

MIDDGUARD

Dana Silver

Adviser: Professor Christopher Andrews

A Thesis

Presented to the Faculty of the Computer Science Department
of Middlebury College

in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Arts

May 2016

ABSTRACT

MiddGuard is a web framework for collaborative and extensible visual analytics. It is built on the idea that a data-driven investigation can be represented as a graph of composable, chained data transformations and visualizations that are completely customizable by users and can take input from other arbitrary tools in the graph. The pairing of customization and arbitrary chainability allows investigation-specific, yet reusable tools. Additionally, MiddGuard is built for teams to collaborate on an investigation both asynchronously and synchronously. Multiple investigators can connect see a database persisted, real-time reflection of their colleagues' work building and generating data for a visual analytics investigation.

ACKNOWLEDGEMENTS

Professor Andrews for his continued advisorship since we started collaborating during the summer of 2014. His work in visual analytics continues to inform and inspire my work on MiddGuard.

My family and friends for their support.

TABLE OF CONTENTS

1	Introduction	1
1.1	Visual Analytics	1
1.2	Previous Work on MiddGuard	2
1.2.1	VAST 2014	2
1.2.2	MiddGuard: Summer 2014	4
1.2.3	VAST 2015	5
1.2.4	View Reference Counting	6
2	Background	9
2.1	State of the Art	9
3	The Framework	11
3.1	Overview	11
3.2	Example: Using Tweets to Investigate Relationships	13
3.3	Collaboration	14
4	Implementation	15
4.1	Technology	15
4.2	Data-flow Model	17
4.2.1	Analytic Nodes	17
4.2.2	Connections	18
4.2.3	Connection Storage	20
4.2.4	Context Generation	22
4.3	Visualization Nodes	24
4.4	Visual Programming	25
4.5	Extensibility	28
4.6	Real-time Collaboration	31
5	Discussion	36
5.1	Use Case	36
5.2	Areas for Improvement	38
6	Conclusion	41
A	Chapter 1 of appendix	42
B	Chapter 2 of Appendix	83
	Bibliography	96

LIST OF FIGURES

1.1	The web interface for Andrews and Silver’s VAST 2014 entry. The visualizations, from left to right, are a list of people, a master brushable timeline and individual timelines for each person listed with selectable events, and a map of GPS traces from the individuals’ cars.	3
1.2	A screen capture of the Google Chrome DevTools Profiler demonstrating the efficacy of View Reference Counting. The panel on the left shows three snapshots. Snapshot 1 was taken before a view has been added. Snapshot 2 was taken after a view was added and rendered with a significant amount of data loaded into the browser. Snapshot 3 was taken after that view was removed and the memory was reclaimed. . .	8
1.3	The snapshots portion of figure 1.2, cropped for readability.	8
4.1	MiddGuard’s graph editor user interface, open on a graph named “Compare Tweets”. On the left, the modules panel lists all loaded modules, from which nodes can be created. In the center, the graph editor canvas has seven nodes initialized from their respective modules, and connections between the nodes. On the right, the detail panel shows the column mappings between the “Difference by Hour” node and its connections to two “Time by Day/Hour” nodes.	26
4.2	An analytic node in a graph. Important features are annotated and the node’s only input group, “tweets”, is moused over to show its accompanying tooltip.	29
4.3	Code for an example analytic module.	33
4.4	Contents of the main configuration file, <i>index.js</i> , for an example visualization module, “Hours Heatmap”.	34
4.5	Code for an investigation’s MiddGuard server. It creates an instance of the MiddGuard server, passing in the database configuration from a Knexfile [9] and a secret key to encrypt authenticated session data. This investigation uses the two modules from figures 4.3 and 4.4 and registers them with calls to <code>app.module</code> . Finally the server starts listening for connections on port 3000.	35
5.1	The complete graph from the mock investigation used to develop and test MiddGuard.	37
5.2	The “Hours Heatmap” visualization from the mock investigation used to develop and test MiddGuard. @DanaRSilver’s tweets are orange, @jack’s are blue. Circle radii are mapped to the number of times each person tweeted in that hour and day of the week.	38

CHAPTER 1

INTRODUCTION

1.1 Visual Analytics

Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces [13]. A visual analytics based investigation combines tooling for data transformation and visualization with human judgment to evaluate information and gain insight. Effective visual analytics tools need to transform disparate types of data from different sources to support visualization and analysis. Investigations often involve responding to or preventing a threat and are time sensitive. In *Illuminating the Path*, Thomas and Cook write that “Research is needed to create software that supports the most complex and time-consuming portions of the analytical process, so that analysts can respond to increasingly more complex questions.” [13]. For an investigation to be effective and conclusions to be convincing, results have to be understandable and reproducible.

MiddGuard aims to address the challenges posed by visual analytics. It partitions the analytic process into a series of data transformations and visualizations, combining them into a unified, transparent model with a visual representation. MiddGuard provides the backing framework and integrated analytic environment to communicate data between teams of investigators and load/unload visualizations. Developing this scaffolding takes time to implement that could be spent on the investigative process.

MiddGuard’s model for extensibility allows developers to focus solely on writing the tools they need to transform data and render visualizations. It exposes simple APIs to extend the framework while remaining agnostic as the the implementation details. Both transformation and visualization tools can be written using any technologies. This allows developers to produce bespoke tools quickly.

1.2 Previous Work on MiddGuard

1.2.1 VAST 2014

The VAST Challenge is a visual analytics competition organized by Visual Analytics Community with results presented at IEEE VIS. The challenge gives competitors a description of a crime scenario and data surrounding the crime. It asks analysts to create and use tools to investigate the data to indentify abnormalities, people of interest, and clues for the police to pursue. The VAST 2014 Challenge [6] posited the following fictitious scenario:

In January, 2014, the leaders of GASTech are celebrating their new-found fortune as a result of the initial public offering of their very successful company. In the midst of this celebration, several employees of GASTech go missing. An organization known as the Protectors of Kronos (POK) is suspected in the disappearance, but things may not be what they seem.

During summer 2014, Christopher Andrews and Dana Silver collaborated on a submission for VAST 2014 Mini-Challenge 2, one of four challenges (including an all encompassing “Grand Challenge”) dealing with the VAST 2014 Challenge scenario.

For our VAST 2014 submission, we created a web interface to visualize and analyze data from the challenge scenario. Data were preprocessed using several disjoint Python scripts and the resulting manipulations were persisted to a SQLite database. On the back-end of the web service, a simple RESTful Python web server implemented with Flask [11] and Flask RESTful [5] queried the database and transformed data for various front-end visualizations. The server also performed manipulations in addition those in the preprocessing stage on a request-by-request basis based on analyst input in the interactive visualizations. Figure 1.1 shows the web interface for our tool.

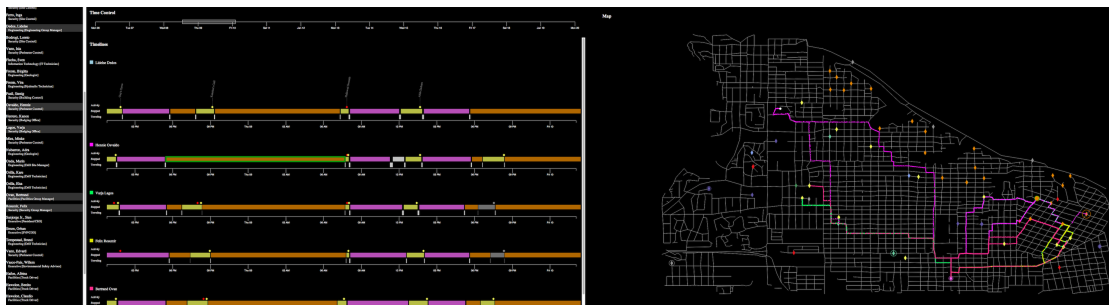


Figure 1.1: The web interface for Andrews and Silver’s VAST 2014 entry. The visualizations, from left to right, are a list of people, a master brushable timeline and individual timelines for each person listed with selectable events, and a map of GPS traces from the individuals’ cars.

For an example of the flow and feedback loop between preprocessing scripts, back-end server, and front-end visualizations we look how we used the Mini-Challenge 2 geographical data to identify points of interest and associate them with car destinations. The VAST 2014 Mini-Challenge 2 dataset included vehicle tracking data from company cars, an ESRI shapefile of the island where GASTech is located, and an illustrated tourist map of the island. Tracking data contained lists of latitude, longitude, timestamp, and car ID.

We wrote a preprocessing script in Python to iterate through individual cars’ GPS traces from the vehicle tracking data, identify periods where a car was stopped, and save the coordinate where the car stopped as a destination for the associated car. On the front-end, an interactive visualization rendered the shapefile and preprocessed tracking data to draw a map of the city overlaid with cars’ movements and destinations. We created points of interest on the map using car destinations and names from the tourist map. Persisting the association of point of interest and a single destination to the database ran a procedure that identified other nearby destinations to automatically associate with the same point of interest.

Our VAST 2014 submission was unsuccessful. Working on the tool took most of the available time and we were not left with sufficient time to complete the investigation

and write up the results.

1.2.2 MiddGuard: Summer 2014

The first version of MiddGuard, which was developed in response to summer research at Middlebury, attempted to generalize parts of the web server and front-end that could be reused throughout multiple investigations, while keeping the framework unopinionated with respect to the data it could handle.

From the VAST 2014 Challenge we drew conclusions that influenced the first version of MiddGuard. We found that while the web could be an effective platform for visual analytics, the overhead of creating custom tools, getting those tools to work with the rest of the system, and implementation bugs in the server-client communication hindered our progress investigating. To address these issues, the framework's primary features were automatic persistence to a database, data transport between the server and connected web clients in real-time, centralized data storage in the web browser, and visualization module loading/unloading in the browser.

This version of MiddGuard achieved flexibility by automatically loading three types of customizable packages. These were referred to as analytics, modules, and models. Analytics were scripts that could be triggered by a remote procedure call from a front-end visualization. They could be passed data from the front-end. Using the VAST 2014 example, they were meant to handle computations like finding other destinations near a point of interest.

Modules were front-end visualizations that used JavaScript and CSS to render and style elements in the browser's DOM. Visualizations were interactive, could communicate with the backend to update and persist data, and could save state to a global state handler to link visualizations to each other. For example, a master brushable timeline saves the boundaries of the brushed region to its state, which other timelines read to

update their detail view.

Models were table-level schema for the database, intended to allow MiddGuard to work with any data. A database table could then be created from each model. The entire database was accessible on the front-end, with each table represented by a Backbone.js Collection, which acts like an array of table rows. Collections were updated in real-time using a publish-subscribe like method. Updates to a collection on the front-end and to a models on the back-end were communicated to one another in real-time. This allowed investigators to modify the data in visualization modules and analytics packages without implementing communication. By listening to changes in a Collection, a visualization could rerender as soon as data changed on the server or in another investigator's browser. The real-time, database-persisted communication protocol for models allowed investigators to collaborate synchronously and asynchronously.

1.2.3 VAST 2015

Christopher Andrews and Jullian Billings used MiddGuard for the VAST 2015 Challenge. They report that the framework allowed them to take a modular approach to developing tools for the investigation, deploying visualizations as needed without needing plan and coordinate the entire investigation before it began. They expanded the front-end state manager and used it to link their visualizations: “The shared state provided by Middguard meant that the modules could be easily snapped together into an integrated environment, facilitating the flow of information between the tools. This sped development because tools could be simple and focused, with data selection and filtering shared between tools.” [1]. MiddGuard was well received by visual analytics professionals, winning a VAST 2015 Challenge award for integrated analysis environment.

The VAST 2015 Challenge investigation revealed some shortcomings of MiddGuard. Storing all data in a web browser wasn't realistic. Datasets for investigations, including

VAST, are often several gigabytes in size, more than can fit in the browser while maintaining the performance required for interactive visualizations. Even with modifications to load subsets of the data, the Backbone Collections quickly grew large, and filled with unnecessary data not reflected in any active visualizations. View Reference Counting was designed to address this issue.

Analytics packages, one of MiddGuard’s built in tools for extensibility, designed to run arbitrary code via remote procedure calls from the front-end, were not sufficient to obviate the need for preprocessing scripts. The investigators still wrote Python scripts to transform data and alter the database outside MiddGuard. The lack of record of how these scripts were used added a layer of opaqueness to the analytic process, making results hard to reproduce and collaboration difficult. The framework designed and implemented in this thesis addresses the issues of transparency and reproducibility in the analytic process, while introducing a method to include the preprocessing script contents in MiddGuard.

1.2.4 View Reference Counting

In the original implementation, one of the MiddGuard’s weaknesses was handling large amounts of data on the front-end. The framework was implemented to load the entire database into the browser with the idea that investigators would need access to all data during the investigation. MiddGuard’s server would continue to push data updates to connected clients as they became available. However, with the large dataset from the VAST 2015 Challenge, the browser was not able to handle all the data at once. MiddGuard was modified with a stopgap solution during VAST 2015. Instead of loading all data from the outset, visualization modules made custom database queries as necessary.

This did not solve the problem of unused data in the browser. Once downloaded to the browser data was never removed, even after the visualization that required it

was. MiddGuard stores all data in a central location to avoid the duplication that would occur by having each visualization store its own data. This makes it impossible for a visualization that has requested data to clean up after itself. Another visualization may have requested and currently be using the same data.

To keep the deduplication advantages of central storage and clean up after visualizations that were removed from the browser, we implemented automatic memory management in the browser called View Reference Counting. View Reference Counting (VRC) maintains an array of references to the views that use each piece of data as an attribute on the datum's Backbone.js Model. When a visualization (a Backbone.js View, hence the name) is removed, its reference is removed from the model. When a model has no view references it is removed from the browser.

Figures 1.2 and 1.3 demonstrate the efficacy of View Reference Counting through the three memory snapshots taken by the Google Chrome DevTools Memory Profiler. After a view with several megabytes of data was added and removed, MiddGuard cleaned up the data and the browser was able to reclaim the memory.

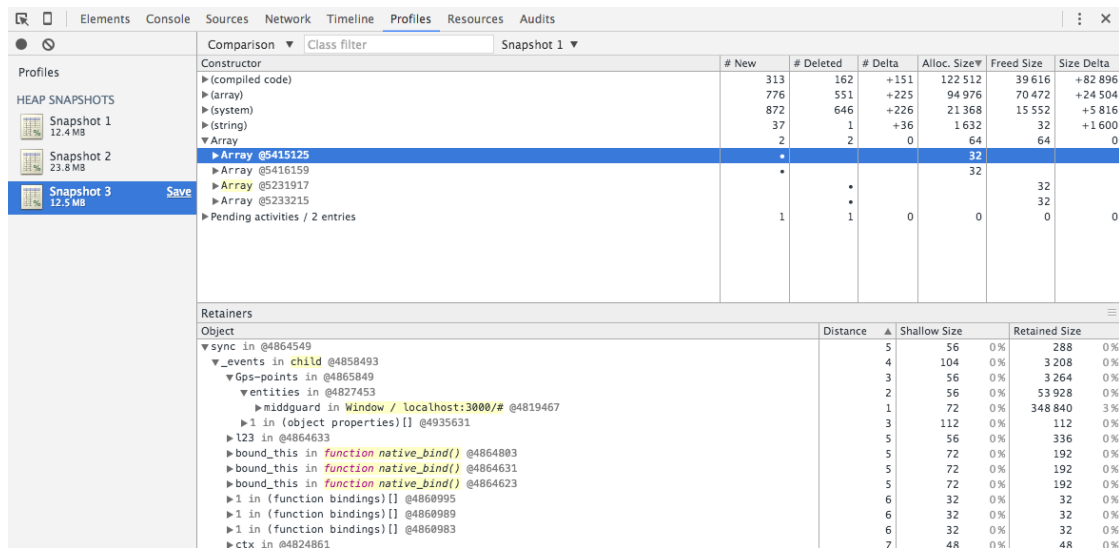


Figure 1.2: A screen capture of the Google Chrome DevTools Profiler demonstrating the efficacy of View Reference Counting. The panel on the left shows three snapshots. Snapshot 1 was taken before a view was added. Snapshot 2 was taken after a view was added and rendered with a significant amount of data loaded into the browser. Snapshot 3 was taken after that view was removed and the memory was reclaimed.

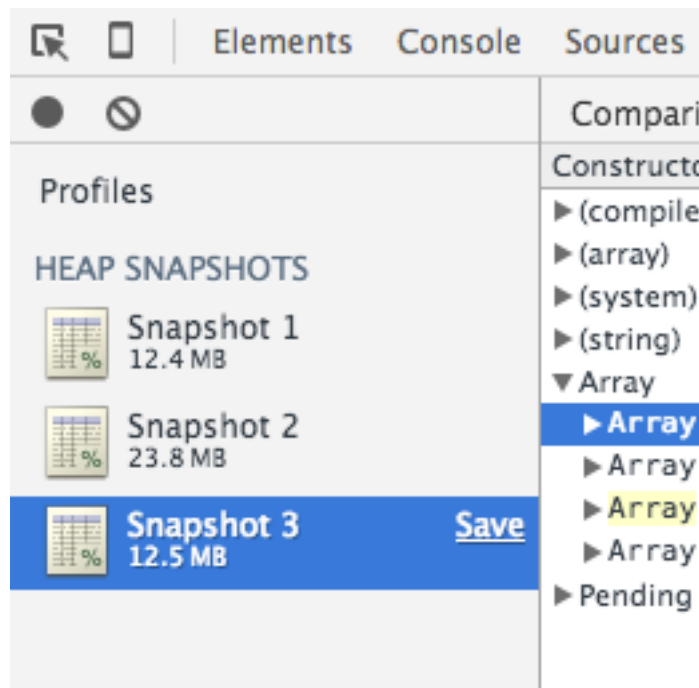


Figure 1.3: The snapshots portion of figure 1.2, cropped for readability.

CHAPTER 2

BACKGROUND

2.1 State of the Art

Jigsaw [12] is a visual analytics tool to explore and understand collections of text documents. Introduced in 2007, Jigsaw provides four visualizations, called views, to present different perspectives of the text and extracted entities. While MiddGuard is not a text based tool, the concepts that drive Jigsaw’s views are of interest. Jigsaw views are coordinated and communicate with each other to update themselves. User interaction with one view updates the others. Multiple copies of each view can be created to reflect different perspectives of the data.

MiddGuard supports customizable views, rather a limited, baked in, set. Like Jigsaw, MiddGuard allows multiple copies of each type of view to be added to the investigation, each with a different view of the data. MiddGuard visualizations can be coordinated using a global state manager. Like the views themselves, coordinations need to be developed manually.

Improvise [14] enables users to build and browse highly-coordinated visualizations interactively. It allows users to load data, create views, specify visual abstractions, and establish coordinations interactively. MiddGuard implements a similar build and browse system, where visualizations are assembled using a visual configuration and simultaneously rendered in the browser. Improvise places strong emphasis on complex, scriptable, and visually representable coordinations between views where MiddGuard relies on a global state manager, which developers can use as necessary to coordinate multiple views.

Eagleyes [3] is an interactive dataflow engine. Customizable modules that can be chained together to transform, query the data, and create visualizations. MiddGuard’s

data-flow model follows a similar data-flow model, extending it through a synchronous collaboration system that allows multiple analysts to work on the same data-flow at once.

CHAPTER 3

THE FRAMEWORK

3.1 Overview

MiddGuard is a web framework that enables software developers and analysts to create the tools to conduct complex, data-driven investigations. It provides a browser based front-end and web server back-end on top of which developers can build customizable tools specific to their data and investigation. Data is not uniform and investigating that data requires bespoke tools. MiddGuard, rather than implementing all the specific tools necessary to address all possible scenarios, provides the scaffolding on which developers can bring and build their own tools. The user interface and web server that MiddGuard implement create a simple environment to connect and use those tools transparently and efficiently.

MiddGuard breaks the operations of a visual analytics based investigation into two general steps: data transformation and data visualization. Data transformation involves any function on the data that results in a different, possibly destructive, representation of the same dataset. These functions might involve reading, filtering, aggregating, annotating, or reformatting the dataset. As a general rule in MiddGuard, if the operation does not produce a visualization, it is a transformation. Data visualization takes place after the transformation steps, creating a visual, often interactive, representation of the dataset. By implementing these two steps, developers can extend MiddGuard to fit their data and investigation.

These extensions to MiddGuard are called modules. Modules are short pieces of code that often live in a single file. We divide modules into two types to represent data transformation and visualization respectively. The former is called an analytic module, while the latter is a visualization module. Modules implement a simple protocol

that MiddGuard is able to read and use to integrate the code into the framework. Analytic modules consist of code that runs solely on the web server. Visualization modules contain code that runs in the web browser to render DOM elements that make up its visualization.

Once the MiddGuard web server is running, investigators use these modules to build a data-flow graph. Modules are the graph's nodes. Edges between nodes describe data-flow from module to module. MiddGuard's web front-end comes with a graph editor that investigators use to add modules to the graph and connect modules to each other. Once added to the graph, a module has been instantiated in the context of the graph and is called a node. Analytic nodes can be chained from one to the next, making the graph a canvas to compose complex data transformations from multiple analytic modules, each with a singular task. Visualization modules can be connected to analytic nodes, which feed in data to create the visualization.

Although modules are customizable and can be written for a particular investigation, they are also reusable within the same graph, different graphs of the same investigation, or multiple different investigations. Modules' relationships to each other are managed by MiddGuard and defined by connections in the graph, rather than hardcoded into the modules themselves. For example, a developer could create a visualization module that renders a heatmap of two entities' activity moving around a city. The module would be written to accept input from Entity A, Entity B, and the data to draw the underlying city map. An investigator can connect the heatmap to any two cars, people, bikes, etc. from another dataset and render a heatmap with no additional development effort. As developers and investigators use MiddGuard, they build up a library of these reusable tools. In an investigation where time is a factor, being able to quickly plug in and test data transformations and visualizations promotes the investigator's efficiency.

3.2 Example: Using Tweets to Investigate Relationships

An example of the data-flow model is using tweets to determine the relationships between multiple people: Alice, Bob, and Carlos. We start by writing three similar modules that use a JavaScript library to access the Twitter API and download all of the tweets for each person, respectively. Between the three modules we only need to change the Twitter handle for which we are downloading tweets. We add a graph called “Tweet Relationships” then create nodes from these modules and add them to the graph. We can use the number of times one user mentions (such as @Bob) another as a metric for the relationship, so we write another module called “Mention Count” that extracts mentions from each tweet and creates a mapping from the Twitter user mentioned to the number of times mentioned throughout the dataset. We add this module to the graph three times, and connect one “Mention Count” node to each of Alice, Bob, and Carlos’s tweet download nodes. Already we are able to reuse “Mention Count” for each person’s tweets. Finally, we visualize the relationships. We can use a force directed graph with a node for each person and strength of the edges proportional to the number of times one mentioned the other. Our visualization module, “Force Directed Graph” will take three inputs, one for each person. We create a node in the graph from the “Force Directed Graph” module and connect each of the outputs from our “Mention Count” nodes to the three inputs of “Force Directed Graph”. Like the “Mention Count” module, “Force Directed Graph” is reusable and can be plugged into any three inputs.

