**VI. Introduction**

As a software engineer, it is very important to choose the right libraries for our projects. Picking the wrong libraries lead to expensive consequences since we need to invest time into learning how to use these libraries and of course, picking the right library for a certain programming task is not a trivial task. There are many things for us to consider when deciding on a library. One important characteristic of a library is its popularity. Using a popular library is more than just simply following the trend. A popular library means better community support such as bug reporting, feature implementation. A popular library has more learning materials, more people that we can go to and ask questions??. Being able to see what libraries are trending and are used by other people will help us better decide what to use for our project.

Our project is heavily inspired by the work of OverOps blog or blog.takipi.com. They have been conducting studies on the most popular Java libraries by scraping Github data since 2013. By looking at the pom.xml files of projects that use Maven dependency management system, they generated a large number of data points. Analysing these data points give us helped them ranked these libraries by the number of Github projects they appeared in.

Despite the usefulness of these studies, they were just snapshots at the time of conducting the studies. With their approach, our project will allow users to indentify upward/downward trends in usage and how these libraries are associated with each other.

***How it works into CS***

The fundamental task of my project was to scrape data from Github projects that use Composer for managing dependencies and inspect their composer.json files to get the data then create a graph in Arangodb.

*Composer the dependency management system*

PHP applications have been maturing into significantly more and more complex applications than ever before. There are increasingly more frameworks and third-party libraries available and use widely among the developers. The benefits of using third-party tools are undeniable since they reduce the time and cost of building software greatly. These perks of course come with a cost: We have to be able to manage these libraries and also the libraries that these libraries depend on. This is where a dependency management tool like Composer is needed for managing such a complex and interdependent arrangement. For a project that uses Composer, their dependencies are saved in a json file named - composer.json. By inspecting these files, I was able to compute a graph that connects different PHP projects.

*Graph database - Arangodb*

Since our application involves exploring many-to-many relationships, graph data structure is particularly fundamental for my project. By putting these projects in a graph, I was able to compute... (E.g. The most popular libraries, how a library's usage has been changing overtime, how many common users between libraries...) We can of course implement a graph data structure with traditional relational database but that would just mean re-inventing the wheel given the wide variety of matured and well-implemented graph database available.

So what is a graph database? Compare to traditional relational database, there isn't any new information that we store that in graph database that we couldn't in a normal relational database. They difference here, according to Neo4j website, a graph database is a database designed to treat the relationships between data as a first-class citizen in the data model. This can be simply understood that they treat edges as a document (Nosql) or a table's row (MySql). This allows graph database to travel through the networks quickly and flexibly to meet the requirement of our application, this term that is used to describe this process is called "*graph traverse*".

*Github*

Github is one of the most popular social software development platforms at the moment. Github allows us to collect data in the past so that we could compute a timeline of how these data changes....

In the remaining of the report, we will cover the background of this project, the preparation steps that we took, the outcomes/deliverables of this project and finally the evaluation.

**VII. Background**

What is the project trying to achieve? : There have been previous studies popular libraries by inspecting github projects. But this project allows us to query popular libraries at given points in time.

Why that is important? : We know what libraries are trending on github. How their usage has been changing over the time.

**VII. Preparation**

1.

* Architecture diagrams

2.

* Programming language learnt: Python, JavaScript.
* Better understanding of NoSql and graph database. (Arangodb)
* Frameworks learnt: React, Flask

3. We need a graph database for our application where we can connect the libraries. Arangodb was chosen instead of Neo4j because it supports saving data on edge nodes. On the other hand Arangodb is a multi-model and it has arguably better performance than Neo4j.

4. Python:

* Using Python for automation tasks are usually better than PHP. We chose Python over PHP for fetching data.
* Since a lot of the logic is handled by Arangodb's AQL, we only needed a simple json service to call the aql queries. I've decided to use Python for the web service. Flask is the chosen micro framework for the job. It supports routing and jsontify which is all we need from our web service.

5. Front end:

* React is a trending and interesting framework so I've decided to use React
* Simple-react-state: I wanted a state management library where I can have a global state for the whole application. Something like Redux would be overkill for my application.

6. Separation of concern:

* Separation of concern is a trending architecture in the world of web application. Where the backend acts as a Json service that provides the data for the frontend (E.g headless cms).
* The architecture allows me to be flexible with my choices of backend and frontend frameworks.
* In the future, having a json service would allow me to build mobile and desktop apps for my application

**VII. Deliverable**

Web application, arangodb database, Backend

1. Web app

* Allow users to search for libraries with their name, query the most popular libraries at given time, see the usage of libraries overtime, see the libraries that are commonly used with a library.

2. Arangodb database

* Graph of libraries.
* Aql queries that perform many complex graph traversing

3. Back end

* Serve json for the webapp
* whoosh library for indexing the document
* Perform frequently update and data fetching to expand the graph
* Flask cache: file system cache instead of memcached ( does not have to connect to memcached over network, simple to implement and cover all the needs)

**VIII. Evaluation**

* The front end meets most of the specified objectives but it is still relatively simple and does not have many functionality
* We couldn't use Celery in the data fetching process
* We added a cache layer for our project
* Used packagist together with github get projects's data

**APENDIX 2**

**Set up:**

* Install composer, Arangodb
* Cherrypy server or Rocket for Flask deployment