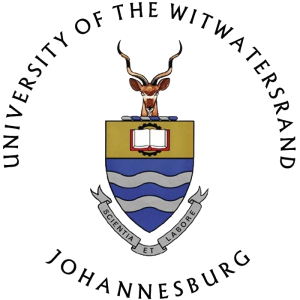
School of Electrical and Information Engineering - Witwatersrand

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](#_Toc455142276)

[Requirements 1](#_Toc455142277)

[Approach 1](#_Toc455142278)

[Challenges 1](#_Toc455142279)

[Design Overview 2](#_Toc455142280)

[Technologies used 3](#_Toc455142281)

[Class Diagram 3](#_Toc455142282)

[Sequence Diagram 5](#_Toc455142283)

[Testing 5](#_Toc455142284)

[Recommendations 5](#_Toc455142285)

[Conclusion 5](#_Toc455142286)

# 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.

# Requirements

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

# Approach

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

# Challenges

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

# Design Overview

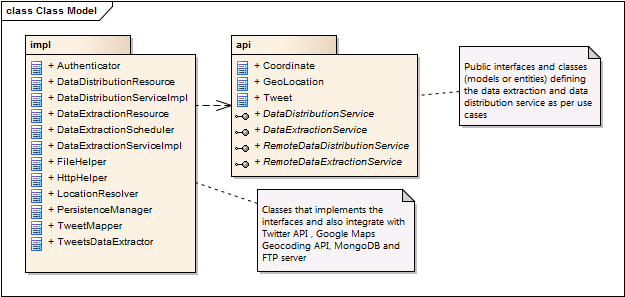
The reasons for splitting the data sourcing into streaming and historic sub-component includes among others decomposing a bigger problem into smaller a problem and therefore benefiting from parallelism, loosely coupling and also saving time by avoiding learning new programming languages; these has been discussed in more detail the group report.

Java was chosen as the implementation language primarily because the author has experience on the language; because time was one of the major constraints in the project avoiding learning new languages helped in saving time. The solution started as a standalone Java SE (Standard Edition) project and evolved into the Java EE (Enterprise Edition) solution as the project matured and some of the capabilities provided by the Java EE were required to fulfil the requirements.

The class model diagram in the next page illustrates a high level view of the structure of the historic data collection component.

The API component includes the interfaces that describe the extraction and distribution use cases. The implementation module includes the actual implementation classes for the API; this module also contains the implementation of the REST endpoint. The project could have chosen to implement the endpoint module as a separate component however due to the abstractions provided by the Java EE, there was no need separate the endpoint as an independent deployable unit. So even though the endpoint is the same component as service implementations classes, they are not tightly coupled to the implementations. The endpoint accesses the implementations via the public interfaces provided by the Java EE EJB (Enterprise Java Beans). However should the need to separate out the REST endpoint arises in the future this can be easily achieved.

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**

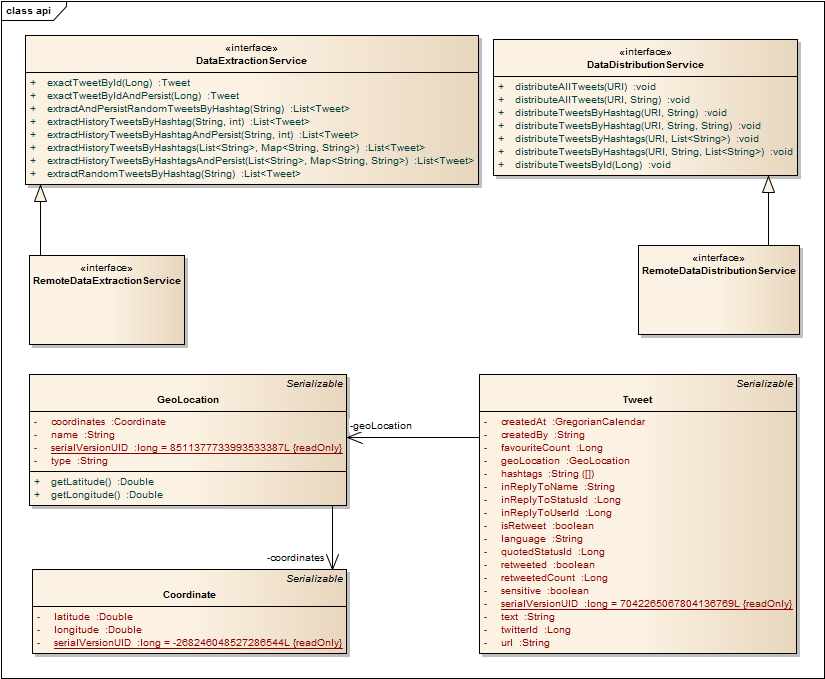
API was separated from implementation to archive loose coupling. The API module is made up of public interfaces and entity classes only, see the class diagram below. The API is packaged as separate Java Archive (jar) file making the component independently shareable; meaning the interest users of the API need not have access to the implementation details of the service. This allows for easy swapping of implementation without affecting the API users. This made possible by using Java EE's Enterprise Java Beans (EJBs), more details on how EJBs work plus reference needed.

## Technologies used

The class diagram below shows the API module design; subsequently the table describes of the class in the diagram.

## Class Diagram

The class diagram below shows the API module design; subsequently the table describes of the class in the diagram.