At this point our graph is ready to produce data and a visualization. We work from the data entry points to the visualization, running the tweet download nodes, then the “Mention Count” nodes, then the “Force Directed Graph” visualization. The analytic nodes report when they are done so we know it is safe to run their dependents. Running the node “Force Directed Graph” renders the visualization next to our graph in the browser window.

3.3 Collaboration

MiddGuard not only enables single investigators to create and work with these tools, but also has built in support for asynchronous and synchronous collaboration between teams of investigators. The framework includes user registration and authentication so multiple investigators can create accounts, log in, and work on the same investigations with the same graphs and access to the same data. All configuration and transformed data is persisted to a database, so investigators can log in and work with each other asynchronously, one picking up where the other left off. Investigators can also work together in real-time. As edits to the data-flow model are persisted to the database, they are pushed to all connected web clients and the user interface updates without a refresh to reflect those changes.

Since developers can collaborate to build the investigation, it follows that they should be able to collaborate to record conclusions from their analysis. MiddGuard comes with an observations tool for investigators to record and share observations about the analysis, creating a chronological record of what investigators saw in the data and when they saw it. An investigator of the tweet-based relationships from the previous example might record “Alice appears to have a close relationship with Bob. See the Force Directed Graph visualization in the Tweet Relationships graph.” Like graphs and data, these observations are persisted to the database and pushed to all connected clients in real-time.

CHAPTER 4

IMPLEMENTATION

4.1 Technology

MiddGuard builds on many open source software projects, some of which are instrumental to its implementation. Node.js, Knex.js, and Backbone.js make possible MiddGuard's structure and flexibility.

Node.js [8] is an asynchronous, event-driven JavaScript runtime built on Google Chrome's V8 JavaScript engine. The runtime is structured around an event loop where callbacks are registered and fired later in the program's life. Most I/O operations are performed indirectly through the event loop, so the process rarely blocks, allowing high concurrency and scalability. MiddGuard's server is implemented in JavaScript running on Node.js. The code for its HTTP and WebSockets, servers take advantage of the event loop. WebSockets are a bidirectional protocol for client-server communication. Since they are event-driven from the server, rather driven by the HTTP request-response cycle, WebSockets are simpler to implement and deploy with Node.js than with many traditional servers for other languages and web frameworks.

Knex.js [9] is a SQL query builder with support for several relational databases including Postgres, MySQL, and SQLite. Knex exposes an API with function calls similar to SQL keywords that generate and execute SQL in the appropriate dialect for the connected database. It supports schema generation and returns the results of queries it runs on the database. MiddGuard uses Knex.js to simplify database connections for custom analytic modules and make the framework flexible to whichever database is best suited to the investigation. For the VAST 2015 Challenge Andrews and Billings used a Postgres database and connected over the network to collaborate from separate machines using the same database. For single person investigations using SQLite is

often preferable since it does not require a database connection.

Backbone.js [2] is a front-end library designed to give structure to web applications. It consists of extendable Models, Views, and Collections, all of which we use to structure MiddGuard’s front-end. Models manage data attributes and trigger events when that data changes. Collections are groups of related models. For example, there may be a Model called *Book* with attributes *title* and *author* and a Collection called *Library* that contains multiple books. Collections also emit events when updated. Both Models and Collections can persist their state to a web server that implements a REST API over HTTP. MiddGuard replaces the REST API persistence with a similar one implemented with WebSockets. Backbone.js Views are pieces of user interface. They render HTML in the browser and listen to events emitted from Models and Collections to update themselves. MiddGuard’s front-end user interface is implemented using Backbone.js Views. Visualization modules extend a MiddGuard View, which is inherited from a Backbone.js View, to support View Reference Counting and automatic layout in MiddGuard’s browser environment.

The browser, front-end, client, and other variants are all used to refer to the web browser, where MiddGuard’s user interface lives. The browser is a non-traditional environment for visual analytics, which are often implemented as desktop applications to achieve higher performance through OS native code over JavaScript, which runs non-natively in the browser. However we were inspired by the expressiveness and ease with which we could create interactive visualizations with tools like D3.js [4], a JavaScript library for manipulating HTML, CSS, and SVG in the DOM based on data, and decided to implement MiddGuard as a web application.

4.2 Data-flow Model

MiddGuard’s data-flow model allows arbitrary nodes, each with their own idea of input and output, to be chained together in a graph of data transformations and visualizations. Nodes are reusable units of code, so multiple instances of the same type of node, or module, can coexist in a single investigation. Connections between nodes allow data to pass between them.

4.2.1 Analytic Nodes

One of the issues with the first version of MiddGuard was no integrated way to create and represent the preprocessing scripts used to transform data before visualization. These scripts did most of the work to setup and populate the database, so they were a major component missing from MiddGuard’s idea of the analytic process. Nodes address this problem, creating a flexible representation within MiddGuard of the data processing phase of an investigation. In this section we will address the implementation of analytic nodes. Visualization nodes and their differences with respect to analytic nodes will be addressed in a subsequent section.

Analytic nodes are instances of modules, made unique from one another by the data they generate. MiddGuard is backed by a relational database where nodes are each assigned their own table. They use this table to persist their data. Nodes generate their data using their module’s handler function, invoked from a button press in the user interface. Nodes can be created throughout an investigation and multiple nodes can be created from each module, so a node’s table is created just before its handler is called.

Analytic modules specify a function that will be used to create all of its nodes tables. That function is passed in the name of the table to create and a connection to the database in the form of an SQL generator called Knex.js [9]. The function uses the

connection and table name to generate the schema for its tables.

Nodes are not standalone scripts, they can work together to perform complex transformations, just as a developer might run one script after the other. Each node can output its data and receive input from other nodes. Inputs and outputs are passed into the node's handler function so it can use one to generate the other. The combination of input and output is a node's *context*. Creating a node's contextual output involves only the node itself. Every node has exactly one output, its own table. Other nodes that receive input from it are simply querying that table. The output passed into a node's handler is a Knex.js database connection already assigned specifically to perform statements on the node's table. Creating a node's contextual inputs, however, requires analyzing its connections to other nodes.

4.2.2 Connections

Connections a two-level protocol of node to node connections and intra-connection name mappings, used to determine the input passed into one node from another. Each node can have multiple named inputs, referred to as *input groups*. Each input group can have exactly one connection to another node, referred to as an *output node*. We refer to the parent node of an input group as the *input node*.

A mapping from an input group to an output node creates a mapping from that input group's name to the output node's table. This mapping is stored as a key-value pair where the key is the input group, and the value is the output table name.

When MiddGuard generates the contextual inputs for a node, they key value mapping allows developers to use the input group names they picked for the module to look up the values to access the input data. For the input context, the table name is translated to a combination of table name and a Knex.js accessor. The table name, while unnecessary for queries that only use that table, allow full flexibility for more advanced queries,


```

    "connections": [
      {
        "output": "handle",
        "input": "handle"
      },
      {
        "output": "tweet",
        "input": "tweet"
      },
      {
        "output": "timestamp",
        "input": "timestamp"
      }
    ]
  }
}

```

Listing 4.1: A node’s connection configuration. The node has a connection from its input group “tweets” to the node with id 9.

4.2.3 Connection Storage

The connections generated within the graph editor are stored in MiddGuard’s table of nodes, as a JSON string in the same row as their corresponding input node. We considered multiple factors when deciding how to store connections in the database. We wanted a storage method that was portable, efficient, and convenient. Portability meant that we could easily export the configuration of nodes and connections to a text file so they could be read back in and the graph could be reassembled in a different system. Efficiency was determined by the number of database operations required to access the configuration. This was important since we have to read and write connections whenever

a node is accessed or modified in the graph editor. Convenience meant that it was not overly complex to access and modify the connections from a programming perspective.

In addition to the JSON string storage method we implemented, we considered storing connections and nodes in separate tables, with either each column-level connection in its own row or each group of column-level connections in a row. The former performed no grouping amongst column mappings, while the latter grouped each input group's columns in a single row.

The first option (each column mapping has its own row) was appealing since it took advantage of the relational database, using foreign keys to associate column mappings with their nodes. However, this method is less portable since it requires multiple steps to export all the node information and their associate column mappings from the database to a structured text file. It is also less efficient since it requires reading a row from the database for every column mapping, in addition to a row for every node. Finally, it would be less convenient to develop with because it would require more queries to the database to obtain all the information to construct the graph than if we stored the connection information close to the nodes.

For similar reasons, we ruled out the second option of storing all column level connections in a row, grouped by their input group. This seemed like a poor compromise between storing all column mappings separately and storing all connection information with their nodes. We would lose the elegance of conforming to the facilities of a relational database, and still have to query the database multiple times to assemble a graph or export/import the data.

The implemented method of storing a node's connection in the same database table row as the node, in a JSON string, satisfied all our requirements. It is portable: JSON is common format to export human readable configuration. We can simply query all nodes and write out their metadata and JSON string as connections. It is efficient to access

nodes and connections to construct a graph. All of a graph's nodes and connections can be accessed by reading n rows from the database, where n is the number of nodes in a graph. It is convenient to work with this format, since all the connection data for a node can be obtained by calling JavaScript's built-in `JSON.parse` method on a node's `connections` column.

4.2.4 Context Generation

A node's connections can be edited in the graph editor until runtime, when a node's handler function is executed. At this point, MiddGuard makes a query for the node in the database and retrieves its stored connections. Parsing the connections JSON string lets MiddGuard access the mapping of input groups to output nodes and the mappings of column names between nodes. MiddGuard makes additional queries to determine the table names of connected output nodes. With just this information, MiddGuard can construct the dynamically generated context to pass into the handler function. Listing 4.2 is a sample of the context passed into one of the same "Time by Day/Hour" nodes whose connection was previously listed. At the top level it includes `inputs` and `table`. `inputs` is an object mapping each of the nodes input groups to data about the connected output node. Within `inputs` are: `knex`, an instance of the Knex.js SQL generator [9], used to access the table connected to an input group; `cols`, the column-level mapping between the node's input group and the connected output node's column names; and `tableName`, the name of the connected output node's table name. `cols` and `tableName` are meant to give access to the information available for more advanced queries, such as table joins.

The other top-level key in the context, `table`, gives access to the output table for this node. Like each input group in `inputs`, it has a `knex` accessor to generate SQL to query the database, and a `name`, which is the node's own table name. `table`, the

output, doesn't need a column mapping, since the column names are the same as the ones the node has assigned itself as outputs.

```
{
  inputs: {
    tweets: {
      knex: [Object],           // database connection instance
      cols: {
        handle: 'handle',
        tweet: 'tweet',
        timestamp: 'timestamp'
      },
      tableName: 'download-tweets-danarsilver_1'
    },
    table: {
      knex: [Object],           // database connection instance
      name: 'aggregate-time_2'
    }
  }
}
```

Listing 4.2: The context passed into a “Time by Day/Hour” node’s handler function.

Having to make additional queries to access output nodes’ table names is a potential source of inefficiency not addressed by our connections storage format. A way around this would be to duplicate the table name each time it appears in a connections JSON string. We decided against duplicating the data and in favor of making additional database queries instead to avoid fragmenting the information, should the table name change. Should we need to update a node’s table name, it can be done once for the row, rather than having to update the connections string in all other connected nodes.

4.3 Visualization Nodes

Our model for visual analytics is incomplete without the visualizations themselves. We include visualizations in the data-flow model as their own nodes, which we refer to as *visualization nodes*. By integrating visualizations into the data-flow model, we can pass data transformed by the analytic nodes directly into our visualizations.

Visualization nodes, like analytic nodes, are added from modules in the graph editor. They have input groups that can be connected to output nodes, and column mappings between the two nodes on the ends of the connections. The primary difference between analytic nodes and visualization nodes is that the handler for a visualization node is a newly instantiated Backbone.js View [2] that is rendered in the web client.

The instantiated view for a visualization node has an instance method called `createContext`, which can be called to dynamically generate the context for a view, just as the MiddGuard generates the context for an analytic node on the back-end and passes it into the handler function. The context for a visualization node has the same structure as that of an analytic node, without the output, since a visualization node's output is a visualization, rather than a table of data.

Additionally, the Knex.js accessors for each input group are replaced with instances of Backbone.js Collections (with a new key aptly named `connection`), which can be used like the Knex.js accessor to access the data from output node connected to that particular input. MiddGuard instantiates a Backbone collection for each analytic node and a corresponding endpoint on the back-end to transmit the analytic node's data to the collection, as required by a visualization node.

Backbone.js and consequentially MiddGuard visualization nodes have are not reliant on library or framework to manipulate the DOM and render a visualization. This keeps MiddGuard flexible for any toolchain a developer wants to use to create visualizations.

Since only the representation of a visualization as a node and not the underlying

structure of a visualization changed from the previous version of MiddGuard, View Reference Counting still works completely.

A potential improvement in the implementation of visualization nodes would be to only instantiate collections for analytic nodes that output to visualization nodes. Other nodes' data will never be accessed, so it is not necessary to maintain collections on the front-end or the endpoints on the back-end to transmit data to them. However, this is a low-priority improvement since there is little overhead in terms of memory usage to create an empty connection on the front-end or add the event listeners that handle data transmission to Node.js's event loop on the back-end.

4.4 Visual Programming

Visual programming abstracts away the details of the data-flow model within MiddGuard as described in the previous sections, and the independent implementation details of each node. A major motivation for MiddGuard is to facilitate quick construction of complex visual analytic tools. MiddGuard's system for visual programming allows investigators to quickly compose data transformations and visualizations. The visual component creates an expressive representation of the steps to reproduce a visualization.

The visual programming interface takes place in the three panels of the graph editor, seen in figure 4.1. The left panel, titled "Modules", lists all modules from which nodes can be instantiated. Clicking a module's button in the list adds a node of that type to the canvas in the middle panel.

The middle panel's canvas is a free-form space limited by the height of the window and a 500 pixel width constraint. Nodes, once added to the canvas, are outlined circles that can be rearranged and connected to one another. Analytic nodes and visualization nodes are outlined in blue and orange respectively, to make them easy to differentiate.



Figure 4.1: MiddGuard’s graph editor user interface, open on a graph named “Compare Tweets”. On the left, the modules panel lists all loaded modules, from which nodes can be created. In the center, the graph editor canvas has seven nodes initialized from their respective modules, and connections between the nodes. On the right, the detail panel shows the column mappings between the “Difference by Hour” node and its connections to two “Time by Day/Hour” nodes.

Figure 4.2 shows an analytic node with all its elements for user interaction in view. The cross in the upper left corner is used to drag the node around the canvas. Allowing nodes to be draggable is a simple solution to problem of node layout. A downside is the additional effort and time required on the part of the user to position and reposition nodes in the canvas, but this is outweighed by both its simplicity to implement over a layout algorithm and the flexibility for the user to customize the graph view as best appeals to their idea of the investigation.

The “play” button, located in the top right of each node abstracts both analytic and

visualization nodes' action. In an analytic node clicking play calls its handler function. In a visualization node, the play button creates a new instance of a visualization. Pressing a visualization node's play button again removes that visualization from the browser window. Like the graph editors, stack horizontally in the browser window. The user can scroll through them from left to right.

While web scrolling is typically done vertically, we implemented view layout horizontally, since MiddGuard was designed to be used on the same system used for the preliminary VAST 2014 and VAST 2015 investigations. These investigations used a system of three 27 inch displays arranged side by side [1].

Each node contains two text indicators: in the center of the node in black is the node's module type. This is a visual indicator of the operation that will occur or visualization that will be rendered. Just below is the node's status indicator, one of "Not run", "In progress", or "Completed" in red, yellow, or green, respectively. The status indicates whether the handler function has already been invoked. Investigators ultimately use the node's status to determine when a visualization is able to be rendered in the browser. Only once all a visualizations dependent nodes have been run and have a status of "Completed", can a visualization be rendered.

The connections between nodes' inputs and outputs are key components in the visual programming interface. They represent connecting code paths and passing data from one node to another. A connection can be created from one node to another by selecting one green input group indicator seen at the top of the node in figure 4.2 and one red output indicator like the one seen at the bottom of the same node. The selected input and output connectors are outlined with a black stroke. It is possible to connect a node's input to its own output, however this would result in no operation since the data required for the input would not exist at runtime. Since nodes can accept input from multiple outputs, hovering an input group indicator opens a tooltip with the name of the input

group under the mouse to aid the investigator in creating the correct mapping.

Clicking a node widens its outline and opens the node's connections in the detail panel, seen on the right of figure 4.1. The detail panel lists each input group's column-level connections, grouped by that input group, and organized so output columns are on the left in red, and input columns are on the right in green. When a connection is made in the graph editor, MiddGuard attempts to automatically match columns based on the names. Any columns that don't match appear below the matched ones in gray. Columns can be connected manually in the same way as nodes: by clicking to select an output and an input to connect. The columns names in each group re-render to indicate the pairing after the connection has been made manually.

The similarity between interactions to edit connections at both the node and column level and the color coding of inputs and outputs in both the graph editor canvas and the detail panel is intentional, meant to make graph construction intuitive for an investigator. The goal of visual programming is to reduce the complications for an investigator to create a complex program. A familiar, easy to learn user interface promotes quick, simple development and reduces the cognitive load devoted to MiddGuard as a tool rather than the investigation itself.

4.5 Extensibility

As mentioned before, the primary motivation for MiddGuard is to create a framework that allows investigators and developers to quickly and effectively create visual analytics tools. MiddGuard needs to be able to adapt to any investigation with any types of data and visualizations. To support any data or visualization, MiddGuard can register and load external code referred to as modules. The API for a module is the user interface for developers who work with MiddGuard, and need to quickly construct bespoke data transformations and visualizations.

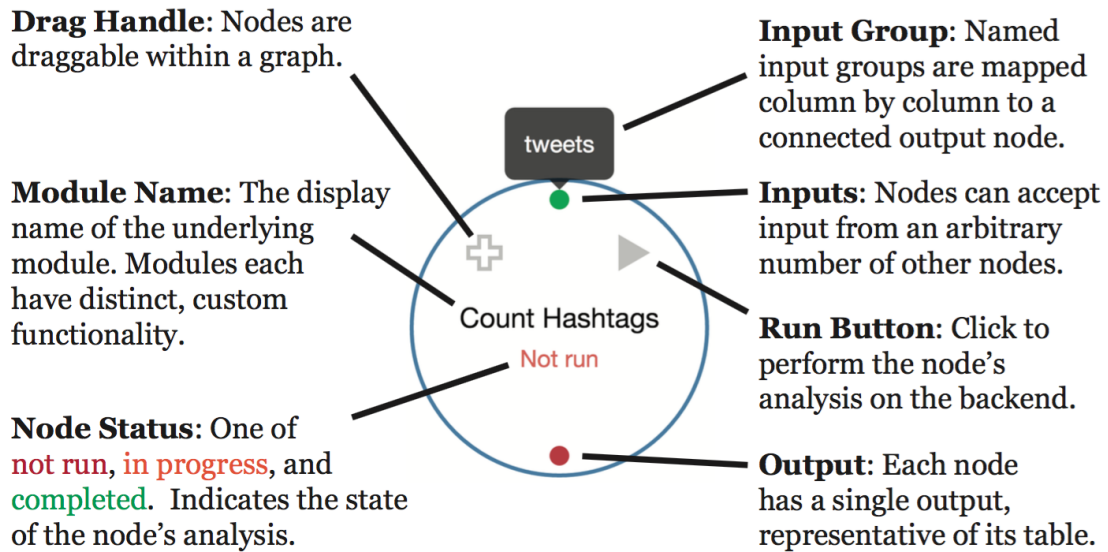


Figure 4.2: An analytic node in a graph. Important features are annotated and the node’s only input group, “tweets”, is moused over to show its accompanying tooltip.

Modules are short and designed to be written quickly. The code in figure 4.3 gives an example of an analytic module that aggregates tweets by the day of the week and hour of their timestamp. This module performs the final step of analysis in the “Compare Tweets” graph of figure 4.1 before data is fed into the visualization. This module is very small, but powerful when instantiated as a node and used in combination with other data transformations. Since MiddGuard modules are just Node.js modules, they can grow as needed, expanding to multiple files as necessary to organize code.

An analytic module can be as simple as one JavaScript file that exports the five objects exported in figure 4.3. Those exports declare inputs, outputs, how to create a table for the module (`createTable`), and what to do when the module is run (`handle`). The display name is a “pretty” version of the module’s name in the file system used to label modules and nodes in the user interface.

The `createTable` and `handle` functions are passed in the node’s dynamically generated context based on its input connections and the name MiddGuard gives its table as described in the previous sections on connections and context. The `handle` function

in figure 4.3 demonstrates the use of its context. It uses `context.inputs.tweets.knex` (lines 28 and 38) to access the table where its input group *tweets*, and `context.inputs.tweets.cols.timestamp` (line 29) to get the name of the timestamp column for that same input group, which it later (line 41) uses to access the timestamp attribute for each tweet. On line 48, the function uses `context.table.knex` to write data to its output.

Visualization modules are simpler than analytic modules in terms of exports, but often more complicated since they have to render a visualization in the browser. It is useful to separate the code for a visualization module into a main file, *index.js*, that exports its configuration and directory, *static*, at the same level, which contains the front-end visualization code.

Figure 4.4 is the configuration code for a sample visualization module called “Hours Heatmap”. Like an analytic module, it exports inputs, outputs, and a display name. To render on the front-end, it also exports (from top to bottom) that it is a visualization, the location of its front-end files (in this case, an adjacent directory called *static*), the JavaScript and CSS files required on the front-end, and the name of the main view to initialize and render on the front-end when MiddGuard loads the visualization. The main view is must be a Backbone View included in one of the of the JavaScript files.

After a module is written, it can be added to a MiddGuard web server (Figure 4.5). Like the modules, a MiddGuard-based server is intended to be short and easy to work with. Only a few lines are required to create the server and start listening for connections and adding a module is a single function call to the server’s `module` function.

Using a simple function call to load modules, rather than discovering them automatically, has a few advantages: it makes the investigation explicit about its module dependencies, raising an error immediately if a required module is missing; it allows a specific server to use different names (the first parameter passed to `module`) to identify

the module in case of a naming conflict between two or more modules; and it allows the user to install and require a module from Node.js's package manager, rather than relatively from the file system.

4.6 Real-time Collaboration

MiddGuard supports asynchronous and synchronous collaboration between multiple developers. Asynchronous collaboration is common in a web application. For example, User A makes changes, which are persisted to a data store. User B logs in some time later and the changes User A made are loaded from the database so User B can view them.

