## Architecture

The graph explorer consists of a front-end web application (client) and a back-end database (server), connected via a REST (representational state transfer) API layer. The front-end handles the entire business logic whereas the back-end only supplies data on-demand in a stateless manner.

The back-end database mainly serve two purposes: it is to be used to store graph data files and support search functionality. Our objective is to select a single technology that can support both for prototyping.

The front-end web application follows the model-view-controller (MVC) design pattern, which breaks out the system into three distinct building blocks:

* The model object encapsulates the data of a graph representation of malware static analysis result and performs the computation of various graph-based features in response to application requirements.
* The view object displays the graph data from the model object, which includes the rendering of the graph structure and additional visualizations. The view object is also responsible for handling user actions.
* The controller object interprets user input from the view object and correspondingly dispatches command to either update the data model or compute graph metrics as requested by the user.



Implementation

Within the client-server model mentioned above, we used Elasticsearch ([www.elastic.co](https://www.elastic.co/)) as the back-end database which powers the full text search capability and also serves as a general-purpose key-value store for graph data in dot and svg formats. The data is stored according to the following schema:

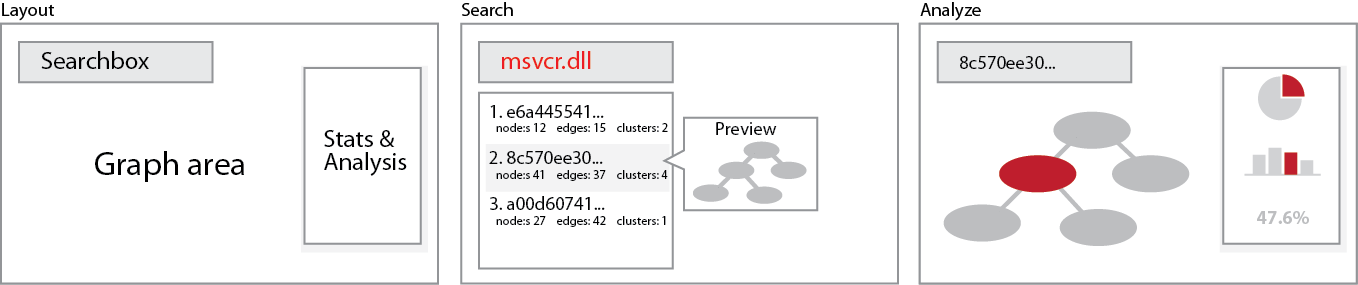
* Metrics fields: Various graph metrics including nodeCount, edgeCount, clusterCount, cycleCout.
* Graph model data: A dot format representation of malware graph including additional metadata such as 1grams attached to each node.
* Graph visualization data: A visualization of graph layout was generated in Graphviz as described in previous section. Both xdot and svg formats are stored in the database.

In the current prototype, we directly consumed data via Elasticsearch REST APIs instead of a dedicated REST API layer. While Elasticsearch APIs provided a host of functionalities from cluster management to data ingestion/query/export, we only used two API endpoints to either perform full text search or retrieve graph files by their name (hash). We also omitted setting up a reverse proxy (e.g. Nginx) which is often recommended to improve security and performance in production.

The front-end web application was built using Angular ([angular.io](https://angular.io/)), a modern web framework developed by Google. Among many other features, Angular supports two-way databinding between data model and view that greatly simplifies the development of MVC applications. The graph data model is implemented using the graphlib library, which parses a dot file into a native graph representation in JavaScript. The view of the graph is created by inserting the pre-computed svg graphics into the HTML. To support interactive visualization, d3.js is used to manipulate the svg graphics and together with Angular to create a two-way data binding between svg elements (nodes and eges) with graphlib data model.

Next steps

UI Development



The user interface follows a minimalist style and was inspired by Google Maps. It mainly consists of two permanent components − a search box and a graph display area, with several initially hidden, context-dependent panels to display additional information.

The search box supports full text search in the malware database using Elasticsearch, and it will return a list of malware graphs based on user’s keyword entered in the search box. The search results are displayed in a dropdown list which include some metadata and a preview graph.

After clicking on a result in the list, a graph will be displayed in the background. The graph will make use of the entire browser display area to provide optimal view. It also fully supports panning and zooming for navigation.

Once the data binding is established to connect the SVG graph and the graphlib data model.

We have created a basic framework of the web application.

Data binding has been established to connect the SVG graph and the graphlib data model.

In the next steps, the development work fill focus on:

* Incorporate additional visualizations from other team members.