

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	23
4.4	Visual Programming	25
4.5	Extensibility	28
4.6	Real-time Collaboration	30
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	74
	Bibliography	85

LIST OF FIGURES

1.1	Web interface for the VAST 2014 entry.	3
1.2	Browser memory profiles with View Reference Counting.	8
1.3	The snapshots portion of figure 1.2, cropped for readability.	8
4.1	The connection configuration for a node.	20
4.2	The context passed into a “Time by Day/Hour” node’s handler function.	23
4.3	MiddGuard’s graph editor user interface.	26
4.4	An annotated analytic node.	28
4.5	Code for an example analytic module.	33
4.6	The main configuration file for a visualization module.	34
4.7	Example code for a MiddGuard-based server.	35
5.1	The complete graph from a use case.	37
5.2	A visualization rendered in the browser.	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.

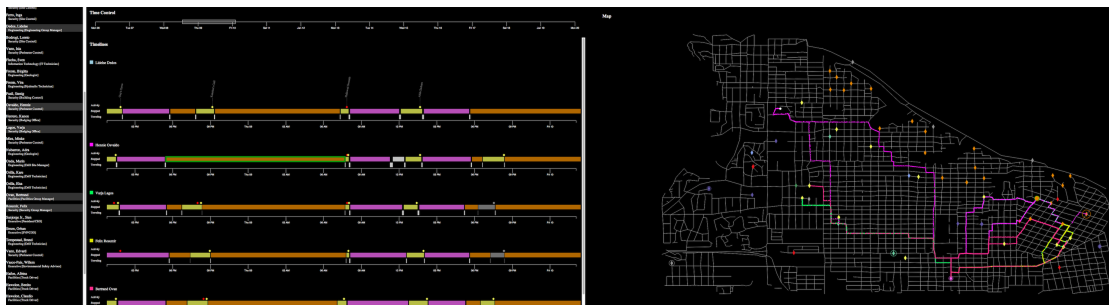


Figure 1.1: The web interface for Andrews and Silver’s VAST 2014 entry. The visualizations, from left to right, are a list of people, a master brushable timeline and individual timelines for each person listed with selectable events, and a map of GPS traces from the individuals’ cars.

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.

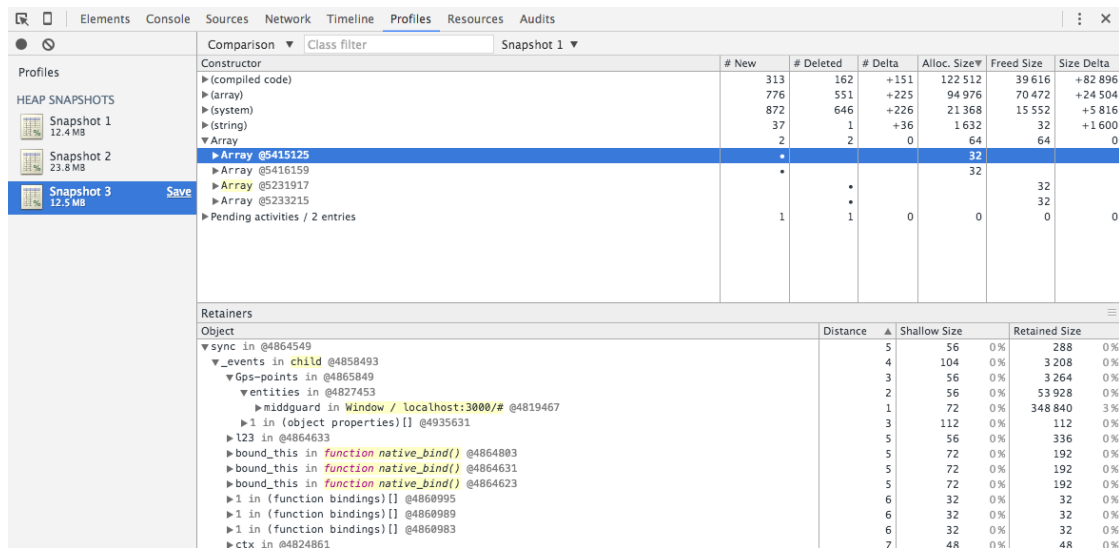


Figure 1.2: A screen capture of the Google Chrome DevTools Profiler demonstrating the efficacy of View Reference Counting. The panel on the left shows three snapshots. Snapshot 1 was taken before a view was added. Snapshot 2 was taken after a view was added and rendered with a significant amount of data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

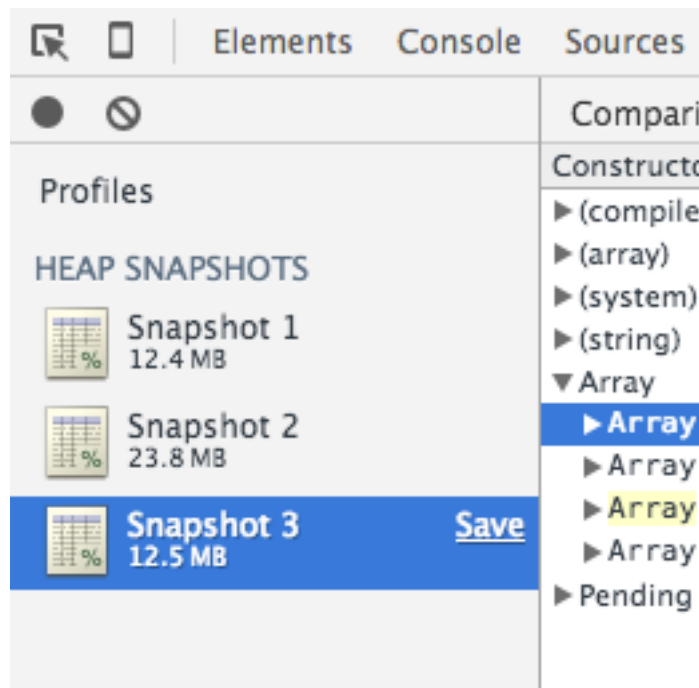


Figure 1.3: The snapshots portion of figure 1.2, cropped for readability.

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,

such as table joins.

Input group to output node mappings tell us where a specific input's data lives, but not what the data looks like or how to refer to it. That is, we have the table to look in, but we don't know what its schema is and in particular, what its columns are named. Unless the only SQL we want to run is `SELECT * FROM 'output table'`, we need more information.

We address this at the second level of our connection protocol: intra-connection mappings within the input group to output node connection, that identify the column names in the output table. This is another set of key-value pairs that map the names the input node has assigned to each attribute in an input group, to the corresponding column to access in a the output table. When generating the contextual input for a node, this mapping is included for each input group. Like at the higher level of input group mappings, developers can look up the the output table column name using a key they pick to represent that attribute.

Listing 4.1 shows an example connection configuration for a node called “Time by Day/Hour” that aggregates data by day of the week and hour of the day. The configuration for “Time by Day/Hour” has one input group, called “tweets”, which is connected to the output node with id 9. The `output_node` field serves as a foreign key referencing another row in the same table. The column-level connections between the input group and output node 9 are stored within the input group. Column mappings are stored in an array called `connections`. Each object within the `connections` array has an a key `input` and a key `output`. The value of `input` is the name the input node has given to the column and the value of `output` is the name the output node has given to the column.


```

{
  "tweets": {
    "output_node": 9,
    "connections": [
      {
        "output": "handle",
        "input": "handle"
      },
      {
        "output": "tweet",
        "input": "tweet"
      },
      {
        "output": "timestamp",
        "input": "timestamp"
      }
    ]
  }
}

```

Figure 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 per-

formed 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 associated 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 a 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.

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

```

{
  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'
    }
  }
}

```

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

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 4.3. 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 4.4 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. Press-



Figure 4.3: 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.

ing 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 4.4 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 4.3. 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.

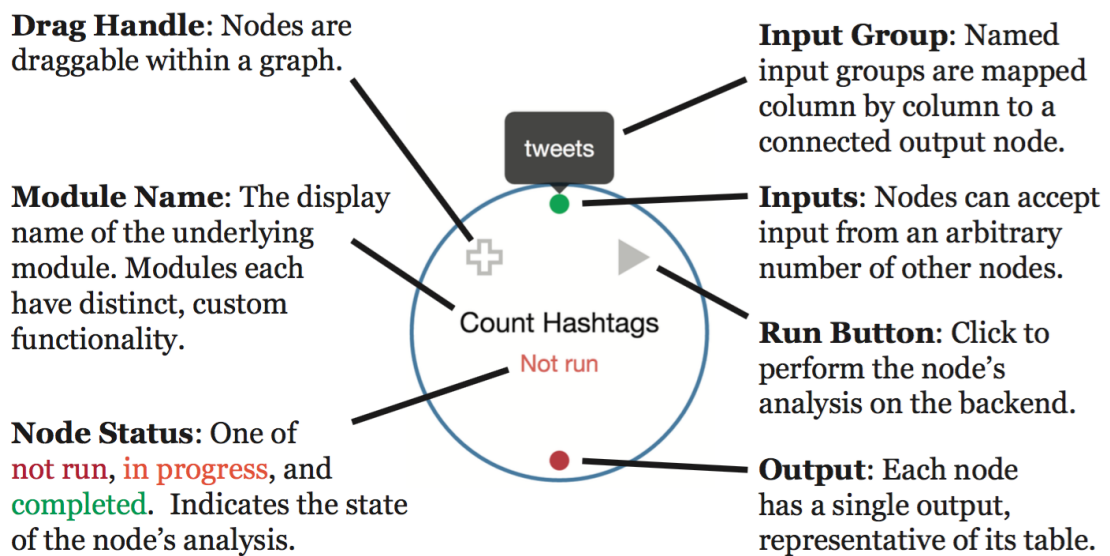


Figure 4.4: An annotated analytic node taken from the MiddGuard graph editor. Important features are annotated and the node's only input group, "tweets", is moused over to show its accompanying tooltip.

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.

Modules are short and designed to be written quickly. The code in figure 4.5 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 4.3 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 4.5. 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 4.5 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 4.6 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 4.7). 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 4.5: 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 4.6: 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 || 'keep me secret'
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 4.7: Code for an investigation’s MiddGuard-based 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 4.5 and 4.6 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 the 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

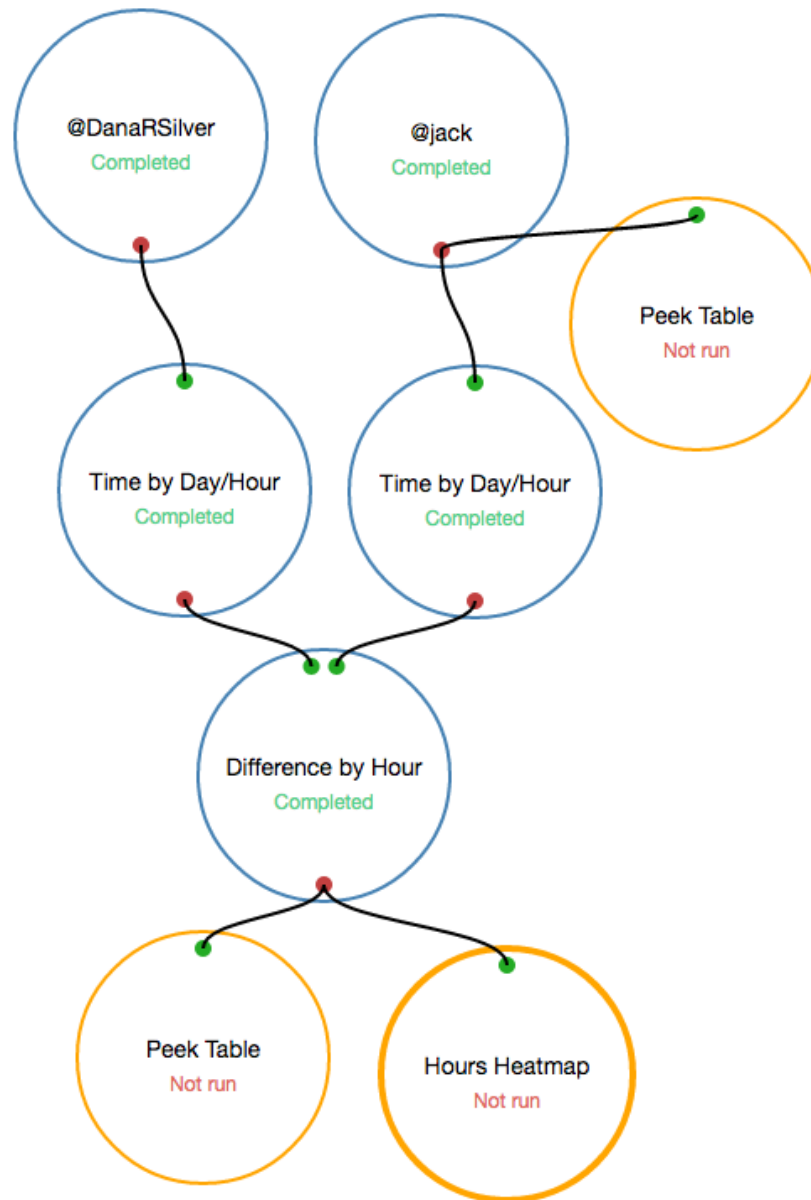


Figure 5.1: The complete graph from the mock investigation used to develop and test MiddGuard.

