**Twitter Brand Analysis**

**Software Architecture Document**

**Date: 2014/12/19**

**Group: Name Pending**

**Members:**

Andreas Lundkvist

Ian Jenkins

Kenny Nilsson

Pontus Hjortskog  
Shan Jiang

Tommy Molin

[Xiaoqian Xiong](https://www.facebook.com/jessie.x.xiong)

Table of Contents

## 1 Introduction

### 1.1 Scope

### 1.2 Intended Audience

### 1.3 Architectural Representation

### 1.4 References

### 1.5 Acknowledgments

## 2 Overview

### 2.1 Presentation Layer

### 2.2 Application Logic

### 2.3 Data Access

### 2.4 Infrastructure Layer

### 2.5 External Systems

## 3. Architectural Drivers

### 3.1 Requirements

### 3.1.1 Functional Requirements

### 3.1.2 System Qualities

### 3.2 Constraints

## 4. Architectural Mechanisms

### 4.1 Analysis Mechanisms

### 4.2 Analysis Mechanism Mappings

## 5. Use Case View

### 5.1 Architecturally Significant Use Cases

## 6. Logical View

### 6.1 Overview

### 6.2 Presentation Layer

### 6.2.1 Web UI

### 6.2.2 IOS UI

### 6.2.3 Android UI

### 6.3 Business Logic

### 6.4 Data Access

### 6.5 Infrastructure Layer

## 7. Deployment View

## 8. Process View

## 9. Availability

### 9.1 Miner

### 9.2 Web-Backend

## 10. Reliability

### 10.1 Training Data

### 10.2 Sentiment Analysis

### 10.3 Cross Fold Validation

## 11. Scalability

# 1. Introduction

This document provides an architectural description of the Twitter Brand Analysis (referred to as TBA hereafter) system using the 4+1 architectural view model to depict different aspects of the system. The document aims to capture the architectural decisions that have been made and elaborates on aspects of the system that are considered to be architecturally significant. The views include use case view, logical view, deployment view and process view.

Also described are the drivers that have influenced the architecture of the system, key requirements that are considered the architectural important, mechanisms and detailed descriptions of architecturally important quality attributes availability; reliability and scalability characteristics.

## 1.1 Scope

This Software Architecture Document (SAD) represents the architecture of the TBA system.

The TBA project is using the Agile Scrum development methodology, and because of this the information included in this document will evolve as the project evolves.

## 1.2 Intended Audience

This is a technical document intended for developers, product owner and related technical resources.

## 1.3 Architectural Representation

The Unified Modelling Language (UML) is an industry standard software system modelling notation administered by the Object Management Group.

This document describes the architecture of TBA using standard UML diagrams, including use case, component, package, and deployment diagrams. Readers of this document need to be familiar with the UML notation.

## 1.4 References

Natural Language Processing Library:<http://www.nltk.org>

Sentiment Analysier <http://www.ravikiranj.net/drupal/201205/code/machine-learning/how-build-twitter-sentiment-analyzer>

Data Driven Documents Library:<http://d3js.org/>

Bootstrap Library: <http://getbootstrap.com/>

HighChart: <http://www.highcharts.com>

ENSwiftSideMenu: [https://github.com/evnaz/ENSwiftSideMenu](https://github.com/evnaz/ENSwiftSideMenu/tree/master/Example)

