

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	A Investigation with MiddGuard	42
B	Core Code from the MiddGuard Framework	54
	Bibliography	87

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.

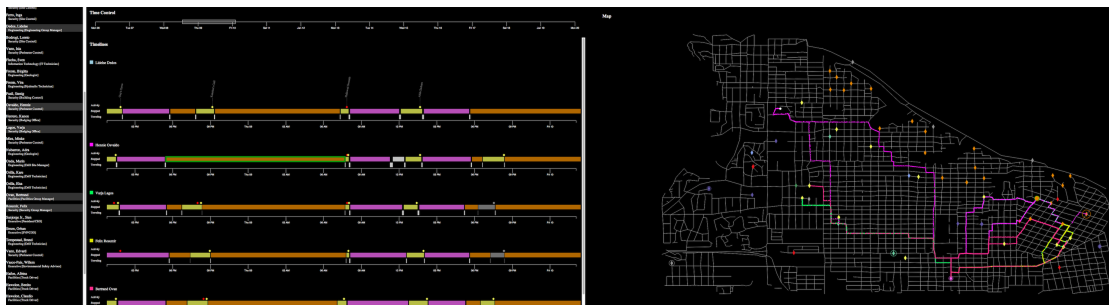


Figure 1.1: The web interface for Andrews and Silver’s VAST 2014 entry. The visualizations, from left to right, are a list of people, a master brushable timeline and individual timelines for each person listed with selectable events, and a map of GPS traces from the individuals’ cars.

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 overlayed 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.

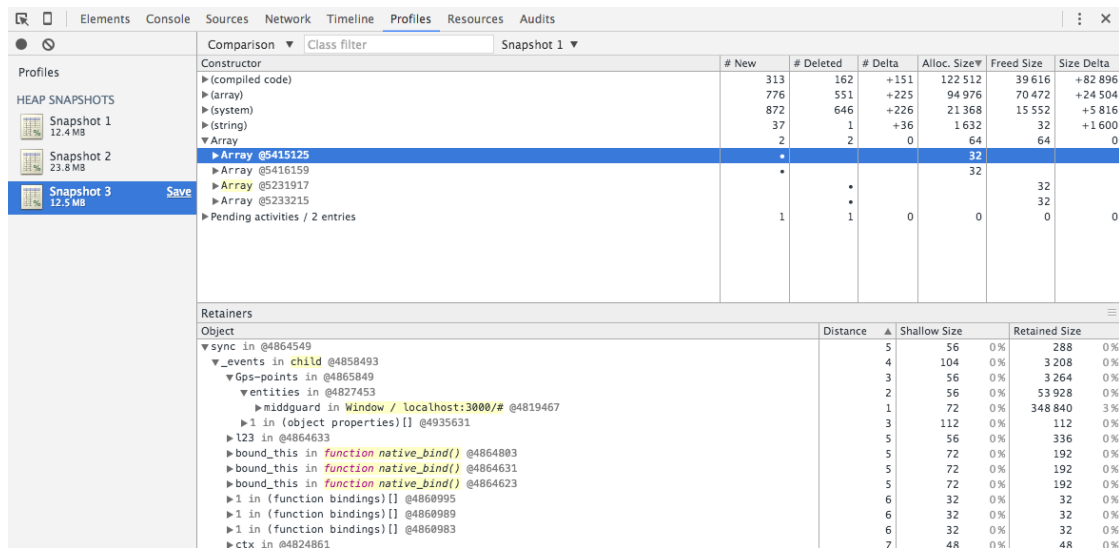


Figure 1.2: A screen capture of the Google Chrome DevTools Profiler demonstrating the efficacy of View Reference Counting. The panel on the left shows three snapshots. Snapshot 1 was taken before a view was added. Snapshot 2 was taken after a view was added and rendered with a significant amount of data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

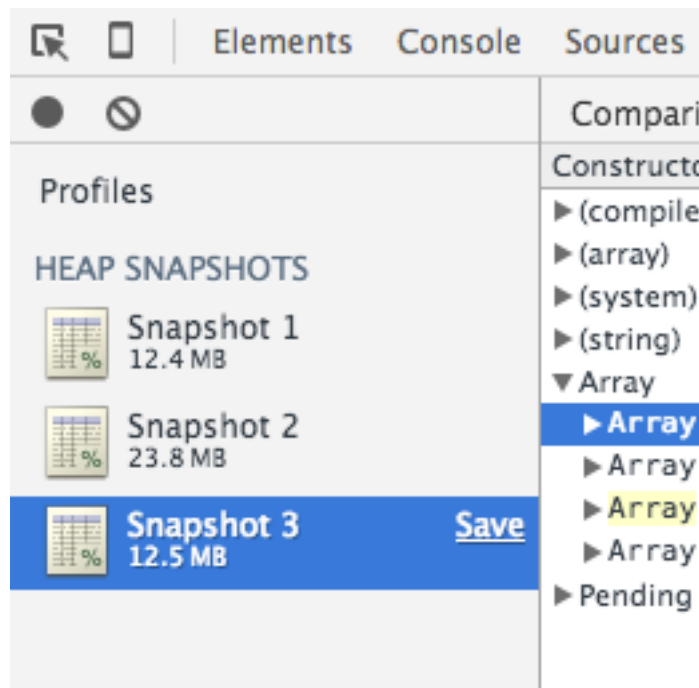


Figure 1.3: The snapshots portion of figure 1.2, cropped for readability.

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.

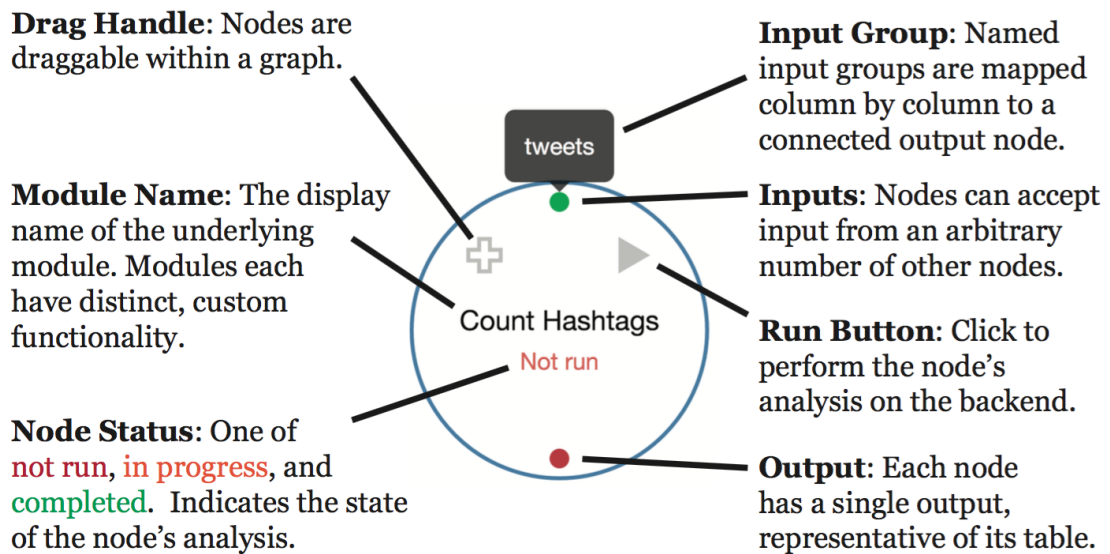


Figure 4.4: An annotated analytic node taken from the MiddGuard graph editor. Important features are annotated and the node's only input group, "tweets", is moused over to show its accompanying tooltip.