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

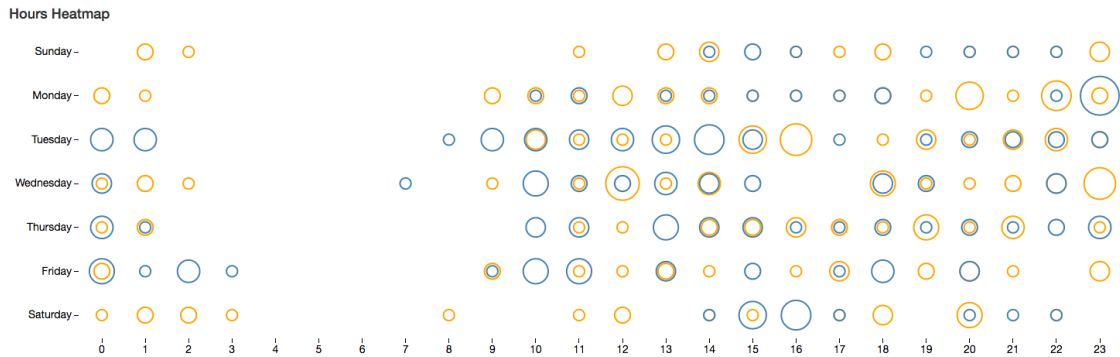


Figure 5.2: The “Hours Heatmap” visualization from the mock investigation used to develop and test MiddGuard. @DanaRSilver’s tweets are orange, @jack’s are blue. Circle radii are mapped to the number of times each person tweeted in that hour and day of the week.

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

```
1 'use strict';
2
3 module.exports = require('./middguard/middguard');
```

middguard/application.js

```
1 'use strict';
2
3 /**
4  * Module dependencies.
5  * @private
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.middguardInit = function () {
29   this.middguardExpressMiddleware();
```

```

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 MiddelGuard.
45 *
46 * @private
47 */
48
49 app.middlewareExpressMiddleware = function
50   middlewareExpressMiddleware() {
51   this.use('/static', express.static(path.join(__dirname, '..
52     ', 'static')));
53
54   var knex = require('knex')(this.get('knex config'));
55   var sessionStore = new KnexSessionStore({knex: knex});
56   this.set('sessionStore', sessionStore);
57
58   // Set up ORM middleware
59   require('./config/bookshelf')(this);
60
61   this.use(cookieParser(this.get('secret key')));
62   this.use(bodyParser.urlencoded({extended: true}));
63   this.use(bodyParser.json());
64
65   this.set('session', session({
66     store: sessionStore,
67     secret: this.get('secret key'),
68     resave: true,
69     saveUninitialized: true,
70     cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
71   }));
72
73   this.use(this.get('view engine', 'jade'));
74   this.set('views', path.join(__dirname, 'views'));
75
76   this.use(this.get('session'));
77
78   var io = this.get('io');
79   var session = this.get('session');
80
81   io.use(io(session));
82
83   io.use((socket, next) => {
84     socket.bookshelf = this.get('bookshelf');
85     next();
86   });
87
88   /**
89    * Register an analytics module with the 'middguard' app.
90    *
91    * @return 'middguard.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.
103         static));
104     }
105
106     register.add(new AnalyticsModule({
107       name: name,
108       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
120  * application (which is a 'Function') as its
121  * callback.
122  *
123  * This is the same as 'express.listen', but uses
124  * the already created server, rather than creating
125  * a new one in 'listen'. The 'http.Server' must
126  * already be created to setup socket.io.
127  *
128  * @return {http.Server}
129  * @public
130  */
131
132 app.listen = function listen() {
133   var server = this.get('http server');
134   return server.listen.apply(server, arguments);
135 };

```

midguard/midguard.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 midguard 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   app.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(io,
40   //   app.get('sessionStore'),

```

```

41 // app.get('cookieParser');
42 //
43 // bookshelfConfig(app);
44 //
45 // // require('./middguard/loaders/models_loader')(app);
46 // // require('./middguard/loaders/analytics_loader')(app);
47 //
48 // sessionSockets.on('connection', require('./middguard/
    socket'));
49 //
50 // require('./middguard/routes')(app);
51
52 app.middguardInit();
53
54 return app;
55 };

```

middguard/socket/index.js

```

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 in user
13   if (!socket.handshake.session.user) return;
14
15   // Set up sockets middguard internal sockets
16   socket.on('messages:create', (data, cb) => message.create(
    socket, data, cb));
17   socket.on('messages:read', (data, cb) => message.readAll(
    socket, data, cb));
18
19   socket.on('modules:read', (data, cb) => modules.readAll(
    socket, data, cb));
20
21   socket.on('analyst:read', (data, cb) => analyst.read(socket
    , data, cb));
22   socket.on('analysts:read', (data, cb) => analyst.readAll(
    socket, data, cb));
23
24   socket.on('node:connect', (data, cb) => node.connect(socket
    , data, cb));
25   socket.on('node:run', (data, cb) => node.run(socket, data,
    cb));
26   socket.on('nodes:create', (data, cb) => node.create(socket,
    data, cb));
27   socket.on('nodes:read', (data, cb) => node.readAll(socket,
    data, cb));
28   socket.on('nodes:update', (data, cb) => node.update(socket,
    data, cb));
29
30   var Graph = Bookshelf.model('Graph');

```

```

31 patchModelToEmit(socket, 'graph', Graph);
32 setupSocketEvents(socket, 'graph', Graph);
33
34 Bookshelf.model('Node').fetchAll()
35 .then(nodes => nodes.each(node => node.createReadStream(socket)
36   socket)));
37
38 // Set up sockets to call analytics from client
39 // Patched models will automatically emit create, update,
40   and delete events
41 // Bookshelf.collection('analytics').each(function (
42   analyticsAttrs) {
43   // var name = analyticsAttrs.get('name');
44   // var requirePath = analyticsAttrs.get('requirePath');
45   //
46   // socket.on('analytics:' + name, function (data,
47     callback) {
48     require(requirePath)(Bookshelf, data);
49     //
50     // });
51     // });
52   };
53
54 function patchModelToEmit(socket, modelName, model) {
55   if (!model.prototype._emitting) {
56     var _initialize = model.prototype.initialize;
57
58     model.prototype.initialize = function () {
59       var args = Array.prototype.slice.call(arguments);
60       _initialize.apply(args);
61
62       this.on('created', function (model, attrs, options) {
63         // If the model was created on the client, we don't
64         want to emit a
65         // create event, since we need to assign an id on the
66         creator via
67         // a callback and do a broadcast.emit for everyone
68         else.
69         // The create listener will take care of this.
70         if (!options.clientCreate) {
71           io.emit(pluralize(modelName) + ':create', model.
72             toJSON());
73         }
74       });
75     };
76   }
77 }
78
79 function setupSocketEvents(socket, modelName, model) {
80   // Set up create, read, update, delete sockets for each
81   model
82   socket.on(pluralize(modelName) + ':create', function (data,
83     callback) {
84       // Pass clientCreate to save so the model won't emit
85       anything on the
86       // created event and confuse the client.
87       // Create is a special case since the model on the
88       creating client doesn't
89       have an id yet.
90       new model().save(data, {clientCreate: true})
91       .then(function (newModel) {
92         callback(null, newModel.toJSON());
93       })
94       .catch(function (error) {
95         throw new Error(error);
96       });
97   });

```

```

98 socket.on(modelName + ':update', function (data, callback) {
99     new model({id: _.result(data, 'id')})
100     .save(_.omit(data, 'id'), {patch: true});
101 });
102
103 socket.on(modelName + 'delete', function (data, callback)
104     {
105         var x = new model({id: _.result(data, 'id')});
106         //console.log(String(x.destroy()).resolveFromSyncValue));
107         new model({id: _.result(data, 'id')}).destroy();
108     });
109
110 socket.on(pluralize(modelName) + 'read', function (data,
111     callback) {
112     if (data) {
113         var fetchData = new model().where(data).fetchAll();
114     } else {
115         //if fetching all models at once
116         var fetchData = new model().fetchAll();
117     }
118     fetchData
119     .then(function (collection) {
120         callback(null, collection.toJSON());
121     })
122     .catch(function (error) {
123         callback(error);
124     });
125
126 socket.on(modelName + 'read', function (data, callback) {
127     new model({id: _.result(data, 'id')}).fetch({require:
128         true})
129     .then(function (fetchedModel) {
130         callback(null, fetchedModel.toJSON());
131     })
132     .catch(model.NotFoundError, function () {
133         callback({'error': 'Model ${modelName} not found.'});
134     })
135     .catch(function (error) {
136         callback(error);
137     });
138 }

```

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 };

