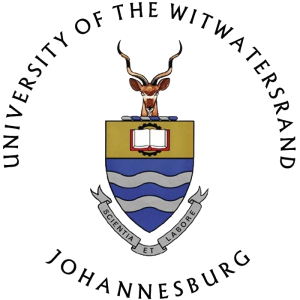
School of Electrical and Information Engineering - Witwatersrand

ELEN 7045 Software Development Methodologies, Analysis and Design



Big Data Visualization using Commodity Hardware and Open Source Software - historic data sourcing

Matsobane Khwinana

779053

4 July 2016

Table of Contents

[Introduction 1](#_Toc455003228)

[Summary of the use cases 1](#_Toc455003229)

[The Solution Design 1](#_Toc455003230)

[High Level Design 2](#_Toc455003231)

[Detailed Designs 2](#_Toc455003232)

[Development approach and technique 2](#_Toc455003233)

[Assumptions and Constraints 2](#_Toc455003234)

[Possible Extensions 2](#_Toc455003235)

[Conclusion 3](#_Toc455003236)

# Introduction

The historic data source component is the sub-component of the data sourcing module of the Big Data Visualization using Commodity Hardware and Open Source Software project undertaken by group 2 of the 2016 ELEN-7046 class at Wits University. As stated in the group project report [1]; the project provides a low cost solution to sourcing, processing and visualizing Big Data using non-commodity hardware and open source software. The data sourcing component was divided into 2 sub-components, namely streaming (for live data feed) and historic data extraction (using start and end dates, and counts). This report discusses the solution provided for the historic data sourcing sub-component or service to illustrate the contribution and made by the author on the project.

The primary use cases are listed and the solution design is outlined, broken down into high level and detailed designs; the development approach and techniques is explained; the assumptions and constraints are listed; the subsequent sub-sections suggests the possible extensions and the conclusion sums up the report; the references and appendices then follows.

# Summary of the use cases

Below is the list of primary use cases addressed by the historic data sourcing component:

* Extract and persist tweets by list of hashtags using random dates
* Extract and persist tweets by list of hashtags using start and end dates
* Distribute all persisted tweets to the data processing component FTP

# The Solution Design

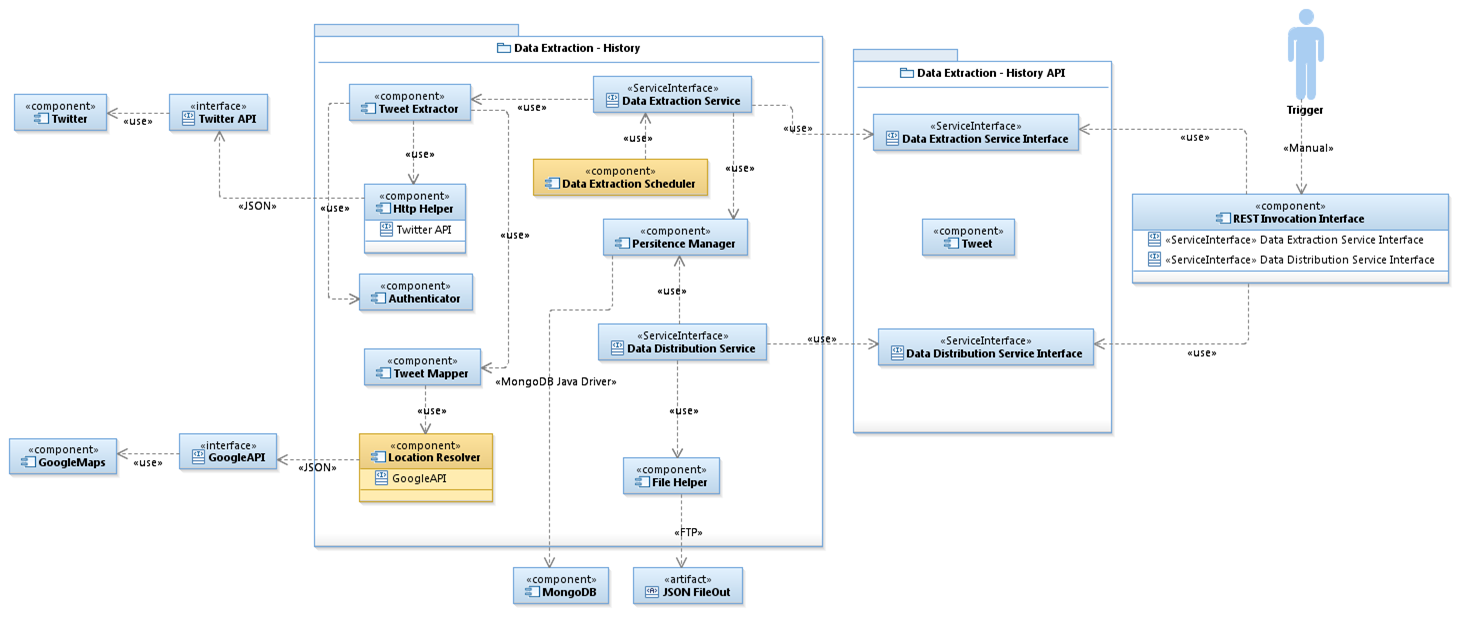
The reasons for splitting the data sourcing into streaming and historic sub-component includes among others decomposing a bigger into smaller problem and therefore benefiting from parallelism, loosely coupling and primary saving time by avoiding learning new programming languages, these has been discussed in more detail the group report. Java was chosen as the implementation language primarily because the author has experience on the language and therefore time would be saved by avoiding to learning new languages. The solution stated as standalone Java SE (Standard Edition) project and evolved into the Java EE (Enterprise Edition) solution as the project matured as some of the capabilities provided by the Java EE were required to fulfil the use cases.

## High Level Design

The component diagram in the next page illustrates a high level design or architecture of the historic data source component.

The API component includes the interfaces describes the extraction and distribution use cases. The implementation module includes the actual implementation components for the API; this module also houses the implementation of the REST endpoint. The project could have chosen to implement the endpoint module as a separate component or module however due to the abstractions provided by the Java EE, there was no need separate the endpoint as independent deployable unit. However the need should arises in the future this can be easily achieved because the endpoint does not dependent on the inner implementations of the service but the public API (Enterprise Java Beans).

The implementation module integrates with both Twitter and Google Maps Geocoding APIs. The reason for using Geocoding API shall be explained in details on the sub-sequent sections. The design was not proposed upfront however it evolved and surfaced as each use case was implemented, one after the one.



**Design rationale**

API was separated from implementation to archive loose coupling. The module is made up of interfaces and entity classes only and packaged as separated Java Archive (jar) file. This allows for swapping of implementation without affecting the API. This made possible by using Java EE's Enterprise Java Beans (EJBs), more details on how EJBs work plus reference.

## Detailed Designs

Show class diagrams discuss the flow (perhaps add sequence diagram)

And a tabular description of classes – refer to work Tech Designs Docs…

## Development approach and technique

TDD and UDD… discuss in detail. Show snippets of test cases

## Assumptions and Constraints

Twitter Location (caching) and Rates limits usages (scheduling)

## Possible Extensions

Software licensing

# Conclusion

Software licensing…

**References**

**Appendices**