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

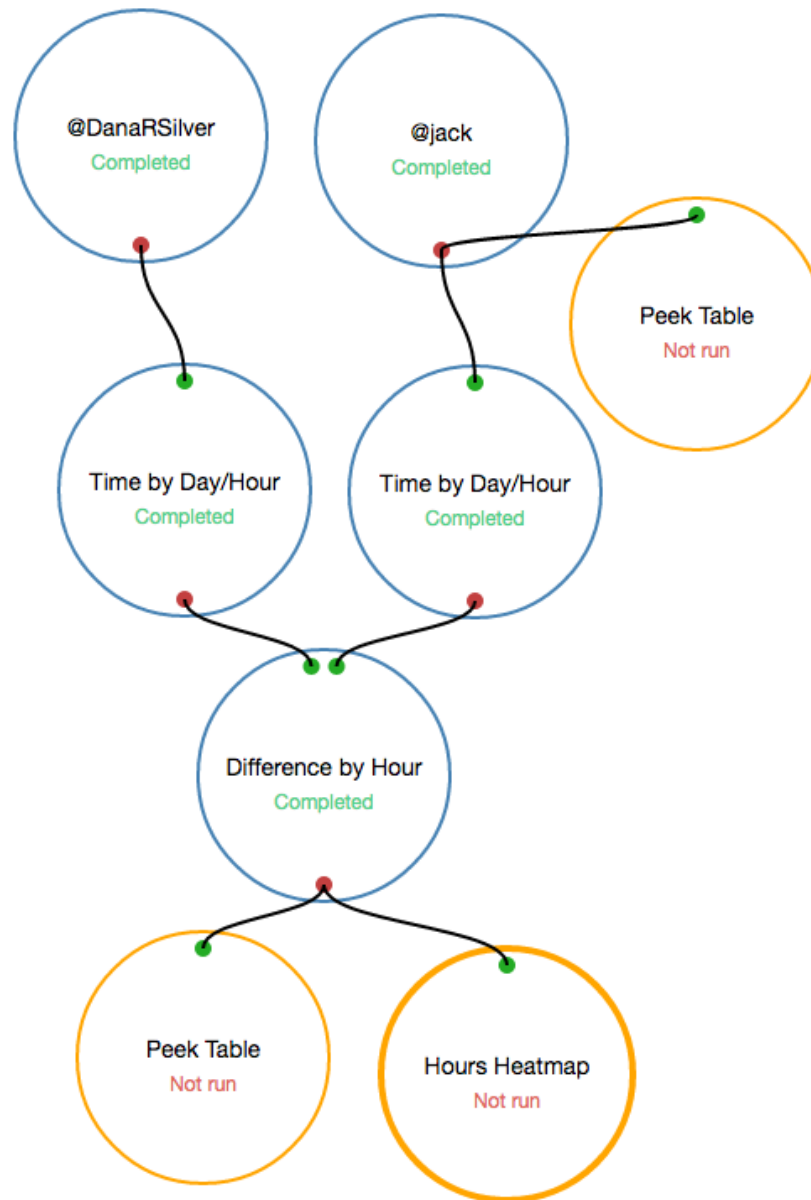


Figure 5.1: The complete graph from the mock investigation used to develop and test MiddGuard.

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

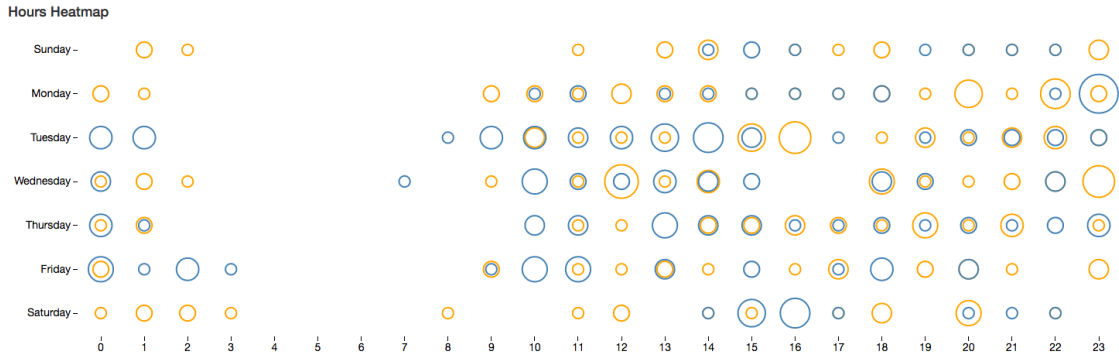


Figure 5.2: The “Hours Heatmap” visualization from the mock investigation used to develop and test MiddGuard. @DanaRSilver’s tweets are orange, @jack’s are blue. Circle radii are mapped to the number of times each person tweeted in that hour and day of the week.

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

A INVESTIGATION WITH MIDDGUARD

The code in Appendix A implements the mock investigation described in the use case. At the top level are *index.js* and *knexfile.js*, which setup the web server and configure the database connection respectively. The other ten files implement the analytic and visualization modules that we used to conduct the investigation. The analytic modules each consist of a single file, an *index.js* in a subdirectory of the project. The visualization modules include an *index.js* to configure the module, and a JavaScript and CSS file each to render their visualizations in the browser.

examples/simple/index.js

```
1 var middguard = 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
24 var port = process.env.PORT || 3000;
25 app.listen(port, function () {
26   console.log('Listening on port %d...', port);
27 });
```

examples/simple/knexfile.js

```

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

```

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

```

1 var Promise = require('bluebird');
2 var fs = Promise.promisifyAll(require('fs'));
3 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
      pAk0lrSEZlhrhbnRG6pJdcQIYkKMTIFNPvsyzV8jyyuhSnOC1',
20   access_token_key: '354037431-
      sd7fd6inZSXWaw9ImC3gmFfaHWx6p8UJq8JUaPDM',
21   access_token_secret: '
      B8clFzqPuJqUWKnSGTSEpV3eFlY35RiIw7HI6YiMSOles'
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   )
30   .spread(function(tweets, response) {
31     tweets = tweets.map(function(tweet) {
32       return {
33         handle: tweet.user.screen_name,
34         tweet: tweet.text,
35         timestamp: new Date(tweet.created_at)
36       };
37     });
38   });
39 }

```


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

```

37
38   return context.table.knex.insert(tweets);
39 });
40 };
41
42 exports.createTable = function (tableName, knex) {
43   return knex.schema.createTable(tableName, function (
44     table) {
45     table.string('handle');
46     table.string('tweet');
47     table.dateTime('timestamp');
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: 'fEYwq7R6fP7np546j799dMXj',
62     consumer_secret: '5
63       pAk0lrSEZlhrhbnRG6pJdcQIYkKMcIFNPvsyzV8jjyuhSnOCl',
64     access_token_key: '354037431-
65       sd7fd6inZSXWaw9ImC3gmFahWx6p8UJq8JUaPDM',
66     access_token_secret: '
67       B8clfzqPuJqUWKnSGTspV3eFlY35RiIw7HI6YiMSOles'
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   )
76   .spread(function (tweets, response) {
77     tweets = tweets.map(function (tweet) {
78       return {
79         handle: tweet.user.screen_name,
80         tweet: tweet.text,
81         timestamp: new Date(tweet.created_at)

```

37 } ;

45

examples/simple/difference/index.js

```

38 return tweets.knex.select('*')
39 .then(function(tweets) {
40   tweets.forEach(function(tweet) {
41     var m = moment(tweet[timestampColl]),
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(
21     table) {
22     table.integer('day');
23     table.integer('hour');
24     table.integer('count1');
25     table.integer('count2');
26     table.integer('difference');
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.
35       knex.select('*'),
36       function(tweets1, tweets2) {
37         _.range(24).forEach(function(hour) {
38           _.range(7).forEach(function(day) {
39             var count1 = _.find(tweets1, {hour: hour, day:

```

```

39     day}).count;
    var count2 = _.find(tweets2, {hour: hour, day:
40     day}).count;
    week.push({
41     day: day,
42     hour: hour,
43     count1: count1,
44     count2: count2,
45     difference: Math.abs(count1 - count2)
46     });
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       ;
30       this..listenTo(this.context.inputs.hours.collection,
31         'reset', this.render);
32
33       this.fetch(tableName, {reset: true, data: {}});
34     },
35
36     render: function () {
37       this.$el.html(this.template());
38       this.$el.css('position', 'relative');
39
40       var data = this.context.inputs.hours.collection.map
41
42       (function (hours) {
43         return _.clone(hours.attributes);
44       });
45
46       var margin = {top: 0, left: 90, right: 0, bottom:
47         20};
48
49       var rowHeight = 60,
50         height = 7 * rowHeight - margin.top - margin.
51         bottom,
52         colWidth = 60,
53         width = colWidth * 24 - margin.left - margin.
54         right;
55
56       var week = ["Sunday", "Monday", "Tuesday", "
57         Wednesday", "Thursday", "Friday", "Saturday"];
58
59       var x = this.x = d3.scale.linear()
60         .domain([0, 23])
61         .range([colWidth / 2, width - colWidth / 2]);
62
63       var y = this.y = d3.scale.linear()
64         .domain([0, 6])
65         .range([rowHeight / 2, height - rowHeight / 2])
66         ;
67
68       var xAxis = d3.svg.axis()
69         .scale(x)
70         .ticks(24)
71         .orient('bottom');
72
73       var yAxis = d3.svg.axis()
74         .scale(y)
75         .orient('left')
76         .ticks(6)
77         .tickFormat(function (d) {
78           return week[d];
79         });
80
81       var size = this.size = d3.scale.sqrt()
82         .domain([0, d3.max(data, function (d) { return

