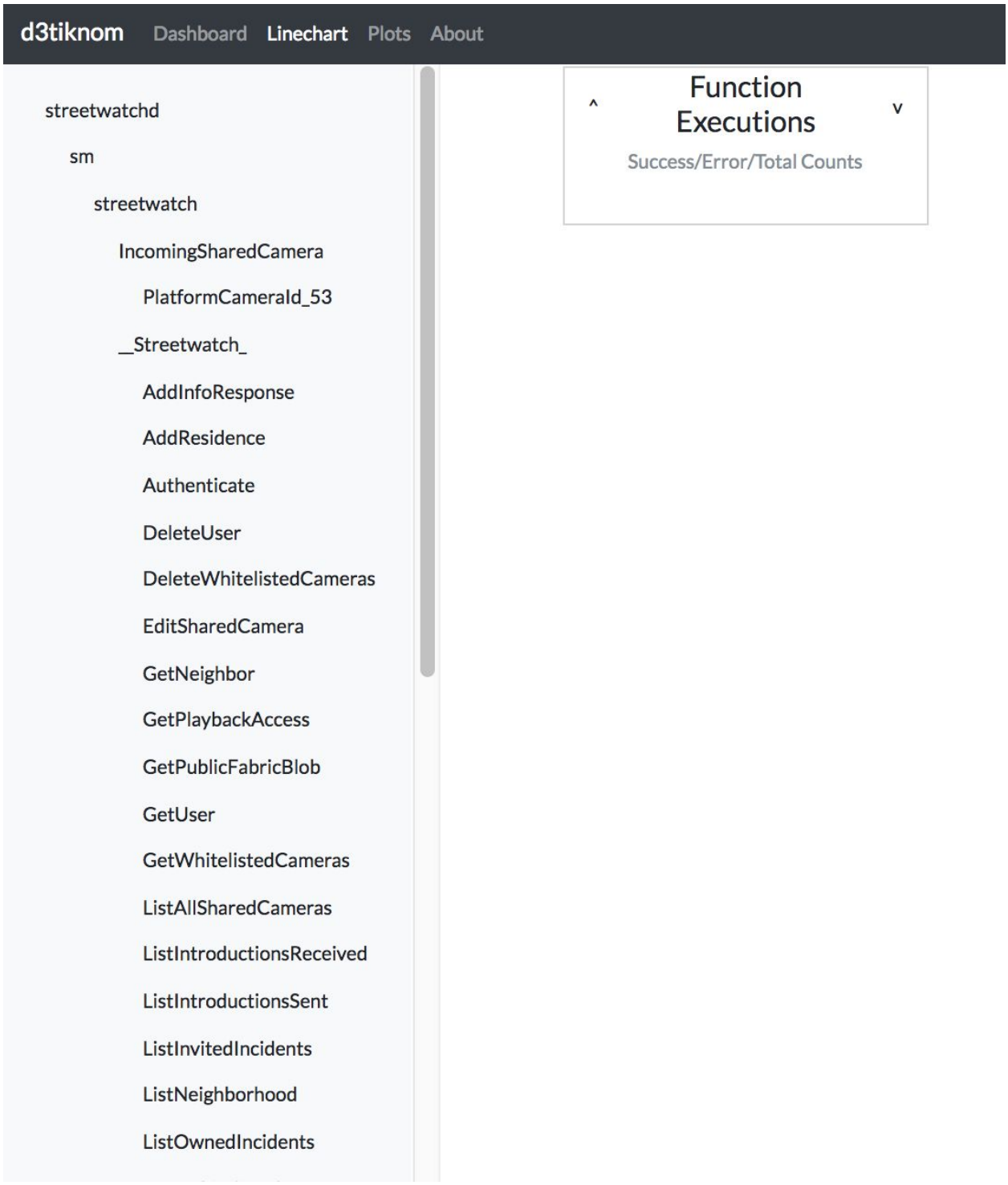
Navigation System:

One of the most important things in our visualization tool would have to be the navigation system, which was one of the things that we did not include in our proposal.

Our first approach to this was to have a file like structure that would allow us to see the different gubbins contained inside a “folder”. We are all familiar with a file structure therefore we thought it would be a good way to organize the data that way.