Synchronous collaboration is more difficult to implement. Web application communications are largely based on the HTTP protocol. Data is transferred from the web server to the client in an HTTP session, which is made up of a request from the client and a response from the server. The client must initiate an HTTP request before the server can send data. This is problematic for real-time communications. Like in the asynchronous example, User A might make a change, which should be immediately pushed to all other connected clients. User A can make an HTTP request to tell the server about the change, but there is no way for the server to tell other clients about the change immediately. With HTTP, User B must explicitly request the update, which requires either knowing when to check for an update (unreasonable) or continuously polling the server for changes (inefficient).

WebSockets help solve real-time communications, and are implemented in place of HTTP for all of MiddGuard's server-client communications after a user is authenticated and logged in. WebSockets is a bidirectional event-driven communication protocol designed for browsers and servers to exchange data without relying on HTTP requests and responses [10]. WebSockets are layered on TCP. The connection from the browser to

the server is initiated with the HTTP Upgrade header and client-server handshake after the browser has received a traditional HTTP response from the server with the code to perform the Upgrade request [7].

The MiddGuard server registers WebSocket event handlers for its internal components and for nodes' data. Data on the front-end is structured using Backbone.js Models and Collections, which traditionally use HTTP to perform create, read, update, and delete (CRUD) operations. We use third-party libraries, Backbone.ioBind and Backbone.ioSync, to replace the HTTP requests with a similar protocol using WebSocket events. A HTTP request `POST /graphs` becomes `socket.emit('graphs:create', data)`. Emitted from the browser, these events offer no real advantage of their corresponding HTTP requests. The use case for WebSockets is emitting events and data from the server to the client, which is impossible over HTTP. With the connection open, we can send events from the server to the client to create, update, and delete (the server does not need to read data from the client) Backbone Models and Collections whenever the data change on the server, enabling real-time updates and collaboration for clients.

```

1  var _ = require('lodash');
2  var Promise = require('bluebird');
3  var moment = require('moment');
4
5  exports.inputs = [
6    {name: 'tweets', inputs: ['handle', 'tweet', 'timestamp']}
7  ];
8
9  exports.outputs = [
10   'handle',
11   'day',
12   'hour',
13   'count'
14 ];
15
16 exports.displayName = 'Time by Day/Hour';
17
18 exports.createTable = function(tableName, knex) {
19   return knex.schema.createTable(tableName, function(table) {
20     table.string('handle');
21     table.integer('day');
22     table.integer('hour');
23     table.integer('count');
24   });
25 };
26
27 exports.handle = function(context) {
28   var tweets = context.inputs.tweets,
29       timestampCol = context.inputs.tweets.cols.timestamp,
30       week = [];
31
32   _.range(24).forEach(function(hour) {
33     _.range(7).forEach(function(day) {
34       week.push({day: day, hour: hour, count: 0});
35     });
36   });
37
38   return tweets.knex.select('*')
39     .then(function(tweets) {
40       tweets.forEach(function(tweet) {
41         var m = moment(tweet[timestampCol]),
42             day = +m.format('d'),
43             hour = +m.format('H');
44
45         _.find(week, {day: day, hour: hour}).count++;
46       });
47
48       return context.table.knex.insert(week);
49     });
50 };

```

Figure 4.3: Code for an example analytic module.

```
1  var path = require('path');
2
3  exports.inputs = [
4    {name: 'hours', inputs: ['day', 'hour', 'count1', 'count2']}
5  ];
6
7  exports.outputs = [];
8
9  exports.displayName = "Hours Heatmap";
10
11 exports.visualization = true;
12
13 exports.static = path.join(__dirname, 'static');
14
15 exports.js = [
16   "hours-heatmap-view.js"
17 ];
18
19 exports.css = [
20   "hours-heatmap.css"
21 ];
22
23 exports.mainView = 'HoursHeatmapView';
```

Figure 4.4: Contents of the main configuration file, *index.js*, for an example visualization module, “Hours Heatmap”.

```

1  var middguard = require('middguard');
2
3  var app = middguard({
4    // database
5    'knex config': require('./knexfile'),
6
7    // sessions
8    'secret key': process.env.SECRET_KEY || 'major 🔑'
9  });
10
11  // Hours Heatmap Visualization Module
12  app.module('hours-heatmap', require.resolve('./hours-heatmap'));
13
14  // Time by Day/Hour Analytic Module
15  app.module('aggregate-time', require.resolve('./aggregate-time'));
16
17  // Start the server
18  var port = process.env.PORT || 3000;
19  app.listen(port, function () {
20    console.log('Listening on port %d...', port);
21  });

```

Figure 4.5: Code for an investigation’s MiddGuard server. It creates an instance of the MiddGuard server, passing in the database configuration from a Knexfile [9] and a secret key to encrypt authenticated session data. This investigation uses the two modules from figures 4.3 and 4.4 and registers them with calls to `app.module`. Finally the server starts listening for connections on port 3000.

CHAPTER 5

DISCUSSION

5.1 Use Case

We constructed a small investigation into Twitter data to help implement and test MidGuard as we implemented the framework. Using tweets from two users' timelines, we wanted to determine who tweets more each hour of each day of the week.

To find an answer we wrote four analytic modules and two visualization modules. Our first two analytic modules accessed the Twitter API to download tweets from the two subjects, “@DanaRSilver” and “@jack”. These are also the names of the respective modules. Next, we wrote “Time by Day/Hour”, which uses tweets' timestamps to aggregate them by day of the week and hour of day. Our last analytic module, “Difference by Hour”, computes the difference between counts for each combination of day and hour and groups the two counts into a single table. We created a new graph and connected the “@DanaRSilver” and “@jack” nodes each to a “Time by Day/Hour” and fed those into a “Difference by Hour” node. Figure 5.1 shows the complete graph.

Since our goal was to figure out who tweets more at each combination of hour of the day and day of the week, we wrote a visualization called “Hours Heatmap”, a bubble chart with hours on the x axis and days on the y axis (Figure 5.2). Two circles, or bubbles, are drawn at entry in the chart, one for each person. The circles' radii are mapped to the number of times the corresponding person tweeted that hour and day. Mousing over a pair of circles adds a tooltip with the exact count.

From the “Hours Heatmap” visualization we are able to answer our question. We can look to any particular day and hour and see who tweets more. Wednesday at 12pm, for example, Dana tweets more than Jack. Dana tweeted nine times and Jack tweeted twice. We are also able to identify some patterns in the tweets. Both people rarely tweet

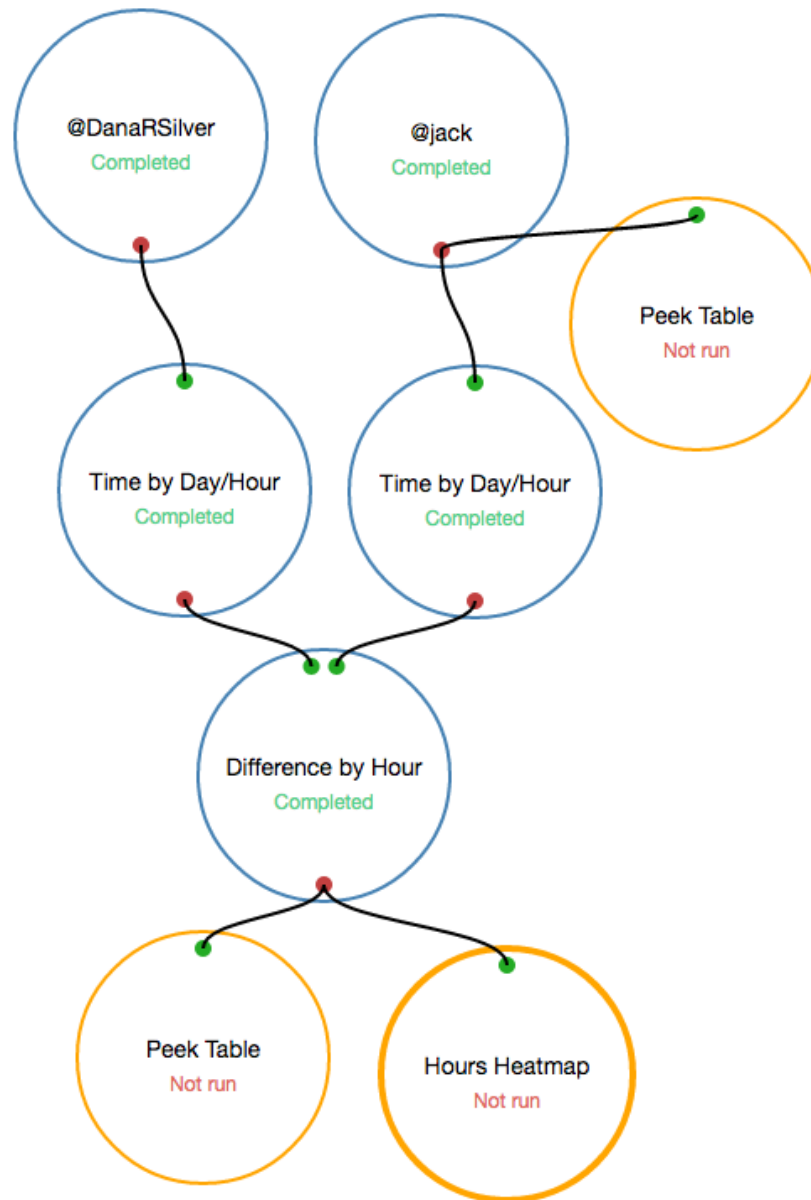


Figure 5.1: The complete graph from the mock investigation used to develop and test MiddGuard.

late at night, and never between 4am and 6am. Jack is more active on Saturday than Dana and both get a late start on the weekends.

While we were investigating our primary question, we wanted to look at the data we received from the Twitter API as well, to make our investigation more transparent, and to test that we had downloaded tweets correctly without having to work with the database

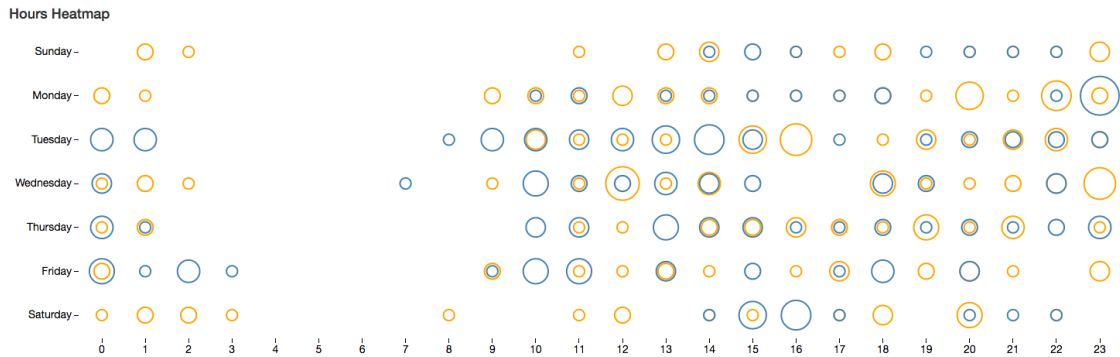


Figure 5.2: The “Hours Heatmap” visualization from the mock investigation used to develop and test MiddGuard. @DanaRSilver’s tweets are orange, @jack’s are blue. Circle radii are mapped to the number of times each person tweeted in that hour and day of the week.

outside MiddGuard. We wrote a visualization “Peek Table” that takes any input and renders it as a table. We hooked this up to both the “@DanaRSilver” and “@Jack” modules and could immediately tell that our download had worked as intended. Since we could see the text of the tweets, we could also see that Jack retweets much more often than Dana.

5.2 Areas for Improvement

The mock investigation into @DanaRSilver and @jack’s tweets revealed two areas for improvement in MiddGuard. The first is that modules only can change context from between nodes with respect to the incoming and outgoing data. We have two almost identical modules to download @DanaRSilver’s tweets and @jack’s tweets. The only difference is the Twitter handle accessed. When one of our goals is reuse of the data transformation logic, it does not make sense to repeat logic just to change a variable. We could improve on this by allowing developers to define variables that can change from node to node and create an interface for investigators to define that variable for the node. This would have allowed us to write one module that downloads tweets, create

two nodes from it, and pass “@DanaRSilver” and “@jack” in as variables.

Developing the modules was challenging, since it was hard to test if the transformation logic worked. We eventually created the “Peek Table” visualization module to check the table contents, however this required creating multiple nodes and running the user interface to test. There is no way to remove data from a node, so if the transformation was applied and saved data incorrectly, additional nodes would have to be created to test the module again. This issue could be solved with a procedure to pass data through a module without creating a node in the user interface, and without persisting that data to the node’s table. Besides simpler development, this solution would make it substantially easier to write tests for modules, which in MiddGuard’s current state would require creating a database, starting the web server, and manipulating the user interface in a web browser.

Outside the areas of improvement discovered during the use case, MiddGuard could be improved to better incorporate visualization nodes into the data flow. Visualization nodes should be able to modify data in the database and have their own data output to support brushing, linking, and detail in the browser. Visualization nodes need to be able to modify data, or at least report user interactions so the server can respond to them. This enables operations like those used in the VAST 2014 Challenge, where we selected car destinations to associate with points of interest on a map. Like analytic nodes take in data to transform, a new type of node, “Event Nodes” could take in events and associated data (like a click the destination under the mouse) and perform a data transformation to respond to that event.

A second output, for visualization nodes to output a subset of the data they take in, could support brushing, linking, and detail interactions within the data-flow model, rather than in a separate and opaque global state with no visual representation. Other visualizations would take the output as their input, using it to render their own visual-

ization. For example, the “Hours Heatmap” from the mock tweet investigation could output the data from a selected day of the week, which would be read by a bar chart visualization and used to render a bar chart of cumulative tweets per day of the week. As the selected day changes in the “Hours Heatmap”, the bar chart would receive updated data and rerender. Since analytic nodes output data following some transformation step, it is intuitive to the data-flow model that visualization nodes do the same. Building interactions into the data-flow model increases the transparency and reproducibility of the investigation.

CHAPTER 6

CONCLUSION

MiddGuard is an effective web framework to create complex visual analytics tools quickly. Its data-flow model and its visual representation allows investigators to see exactly what steps were applied to a dataset to produce a visualization, increasing transparency throughout the investigation and reproducibility of results afterwards. While the previous version of MiddGuard required developers to load and preprocess data with scripts external to the framework, our use case demonstrated the ease with which we can write those scripts into the current version of MiddGuard and represent them in the data-flow model.

Connections between nodes are a useful abstraction to simulate data flowing through the graph and generate the contextual input required to actually send data from node to node, while persisting it to a database. The model for extensibility, analytic and visualization modules, is able to encompass operations that take place in the back-end and front-end and offer plug-and-play capabilities without sacrificing flexibility or developer choices for module implementation. The synchronous communication protocol implemented over WebSockets allows developers to work together to develop tools and share results.

By abstracting away the details required to structure components and communicate data to a simple graph builder and built in tooling, MiddGuard lets analysts focus on writing and using the tools they need to analyze data in a timely manner. When taken together, these features make MiddGuard a novel tool for visual analytics.

We plan on making MiddGuard open source software. Open sourcing MiddGuard will encourage contributions to the core framework from outside collaborators. As other users investigate data with MiddGuard and write their own modules, they can contribute these back to the community to create a resource of analytic and visualization modules.

APPENDIX A

CHAPTER 1 OF APPENDIX

```
index.js                                middguard/application.js
1 'use strict';
2
3 module.exports = require('./middguard/
  middguard');
4
5
6
7
8 var path = require('path');
9 var http = require('http');
10
11 var bodyParser = require('body-parser');
12 var cookieParser = require('cookie-parser');
13 var express = require('express');
14 var socketio = require('socket.io');
15 var ios = require('socket.io-express-session
  ');
16 var session = require('express-session');
17 var KnexSessionStore = require('connect-
  session-knex')(session);
18 var _ = require('lodash');
19
20 /**
21 * Application prototype methods to extend
22 * the express application prototype.
23 */
24
```

```

25 var app = exports = module.exports = {};
26
27
28 app.middlewareInit = function () {
29   this.middlewareExpressMiddleware();
30
31   var server = http.createServer(this);
32   this.set('http server', server);
33
34   var io = socketio(server);
35   this.set('io', io);
36   this.middlewareSocketMiddleware();
37
38   io.on('connection', require('./socket'));
39
40   require('./routes')(this);
41 };
42
43 /**
44  * Setup the express middleware for
45  *   MiddelGuard.
46  * @private
47  */
48
49 app.middlewareExpressMiddleware = function
50   middlewareExpressMiddleware() {
51   this.use('/static', express.static(path.
52     join(__dirname, '..', 'static')));
53
54   var knex = require('knex')(this.get('knex
55     config'));
56   var sessionStore = new KnexSessionStore({
57     knex: knex});
58   this.set('sessionStore', sessionStore);
59
60   // Set up ORM middleware
61   require('./config/bookshelf')(this);
62
63   this.use(cookieParser(this.get('secret key
64     ')));
65   this.use(bodyParser.urlencoded({extended:
66     true}));
67   this.use(bodyParser.json());
68
69   this.set('session', session({
70     store: sessionStore,
71     secret: this.get('secret key'),
72     resave: true,
73     saveUninitialized: true,
74     cookie: {maxAge: 7 * 24 * 60 * 60 *
75       1000} // 1 week
76   }));
77   this.use(this.get('session'));
78
79   this.set('views', path.join(__dirname, '
80     views'));
81   this.set('view engine', 'jade');
82
83   app.middlewareSocketMiddleware = function
84     middlewareSocketMiddleware() {
85     var io = this.get('io');
86     var session = this.get('session');
87
88     io.use(io.s(session));
89
90     io.use((socket, next) => {
91       socket.bookshelf = this.get('bookshelf')
92       ;
93       next();
94     });
95
96

```



```

86 };
87
88 /**
89  * Register an analytics module with the `
  middleware` app.
90  *
91  * @return `middleware.Analytics`
92  * @public
93  */
94 app.module = function module(name,
  requirePath) {
95   var Bookshelf = this.get('bookshelf');
96   var AnalyticsModule = Bookshelf.model('
    AnalyticsModule');
97   var register = Bookshelf.collection('
    analytics');
98
99   var attributes = require(requirePath);
100
101   if (_.has(attributes, 'visualization')) {
102     this.use('/', modules/${name}`, express.
      static(attributes.static));
103   }
104
105   register.add(new AnalyticsModule({
106     name: name,
107     requirePath: requirePath,
108     displayName: attributes.displayName,
109     inputs: attributes.inputs,
110     outputs: attributes.outputs,
111     visualization: attributes.visualization,
112     main: attributes.visualization ?
      attributes.mainView : null
113   }));
114 };
115
116 /**
117  * Listen for connections.
118  *
119  * A node `http.Server` is returned, with
  this
  application (which is a `Function`) as
  its
  callback.
120  *
121  * This is the same as `express.listen`, but
  uses
  the already created server, rather than
  creating
  a new one in `listen`. The `http.Server`
  must
  already be created to setup socket.io.
122  *
123  * @return {http.Server}
124  * @public
125  */
126
127 app.listen = function listen() {
128   var server = this.get('http server');
129   return server.listen.apply(server,
    arguments);
130 }

```

middguard/middguard.js

```

1 'use strict';
2
3 /**
4  * Module dependencies.
5  */
6
7 var _ = require('lodash');
8 var express = require('express');
9 var proto = require('./application');
10
11 /**
12  * Expose `createApplication()`.
13  */
14
15 exports = module.exports = createApplication
16 ;
17 /**
18  * Create a middguard application.
19  *
20  * @return {Function}
21  * @public
22  */
23
24 function createApplication(settings) {
25   var app = express();
26
27   _.each(settings, (value, key) => {
28     app.set(key, value);
29   });
30
31   _.extend(app, proto);
32
33   // expressConfig(app);
34   //

```

```

35 // var server = http.createServer(app);
36 // var io = socketio(server);
37 // app.set('io', io);
38 //
39 // var sessionSockets = new SessionSockets
40 //   (io,
41 //     app.get('sessionStore'),
42 //     app.get('cookieParser'));
43 // bookshelfConfig(app);
44 //
45 // // require('./middguard/loaders/
46 //   models_loader')(app);
47 // // require('./middguard/loaders/
48 //   analytics_loader')(app);
49 // // sessionSockets.on('connection', require
50 //   ('./middguard/socket'));
51 // // require('./middguard/routes')(app);
52 app.middguardInit();
53
54 return app;
55 };

```

```

1  var _ = require('lodash'),
2  pluralize = require('pluralize'),
3  analyst = require('./analyst'),
4  message = require('./message'),
5  modules = require('./modules'),
6  node = require('./node'),
7  io = require('socket.io')();
8
9  module.exports = function (socket) {
10 var Bookshelf = socket.bookshelf;
11
12 // Only set up sockets if we have a logged
13   in user
14 if (!socket.handshake.session.user) return
15   ;
16 // Set up sockets middleware internal
17   sockets
18 socket.on('messages:create', (data, cb) =>
19   message.create(socket, data, cb));
20 socket.on('messages:read', (data, cb) =>
21   message.readAll(socket, data, cb));
22 socket.on('modules:read', (data, cb) =>
23   socket.on('modules:read', (data, cb) =>
24     analyst.read(socket, data, cb));
25 socket.on('analysts:read', (data, cb) =>
26   analyst.readAll(socket, data, cb));
27 socket.on('node:connect', (data, cb) =>
28   node.connect(socket, data, cb));
29 socket.on('node:run', (data, cb) => node.
30   run(socket, data, cb));
31
32 }
33
34 // Set up sockets to call analytics from
35   client
36 // Patched models will automatically emit
37   create, update, and delete events
38 // Bookshelf.collection('analytics').each() {
39   function (analyticsAttrs) {
40     var name = analyticsAttrs.get('name')
41     ;
42     var requirePath = analyticsAttrs.get
43       ('requirePath');
44     socket.on('analytics:' + name,
45       function (data, callback) {
46         require(requirePath)(Bookshelf,
47           data);
48       });
49     // });
50   };
51 }
52
53 var Graph = Bookshelf.model('Graph');
54 patchModelToEmit(socket, 'graph', Graph);
55 setupSocketEvents(socket, 'graph', Graph);
56
57 Bookshelf.model('Node').fetchAll()
58 .then(nodes => nodes.each(node => node.
59   createReadSocket(socket)));
60
61 // Set up sockets to call analytics from
62   client
63 // Patched models will automatically emit
64   create, update, and delete events
65 // Bookshelf.collection('analytics').each() {
66   function (analyticsAttrs) {
67     var name = analyticsAttrs.get('name')
68     ;
69     var requirePath = analyticsAttrs.get
70       ('requirePath');
71     socket.on('analytics:' + name,
72       function (data, callback) {
73         require(requirePath)(Bookshelf,
74           data);
75       });
76     // });
77   };
78 }
79
80 function patchModelToEmit(socket, modelName,
81   model) {
82