```

```

74         Math.max(d.count1, d.count2); }]])
75         .range([0, 25]);
76
77     var svg = d3.select(this.el).select('svg')[0][0]
78     ? d3.select(this.el).select('svg')
79     : d3.select(this.el).append('svg');
80
81     svg = svg
82     .attr('width', width + margin.left + margin.
83         right)
84     .attr('height', height + margin.top + margin.
85         bottom)
86     .append('g')
87     .attr('transform', 'translate(' + margin.left +
88         ', ' + margin.top + ')')
89
90     var circles = svg
91     .selectAll('g.dayhour')
92     .data(data)
93     .enter().append('g')
94     .attr('class', 'dayhour')
95     .attr('transform', function(d, i) {
96         return 'translate(' + x(d.hour) + ', ' + y(d.
97             day) + ')';
98     });
99
100     circles.append('circle')
101     .attr('r', function(d) { return size(d.count1);
102     })
103     .style('fill', 'transparent')
104     .style('stroke-width', 2)
105     .style('stroke', 'orange');
106
107     circles.append('circle')
108     .attr('r', function(d) { return size(d.count2);
109     })
110     .style('fill', 'transparent')
111     .style('stroke-width', 2)
112     .style('stroke', 'steelblue');
113
114     showInputTooltip: function(event) {
115         var tooltip = d3.select('.heatmap-tooltip');
116
117         var d = d3.select(event.target).datum();
118         tooltip.select('.count1').text(d.count1);
119         tooltip.select('.count2').text(d.count2);
120
121         var bounds = event.currentTarget.
122             getBoundingClientRect(),
123             inputRadius = 5,
124             tooltipWidth = parseFloat(tooltip.style('width'
125                 )) / 2,
126             tooltipHeight = parseFloat(tooltip.style('
127                 height')) + 5;
128
129         tooltip
130         .style('left', (this.x(d.hour) + 65) + 'px')
131         .style('top', (this.y(d.day) - this.size(Math.max
132             (d.count1, d.count2)) - 10) + 'px')
133         .style('visibility', 'visible');
134     },
135     },
136
137     hideInputTooltip: function() {
138         d3.select('.heatmap-tooltip')
139         .style('visibility', 'hidden');
140     }
141     });
142
143     middguard.addModule('HoursHeatmapView',

```

```

HoursHeatmapView);
144 })();

```

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 middleware = middleware || {};
2
3 (function() {
4   var PeekTableView = middleware.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><%- coll %></td><td><%- col2 %></td><td><%-
13         td><%- col3 %></td><td><%- col4 %></td></tr></tr></td><td><%- col4 %></td></tr></td><td><%-
14         th>'
15       ),
16
17     rowTemplate: _.template(
18       '<tr><td><%- coll %></td><td><%- col2 %></td><td><%-
19         ><%- col3 %></td><td><%- col4 %></td></tr>'
20       ),
21
22     initialize: function() {
23       this.context = this.createContext();
24
25       var collection = this.context.inputs.table.collection;
26
27       var tableName = this.context.inputs.table.tableName;
28
29       this.listenTo(collection, 'reset', this.addAllRows);
30
31       this.fetch(tableName, {reset: true, data: {}});
32
33     },
34
35     render: function() {
36       var cols = this.context.inputs.table.cols;
37
38       this.$el.html(this.template({
39         coll: cols.coll,

```