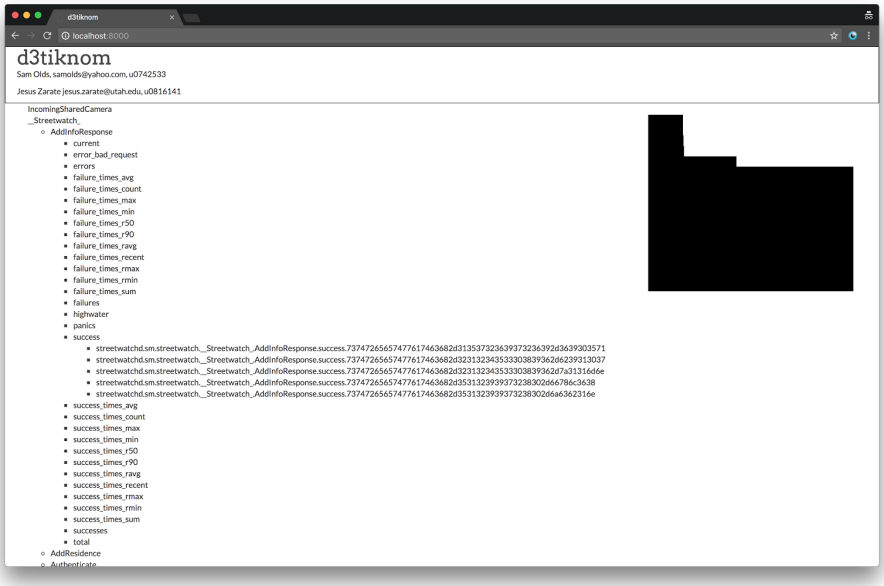


After making modifications to the data to better traverse the path to the desired gubbin the navigation system turned out like the following.

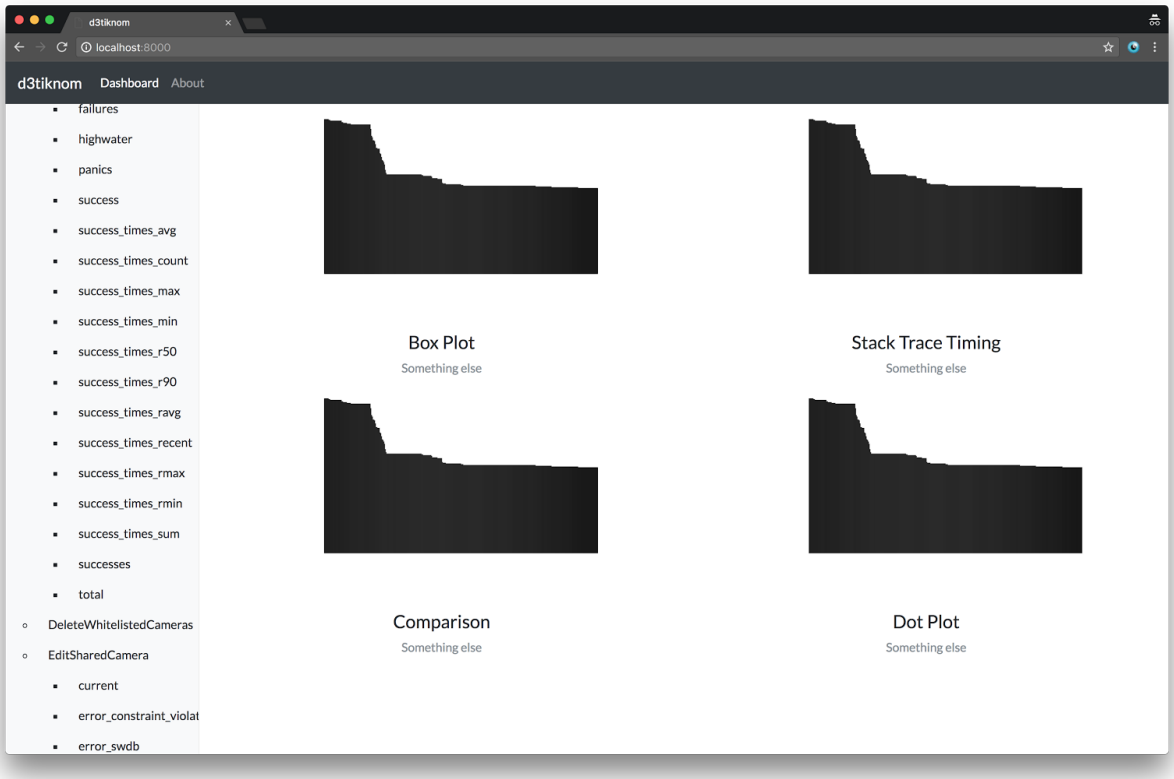

d3tiknom
Dashboard
Linechart
Plots
About
streetwatchd
sm
streetwatch
IncomingSharedCamera
PlatformCameraId_53
__Streetwatch_
AddInfoResponse
AddResidence
Authenticate
DeleteUser
DeleteWhitelistedCameras
EditSharedCamera
GetNeighbor
GetPlaybackAccess
GetPublicFabricBlob
GetUser
GetWhitelistedCameras
ListAllSharedCameras
ListIntroductionsReceived
ListIntroductionsSent
ListInvitedIncidents
ListNeighborhood
ListOwnedIncidents
^
Function Executions
v
Success/Error/Total Counts