```

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', node.toJSON());
35     })
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.
51       get('module')});
52     var inputModule = modules.findWhere({name: inputNode.get(
53       'module')});
54
55     // Get outputs list from the corresponding output module
56     var outputs = require(outputModule.get('requirePath')).
57       outputs;
58
59     // Get inputs list from the corresponding input module
60     var inputGroups = require(inputModule.get('requirePath'))
61       .inputs;
62
63     var inputs = inputGroups.filter(function(group) {
64       return group.name === data.inputGroup;
65     })[0].inputs;
66
67     // The array of connections we'll set on the input node
68     var connections = {
69       output_node: data.outputNode,
70       connections: []
71     };
72
73     if (data.connections &&
74         validateConnections(data.connections, inputs, outputs
75         )) {
76       // Use 'data.connections' if the connections are valid
77       connections.connections = data.connections;
78     } else {
79       // Match input and output names
80       connections.connections = connectionsByName(inputs,
81         outputs);
82     }
83
84     * Generate the connections array by matching names between
85
86     */
87
88     * Validate that all potential inputs and outputs have inputs
89     * and outputs on
90     * with the same name on the respective nodes.
91     *
92     * @private
93     * @param {Object[]} connections The passed in data of
94     * connections to set.
95     * @param {Object[]} inputs Named inputs on the existing
96     * input node.
97     * @param {Object[]} outputs Named outputs on the existing
98     * output node.
99
100     */
101     function validateConnections(connections, inputs, outputs) {
102       var potentialInputs = connections.map(connection =>
103         connection.input);
104       var potentialOutputs = connections.map(connection =>
105         connection.output);
106
107       return potentialInputs.length === potentialOutputs.length
108         &&
109         inputs.every(input => _.has(potentialInputs, input))
110         &&
111         outputs.every(output => _.has(potentialOutputs,
112           output));
113     }
114
115     // Match input and output names
116     connections.connections = connectionsByName(inputs,
117       outputs);
118
119     * Generate the connections array by matching names between
120
121     */
122
123     */
124
125     */
126
127     */
128
129     */
130
131     */
132
133     */
134
135     */
136
137     */
138
139     */
140
141     */
142
143     */
144
145     */
146
147     */
148
149     */
150
151     */
152
153     */
154
155     */
156
157     */
158
159     */
160
161     */
162
163     */
164
165     */
166
167     */
168
169     */
170
171     */
172
173     */
174
175     */
176
177     */
178
179     */
180
181     */
182
183     */
184
185     */
186
187     */
188
189     */
190
191     */
192
193     */
194
195     */
196
197     */
198
199     */
200
201     */
202
203     */
204
205     */
206
207     */
208
209     */
210
211     */
212
213     */
214
215     */
216
217     */
218
219     */
220
221     */
222
223     */
224
225     */
226
227     */
228
229     */
230
231     */
232
233     */
234
235     */
236
237     */
238
239     */
240
241     */
242
243     */
244
245     */
246
247     */
248
249     */
250
251     */
252
253     */
254
255     */
256
257     */
258
259     */
260
261     */
262
263     */
264
265     */
266
267     */
268
269     */
270
271     */
272
273     */
274
275     */
276
277     */
278
279     */
280
281     */
282
283     */
284
285     */
286
287     */
288
289     */
290
291     */
292
293     */
294
295     */
296
297     */
298
299     */
300
301     */
302
303     */
304
305     */
306
307     */
308
309     */
310
311     */
312
313     */
314
315     */
316
317     */
318
319     */
320
321     */
322
323     */
324
325     */
326
327     */
328
329     */
330
331     */
332
333     */
334
335     */
336
337     */
338
339     */
340
341     */
342
343     */
344
345     */
346
347     */
348
349     */
350
351     */
352
353     */
354
355     */
356
357     */
358
359     */
360
361     */
362
363     */
364
365     */
366
367     */
368
369     */
370
371     */
372
373     */
374
375     */
376
377     */
378
379     */
380
381     */
382
383     */
384
385     */
386
387     */
388
389     */
390
391     */
392
393     */
394
395     */
396
397     */
398
399     */
400
401     */
402
403     */
404
405     */
406
407     */
408
409     */
410
411     */
412
413     */
414
415     */
416
417     */
418
419     */
420
421     */
422
423     */
424
425     */
426
427     */
428
429     */
430
431     */
432
433     */
434
435     */
436
437     */
438
439     */
440
441     */
442
443     */
444
445     */
446
447     */
448
449     */
450
451     */
452
453     */
454
455     */
456
457     */
458
459     */
460
461     */
462
463     */
464
465     */
466
467     */
468
469     */
470
471     */
472
473     */
474
475     */
476
477     */
478
479     */
480
481     */
482
483     */
484
485     */
486
487     */
488
489     */
490
491     */
492
493     */
494
495     */
496
497     */
498
499     */
500
501     */
502
503     */
504
505     */
506
507     */
508
509     */
510
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     */
1000
1001     */
1002
1003     */
1004
1005     */
1006
1007     */
1008
1009     */
1010
1011     */
1012
1013     */
1014
1015     */
1016
1017     */
1018
1019     */
1020
1021     */
1022
1023     */
1024
1025     */
1026
1027     */
1028
1029     */
1030
1031     */
1032
1033     */
1034
1035     */
1036
1037     */
1038
1039     */
1040
1041     */
1042
1043     */
1044
1045     */
1046
1047     */
1048
1049     */
1050
1051     */
1052
1053     */
1054
1055     */
1056
1057     */
1058
1059     */
1060
1061     */
1062
1063     */
1064
1065     */
1066
1067     */
1068
1069     */
1070
1071     */
1072
1073     */
1074
1075     */
1076
1077     */
1078
1079     */
1080
1081     */
1082
1083     */
1084
1085     */
1086
1087     */
1088
1089     */
1090
1091     */
1092
1093     */
1094
1095     */
1096
1097     */
1098
1099     */
1100
1101     */
1102
1103     */
1104
1105     */
1106
1107     */
1108
1109     */
1110
1111     */
1112
1113     */
1114
1115     */
1116
1117     */
1118
1119     */
1120
1121     */
1122
1123     */
1124
1125     */
1126
1127     */
1128
1129     */
1130
1131     */
1132
1133     */
1134
1135     */
1136
1137     */
1138
1139     */
1140
1141     */
1142
1143     */
1144
1145     */
1146
1147     */
1148
1149     */
1150
1151     */
1152
1153     */
1154
1155     */
1156
1157     */
1158
1159     */
1160
1161     */
1162
1163     */
1164
1165     */
1166
1167     */
1168
1169     */
1170
1171     */
1172
1173     */
1174
1175     */
1176
1177     */
1178
1179     */
1180
1181     */
1182
1183     */
1184
1185     */
1186
1187     */
1188
1189     */
1190
1191     */
1192
1193     */
1194
1195     */
1196
1197     */
1198
1199     */
1200
1201     */
1202
1203     */
1204
1205     */
1206
1207     */
1208
1209     */
1210
1211     */
1212
1213     */
1214
1215     */
1216
1217     */
1218
1219     */
1220
1221     */
1222
1223     */
1224
1225     */
1226
1227     */
1228
1229     */
1230
1231     */
1232
1233     */
1234
1235     */
1236
1237     */
1238
1239     */
1240
1241     */
1242
1243     */
1244
1245     */
1246
1247     */
1248
1249     */
1250
1251     */
1252
1253     */
1254
1255     */
1256
1257     */
1258
1259     */
1260
1261     */
1262
1263     */
1264
1265     */
1266
1267     */
1268
1269     */
1270
1271     */
1272
1273     */
1274
1275     */
1276
1277     */
1278
1279     */
1280
1281     */
1282
1283     */
1284
1285     */
1286
1287     */
1288
1289     */
1290
1291     */
1292
1293     */
1294
1295     */
1296
1297     */
1298
1299     */
1300
1301     */
1302
1303     */
1304
1305     */
1306
1307     */
1308
1309     */
1310
1311     */
1312
1313     */
1314
1315     */
1316
1317     */
1318
1319     */
1320
1321     */
1322
1323     */
1324
1325     */
1326
1327     */
1328
1329     */
1330
1331     */
1332
1333     */
1334
1335     */
1336
1337     */
1338
1339     */
1340
1341     */
1342
1343     */
1344
1345     */
1346
1347     */
1348
1349     */
1350
1351     */
1352
1353     */
1354
1355     */
1356
1357     */
1358
1359     */
1360
1361     */
1362
1363     */
1364
1365     */
1366
1367     */
1368
1369     */
1370
1371     */
1372
1373     */
1374
1375     */
1376
1377     */
1378
1379     */
1380
1381     */
1382
1383     */
1384
1385     */
1386
1387     */
1388
1389     */
1390
1391     */
1392
1393     */
1394
1395     */
1396
1397     */
1398
1399     */
1400
1401     */
1402
1403     */
1404
1405     */
1406
1407     */
1408
1409     */
1410
1411     */
1412
1413     */
1414
1415     */
1416
1417     */
1418
1419     */
1420
1421     */
1422
1423     */
1424
1425     */
1426
1427     */
1428
1429     */
1430
1431     */
1432
1433     */
1434
1435     */
1436
1437     */
1438
1439     */
1440
1441     */
1442
1443     */
1444
1445     */
1446
1447     */
1448
1449     */
1450
1451     */
1452
1453     */
1454
1455     */
1456
1457     */
1458
1459     */
1460
1461     */
1462
1463     */
1464
1465     */
1466
1467     */
1468
1469     */
1470
1471     */
1472
1473     */
1474
1475     */
1476
1477     */
1478
1479     */
1480
1481     */
1482
1483     */
1484
1485     */
1486
1487     */
1488
1489     */
1490
1491     */
1492
1493     */
1494
1495     */
1496
1497     */
1498
1499     */
1500
1501     */
1502
1503     */
1504
1505     */
1506
1507     */
1508
1509     */
1510
1511     */
1512
1513     */
1514
1515     */
1516
1517     */
1518
1519     */
1520
1521     */
1522
1523     */
1524
1525     */
1526
1527     */
1528
1529     */
1530
1531     */
1532
1533     */
1534
1535     */
1536
1537     */
1538
1539     */
1540
1541     */
1542
1543     */
1544
1545     */
1546
1547     */
1548
1549     */
1550
1551     */
1552
1553     */
1554
1555     */
1556
1557     */
1558
1559     */
1560
1561     */
1562
1563     */
1564
1565     */
1566
1567     */
1568
1569     */
1570
1571     */
1572
1573     */
1574
1575     */
1576
1577     */
1578
1579     */
1580
1581     */
1582
1583     */
1584
1585     */
1586
1587     */
1588
1589     */
1590
1591     */
1592
1593     */
1594
1595     */
1596
1597     */
1598
1599     */
1600
1601     */
1602
1603     */
1604
1605     */
1606
1607     */
1608
1609     */
1610
1611     */
1612
1613     */
1614
1615     */
1616
1617     */
1618
1619     */
1620
1621     */
1622
1623     */
1624
1625     */
1626
1627     */
1628
1629     */
1630
1631     */
1632
1633     */
1634
1635     */
1636
1637     */
1638
1639     */
1640
1641     */
1642
1643     */
1644
1645     */
1646
1647     */
1648
1649     */
1650
1651     */
1652
1653     */
1654
1655     */
1656
1657     */
1658
1659     */
1660
1661     */
1662
1663     */
1664
1665     */
1666
1667     */
1668
1669     */
1670
1671     */
1672
1673     */
1674
1675     */
1676
1677     */
1678
1679     */
1680
1681     */
1682
1683     */
1684
1685     */
1686
1687     */
1688
1689     */
1690
1691     */
1692
1693     */
1694
1695     */
1696
1697     */
1698
1699     */
1700
1701     */
1702
1703     */
1704
1705     */
1706
1707     */
1708
1709     */
1710
1711     */
1712
1713     */
1714
1715     */
1716
1717     */
1718
1719     */
1720
1721     */
1722
1723     */
1724
1725     */
1726
1727     */
1728
1729     */
1730
1731     */
1732
1733     */
1734
1735     */
1736
1737     */
1738
1739     */
1740
1741     */
1742
1743     */
1744
1745     */
1746
1747     */
1748
1749     */
1750
1751     */
1752
1753     */
1754
1755     */
1756
1757     */
1758
1759     */
1760
1761     */
1762
1763     */
1764
1765     */
1766
1767     */
1768
1769     */
1770
1771     */
1772
1773     */
1774
1775     */
1776
1777     */
1778
1779     */
1780
1781     */
1782
1783     */
1784
1785     */
1786
1787     */
1788
1789     */
1790
1791     */
1792
1793     */
1794
1795     */
1796
1797     */
1798
1799     */
1800
1801     */
1802
1803     */
1804
1805     */
1806
1807     */
1808
1809     */
1810
1811     */
1812
1813     */
1814
1815     */
1816
1817     */
1818
1819     */
1820
1821     */
1822
1823     */
1824
1825     */
1826
1827     */
1828
1829     */
1830
1831     */
1832
1833     */
1834
1835     */
1836
1837     */
1838
1839     */
1840
1841     */
1842
1843     */
1844
1845     */
1846
1847     */
1848
1849     */
1850
1851     */
1852
1853     */
1854
1855     */
1856
1857     */
1858
1859     */
1860
1861     */
1862
1863     */
1864
1865     */
1866
1867     */
1868
1869     */
1870
1871     */
1872
1873     */
1874
1875     */
1876
1877     */
1878
1879     */
1880
1881     */
1882
1883     */
1884
1885     */
1886
1887     */
1888
1889     */
1890
1891     */
1892
1893     */
1894
1895     */
1896
1897     */
1898
1899     */
1900
1901     */
1902
1903     */
1904
1905     */
1906
1907     */
1908
1909     */
1909
1910     */
1911     */
1912     */
1913     */
1914     */
1915     */
1916     */
1917     */
1918     */
1919     */
1920     */
1921     */
1922     */
1923     */
1924     */
1925     */
1926     */
1927     */
1928     */
1929     */
1930     */
1931     */
1932     */
1933     */
1934     */
1935     */
1936     */
1937     */
1938     */
1939     */
1940     */
1941     */
1942     */
1943     */
1944     */
1945     */
1946     */
1947     */
1948     */
1949     */
1950     */
1951     */
1952     */
1953     */
1954     */
1955     */
1956     */
1957     */
1958     */
1959     */
1960     */
1961     */
1962     */
1963     */
1964     */
1965     */
1966     */
1967     */
1968     */
1969     */
1970     */
1971     */
1972     */
1973     */
1974     */
1975     */
1976     */
1977     */
1978     */
1979     */
1980     */
1981     */
1982     */
1983     */
1984     */
1985     */
1986     */
1987     */
1988     */
1989     */
1990     */
1991     */
1992     */
1993     */
1994     */
1995     */
1996     */
1997     */
1998     */
1999     */
2000     */

```