```

35 col2: cols.col2,
36 col3: cols.col3,
37 col4: cols.col4
38 ));
39
40 return this;
41 },
42
43 addAllRows: function () {
44   var collection = this.context.inputs.table.
         collection;
45   collection.each(this.addOneRow, this);
46 },
47
48 addOneRow: function (row) {
49   var cols = this.context.inputs.table.cols;
50
51   console.log(cols, row)
52
53   this.$el.append(this.rowTemplate({
54     col1: row.get(cols.col1),
55     col2: row.get(cols.col2),
56     col3: row.get(cols.col3),
57     col4: row.get(cols.col4)
58   }));
59 }
60 });
61
62 middleware.PeekTableView = PeekTableView;
63 middleware.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 }
```

APPENDIX B

CORE CODE FROM THE MIDDGUARD FRAMEWORK

The code in Appendix B is the core of MiddGuard’s data-flow model and front-end visualization loading/unloading. Additional code, style sheets, and HTML templates that make up complete MiddGuard framework have been omitted from this listing.

index.js, *middguard/application.js*, and *middguard/middguard.js* implement the web server from which all MiddGuard investigations (including that in Appendix A) are instantiated. Code in *middguard/socket* implements server side of MiddGuard’s WebSocket protocol. *middguard/models* and *middguard/migrations* contain models and schema to persist graphs and data to a database. *static* contains all front-end code for MiddGuard’s client-side environment.

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')(
18   session);
19
20 var _ = require('lodash');
21
22 /**
23  * Application prototype methods to extend
24  * the express application prototype.
25  */
26
27 var app = exports = module.exports = {};
28
29 app.middguardInit = function () {
30   this.middguardExpressMiddleware();
31 }
```

```

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('session'));
74
75   this.set('views', path.join(__dirname, 'views'));
76   this.set('view engine', 'jade');
77
78   app.middlewareSocketMiddleware = function
79     middlewareSocketMiddleware() {
80     var io = this.get('io');
81     var session = this.get('session');
82
83     io.use(io.use(session));
84
85     io.use((socket, next) => {
86       socket.bookshelf = this.get('bookshelf');
87       next();
88     });
89
90     app.middlewareExpressMiddleware = function
91       middlewareExpressMiddleware() {
92       this.use('/static', express.static(path.join(__dirname,
93         '..', 'static')));
94
95       var knex = require('knex')(this.get('knex config'));
96       var sessionStore = new KnexSessionStore({knex: knex});
97       this.set('sessionStore', sessionStore);
98
99       // Set up ORM middleware
100       require('./config/bookshelf')(this);
101
102       this.use(cookieParser(this.get('secret key')));
103       this.use(bodyParser.urlencoded({extended: true}));
104       this.use(bodyParser.json());
105
106       this.set('session', session({
107         store: sessionStore,
108         secret: this.get('secret key'),
109         resave: true,
110         saveUninitialized: true,
111         cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
112       }));
113
114       this.use(this.get('session'));
115
116       this.set('views', path.join(__dirname, 'views'));
117       this.set('view engine', 'jade');
118
119       app.middlewareSocketMiddleware = function
120         middlewareSocketMiddleware() {
121         var io = this.get('io');
122         var session = this.get('session');
123
124         io.use(io.use(session));
125
126         io.use((socket, next) => {
127           socket.bookshelf = this.get('bookshelf');
128           next();
129         });
130
131         app.middlewareExpressMiddleware = function
132           middlewareExpressMiddleware() {
133           this.use('/static', express.static(path.join(__dirname,
134             '..', 'static')));
135
136           var knex = require('knex')(this.get('knex config'));
137           var sessionStore = new KnexSessionStore({knex: knex});
138           this.set('sessionStore', sessionStore);
139
140           // Set up ORM middleware
141           require('./config/bookshelf')(this);
142
143           this.use(cookieParser(this.get('secret key')));
144           this.use(bodyParser.urlencoded({extended: true}));
145           this.use(bodyParser.json());
146
147           this.set('session', session({
148             store: sessionStore,
149             secret: this.get('secret key'),
150             resave: true,
151             saveUninitialized: true,
152             cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
153           }));
154
155           this.use(this.get('session'));
156
157           this.set('views', path.join(__dirname, 'views'));
158           this.set('view engine', 'jade');
159
160           app.middlewareSocketMiddleware = function
161             middlewareSocketMiddleware() {
162             var io = this.get('io');
163             var session = this.get('session');
164
165             io.use(io.use(session));
166
167             io.use((socket, next) => {
168               socket.bookshelf = this.get('bookshelf');
169               next();
170             });
171
172             app.middlewareExpressMiddleware = function
173               middlewareExpressMiddleware() {
174               this.use('/static', express.static(path.join(__dirname,
175                 '..', 'static')));
176
177               var knex = require('knex')(this.get('knex config'));
178               var sessionStore = new KnexSessionStore({knex: knex});
179               this.set('sessionStore', sessionStore);
180
181               // Set up ORM middleware
182               require('./config/bookshelf')(this);
183
184               this.use(cookieParser(this.get('secret key')));
185               this.use(bodyParser.urlencoded({extended: true}));
186               this.use(bodyParser.json());
187
188               this.set('session', session({
189                 store: sessionStore,
190                 secret: this.get('secret key'),
191                 resave: true,
192                 saveUninitialized: true,
193                 cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
194               }));
195
196               this.use(this.get('session'));
197
198               this.set('views', path.join(__dirname, 'views'));
199               this.set('view engine', 'jade');
200
201               app.middlewareSocketMiddleware = function
202                 middlewareSocketMiddleware() {
203                 var io = this.get('io');
204                 var session = this.get('session');
205
206                 io.use(io.use(session));
207
208                 io.use((socket, next) => {
209                   socket.bookshelf = this.get('bookshelf');
210                   next();
211                 });
212
213                 app.middlewareExpressMiddleware = function
214                   middlewareExpressMiddleware() {
215                   this.use('/static', express.static(path.join(__dirname,
216                     '..', 'static')));
217
218                   var knex = require('knex')(this.get('knex config'));
219                   var sessionStore = new KnexSessionStore({knex: knex});
220                   this.set('sessionStore', sessionStore);
221
222                   // Set up ORM middleware
223                   require('./config/bookshelf')(this);
224
225                   this.use(cookieParser(this.get('secret key')));
226                   this.use(bodyParser.urlencoded({extended: true}));
227                   this.use(bodyParser.json());
228
229                   this.set('session', session({
230                     store: sessionStore,
231                     secret: this.get('secret key'),
232                     resave: true,
233                     saveUninitialized: true,
234                     cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
235                   }));
236
237                   this.use(this.get('session'));
238
239                   this.set('views', path.join(__dirname, 'views'));
240                   this.set('view engine', 'jade');
241
242                   app.middlewareSocketMiddleware = function
243                     middlewareSocketMiddleware() {
244                     var io = this.get('io');
245                     var session = this.get('session');
246
247                     io.use(io.use(session));
248
249                     io.use((socket, next) => {
250                       socket.bookshelf = this.get('bookshelf');
251                       next();
252                     });
253
254                     app.middlewareExpressMiddleware = function
255                       middlewareExpressMiddleware() {
256                       this.use('/static', express.static(path.join(__dirname,
257                         '..', 'static')));
258
259                       var knex = require('knex')(this.get('knex config'));
260                       var sessionStore = new KnexSessionStore({knex: knex});
261                       this.set('sessionStore', sessionStore);
262
263                       // Set up ORM middleware
264                       require('./config/bookshelf')(this);
265
266                       this.use(cookieParser(this.get('secret key')));
267                       this.use(bodyParser.urlencoded({extended: true}));
268                       this.use(bodyParser.json());
269
270                       this.set('session', session({
271                         store: sessionStore,
272                         secret: this.get('secret key'),
273                         resave: true,
274                         saveUninitialized: true,
275                         cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
276                       }));
277
278                       this.use(this.get('session'));
279
280                       this.set('views', path.join(__dirname, 'views'));
281                       this.set('view engine', 'jade');
282
283                       app.middlewareSocketMiddleware = function
284                         middlewareSocketMiddleware() {
285                         var io = this.get('io');
286                         var session = this.get('session');
287
288                         io.use(io.use(session));
289
290                         io.use((socket, next) => {
291                           socket.bookshelf = this.get('bookshelf');
292                           next();
293                         });
294
295                         app.middlewareExpressMiddleware = function
296                           middlewareExpressMiddleware() {
297                           this.use('/static', express.static(path.join(__dirname,
298                             '..', 'static')));
299
300                           var knex = require('knex')(this.get('knex config'));
301                           var sessionStore = new KnexSessionStore({knex: knex});
302                           this.set('sessionStore', sessionStore);
303
304                           // Set up ORM middleware
305                           require('./config/bookshelf')(this);
306
307                           this.use(cookieParser(this.get('secret key')));
308                           this.use(bodyParser.urlencoded({extended: true}));
309                           this.use(bodyParser.json());
310
311                           this.set('session', session({
312                             store: sessionStore,
313                             secret: this.get('secret key'),
314                             resave: true,
315                             saveUninitialized: true,
316                             cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
317                           }));
318
319                           this.use(this.get('session'));
320
321                           this.set('views', path.join(__dirname, 'views'));
322                           this.set('view engine', 'jade');
323
324                           app.middlewareSocketMiddleware = function
325                             middlewareSocketMiddleware() {
326                             var io = this.get('io');
327                             var session = this.get('session');
328
329                             io.use(io.use(session));
330
331                             io.use((socket, next) => {
332                               socket.bookshelf = this.get('bookshelf');
333                               next();
334                             });
335
336                             app.middlewareExpressMiddleware = function
337                               middlewareExpressMiddleware() {
338                               this.use('/static', express.static(path.join(__dirname,
339                                 '..', 'static')));
340
341                               var knex = require('knex')(this.get('knex config'));
342                               var sessionStore = new KnexSessionStore({knex: knex});
343                               this.set('sessionStore', sessionStore);
344
345                               // Set up ORM middleware
346                               require('./config/bookshelf')(this);
347
348                               this.use(cookieParser(this.get('secret key')));
349                               this.use(bodyParser.urlencoded({extended: true}));
350                               this.use(bodyParser.json());
351
352                               this.set('session', session({
353                                 store: sessionStore,
354                                 secret: this.get('secret key'),
355                                 resave: true,
356                                 saveUninitialized: true,
357                                 cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
358                               }));
359
360                               this.use(this.get('session'));
361
362                               this.set('views', path.join(__dirname, 'views'));
363                               this.set('view engine', 'jade');
364
365                               app.middlewareSocketMiddleware = function
366                                middlewareSocketMiddleware() {
367                                var io = this.get('io');
368                                var session = this.get('session');
369
370                                io.use(io.use(session));
371
372                                io.use((socket, next) => {
373                                  socket.bookshelf = this.get('bookshelf');
374                                  next();
375                                });
376
377                                app.middlewareExpressMiddleware = function
378                                  middlewareExpressMiddleware() {
379                                  this.use('/static', express.static(path.join(__dirname,
380                                    '..', 'static')));
381
382                                  var knex = require('knex')(this.get('knex config'));
383                                  var sessionStore = new KnexSessionStore({knex: knex});
384                                  this.set('sessionStore', sessionStore);
385
386                                  // Set up ORM middleware
387                                  require('./config/bookshelf')(this);
388
389                                  this.use(cookieParser(this.get('secret key')));
390                                  this.use(bodyParser.urlencoded({extended: true}));
391                                  this.use(bodyParser.json());
392
393                                  this.set('session', session({
394                                    store: sessionStore,
395                                    secret: this.get('secret key'),
396                                    resave: true,
397                                    saveUninitialized: true,
398                                    cookie: {maxAge: 7 * 24 * 60 * 60 * 1000} // 1 week
399                                  }));
400
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

```

```

107 requirePath: requirePath,
108 displayName: attributes.displayName,
109 inputs: attributes.inputs,
110 outputs: attributes.outputs,
111 visualization: attributes.visualization,
112 main: attributes.visualization ? attributes.mainView
    : null
113 }));
114 };
115
116 /**
117  * Listen for connections.
118  *
119  * A node 'http.Server' is returned, with this
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 };

```

middguard/middguard.js

```

1 'use strict';
2
3 /**
4  * Module dependencies.
5  */
6
7 var _ = require('lodash');
8 var express = require('express');
9 var proto = require('./application');
10
11 /**
12  * Expose `createApplication()`.
13  */
14
15 exports = module.exports = createApplication;
16
17 /**
18  * Create a middguard application.
19  *
20  * @return {Function}
21  * @public
22  */
23
24 function createApplication(settings) {
25   var app = express();
26
27   _.each(settings, (value, key) => {
28     app.set(key, value);
29   });
30
31   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')(
47 // // app);
48 // sessionSockets.on('connection', require('./middguard
49 // /socket'));
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.
17         create(socket, data, cb));
18
19     socket.on('messages:read', (data, cb) => message.
20         readAll(socket, data, cb));
21
22     socket.on('modules:read', (data, cb) => modules.readAll
23         (socket, data, cb));
24
25     socket.on('analyst:read', (data, cb) => analyst.read(
26         socket, data, cb));
27
28     socket.on('analysts:read', (data, cb) => analyst.
29         readAll(socket, data, cb));
30
31     socket.on('node:connect', (data, cb) => node.connect(
32         socket, data, cb));
33
34     socket.on('node:run', (data, cb) => node.run(socket,
35         data, cb));
36
37     socket.on('nodes:create', (data, cb) => node.create(
38         socket, data, cb));
39
40     socket.on('nodes:read', (data, cb) => node.readAll(
41         socket, data, cb));
42
43     socket.on('nodes:update', (data, cb) => node.update(
44         socket, data, cb));
45
46     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
37 // Set up sockets to call analytics from client
38 // Patched models will automatically emit create,
    update, and delete events
39 // Bookshelf.collection('analytics').each(function (
    analyticsAttrs) {
40 //   var name = analyticsAttrs.get('name');
41 //   var requirePath = analyticsAttrs.get('requirePath
    ');
42 //
43 //   socket.on('analytics:' + name, function (data,
    callback) {
44 //     require(requirePath)(Bookshelf, data);
45 //   });
46 // });
47 };
48
49 function patchModelToEmit(socket, modelName, model) {
50   if (!model.prototype._emitting) {
51     var _initialize = model.prototype.initialize;
52
53     model.prototype.initialize = function () {
54       var args = Array.prototype.slice.call(arguments);
55       _initialize.apply(args);
56
57       this.on('created', function (model, attrs, options)
        {
58         // If the model was created on the client, we don
        't want to emit a
59         // create event, since we need to assign an id on
        the creator via
60         // a callback and do a broadcast.emit for
        everyone else.
61         // The create listener will take care of this.
62         if (!options.clientCreate) {