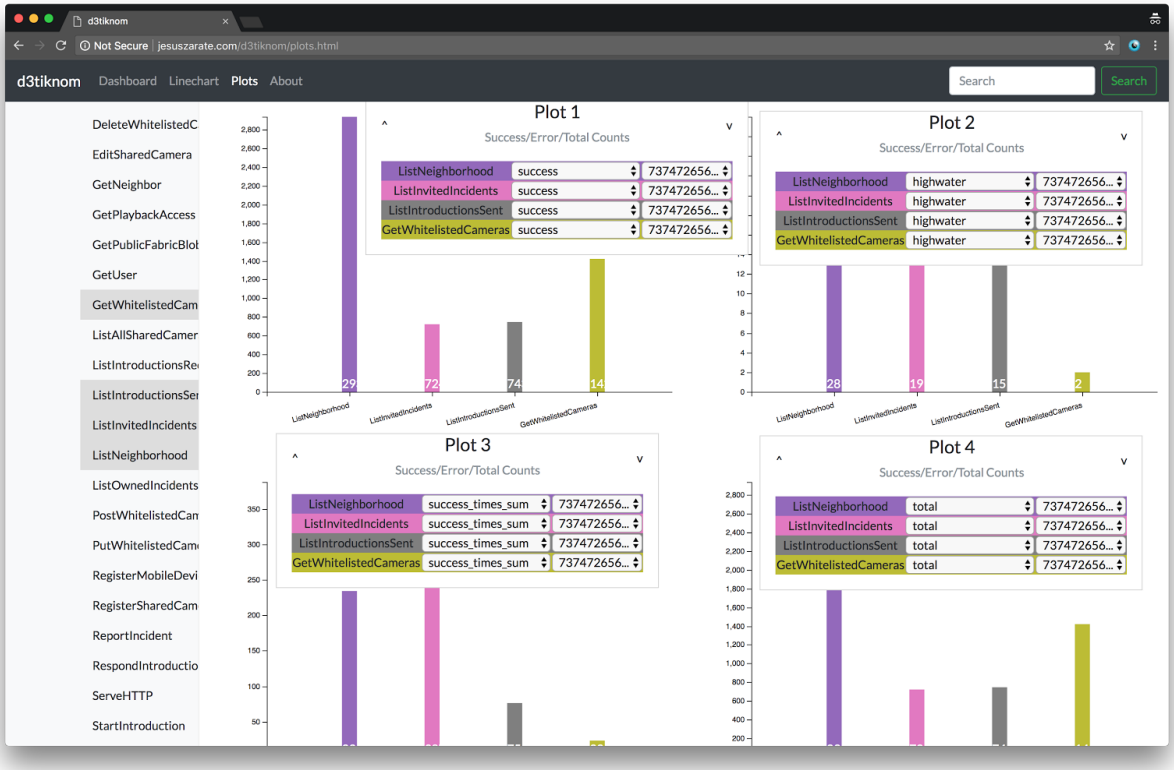
Plot comparison:

Being able to easily compare the metrics across multiple functions was a primary motivation behind this project. So developing a way to insightfully compare all of the metrics had a number of revisions.