HelloCharts: https://github.com/lecho/hellocharts-android

SOLA SAD

ENSwiftSideMenu: <https://github.com/evnaz/ENSwiftSideMenu/tree/master/Example>

SwiftHTTP: <https://github.com/daltoniam/SwiftHTTP>

SwiftyJSON: <https://github.com/SwiftyJSON/SwiftyJSON>

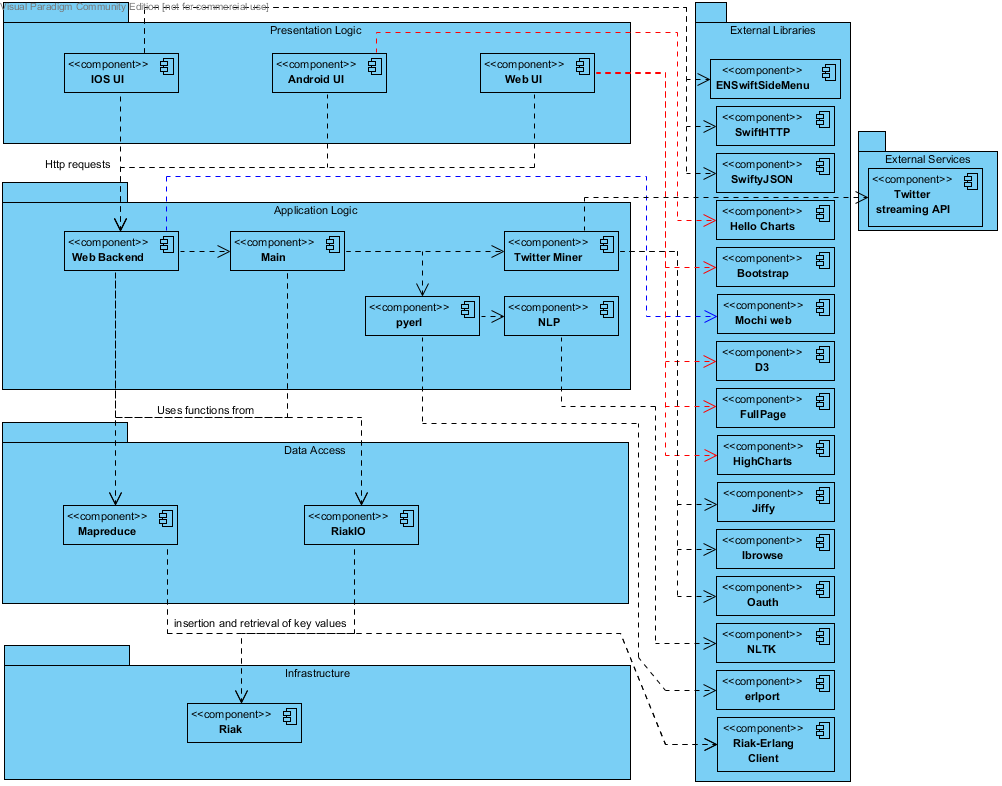
## 1.5 Acknowledgements

The following people have provided information or feedback that has contributed to the content of this document:

William Granli

# 2. Overview

The architecture of TBA has been designed using the a layered architectural style. A layering approach has been used to structure the software by responsibility. The following Component Model diagram illustrates the layers of TBA.



## 2.1 Presentation Layer

The web UI consists of a responsive web page, built in HTML and JavaScript to ensure maintainability and usability. It utilises HighCharts, a HTML and JavaScript data visualization tool to ensure high performance and maintainability. The system supports login for multiple users, users can then search and track keywords and view visualizations of positive and negative sentiment values of tweets over a timespan and demographics of brand opinion.

The Android UI is an Android smartphone application that is deployed to the end users by installing an .apk file. It runs on latest version of Android Lollipop 5.0 and is backwards compatible to 4.1.

The IOS UI is an iPhone application that runs on the newest available version iOS 8.1. It supports deployment back to iOS 6.0. Basic functionalities are same as the Android UI.

Both platforms, Android and IOS which run on handheld devices has been developed to improve accessibility of the system. The apps supports login for registered users to view the sentiment value of individual keywords as well as the map.

## 2.2 Application Logic

The application logic consists of a UI-backend and a miner. The

UI-backend handles requests from the user interfaces.

The miner-component handles collecting of raw data from twitter and the processing of collected data. It implements Erlang/OTP Supervisors. Processing is done in three steps:

1. Extract useful data from raw data

2. Classify data with NLP

3. Insert into database

The Natural Language Processing (NLP) component is written in Python utilizing the NLTK library to process language data. It is the largest part of our tweet processing. It gets the sentiment value of the tweet text using a Naive Bayes classifier.

## 2.3 Data Access

The Data Access handles data input and output of a NoSQL database and consists of two components. RiakIO handles basic storing, fetching and queries from and to the database. MapReduce handles advanced data calculations.