```

107 * Returns an array with size equivalent to the cardinality
108 * of inputs    outputs.
109 * @private
110 * @param {Object[]} inputs Inputs to match
111 * @param {Object[]} outputs Outputs to match
112 */
113 function connectionsByName(inputs, outputs) {
114   return outputs.filter(output => _.indexOf(inputs, output) >
115     -1)
116 }
117
118 exports.run = function(socket, data, callback) {
119   var Node = socket.bookshelf.model('Node');
120   var modules = socket.bookshelf.collection('analytics');
121
122   new Node({id: data.id})
123   .fetch()
124   .tap(node => node.ensureTable())
125   .then(node => node.save({status: 1}))
126   .then(function(node) {
127     socket.emit('nodes:update', node.toJSON());
128     socket.broadcast.emit('nodes:update', node.toJSON());
129     return node;
130   })
131   .then(node => Promise.join(node, node.outputNodes()))
132   .spread(function(node, outputs) {
133     var module = modules.findWhere({name: node.get('module')
134     }),
135     connections = JSON.parse(node.get('connections')),
136     context = {};
137
138     context.inputs = _.reduce(_.keys(connections), function(
139       inputs, inputGroup) {
140         var groupConnections = connections[inputGroup].
141           connections;
142
143         // mapping input to output.
144         var columns = _.reduce(groupConnections, function(
145           connections, pair) {
146             connections[pair.input] = pair.output;
147             return connections;
148           }, {});
149
150         inputs[inputGroup] = {};
151         inputs[inputGroup].knex = socket.bookshelf.knex(outputs
152           [inputGroup].get('table'));
153         inputs[inputGroup].cols = columns;
154         inputs[inputGroup].tableName = outputs[inputGroup].get(
155           'table');
156
157         context.table = {};
158         context.table.knex = socket.bookshelf.knex(node.get('
159           table'));
160         context.table.name = node.get('table');
161
162         var handle = require(module.get('requirePath')).handle;
163         return Promise.join(node, handle(context));
164       })
165       .spread(function(node, result) {
166         return node.save({status: 2});
167       })
168       .then(function(node) {
169         socket.emit('nodes:update', node.toJSON());
170         socket.broadcast.emit('nodes:update', node.toJSON());
171         .catch(callback);
172       })
173     );
174
175     // Reduce the array of input output pairs to a single

```

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 };
```


middguard/models/node.js

```

1  'use strict';
2
3  var _ = require('lodash');
4  var Promise = require('bluebird');
5
6  /**
7   * Register the 'Node' model in the Bookshelf registry.
8   *
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.belongsTo('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', `${this.get('module')}_${count + 1}`);
43        });
44      },
45
46      /**
47       * Get a mapping from input group names to output nodes.
48       *
49       * @return a promise for an object mapping input group
50       name
51       to a fetched output node
52       */
53      outputNodes: function () {
54        var connections = JSON.parse(this.get('connections'));
55
56        return Promise.reduce(_.keys(connections), function (
57          outputs, inputGroup) {
58            var outputId = connections[inputGroup].output_node;
59
60            return new Node({id: outputId}).fetch()
61              .then(node => {
62                outputs[inputGroup] = node;
63                return outputs;
64              }, {});
65          }, {});
66
67      /**
68       * Create this node's table if it doesn't already.
69       */
70      ensureTable: function () {
71        return Bookshelf.knex.schema.hasTable(this.get('table'))
72          ?
73          .then(exists => {
74            if (!exists) {
75              return this.module().createTable(this.get('table'),
76                Bookshelf.knex);
77            }
78          })
79        ;
80      }
81    });
82  }
83