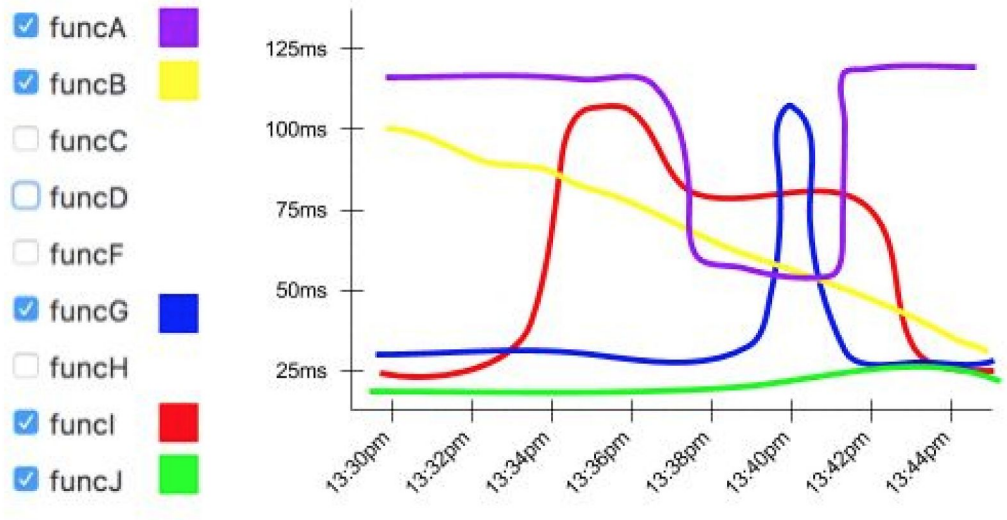
We knew that the user of this visualization probably has specific information they're after and wants to be able to easily compare many different types of thing all at once. So we decided to display multiple plots all at once, each of which customizable to show something different.

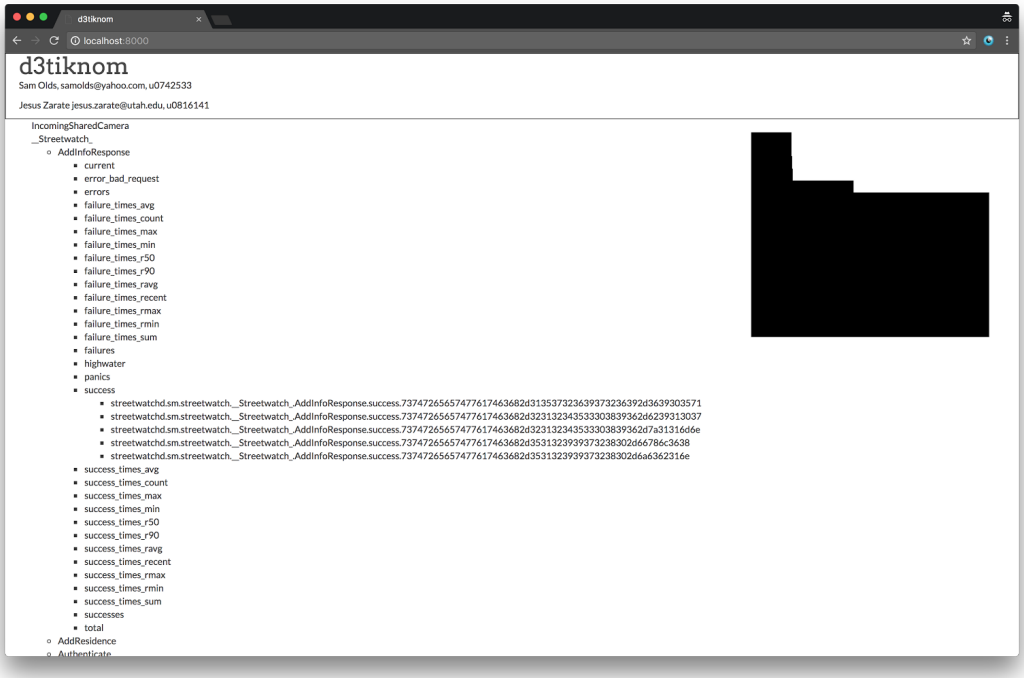

d3tiknom
localhost:8000
d3tiknom
Dashboard
About
failures
highwater
panics
success
success_times_avg
success_times_count
success_times_max
success_times_min
success_times_r50
success_times_r90
success_times_ravg
success_times_recent
success_times_rmax
success_times_rmin
success_times_sum
successes
total
DeleteWhitelistedCameras
EditSharedCamera
current
error_constraint_violat
error_swdb
Box Plot
Something else
Stack Trace Timing
Something else
Comparison
Something else
Dot Plot
Something else