```

```

63   io.emit(pluralize(modelName) + ':create', model
    .toJSON());
64   } else {
65     socket.broadcast.emit(pluralize(modelName) + ':
    create', model.toJSON());
66   }
67 });
68
69 this.on('updated', function (model) {
70   socket.broadcast.emit(pluralize(modelName) + ':
    update', model.toJSON());
71 });
72
73 this.on('destroying', function (model) {
74   socket.broadcast.emit(pluralize(modelName) + ':
    delete', model.toJSON());
75 });
76 };
77
78 model.prototype._emitting = true;
79 }
80 }
81
82 function setupSocketEvents(socket, modelName, model) {
83   // Set up create, read, update, delete sockets for each
    model
84   socket.on(pluralize(modelName) + ':create', function (
    data, callback) {
85     // Pass clientCreate to save so the model won't emit
    anything on the
86     // created event and confuse the client.
87     // Create is a special case since the model on the
    creating client doesn't
88     // have an id yet.
89     new model().save(data, {clientCreate: true})
90     .then(function (newModel) {
91       callback(null, newModel.toJSON());
92     })
93     .catch(function (error) {
94       throw new Error(error);
95     })

```

```

96 });
97
98 socket.on(modelName + ':update', function (data,
99   callback) {
100   new model({id: _.result(data, 'id')}}
101     .save(_.omit(data, 'id'), {patch: true});
102 });
103
104 socket.on(modelName + ':delete', function (data,
105   callback) {
106   var x = new model({id: _.result(data, 'id')});
107   //console.log(String(x.destroy().
108     _resolveFromSyncValue));
109   new model({id: _.result(data, 'id')}).destroy();
110 });
111
112 socket.on(pluralize(modelName) + ':read', function (
113   data, callback) {
114   if (data) {
115     var fetchData = new model().fetchAll();
116   } else {
117     //if fetching all models at once
118     var fetchData = new model().fetchAll();
119   }
120   fetchData
121     .then(function (collection) {
122       callback(null, collection.toJSON());
123     })
124     .catch(function (error) {
125       callback(error);
126     });
127 });
128
129 socket.on(modelName + ':read', function (data, callback
130   ) {
131   new model({id: _.result(data, 'id')}).fetch({require:
132     true})
133     .then(function (fetchedModel) {
134       callback(null, fetchedModel.toJSON());
135     })
136     .catch(model.NotFoundError, function () {
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

```


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       )
25     .catch(callback);
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.

```

```

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:
51         outputNode.get('module')});
52       var inputModule = modules.findWhere({name: inputNode.
53         get('module')});
54
55       // Get outputs list from the corresponding output
56       module
57       outputs = require(outputModule.get('requirePath'))
58         ).outputs;
59
60       // Get inputs list from the corresponding input
61       module
62       inputGroups = require(inputModule.get('
63         requirePath')).inputs;
64
65       var inputs = inputGroups.filter(function(group) {
66         return group.name === data.inputGroup;
67       })[0].inputs;
68
69       // The array of connections we'll set on the input
70       node
71       connections = {
72         output_node: data.outputNode,
73         connections: [
74
75         ];
76
77       if (data.connections &&
78         validateConnections(data.connections, inputs,
79         outputs)) {
80
81         // Use 'data.connections' if the connections are
82         valid
83
84         connections.connections = data.connections;
85       } else {
86         // Match input and output names
87         connections.connections = connectionsByName(inputs,
88         outputs);
89       }
90
91       inputNode.setInputGroup(data.inputGroup, connections)
92       ;
93       return inputNode.save();
94     }).then(node => {
95       socket.emit('nodes:update', node.toJSON());
96       socket.broadcast.emit('nodes:update', node.toJSON());
97     })
98     .catch(callback);
99   };
100
101   // Validate that all potential inputs and outputs have
102   inputs and outputs on
103   * with the same name on the respective nodes.
104
105   *
106   * @private
107   * @param {Object[]} connections The passed in data of
108   connections to set.
109   * @param {Object[]} inputs Named inputs on the existing
110   input node.
111   * @param {Object[]} outputs Named outputs on the
112   existing output node.
113
114   */
115   function validateConnections(connections, inputs, outputs
116   ) {
117     var potentialInputs = connections.map(connection =>
118     connection.input);
119     var potentialOutputs = connections.map(connection =>
120     connection.output);
121
122     return potentialInputs.length === potentialOutputs.
123     length &&
124     inputs.every(input => _.has(potentialInputs,

```

```

102         input)) &&
103         outputs.every(output => _.has(potentialOutputs,
104             output));
105     }
106     /**
107      * Generate the connections array by matching names
108      * between inputs and outputs.
109      * Returns an array with size equivalent to the
110      * cardinality of inputs    outputs.
111      * @private
112      * @param {Object[]} inputs Inputs to match
113      * @param {Object[]} outputs Outputs to match
114      */
115     function connectionsByName(inputs, outputs) {
116         return outputs.filter(output => _.indexOf(inputs,
117             output) > -1)
118             .map(output => ({output: output, input:
119                 output}));
120     }
121
122     exports.run = function(socket, data, callback) {
123         var Node = socket.bookshelf.model('Node');
124         var modules = socket.bookshelf.collection('analytics');
125
126         new Node({id: data.id})
127             .fetch()
128             .tap(node => node.ensureTable())
129             .then(node => node.save({status: 1}))
130             .then(function(node) {
131                 socket.emit('nodes:update', node.toJSON());
132                 socket.broadcast.emit('nodes:update', node.toJSON());
133                 return node;
134             })
135             .then(node => Promise.join(node, node.outputNodes()))
136             .spread(function(node, outputs) {
137                 var module = modules.findWhere({name: node.get('
138                     module')}),
139                     connections = JSON.parse(node.get('connections'))
140                 ,
141
142                 context = {};
143
144                 context.inputs = _.reduce(_.keys(connections),
145                     function(inputs, inputGroup) {
146                         var groupConnections = connections[inputGroup].
147                             connections;
148
149                         // Reduce the array of input output pairs to a
150                         // single associative array
151                         // mapping input to output.
152                         var columns = _.reduce(groupConnections, function(
153                             connections, pair) {
154                             connections[pair.input] = pair.output;
155                             return connections;
156                         }, {});
157
158                         inputs[inputGroup] = {};
159                         inputs[inputGroup].knex = socket.bookshelf.knex(
160                             outputs[inputGroup].get('table'));
161                         inputs[inputGroup].cols = columns;
162                         inputs[inputGroup].tableName = outputs[inputGroup].
163                             get('table');
164
165                         return inputs;
166                     }, {});
167
168                 context.table = {};
169                 context.table.knex = socket.bookshelf.knex(node.get('
170                     table'));
171                 context.table.name = node.get('table');
172
173                 var handle = require(module.get('requirePath')).
174                     handle;
175                 return Promise.join(node, handle(context));
176             })
177             .spread(function(node, result) {
178                 return node.save({status: 2});
179             })
180             .then(function(node) {
181                 socket.emit('nodes:update', node.toJSON());
182                 socket.broadcast.emit('nodes:update', node.toJSON());
183             })

```

```

168 })
169 .catch(callback);
170 };

```

```

midguard/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 };

```

midguard/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  };

```

midguard/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', `${this.get('module')}_${`
43     {count + 1}`}`);
44 });
45
46 /**
47  * Get a mapping from input group names to output
48  *   nodes.
49  * @return a promise for an object mapping input
50  *   group 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].
59     output_node;
60     return new Node({id: outputId}).fetch()
61     .then(node => {
62       outputs[inputGroup] = node;
63       return outputs;
64     }, {});
65   }, {});
66 /**
67  * Create this node's table if it doesn't already.
68  */
69 ensureTable: function () {
70   return Bookshelf.knex.schema.hasTable(this.get('
71   table'))
72   .then(exists => {
73     if (!exists) {
74       return this.module().createTable(this.get('
75       table'), Bookshelf.knex);
76     }
77   });
78 }
79
80 var modules = Bookshelf.collection('analytics'),
81   moduleName = this.get('module'),
82   module = modules.findWhere({name: moduleName});
83
84 return require(module.get('requirePath'));
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   {};
99   groups[inputGroup] = connections;
100
101   return this.set('connections', JSON.stringify(
102   groups));
103 }
104
105 createReadSocket: function (socket) {
106   let table = Bookshelf.knex(this.get('table'));
107
108   socket.on(`${this.get('table')}:read`, (data,
109   callback) => {
110     if (!_.isEmpty(data)) {
111       var query = Bookshelf.knex(this.get('table')).
112       where(data).select('*');
113     } else {
114       var query = Bookshelf.knex(this.get('table')).

```