```

```

50 if (!model.prototype._emitting) {
51   var _initialize = model.prototype.
      initialize;
52
53   model.prototype.initialize = function ()
      {
54     var args = Array.prototype.slice.call(
        arguments);
55     _initialize.apply(args);
56
57     this.on('created', function (model,
        attrs, options) {
58       // If the model was created on the
        client, we don't want to emit a
59       // create event, since we need to
        assign an id on the creator via
60       // a callback and do a broadcast.
        emit for everyone else.
61       // The create listener will take
        care of this.
62       if (!options.clientCreate) {
63         io.emit(pluralize(modelName) + ':
            create', model.toJSON());
64       } else {
65         socket.broadcast.emit(pluralize(
            modelName) + ':create', model.
            toJSON());
66       }
67     });
68
69     this.on('updated', function (model) {
70       socket.broadcast.emit(pluralize(
            modelName) + ':update', model.
            toJSON());
71     });
72
73     this.on('destroying', function (model)
        {
74       socket.broadcast.emit(pluralize(
            modelName) + ':delete', model.
            toJSON());
75     });
76   };
77
78   model.prototype._emitting = true;
79 }
80 }
81
82 function setupSocketEvents(socket, modelName
    , model) {
83   // Set up create, read, update, delete
    sockets for each model
84   socket.on(pluralize(modelName) + ':create'
    , function (data, callback) {
85     // Pass clientCreate to save so the
        model won't emit anything on the
86     // created event and confuse the client.
87     // Create is a special case since the
        model on the creating client doesn't
88     // have an id yet.
89     new model().save(data, {clientCreate:
        true})
90     .then(function (newModel) {
91       callback(null, newModel.toJSON());
92     })
93     .catch(function (error) {
94       throw new Error(error);
95     })
96   });
97
98   socket.on(modelName + ':update', function
    (data, callback) {

```


middguard/socket/modules.js

```

1 /**
2  * Respond to the modules:read event from a
   connected client.
3  * Emits all registered analytics modules.
4  *
5  * @return undefined
6  * @private
7  */
8
9 exports.readAll = function (socket, data,
   callback) {
10   var register = socket.bookshelf.collection
     ('analytics');
11
12   callback(null, register.toJSON());
13 };

```

49

middguard/socket/node.js

```

1 var Promise = require('bluebird');
2 var _ = require('lodash');
3
4 exports.create = function(socket, data,
   callback) {
5   var Node = socket.bookshelf.model('Node');
6
7   new Node()
8     .save(data, {clientCreate: true})
9     .then(node => {
10       node.createReadStream(socket);
11       callback(null, node.toJSON());
12       socket.broadcast.emit('nodes:create',
         node.toJSON());
13     });
14 };
15
16 exports.readAll = function(socket, data,
   callback) {
17   var Node = socket.bookshelf.model('Node');
18   var nodes = new Node();
19
20   if (data) nodes = nodes.where(data);
21
22   nodes.fetchAll()
23     .then(collection => callback(null,
       collection.toJSON()))
24     .catch(callback);
25 };
26
27 exports.update = function(socket, data,
   callback) {
28   var Node = socket.bookshelf.model('Node');
29
30   new Node({id: data.id})

```

```

31 .save(_.omit(data, 'id'), {patch: true})
32 .then(function(node) {
33   callback(null, node.toJSON());
34   socket.broadcast.emit('nodes:update',
35     node.toJSON());
36   .catch(callback);
37 });
38
39 /* Connect data.inputNode at data.inputGroup
   to data.outputNode.
40 */
41 exports.connect = function(socket, data,
   callback) {
42   var Node = socket.bookshelf.model('Node');
43   var modules = socket.bookshelf.collection(
   'analytics');
44
45   var outputNode = new Node({id: data.
   outputNode});
46   var inputNode = new Node({id: data.
   inputNode});
47
48   Promise.all([outputNode.fetch(), inputNode
   .fetch()])
49   .spread(function(outputNode, inputNode) {
50     var outputModule = modules.findWhere({
   name: outputNode.get('module')});
51     var inputModule = modules.findWhere({
   name: inputNode.get('module')});
52
53     // Get outputs list from the
   corresponding output module
54     var outputs = require(outputModule.get('
   requirePath')).outputs;
55
56     // Get inputs list from the
   corresponding input module
57     var inputGroups = require(inputModule.
   get('requirePath')).inputs;
58     var inputs = inputGroups.filter(function
   (group) {
59       return group.name === data.inputGroup;
60     })[0].inputs;
61
62     // The array of connections we'll set on
   the input node
63     var connections = {
64       output_node: data.outputNode,
65       connections: []
66     };
67
68     if (data.connections &&
   validateConnections(data.connections
   , inputs, outputs)) {
69       // Use 'data.connections' if the
   connections are valid
70       connections.connections = data.
   connections;
71
72     } else {
73       // Match input and output names
74       connections.connections =
   connectionsByName(inputs, outputs);
75     }
76
77     inputNode.setInputGroup(data.inputGroup,
   connections);
78     return inputNode.save();
79   })
80   .then(node => {
81     socket.emit('nodes:update', node.toJSON
   ());

```

```

82     socket.broadcast.emit('nodes:update',
83         node.toJSON());
84     }.catch(callback);
85 };
86
87 /**
88  * Validate that all potential inputs and
89  * outputs have inputs and outputs on
90  * with the same name on the respective
91  * nodes.
92  *
93  * @param {Object[]} connections The passed
94  *   in data of connections to set.
95  * @param {Object[]} inputs Named inputs on
96  *   the existing input node.
97  * @param {Object[]} outputs Named outputs
98  *   on the existing output node.
99
100  * function validateConnections(connections,
101  *   inputs, outputs) {
102  *   var potentialInputs = connections.map(
103  *     connection => connection.input);
104  *   var potentialOutputs = connections.map(
105  *     connection => connection.output);
106
107  *   return potentialInputs.length ===
108  *     potentialOutputs.length &&
109  *     inputs.every(input => _.has(
110  *       potentialInputs, input)) &&
111  *     outputs.every(output => _.has(
112  *       potentialOutputs, output));
113  }
114
115  /**
116  * Generate the connections array by
117  * matching names between inputs and
118  * outputs.
119  *
120  * Returns an array with size equivalent to
121  * the cardinality of inputs outputs.
122  *
123  * @private
124  * @param {Object[]} inputs Inputs to match
125  * @param {Object[]} outputs Outputs to
126  *   match
127  */
128 function connectionsByName(inputs, outputs)
129 {
130     return outputs.filter(output => _.indexOf(
131       inputs, output) > -1)
132       .map(output => ({output:
133         output, input: output}));
134 }
135
136 exports.run = function(socket, data,
137   callback) {
138     var Node = socket.bookshelf.model('Node');
139     var modules = socket.bookshelf.collection(
140       'analytics');
141
142     new Node({id: data.id})
143       .fetch()
144       .tap(node => node.ensureTable())
145       .then(node => node.save({status: 1}))
146       .then(function(node) {
147         socket.emit('nodes:update', node.toJSON(
148           ));
149         socket.broadcast.emit('nodes:update',
150           node.toJSON());
151         return node;
152       })
153 }

```



```

131 .then(node => Promise.join(node, node.
132   outputNodes()))
133   .spread(function(node, outputs) {
134     var module = modules.findWhere(({name:
135       node.get('module')}),
136       connections = JSON.parse(node.get('
137         connections')),
138         context = {});
139
140     context.inputs = _.reduce(_.keys(
141       connections), function(inputs,
142       inputGroup) {
143         var groupConnections = connections[
144           inputGroup].connections;
145
146         // Reduce the array of input output
147         // pairs to a single associative array
148         // mapping input to output.
149         var columns = _.reduce(
150           groupConnections, function(
151             connections, pair) {
152               connections[pair.input] = pair.
153                 output;
154               return connections;
155             }, {});
156
157     inputs[inputGroup] = {};
158     inputs[inputGroup].knex = socket.
159       bookshelf.knex(outputs[inputGroup].
160         get('table'));
161     inputs[inputGroup].cols = columns;
162     inputs[inputGroup].tableName = outputs
163       [inputGroup].get('table');
164
165     return inputs;
166   }, {});
167
168   context.table = {};
169   context.table.knex = socket.bookshelf.
170     knex(node.get('table'));
171   context.table.name = node.get('table');
172
173   var handle = require(module.get('
174     requirePath')).handle;
175   return Promise.join(node, handle(context
176     ));
177   })
178   .spread(function(node, result) {
179     return node.save({status: 2});
180   })
181   .then(function(node) {
182     socket.emit('nodes:update', node.toJSON
183       ());
184     socket.broadcast.emit('nodes:update',
185       node.toJSON());
186   })
187   .catch(callback);
188 }
189
190 inputs[inputGroup] = {};
191 inputs[inputGroup].knex = socket.
192   bookshelf.knex(outputs[inputGroup].
193     get('table'));
194 inputs[inputGroup].cols = columns;
195 inputs[inputGroup].tableName = outputs
196   [inputGroup].get('table');
197
198 return inputs;
199 }, {});

```

middguard/models/connection.js

```

1  /**
2  * Register the 'Connection' model in the
   Bookshelf registry.
3  *
4  * Access this model using 'Bookshelf.model
   ('Connection')`.
5  *
6  * @return {Bookshelf.Model}
7  * @private
8  */
9
10 module.exports = function (app) {
11   var Bookshelf = app.get('bookshelf');
12
13   var Connection = Bookshelf.Model.extend({
14     tableName: 'connection',
15
16     from: function () {
17       return this.belongsTo('Node');
18     },
19
20     to: function () {
21       return this.belongsTo('Node');
22     }
23   });
24
25   return Bookshelf.model('Connection',
     Connection);
26 };

```

middguard/models/graph.js

```

1  /**
2  *
3  */
4
5 module.exports = function (app) {
6   var Bookshelf = app.get('bookshelf');
7
8   var Graph = Bookshelf.Model.extend({
9     tableName: 'graph',
10
11     nodes: function () {
12       return this.hasMany('Node')
13     }
14   });
15
16   return Bookshelf.model('Graph', Graph);
17 };

```

```

1 'use strict';
2
3 var _ = require('lodash');
4 var Promise = require('bluebird');
5
6 /**
7  * Register the 'Node' model in the
8  * Bookshelf registry.
9  * @return {Bookshelf.Model}
10  * @private
11  */
12
13 module.exports = function(app) {
14   var Bookshelf = app.get('bookshelf');
15
16   var Node = Bookshelf.Model.extend({
17     tableName: 'node',
18
19     initialize: function() {
20       this.on('creating', this.createTableName);
21     },
22
23     graph: function() {
24       return this.belongsToMany('Graph');
25     },
26
27     status: function() {
28       var statuses = {
29         0: 'Not run',
30         1: 'In progress',
31         2: 'Done'
32       };
33
34       return statuses[this.get('status')];
35     },
36
37     createTableName: function() {
38       return Node
39         .where('module', this.get('module'))
40         .count()
41         .then(count => {
42           return this.set('table', `${this.get(
43             'module'
44           )}_${count + 1}`);
45         });
46       /**
47        * Get a mapping from input group names
48        * to output nodes.
49        * @return a promise for an object
50        * mapping input group name
51        * to a fetched output node
52        */
53       outputNodes: function() {
54         var connections = JSON.parse(this.get(
55           'connections'
56         ));
57
58         return Promise.reduce(_.keys(
59           connections
60         ), function(outputs,
61           inputGroup) {
62           var outputId = connections[
63             inputGroup].output_node;
64
65           return new Node({id: outputId}).
66             fetch()
67             .then(node => {
68               outputs[inputGroup] = node;
69               return outputs;
70             });
71         });
72     }
73   });
74 }

```

```

62     });
63     }, {});
64 },
65
66 /**
67  * Create this node's table if it doesn't
68  * already.
69  */
70 ensureTable: function() {
71     return Bookshelf.knex.schema.hasTable(
72         this.get('table'))
73     .then(exists => {
74         if (!exists) {
75             return this.module().createTable(
76                 this.get('table'), Bookshelf.
77                     knex);
78         }
79     });
80 },
81
82 module: function() {
83     var modules = Bookshelf.collection('
84         analytics'),
85     moduleName = this.get('module'),
86     module = modules.findWhere({name:
87         moduleName});
88
89     return require(module.get('requirePath
90         '));
91 },
92
93 /**
94  * Set an input group on the node's
95  * connections.
96  * The text column "connections" remains
97  * in its stringified JSON state.
98  */
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113

```

```

114   });
115
116   return Bookshelf.model('Node', Node);
117 };

middguard/migrations/20140728124252'initial.js
1 'use strict';
2
3 exports.up = function (knex, Promise) {
4   return knex.schema.createTable('analyst',
5     function (table) {
6     table.increments('id').primary();
7     table.text('username').unique();
8     table.text('password');
9   })
10   .createTable('message', function (table) {
11     table.increments('id').primary();
12     table.integer('analyst_id').references('
13       analyst.id');
14     table.text('state');
15     table.text('content');
16     table.dateTime('timestamp');
17   })
18   .createTable('graph', function (table) {
19     table.increments('id').primary();
20     table.string('name');
21   })
22   .createTable('node', function (table) {
23     table.increments('id').primary();
24     table.integer('graph_id').references('
25       graph.id');
26     table.string('module');
27     table.string('table');
28     table.integer('status').defaultTo(0);
29     table.string('connections').defaultTo('
30       {}');
31   });
32 });
33
34 exports.down = function (knex, Promise) {
35   return knex.schema.dropTable('analyst')

```

```

32 .dropTable('message')
33 .dropTable('graph')
34 .dropTable('node')
35 .dropTable('connection')
    ;
36 };

    middguard/migrations/20160405022013'node'coordinates.js
1  'use strict';
2
3 exports.up = function(knex, Promise) {
4   return knex.schema.table('node', function(
5     table) {
6     table.integer('radius').defaultTo(75);
7
8     // These are the top left coordinates of
9     // the node,
10    // not the center coordinates.
11    table.integer('position_x').defaultTo(0)
12    ;
13    table.integer('position_y').defaultTo(0)
14    ;
15  });
16
17 exports.down = function(knex, Promise) {
18   return knex.schema.table('node', function(
19     table) {
20     table.dropColumns('radius', 'position_x'
21       , 'position_y');
22   });
23 };

```

```

1  var middguard = middguard || {};
2
3  (function() {
4    middguard.entities = {};
5
6    middguard.EntityCollection = Backbone.
      Collection.extend({
7      initialize: function (models, options) {
8        this.url = _.result(options, 'url');
9
10       _.bindAll(this, 'serverCreate', '
          serverUpdate', 'serverDelete');
11       this.ioBind('create', this.
          serverCreate, this);
12       this.ioBind('update', this.
          serverUpdate, this);
13       this.ioBind('delete', this.
          serverDelete, this);
14
15       this.listenTo(this, 'sync', this.
          addViewReferences);
16     },
17     serverCreate: function (data) {
18       var exists = this.get(data.id);
19       if (!exists) {
20         this.add(data);
21       } else {
22         exists.set(data);
23       }
24     },
25     serverUpdate: function (data) {
26       var exists = this.get(data.id);
27       if (exists) exists.set(data);
28     },
29     serverDelete: function (data) {
30
31       // Already deleted from database, so
32       // don't need to model.destroy
33       var exists = this.get(data.id);
34       if (exists) this.remove(exists);
35     },
36     addViewReferences: function (collection,
37       response, options) {
38       var middguard_view_name = options.
39         middguard_view_name;
40
41       // if a view name wasn't passed in we
42       // can't do anything about it
43       if (!middguard_view_name)
44         return;
45
46       console.log('Adding view references
47         for view "' + middguard_view_name +
48         ' " to ' + response.length
49         + ' fetched models.');
```

```

55 static/js/visualization-manager.js
56
57   currentViews = model.get('
58     middguard_views');
59
60   // if the view has already been
61     added
62   if (currentViews.indexOf(options.
63     middguard_view_name) > -1)
64     return;
65
66   // add the view to the model's '
67     middguard_views'
68   currentViews.push(
69     middguard_view_name);
70   model.set('middguard_views',
71     currentViews);
72   });
73   }
74   });
75   }());
76
77   static/js/visualization-manager.js
78   1 var middguard = middguard || {};
79   2
80   3 (function() {
81   4
82   5     middguard.View = Backbone.View.extend({
83   6       className: 'middguard-module',
84   7       fetch: function (collection, options) {
85   8         // set the view name to add to the
86           middguard_views when we create/
87           update
88         // the models
89         options.middguard_view_name = this.cid
90         ;
91
92         // add the entity to this view
93         // so we can check the entities and
94         // remove the view from
95         // middguard_views
96         // when the view is destroyed
97         if (this.middguard_entities.indexOf(
98           collection) < 0) {
99             this.middguard_entities.push(
100               collection);
101         }
102
103         middguard.entities[collection].fetch(
104           options);
105       },
106
107       /* middguard.View.prototype.remove
108       * Extend the view remove function to
109       * remove referenced models
110       *
111       * Important: If you need to extend
112       * remove functionality, you must call

```



```

26 * 'middguard.View.prototype.remove.call
27 (this)' as the super call instead
28 * of the usual 'Backbone.View.prototype
29 .remove.call(this)'.
30 */
31 remove: function () {
32   var viewName = this.cid;
33
34   console.log('About to remove view ' +
35     viewName + '");
36
37   // For each model this view references
38   this.middguard_entities.forEach(
39     function (entityName) {
40       var collection = middguard.entities[
41         entityName];
42
43       // First iteration to remove
44       reference to this model
45       collection.each(function (model, i)
46       {
47         if (model.get('middguard_views').
48           indexOf(viewName) > -1) {
49           removeFromArray(model.get('
50             middguard_views'), viewName);
51         }
52       });
53
54       // Get an array of models from this
55       entity collection to remove
56       var toRemove = collection.filter(
57         function (model) {
58           if (model.get('middguard_views').
59             length === 0) {
60             delete model.attributes.
61               middguard_views;
62
63             * 'middguard.View.prototype.remove.call
64             (this)' as the super call instead
65             * of the usual 'Backbone.View.prototype
66             .remove.call(this)'.
67             */
68             remove: function () {
69               var viewName = this.cid;
70
71               console.log('About to remove view ' +
72                 viewName + '");
73
74               // For each model this view references
75               this.middguard_entities.forEach(
76                 function (entityName) {
77                   var collection = middguard.entities[
78                     entityName];
79
80                   // First iteration to remove
81                   reference to this model
82                   collection.each(function (model, i)
83                   {
84                     if (model.get('middguard_views').
85                       indexOf(viewName) > -1) {
86                       removeFromArray(model.get('
87                         middguard_views'), viewName);
88                     }
89                   });
90
91                   // Get an array of models from this
92                   entity collection to remove
93                   var toRemove = collection.filter(
94                     function (model) {
95                       if (model.get('middguard_views').
96                         length === 0) {
97                         delete model.attributes.
98                           middguard_views;
99
100                         console.log('Removing ' + toRemove.
101                           length +
102                           ' models that are no
103                           longer in use from
104                           collection "' +
105                           entityName + '".');
106                         // remove them without sending
107                         anything to the server
108                         collection.remove(toRemove);
109                       });
110
111                       console.log('Done removing view ' +
112                         viewName + '".');
113
114                       // call super
115                       Backbone.View.prototype.remove.call(
116                         this);
117                     }
118                   },
119                   createContext: function () {
120                     var moduleName = this.model.get('
121                       module')
122                     module = middguard.PackagedModules
123                       .findWhere({name: moduleName}),
124                     connections = JSON.parse(this.
125                       model.get('connections')),
126                     context = {};
127
128                     context.inputs = _.reduce(_.keys(
129                       connections), function (inputs,
130                         inputGroup) {
131                           var groupConnections = connections[

```

```

74         inputGroup].connections,
75         outputNode = middguard.Nodes.get
76             (connections[inputGroup].
77              output_node);
78
79         var columns = _.reduce(
80             groupConnections, function (
81                 connections, pair) {
82                     connections[pair.input] = pair.
83                         output;
84                     return connections;
85                 }, {});
86
87         inputs[inputGroup] = {};
88         inputs[inputGroup].collection =
89             middguard.entities[outputNode.get
90                 ('table')];
91         inputs[inputGroup].cols = columns;
92         inputs[inputGroup].tableName =
93             outputNode.get('table');
94
95         return inputs;
96     }, {});
97
98     return context;
99 }
100 }
101
102 middguard.activateView = function (node) {
103     if (middguard.__modules[node] &&
104         middguard.__modules[node].live) {
105         middguard.deactivateView(node);
106     }
107     middguard.__modules[node].live = null;
108
109     middguard.toggleView = function (node) {
110         if (middguard.__modules[node] &&
111             middguard.__modules[node].live) {
112             middguard.deactivateView(node);
113         } else {
114             middguard.activateView(node);
115         }
116     };
117
118     // Internal hash of module views
119     middguard.__modules = {};
120
121     // Internal hash of submodule views
122     middguard.__submodules = {};
123
124     /* middguard.addModule
125      * Makes MiddGuard aware of a top level
126      * view.
127      * Top level views are listed under "
128      * Modules" in the sidebar.
129      */
130     middguard.addModule = function (name, view
131         ) {

