Architecture Document

Jake Ballinger, Hawk Weisman, Dibyo Mukherjee, Gabe Kelly, Ian Macmillan

December 13, 2013

”Add little to little and there will be a big pile.”
- Ovid

1. Introduction to Document

1.1 Identifying Information

This Architecture Document describes the system architecture and organization for *Knightingale*, a *Twitter* analytics system.

1.2 Document Conventions

This document was written following *IEEE* conventions. It was formatted with LaTeX. This is the mid-level of the trio of documents; the design document is at a lower level of description, and the requirements document is more concerned with the high-level concepts and expectations of the procuct.

Names, classes, and methods will be italicized, and both numbers and variables are modified with the $ wrapper in LaTeX.

1.3 Acronyms, Abbreviations, and Definitions

cli: Command-Line Interface
GUI: Graphical User Interface
LaTeX: The document-formatting language used to organize the information in the SRS, Architecture Document, and the Design Document.
SRS: Software Requirements Specification Document (also known as the Requirements Document)

1.4 System Overview

To give a brief overview, *Knightingale* provides useful information about your tweeting habits. It was developed by students at *Allegheny College* as part of the *Computer Science 290: Principles of Software Development* class in *Fall* of 2013. Please refer to Section 1 of the Requirements Document for a detailed explanation of *Knightingale.*

1.5 References

1.6 Outline of the rest of the SRS

2. System Architecture

2.1 Architectural Design

There are five major architectural components in the system:

1. Analytics
2. Database
3. Parser
4. UI
5. Refresh
6. Test Suite

The Analytics package quantifies all of the metrics outlined in the Requirements Document.

The Database package provides an access layer to the database. Database access for the rest of the system is localised through this layer.

The Parser package contains the classes that allow the system to extract information from the tweets archive.

The Refresher package uses the *Twitter4j* library to connect to the Twitter API and retrieve new tweets.

The UI package allows the user to interact with the system. It contains two distinct packages: the cli and the GUI. The cli package contains the primary command line interface while the GUI is used for creating word clouds.

The Test Suite tests the system. We use a test suite to establish a confidence in the correctness of our code.

Knightingale's essential path of function begins when the user downloads their twitter archive (a *.ZIP* file) of their tweets from *Twitter* through *Knightingale's* UI package. The Parser makes sense of that information, and it is then stored in the database.

Next, the Analytics package takes this data and extracts the necessary analytic information. These metrics and fed back to the user through the UI. The Refresher can be used to retreive new tweets that the user has posted after downloading the archive and add them to the database for analysis. The Test Suite is used exclusively by the developers and does not interact with the user.

2.2 Design Rationale

The system is designed to be modular and comply with best practices such as localising inputs and ouputs. The ui package handles all input and output. Wherever needed logging is used to log errors and other informative messages using the *java.util.logging* package. The database module is provides an access layer that all other modules use to access the database and execute quieries. This design makes it easy to swap out our current database for a different database in the future with minor refactoring. We use mocking in our test suite to test localise error detection using the Mockito library.

2.3 Known Inconsistencies

The Refresh module has a input output component where a user may be asked to go to a link to authenticate with Twitter and input an authentication PIN from the Twitter website. This input and output is very specific to this module and therefore makes more sense here than in the ui package. Some modules are not included in the test suite for various reasons:

1. GUI - this module was excluded because of time constraints.
2. DbHelper - we had difficulty in setting up dbUnit. Hence, we manually tested the db with various configurations.
3. Refresh - this was difficult to test because it relied on connections to the Twitter API.