## 2.4 Infrastructure Layer

The infrastructure layer consists of a NoSQL database using Riak.

## 2.5 External Systems

The TBA system is reliant on one external system: Twitter’s Streaming API. The Streaming API is used to collect raw data from Twitter.

# 3. Architectural Drivers

This section discusses the architectural drivers that have been identified for TBA. The architectural drivers include the requirements that have architectural significance, constraints the system must comply with and architectural principles. The drivers provide rationale for key aspects of the TBA architecture.

## 3.1 Requirements

### 3.1.1 Functional Requirements

The following table identifies and discusses the functional requirements of TBA that have architectural significance.

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Description | Significance | Resolution |
| Track Brand | The system will allow users to search, track keywords/brands. FR5 – FR6 | Tracking brands over time is the core functionality of the system. | Data is stored in a NoSQL database. |
| View Brand Data | The system will allow users to select and view data. FR7 – FR11 | Being able to view the data that is tracked is important to be able to analyse large amounts of data. | Web data is displayed with a JavaScript charting engine, HighCharts. Data is queried by using Map/Reduce functions. |
| User Account | The system allows users to create and access an account. FR1 - 2 | User accounts are important for providing different users with personalized data. | Authenticated through the RESTful api and remembered by having cookies with encrypted tokens |

### 3.1.2 System Qualities

The following table identifies and discusses the qualities of the TBA system (i.e. non functional requirements) that have architectural significance.

|  |  |  |  |
| --- | --- | --- | --- |
| Quality | Description | Significance | Resolution |
| Availability | The system should be available to users at all times. | The system must be available to users whenever they need it. | OTP guidelines ensures availability for miner and client server. |
| Reliability | The system must be able to judge tweets with a 80% accuracy. | If the system cannot reliably judge tweets the data would be inconsistent and inaccurate. | Classifier using Navies Bayes algorithm implemented from NLTK library. Cross validation to determine accuracy. |
| Scalability | The system must be distributable to multiple physical devices. | By spreading the system to multiple nodes the workload of each machine is reduced and hardware errors become less significant. | By having the system run on Erlang with a Riak database distribution is greatly simplified. |

## 3.2 Constraints

The following table identifies and discusses the main architectural constraints that TBA must comply with.

|  |  |
| --- | --- |
| Constraint | Description |
| Erlang | The Client Server Architecture must be written in Erlang. |
| Riak | Riak was the database specified by the client. |
| Map/Reduce | The system must utilize the map/reduce functionality of Riak in some way. |

# 4. Architectural Mechanisms

This section presents the architectural mechanisms of the TBA system and identifies how these will be incorporated during design and implementation. Architectural mechanisms represent common solutions to frequently encountered architectural problems and are often used to realise architectural requirements.

## 4.1 Analysis Mechanisms

The following table describes the analysis mechanisms applicable to TBA.

|  |  |
| --- | --- |
| Analysis Mechanism | Description |
| Persistence | To keep an element persistent.(ie. save an object’s state across multiple executions of the application.) |
| Communication | How distributed processes communicate. |
| Business Logic Encapsulation | How the Business Logic is encapsulated. |
| Security | How user passwords will be protected. |
| User Interface | Provide access and user interaction of data. |
| Electronic Data Interchange | How data is exchanged within and to the system. |

## 4.2 Analysis Mechanism Mappings

The following shows how the analysis mechanisms above have been mapped to design and Implementation mechanisms.