d3tiknom
Not Secure | jesuszarate.com/d3tiknom/plots.html
d3tiknom
Dashboard
Linechart
Plots
About
Search
Search
DeleteWhitelistedC
EditSharedCamera
GetNeighbor
GetPlaybackAccess
GetPublicFabricBlob
GetUser
GetWhitelistedCam
ListAllSharedCamer
ListIntroductionsRe
ListIntroductionsSe
ListInvitedIncidents
ListNeighborhood
ListOwnedIncidents
PostWhitelistedCam
PutWhitelistedCam
RegisterMobileDevi
RegisterSharedCam
ReportIncident
RespondIntroductio
ServeHTTP
StartIntroduction
Plot 1
Success/Error/Total Counts
ListNeighborhood success 737472656...
ListInvitedIncidents success 737472656...
ListIntroductionsSent success 737472656...
GetWhitelistedCameras success 737472656...
Plot 2
Success/Error/Total Counts
ListNeighborhood highwater 737472656...
ListInvitedIncidents highwater 737472656...
ListIntroductionsSent highwater 737472656...
GetWhitelistedCameras highwater 737472656...
Plot 3
Success/Error/Total Counts
ListNeighborhood success_times_sum 737472656...
ListInvitedIncidents success_times_sum 737472656...
ListIntroductionsSent success_times_sum 737472656...
GetWhitelistedCameras success_times_sum 737472656...
Plot 4
Success/Error/Total Counts
ListNeighborhood total 737472656...
ListInvitedIncidents total 737472656...
ListIntroductionsSent total 737472656...
GetWhitelistedCameras total 737472656...

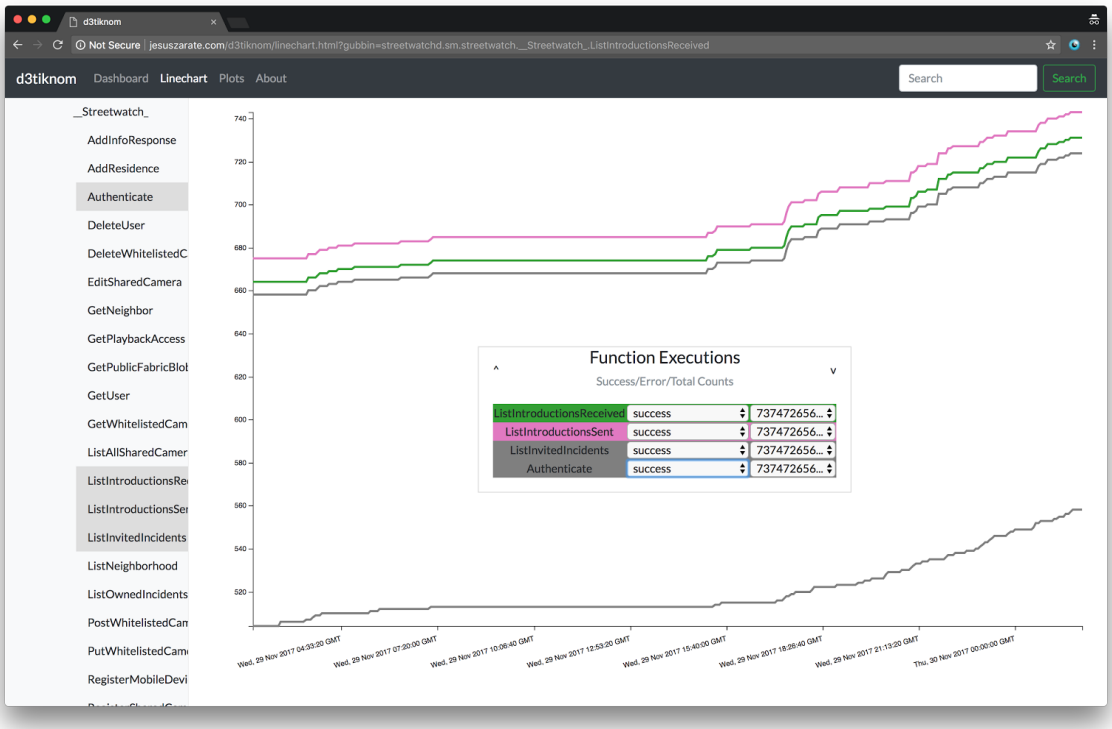
Linechart:

The linechart was another core requirement of this project. The data reported by monkit is always “over time”. There are metrics that display “highwater” or all of the requests that happened during a specific 2 minute interval to indicate if there are large spikes in usage. This sort of information is valuable in a rest service to identify periods of high use. It also useful to be able to compare the running times of endpoints against eachother to be able to identify which endpoints run slowly and need to be optimized. This was our initial vision of how it might look.