```

```

77 module: function () {
78
79     var modules = Bookshelf.collection('analytics'),
80         moduleName = this.get('module'),
81         module = modules.findWhere({name: moduleName});
82
83     return require(module.get('requirePath'));
84 },
85
86 /**
87  * Set an input group on the node's connections.
88  * The text column "connections" remains in its
89     stringified JSON state.
90  *
91  * @param {String} inputGroup Input group to set.
92  * @param {Object} connections Connections to set for `
93     inputGroup`.
94  * @return this
95  */
96 setInputGroup: function(inputGroup, connections) {
97     let groups = JSON.parse(this.get('connections')) || {};
98     groups[inputGroup] = connections;
99
100     return this.set('connections', JSON.stringify(groups));
101 },
102
103 createReadStream: function(socket) {
104     let table = Bookshelf.knex(this.get('table'));
105
106     socket.on(`${this.get('table')}:read`, (data, callback)
107         => {
108         if (!_.isEmpty(data)) {
109             var query = Bookshelf.knex(this.get('table')).where
110                 (data).select('*');
111         } else {
112             var query = Bookshelf.knex(this.get('table')).
113                 select('*');
114         }
115
116         query.then(results => callback(null, results));
117     });
118 }
119
120 }
121
122 }
123
124 }
125
126 return Bookshelf.model('Node', Node);
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343 }
344
345 }
346
347 }
348
349 }
350
351 }
352
353 }
354
355 }
356
357 }
358
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448
449 }
450
451 }
452
453 }
454
455 }
456
457 }
458
459 }
460
461 }
462
463 }
464
465 }
466
467 }
468
469 }
470
471 }
472
473 }
474
475 }
476
477 }
478
479 }
480
481 }
482
483 }
484
485 }
486
487 }
488
489 }
490
491 }
492
493 }
494
495 }
496
497 }
498
499 }
500
501 }
502
503 }
504
505 }
506
507 }
508
509 }
510
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 }
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
740 }
740
741 }
741
742 }
742
743 }
743
744 }
744
745 }
745
746 }
746
747 }
747
748 }
748
749 }
749
750 }
750
751 }
751
752 }
752
753 }
753
754 }
754
755 }
755
756 }
756
757 }
757
758 }
758
759 }
759
760 }
760
761 }
761
762 }
762
763 }
763
764 }
764
765 }
765
766 }
766
767 }
767
768 }
768
769 }
769
770 }
770
771 }
771
772 }
772
773 }
773
774 }
774
775 }
775
776 }
776
777 }
777
778 }
778
779 }
779
780 }
780
781 }
781
782 }
782
783 }
783
784 }
784
785 }
785
786 }
786
787 }
787
788 }
788
789 }
789
790 }
790
791 }
791
792 }
792
793 }
793
794 }
794
795 }
795
796 }
796
797 }
797
798 }
798
799 }
799
800 }
800
801 }
801
802 }
802
803 }
803
804 }
804
805 }
805
806 }
806
807 }
807
808 }
808
809 }
809
810 }
810
811 }
811
812 }
812
813 }
813
814 }
814
815 }
815
816 }
816
817 }
817
818 }
818
819 }
819
820 }
820
821 }
821
822 }
822
823 }
823
824 }
824
825 }
825
826 }
826
827 }
827
828 }
828
829 }
829
830 }
830
831 }
831
832 }
832
833 }
833
834 }
834
835 }
835
836 }
836
837 }
837
838 }
838
839 }
839
840 }
840
841 }
841
842 }
842
843 }
843
844 }
844
845 }
845
846 }
846
847 }
847
848 }
848
849 }
849
850 }
850
851 }
851
852 }
852
853 }
853
854 }
854
855 }
855
856 }
856
857 }
857
858 }
858
859 }
859
860 }
860
861 }
861
862 }
862
863 }
863
864 }
864
865 }
865
866 }
866
867 }
867
868 }
868
869 }
869
870 }
870
871 }
871
872 }
872
873 }
873
874 }
874
875 }
875
876 }
876
877 }
877
878 }
878
879 }
879
880 }
880
881 }
881
882 }
882
883 }
883
884 }
884
885 }
885
886 }
886
887 }
887
888 }
888
889 }
889
890 }
890
891 }
891
892 }
892
893 }
893
894 }
894
895 }
895
896 }
896
897 }
897
898 }
898
899 }
899
900 }
900
901 }
901
902 }
902
903 }
903
904 }
904
905 }
905
906 }
906
907 }
907
908 }
908
909 }
909
910 }
910
911 }
911
912 }
912
913 }
913
914 }
914
915 }
915
916 }
916
917 }
917
918 }
918
919 }
919
920 }
920
921 }
921
922 }
922
923 }
923
924 }
924
925 }
925
926 }
926
927 }
927
928 }
928
929 }
929
930 }
930
931 }
931
932 }
932
933 }
933
934 }
934
935 }
935
936 }
936
937 }
937
938 }
938
939 }
939
940 }
940
941 }
941
942 }
942
943 }
943
944 }
944
945 }
945
946 }
946
947 }
947
948 }
948
949 }
949
950 }
950
951 }
951
952 }
952
953 }
953
954 }
954
955 }
955
956 }
956
957 }
957
958 }
958
959 }
959
960 }
960
961 }
961
962 }
962
963 }
963
964 }
964
965 }
965
966 }
966
967 }
967
968 }
968
969 }
969
970 }
970
971 }
971
972 }
972
973 }
973
974 }
974
975 }
975
976 }
976
977 }
977
978 }
978
979 }
979
980 }
980
981 }
981
982 }
982
983 }
983
984 }
984
985 }
985
986 }
986
987 }
987
988 }
988
989 }
989
990 }
990
991 }
991
992 }
992
993 }
993
994 }
994
995 }
995
996 }
996
997 }
997
998 }
998
999 }
999
1000 }
1000
1001 }
1001
1002 }
1002
1003 }
1003
1004 }
1004
1005 }
1005
1006 }
1006
1007 }
1007
1008 }
1008
1009 }
1009
1010 }
1010
1011 }
1011
1012 }
1012
1013 }
1013
1014 }
1014
1015 }
1015
1016 }
1016
1017 }
1017
1018 }
1018
1019 }
1019
1020 }
1020
1021 }
1021
1022 }
1022
1023 }
1023
1024 }
1024
1025 }
1025
1026 }
1026
1027 }
1027
1028 }
1028
1029 }
1029
1030 }
1030
1031 }
1031
1032 }
1032
1033 }
1033
1034 }
1034
1035 }
1035
1036 }
1036
1037 }
1037
1038 }
1038
1039 }
1039
1040 }
1040
1041 }
1041
1042 }
1042
1043 }
1043
1044 }
1044
1045 }
1045
1046 }
1046
1047 }
1047
1048 }
1048
1049 }
1049
1050 }
1050
1051 }
1051
1052 }
1052
1053 }
1053
1054 }
1054
1055 }
1055
1056 }
1056
1057 }
1057
1058 }
1058
1059 }
1059
1060 }
1060
1061 }
1061
1062 }
1062
1063 }
1063
1064 }
1064
1065 }
1065
1066 }
1066
1067 }
1067
1068 }
1068
1069 }
1069
1070 }
1070
1071 }
1071
1072 }
1072
1073 }
1073
1074 }
1074
1075 }
1075
1076 }
1076
1077 }
1077
1078 }
1078
1079 }
1079
1080 }
1080
1081 }
1081
1082 }
1082
1083 }
1083
1084 }
1084
1085 }
1085
1086 }
1086
1087 }
1087
1088 }
1088
1089 }
1089
1090 }
1090
1091 }
1091
1092 }
1092
1093 }
1093
1094 }
1094
1095 }
1095
1096 }
1096
1097 }
1097
1098 }
1098
1099 }
1099
1100 }
1100
1101 }
1101
1102 }
1102
1103 }
1103
1104 }
1104
1105 }
1105
1106 }
1106
1107 }
1107
1108 }
1108
1109 }
1109
1110 }
1110
1111 }
1111
1112 }
1112
1113 }
1113
1114 }
1114
1115 }
1115
1116 }
1116
1117 }
1117
1118 }
1118
1119 }
1119
1120 }
1120
1121 }
1121
1122 }
1122
1123 }
1123
1124 }
1124
1125 }
1125
1126 }
1126
1127 }
1127
1128 }
1128
1129 }
1129
1130 }
1130
1131 }
1131
1132 }
1132
1133 }
1133
1134 }
1134
1135 }
1135
1136 }
1136
1137 }
1137
1138 }
1138
1139 }
1139
1140 }
1140
1141 }
1141
1142 }
1142
1143 }
1143
1144 }
1144
1145 }
1145
1146 }
1146
1147 }
1147
1148 }
1148
1149 }
1149
1150 }
1150
1151 }
1151
1152 }
1152
1153 }
1153
1154 }
1154
1155 }
1155
1156 }
1156
1157 }
1157
1158 }
1158
1159 }
1159
1160 }
1160
1161 }
1161
1162 }
1162
1163 }
1163
1164 }
1164
1165 }
1165
1166 }
1166
1167 }
1167
1168 }
1168
1169 }
1169
1170 }
1170
1171 }
1171
1172 }
1172
1173 }
1173
1174 }
1174
1175 }
1175
1176 }
1176
1177 }
1177
1178 }
1178
1179 }
1179
1180 }
1180
1181 }
1181
1182 }
1182
1183 }
1183
1184 }
1184
1185 }
1185
1186 }
1186
1187 }
1187
1188 }
1188
1189 }
1189
1190 }
1190
1191 }
1191
1192 }
1192
1193 }
1193
1194 }
1194
1195 }
1195
1196 }
1196
1197 }
1197
1198 }
1198
1199 }
1199
1200 }
1200
1201 }
1201
1202 }
1202
1203 }
1203
1204 }
1204
1205 }
1205
1206 }
1206
1207 }
1207
1208 }
1208
1209 }
1209
1210 }
1210
1211 }
1211
1212 }
1212
1213 }
1213
1214 }
1214
1215 }
1215
1216 }
1216
1217 }
1217
1218 }
1218
1219 }
1219
1220 }
1220
1221 }
1221
1222 }
1222
1223 }
1223
1224 }
1224
1225 }
1225
1226 }
1226
1227 }
1227
1228 }
1228
1229 }
1229
1230 }
1230
1231 }
1231
1232 }
1232
1233 }
1233
1234 }
1234
1235 }
1235
1236 }
1236
1237 }
1237
1238 }
1238
1239 }
1239
1240 }
1240
1241 }
1241
1242 }
1242
1243 }
1243
1244 }
1244
1245 }
1245
1246 }
1246
1247 }
1247
1248 }
1248
1249 }
1249
1250 }
1250
1251 }
1251
1252 }
1252
1253 }
1253
1254 }
1254
1255 }
1255
1256 }
1256
1257 }
1257
1258 }
1258
1259 }
1259
1260 }
1260
1261 }
1261
1262 }
1262
1263 }
1263
1264 }
1264
1265 }
1265
1266 }
1266
1267 }
1267
1268 }
1268
1269 }
1269
1270 }
1270
1271 }
1271
1272 }
1272
1273 }
1273
1274 }
1274
1275 }
1275
1276 }
1276
1277 }
1277
1278 }
1278
1279 }
1279
1280 }
1280
1281 }
1281
1282 }
1282
1283 }
1283
1284 }
1284
1285 }
1285
1286 }
1286
1287 }
1287
1288 }
1288
1289 }
1289
1290 }
1290
1291 }
1291
1292 }
1292
1293 }
1293
1294 }
1294
1295 }
1295
1296 }
1296
1297 }
1297
1298 }
1298
1299 }
1299
1300 }
1300
1301 }
1301
1302 }
1302
1303 }
1303
1304 }
1304
1305 }
1305
1306 }
1306
1307 }
1307
1308 }
1308
1309 }
1309
1310 }
1310
1311 }
1311
1312 }
1312
1313 }
1313
1314 }
1314
1315 }
1315
1316 }
1316
1317 }
1317
1318 }
1318
1319 }
1319
1320 }
1320
1321 }
1321
1322 }
1322
1323 }
1323
1324 }
1324
1325 }
1325
1326 }
1326
1327 }
1327
1328 }
1328
1329 }
1329
1330 }
1330
1331 }
1331
1332 }
1332
1333 }
1333
1334 }
1334
1335 }
1335
1336 }
1336
1337 }
1337
1338 }
1338
1339 }
1339
1340 }
1340
1341 }
1341
1342 }
1342
1343 }
1343
1344 }
1344
1345 }
1345
1346 }
1346
1347 }
1347
1348 }
1348
1349 }
1349
1350 }
1350
1351 }
1351
1352 }
1352
1353 }
1353
1354 }
1354
1355 }
1355
1356 }
1356
1357 }
1357
1358 }
1358
1359 }
1359
1360 }
1360
1361 }
1361
1362 }
1362
1363 }
1363
1364 }
1364
1365 }
1365
1366 }
1366
1367 }
1367
1368 }
1368
1369 }
1369
1370 }
1370
1371 }
1371
1372 }
1372
1373 }
1373
1374 }
1374
1375 }
1375
1376 }
1376
1377 }
1377
1378 }
1378
1379 }
1379
1380 }
1380
1381 }
1381
1382 }
1382
1383 }
1383
1384 }
1384
1385 }
1385
1386 }
1386
1387 }
1387
1388 }
1388
1389 }
1389
1390 }
1390
1391 }
1391
1392 }
1392
1393 }
1393
1394 }
1394
1395 }
1395
1396 }
1396
1397 }
1397
1398 }
1398
1399 }
1399
1400 }
1400
1401 }
1401
1402 }
1402
1403 }
1403
1404 }
1404
1405 }
1405
1406 }
1406
1407 }
1407
1408 }
1408
1409 }
1409
1410 }
1410
1411 }
1411
1412 }
1412
1413 }
1413
1414 }
1414
1415 }
1415
1416 }
1416
1417 }
1417
1418 }
1418
1419 }
1419
1420 }
1420
1421 }
1421
1422 }
1422
1423 }
1423
1424 }
1424
1425 }
1425
1426 }
1426
1427 }
1427
1428 }
1428
1429 }
1429
1430 }
1430
1431 }
1431
1432 }
1432
1433 }
1433
1434 }
1434
1435 }
1435
1436 }
1436
1437 }
1437
1438 }
1438
1439 }
1439
1440 }
1440
1441 }
1441
1442 }
1442
1443 }
1443
1444 }
1444
1445 }
1445
1446 }
1446
1447 }
1447
1448 }
1448
1449 }
1449
1450 }
1450
1451 }
1451
1452 }
1452
1453 }
1453
1454 }
1454
1455 }
1455
1456 }
1456
1457 }
1457
1458 }
1458
1459 }
1459
1460 }
1460
1461 }
1461
1462 }
1462
1463 }
1463
1464 }
1464
1465 }
1465
1466 }
1466
1467 }
1467
1468 }
1468
1469 }
1469
1470 }
1470
1471 }
1471
1472 }
1472
1473 }
1473
1474 }
1474
1475 }
1475
1476 }
1476
1477 }
1477
1478 }
1478
1479 }
1479
1480 }
1480
1481 }
1481
1482 }
1482
1483 }
1483
1484 }
1484
1485 }
1485
1486 }
1486
1487 }
1487
1488 }
1488
1489 }
1489
1490 }
1490
1491 }
1491
1492 }
1492
1493 }
1493
1494 }
1494
1495 }
1495
1496 }
1496
1497 }
1497
1498 }
1498
1499 }
1499
1500 }
1500
1501 }
1501
1502 }
1502
1503 }
1503
1504 }
1504
1505 }
1505
1506 }
1506
1507 }
1507
1508 }
1508
1509 }
1509
1510 }
1510
1511 }
1511
1512 }
1512
1513 }
1513
1514 }
1514
1515 }
1515
1516 }
1516
1517 }
1517
1518 }
1518
1519 }
1519
1520 }
1520
1521 }
1521
1522 }
1522
1523 }
1523
1524 }
1524
1525 }
1525
1526 }
1526
1527 }
1527
1528 }
1528
1529 }
1529
1530 }
1530
1531 }
1531
1532 }
1532
1533 }
1533
1534 }
1534
1535 }
1535
1536 }
1536
1537 }
1537
1538 }
1538
1539 }
1539
1540 }
1540
1541 }
1541
1542 }
1542
1543 }
1543
1544 }
1544
1545 }
1545
1546 }
1546
1547 }
1547
1548 }
1548
1549 }
1549
1550 }
1550
1551 }
1551
1552 }
1552
1553 }
1553
1554 }
1554
1555 }
1555
1556 }
1556
1557 }
1557
1558 }
1558
1559 }
1559
1560 }
1560
1561 }
1561
1562 }
1562
1563 }
1563
1564 }
1564
1565 }
1565
1566 }
1566
1567 }
1567
1568 }
1568
1569 }
1569
1570 }
1570
1571 }
1571
1572 }
1572
1573 }
1573
1574 }
1574
1575 }
1575
1576 }
1576
1577 }
1577
1578 }
1578
1579 }
1579
1580 }
1580
1581 }
1581
1582 }
1582
1583 }
1583
1584 }
1584
1585 }
1585
1586 }
1586
1587 }
1587
1588 }
1588
1589 }
1589
1590 }
1590
1591 }
1591
1592 }
1592
1593 }
1593
1594 }
1594
1595 }
1595
1596 }
1596
1597 }
1597
1598 }
1598
1599 }
1599
1600 }
1600
1601 }
1601
1602 }
1602
1603 }
1603
1604 }
1604
1605 }
1605
1606 }
1606
1607 }
1607
1608 }
1608
1609 }
1609
1610 }
1610
161
```

middguard/migrations/20140728124252`initial.js

```

1 'use strict';
2
3 exports.up = function(knex, Promise) {
4   return knex.schema.createTable('analyst', function(table) {
5     table.increments('id').primary();
6     table.text('username').unique();
7     table.text('password');
8   })
9   .createTable('message', function(table) {
10     table.increments('id').primary();
11     table.integer('analyst_id').references('analyst.id');
12     table.text('state');
13     table.text('content');
14     table.dateTime('timestamp');
15   })
16   .createTable('graph', function(table) {
17     table.increments('id').primary();
18     table.string('name');
19   })
20   .createTable('node', function(table) {
21     table.increments('id').primary();
22     table.integer('graph_id').references('graph.id');
23     table.string('module');
24     table.string('table');
25     table.integer('status').defaultTo(0);
26     table.string('connections').defaultTo('{}');
27   });
28 };
29
30 exports.down = function(knex, Promise) {
31   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(table) {
5     table.integer('radius').defaultTo(75);
6
7     // These are the top left coordinates of the node,
8     // not the center coordinates.
9     table.integer('position_x').defaultTo(0);
10    table.integer('position_y').defaultTo(0);
11  });
12 };
13
14 exports.down = function(knex, Promise) {
15   return knex.schema.table('node', function(table) {
16     table.dropColumns('radius', 'position_x', 'position_y');
17   });
18 };