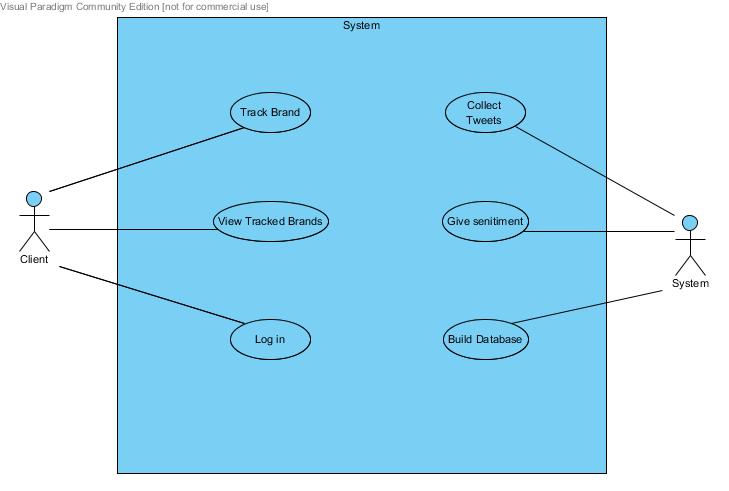
|  |  |  |
| --- | --- | --- |
| Analysis Mechanism | Design Mechanism | Implementation Mechanism |
| Persistence | NoSQL Database | Riak |
| Persistence | serializing and deserializing a Python object structure | Pickle |
| Communication | OTP | Erlang 16.0 |
| Business Logic Encapsulation | Separation of concerns | Layered design |
| Business Logic Encapsulation | Low coupling | Interface between packages |
| Security | Password Encryption | SHA 512 with salt |
| Security | Client Server | Sensitive information is kept on the server. |
| User Interface | Web, Android, IOS | Html, JavaScript, Css, Android API, Java, XML, Swift |

# 5. Use Case View

This section identifies use cases from the use case model that have architectural significance.

## 5.1 Architecturally Significant Use Cases

The following use case diagram shows those use cases that are considered architecturally significant.



**Track Brand**

The user will be able to enter one or more brands to be tracked.

**View Track Brands**

The user can view information collected about the tracked brands on a map and in charts.

**Login**

Users are able to log in to get access to their tracked brands.

**Collect Tweets**

Streaming data from Twitter must be collected to be processed by the system.

**Give Sentiment**

The system should analyse a tweet and provide a positive, negative or neutral sentiment.

**Build Database**

Processed data should be collected in a database.

# 6. Logical View

This section describes the architecturally significant parts of the design and its decomposition into packages and subsystems. The following section describes the high level packages and elements that make up TBA.

## 6.1 Presentation Layer

The Presentation Layer encapsulates the components that end users will interact with in the Web, IOS and Android interfaces.

### 6.1.1 Web UI

The web user interface is a web page built in HTML and javascript using javascript libraries for visual data presentation.

### 6.1.2 IOS UI

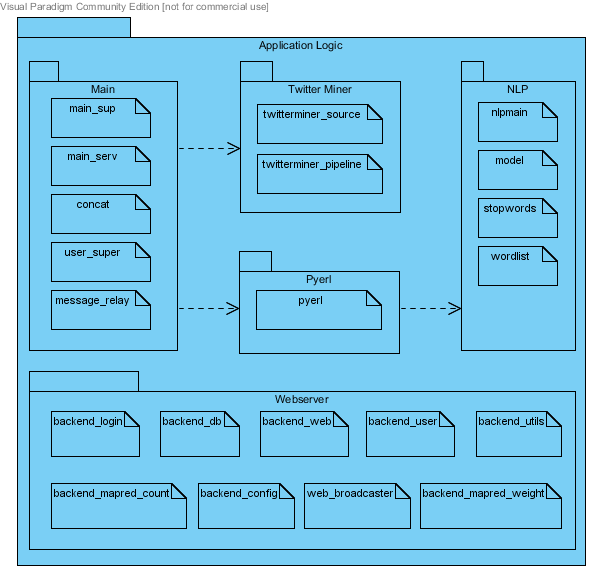
The iOS user interface is an application built in Swift with graphs loaded from the web UI.

### 6.1.3 Android UI

The android user interface is an application built in Java with the worldmap loaded from the web UI and java libraries for the other graphs.

## 6.2 Application Logic

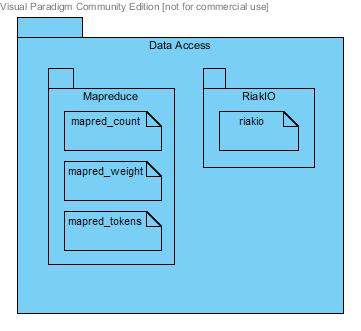
The application logic layer encapsulate components that makes up the core processing of the system.