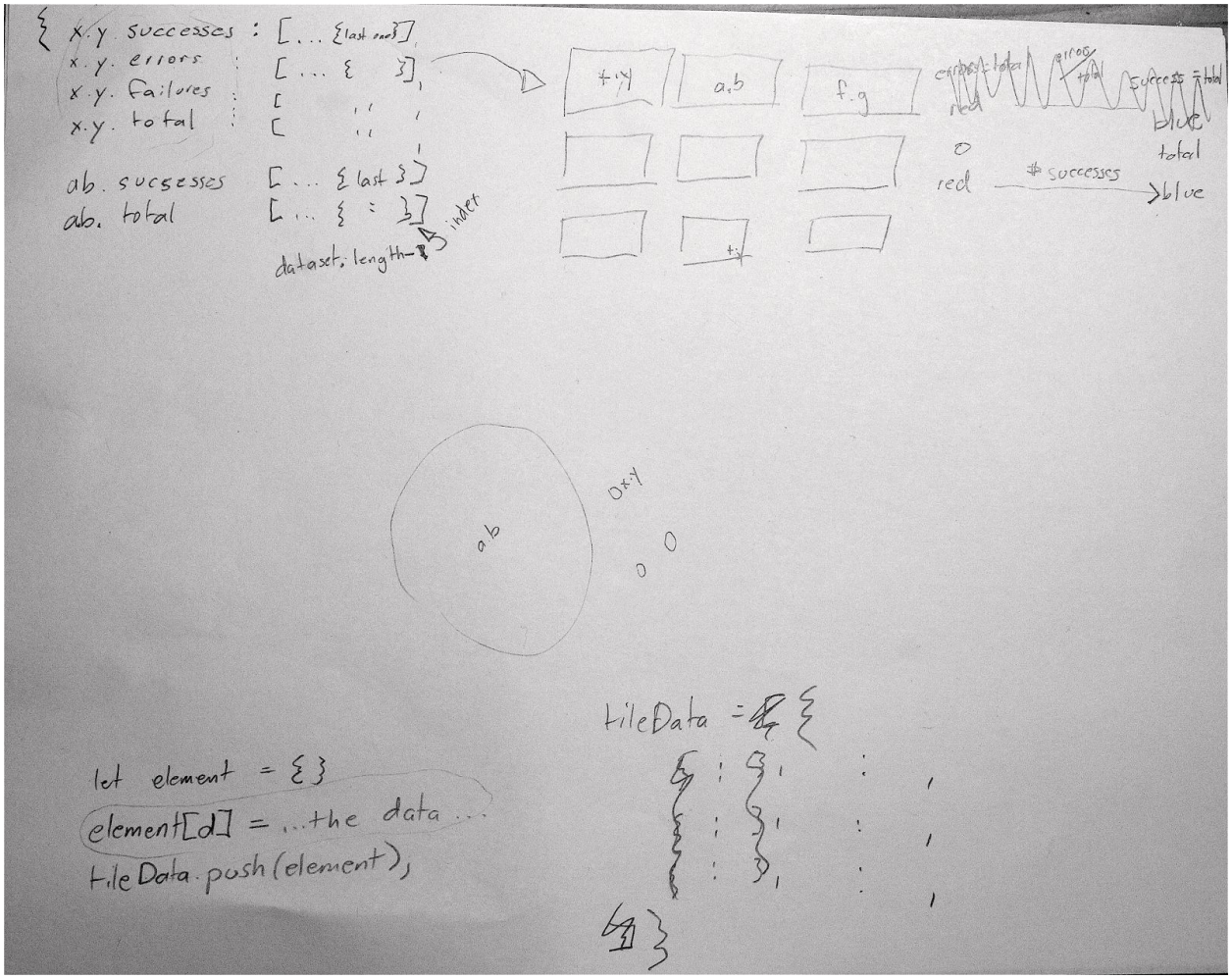


Our final implementation looks like:



Overview heat map:

After meeting with the TA for our midpoint evaluation he pointed out that there was still no clear way to see problematic areas in our data. In the proposal we included in our optional features a global visualization that would allow us to show all of the data in a heat map. We decided to apply this visualization in order to be able to see the overview and be able to spot those areas of problem with ease.