The table below describes the major classes in the API module and their roles.

|  |  |
| --- | --- |
| Class | Description |
| DataExtractionService | A public interface defining the data extraction service as per use cases. The interface uses Tweet class to represent a Twitter post or message. |
| DataDistriibutionService | A public interface defining the data distribution service as per use cases. The interface uses Tweet class to represent a Twitter post or message. |
| Tweet | A class that represents a Twitter post or message. It encapsulates the GeoLocation. |
| GeoLocation | A class that represents geographical location of a Tweet. It is made of an array of Coordinates; the type may have values such Polygon, Point, MultiPoint, etc. |
| Coordinate | A class that represents a geographical location coordinates, made of latitude and longitude. |

The implementation class diagram in Appendix A shows the implementation module design; the table below describes the major classes in the diagram and omit other helper or utility classes because of the size of this report.

|  |  |
| --- | --- |
| Class | Description |
| DataExtractionServiceImpl | A public interface defining the data extraction service as per use cases. The interface uses Tweet class to represent a Twitter post or message. |
| DataDistriibutionServiceImpl | A public interface defining the data distribution service as per use cases. The interface uses Tweet class to represent a Twitter post or message. |
| TweetsDataExtractor | A class that represents a Twitter post or message. It encapsulates the GeoLocation. |
| PersistenceManager | A class that represents geographical location of a Tweet. It is made of an array of Coordinates; the type may have values such Polygon, Point, MultiPoint, etc. |
| DataExtractionResource | A class that represents a geographical location coordinates, made of latitude and longitude. |
| DataDistributionResource |  |
| DataExtractionScheduler |  |

## Sequence Diagram

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

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

## Testing

Validation of results…

Show test cases snippets

# Recommendations

Software licensing

# Conclusion

Software licensing…

**References**

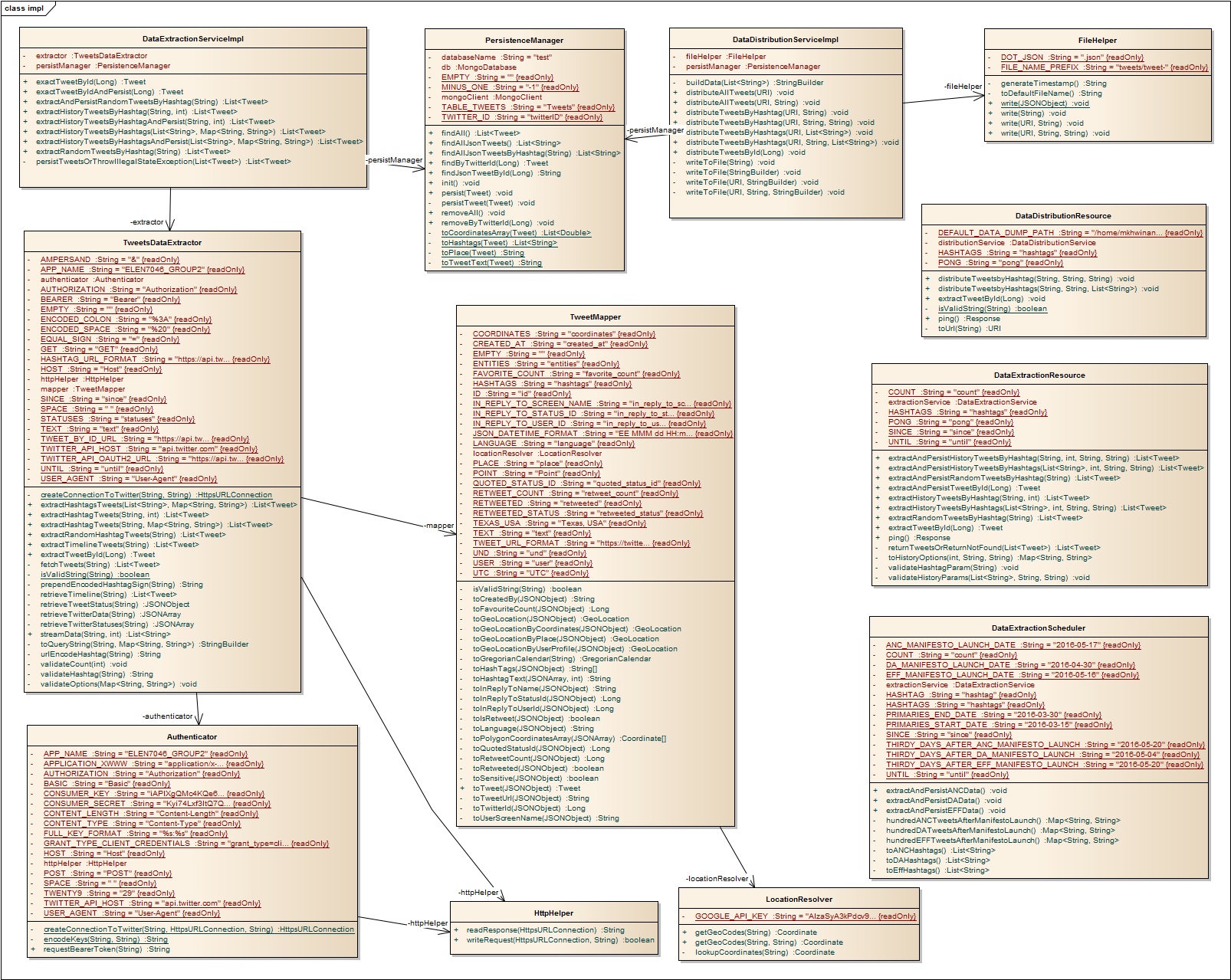
**Appendices**

**Appendix A**

**API module class diagram**

**Appendix B**

**Implementation module class diagram**

****