```

```

128 _addView(name, view, true /* top level
129 */);
130 };
131 /* middguard.addSubview
132 * Makes MiddGuard aware of a subview (a
133   view instantiated from another view)
134 * Subviews are not listed in the sidebar,
135   but have models they fetch tracked
136 * and removed when the view is removed.
137 */
138 middguard.addSubview = function (name,
139   view) {
140   _addView(name, view, false /* not top
141     level */);
142 };
143 var _addView = function (name, view,
144   topLevel) {
145   if (!Object.prototype.hasOwnProperty.
146     call(middguard.__modules, name)) {
147     view.prototype.middguard_view_name =
148       name;
149     view.prototype.middguard_entities =
150       [];
151     if (topLevel) {
152       middguard.__modules[name] = {ctor:
153         view, live: null};
154     } else {
155       middguard.__submodules[name] = {ctor
156         : view, live: null};
157     }
158   } else {
159     throw new Error('Module ' + name + '
160       already loaded');
161   }
162 }
163
164 function removeFromArray(arr) {
165   var what, a = arguments, L = a.length,
166   ax;
167   while (L > 1 && arr.length) {
168     what = a[--L];
169     while ((ax= arr.indexOf(what)) !== -1)
170     {
171       arr.splice(ax, 1);
172     }
173   }
174   return arr;
175 }
176
177 /* Remove elements from an array.
178 * arr is the array to remove from (param
179   0).
180 * Elements to remove are arguments 1 .. n
181   .
182 * Source: http://stackoverflow.com/
183   questions/3954438
184 */
185 function removeFromArray(arr) {
186   var what, a = arguments, L = a.length,
187   ax;
188   while (L > 1 && arr.length) {
189     what = a[--L];
190     while ((ax= arr.indexOf(what)) !== -1)
191     {
192       arr.splice(ax, 1);
193     }
194   }
195   return arr;
196 }
197
198 /* Remove elements from an array.
199 * arr is the array to remove from (param
200   0).
201 * Elements to remove are arguments 1 .. n
202   .
203 * Source: http://stackoverflow.com/
204   questions/3954438
205 */
206 function removeFromArray(arr) {
207   var what, a = arguments, L = a.length,
208   ax;
209   while (L > 1 && arr.length) {
210     what = a[--L];
211     while ((ax= arr.indexOf(what)) !== -1)
212     {
213       arr.splice(ax, 1);
214     }
215   }
216   return arr;
217 }
218
219 /* Remove elements from an array.
220 * arr is the array to remove from (param
221   0).
222 * Elements to remove are arguments 1 .. n
223   .
224 * Source: http://stackoverflow.com/
225   questions/3954438
226 */
227 function removeFromArray(arr) {
228   var what, a = arguments, L = a.length,
229   ax;
230   while (L > 1 && arr.length) {
231     what = a[--L];
232     while ((ax= arr.indexOf(what)) !== -1)
233     {
234       arr.splice(ax, 1);
235     }
236   }
237   return arr;
238 }
239
240 /* Remove elements from an array.
241 * arr is the array to remove from (param
242   0).
243 * Elements to remove are arguments 1 .. n
244   .
245 * Source: http://stackoverflow.com/
246   questions/3954438
247 */
248 function removeFromArray(arr) {
249   var what, a = arguments, L = a.length,
250   ax;
251   while (L > 1 && arr.length) {
252     what = a[--L];
253     while ((ax= arr.indexOf(what)) !== -1)
254     {
255       arr.splice(ax, 1);
256     }
257   }
258   return arr;
259 }
260
261 /* Remove elements from an array.
262 * arr is the array to remove from (param
263   0).
264 * Elements to remove are arguments 1 .. n
265   .
266 * Source: http://stackoverflow.com/
267   questions/3954438
268 */
269 function removeFromArray(arr) {
270   var what, a = arguments, L = a.length,
271   ax;
272   while (L > 1 && arr.length) {
273     what = a[--L];
274     while ((ax= arr.indexOf(what)) !== -1)
275     {
276       arr.splice(ax, 1);
277     }
278   }
279   return arr;
280 }
281
282 /* Remove elements from an array.
283 * arr is the array to remove from (param
284   0).
285 * Elements to remove are arguments 1 .. n
286   .
287 * Source: http://stackoverflow.com/
288   questions/3954438
289 */
290 function removeFromArray(arr) {
291   var what, a = arguments, L = a.length,
292   ax;
293   while (L > 1 && arr.length) {
294     what = a[--L];
295     while ((ax= arr.indexOf(what)) !== -1)
296     {
297       arr.splice(ax, 1);
298     }
299   }
300   return arr;
301 }
302
303 /* Remove elements from an array.
304 * arr is the array to remove from (param
305   0).
306 * Elements to remove are arguments 1 .. n
307   .
308 * Source: http://stackoverflow.com/
309   questions/3954438
310 */
311 function removeFromArray(arr) {
312   var what, a = arguments, L = a.length,
313   ax;
314   while (L > 1 && arr.length) {
315     what = a[--L];
316     while ((ax= arr.indexOf(what)) !== -1)
317     {
318       arr.splice(ax, 1);
319     }
320   }
321   return arr;
322 }
323
324 /* Remove elements from an array.
325 * arr is the array to remove from (param
326   0).
327 * Elements to remove are arguments 1 .. n
328   .
329 * Source: http://stackoverflow.com/
330   questions/3954438
331 */
332 function removeFromArray(arr) {
333   var what, a = arguments, L = a.length,
334   ax;
335   while (L > 1 && arr.length) {
336     what = a[--L];
337     while ((ax= arr.indexOf(what)) !== -1)
338     {
339       arr.splice(ax, 1);
340     }
341   }
342   return arr;
343 }
344
345 /* Remove elements from an array.
346 * arr is the array to remove from (param
347   0).
348 * Elements to remove are arguments 1 .. n
349   .
350 * Source: http://stackoverflow.com/
351   questions/3954438
352 */
353 function removeFromArray(arr) {
354   var what, a = arguments, L = a.length,
355   ax;
356   while (L > 1 && arr.length) {
357     what = a[--L];
358     while ((ax= arr.indexOf(what)) !== -1)
359     {
360       arr.splice(ax, 1);
361     }
362   }
363   return arr;
364 }
365
366 /* Remove elements from an array.
367 * arr is the array to remove from (param
368   0).
369 * Elements to remove are arguments 1 .. n
370   .
371 * Source: http://stackoverflow.com/
372   questions/3954438
373 */
374 function removeFromArray(arr) {
375   var what, a = arguments, L = a.length,
376   ax;
377   while (L > 1 && arr.length) {
378     what = a[--L];
379     while ((ax= arr.indexOf(what)) !== -1)
380     {
381       arr.splice(ax, 1);
382     }
383   }
384   return arr;
385 }
386
387 /* Remove elements from an array.
388 * arr is the array to remove from (param
389   0).
390 * Elements to remove are arguments 1 .. n
391   .
392 * Source: http://stackoverflow.com/
393   questions/3954438
394 */
395 function removeFromArray(arr) {
396   var what, a = arguments, L = a.length,
397   ax;
398   while (L > 1 && arr.length) {
399     what = a[--L];
400     while ((ax= arr.indexOf(what)) !== -1)
401     {
402       arr.splice(ax, 1);
403     }
404   }
405   return arr;
406 }
407
408 /* Remove elements from an array.
409 * arr is the array to remove from (param
410   0).
411 * Elements to remove are arguments 1 .. n
412   .
413 * Source: http://stackoverflow.com/
414   questions/3954438
415 */
416 function removeFromArray(arr) {
417   var what, a = arguments, L = a.length,
418   ax;
419   while (L > 1 && arr.length) {
420     what = a[--L];
421     while ((ax= arr.indexOf(what)) !== -1)
422     {
423       arr.splice(ax, 1);
424     }
425   }
426   return arr;
427 }
428
429 /* Remove elements from an array.
430 * arr is the array to remove from (param
431   0).
432 * Elements to remove are arguments 1 .. n
433   .
434 * Source: http://stackoverflow.com/
435   questions/3954438
436 */
437 function removeFromArray(arr) {
438   var what, a = arguments, L = a.length,
439   ax;
440   while (L > 1 && arr.length) {
441     what = a[--L];
442     while ((ax= arr.indexOf(what)) !== -1)
443     {
444       arr.splice(ax, 1);
445     }
446   }
447   return arr;
448 }
449
450 /* Remove elements from an array.
451 * arr is the array to remove from (param
452   0).
453 * Elements to remove are arguments 1 .. n
454   .
455 * Source: http://stackoverflow.com/
456   questions/3954438
457 */
458 function removeFromArray(arr) {
459   var what, a = arguments, L = a.length,
460   ax;
461   while (L > 1 && arr.length) {
462     what = a[--L];
463     while ((ax= arr.indexOf(what)) !== -1)
464     {
465       arr.splice(ax, 1);
466     }
467   }
468   return arr;
469 }
470
471 /* Remove elements from an array.
472 * arr is the array to remove from (param
473   0).
474 * Elements to remove are arguments 1 .. n
475   .
476 * Source: http://stackoverflow.com/
477   questions/3954438
478 */
479 function removeFromArray(arr) {
480   var what, a = arguments, L = a.length,
481   ax;
482   while (L > 1 && arr.length) {
483     what = a[--L];
484     while ((ax= arr.indexOf(what)) !== -1)
485     {
486       arr.splice(ax, 1);
487     }
488   }
489   return arr;
490 }
491
492 /* Remove elements from an array.
493 * arr is the array to remove from (param
494   0).
495 * Elements to remove are arguments 1 .. n
496   .
497 * Source: http://stackoverflow.com/
498   questions/3954438
499 */
500 function removeFromArray(arr) {
501   var what, a = arguments, L = a.length,
502   ax;
503   while (L > 1 && arr.length) {
504     what = a[--L];
505     while ((ax= arr.indexOf(what)) !== -1)
506     {
507       arr.splice(ax, 1);
508     }
509   }
510   return arr;
511 }
512
513 /* Remove elements from an array.
514 * arr is the array to remove from (param
515   0).
516 * Elements to remove are arguments 1 .. n
517   .
518 * Source: http://stackoverflow.com/
519   questions/3954438
520 */
521 function removeFromArray(arr) {
522   var what, a = arguments, L = a.length,
523   ax;
524   while (L > 1 && arr.length) {
525     what = a[--L];
526     while ((ax= arr.indexOf(what)) !== -1)
527     {
528       arr.splice(ax, 1);
529     }
530   }
531   return arr;
532 }
533
534 /* Remove elements from an array.
535 * arr is the array to remove from (param
536   0).
537 * Elements to remove are arguments 1 .. n
538   .
539 * Source: http://stackoverflow.com/
540   questions/3954438
541 */
542 function removeFromArray(arr) {
543   var what, a = arguments, L = a.length,
544   ax;
545   while (L > 1 && arr.length) {
546     what = a[--L];
547     while ((ax= arr.indexOf(what)) !== -1)
548     {
549       arr.splice(ax, 1);
550     }
551   }
552   return arr;
553 }
554
555 /* Remove elements from an array.
556 * arr is the array to remove from (param
557   0).
558 * Elements to remove are arguments 1 .. n
559   .
560 * Source: http://stackoverflow.com/
561   questions/3954438
562 */
563 function removeFromArray(arr) {
564   var what, a = arguments, L = a.length,
565   ax;
566   while (L > 1 && arr.length) {
567     what = a[--L];
568     while ((ax= arr.indexOf(what)) !== -1)
569     {
570       arr.splice(ax, 1);
571     }
572   }
573   return arr;
574 }
575
576 /* Remove elements from an array.
577 * arr is the array to remove from (param
578   0).
579 * Elements to remove are arguments 1 .. n
580   .
581 * Source: http://stackoverflow.com/
582   questions/3954438
583 */
584 function removeFromArray(arr) {
585   var what, a = arguments, L = a.length,
586   ax;
587   while (L > 1 && arr.length) {
588     what = a[--L];
589     while ((ax= arr.indexOf(what)) !== -1)
590     {
591       arr.splice(ax, 1);
592     }
593   }
594   return arr;
595 }
596
597 /* Remove elements from an array.
598 * arr is the array to remove from (param
599   0).
600 * Elements to remove are arguments 1 .. n
601   .
602 * Source: http://stackoverflow.com/
603   questions/3954438
604 */
605 function removeFromArray(arr) {
606   var what, a = arguments, L = a.length,
607   ax;
608   while (L > 1 && arr.length) {
609     what = a[--L];
610     while ((ax= arr.indexOf(what)) !== -1)
611     {
612       arr.splice(ax, 1);
613     }
614   }
615   return arr;
616 }
617
618 /* Remove elements from an array.
619 * arr is the array to remove from (param
620   0).
621 * Elements to remove are arguments 1 .. n
622   .
623 * Source: http://stackoverflow.com/
624   questions/3954438
625 */
626 function removeFromArray(arr) {
627   var what, a = arguments, L = a.length,
628   ax;
629   while (L > 1 && arr.length) {
630     what = a[--L];
631     while ((ax= arr.indexOf(what)) !== -1)
632     {
633       arr.splice(ax, 1);
634     }
635   }
636   return arr;
637 }
638
639 /* Remove elements from an array.
640 * arr is the array to remove from (param
641   0).
642 * Elements to remove are arguments 1 .. n
643   .
644 * Source: http://stackoverflow.com/
645   questions/3954438
646 */
647 function removeFromArray(arr) {
648   var what, a = arguments, L = a.length,
649   ax;
650   while (L > 1 && arr.length) {
651     what = a[--L];
652     while ((ax= arr.indexOf(what)) !== -1)
653     {
654       arr.splice(ax, 1);
655     }
656   }
657   return arr;
658 }
659
660 /* Remove elements from an array.
661 * arr is the array to remove from (param
662   0).
663 * Elements to remove are arguments 1 .. n
664   .
665 * Source: http://stackoverflow.com/
666   questions/3954438
667 */
668 function removeFromArray(arr) {
669   var what, a = arguments, L = a.length,
670   ax;
671   while (L > 1 && arr.length) {
672     what = a[--L];
673     while ((ax= arr.indexOf(what)) !== -1)
674     {
675       arr.splice(ax, 1);
676     }
677   }
678   return arr;
679 }
680
681 /* Remove elements from an array.
682 * arr is the array to remove from (param
683   0).
684 * Elements to remove are arguments 1 .. n
685   .
686 * Source: http://stackoverflow.com/
687   questions/3954438
688 */
689 function removeFromArray(arr) {
690   var what, a = arguments, L = a.length,
691   ax;
692   while (L > 1 && arr.length) {
693     what = a[--L];
694     while ((ax= arr.indexOf(what)) !== -1)
695     {
696       arr.splice(ax, 1);
697     }
698   }
699   return arr;
700 }
701
702 /* Remove elements from an array.
703 * arr is the array to remove from (param
704   0).
705 * Elements to remove are arguments 1 .. n
706   .
707 * Source: http://stackoverflow.com/
708   questions/3954438
709 */
710 function removeFromArray(arr) {
711   var what, a = arguments, L = a.length,
712   ax;
713   while (L > 1 && arr.length) {
714     what = a[--L];
715     while ((ax= arr.indexOf(what)) !== -1)
716     {
717       arr.splice(ax, 1);
718     }
719   }
720   return arr;
721 }
722
723 /* Remove elements from an array.
724 * arr is the array to remove from (param
725   0).
726 * Elements to remove are arguments 1 .. n
727   .
728 * Source: http://stackoverflow.com/
729   questions/3954438
730 */
731 function removeFromArray(arr) {
732   var what, a = arguments, L = a.length,
733   ax;
734   while (L > 1 && arr.length) {
735     what = a[--L];
736     while ((ax= arr.indexOf(what)) !== -1)
737     {
738       arr.splice(ax, 1);
739     }
740   }
741   return arr;
742 }
743
744 /* Remove elements from an array.
745 * arr is the array to remove from (param
746   0).
747 * Elements to remove are arguments 1 .. n
748   .
749 * Source: http://stackoverflow.com/
750   questions/3954438
751 */
752 function removeFromArray(arr) {
753   var what, a = arguments, L = a.length,
754   ax;
755   while (L > 1 && arr.length) {
756     what = a[--L];
757     while ((ax= arr.indexOf(what)) !== -1)
758     {
759       arr.splice(ax, 1);
760     }
761   }
762   return arr;
763 }
764
765 /* Remove elements from an array.
766 * arr is the array to remove from (param
767   0).
768 * Elements to remove are arguments 1 .. n
769   .
770 * Source: http://stackoverflow.com/
771   questions/3954438
772 */
773 function removeFromArray(arr) {
774   var what, a = arguments, L = a.length,
775   ax;
776   while (L > 1 && arr.length) {
777     what = a[--L];
778     while ((ax= arr.indexOf(what)) !== -1)
779     {
780       arr.splice(ax, 1);
781     }
782   }
783   return arr;
784 }
785
786 /* Remove elements from an array.
787 * arr is the array to remove from (param
788   0).
789 * Elements to remove are arguments 1 .. n
790   .
791 * Source: http://stackoverflow.com/
792   questions/3954438
793 */
794 function removeFromArray(arr) {
795   var what, a = arguments, L = a.length,
796   ax;
797   while (L > 1 && arr.length) {
798     what = a[--L];
799     while ((ax= arr.indexOf(what)) !== -1)
800     {
801       arr.splice(ax, 1);
802     }
803   }
804   return arr;
805 }
806
807 /* Remove elements from an array.
808 * arr is the array to remove from (param
809   0).
810 * Elements to remove are arguments 1 .. n
811   .
812 * Source: http://stackoverflow.com/
813   questions/3954438
814 */
815 function removeFromArray(arr) {
816   var what, a = arguments, L = a.length,
817   ax;
818   while (L > 1 && arr.length) {
819     what = a[--L];
820     while ((ax= arr.indexOf(what)) !== -1)
821     {
822       arr.splice(ax, 1);
823     }
824   }
825   return arr;
826 }
827
828 /* Remove elements from an array.
829 * arr is the array to remove from (param
830   0).
831 * Elements to remove are arguments 1 .. n
832   .
833 * Source: http://stackoverflow.com/
834   questions/3954438
835 */
836 function removeFromArray(arr) {
837   var what, a = arguments, L = a.length,
838   ax;
839   while (L > 1 && arr.length) {
840     what = a[--L];
841     while ((ax= arr.indexOf(what)) !== -1)
842     {
843       arr.splice(ax, 1);
844     }
845   }
846   return arr;
847 }
848
849 /* Remove elements from an array.
850 * arr is the array to remove from (param
851   0).
852 * Elements to remove are arguments 1 .. n
853   .
854 * Source: http://stackoverflow.com/
855   questions/3954438
856 */
857 function removeFromArray(arr) {
858   var what, a = arguments, L = a.length,
859   ax;
860   while (L > 1 && arr.length) {
861     what = a[--L];
862     while ((ax= arr.indexOf(what)) !== -1)
863     {
864       arr.splice(ax, 1);
865     }
866   }
867   return arr;
868 }
869
870 /* Remove elements from an array.
871 * arr is the array to remove from (param
872   0).
873 * Elements to remove are arguments 1 .. n
874   .
875 * Source: http://stackoverflow.com/
876   questions/3954438
877 */
878 function removeFromArray(arr) {
879   var what, a = arguments, L = a.length,
880   ax;
881   while (L > 1 && arr.length) {
882     what = a[--L];
883     while ((ax= arr.indexOf(what)) !== -1)
884     {
885       arr.splice(ax, 1);
886     }
887   }
888   return arr;
889 }
890
891 /* Remove elements from an array.
892 * arr is the array to remove from (param
893   0).
894 * Elements to remove are arguments 1 .. n
895   .
896 * Source: http://stackoverflow.com/
897   questions/3954438
898 */
899 function removeFromArray(arr) {
900   var what, a = arguments, L = a.length,
901   ax;
902   while (L > 1 && arr.length) {
903     what = a[--L];
904     while ((ax= arr.indexOf(what)) !== -1)
905     {
906       arr.splice(ax, 1);
907     }
908   }
909   return arr;
910 }
911
912 /* Remove elements from an array.
913 * arr is the array to remove from (param
914   0).
915 * Elements to remove are arguments 1 .. n
916   .
917 * Source: http://stackoverflow.com/
918   questions/3954438
919 */
920 function removeFromArray(arr) {
921   var what, a = arguments, L = a.length,
922   ax;
923   while (L > 1 && arr.length) {
924     what = a[--L];
925     while ((ax= arr.indexOf(what)) !== -1)
926     {
927       arr.splice(ax, 1);
928     }
929   }
930   return arr;
931 }
932
933 /* Remove elements from an array.
934 * arr is the array to remove from (param
935   0).
936 * Elements to remove are arguments 1 .. n
937   .
938 * Source: http://stackoverflow.com/
939   questions/3954438
940 */
941 function removeFromArray(arr) {
942   var what, a = arguments, L = a.length,
943   ax;
944   while (L > 1 && arr.length) {
945     what = a[--L];
946     while ((ax= arr.indexOf(what)) !== -1)
947     {
948       arr.splice(ax, 1);
949     }
950   }
951   return arr;
952 }
953
954 /* Remove elements from an array.
955 * arr is the array to remove from (param
956   0).
957 * Elements to remove are arguments 1 .. n
958   .
959 * Source: http://stackoverflow.com/
960   questions/3954438
961 */
962 function removeFromArray(arr) {
963   var what, a = arguments, L = a.length,
964   ax;
965   while (L > 1 && arr.length) {
966     what = a[--L];
967     while ((ax= arr.indexOf(what)) !== -1)
968     {
969       arr.splice(ax, 1);
970     }
971   }
972   return arr;
973 }
974
975 /* Remove elements from an array.
976 * arr is the array to remove from (param
977   0).
978 * Elements to remove are arguments 1 .. n
979   .
980 * Source: http://stackoverflow.com/
981   questions/3954438
982 */
983 function removeFromArray(arr) {
984   var what, a = arguments, L = a.length,
985   ax;
986   while (L > 1 && arr.length) {
987     what = a[--L];
988     while ((ax= arr.indexOf(what)) !== -1)
989     {
990       arr.splice(ax, 1);
991     }
992   }
993   return arr;
994 }
995
996 /* Remove elements from an array.
997 * arr is the array to remove from (param
998   0).
999 * Elements to remove are arguments 1 .. n
1000   .
1001 * Source: http://stackoverflow.com/
1002   questions/3954438
1003 */
1004 function removeFromArray(arr) {
1005   var what, a = arguments, L = a.length,
1006   ax;
1007   while (L > 1 && arr.length) {
1008     what = a[--L];
1009     while ((ax= arr.indexOf(what)) !== -1)
1010     {
1011       arr.splice(ax, 1);
1012     }
1013   }
1014   return arr;
1015 }
1016
1017 /* Remove elements from an array.
1018 * arr is the array to remove from (param
1019   0).
1020 * Elements to remove are arguments 1 .. n
1021   .
1022 * Source: http://stackoverflow.com/
1023   questions/3954438
1024 */
1025 function removeFromArray(arr) {
1026   var what, a = arguments, L = a.length,
1027   ax;
1028   while (L > 1 && arr.length) {
1029     what = a[--L];
1030     while ((ax= arr.indexOf(what))
```

static/js/collections/graphs.js

```

1 var middguard = middguard || {};
2
3 function() {
4   'use strict';
5
6   var Graphs = middguard.BaseCollection.
      extend({
7     model: middguard.Graph,
8     url: 'graphs'
9   });
10
11   middguard.Graphs = new Graphs();
12 })();

```

static/js/collections/nodes.js

```

1 var middguard = middguard || {};
2
3 function() {
4   'use strict';
5
6   var Nodes = middguard.BaseCollection.
      extend({
7     model: middguard.Node,
8
9     url: 'nodes',
10
11     initialize: function() {
12       _.bindAll(this, 'serverCreate', '
          serverUpdate');
13
14       this.ioBind('create', this.
          serverCreate, this);
15       this.ioBind('update', this.
          serverUpdate, this);
16     },
17
18     serverCreate: function(data) {
19       var exists = this.get(data.id);
20       if (!exists) {
21         this.add(data);
22       } else {
23         exists.set(data);
24       }
25     },
26
27     serverUpdate: function(data) {
28       var exists = this.get(data.id);
29       if (exists) {
30         exists.set(data);
31       }