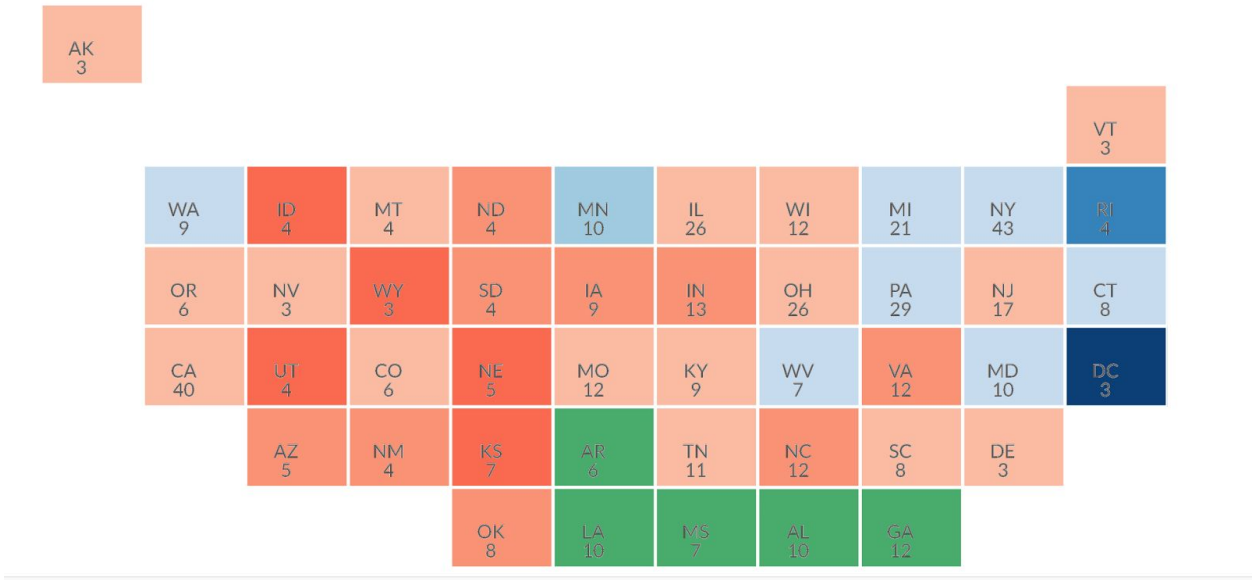
The sketch from above was created during one of our meetings after meeting with the TA. We discussed a couple different ways of representing the data. One using a heat map that would show have a color scale based on the ratio of failures and successful executions.

The other one was to have a bubble representation, which would represent the number of total calls to the that function and the size of the bubble would represent the total.

Heatmap Visualization:

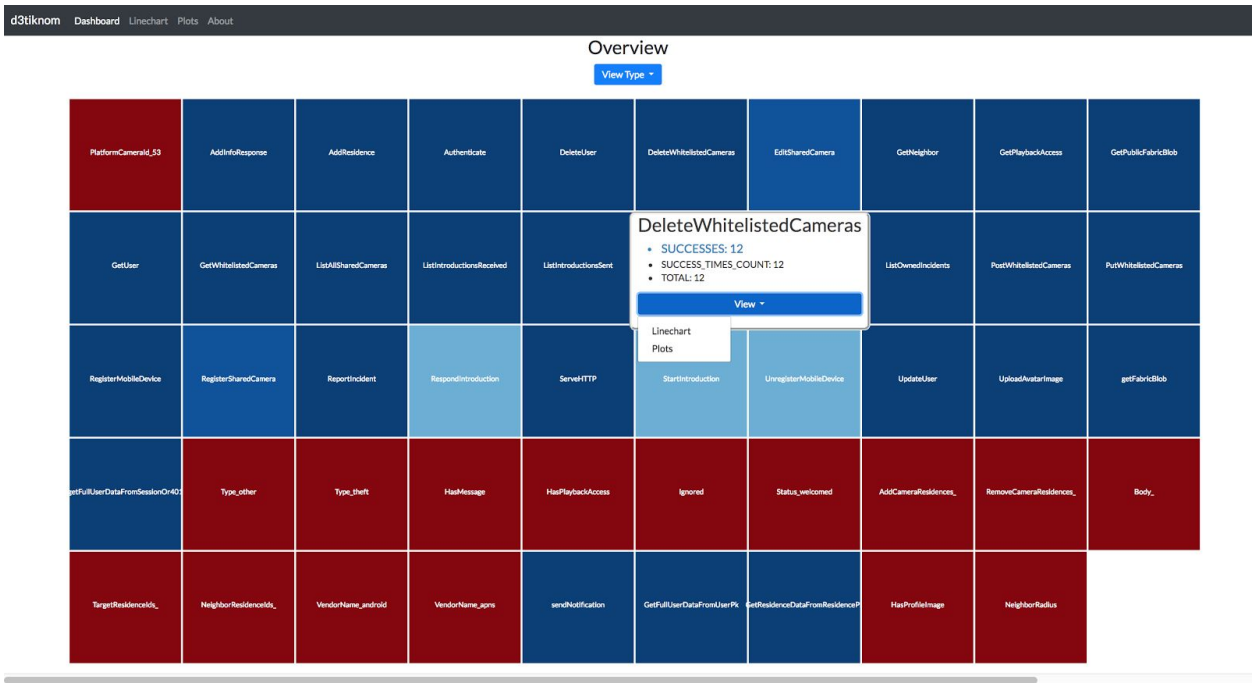
For the heatmap we decided to use a similar approach to the U.S. tile heat map. This visualization was intended to be able to spot problematic areas and a simple tile heat map gave us the ability to do just that.