```

static/js/entities.js

```

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', '
11       serverDelete');
12     this.ioBind('create', this.serverCreate, this);
13     this.ioBind('update', this.serverUpdate, this);
14     this.ioBind('delete', this.serverDelete, this);
15
16     this.listenTo(this, 'sync', this.addViewReferences);
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   serverUpdate: function (data) {
27     var exists = this.get(data.id);
28     if (exists) exists.set(data);
29   },
30   serverDelete: function (data) {
31     // Already deleted from database, so don't need to
32     model.destroy
33     var exists = this.get(data.id);
34     if (exists) this.remove(exists);
35   },
36   addViewReferences: function (collection, response,
37     options) {
38     var middguard_view_name = options.middguard_view_name;
39
40     // if a view name wasn't passed in we can't do anything

```

```

      about it
    if (!middguard_view_name)
      return;

    console.log('Adding view references for view "' +
      middguard_view_name +
      '" to ' + response.length + ' fetched
      models.');
```

```

    // get models added to the collection that match the
    criteria
    // we fetched for
    (options.data
      ? collection.where(options.data)
      : collection).forEach(function (model) {
        var currentViews = model.get('middguard_views');

        // if 'middguard_views' doesn't exist on the model,
        // set it to an empty
        // array
        if (!currentViews)
          model.set('middguard_views', []);

        currentViews = model.get('middguard_views');

        // if the view has already been added
        if (currentViews.indexOf(options.middguard_view_name)
          > -1)
          return;

        // add the view to the model's 'middguard_views'
        currentViews.push(middguard_view_name);
        model.set('middguard_views', currentViews);
      });
    }
  }
});

```

static/js/visualization-manager.js

```

1  var middguard = middguard || {};
2
3  (function () {
4
5      middguard.View = Backbone.View.extend({
6          className: 'middguard-module',
7          fetch: function (collection, options) {
8              // set the view name to add to the middguard_views when
9              // we create/update
10             // the models
11             options.middguard_view_name = this.cid;
12
13             // add the entity to this view
14             // so we can check the entities and remove the view
15             // from middguard_views
16             // when the view is destroyed
17             if (this.middguard_entities.indexOf(collection) < 0) {
18                 this.middguard_entities.push(collection);
19             }
20
21             middguard.entities[collection].fetch(options);
22         },
23
24         /* middguard.View.prototype.remove
25         * Extend the view remove function to remove referenced
26         * models
27         * Important: If you need to extend remove functionality,
28         * you must call
29         * super call instead
30         * of the usual 'Backbone.View.prototype.remove.call(this
31         * )\'
32         */
33         remove: function () {
34             var viewName = this.cid;
35
36             console.log('About to remove view "' + viewName + '"');
37
38             // For each model this view references
39             this.middguard_entities.forEach(function (entityName) {
40                 var collection = middguard.entities[entityName];
41
42                 // First iteration to remove reference to this model
43                 collection.each(function (model, i) {
44                     if (model.get('middguard_views').indexOf(viewName)
45                         > -1) {
46                         removeFromArray(model.get('middguard_views'),
47                             viewName);
48                     }
49                 });
50
51                 // Get an array of models from this entity collection
52                 // to remove
53                 var toRemove = collection.filter(function (model) {
54                     if (model.get('middguard_views').length === 0) {
55                         delete model.attributes.middguard_views;
56                         return true;
57                     }
58                 });
59
60                 console.log('Removing ' + toRemove.length +
61                     ' models that are no longer in use from
62                     collection "' +
63                     entityName + '"');
64                 // remove them without sending anything to the server
65                 collection.remove(toRemove);
66
67                 console.log('Done removing view "' + viewName + '"');
68
69                 // call super
70                 Backbone.View.prototype.remove.call(this);
71             });
72
73             createContext: function () {
74                 var moduleName = this.model.get('module')
75                 module = middguard.PackagedModules.findWhere({name:
76                     moduleName}),
77                 connections = JSON.parse(this.model.get('

```

```

70         connections'),
71         context = {};
72         context.inputs = _.reduce(_.keys(connections), function
73         (inputs, inputGroup) {
74             var groupConnections = connections[inputGroup].
75             connections,
76             outputNode = middguard.Nodes.get(connections[
77             inputGroup].output_node);
78
79             var columns = _.reduce(groupConnections, function (
80             connections, pair) {
81                 connections[pair.input] = pair.output;
82                 return connections;
83             }, {});
84             inputs[inputGroup] = {};
85             inputs[inputGroup].collection = middguard.entities[
86             outputNode.get('table')];
87             inputs[inputGroup].cols = columns;
88             inputs[inputGroup].tableName = outputNode.get('table'
89             );
90             return inputs;
91         }, {});
92         return context;
93     }
94     middguard.activateView = function (node) {
95         var main = middguard.Nodes.get(node).module().get('main'
96         );
97         var ctor = middguard.__modules[main].ctor;
98         var live = new ctor({model: middguard.Nodes.get(node)});
99         middguard.__modules[node] = {};
100         middguard.__modules[node].live = live;
101         $('.'+middguard-views').append(live.render().el);
102     };
103
104     middguard.deactivateView = function (node) {
105         middguard.__modules[node].live.remove();
106         middguard.__modules[node].live = null;
107     };
108
109     middguard.toggleView = function (node) {
110         if (middguard.__modules[node] && middguard.__modules[node
111         ].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 view.
126     * Top level views are listed under "Modules" in the
127     * sidebar.
128     */
129     middguard.addModule = function (name, view) {
130         _addView(name, view, true /* top level */);
131     };
132
133     /* middguard.addSubview
134     * Makes MiddGuard aware of a subview (a view instantiated
135     * from another view)
136     * Subviews are not listed in the sidebar, but have models
137     * they fetch tracked
138     * and removed when the view is removed.
139     */
140     middguard.addSubview = function (name, view) {
141         _addView(name, view, false /* not top level */);
142     };

```

```

140 var _addView = function (name, view, topLevel) {
141   if (!Object.prototype.hasOwnProperty.call(middguard,
142     __modules, name)) {
143     view.prototype.middguard_view_name = name;
144     view.prototype.middguard_entities = [];
145
146     if (topLevel) {
147       middguard.__modules[name] = {ctor: view, live: null};
148     } else {
149       middguard.__submodules[name] = {ctor: view, live:
150         null};
151     }
152     throw new Error('Module ' + name + ' already loaded');
153   }
154
155   /* Remove elements from an array.
156   * arr is the array to remove from (param 0).
157   * Elements to remove are arguments 1 .. n.
158   * Source: http://stackoverflow.com/questions/3954438
159   */
160   function removeFromArray(arr) {
161     var what, a = arguments, L = a.length, ax;
162     while (L > 1 && arr.length) {
163       what = a[--L];
164       while ((ax= arr.indexOf(what)) !== -1) {
165         arr.splice(ax, 1);
166       }
167     }
168     return arr;
169   }
170 }());

```

```

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, inputGroup) {
27      middguard.socket.emit('node:connect', {
28        outputNode: other.get('id'),
29        inputNode: this.get('id'),
30        inputGroup: inputGroup
31      });
32    },
33
34    run: function () {
35      middguard.socket.emit('node:run', {
36        id: this.get('id')
37      });
38    },
39
40    position: function (x, y) {

```

```

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

```

static/js/models/packaged-module.js

```

1 var middguard = middguard || {};
2
3 (function () {
4   middguard.PackagedModule = Backbone.Model.extend({
5     defaults: {
6       '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 createGraph: function(e) {
40     e.preventDefault();
41     var name = this.$('#new-graph-name').val().trim();
42     middguard.Graphs.create({name: name}, {wait: true});
43     this.$('#new-graph-name').val('');
44 }
45 });
46
47
48 var GraphView = Backbone.View.extend({
49     className: 'middguard-graph list-group-item',
50     tagName: 'a',
51     template: _.template('<%= name %>'),
52     events: {
53         'click': 'toggleEditor'
54     },
55     initialize: function() {
56         this.editing = false;
57         this.listenTo(this.model, 'update', this.render);
58     },
59     render: function() {
60         this.$el.html(this.template(this.model.toJSON()));
61         this.$el.attr('href', '#');
62         return this;
63     },
64     toggleEditor: function() {
65         if (this.editor) {
66             this.editor.remove();
67             this.editor = null;
68         } else {
69             this.editor = new middguard.GraphEditorView({graph:
70                 this.model});
71         }
72     }
73 });
74
75
76
77
78

```

```

79     $('#middguard-views').append(this.editor.render().el)
80     ;
81 }
82 this.$el.toggleClass('active', Boolean(this.editor));
83 }
84 });
85
86
87 })();

```

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

```

1  var middguard = middguard || {};
2
3  (function() {
4    'use strict';
5
6    middguard.GraphEditorView = Backbone.View.extend({
7      className: 'middguard-graph-editor middguard-module',
8
9      tagName: 'div',
10
11     template: _.template($('#graph-editor-template').html()),
12
13     initialize: function(options) {
14       this.graph = options.graph;
15       this.detailView = null;
16
17       this.listenTo(middguard.PackagedModules, 'reset', this.
         addModules);
18       this.listenTo(middguard.Nodes, 'reset', this.
         addAllNodes);
19       this.listenTo(middguard.Nodes, 'reset', this.
         addAllConnectorGroups);
20       this.listenTo(middguard.Nodes, 'reset', this.
         ensureEntityCollections);
21       this.listenTo(middguard.Nodes, 'add', this.addNode);
22       this.listenTo(middguard.Nodes, 'add', this.
         addConnectorGroup);
23
24       middguard.PackagedModules.fetch({reset: true, data:
         {}});
25       middguard.Nodes.fetch({reset: true, data: {}});
26     },
27
28     render: function() {
29       this.$el.html(this.template(this.graph.toJSON()));
30       d3.select(this.el).select('.editor').append('svg')
         .attr('class', 'graph')
         .attr('width', 500);
31
32       this.resizeEditor();
33
34       addNode: function(node) {
35         return this;
36       },
37
38       ensureEntityCollections: function() {
39         middguard.Nodes.each(this.ensureEntityCollection, this)
40       };
41
42       ensureEntityCollection: function(node) {
43         var tableName = node.get('table');
44
45         if (!tableName || middguard.entities[tableName]) return
46           ;
47
48         var collection = new middguard.EntityCollection([], {
49           url: tableName
50         });
51
52         middguard.entities[tableName] = collection;
53       },
54
55       resizeEditor: function() {
56         d3.select(this.el).select('.editor svg')
57           .attr('height', $(window).height() - this.$('.
             header').outerHeight());
58       },
59
60       addModules: function() {
61         this.$('.modules-list').html('');
62
63         middguard.PackagedModules.each(function(model) {
64           var view = new ModuleListItemView({model: model,
             graph: this.graph});
65           this.$('.modules-list').append(view.render().el);
66         }.bind(this));
67
68         this.resizeEditor();
69       },
70
71       addNode: function(node) {

```

```

72   if (node.get('graph_id') !== this.graph.get('id')) {
73       return;
74   }
75
76   var view = new NodeView({model: node, editor: this});
77   this.$('.graph').append(view.render().el);
78   },
79
80   addAllNodes: function(node) {
81       middleware.Nodes.each(this.addNode, this);
82   },
83
84   addConnectorGroup: function(node) {
85       if (node.get('graph_id') !== this.graph.get('id')) {
86           return;
87       }
88
89       var view = new ConnectorGroupView({model: node});
90       this.$('.graph').append(view.render().el);
91   },
92
93   addAllConnectorGroups: function() {
94       middleware.Nodes.each(this.addConnectorGroup, this);
95   },
96
97   setDetailView: function(view) {
98       if (this.detailView) {
99           this.detailView.remove();
100      }
101
102      this.$('.detail').html(view.render().el);
103      this.detailView = view;
104  }
105  });
106
107  var ModuleListItemView = Backbone.View.extend({
108      tagName: 'li',
109      className: 'btn btn-default module',
110
111      template: _.template('<%= displayName %>'),
112
113      events: {
114          'click': 'createNode',
115      },
116
117      initialize: function(options) {
118          this.model = options.model;
119          this.graph = options.graph;
120      },
121
122      render: function() {
123          this.$el.html(this.template(this.model.toJSON()));
124          return this;
125      },
126
127      createNode: function() {
128          middleware.Nodes.create({
129              module: this.model.get('name'),
130              graph_id: this.graph.get('id')
131          });
132      },
133  }
134  });
135
136  /* Nodes' connections are stored on the input node.
137   * All the connecting lines from an a node's connections
138   * to the corresponding output node.
139   */
140  var ConnectorGroupView = Backbone.NSView.extend({
141      tagName: 'svg:g',
142
143      initialize: function() {
144          this.connections = [];
145
146          if (this.model.get('connections'))
147              this.addAllConnectingLines();
148
149          // 'this.model' is the "input" node
150          this.listenTo(this.model, 'change', this.render);
151      },
152
153      render: function() {