```

```

32     },
33   });
34
35   middguard.Nodes = new Nodes();
36 })();

static/js/collections/packaged-modules.js
1  var middguard = middguard || {};
2
3  (function () {
4    var PackagedModules = Backbone.Collection.
      extend({
5      url: 'modules',
6      model: middguard.PackagedModule
7    });
8
9    middguard.PackagedModules = new
      PackagedModules();
10 })();

```

static/js/models/graph.js

```
1 var middguard = middguard || {};  
2  
3 (function() {  
4   middguard.Graph = Backbone.Model.extend();  
5 })();
```

static/js/models/node.js

```
1 var middguard = middguard || {};  
2  
3 (function() {  
4   middguard.Node = Backbone.Model.extend({  
5     blacklistAttributes: [  
6       'selectedInput',  
7       'selectedOutput'  
8     ],  
9  
10    defaults: {  
11      status: 0,  
12      radius: 75,  
13      position_x: 0,  
14      position_y: 0,  
15      selectedInput: null,  
16      selectedOutput: null,  
17      connections: '{}'  
18    },  
19  
20    statusMap: {  
21      0: 'Not run',  
22      1: 'In progress',  
23      2: 'Completed'  
24    },  
25  
26    connectToOutput: function(other,  
27      inputGroup) {  
28      middguard.socket.emit('node:connect',  
29        {  
30          outputNode: other.get('id'),  
31          inputNode: this.get('id'),  
32          inputGroup: inputGroup  
33        });  
34    },  
35  });  
36 }());
```

```

34 run: function () {
35   middleware.socket.emit('node:run', {
36     id: this.getId()
37   });
38 },
39
40 position: function(x, y) {
41   if (!arguments.length) {
42     return {x: this.getPosition_x(), y
43       : this.getPosition_y()};
44   } else {
45     this.set('position_x', x);
46     this.set('position_y', y);
47   }
48 },
49
50 toJSON: function(options) {
51   return _.omit(this.attributes, this.
52     blacklistAttributes);
53 },
54
55 statusText: function () {
56   return this.statusMap[this.get('status
57     ')];
58 },
59
60 module: function () {
61   return middleware.PackagedModules.
62     findWhere({
63       name: this.get('module')
64     });
65 },
66
67 unconnectedInputs: function(inputGroup)
68 {
69   var connections = JSON.parse(this.get(
70     'connections'))[inputGroup];
71
72   if (!connections.output_node) {
73     return [];
74   }
75
76   var connectedOutputs = function(inputGroup)
77   {
78     var connections = JSON.parse(this.get(
79     'connections'))[inputGroup];
80
81     if (!connections.output_node) {
82       return [];
83     }
84
85     var connectedOutputs = connections.
86       connections.map(c => c.output);
87
88     var outputNode = middleware.Nodes.get(
89     connections.output_node);
90
91     var allOutputs = middleware.
92       PackagedModules
93       .find({name: outputNode.get('
94         module')})
95       .get('outputs');
96   }
97 }
98
99 allInputs = _.find(this.module().
100   get('inputs'),
101   {name:
102     inputGroup}.
103   inputs);
104 }
105 }

```

```

89     return _.difference(allOutputs,
90         connectedOutputs);
91     },
92     isVisualization: function() {
93         return this.module().get('
94             visualization');
95     }
96 })();

static/js/models/packaged-module.js
1  var middguard = middguard || {};
2
3  (function () {
4      middguard.PackagedModule = Backbone.Model.
5          extend({
6              defaults: {
7                  'name': '',
7                  'main': '',
8                  visualization: false
9              }
10     });
11 })();

```



```

static/js/views/graphs-view.js
1  var middguard = middguard || {};
2
3  (function() {
4    'use strict';
5
6    middguard.GraphsView = Backbone.View.
      extend({
7      className: 'middguard-graphs',
8
9      template: _.template($('#graphs-panel-
        template').html()),
10
11     events: {
12       'click #create-new-graph': '
         createGraph'
13     },
14
15     initialize: function() {
16       this.listenTo(middguard.Graphs, 'add',
         this.addOneGraph);
17       this.listenTo(middguard.Graphs, 'reset',
         this.addAllGraphs);
18
19       middguard.Graphs.fetch({reset: true,
        data: {}});
20     },
21
22     render: function() {
23       this.$el.html(this.template());
24
25       this.$graphs = this.$('.graphs-list');
26
27       return this;
28     },
29
30     addOneGraph: function(graph) {
31       var graphView = new GraphView({model:
        graph});
32
33       this.$graphs.append(graphView.render()
        .el);
34     },
35
36     addAllGraphs: function() {
37       middguard.Graphs.each(this.addOneGraph,
        this);
38     },
39
40     createGraph: function(e) {
41       e.preventDefault();
42       var name = this.$('#new-graph-name').
        val().trim();
43
44       middguard.Graphs.create({name: name},
        {wait: true});
45       this.$('#new-graph-name').val('');
46     }
47   });
48
49   var GraphView = Backbone.View.extend({
50     className: 'middguard-graph list-group-
        item',
51
52     tagName: 'a',
53
54     template: _.template('<%= name %>'),
55
56     events: {
57       'click': 'toggleEditor'
58     },
59

```

```

60 initialize: function() {
61   this.editing = false;
62
63   this.listenTo(this.model, 'update',
64     this.render);
65 },
66 render: function() {
67   this.$el.html(this.template(this.model
68     .toJSON()));
69   this.$el.attr('href', '#');
70   return this;
71 },
72
73 toggleEditor: function() {
74   if (this.editor) {
75     this.editor.remove();
76     this.editor = null;
77   } else {
78     this.editor = new middguard.
79       GraphEditorView({graph: this.
80         model});
81     $('<div class="middguard-views">').append(this.
82       editor.render().el);
83   }
84 });
85
86 this.$el.toggleClass('active', Boolean
87   (this.editor));
88
89 middguard.PackagedModules.fetch({reset
90   : true, data: {}});

```

static/js/views/graph-editor-view.js

```

1 var middguard = middguard || {};
2
3 (function() {
4   'use strict';
5
6   middguard.GraphEditorView = Backbone.View.
7     extend({
8       className: 'middguard-graph-editor
9         middguard-module',
10       tagName: 'div',
11       template: _.template($('#graph-editor-
12         template').html()),
13       initialize: function(options) {
14         this.graph = options.graph;
15         this.detailView = null;
16
17         this.listenTo(middguard.
18           PackagedModules, 'reset', this.
19             addModules);
20         this.listenTo(middguard.Nodes, 'reset'
21           , this.addAllNodes);
22         this.listenTo(middguard.Nodes, 'reset'
23           , this.addAllConnectorGroups);
24         this.listenTo(middguard.Nodes, 'reset'
25           , this.ensureEntityCollections);
26         this.listenTo(middguard.Nodes, 'add',
27           this.addNode);
28         this.listenTo(middguard.Nodes, 'add',
29           this.addConnectorGroup);
30
31         middguard.PackagedModules.fetch({reset
32           : true, data: {}});

```

```

25   middguard.Nodes.fetch({reset: true,
26     data: {}});
27
28   render: function() {
29     this.$el.html(this.template(this.graph
30       .toJSON()));
31     d3.select(this.el).select('.editor').
32       append('svg')
33       .attr('class', 'graph')
34       .attr('width', 500);
35
36     this.resizeEditor();
37
38     return this;
39   },
40   ensureEntityCollections: function() {
41     middguard.Nodes.each(this.
42       ensureEntityCollection, this);
43   },
44   ensureEntityCollection: function(node) {
45     var tableName = node.get('table');
46     if (!tableName || middguard.entities[
47       tableName]) return;
48     var collection = new middguard.
49       EntityCollection([], {
50         url: tableName
51       });
52     middguard.entities[tableName] =
53       collection;
54   },
55   resizeEditor: function() {
56     d3.select(this.el).select('.editor svg
57       ')
58       .attr('height', $(window).height()
59         - this.$('header').
60         outerHeight());
61   },
62   addModules: function() {
63     this.$('.modules-list').html('');
64     middguard.PackagedModules.each(
65       function(model) {
66         var view = new ModuleListItemView({
67           model: model, graph: this.graph})
68         ;
69         this.$('.modules-list').append(view.
70           render().el);
71       }.bind(this));
72     this.resizeEditor();
73   },
74   addNode: function(node) {
75     if (node.get('graph_id') !== this.
76       graph.get('id')) {
77       return;
78     }
79     var view = new NodeView({model: node,
80       editor: this});
81     this.$('.graph').append(view.render().
82       el);
83   },
84   },
85   },
86   },
87   },
88   },
89   },
90   },
91   },
92   },
93   },
94   },
95   },
96   },
97   },
98   },
99   },
100  },

```

```

80 addAllNodes: function(node) {
81     middleware.Nodes.each(this.addNode,
82         this);
83 }
84
85 addConnectorGroup: function(node) {
86     if (node.get('graph_id') !== this.
87         graph.get('id')) {
88         return;
89     }
90     var view = new ConnectorGroupView({
91         model: node});
92     this.$('graph').append(view.render().
93         el);
94 }
95
96 addAllConnectorGroups: function() {
97     middleware.Nodes.each(this.
98         addConnectorGroup, this);
99 }
100
101 setDetailView: function(view) {
102     if (this.detailView) {
103         this.detailView.remove();
104     }
105     this.$('detail').html(view.render().
106         el);
107     this.detailView = view;
108 }
109
110 }
111
112 className: 'btn btn-default module',
113
114 template: _.template('<%= displayName %>
115     '),
116
117 events: {
118     'click': 'createNode',
119 },
120
121 initialize: function(options) {
122     this.model = options.model;
123     this.graph = options.graph;
124 }
125
126 render: function() {
127     this.$el.html(this.template(this.model
128         .toJSON()));
129     return this;
130 }
131
132 createNode: function() {
133     middleware.Nodes.create({
134         module: this.model.get('name'),
135         graph_id: this.graph.get('id')
136     });
137 }
138
139 /* Nodes' connections are stored on the
140    input node.
141    * All the connecting lines from an a node
142    's connections
143    * to the corresponding output node.
144    */
145 var ConnectorGroupView = Backbone.NSView.

```

```

141         extend({
142             tagName: 'svg:g',
143             initialize: function() {
144                 this.connections = [];
145             }
146         });
147         if (this.model.get('connections')) {
148             this.addAllConnectingLines();
149         }
150         // 'this.model' is the "input" node
151         this.listenTo(this.model, 'change',
152             this.render);
153     },
154     render: function() {
155         this.connections.forEach(connection =>
156             connection.render());
157         this.unrenderedConnections().forEach(
158             this.addConnectingLine, this);
159         return this;
160     },
161     addAllConnectingLines: function() {
162         _.chain(JSON.parse(this.model.get('
163             connections'))).
164             .keys()
165             .each(this.addConnectingLine, this
166             );
167     },
168     addConnectingLine: function(inputGroup)
169     {
170         var view = new ConnectorView({
171             model: this.model,
172             inputGroup: inputGroup
173         });
174         this.$el.append(view.render().el);
175         this.connections.push(view);
176     },
177     renderedConnections: function() {
178         return this.connections.map(connection
179             => connection.inputGroup);
180     },
181     unrenderedConnections: function() {
182         return _.chain(JSON.parse(this.model.
183             get('connections')))
184             .keys()
185             .difference(this.
186                 renderedConnections());
187     },
188     var ConnectorView = Backbone.NSView.extend
189     ({
190         tagName: 'svg:path',
191         className: 'connecting-line',
192         initialize: function(options) {
193             this.model = options.model;
194             this.inputGroup = options.inputGroup;
195             this.outputNode = middguard.Nodes.
196                 findWhere({
197                     id: JSON.parse(this.model.get('
198                         connections'))[this.inputGroup].
199                     output_node
200                 });
201             this.module = middguard.
202                 PackagedModules.findWhere({