```

109         select('*');
110     }
111     query.then(results => callback(null, results));
112 });
113 }
114 });
115
116 return Bookshelf.model('Node', Node);
117 };

```

middguard/migrations/20140728124252:initial.js

```

1 'use strict';
2
3 exports.up = function(knex, Promise) {
4   return knex.schema.createTable('analyst', function(
5     table) {
6     table.increments('id').primary();
7     table.text('username').unique();
8     table.text('password');
9   });
10   .createTable('message', function(table) {
11     table.increments('id').primary();
12     table.integer('analyst_id').references('analyst.id');
13     table.text('state');
14     table.text('content');
15     table.dateTime('timestamp');
16   });
17   .createTable('graph', function(table) {
18     table.increments('id').primary();
19     table.string('name');
20   });
21   .createTable('node', function(table) {
22     table.increments('id').primary();
23     table.integer('graph_id').references('graph.id');
24     table.string('module');
25     table.string('table');
26     table.integer('status').defaultTo(0);
27     table.string('connections').defaultTo('{}');
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     ({
8       this.url = _.result(options, 'url');
9
10      _.bindAll(this, 'serverCreate', 'serverUpdate', 'serverDelete');
11
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
19      serverCreate: function(data) {
20        var exists = this.get(data.id);
21        if (!exists) {
22          this.add(data);
23        } else {
24          exists.set(data);
25        }
26      },
27      serverUpdate: function(data) {
28        var exists = this.get(data.id);
29        if (exists) exists.set(data);
30      },
31      serverDelete: function(data) {
32        // Already deleted from database, so don't need to
33        // model.destroy
34        var exists = this.get(data.id);
35        if (exists) this.remove(exists);
36      },
37      addViewReferences: function(collection, response, options) {
38        var middguard_view_name = options.

```


static/js/visualization-manager.js

```

36     middguard_view_name;
37     // if a view name wasn't passed in we can't do
38     // anything about it
39     if (!middguard_view_name)
40         return;
41     console.log('Adding view references for view "' +
42         middguard_view_name +
43         '" to ' + response.length + ' fetched
44         models.');
```

```

45     // get models added to the collection that match
46     // the criteria
47     // we fetched for
48     (options.data
49     ? collection.where(options.data)
50     : collection).forEach(function (model) {
51         var currentViews = model.get('middguard_views');
52         // if 'middguard_views' doesn't exist on the
53         // model, set it to an empty
54         // array
55         if (!currentViews)
56             model.set('middguard_views', []);
57         currentViews = model.get('middguard_views');
58         // if the view has already been added
59         if (currentViews.indexOf(options.
60             middguard_view_name) > -1)
61             return;
62         // add the view to the model's 'middguard_views'
63         currentViews.push(middguard_view_name);
64         model.set('middguard_views', currentViews);
65     });
66     }
67     });
68     });
```

```

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
9             // when 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) <
18                0) {
19                this.middguard_entities.push(collection);
20            }
21            middguard.entities[collection].fetch(options);
22        },
23        /* middguard.View.prototype.remove
24        * Extend the view remove function to remove
25        * referenced models
26        *
27        * Important: If you need to extend remove
28        * functionality, you must call
29        * 'middguard.View.prototype.remove.call(this)' as
30        * the super call instead
31        * of the usual 'Backbone.View.prototype.remove.call(
32        * this)'.
```

```

33        */
34        remove: function () {
35            var viewName = this.cid;
36            console.log('About to remove view "' + viewName + '
37                "'.');
```



```

96  var live = new ctor({model: middguard.Nodes.get(node)
97  });
98  middguard.__modules[node] = {};
99  middguard.__modules[node].live = live;
100
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[
111  node].live) {
112  middguard.deactivateView(node);
113  } else {
114  middguard.activateView(node);
115  };
116
117  // Internal hash of module views
118  middguard.__modules = {};
119
120  // Internal hash of submodule views
121  middguard.__submodules = {};
122
123  /* middguard.addModule
124  * Makes MiddGuard aware of a top level view.
125  * Top level views are listed under "Modules" in the
126  sidebar.
127  */
128  middguard.addModule = function (name, view) {
129  _addView(name, view, true /* top level */);
130  };
131
132  /* middguard.addSubview
133  * Makes MiddGuard aware of a subview (a view
134  instantiated from another view)
135  */
136  middguard.addSubview = function (name, view) {
137  _addView(name, view, false /* not top level */);
138  };
139
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:
148  null};
149  } else {
150  throw new Error('Module ' + name + ' already loaded
151  ');
152  }
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 })();

static/js/collections/graphs.js

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

```

static/js/collections/nodes.js

```

1 var middguard = middguard || {};
2
3 (function() {
4   'use strict';
5
6   var Nodes = middguard.BaseCollection.extend({
7     model: middguard.Node,
8
9     url: 'nodes',
10
11     initialize: function() {
12       _.bindAll(this, 'serverCreate', 'serverUpdate');
13
14       this.ioBind('create', this.serverCreate, this);
15       this.ioBind('update', this.serverUpdate, this);
16     },
17
18     serverCreate: function(data) {
19       var exists = this.get(data.id);
20       if (!exists) {
21         this.add(data);
22       } else {
23         exists.set(data);
24       }
25     },
26
27     serverUpdate: function(data) {
28       var exists = this.get(data.id);
29       if (exists) {
30         exists.set(data);
31       }
32     },
33   });
34
35   middguard.Nodes = new Nodes();
36 })();

```

static/js/collections/packaged-modules.js

```

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

```

static/js/models/graph.js

```

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

```

static/js/models/node.js

```

1 var middguard = middguard || {};
2
3 (function() {
4   middguard.Node = Backbone.Model.extend({
5     blacklistAttributes: [
6       'selectedInput',
7       'selectedOutput'
8     ],
9
10    defaults: {
11      status: 0,
12      radius: 75,
13      position_x: 0,
14      position_y: 0,
15      selectedInput: null,
16      selectedOutput: null,
17      connections: '{}',
18    },
19
20    statusMap: {
21      0: 'Not run',
22      1: 'In progress',
23      2: 'Completed'
24    },
25
26    connectToOutput: function(other, 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'))[inputGroup],
65       allInputs = _.find(this.module().get('inputs'), {name: inputGroup}).inputs;
66
67   if (!connections) {
68     return allInputs;
69   }
70
71   var connectedInputs = connections.connections.map(c => c.input);
72   return _.difference(allInputs, connectedInputs);
73 },
74
75 unconnectedOutputs: function(inputGroup) {
76   var connections = JSON.parse(this.get('connections'))[inputGroup];
77
78   if (!connections.output_node) {
79     return [];
80   }
81
82   var connectedOutputs = connections.connections.map(c => c.output);
83
84   var outputNode = middguard.Nodes.get(connections.output_node);
85   allOutputs = middguard.PackagedModules.find({name: outputNode.get('module')}).get('outputs');
86
87   return _.difference(allOutputs, connectedOutputs);
88 },
89
90 isVisualization: function() {
91   return this.module().get('visualization');
92 }
93
94 }
95
96 }());

```

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($('#graphs-panel-template').html()
10    ),
11
12    events: {
13      'click #create-new-graph': 'createGraph'
14    },
15
16    initialize: function () {
17      this.listenTo(middguard.Graphs, 'add', this.
18        addOneGraph);
19      this.listenTo(middguard.Graphs, 'reset', this.
20        addAllGraphs);
21
22      middguard.Graphs.fetch({reset: true, data: {}});
23    },
24
25    render: function () {
26      this.$el.html(this.template());
27
28      this.$graphs = this.$('.graphs-list');
29
30      return this;
31    },
32
33    addOneGraph: function (graph) {
34      var graphView = new GraphView({model: graph});
35
36      this.$graphs.append(graphView.render().el);
37    },
38
39    addAllGraphs: function () {
40      middguard.Graphs.each(this.addOneGraph, this);
41    }
42  });
43
44  middguard.Views.add(middguard.GraphsView);
45
46  return middguard.GraphsView;
47
48 })();