```

```

154 this.connections.forEach(connection => connection.
155   render());
156 this.unrenderedConnections().forEach(this.
157   addConnectingLine, this);
158   return this;
159 }},
160 addAllConnectingLines: function () {
161   _.chain(JSON.parse(this.model.get('connections'))))
162     .keys()
163     .each(this.addConnectingLine, this);
164 },
165 addConnectingLine: function (inputGroup) {
166   var view = new ConnectorView({
167     model: this.model,
168     inputGroup: inputGroup
169   });
170   this.$el.append(view.render().el);
171   this.connections.push(view);
172 },
173 },
174
175 renderedConnections: function () {
176   return this.connections.map(connection => connection.
177     inputGroup);
178 },
179 unrenderedConnections: function () {
180   return _.chain(JSON.parse(this.model.get('connections')
181     ))
182     .keys()
183     .difference(this.renderedConnections());
184   });
185
186   var ConnectorView = Backbone.NSView.extend({
187     tagName: 'svg:path',
188     className: 'connecting-line',
189
190     initialize: function (options) {
191       this.model = options.model;
192       this.inputGroup = options.inputGroup;
193       this.outputNode = middguard.Nodes.findWhere({
194         id: JSON.parse(this.model.get('connections'))[this.
195           inputGroup].output_node
196       });
197       this.module = middguard.PackagedModules.findWhere({
198         name: this.model.get('module')
199       });
200
201       this.diagonal = d3.svg.diagonal();
202
203       // Only rerender this particular line when the output
204       // node moves.
205       // We rerender all the lines (parent view) when the
206       // input node moves.
207       this.listenTo(this.outputNode, 'change', this.render);
208
209       // Check if the connection has changed. In this context
210       // "changed"
211       // means that the connection's input group either no
212       // longer has a
213       // connection, or the input group is connected to a
214       // different output.
215       this.listenTo(this.model, 'change', this.
216         connectionChanged);
217     },
218     render: function () {
219       this.diagonal
220         .source(this.outputPosition())
221         .target(this.inputPosition());
222
223       this.$el.attr('d', this.diagonal());
224
225       return this;
226     },
227     inputPosition: function () {
228       var i = _.findIndex(this.module.get('inputs'), input =>

```

```

225     {
226         return input.name === this.inputGroup;
227     },
228     r = this.model.get('radius'),
229     n = this.module.get('inputs').length,
230     offset = NodeView.prototype.inputPosition(i, r, n);
231
232     return {
233         x: this.model.position().x + offset.x,
234         y: this.model.position().y + offset.y
235     };
236
237     outputPosition: function() {
238         var r = this.outputNode.get('radius');
239
240         return {
241             x: this.outputNode.position().x + r,
242             y: this.outputNode.position().y + 2 * r - 10
243         };
244     },
245
246     connectionChanged: function() {
247         var connections = this.model.get('connections'),
248             connection = JSON.parse(connections)[this.
249                 inputGroup];
250
251         // No longer a connection for this input group
252         if (!connection) {
253             this.remove();
254         }
255
256         // A connection exists for this input group, but
257         // connected to a
258         // different output node
259         if (connection.output_node !== this.outputNode.get('id')) {
260             // Stop listening to changes in the old output node
261             this.stopListening(this.outputNode);
262
263             // Find and bind to the new output node
264
265             this.outputNode = middleware.Nodes.get(connection.
266                 output_node);
267             this.listenTo(this.outputNode, 'change', this.render)
268         };
269     },
270     this.render();
271 }
272
273 var NodeView = Backbone.NSView.extend({
274     tagName: 'svg:g',
275     className: 'node',
276     events: {
277         'mouseover .input': 'showInputTooltip',
278         'mouseout .input': 'hideInputTooltip',
279         'click .input': 'toggleInputSelected',
280         'click .output': 'toggleOutputSelected',
281         'click .run': 'runNode',
282         'click': 'toggleDetail'
283     },
284
285     initialize: function(options) {
286         this.editor = options.editor;
287         this.model = options.model;
288         this.module = middleware.PackagedModules.findWhere({
289             name: this.model.get('module')
290         });
291
292         this.d3el = d3.select(this.el)
293             .datum(this.model.position());
294
295         this.drag = d3.behavior.drag({
296             .origin(function(d) { return d; })
297             .on('dragstart', this.dragstarted.bind(this))
298             .on('drag', this.dragged.bind(this))
299             .on('dragend', this.dragended.bind(this));
300
301         this.listenTo(this.model, 'change', this.render);
302     },

```



```

301 template: _.template($('#graph-node-template').html()),
302
303
304 render: function() {
305     var x = this.model.position().x;
306     var y = this.model.position().y;
307
308     this.d3el
309         .datum(this.model.position())
310         .attr('transform', 'translate(' + x + ', ' + y + ')')
311         .call(this.drag);
312
313     this.$el.html(this.template({
314         r: this.model.get('radius'),
315         handlePosition: this.dragHandlePosition(),
316         dragHandlePath: d3.svg.symbol().type('cross').size
317             (150)(),
318         runPosition: this.runPosition(),
319         runPath: d3.svg.symbol().type('triangle-up').size
320             (150)(),
321         status: this.model.get('status'),
322         statusText: this.model.statusText(),
323         displayName: this.module.get('displayName'),
324         inputs: this.module.get('inputs'),
325         output: this.module.get('outputs').length,
326         inputPosition: this.inputPosition
327     }));
328
329     var selectedInput = this.model.get('selectedInput'),
330         selectedOutput = this.model.get('selectedOutput');
331     if (selectedInput)
332         this.d3el.select('[data-name="' + selectedInput.name
333             + '"]')
334             .classed('selected', true);
335
336     if (selectedOutput)
337         this.d3el.select('.output')
338             .classed('selected', true);
339
340     if (this.model.isVisualization())
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375

```

```

338     this.d3el.classed('visualization', true);
339
340
341     return this;
342 },
343
344 dragstarted: function(d) {
345     this.dragStartPosition = _.clone(d);
346 },
347
348 dragged: function(d) {
349     if (!d3.select(d3.event.sourceEvent.target).classed('
350         drag-handle'))
351         return;
352     var x = d3.event.x;
353     var y = d3.event.y;
354     var r = this.model.get('radius');
355
356     var svg = d3.select(this.editor.el).select('svg');
357     var bounds = {x: svg.attr('width'), y: svg.attr('height
358         ')});
359
360     // Prevent element from being dragged out bounds
361     if (x < 0) x = 0;
362     if (y < 0) y = 0;
363     if (y + r * 2 > bounds.y) y = bounds.y - r * 2;
364     if (x + r * 2 > bounds.x) x = bounds.x - r * 2;
365
366     this.model.position(x, y);
367     d3.select(this.el)
368         .attr('transform', 'translate(' + (d.x = x) + ', ' +
369             (d.y = y) + ')');
370 },
371
372 dragended: function() {
373     if (this.dragMoved())
374         this.model.save();
375 },
376
377 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
383 showInputTooltip: function (event) {
384   var tooltip = d3.select('.input-tooltip');
385
386   if (!tooltip[0][0])
387     tooltip = d3.select('body').append('div')
388       .attr('class', 'input-tooltip');
389
390   var input = _.find(this.module.get('inputs'), function (
391     input) {
392     return input.name === $(event.currentTarget).data('
393       name');
394   });
395   tooltip.html(input.name);
396
397   var bounds = event.currentTarget.getBoundingClientRect
398     (),
399     inputRadius = 5,
400     tooltipWidth = parseFloat(tooltip.style('width')) /
401       2,
402     tooltipHeight = parseFloat(tooltip.style('height'))
403       + 5;
404
405   tooltip
406     .style('left', bounds.left - tooltipWidth +
407       inputRadius + 'px')
408     .style('top', bounds.top - tooltipHeight + 'px')
409     .style('visibility', 'visible');
410 },
411
412 hideInputTooltip: function () {
413   d3.select('.input-tooltip')
414     .style('visibility', 'hidden');
415 },
416
417 toggleInputSelected: function (event) {
418   var previouslySelected = middguard.Nodes.find(function (
419     node) {
420     return node.get('selectedInput');
421   });
422
423   // Deselect the previously selected input.
424   previouslySelected && previouslySelected.set('
425     selectedInput', null);
426
427   var selectedGroup = _.find(this.module.get('inputs'),
428     function (input) {
429     return input.name === $(event.target).data('name');
430   });
431
432   // If the clicked node was already selected, return
433   // after toggling it off.
434   if (previouslySelected &&
435     this.model.get('id') === previouslySelected.get('id
436       ') &&
437     selectedGroup.name === previouslySelected.get('name
438       ')) {
439     return;
440   }
441
442   this.model.set('selectedInput', selectedGroup);
443   this.connectNodes();
444 },
445
446 toggleOutputSelected: function (event) {
447   var previouslySelected = middguard.Nodes.find(function (
448     node) {
449     return node.get('selectedOutput');
450   });
451
452   previouslySelected && previouslySelected.set('
453     selectedOutput', null);
454
455   this.model.set('selectedOutput', true);
456   this.connectNodes();
457 },

```

```

444 connectNodes: function () {
445     var input = middguard.Nodes.find(function (node) {
446         return node.get('selectedInput');
447     });
448     var output = middguard.Nodes.find(function (node) {
449         return node.get('selectedOutput');
450     });
451     if (!input || !output)
452         return;
453     var group = input.get('selectedInput').name;
454     input.connectToOutput (output, group);
455     input.set('selectedInput', null);
456     output.set('selectedOutput', null);
457 },
458
459 runNode: function () {
460     if (this.model.isVisualization()) {
461         middguard.toggleView(this.model.get('id'));
462     } else {
463         this.model.run();
464     }
465 },
466
467 toggleDetail: function () {
468     var view = new NodeDetailView({model: this.model});
469     this.editor.$('.node').removeClass('selected');
470     this.$el.addClass('selected');
471     this.editor.setDetailView(view);
472 },
473
474 dragHandlePosition: function () {
475     var r = this.model.get('radius');
476     return {
477         x: r + -r * Math.sqrt(2) / 2 + 15,
478         y: r - r * Math.sqrt(2) / 2 + 15
479     };
480 },
481
482 runPosition: function () {
483     var r = this.model.get('radius');
484     return {
485         x: r + r * Math.sqrt(2) / 2 - 15,
486         y: r - r * Math.sqrt(2) / 2 + 15
487     };
488 },
489
490 /* Calculate each input circle's position.
491 * Circles are arranged in rows of three from the top
492 * down.
493 * Assume 5 pixel circle radius and 15 pixels spacing
494 * between
495 * circle centerpoints. Circles are centered around the
496 * node's centerline.
497
498 * Example: 5 inputs (x is an input circle)
499 * x <--15px--> x <--15px--> x
500 * (15px between rows)
501 * x <-- 15px --> x
502
503 * @param i: input index
504 * @param r: the input parent node's radius
505 * @param n: total number of inputs for the node
506
507 * @return the center position for the input circle
508 */
509 inputPosition: function (i, r, n) {
510     var rowIndexX = i % 3,
511         rowIndexY = Math.floor(i / 3),
512         rowLength = i >= n - n % 3 ? n % 3 : 3,
513         baseX = r - (rowLength - 1) * 7.5,
514         baseY = 10;
515     return {
516         x: baseX + 15 * rowIndexX,
517         y: baseY + 15 * rowIndexY
518     };
519 }