```

```

198     name: this.model.get('module')
199   });
200
201   this.diagonal = d3.svg.diagonal();
202
203   // Only rerender this particular line
204   // when the output node moves.
205   this.listenTo(this.outputNode, 'change
206   ', this.render);
207
208   // Check if the connection has changed
209   // . In this context, "changed"
210   // means that the connection's input
211   // group either no longer has a
212   // connection, or the input group is
213   // connected to a different output.
214   this.listenTo(this.model, 'change',
215   this.connectionChanged);
216
217   },
218   render: function() {
219     this.diagonal
220     .source(this.outputPosition())
221     .target(this.inputPosition());
222
223     this.$el.attr('d', this.diagonal());
224
225     return this;
226   },
227
228   inputPosition: function() {
229     var i = _.findIndex(this.module.get('
230     inputs'), input => {
231       return input.name === this.
232
233     });
234
235     if (i < 0) {
236       // No longer a connection for this
237       // input group
238       if (!connection) {
239         this.remove();
240       }
241     }
242
243     return this.module.get('inputs')[i];
244   },
245
246   connectionChanged: function() {
247     var connections = this.model.get('
248     connections'),
249     connection = JSON.parse(
250     connections[this.inputGroup]);
251
252     // No longer a connection for this
253     // input group
254     if (!connection) {
255       this.remove();
256     }
257
258     return this.module.get('radius') +
259     offset;
260   },
261
262   outputPosition: function() {
263     var r = this.outputNode.get('radius');
264
265     return {
266       x: this.outputNode.position().x + r,
267       y: this.outputNode.position().y + 2
268       * r - 10
269     };
270   },
271
272   connectionChanged: function() {
273     var connections = this.model.get('
274     connections'),
275     connection = JSON.parse(
276     connections[this.inputGroup]);
277
278     // No longer a connection for this
279     // input group
280     if (!connection) {
281       this.remove();
282     }
283
284     return this.module.get('radius') +
285     offset;
286   },
287
288   outputPosition: function() {
289     var r = this.outputNode.get('radius');
290
291     return {
292       x: this.outputNode.position().x + r,
293       y: this.outputNode.position().y + 2
294       * r - 10
295     };
296   },
297
298   connectionChanged: function() {
299     var connections = this.model.get('
300     connections'),
301     connection = JSON.parse(
302     connections[this.inputGroup]);
303
304     // No longer a connection for this
305     // input group
306     if (!connection) {
307       this.remove();
308     }
309
310     return this.module.get('radius') +
311     offset;
312   },
313
314   outputPosition: function() {
315     var r = this.outputNode.get('radius');
316
317     return {
318       x: this.outputNode.position().x + r,
319       y: this.outputNode.position().y + 2
320       * r - 10
321     };
322   },
323
324   connectionChanged: function() {
325     var connections = this.model.get('
326     connections'),
327     connection = JSON.parse(
328     connections[this.inputGroup]);
329
330     // No longer a connection for this
331     // input group
332     if (!connection) {
333       this.remove();
334     }
335
336     return this.module.get('radius') +
337     offset;
338   },
339
340   outputPosition: function() {
341     var r = this.outputNode.get('radius');
342
343     return {
344       x: this.outputNode.position().x + r,
345       y: this.outputNode.position().y + 2
346       * r - 10
347     };
348   },
349
350   connectionChanged: function() {
351     var connections = this.model.get('
352     connections'),
353     connection = JSON.parse(
354     connections[this.inputGroup]);
355
356     // No longer a connection for this
357     // input group
358     if (!connection) {
359       this.remove();
360     }
361
362     return this.module.get('radius') +
363     offset;
364   },
365
366   outputPosition: function() {
367     var r = this.outputNode.get('radius');
368
369     return {
370       x: this.outputNode.position().x + r,
371       y: this.outputNode.position().y + 2
372       * r - 10
373     };
374   },
375
376   connectionChanged: function() {
377     var connections = this.model.get('
378     connections'),
379     connection = JSON.parse(
380     connections[this.inputGroup]);
381
382     // No longer a connection for this
383     // input group
384     if (!connection) {
385       this.remove();
386     }
387
388     return this.module.get('radius') +
389     offset;
390   },
391
392   outputPosition: function() {
393     var r = this.outputNode.get('radius');
394
395     return {
396       x: this.outputNode.position().x + r,
397       y: this.outputNode.position().y + 2
398       * r - 10
399     };
400   },
401
402   connectionChanged: function() {
403     var connections = this.model.get('
404     connections'),
405     connection = JSON.parse(
406     connections[this.inputGroup]);
407
408     // No longer a connection for this
409     // input group
410     if (!connection) {
411       this.remove();
412     }
413
414     return this.module.get('radius') +
415     offset;
416   },
417
418   outputPosition: function() {
419     var r = this.outputNode.get('radius');
420
421     return {
422       x: this.outputNode.position().x + r,
423       y: this.outputNode.position().y + 2
424       * r - 10
425     };
426   },
427
428   connectionChanged: function() {
429     var connections = this.model.get('
430     connections'),
431     connection = JSON.parse(
432     connections[this.inputGroup]);
433
434     // No longer a connection for this
435     // input group
436     if (!connection) {
437       this.remove();
438     }
439
440     return this.module.get('radius') +
441     offset;
442   },
443
444   outputPosition: function() {
445     var r = this.outputNode.get('radius');
446
447     return {
448       x: this.outputNode.position().x + r,
449       y: this.outputNode.position().y + 2
450       * r - 10
451     };
452   },
453
454   connectionChanged: function() {
455     var connections = this.model.get('
456     connections'),
457     connection = JSON.parse(
458     connections[this.inputGroup]);
459
460     // No longer a connection for this
461     // input group
462     if (!connection) {
463       this.remove();
464     }
465
466     return this.module.get('radius') +
467     offset;
468   },
469
470   outputPosition: function() {
471     var r = this.outputNode.get('radius');
472
473     return {
474       x: this.outputNode.position().x + r,
475       y: this.outputNode.position().y + 2
476       * r - 10
477     };
478   },
479
480   connectionChanged: function() {
481     var connections = this.model.get('
482     connections'),
483     connection = JSON.parse(
484     connections[this.inputGroup]);
485
486     // No longer a connection for this
487     // input group
488     if (!connection) {
489       this.remove();
490     }
491
492     return this.module.get('radius') +
493     offset;
494   },
495
496   outputPosition: function() {
497     var r = this.outputNode.get('radius');
498
499     return {
500       x: this.outputNode.position().x + r,
501       y: this.outputNode.position().y + 2
502       * r - 10
503     };
504   },
505
506   connectionChanged: function() {
507     var connections = this.model.get('
508     connections'),
509     connection = JSON.parse(
510     connections[this.inputGroup]);
511
512     // No longer a connection for this
513     // input group
514     if (!connection) {
515       this.remove();
516     }
517
518     return this.module.get('radius') +
519     offset;
520   },
521
522   outputPosition: function() {
523     var r = this.outputNode.get('radius');
524
525     return {
526       x: this.outputNode.position().x + r,
527       y: this.outputNode.position().y + 2
528       * r - 10
529     };
530   },
531
532   connectionChanged: function() {
533     var connections = this.model.get('
534     connections'),
535     connection = JSON.parse(
536     connections[this.inputGroup]);
537
538     // No longer a connection for this
539     // input group
540     if (!connection) {
541       this.remove();
542     }
543
544     return this.module.get('radius') +
545     offset;
546   },
547
548   outputPosition: function() {
549     var r = this.outputNode.get('radius');
550
551     return {
552       x: this.outputNode.position().x + r,
553       y: this.outputNode.position().y + 2
554       * r - 10
555     };
556   },
557
558   connectionChanged: function() {
559     var connections = this.model.get('
560     connections'),
561     connection = JSON.parse(
562     connections[this.inputGroup]);
563
564     // No longer a connection for this
565     // input group
566     if (!connection) {
567       this.remove();
568     }
569
570     return this.module.get('radius') +
571     offset;
572   },
573
574   outputPosition: function() {
575     var r = this.outputNode.get('radius');
576
577     return {
578       x: this.outputNode.position().x + r,
579       y: this.outputNode.position().y + 2
580       * r - 10
581     };
582   },
583
584   connectionChanged: function() {
585     var connections = this.model.get('
586     connections'),
587     connection = JSON.parse(
588     connections[this.inputGroup]);
589
590     // No longer a connection for this
591     // input group
592     if (!connection) {
593       this.remove();
594     }
595
596     return this.module.get('radius') +
597     offset;
598   },
599
600   outputPosition: function() {
601     var r = this.outputNode.get('radius');
602
603     return {
604       x: this.outputNode.position().x + r,
605       y: this.outputNode.position().y + 2
606       * r - 10
607     };
608   },
609
610   connectionChanged: function() {
611     var connections = this.model.get('
612     connections'),
613     connection = JSON.parse(
614     connections[this.inputGroup]);
615
616     // No longer a connection for this
617     // input group
618     if (!connection) {
619       this.remove();
620     }
621
622     return this.module.get('radius') +
623     offset;
624   },
625
626   outputPosition: function() {
627     var r = this.outputNode.get('radius');
628
629     return {
630       x: this.outputNode.position().x + r,
631       y: this.outputNode.position().y + 2
632       * r - 10
633     };
634   },
635
636   connectionChanged: function() {
637     var connections = this.model.get('
638     connections'),
639     connection = JSON.parse(
640     connections[this.inputGroup]);
641
642     // No longer a connection for this
643     // input group
644     if (!connection) {
645       this.remove();
646     }
647
648     return this.module.get('radius') +
649     offset;
650   },
651
652   outputPosition: function() {
653     var r = this.outputNode.get('radius');
654
655     return {
656       x: this.outputNode.position().x + r,
657       y: this.outputNode.position().y + 2
658       * r - 10
659     };
660   },
661
662   connectionChanged: function() {
663     var connections = this.model.get('
664     connections'),
665     connection = JSON.parse(
666     connections[this.inputGroup]);
667
668     // No longer a connection for this
669     // input group
670     if (!connection) {
671       this.remove();
672     }
673
674     return this.module.get('radius') +
675     offset;
676   },
677
678   outputPosition: function() {
679     var r = this.outputNode.get('radius');
680
681     return {
682       x: this.outputNode.position().x + r,
683       y: this.outputNode.position().y + 2
684       * r - 10
685     };
686   },
687
688   connectionChanged: function() {
689     var connections = this.model.get('
690     connections'),
691     connection = JSON.parse(
692     connections[this.inputGroup]);
693
694     // No longer a connection for this
695     // input group
696     if (!connection) {
697       this.remove();
698     }
699
700     return this.module.get('radius') +
701     offset;
702   },
703
704   outputPosition: function() {
705     var r = this.outputNode.get('radius');
706
707     return {
708       x: this.outputNode.position().x + r,
709       y: this.outputNode.position().y + 2
710       * r - 10
711     };
712   },
713
714   connectionChanged: function() {
715     var connections = this.model.get('
716     connections'),
717     connection = JSON.parse(
718     connections[this.inputGroup]);
719
720     // No longer a connection for this
721     // input group
722     if (!connection) {
723       this.remove();
724     }
725
726     return this.module.get('radius') +
727     offset;
728   },
729
730   outputPosition: function() {
731     var r = this.outputNode.get('radius');
732
733     return {
734       x: this.outputNode.position().x + r,
735       y: this.outputNode.position().y + 2
736       * r - 10
737     };
738   },
739
740   connectionChanged: function() {
741     var connections = this.model.get('
742     connections'),
743     connection = JSON.parse(
744     connections[this.inputGroup]);
745
746     // No longer a connection for this
747     // input group
748     if (!connection) {
749       this.remove();
750     }
751
752     return this.module.get('radius') +
753     offset;
754   },
755
756   outputPosition: function() {
757     var r = this.outputNode.get('radius');
758
759     return {
760       x: this.outputNode.position().x + r,
761       y: this.outputNode.position().y + 2
762       * r - 10
763     };
764   },
765
766   connectionChanged: function() {
767     var connections = this.model.get('
768     connections'),
769     connection = JSON.parse(
770     connections[this.inputGroup]);
771
772     // No longer a connection for this
773     // input group
774     if (!connection) {
775       this.remove();
776     }
777
778     return this.module.get('radius') +
779     offset;
780   },
781
782   outputPosition: function() {
783     var r = this.outputNode.get('radius');
784
785     return {
786       x: this.outputNode.position().x + r,
787       y: this.outputNode.position().y + 2
788       * r - 10
789     };
790   },
791
792   connectionChanged: function() {
793     var connections = this.model.get('
794     connections'),
795     connection = JSON.parse(
796     connections[this.inputGroup]);
797
798     // No longer a connection for this
799     // input group
800     if (!connection) {
801       this.remove();
802     }
803
804     return this.module.get('radius') +
805     offset;
806   },
807
808   outputPosition: function() {
809     var r = this.outputNode.get('radius');
810
811     return {
812       x: this.outputNode.position().x + r,
813       y: this.outputNode.position().y + 2
814       * r - 10
815     };
816   },
817
818   connectionChanged: function() {
819     var connections = this.model.get('
820     connections'),
821     connection = JSON.parse(
822     connections[this.inputGroup]);
823
824     // No longer a connection for this
825     // input group
826     if (!connection) {
827       this.remove();
828     }
829
830     return this.module.get('radius') +
831     offset;
832   },
833
834   outputPosition: function() {
835     var r = this.outputNode.get('radius');
836
837     return {
838       x: this.outputNode.position().x + r,
839       y: this.outputNode.position().y + 2
840       * r - 10
841     };
842   },
843
844   connectionChanged: function() {
845     var connections = this.model.get('
846     connections'),
847     connection = JSON.parse(
848     connections[this.inputGroup]);
849
850     // No longer a connection for this
851     // input group
852     if (!connection) {
853       this.remove();
854     }
855
856     return this.module.get('radius') +
857     offset;
858   },
859
860   outputPosition: function() {
861     var r = this.outputNode.get('radius');
862
863     return {
864       x: this.outputNode.position().x + r,
865       y: this.outputNode.position().y + 2
866       * r - 10
867     };
868   },
869
870   connectionChanged: function() {
871     var connections = this.model.get('
872     connections'),
873     connection = JSON.parse(
874     connections[this.inputGroup]);
875
876     // No longer a connection for this
877     // input group
878     if (!connection) {
879       this.remove();
880     }
881
882     return this.module.get('radius') +
883     offset;
884   },
885
886   outputPosition: function() {
887     var r = this.outputNode.get('radius');
888
889     return {
890       x: this.outputNode.position().x + r,
891       y: this.outputNode.position().y + 2
892       * r - 10
893     };
894   },
895
896   connectionChanged: function() {
897     var connections = this.model.get('
898     connections'),
899     connection = JSON.parse(
900     connections[this.inputGroup]);
901
902     // No longer a connection for this
903     // input group
904     if (!connection) {
905       this.remove();
906     }
907
908     return this.module.get('radius') +
909     offset;
910   },
911
912   outputPosition: function() {
913     var r = this.outputNode.get('radius');
914
915     return {
916       x: this.outputNode.position().x + r,
917       y: this.outputNode.position().y + 2
918       * r - 10
919     };
920   },
921
922   connectionChanged: function() {
923     var connections = this.model.get('
924     connections'),
925     connection = JSON.parse(
926     connections[this.inputGroup]);
927
928     // No longer a connection for this
929     // input group
930     if (!connection) {
931       this.remove();
932     }
933
934     return this.module.get('radius') +
935     offset;
936   },
937
938   outputPosition: function() {
939     var r = this.outputNode.get('radius');
940
941     return {
942       x: this.outputNode.position().x + r,
943       y: this.outputNode.position().y + 2
944       * r - 10
945     };
946   },
947
948   connectionChanged: function() {
949     var connections = this.model.get('
950     connections'),
951     connection = JSON.parse(
952     connections[this.inputGroup]);
953
954     // No longer a connection for this
955     // input group
956     if (!connection) {
957       this.remove();
958     }
959
960     return this.module.get('radius') +
961     offset;
962   },
963
964   outputPosition: function() {
965     var r = this.outputNode.get('radius');
966
967     return {
968       x: this.outputNode.position().x + r,
969       y: this.outputNode.position().y + 2
970       * r - 10
971     };
972   },
973
974   connectionChanged: function() {
975     var connections = this.model.get('
976     connections'),
977     connection = JSON.parse(
978     connections[this.inputGroup]);
979
980     // No longer a connection for this
981     // input group
982     if (!connection) {
983       this.remove();
984     }
985
986     return this.module.get('radius') +
987     offset;
988   },
989
990   outputPosition: function() {
991     var r = this.outputNode.get('radius');
992
993     return {
994       x: this.outputNode.position().x + r,
995       y: this.outputNode.position().y + 2
996       * r - 10
997     };
998   },
999
1000   connectionChanged: function() {
1001     var connections = this.model.get('
1002     connections'),
1003     connection = JSON.parse(
1004     connections[this.inputGroup]);
1005
1006     // No longer a connection for this
1007     // input group
1008     if (!connection) {
1009       this.remove();
1010     }
1011
1012     return this.module.get('radius') +
1013     offset;
1014   },
1015
1016   outputPosition: function() {
1017     var r = this.outputNode.get('radius');
1018
1019     return {
1020       x: this.outputNode.position().x + r,
1021       y: this.outputNode.position().y + 2
1022       * r - 10
1023     };
1024   },
1025
1026   connectionChanged: function() {
1027     var connections = this.model.get('
1028     connections'),
1029     connection = JSON.parse(
1030     connections[this.inputGroup]);
1031
1032     // No longer a connection for this
1033     // input group
1034     if (!connection) {
1035       this.remove();
1036     }
1037
1038     return this.module.get('radius') +
1039     offset;
1040   },
1041
1042   outputPosition: function() {
1043     var r = this.outputNode.get('radius');
1044
1045     return {
1046       x: this.outputNode.position().x + r,
1047       y: this.outputNode.position().y + 2
1048       * r - 10
1049     };
1050   },
1051
1052   connectionChanged: function() {
1053     var connections = this.model.get('
1054     connections'),
1055     connection = JSON.parse(
1056     connections[this.inputGroup]);
1057
1058     // No longer a connection for this
1059     // input group
1060     if (!connection) {
1061       this.remove();
1062     }
1063
1064     return this.module.get('radius') +
1065     offset;
1066   },
1067
1068   outputPosition: function() {
1069     var r = this.outputNode.get('radius');
1070
1071     return {
1072       x: this.outputNode.position().x + r,
1073       y: this.outputNode.position().y + 2
1074       * r - 10
1075     };
1076   },
1077
1078   connectionChanged: function() {
1079     var connections = this.model.get('
1080     connections'),
1081     connection = JSON.parse(
1082     connections[this.inputGroup]);
1083
1084     // No longer a connection for this
1085     // input group
1086     if (!connection) {
1087       this.remove();
1088     }
1089
1090     return this.module.get('radius') +
1091     offset;
1092   },
1093
1094   outputPosition: function() {
1095     var r = this.outputNode.get('radius');
1096
1097     return {
1098       x: this.outputNode.position().x + r,
1099       y: this.outputNode.position().y + 2
1100       * r - 10
1101     };
1102   },
1103
1104   connectionChanged: function() {
1105     var connections = this.model.get('
1106     connections'),
1107     connection = JSON.parse(
1108     connections[this.inputGroup]);
1109
1110     // No longer a connection for this
1111     // input group
1112     if (!connection) {
1113       this.remove();
1114     }
1115
1116     return this.module.get('radius') +
1117     offset;
1118   },
1119
1120   outputPosition: function() {
1121     var r = this.outputNode.get('radius');
1122
1123     return {
1124       x: this.outputNode.position().x + r,
1125       y: this.outputNode.position().y + 2
1126       * r - 10
1127     };
1128   },
1129
1130   connectionChanged: function() {
1131     var connections = this.model.get('
1132     connections'),
1133     connection = JSON.parse(
1134     connections[this.inputGroup]);
1135
1136     // No longer a connection for this
1137     // input group
1138     if (!connection) {
1139       this.remove();
1140     }
1141
1142     return this.module.get('radius') +
1143     offset;
1144   },
1145
1146   outputPosition: function() {
1147     var r = this.outputNode.get('radius');
1148
1149     return {
1150       x: this.outputNode.position().x + r,
1151       y: this.outputNode.position().y + 2
1152       * r - 10
1153     };
1154   },
1155
1156   connectionChanged: function() {
1157     var connections = this.model.get('
1158     connections'),
1159     connection = JSON.parse(
1160     connections[this.inputGroup]);
1161
1162     // No longer a connection for this
1163     // input group
1164     if (!connection) {
1165       this.remove();
1166     }
1167
1168     return this.module.get('radius') +
1169     offset;
1170   },
1171
1172   outputPosition: function() {
1173     var r = this.outputNode.get('radius');
1174
1175     return {
1176       x: this.outputNode.position().x + r,
1177       y: this.outputNode.position().y + 2
1178       * r - 10
1179     };
1180   },
1181
1182   connectionChanged: function() {
1183     var connections = this.model.get('
1184     connections'),
1185     connection = JSON.parse(
1186     connections[this.inputGroup]);
1187
1188     // No longer a connection for this
1189     // input group
1190     if (!connection) {
1191       this.remove();
1192     }
1193
1194     return this.module.get('radius') +
1195     offset;
1196   },
1197
1198   outputPosition: function() {
1199     var r = this.outputNode.get('radius');
1200
1201     return {
1202       x: this.outputNode.position().x + r,
1203       y: this.outputNode.position().y + 2
1204       * r - 10
1205     };
1206   },
1207
1208   connectionChanged: function() {
1209     var connections = this.model.get('
1210     connections'),
1211     connection = JSON.parse(
1212     connections[this.inputGroup]);
1213
1214     // No longer a connection for this
1215     // input group
1216     if (!connection) {
1217       this.remove();
1218     }
1219
1220     return this.module.get('radius') +
1221     offset;
1222   },
1223
1224   outputPosition: function() {
1225     var r = this.outputNode.get('radius');
1226
1227     return {
1228       x: this.outputNode.position().x + r,
1229       y: this.outputNode.position().y + 2
1230       * r - 10
1231     };
1232   },
1233
1234   connectionChanged
```

```

253     }
254
255     // A connection exists for this input
256     // group, but connected to a
257     // different output node
258     if (connection.output_node !== this.
259         outputNode.get('id')) {
260         // Stop listening to changes in the
261         // old output node
262         this.stopListening(this.outputNode);
263
264         // Find and bind to the new output
265         // node
266         this.outputNode = middleware.Nodes.
267             get(connection.output_node);
268         this.listenTo(this.outputNode, '
269             change', this.render);
270         this.render();
271     }
272 }
273
274 var NodeView = Backbone.NSView.extend({
275     tagName: 'svg:g',
276     className: 'node',
277     events: {
278         'mouseover .input': 'showInputTooltip'
279         ,
280         'mouseout .input': 'hideInputTooltip',
281         'click .input': 'toggleInputSelected',
282         'click .output': 'toggleOutputSelected'
283         ,
284         'click .run': 'runNode',
285         'click': 'toggleDetail'
286     }
287 }, {
288     initialize: function(options) {
289         this.editor = options.editor;
290         this.model = options.model;
291         this.module = middleware.
292             PackagedModules.findWhere({
293                 name: this.model.get('module')
294             });
295
296         this.d3el = d3.select(this.el)
297             .datum(this.model.position());
298
299         this.drag = d3.behavior.drag()
300             .origin(function(d) { return d; })
301             .on('dragstart', this.dragstarted.
302                 bind(this))
303             .on('drag', this.dragged.bind(this
304                 ))
305             .on('dragend', this.dragended.bind
306                 (this));
307
308         this.listenTo(this.model, 'change',
309             this.render);
310     },
311     template: _.template($('#graph-node-
312         template').html()),
313     render: function() {
314         var x = this.model.position().x;
315         var y = this.model.position().y;
316
317         this.d3el
318             .datum(this.model.position())
319             .attr('transform', 'translate(' +

```

```

311         x + ',,' + y + ''))
312         .call(this.drag);
313
314 this.$el.html(this.template({
315   r: this.model.get('radius'),
316   handlePosition: this.
317     dragHandlePosition(),
318   dragHandlePath: d3.svg.symbol().type
319     ('cross').size(150)(),
320   runPosition: this.runPosition(),
321   runPath: d3.svg.symbol().type('
322     triangle-up').size(150)(),
323   status: this.model.get('status'),
324   statusText: this.model.statusText(),
325   displayName: this.module.get('
326     displayName'),
327   inputs: this.module.get('inputs'),
328   output: this.module.get('outputs').
329     length,
330   inputPosition: this.inputPosition
331     });
332
333 var selectedInput = this.model.get('
334   selectedInput'),
335   selectedOutput = this.model.get('
336     selectedOutput');
337
338 if (selectedInput)
339   this.d3el.select('[data-name="' +
340     selectedInput.name + '"]')
341     .classed('selected', true);
342
343 if (selectedOutput)
344   this.d3el.select('.output')
345     .classed('selected', true);
346
347 if (this.model.isVisualization())
348
349   this.d3el.classed('visualization',
350     true);
351
352   return this;
353 },
354
355   dragstarted: function (d) {
356     this.dragStartPosition = _.clone(d);
357   },
358
359   dragged: function (d) {
360     if (!d3.select(d3.event.sourceEvent.
361       target).classed('drag-handle'))
362       return;
363
364     var x = d3.event.x;
365     var y = d3.event.y;
366     var r = this.model.get('radius');
367
368     var svg = d3.select(this.editor.el).
369       select('svg');
370     var bounds = {x: svg.attr('width'), y:
371       svg.attr('height')};
372
373     // Prevent element from being dragged
374     out bounds
375     if (x < 0) x = 0;
376     if (y < 0) y = 0;
377     if (y + r * 2 > bounds.y) y = bounds.y
378       - r * 2;
379     if (x + r * 2 > bounds.x) x = bounds.x
380       - r * 2;
381
382     this.model.position(x, y);
383     d3.select(this.el)

```



```

367         .attr('transform', 'translate(' + 396
368             (d.x = x) + ', ' + (d.y = y) + ', ' 397
369             ');');
370     },
371     dragended: function() {
372         if (this.dragMoved())
373             this.model.save();
374     },
375     dragMoved: function() {
376         var origin = this.dragStartPosition,
377             current = this.model.position();
378     },
379     return origin.x !== current.x ||
380             origin.y !== current.y;
381     },
382     showInputTooltip: function(event) {
383         var tooltip = d3.select('.input-
384             tooltip');
385         if (!tooltip[0][0])
386             tooltip = d3.select('body').append('
387                 div'
388                 .attr('class', 'input-tooltip');
389         var input = _.find(this.module.get('
390             inputs'), function(input) {
391                 return input.name === $(event.
392                     currentTarget).data('name');
393             });
394         tooltip.html(input.name);
395         var bounds = event.currentTarget.
396             getBoundingClientRect(),

```

```

        inputRadius = 5,
        tooltipWidth = parseFloat(tooltip.
            style('width')) / 2,
        tooltipHeight = parseFloat(tooltip
            .style('height')) + 5;

        tooltip
            .style('left', bounds.left -
                tooltipWidth + inputRadius + 'px'
            )
            .style('top', bounds.top -
                tooltipHeight + 'px')
            .style('visibility', 'visible');
    },
    hideInputTooltip: function() {
        d3.select('.input-tooltip')
            .style('visibility', 'hidden');
    },
    toggleInputSelected: function(event) {
        var previouslySelected = middguard.
            Nodes.find(function(node) {
                return node.get('selectedInput');
            });
        // Deselect the previously selected
        input.
            previouslySelected &&
            previouslySelected.set('
                selectedInput', null);
        var selectedGroup = _.find(this.module
            .get('inputs'), function(input) {
                return input.name === $(event.target
                    ).data('name');
            });

```

```

421     });
422     // If the clicked node was already
423     selected, return after toggling it
424     off.
425     if (previouslySelected &&
426         this.model.get('id') ===
427         previouslySelected.get('id') &&
428         selectedGroup.name ===
429         previouslySelected.get('name'))
430     {
431         return;
432     }
433     this.model.set('selectedInput',
434         selectedGroup);
435     this.connectNodes();
436     },
437     toggleOutputSelected: function (event) {
438         var previouslySelected = middguard.
439         Nodes.find(function (node) {
440             return node.get('selectedOutput');
441         })
442         previouslySelected &&
443         previouslySelected.set('
444             selectedOutput', null);
445         this.model.set('selectedOutput', true)
446         ;
447         this.connectNodes();
448     },
449     connectNodes: function () {
450         var input = middguard.Nodes.find(
451             function (node) {
452                 return node.get('selectedInput');
453             })
454         ;
455         var output = middguard.Nodes.find(
456             function (node) {
457                 return node.get('selectedOutput');
458             })
459         ;
460         if (!input || !output)
461             return;
462         var group = input.get('selectedInput')
463             .name;
464         input.connectToOutput(output, group);
465         input.set('selectedInput', null);
466         output.set('selectedOutput', null);
467     },
468     runNode: function () {
469         if (this.model.isVisualization()) {
470             middguard.toggleView(this.model.get(
471                 'id'));
472         } else {
473             this.model.run();
474         }
475     },
476     toggleDetail: function () {
477         var view = new NodeDetailView({model:
478             this.model});
479         this.editor.$('.node').removeClass('
480             selected');
481         this.$el.addClass('selected');
482     }
483 }

```

```

477         this.editor.setDetailView(view);
478     },
479
480     dragHandlePosition: function() {
481         var r = this.model.get('radius');
482         return {
483             x: r + -r * Math.sqrt(2) / 2 + 15,
484             y: r - r * Math.sqrt(2) / 2 + 15
485         };
486     },
487
488     runPosition: function() {
489         var r = this.model.get('radius');
490         return {
491             x: r + r * Math.sqrt(2) / 2 - 15,
492             y: r - r * Math.sqrt(2) / 2 + 15
493         };
494     },
495
496     /* Calculate each input circle's
497        position.
498        * Circles are arranged in rows of three
499        *   from the top down.
500        * Assume 5 pixel circle radius and 15
501        *   pixels spacing between
502        * circle centerpoints. Circles are
503        * centered around the node's
504        * centerline.
505
506        * Example: 5 inputs (x is an input
507        *   circle)
508        *   x <--15px--> x <--15px--> x
509        *   (15px between rows)
510        *   x <-- 15px --> x
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

537         this.render();
538     },
539     template: _.template(
540         '<h4><%- name %></h4>'
541         <div class="connection-groups"><div>'
542     ,
543     connectionGroupTemplate: _.template($(' #
544         connection-group-template').html()),
545     events: {
546         'click .connection': 'selectConnector'
547     },
548     render: function() {
549         this.$el.html(this.template({
550             name: this.module.get('displayName')
551         }));
552     },
553     this.addAllConnectionGroups();
554     return this;
555 },
556 },
557 },
558
559     addAllConnectionGroups: function() {
560         _.each(this.connections, (value, key)
561             => {
562             var inputs = value.connections.map(
563                 connection => connection.input),
564             outputs = value.connections.map(
565                 connection => connection.
566                     output),
567             outputNode = middguard.Nodes.get
568                 (value.output_node),
569
570             outputModule = middguard.
571                 PackagedModules.findWhere({
572                     name: outputNode.get('module')
573                 });
574
575             this.$(' .connection-groups').prepend
576                 (this.connectionGroupTemplate({
577                     inputGroupName: key,
578                     inputs: inputs,
579                     unconnectedInputs: this.model.
580                         unconnectedInputs(key),
581                     outputModuleName: outputModule.get
582                         ('displayName'),
583                     outputs: outputs,
584                     unconnectedOutputs: this.model.
585                         unconnectedOutputs(key)
586                 }));
587         });
588     },
589     deselectOutput: function() {
590         this.selectedOutput = null;
591         this.$(' .connection.output').
592             removeClass('selected');
593     },
594     deselectInput: function() {
595         this.selectedInput = null;
596         this.$(' .connection.input').
597             removeClass('selected');
598     },
599     selectConnector: function(event) {
600         var $clicked = $(event.target),
601         group = $clicked.closest(' .
602             connection-list-group').data('

```

```

592         inputgroup'),
593         name = $clicked.text(),
594         isInput = $clicked.hasClass('input
595         '),
596         isOutput = $clicked.hasClass('
597         output'),
598         sameGroup = this.
599         selectedInputGroup === group;
600
601         if (isInput) {
602             if (sameGroup) this.deselectInput();
603             else this.deselectOutput();
604
605             this.selectedInput = name;
606         }
607
608         if (isOutput) {
609             if (sameGroup) this.deselectOutput()
610             ;
611             else this.deselectInput();
612
613             this.selectedOutput = name;
614         }
615
616         this.selectedInputGroup = group;
617         $clicked.addClass('selected');
618         this.connectSelection();
619     },
620     connectSelection: function () {
621         if (!this.selectedInputGroup ||
622         !this.selectedInput ||
623         !this.selectedOutput) {
624             return;
625         }
626
627         var connections = this.connections[
628         this.selectedInputGroup].
629         connections;
630
631         var exists = _.find(connections, {
632         input: this.selectedInput ||
633         _.find(connections, {
634         output: this.
635         selectedOutput});
636
637         if (exists) {
638             exists.input = this.selectedInput;
639             exists.output = this.selectedOutput;
640         } else {
641             connections.push({
642             input: this.selectedInput,
643             output: this.selectedOutput
644             });
645         }
646
647         this.connections[this.
648         selectedInputGroup].connections =
649         connections;
650
651         this.deselectInput();
652         this.deselectOutput();
653         this.selectedInputGroup = null;
654
655         this.model.set('connections', JSON.
656         stringify(this.connections));
657         this.model.save();
658     }
659 }
660 }
661 }
662 }