```



```

38 },
39
40 createGraph: function(e) {
41   e.preventDefault();
42   var name = this.$('#new-graph-name').val().trim();
43
44   middguard.Graphs.create({name: name}, {wait: true})
45   ;
46   this.$('#new-graph-name').val('');
47 }
48
49 var GraphView = Backbone.View.extend({
50   className: 'middguard-graph list-group-item',
51
52   tagName: 'a',
53
54   template: _.template('<%= name %>'),
55
56   events: {
57     'click': 'toggleEditor'
58   },
59
60   initialize: function() {
61     this.editing = false;
62
63     this.listenTo(this.model, 'update', this.render);
64   },
65
66   render: function() {
67     this.$el.html(this.template(this.model.toJSON()));
68
69     this.$el.attr('href', '#');
70     return this;
71   },
72
73   toggleEditor: function() {
74     if (this.editor) {
75       this.editor.remove();
76       this.editor = null;
77     } else {
78       this.editor = new middguard.GraphEditorView({
79         graph: this.model});
80       $('#middguard-views').append(this.editor.render().el);
81     }
82     this.$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($('#graph-editor-template').html()
12       ()),
13
14     initialize: function(options) {
15       this.graph = options.graph;
16       this.detailView = null;
17
18       this.listenTo(middguard.PackagedModules, 'reset',
19         this.addModules);
20
21       this.listenTo(middguard.Nodes, 'reset', this.
22         addAllNodes);
23
24       this.listenTo(middguard.Nodes, 'reset', this.
25         addAllConnectorGroups);
26
27       this.listenTo(middguard.Nodes, 'reset', this.
28         ensureEntityCollections);
29
30       this.listenTo(middguard.Nodes, 'add', this.addNode)
31       ;
32
33       this.listenTo(middguard.Nodes, 'add', this.
34         addConnectorGroup);
35
36       middguard.PackagedModules.fetch({reset: true, data:
37         {}});
38
39       middguard.Nodes.fetch({reset: true, data: {}});
40
41       render: function() {
42         this.$el.html(this.template(this.graph.toJSON()));
43         d3.select(this.el).select('.editor').append('svg')
44           .attr('class', 'graph')
45           .attr('width', 500);
46
47         this.resizeEditor();
48
49         return this;
50       },
51
52       ensureEntityCollections: function() {
53         middguard.Nodes.each(this.ensureEntityCollection,
54           this);
55       },
56
57       ensureEntityCollection: function(node) {
58         var tableName = node.get('table');
59
60         if (!tableName || middguard.entities[tableName])
61           return;
62
63         var collection = new middguard.EntityCollection([],
64           {
65             url: tableName
66           });
67
68         middguard.entities[tableName] = collection;
69
70         resizeEditor: function() {
71           d3.select(this.el).select('.editor svg')
72             .attr('height', $(window).height() - this.$('h1.
73               header').outerHeight());
74
75         },
76
77         addModules: function() {
78           this.$('.modules-list').html('');
79
80           middguard.PackagedModules.each(function(model) {
81             var view = new ModuleListItemView({model: model,
82               graph: this.graph});
83             this.$('.modules-list').append(view.render().el);
84             }.bind(this));
85
86           this.resizeEditor();
87
88         }
89       }
90     });
91   })();
92 }
93 
```

```

69 },
70
71 addNode: function (node) {
72   if (node.get('graph_id') !== this.graph.get('id'))
73     {
74       return;
75     }
76   var view = new NodeView({model: node, editor: this
77     this.$('.graph').append(view.render().el);
78   },
79
80   addAllNodes: function (node) {
81     middleware.Nodes.each(this.addNode, this);
82   },
83
84   addConnectorGroup: function (node) {
85     if (node.get('graph_id') !== this.graph.get('id'))
86       {
87         return;
88       }
89     var view = new ConnectorGroupView({model: node});
90     this.$('.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.$('.detail').html(view.render().el);
103     this.detailView = view;
104   }
105   });
106
107   var ModuleListItemView = Backbone.View.extend({
108     tagName: 'li',
109
110     className: 'btn btn-default module',
111
112     template: _.template('<%= displayName %>'),
113
114     events: {
115       'click': 'createNode',
116     },
117
118     initialize: function (options) {
119       this.model = options.model;
120       this.graph = options.graph;
121     },
122
123     render: function () {
124       this.$el.html(this.template(this.model.toJSON()));
125       return this;
126     },
127
128     createNode: function () {
129       middleware.Nodes.create({
130         module: this.model.get('name'),
131         graph_id: this.graph.get('id')
132       });
133     }
134   });
135
136   /* Nodes' connections are stored on the input node.
137    * All the connecting lines from an a node's
138    connections
139    * to the corresponding output node.
140    */
141   var ConnectorGroupView = Backbone.NSView.extend({
142     tagName: 'svg:g',
143
144     initialize: function () {
145       this.connections = [];
146
147       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
161 addAllConnectingLines: function () {
162     _.chain(JSON.parse(this.model.get('connections'))).
163         .keys()
164         .each(this.addConnectingLine, this);
165 }
166
167 addConnectingLine: function (inputGroup) {
168     var view = new ConnectorView({
169         model: this.model,
170         inputGroup: inputGroup
171     });
172     this.$el.append(view.render().el);
173     this.connections.push(view);
174 }
175
176 renderedConnections: function () {
177     return this.connections.map(connection =>
178         connection.inputGroup);
179 }
180
181 unrenderedConnections: function () {
182     return _.chain(JSON.parse(this.model.get('
183         connections'))).
184         .keys()
185         .difference(this.renderedConnections());
186 }
187
188 this.addAllConnectingLines();
189
190 // 'this.model' is the "input" node
191 this.listenTo(this.model, 'change', this.render);
192 }
193
194 render: function () {
195     this.connections.forEach(connection => connection.
196         render());
197     this.unrenderedConnections().forEach(this.
198         addConnectingLine, this);
199     return this;
200 }
201
202 addAllConnectingLines: function () {
203     _.chain(JSON.parse(this.model.get('connections'))).
204         .keys()
205         .each(this.addConnectingLine, this);
206 }
207
208 addConnectingLine: function (inputGroup) {
209     var view = new ConnectorView({
210         model: this.model,
211         inputGroup: inputGroup
212     });
213     this.$el.append(view.render().el);
214     this.connections.push(view);
215 }
216
217 renderedConnections: function () {
218     return this.connections.map(connection =>
219         connection.inputGroup);
220 }
221
222 unrenderedConnections: function () {
223     return _.chain(JSON.parse(this.model.get('
224         connections'))).
225         .keys()
226         .difference(this.renderedConnections());
227 }

```

```

217         this.$el.attr('d', this.diagonal());
218
219     return this;
220 },
221
222     inputPosition: function() {
223         var i = _.findIndex(this.model.get('inputs'),
224             input => {
225                 return input.name === this.inputGroup;
226             }),
227             r = this.model.get('radius'),
228             n = this.model.get('inputs').length,
229             offset = NodeView.prototype.inputPosition(i, r,
230                 n);
231
232         return {
233             x: this.model.position().x + offset.x,
234             y: this.model.position().y + offset.y
235         };
236     },
237
238     outputPosition: function() {
239         var r = this.outputNode.get('radius');
240
241         return {
242             x: this.outputNode.position().x + r,
243             y: this.outputNode.position().y + 2 * r - 10
244         };
245     },
246
247     connectionChanged: function() {
248         var connections = this.model.get('connections'),
249             connection = JSON.parse(connections)[this.
250                 inputGroup];
251
252         // No longer a connection for this input group
253         if (!connection) {
254             this.remove();
255         }
256     },
257
258     initialize: function(options) {
259         this.editor = options.editor;
260         this.model = options.model;
261         this.module = middguard.PackagedModules.findWhere({
262             name: this.model.get('module')
263         });
264
265         this.d3el = d3.select(this.el)