Main is the core of the miner, consisting of supervisors and underlying servers. It creates child processes of twitter miner to collect streaming data from Twitter and connects to the NLP by using pyerl to connect main (running Erlang) and NLP (running Python) together.

Webserver is the interface between frontend and backend. It receives HTTP requests from the frontend and retrieves and returns the requested data.

## 6.3 Data Access



The Data Access layer encapsulates interfaces used to insert and retrieve data from the database. RiakIO handles all insertion and basic retrieval functions. Mapreduce is used for more advanced retrievals.

mapred\_count: Mapreduce for calculating amount of positive, neutral and negative sentiment values in a dataset.

mapred\_weight: Mapreduce for calculating the average sentiment value on a location basis in a dataset.

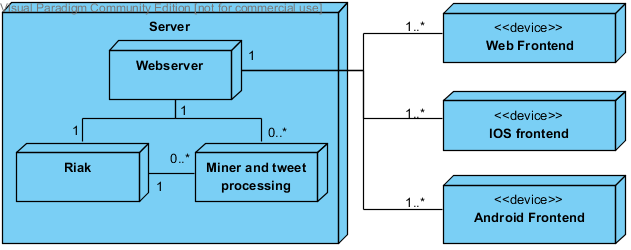
mapred\_tokens: Mapreduce for calculating which words, hashtags and users that has been used/mentioned most in a dataset.

## 6.4 Infrastructure Layer

The infrastructure layer consists only of a Riak database running a LevelDB backend.

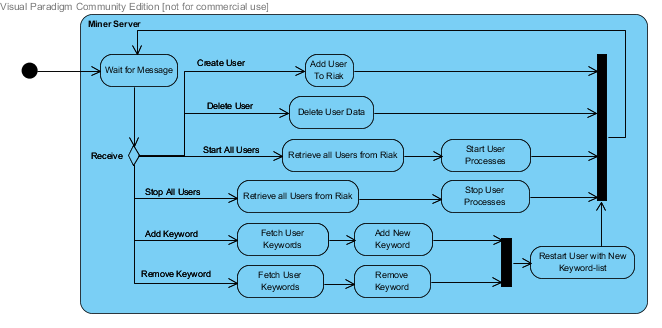
# 7. Deployment View

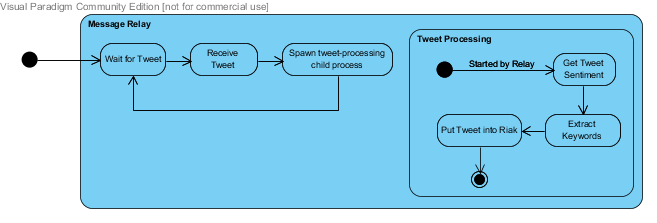
This section shows the physical configuration on which the software is deployed to. It shows the connections between components at run time.



TBA is a two-tier application, both database and webserver on one tier and the client(Web,IOS and Android) on the second tier. Since performance is not a architectural driver in this software project a two-tier was appropriate to for quicker development. Using a two-tier application the product is easier to maintain and could be easily scaled to three or four tiers to allow for more users of the system by moving the Miner and tweet processing or Riak to other devices.

# 8. Process View





# 9. Availability

It is of high importance that the system is available to its users whenever they need it. Whenever the system is unavailable money is lost. This section describes how we deal with this quality requirement.

## 9.1 Miner

The miner is implemented using Erlang’s OTP Supervisors. A main supervisor using the one\_for\_one restart strategy creates one child supervisor with the one\_for\_all restart strategy for each user with a twitterminer, message-relay and concatenation process each. This means that if one process crashes the supervisors will see that it is restarted properly again.



