Evolving DependencyVis -
An Interactive Visualization Tool for External Software Dependency Information
Srirag Vuppala, B.S & Bruno da Silva, Ph.D

Abstract
The growing prevalence of package managers such as JavaScript’s npm has increased the popularity of reusing software through external dependencies. Because of this ease of reusing software, developers prefer to install software dependencies over writing their own code. However, software dependencies carry risks such as conflicting licenses, security vulnerabilities, and sometimes not having enough maintainers to continue developments.

We built DependencyVis to support the understanding of the dependency network of a given project. The first iteration of the project was made by Nathan Lui for his master’s thesis. My work involved extending and refactoring the user interface, and implementing new functionality such as the Truck Factor computation and visualization, and the license categorization.

Technologies Used
The application utilized the following:
1. React.js, and Chakra UI for the new user interface styling.
2. Express.js and Node.js to build an API that calculates the Truck Factor for a git repo.
3. Heroku for deployment of an custom Truck Factor API and the website itself.
4. MongoDB atlas to host the database which is used as a cache.

Usage
Once the user adds in the repository they want to observe they view a loader until the graph is ready to render. They finally see a screen where on the left of the screen is the sidebar where DependencyVis displays all the information it gathers in text form. This is helpful for when users want to know and compare exact metric values. The to has a “Load Next Layer” Button that will load each layer of dependencies when clicked. A layer is defined by the distance (in terms of transitive dependencies) from the project being analyzed. For example, all direct dependencies form one layer and all dependencies of the direct dependencies form a different layer. The bottom right input box allows the user to input the name of a npm library they wish to add to the project being analyzed.

Architecture
To construct this tool, we first divided the responsibilities of the tool into client side and server side tasks:
- The client side is responsible for visualization of the data, all user interactions, and requests to the server side.
- The server is responsible for making all the API requests and managing the mongodb database that DependencyVis uses as a cache.

Conclusions
Package managers have made software reuse so quick and convenient that software developers today give little to no thought about the dependencies they rely on. Dependencies can save a lot of time and can even be more secure or more efficient than what most developers would have written if they coded the functionality themselves. However, these potential benefits can be lost if software developers are not careful. For example, developers could be relying on outdated or buggy code from a library that is not well maintained or abandoned.

In terms of future work, we have a few possibilities as a path forward:
1. Expand into other package managers like yarn, PyPi, or gem.
2. Conduct a case study for empirical evidence on whether and to what extent DependencyVis can be useful to software developers.