```

./middguard/views/graph-editor-template.jade

```

1 script (type="text/template", id="graph-
  editor-template").
2 <div class="header">
3   <h4><%= name %></h4>
4 </div>
5 <div class="sidebar">
6   <h4>Modules</h4>
7   <ul class="modules-list"></ul>
8 </div>
9 <div class="editor"></div>
10 <div class="detail">
11 </div>
12
13 script (type="text/template", id="graph-node-
  template").
14 <circle class="outline" r="<%= r %>" cx
  = "<%= r %>" cy="<%= r %>"></circle>
15 <text x="<%= r %>" y="<%= r %>" style="
  text-anchor: middle;">
16   <%= displayName %>
17 </text>
18 <text class="status status-<%= status %>"
  style="text-anchor: middle;"
19   x="<%= r %>" y="<%= r + 20 %>">
20   <%= statusText %>
21 </text>
22 <circle class="drag-handle"
23   cx="<%= handlePosition.x %>" cy="<%=
  handlePosition.y %>" r="20"></circle>
24 <path
25   class="drag-handle"
26   transform="translate(<%= handlePosition.
  x %>, <%= handlePosition.y %>)"
27   d="<%= dragHandlePath %>"></path>
28 <path class="run"
  transform="translate(<%= runPosition.x
  %>, <%= runPosition.y %>) rotate(90)"
  d="<%= runPath %>"></path>
30 <%= inputs.forEach(function(input, i) { %>
  <circle class="connector input" data-
  name="<%= input.name %>"
  cx="<%= print(inputPosition(i, r,
    inputs.length).x) %>"
  cy="<%= print(inputPosition(i, r,
    inputs.length).y) %>"
  r="5"></circle>
36 <%= }); %>
37 <%= if (output) { %>
38   <circle class="connector output" r="5"
  cx="<%= r %>" cy="<%= 2 * r - 10 %>">
39 <%= } %>
40
41 script (type="text/template", id="connection-
  group-template").
42 <div class="connection-list-group clearfix"
  ">
43   data-inputgroup="<%= inputGroupName %>">
44   <div class="connection-list outputs">
45     <h5><%= outputModuleName %></h5>
46     <ul class="list-unstyled">
47       <%= outputs.forEach(function(output) {
  %>
48         <li class="connection connected
  output"><%= output %></li>
49       <%= }); %>
50       <%= unconnectedOutputs.forEach(function
  (output) { %>
51         <li class="connection unconnected
  output"><%= output %></li>
52       <%= }); %>
53     </ul>

```

```

54 </div>
55 <div class="connection-list inputs">
56 <h5><%- inputGroupName %></h5>
57 <ul class="list-unstyled">
58 <% inputs.forEach(function(input) { %>
59   <li class="connection connected
        input"><%- input %></li>
60   <% }); %>
61 <% unconnectedInputs.forEach(function(
        input) { %>
62   <li class="connection unconnected
        input"><%- input %></li>
63   <% }); %>
64 </ul>
65 </div>
66 </div>

/midguard/views/graphs-panel-template.jade
1 script(type='text/template', id='graphs-
  panel-template').
2   <form class="new-graph">
3     <div class="input-group">
4       <input type="text" id="new-graph-name"
          class="form-control" placeholder="
            Graph name">
5       <span class="input-group-btn">
6         <button class="btn btn-default" type
          ="button" id="create-new-graph">
          Create</button>
7       </span>
8     </div>
9   </form>
10 <h4>Graphs</h4>
11 <div class="list-group graphs-list"></div>

```

CHAPTER 2 OF APPENDIX

```

examples/simple/index.js
1 var middleware = require('...');
2
3 var app = middleguard({
4   // database
5   'knex config': require('./knexfile'),
6
7   // sessions
8   'secret key': process.env.SECRET_KEY || '
      major
9 });
10
11 app.module('read-tweets', require.resolve('
    ./read-tweets'));
12 app.module('count-hashtags', require.resolve(
    ('./count-hashtags')));
13 app.module('read-hashtags', require.resolve(
    './read-hashtags'));
14 app.module('hashtags-table', require.resolve(
    ('./hashtags-table'));
15 app.module('peek-table', require.resolve('./
    peek-table'));
16 app.module('hours-heatmap', require.resolve(
    './hours-heatmap'));
17 app.module('download-tweets-danarsilver',
18   require.resolve('./download-tweets-
    danarsilver'));

```

```

19 app.module('download-tweets-jack', require.
    resolve('./download-tweets-jack'));
20 app.module('aggregate-time', require.resolve(
    './aggregate-time'));
21 app.module('difference', require.resolve('./
    difference'));
22 app.module('mean-difference', require.
    resolve('./mean-difference'));
23
24 var port = process.env.PORT || 3000;
25 app.listen(port, function () {
26   console.log('Listening on port %d...',
    port);
27 });

```


examples/simple/knexfile.js

```

1 module.exports = {
2   client: 'sqlite3',
3   connection: {
4     filename: 'simple.db'
5   },
6   pool: {
7     min: 0,
8     max: 1,
9     afterCreate: function(conn, cb) {
10       conn.run('PRAGMA foreign_keys = ON', cb
11     );
12   }
13 }

```

examples/simple/download-tweets-danarsilver/index.js

```

1 var Promise = require('bluebird');
2 var fs = Promise.promisifyAll(require('fs'))
3 ;
4 var path = require('path');
5 var _ = require('lodash');
6 var Twitter = require('twitter');
7
8 exports.inputs = [];
9
10 exports.outputs = [
11   'handle',
12   'tweet',
13   'timestamp'
14 ];
15
16 exports.displayName = "@DanaRSilver";
17
18 var client = new Twitter({
19   consumer_key: 'fYwq7R6fP7np546j799dMXJj',
20   consumer_secret: '5
21     pAk01rSEZ1hrhbnRG6pJdcQIYkKmtIFNPvsyzV8jjyuhSnOc1
22     ',
23   access_token_key: '354037431-
24     sd7fd6inZSXXWaw9LmC3gmFfaHWx6p8UJq8JUaPDM
25     ',
26   access_token_secret: '
27     B8clfzqPuJqUWKnsGTSepV3eF1Y35RiIw7HI6YiMSOleS
28     '
29 });
30
31 client = Promise.promisifyAll(client);
32
33 exports.handle = function(context) {
34   var params = {screen_name: 'DanaRSilver',
35     count: 200};

```

```

28 return client.getAsync('statuses/
    user_timeline', params)
29 .spread(function(tweets, response) {
30   tweets = tweets.map(function(tweet) {
31     return {
32       handle: tweet.user.screen_name,
33       tweet: tweet.text,
34       timestamp: new Date(tweet.created_at
35         )
36     };
37   });
38   return context.table.knex.insert(tweets)
39   ;
40 });
41
42 exports.createTable = function(tableName,
    knex) {
43   return knex.schema.createTable(tableName,
    function(table) {
44     table.string('handle');
45     table.string('tweet');
46     table.dateTime('timestamp');
47   });
48 }

```

```

examples/simple/download-tweets-jack/index.js
1 var Promise = require('bluebird');
2 var fs = Promise.promisifyAll(require('fs'))
    ;
3 var path = require('path');
4 var _ = require('lodash');
5 var Twitter = require('twitter');
6
7 exports.inputs = [];
8
9 exports.outputs = [
10   'handle',
11   'tweet',
12   'timestamp'
13 ];
14
15 exports.displayName = "@jack";
16
17
18
19 var client = new Twitter({
20   consumer_key: 'fEYwq7R6fP7np546j799dMXJj',
21   consumer_secret: '5
    pAk01rSEZ1hrhbnRG6pJdcQIYkKMTIFNPvsyzV8jjyuhSnOC1
    ',
22   access_token_key: '354037431-
    sd7fd6inZSXWaw9LmC3gmFfaHWx6p8UJq8JUaPDM
    ',
23   access_token_secret: '
    B8clfzqPuJqUWKnsGTsEpV3eF1Y35RiIw7HI6YiMSOLEs
    ',
24 });
25
26 client = Promise.promisifyAll(client);
27
28 exports.handle = function(context) {

```

```

29 var params = {screen_name: 'jack', count:
200};
30 return client.getAsync('statuses/
  user_timeline', params)
31 .spread(function(tweets, response) {
32   tweets = tweets.map(function(tweet) {
33     return {
34       handle: tweet.user.screen_name,
35       tweet: tweet.text,
36       timestamp: new Date(tweet.created_at
37         )
38     };
39   });
40   return context.table.knex.insert(tweets)
41   ;
42   });
43
44 exports.createTable = function(tableName,
  knex) {
45   return knex.schema.createTable(tableName,
    function(table) {
46     table.string('handle');
47     table.string('tweet');
48     table.dateTime('timestamp');
49   });
50 };

      examples/simple/aggregate-time/index.js
1 var _ = require('lodash');
2 var Promise = require('bluebird');
3 var moment = require('moment');
4
5 exports.inputs = [
6   {name: 'tweets', inputs: ['handle', 'tweet
  ', 'timestamp']}
7 ];
8
9 exports.outputs = [
10  'handle',
11  'day',
12  'hour',
13  'count'
14 ];
15
16 exports.displayName = 'Time by Day/Hour';
17
18 exports.createTable = function(tableName,
  knex) {
19   return knex.schema.createTable(tableName,
    function(table) {
20     table.string('handle');
21     table.integer('day');
22     table.integer('hour');
23     table.integer('count');
24   });
25 };
26
27 exports.handle = function(context) {
28   var tweets = context.inputs.tweets,
29     timestampCol = context.inputs.tweets.
      cols.timestamp,
30     week = [];
31

```

```

32  __.range(24).forEach(function(hour) {
33  __.range(7).forEach(function(day) {
34    week.push({day: day, hour: hour, count
      : 0});
35  });
36  });
37
38  return tweets.knex.select('*')
39  .then(function(tweets) {
40    tweets.forEach(function(tweet) {
41      var m = moment(tweet[timestampCol]),
42      day = +m.format('d'),
43      hour = +m.format('H');
44
45      __.find(week, {day: day, hour: hour}).
46      count++;
47
48      return context.table.knex.insert(week);
49    });
50  });

```

```

      examples/simple/difference/index.js
1  var __ = require('lodash');
2  var Promise = require('bluebird');
3
4  exports.inputs = [
5    {name: 'tweets1', inputs: ['day', 'hour',
      'count']},
6    {name: 'tweets2', inputs: ['day', 'hour',
      'count']}
7  ];
8
9  exports.outputs = [
10   'day',
11   'hour',
12   'count1',
13   'count2',
14   'difference'
15  ];
16
17  exports.displayName = 'Difference by Hour';
18
19  exports.createTable = function(tableName,
    knex) {
20    return knex.schema.createTable(tableName,
      function(table) {
21      table.integer('day');
22      table.integer('hour');
23      table.integer('count1');
24      table.integer('count2');
25      table.integer('difference');
26    });
27  };
28
29  exports.handle = function(context) {
30    var tweets1 = context.inputs.tweets1,
31    tweets2 = context.inputs.tweets2,

```

```

32     week = [];
33
34     return Promise.join(tweets1.knex.select('*
35     '), tweets2.knex.select('*'),
36     function(tweets1, tweets2) {
37         _.range(24).forEach(function(hour) {
38             _.range(7).forEach(function(day) {
39                 var count1 = _.find(tweets1, {hour:
40                     hour, day: day}).count;
41                 var count2 = _.find(tweets2, {hour:
42                     hour, day: day}).count;
43                 week.push({
44                     day: day,
45                     hour: hour,
46                     count1: count1,
47                     count2: count2,
48                     difference: Math.abs(count1 -
49                         count2)
50                 });
51             });
52         });
53     });
54
55     return context.table.knex.insert(week);
56 }
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

examples/simple/hours-heatmap/index.js

```

1 var path = require('path');
2
3 exports.inputs = [
4     {name: 'hours', inputs: ['day', 'hour', '
5         count1', 'count2']}
6 ];
7 exports.outputs = [];
8
9 exports.displayName = "Hours Heatmap";
10
11 exports.visualization = true;
12
13 exports.static = path.join(__dirname, '
14     static');
15
16 exports.js = [
17     "hours-heatmap-view.js"
18 ];
19
20 exports.css = [
21     "hours-heatmap.css"
22 ];
23
24 exports.mainView = 'HoursHeatmapView';

```

```

examples/simple/hours-heatmap-static/hours-heatmap-view.js
1  var middguard = middguard || {};
2
3  (function() {
4    var HoursHeatmapView = middguard.View.
      extend({
5      id: 'hours-heatmap',
6
7      className: 'list-unstyled middguard-
        module',
8
9      tagName: 'div',
10
11     events: {
12       'mouseover .dayhour': '
         showInputTooltip',
13       'mouseout .dayhour': 'hideInputTooltip
         ',
14     },
15
16     template: _.template(
17       '<h4>Hours Heatmap</h4>' +
18       '<div class="heatmap-tooltip">' +
19       '  <span class="count1"></span>' +
20       '  <span class="count2"></span>' +
21       '</div>'
22     ),
23
24     initialize: function() {
25       this.context = this.createContext();
26       console.log(this.context);
27
28       var tableName = this.context.inputs.
29         hours.tableName;
30
31       this.context.listenTo(this.context.inputs.
32         hours.collection, 'reset', this.
33         render);
34
35       this.fetch(tableName, {reset: true,
36         data: {}});
37     },
38
39     render: function() {
40       this.$el.html(this.template());
41       this.$el.css('position', 'relative');
42
43       var data = this.context.inputs.hours.
44         collection.map(function(hours) {
45         return _.clone(hours.attributes);
46       });
47
48       var margin = {top: 0, left: 90, right:
49         0, bottom: 20};
50
51       var rowHeight = 60,
52         height = 7 * rowHeight - margin.
53         top - margin.bottom,
54         colWidth = 60,
55         width = colWidth * 24 - margin.
56         left - margin.right;
57
58       var week = ["Sunday", "Monday", "
59         Tuesday", "Wednesday", "Thursday",
60         "Friday", "Saturday"];
61
62       var x = this.x = d3.scale.linear()
63         .domain([0, 23])
64         .range([colWidth / 2, width -
65           colWidth / 2]);
66
67       var y = this.y = d3.scale.linear()
68         .domain([0, 6])

```

```

57 .range([rowHeight / 2, height -
58   rowHeight / 2]);
59
60 var xAxis = d3.svg.axis()
61   .scale(x)
62   .ticks(24)
63   .orient('bottom');
64
65 var yAxis = d3.svg.axis()
66   .scale(y)
67   .orient('left')
68   .ticks(6)
69   .tickFormat(function(d) {
70     return week[d];
71   });
72
73 var size = this.size = d3.scale.sqrt()
74   .domain([0, d3.max(data, function(
75     d) { return Math.max(d.count1,
76       d.count2); }])
77   .range([0, 25]);
78
79 var svg = d3.select(this.el).select('
80   svg')[0][0]
81   ? d3.select(this.el).select('
82     svg')
83   : d3.select(this.el).append('
84     svg');
85
86 svg = svg
87   .attr('width', width + margin.left
88     + margin.right)
89   .attr('height', height + margin.
90     top + margin.bottom)
91   .append('g')
92   .attr('transform', 'translate(' +
93     margin.left + ', ' + margin.top
94     + ')');
95
96 var circles = svg
97   .selectAll('g.dayhour')
98   .data(data)
99   .enter().append('g')
100   .attr('class', 'dayhour')
101   .attr('transform', function(d, i)
102     {
103       return 'translate(' + x(d.hour)
104         + ', ' + y(d.day) + ')';
105     });
106
107 circles.append('circle')
108   .attr('r', function(d) { return
109     size(d.count1); })
110   .style('fill', 'transparent')
111   .style('stroke-width', 2)
112   .style('stroke', 'orange');
113
114 circles.append('circle')
115   .attr('r', function(d) { return
116     size(d.count2); })
117   .style('fill', 'transparent')
118   .style('stroke-width', 2)
119   .style('stroke', 'steelblue');
120
121 svg.append('g')
122   .attr('class', 'y axis')
123   .call(yAxis);
124
125 svg.append('g')
126   .attr('class', 'x axis')
127   .attr('transform', 'translate(0,'
128     + height + ')')

```

```

114         .call(xAxis);
115
116     return this;
117 },
118
119 showInputTooltip: function(event) {
120     var tooltip = d3.select('.heatmap-
121         tooltip');
122
123     var d = d3.select(event.target).datum
124         ();
125     tooltip.select('.count1').text(d.
126         count1);
127     tooltip.select('.count2').text(d.
128         count2);
129
130     var bounds = event.currentTarget.
131         getBoundingClientRect(),
132         inputRadius = 5,
133         tooltipWidth = parseFloat(tooltip.
134             style('width')) / 2,
135         tooltipHeight = parseFloat(tooltip
136             .style('height')) + 5;
137
138     tooltip
139         .style('left', (this.x(d.hour) + 65)
140             + 'px')
141         .style('top', (this.y(d.day) - this.
142             size(Math.max(d.count1, d.count2)
143                 ) - 10) + 'px')
144         .style('visibility', 'visible');
145 },
146
147 hideInputTooltip: function() {
148     d3.select('.heatmap-tooltip')
149         .style('visibility', 'hidden');

```



```

examples/simple/hours-heatmap/static/hours-heatmap.css
1 .axis line {
2   fill: none;
3   stroke: #000;
4   shape-rendering: crispEdges;
5 }
6
7 .axis path {
8   fill: none;
9   stroke: none;
10 }
11
12 span.count1, span.count2 {
13   font-size: 16px;
14 }
15
16 span.count1 {
17   color: orange;
18   padding-right: 10px;
19 }
20
21 span.count2 {
22   color: steelblue;
23 }
24
25 .heatmap-tooltip {
26   position: absolute;
27   padding: 10px;
28   border-radius: 5px;
29   font-size: 12px;
30   line-height: 1.4;
31   text-align: center;
32   color: #fff;
33   background-color: rgba(0, 0, 0, 0.7);
34   visibility: hidden;
35   z-index: 1;
36 }
37
38 .heatmap-tooltip:after {
39   position: absolute;
40   top: 100%;
41   left: 50%;
42   margin-left: -5px;
43   width: 0;
44   border-top: 5px solid rgba(0, 0, 0, 0.7);
45   border-right: 5px solid transparent;
46   border-left: 5px solid transparent;
47   content: " ";
48 }

```

examples/simple/peek-table/index.js

```

1 var path = require('path');
2
3 exports.inputs = [
4   {name: 'table', inputs: ['col1', 'col2', '
      col3', 'col4']}
5 ];
6
7 exports.outputs = [];
8
9 exports.displayName = "Peek Table";
10
11 exports.visualization = true;
12
13 exports.static = path.join(__dirname, '
      static');
14
15 exports.js = [
16   "peek-table-view.js"
17 ];
18
19 exports.css = [
20   "peek-table.css"
21 ];
22
23 exports.mainView = 'PeekTableView';

```

examples/simple/peek-table/static/peek-table-view.js

```

1 var middguard = middguard || {};
2
3 (function() {
4   var PeekTableView = middguard.View.extend
      ({
5     id: 'hashtags-table',
6
7     className: 'list-unstyled middguard-
        module',
8
9     tagName: 'table',
10
11    template: _.template(
12      '<th><tr><td><%- col1 %></td><td><%-
        col2 %></td><td><%- col3 %></td><td>
        <%- col4 %></td></tr></th>',
13    ),
14
15    rowTemplate: _.template(
16      '<tr><td><%- col1 %></td><td><%- col2
        %></td><td><%- col3 %></td><td><%-
        col4 %></td></tr>',
17    ),
18
19    initialize: function() {
20      this.context = this.createContext();
21
22      var collection = this.context.inputs.
        table.collection;
23
24      var tableName = this.context.inputs.
        table.tableName;
25      this.listenTo(collection, 'reset',
        this.addAllRows);
26

```

```
27  this.fetch(tableName, {reset: true,
28    data: {}});
29  },
30  render: function() {
31    var cols = this.context.inputs.table.
      cols;
32
33    this.$el.html(this.template({
34      col1: cols.col1,
35      col2: cols.col2,
36      col3: cols.col3,
37      col4: cols.col4
38    }));
39
40    return this;
41  },
```

```

42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
    addAllRows: function() {
        var collection = this.context.inputs.
            table.collection;
        collection.each(this.addRow, this);
    },
    addOneRow: function(row) {
        var cols = this.context.inputs.table.
            cols;
        console.log(cols, row)
    },
    this.$el.append(this.rowTemplate({
        col1: row.get(cols.col1),
        col2: row.get(cols.col2),
        col3: row.get(cols.col3),
        col4: row.get(cols.col4)
    }));
    }
    });
    middguard.PeekTableView = PeekTableView;
    middguard.addModule('PeekTableView',
        PeekTableView);
    })();

```

examples/simple/peek-table/static/peek-table.css

```

1 #hashtags-table {
2   padding: 10px;
3 }
4
5 #hashtags-table tr:nth-child(even) {
6   background-color: #e5e5e5;
7 }
8
9 #hashtags-table tr {
10  padding: 4px;
11 }

```

BIBLIOGRAPHY

- [1] C. Andrews and J. Billings. Middguard at dinofun world. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pages 129–130, Oct 2015.
- [2] Jeremy Ashkenas and DocumentCloud. Backbone.js. <http://backbonejs.org/>, 2016.
- [3] Liu Bin and Chen Gang. Eagleeyes: Performing data analysis using an interactive dataflow. In *Visual Analytics Science and Technology (VAST), 2015 IEEE Conference on*, pages 165–166, Oct 2015.
- [4] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.
- [5] Kevin Burke, Kyle Conroy, Ryan Horn, Frank Stratton, and Guillaume Binet. Flask restful. <http://flask-restful-cn.readthedocs.io/en/0.3.5/>, 2015.
- [6] VA Community. Vast challenge 2014. <http://www.vacommunity.org/VAST+Challenge+2014>, 2014.
- [7] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. <http://www.rfc-editor.org/rfc/rfc6455.txt>.
- [8] Node.js Foundation. About — node.js. <https://nodejs.org/en/about/>, 2016.
- [9] Tim Griesser. Knex.js - a sql query builder for javascript. <http://knexjs.org/>, 2016.
- [10] Mozilla Developer Network and individual contributors. Websockets - web apis — mdn. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, 2016.
- [11] Armin Ronacher. Flask. <http://flask.pocoo.org/>, 2016.
- [12] John Stasko, Carsten Görg, and Zhicheng Liu. Jigsaw: Supporting investigative analysis through interactive visualization. *Information Visualization*, 7(2):118–132, April 2008.

- [13] James J. Thomas and Kristin A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 2005.
- [14] Chris Weaver. Building highly-coordinated visualizations in Improvise. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis)*, pages 159–166, Austin, TX, October 2004. IEEE Computer Society.