## 9.2 Web-Backend

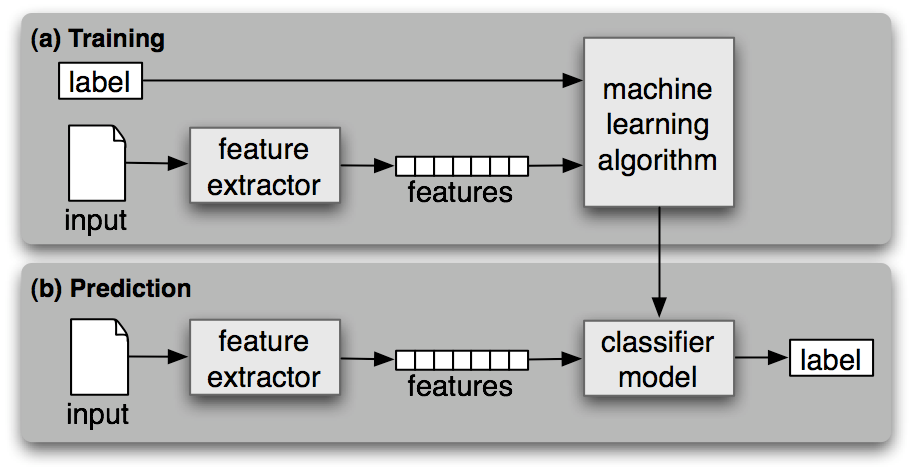
The web-backend part of the system also utilizes Erlang’s OTP modules by using the mochiweb-module which is built on top of OTP. The http server uses the module reloader from the mochiweb project that monitors if the backend’s modules has been recompiled, which combined with Erlang’s support for hot-swapping modules allows seamless injection of new module versions into the running processes. By keeping all runtime data in the database separate from the web-backend we can achieve high uptime even when upgrading the system.

# 10. Reliability

The application’s main concern is to accurately decipher sentiments of tweets to provide the user opinions of keywords/brands. To address this quality requirement a Naive Bayes classifier was implemented using the Natural Language Toolkit (NLTK) library. Naive Bayes classifier is a well documented probabilistic classifier used for text classification. Several sources(References section) were re-used and modified for the application’s needs. To test the accuracy of the the classifier model a 10 fold cross validation method is used which is a technique widely used for machine learning.

Figure: 1

The following diagram illustrates a process of sentiment analysis



## 10.1 Training Data

Approximately 1000 tweets have been manually classified for positive and negative sentiments (500 respectively) to provide a training set for analysis. Data is specific to brand opinion collected from twitter to increase accuracy. The data is put into a feature vector and processed by the machine learning algorithm, the output is a classifier model stored as an pickle file.

**10.2 Sentiment Analysis**

The NLP component utilizes the NLTK library functions to train and classify data. Classify. Classification is done using Naive Bayes algorithm which outputs the highest probability that the combination of words together are positive negative or neutral using the formula.

**10.3 Cross Fold Validation**

A 10 fold cross validation method to check the accuracy of a model is commonly used in machine learning and the following text displays the results of such a test on our model.

Test description:

10 buckets, each containing approximately 50 tweets contained expected sentiment values are tested against the remaining 9 buckets serving as the model which classifies each tweet. Each bucket is rotated along with subsequent models. A built in NLTK accuracy function calculated the expected sentiment over the outcome and returns the over accuracy of the model.

Results:

|  |  |
| --- | --- |
| Bucket number | Accuracy (percent) |
| 0 | 55.15 |
| 1 | 55.34 |
| 2 | 52.94 |
| 3 | 53.73 |
| 4 | 57.74 |
| 5 | 57.33 |
| 6 | 59.28 |
| 7 | 57.59 |
| 8 | 58.25 |
| 9 | 55.90 |
| Average Score | 56.325 |

The training set used in our model is small therefore accuracy is compromised.To increase accuracy the training set can be increased.

# 11. Scalability

For systems with multiple concurrent users it is important that it is a trivial task to add more processing power to the system in the form of more hardware. We have solved this with our choice of language and database.

The system is running Erlang with a Riak database. Both Erlang and Riak work with individual nodes running in clusters and each node can be run on different physical machines if needed, providing theoretical infinite scalability.