System Design

Cyberthreat Insight Discovery & Visualization Tool

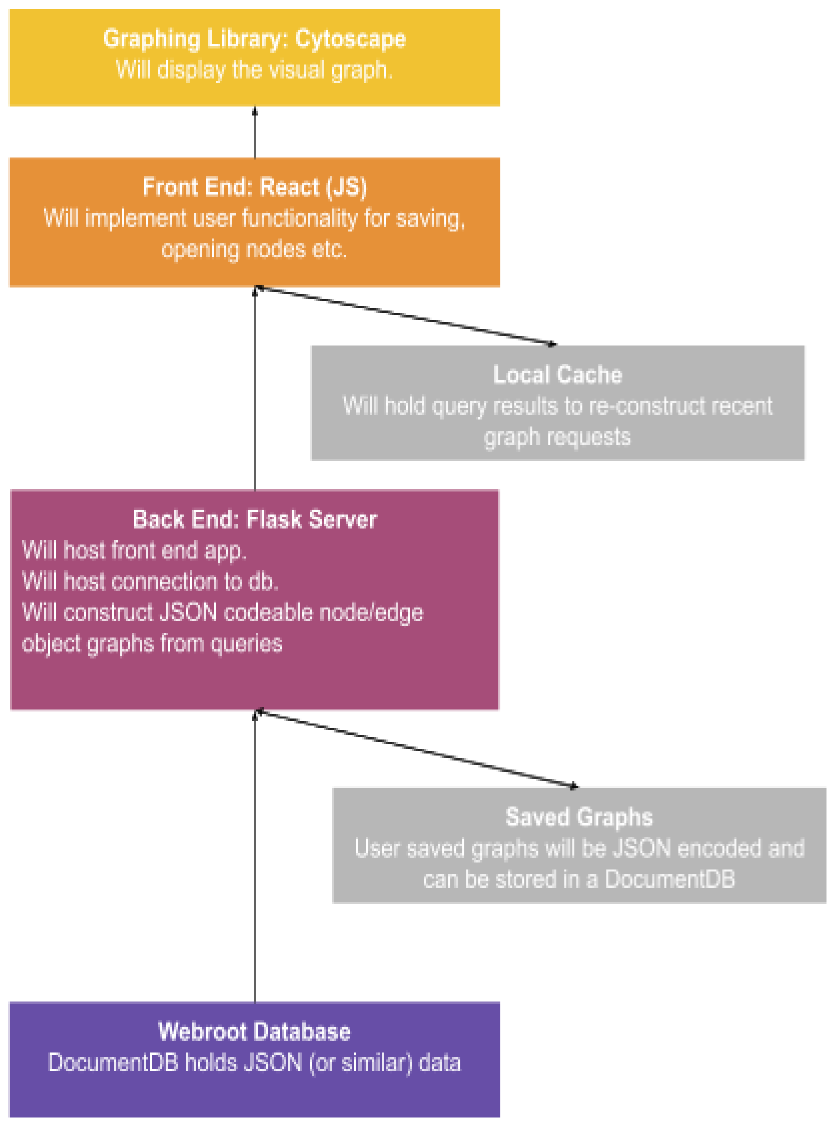
Webroot

Prepared by CU Boulder Capstone Team

November 20, 2019

The following ground-up approach provides a flexible framework. We will leverage as many open source tools as possible, such as Cytoscape for front end graphing. At the same time, we propose to use tools that are common to most developers to ensure the simplicity and longevity of the project.

Stack Diagram:


Graphing Library: Cytoscape
Will display the visual graph.
Front End: React (JS)
Will implement user functionality for saving, opening nodes etc.
Local Cache
Will hold query results to re-construct recent graph requests
Back End: Flask Server
Will host front end app.
Will host connection to db.
Will construct JSON codeable node/edge object graphs from queries
Saved Graphs
User saved graphs will be JSON encoded and can be stored in a DocumentDB
Webroot Database
DocumentDB holds JSON (or similar) data

Technology Details:

Graphing Library		
Technology Names	**Notes**	**Links**
Cytoscape	Cytoscape.js is a powerful graph theory visualization library that can create rich, interactive graphs. It is open source and has comprehensive documentation, along with an active plugin & extension development community. There is already a react component for cytoscape, so frontend interfacing will be quite easy. Supports the cola.js physics simulation, allowing for fluid node/edge movement, similar to virustotal. See demo1, demo2. Can be used with only 1 cytoscape function call. Uses an intuitive JSON format to store data about nodes (including x,y coords - good for keeping relative positions) and edges, allowing for easy saving/exporting/rendering of graphs.	Cytoscape

Front End		
Technology Names	**Notes**	**Links**
React	These libraries will help us with rendering the GUI and graph interactions as well as making the interface user-friendly.	ReactJS

Local Cache		
Technology Names	**Notes**	**Links**
Javascript	The local cache will be a front end optimization to reduce queries to the database. Results for queries will be stored in the cache, so that the cache may be queried before sending a query to the database. Cache will be implemented so that when full, the oldest queries will be replaced by the newest. Time to live may be implemented in the cache if necessary.	

Back End		
Technology Names	**Notes**	**Links**
Flask	We will use a Flask server for setting up the app. Flask can easily connect to Amazon DocumentDB NetworkX is a supported Python library we have previous experience with for creating and manipulating graphs. This will simplify much of the code needed to generate JSON graphs (iigraph is a similar tool) Note on Django vs Flask: Flask doesn't have "batteries included" for user auth etc - makes it customizable. Better for us. Also, Flask is easier for NoSQL.	Flask Connecting DocumentDB Docs for querying DocumentDB in Python
Docker	Flask apps are easily Dockerized	Dockerize Flask

Saving Local Data Without an Additional DB		
Technology Names	**Notes**	**Links**
S3, JSON on Disk, or in existing DB	All node data will be retrieved from and stored in Webroot's pipeline database, no duplicate data. User saved graphs will be represented as JSON structures, possibly stored in an S3 bucket. JSON will only provide references to the nodes in the graph, Flask will have to query the pipeline database to fill in node information and re-build the user graph. Node data will be cached on the client side, so data will be re-queried from the pipeline database each new session.	

Query Webroot's DocumentDB		
Technology Names	**Notes**	**Links**
DocumentDB	Queried via NoSQL modules in Python. These will be modular, and can be switched if the structure of the DB changes without impacting the front end.	N/A