**VI. Introduction**

Picking the right library for a certain programming task is not a trivial task. 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. There were different studies about the most popular libraries conducted by different people. Takipi blog conducted a study for the top 100 most used libraries on github by analysing their POM.xml to get the dependencies. Despite the usefulness of this study, it was just a snapshot at the time of conducting the study. With their approach, our project will allow users to indentify upward/downward trends in usage and how these libraries are associated with each other.

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