```

```

523     };
524   }
525   });
526
527   var NodeDetailView = Backbone.View.extend({
528     initialize: function() {
529       this.connections = JSON.parse(this.model.get('
530         connections'));
531       this.module = this.model.module();
532
533       this.selectedInputGroup = null;
534       this.selectedOutput = null;
535       this.selectedInput = null;
536
537       this.listenTo(this.model, 'change', this.render);
538     },
539     template: _.template(
540       '<h4><%= name %></h4>
541       <div class="connection-groups"><div>',
542     ),
543     connectionGroupTemplate: _.template($('#connection-group-
544       template').html()),
545     events: {
546       'click .connection': 'selectConnector',
547     },
548     render: function() {
549       this.$el.html(this.template({
550         name: this.model.get('displayName')
551       }));
552     },
553     this.addAllConnectionGroups();
554
555     return this;
556   },
557
558   addAllConnectionGroups: function() {
559     _.each(this.connections, (value, key) => {
560       var inputs = value.connections.map(connection =>
561         {
562           connection.input,
563           outputs = value.connections.map(connection =>
564             connection.output),
565           outputNode = middleware.Nodes.get(value.
566             output_node),
567           outputModule = middleware.PackagedModules.
568             findWhere({
569               name: outputNode.get('module')
570             });
571
572           this.$('.connection-groups').prepend(this.
573             connectionGroupTemplate({
574               inputGroupName: key,
575               inputs: inputs,
576               unconnectedInputs: this.model.unconnectedInputs(key
577                 ),
578               outputModuleName: outputModule.get('displayName'),
579               outputs: outputs,
580               unconnectedOutputs: this.model.unconnectedOutputs(
581                 key)
582             }));
583
584           deselectOutput: function() {
585             this.selectedOutput = null;
586             this.$('.connection.output').removeClass('selected');
587           },
588
589           deselectInput: function() {
590             this.selectedInput = null;
591             this.$('.connection.input').removeClass('selected');
592           },
593
594           selectConnector: function(event) {
595             var $clicked = $(event.target),
596             group = $clicked.closest('.connection-list-group').
597               data('inputgroup'),
598             name = $clicked.text(),
599             isInput = $clicked.hasClass('input'),
600             isOutput = $clicked.hasClass('output'),

```

```

595 sameGroup = this.selectedInputGroup === group;
596
597 if (isInput) {
598   if (sameGroup) this.deselectInput();
599   else this.deselectOutput();
600
601   this.selectedInput = name;
602 }
603
604 if (isOutput) {
605   if (sameGroup) this.deselectOutput();
606   else this.deselectInput();
607
608   this.selectedOutput = name;
609 }
610
611 this.selectedInputGroup = group;
612 $clicked.addClass('selected');
613 this.connectSelection();
614 },
615
616 connectSelection: function () {
617   if (!this.selectedInputGroup ||
618     !this.selectedInput ||
619     !this.selectedOutput) {

```

```

620         return;
621     }
622
623     var connections = this.connections[this.
        selectedInputGroup].connections;
624
625     var exists = _.find(connections, {input: this.
        selectedInput}) ||
        _.find(connections, {output: this.
            selectedOutput});
626
627
628     if (exists) {
629         exists.input = this.selectedInput;
630         exists.output = this.selectedOutput;
631     } else {
632         connections.push({
633             input: this.selectedInput,
634             output: this.selectedOutput
635         });
636     }
637
638     this.connections[this.selectedInputGroup].connections =
        connections;
639
640     this.deselectInput();
641     this.deselectOutput();
642     this.selectedInputGroup = null;
643
644     this.model.set('connections', JSON.stringify(this.
        connections));
645     this.model.save();
646 }
647 });
648 })();

```

APPENDIX B

CHAPTER 2 OF APPENDIX

examples/simple/index.js

```
1 var middleware = require('../..');
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 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
```

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 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 = "@DanaRSilver";
16
17 var client = new Twitter({
18   consumer_key: 'fEYwq7R6fP7np546j799dMXJj',
19   consumer_secret: '5
    pAk01rSEZ1hrhbnRG6pJdcQIYkKmtIFNPvsyzV8jjyhSnOc1',
20   access_token_key: '354037431-
    sd7fd6inZSXWaw9LmC3gmFfaHWx6p8UJq8JUaPDM',
21   access_token_secret: '
    B8clfzqPuJqUWKnSGTsEpV3eF1Y35RiIw7HI6YiMSoles'
22 });
23
24 client = Promise.promisifyAll(client);
25
26 exports.handle = function(context) {
27   var params = {screen_name: 'DanaRSilver', 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 };
49
50 exports.inputs = [];
51
52 exports.outputs = [
53   'handle',
54   'tweet',
55   'timestamp'
56 ];
57
58 exports.displayName = "@jack";
59
60 var client = new Twitter({
61   consumer_key: 'fEYwq7R6fP7np546j799dMXJj',
62   consumer_secret: '5
63     pAk01rSEZ1hrhbnRG6pJdcQIYkKmtIFNPvsyzV8jjyuhSnOC1',
64   access_token_key: '354037431-
65     sd7fd6inZSXWaw9LmC3gmFfaHWx6p8UJq8JUaPDM',
66   access_token_secret: '
67     B8clfzqPuJqUWKnSGTsEpV3eFlY35RiIw7HI6YiMSoles'
68 });
69
70 client = Promise.promisifyAll(client);
71
72 exports.handle = function(context) {
73   var params = {screen_name: 'jack', count: 200};
74   return client.getAsync('statuses/user_timeline', params)
75     .spread(function(tweets, response) {
76       tweets = tweets.map(function(tweet) {
77         return {
78           handle: tweet.user.screen_name,
79           tweet: tweet.text,
80           timestamp: new Date(tweet.created_at)
81         };
82       });
83     });
84 }

```

examples/simple/aggregate-time/index.js

```

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   };

```

```

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) {

```

examples/simple/difference/index.js

```

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 };

```

```

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('*'), tweets2.knex.
     select('*'),
35     function(tweets1, tweets2) {
36       _.range(24).forEach(function(hour) {
37         _.range(7).forEach(function(day) {
38           var count1 = _.find(tweets1, {hour: hour, day: day}).
             count;

```

```

39   var count2 = _.find(tweets2, {hour: hour, day: day}).
40     count;
41   week.push({
42     day: day,
43     hour: hour,
44     count1: count1,
45     count2: count2,
46     difference: Math.abs(count1 - count2)
47   });
48   });
49
50   return context.table.knex.insert(week);
51   });
52 };

```

examples/simple/hours-heatmap/index.js

```

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';

```

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.hours.tableName;
29       this.listenTo(this.context.inputs.hours.collection, '
30         reset', this.render);
31
32       this.fetch(tableName, {reset: true, data: {}});
33     },
34
35     render: function() {
36       this.$el.html(this.template());
37       this.$el.css('position', 'relative');
38
39       var data = this.context.inputs.hours.collection.map(
40         function(hours) {
41
42           return _.clone(hours.attributes);
43         });
44
45       var margin = {top: 0, left: 90, right: 0, bottom: 20};
46
47       var rowHeight = 60,
48         height = 7 * rowHeight - margin.top - margin.bottom
49         ,
50         colWidth = 60,
51         width = colWidth * 24 - margin.left - margin.right;
52
53       var week = ["Sunday", "Monday", "Tuesday", "Wednesday",
54         "Thursday", "Friday", "Saturday"];
55
56       var x = this.x = d3.scale.linear()
57         .domain([0, 23])
58         .range([colWidth / 2, width - colWidth / 2]);
59
60       var y = this.y = d3.scale.linear()
61         .domain([0, 6])
62         .range([rowHeight / 2, height - rowHeight / 2]);
63
64       var xAxis = d3.svg.axis()
65         .scale(x)
66         .ticks(24)
67         .orient('bottom');
68
69       var yAxis = d3.svg.axis()
70         .scale(y)
71         .orient('left')
72         .ticks(6)
73         .tickFormat(function(d) {
74           return week[d];
75         });
76
77       var size = this.size = d3.scale.sqrt()
78         .domain([0, d3.max(data, function(d) { return Math.
79           max(d.count1, d.count2); })])
80         .range([0, 25]);
81
82       var svg = d3.select(this.el).select('svg')[0][0]

```

```

77         ? d3.select(this.el).select('svg')
78         : d3.select(this.el).append('svg');
79
80     svg = svg
81     .attr('width', width + margin.left + margin.right)
82     .attr('height', height + margin.top + margin.bottom
83         )
84     .append('g')
85     .attr('transform', 'translate(' + margin.left + ', '
86         + margin.top + ')')
87
88     var circles = svg
89     .selectAll('g.dayhour')
90     .data(data)
91     .enter().append('g')
92     .attr('class', 'dayhour')
93     .attr('transform', function(d, i) {
94         return 'translate(' + x(d.hour) + ', ' + y(d.day)
95         + ')';
96     });
97
98     circles.append('circle')
99     .attr('r', function(d) { return size(d.count1); })
100     .style('fill', 'transparent')
101     .style('stroke-width', 2)
102     .style('stroke', 'orange');
103
104     circles.append('circle')
105     .attr('r', function(d) { return size(d.count2); })
106     .style('fill', 'transparent')
107     .style('stroke-width', 2)
108     .style('stroke', 'steelblue');
109
110     svg.append('g')
111     .attr('class', 'x axis')
112     .attr('transform', 'translate(0, ' + height + ')')
113     .call(xAxis);
114
115     ? return this;
116     },
117
118     showInputTooltip: function(event) {
119         var tooltip = d3.select('heatmap-tooltip');
120
121         var d = d3.select(event.target).datum();
122         tooltip.select('.count1').text(d.count1);
123         tooltip.select('.count2').text(d.count2);
124
125         var bounds = event.currentTarget.getBoundingClientRect
126         (),
127         inputRadius = 5,
128         tooltipWidth = parseFloat(tooltip.style('width')) /
129         2,
130         tooltipHeight = parseFloat(tooltip.style('height'))
131         + 5;
132
133         tooltip
134         .style('left', (this.x(d.hour) + 65) + 'px')
135         .style('top', (this.y(d.day) - this.size(Math.max(d.
136             count1, d.count2)) - 10) + 'px')
137         .style('visibility', 'visible');
138
139         hideInputTooltip: function() {
140             d3.select('heatmap-tooltip')
141             .style('visibility', 'hidden');
142         }
143
144         middguard.addModule('HoursHeatmapView', HoursHeatmapView);
145     }());

```

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>
13         ><%- col3 %></td><td><%- col4 %></td></tr></th><
14       ><%- col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
15       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
16       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
17       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
18       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
19       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
20       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
21       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
22       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
23       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
24       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
25       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
26       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
27       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
28       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
29       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
30       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
31       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
32       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
33       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
34       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
35       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
36       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
37       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-
38       col3 %></td><td><%- col4 %></td></tr></tr></td><td><%-

```



```

39         return this;
40     },
41
42     addAllRows: function () {
43         var collection = this.context.inputs.table.collection;
44         collection.each(this.addRow, this);
45     },
46
47     addOneRow: function (row) {
48         var cols = this.context.inputs.table.cols;
49         console.log(cols, row)
50
51         this.$el.append(this.rowTemplate({
52             col1: row.get(cols.col1),
53             col2: row.get(cols.col2),
54             col3: row.get(cols.col3),
55             col4: row.get(cols.col4)
56         }));
57     },
58
59     },
60 });
61
62 middguard.PeekTableView = PeekTableView;
63 middguard.addModule('PeekTableView', PeekTableView);
64 }) ();

```

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.