Inspiration:



Another very useful part of the U.S. heatmap that we decided to apply was a tool tip. However, more than a tooltip that would allow the user to hover over and view more information, we wanted a way to navigate to the specific function. One click will expand the tooltip and provide a dropdown so that the user can pick which type of visualization they would like to see that function in. The drop down provides a line chart visualization and a plot option.

Inspiration:

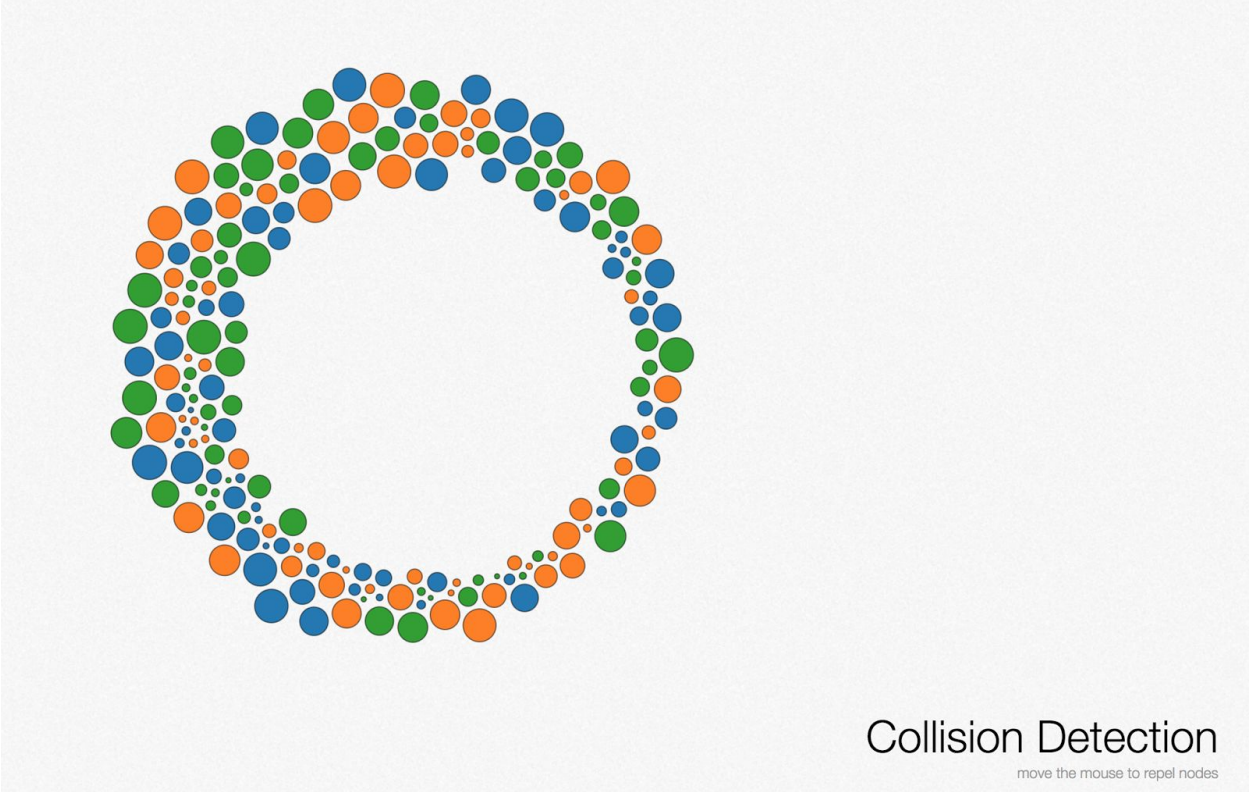


Bubble Visualization:

The other visualization we decided to use as an overview of the gubbins was a visualization that used bubbles to represent the total amount of times that a function was being requested.

We decided to use this other form of visualization to allow the user to quickly see which functions are the most requested functions, and be able to see if they are taking a big load, and will allow the user to perhaps redistribute the load.

We wanted to have no specific order in which the bubbles appeared in, but be able to not be colliding and obstructing each other. We used as inspiration and as a starting point the following visualization from here.



The final version of this visualization also included a way to hover over the bubbles and see more information on it as well as be able to navigate to the specific function and visualize it as a line chart or a plot.