```

```

291 .datum(this.model.position());
292
293 this.drag = d3.behavior.drag()
294   .origin(function(d) { return d; })
295   .on('dragstart', this.dragstarted.bind(this))
296   .on('drag', this.dragged.bind(this))
297   .on('dragend', this.dragended.bind(this));
298
299 this.listenTo(this.model, 'change', this.render);
300 },
301
302 template: _.template($('#graph-node-template').html()
303   ),
304
305 render: function() {
306   var x = this.model.position().x;
307   var y = this.model.position().y;
308
309   this.d3el
310     .datum(this.model.position())
311     .attr('transform', 'translate(' + x + ',' + y +
312       ')')
313     .call(this.drag);
314
315   this.$el.html(this.template({
316     r: this.model.get('radius'),
317     handlePosition: this.dragHandlePosition(),
318     dragHandlePath: d3.svg.symbol().type('cross').
319       size(150)(),
320     runPosition: this.runPosition(),
321     runPath: d3.svg.symbol().type('triangle-up').size
322       (150)(),
323     status: this.model.get('status'),
324     statusText: this.model.statusText(),
325     displayName: this.module.get('displayName'),
326     inputs: this.module.get('inputs'),
327     output: this.module.get('outputs').length,
328     inputPosition: this.inputPosition
329   ));
330
331   var selectedInput = this.model.get('selectedInput')
332
333   ,
334   selectedOutput = this.model.get('selectedOutput')
335   );
336
337   if (selectedInput)
338     this.d3el.select('[data-name="' + selectedInput.
339       name + '"]')
340       .classed('selected', true);
341
342   if (selectedOutput)
343     this.d3el.select('.output')
344       .classed('selected', true);
345
346   if (this.model.isVisualization())
347     this.d3el.classed('visualization', true);
348
349   return this;
350 },
351
352 dragstarted: function(d) {
353   this.dragStartPosition = _.clone(d);
354 },
355
356 dragged: function(d) {
357   if (!d3.select(d3.event.sourceEvent.target).classed
358     ('drag-handle'))
359     return;
360
361   var x = d3.event.x;
362   var y = d3.event.y;
363   var r = this.model.get('radius');
364
365   var svg = d3.select(this.editor.el).select('svg');
366   var bounds = {x: svg.attr('width'), y: svg.attr('
367     height')};
368
369   // Prevent element from being dragged out bounds
370   if (x < 0) x = 0;
371   if (y < 0) y = 0;
372   if (y + r * 2 > bounds.y) y = bounds.y - r * 2;
373   if (x + r * 2 > bounds.x) x = bounds.x - r * 2;

```

```

364     this.model.position(x, y);
365     d3.select(this.el)
366       .attr('transform', 'translate(' + (d.x = x) + '
367         , ' + (d.y = y) + ')');
368   },
369
370   dragended: function() {
371     if (this.dragMoved())
372       this.model.save();
373   },
374
375   dragMoved: function() {
376     var origin = this.dragStartPosition,
377         current = this.model.position();
378
379     return origin.x !== current.x ||
380       origin.y !== current.y;
381   },
382
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'),
391       function(input) {
392         return input.name === $(event.currentTarget).data(
393           'name');
394       });
395     tooltip.html(input.name);
396
397     var bounds = event.currentTarget.
398       getBoundingClientRect(),
399         inputRadius = 5,
400         tooltipWidth = parseFloat(tooltip.style('width')
401           ) / 2,
402         tooltipHeight = parseFloat(tooltip.style('
403           height')) + 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 = middleware.Nodes.find(
419       function(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
433     // If the clicked node was already selected, return
434     // after toggling it off.
435     if (previouslySelected &&
436       this.model.get('id') === previouslySelected.get(
437         'id') &&
438       selectedGroup.name === previouslySelected.get('
439         name')) {
440       return;
441     }
442
443     this.model.set('selectedInput', selectedGroup);
444     this.connectNodes();

```

```

432 },
433
434 toggleOutputSelected: function(event) {
435     var previouslySelected = middguard.Nodes.find(
436         function(node) {
437             return node.get('selectedOutput');
438         })
439     previouslySelected && previouslySelected.set('
440         selectedOutput', null);
441
442     this.model.set('selectedOutput', true);
443     this.connectNodes();
444
445     connectNodes: function() {
446         var input = middguard.Nodes.find(function(node) {
447             return node.get('selectedInput');
448         });
449
450         var output = middguard.Nodes.find(function(node) {
451             return node.get('selectedOutput');
452         });
453
454         if (!input || !output)
455             return;
456
457         var group = input.get('selectedInput').name;
458
459         input.connectToOutput(output, group);
460         input.set('selectedInput', null);
461         output.set('selectedOutput', null);
462     },
463
464     runNode: function() {
465         if (this.model.isVisualization()) {
466             middguard.toggleView(this.model.get('id'));
467         } else {
468             this.model.run();
469         }
470     },
471
472     toggleDetail: function() {
473         var view = new NodeDetailView({model: this.model});
474
475         this.editor.$('.node').removeClass('selected');
476         this.$el.addClass('selected');
477
478         this.editor.setDetailView(view);
479     },
480
481     dragHandlePosition: function() {
482         var r = this.model.get('radius');
483         return {
484             x: r + -r * Math.sqrt(2) / 2 + 15,
485             y: r - r * Math.sqrt(2) / 2 + 15
486         };
487     },
488
489     runPosition: function() {
490         var r = this.model.get('radius');
491         return {
492             x: r + r * Math.sqrt(2) / 2 - 15,
493             y: r - r * Math.sqrt(2) / 2 + 15
494         };
495     },
496
497     /* Calculate each input circle's position.
498     * Circles are arranged in rows of three from the top
499     * down.
500     * Assume 5 pixel circle radius and 15 pixels spacing
501     * between
502     * circle centerpoints. Circles are centered around
503     * the node's centerline.
504     *
505     * Example: 5 inputs (x is an input circle)
506     *           x <--15px--> x <--15px--> x
507     *           (15px between rows)
508     *           x <-- 15px --> x
509     *
510     * @param i: input index
511     * @param r: the input parent node's radius

```



```

509 * @param n: total number of inputs for the node
510 *
511 * @return the center position for the input circle
512 */
513 inputPosition: function(i, r, n) {
514     var rowIndexX = i % 3,
515         rowIndexY = Math.floor(i / 3),
516         rowLength = i >= n - n % 3 ? n % 3 : 3,
517         baseX = r - (rowLength - 1) * 7.5,
518         baseY = 10;
519
520     return {
521         x: baseX + 15 * rowIndexX,
522         y: baseY + 15 * rowIndexY
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($('#connection-
544         group-template').html()),
545     events: {
546         'click .connection': 'selectConnector',
547     },
548
549     render: function() {
550         this.$el.html(this.template({
551             name: this.module.get('displayName')
552         }));
553
554         this.addAllConnectionGroups();
555
556         return this;
557     },
558
559     addAllConnectionGroups: function() {
560         _.each(this.connections, (value, key) => {
561             var inputs = value.connections.map(connection =>
562                 connection.input),
563                 outputs = value.connections.map(connection =>
564                     connection.output),
565                 outputNode = midguard.Nodes.get(value.
566                     output_node),
567                 outputModule = midguard.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
577                             (key),
578                         outputModuleName: outputModule.get('displayName')
579                             },
580                         outputs: outputs,
581                         unconnectedOutputs: this.model.
582                             unconnectedOutputs(key)
583                     ));
584         });
585     },
586
587     deselectOutput: function() {
588         this.selectedOutput = null;

```

```
581   this.$('.connection.output').removeClass('selected'
582   );
583   },
584   deselectInput: function() {
585     this.selectedInput = null;
586     this.$('.connection.input').removeClass('selected')
587   ;
```

```

587     },
588
589     selectConnector: function (event) {
590         var $clicked = $(event.target),
591             group = $clicked.closest('.connection-list-
592                 group').data('inputgroup'),
593             name = $clicked.text(),
594             isInput = $clicked.hasClass('input'),
595             isOutput = $clicked.hasClass('output'),
596             sameGroup = this.selectedInputGroup === group;
597
598         if (isInput) {
599             if (sameGroup) this.deselectInput();
600             else this.deselectOutput();
601
602             this.selectedInput = name;
603         }
604
605         if (isOutput) {
606             if (sameGroup) this.deselectOutput();
607             else this.deselectInput();
608
609             this.selectedOutput = name;
610         }
611
612         this.selectedInputGroup = group;
613         $clicked.addClass('selected');
614         this.connectSelection();
615
616         connectSelection: function () {
617             if (!this.selectedInputGroup ||
618                 !this.selectedInput ||
619                 !this.selectedOutput) {
620
621                 return;
622             }
623
624             var connections = this.connections[this.
625                 selectedInputGroup].connections;
626
627             var exists = this.find(connections, {input: this.
628                 selectedInput} ||
629                 this.find(connections, {output: this.
630                 selectedOutput}));
631
632             if (exists) {
633                 exists.input = this.selectedInput;
634                 exists.output = this.selectedOutput;
635             } else {
636                 connections.push({
637                     input: this.selectedInput,
638                     output: this.selectedOutput
639                 });
640             }
641
642             this.connections[this.selectedInputGroup].
643                 connections = connections;
644
645             this.deselectInput();
646             this.deselectOutput();
647             this.selectedInputGroup = null;
648
649             this.model.set('connections', JSON.stringify(this.
650                 connections));
651             this.model.save();
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 }

```

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.